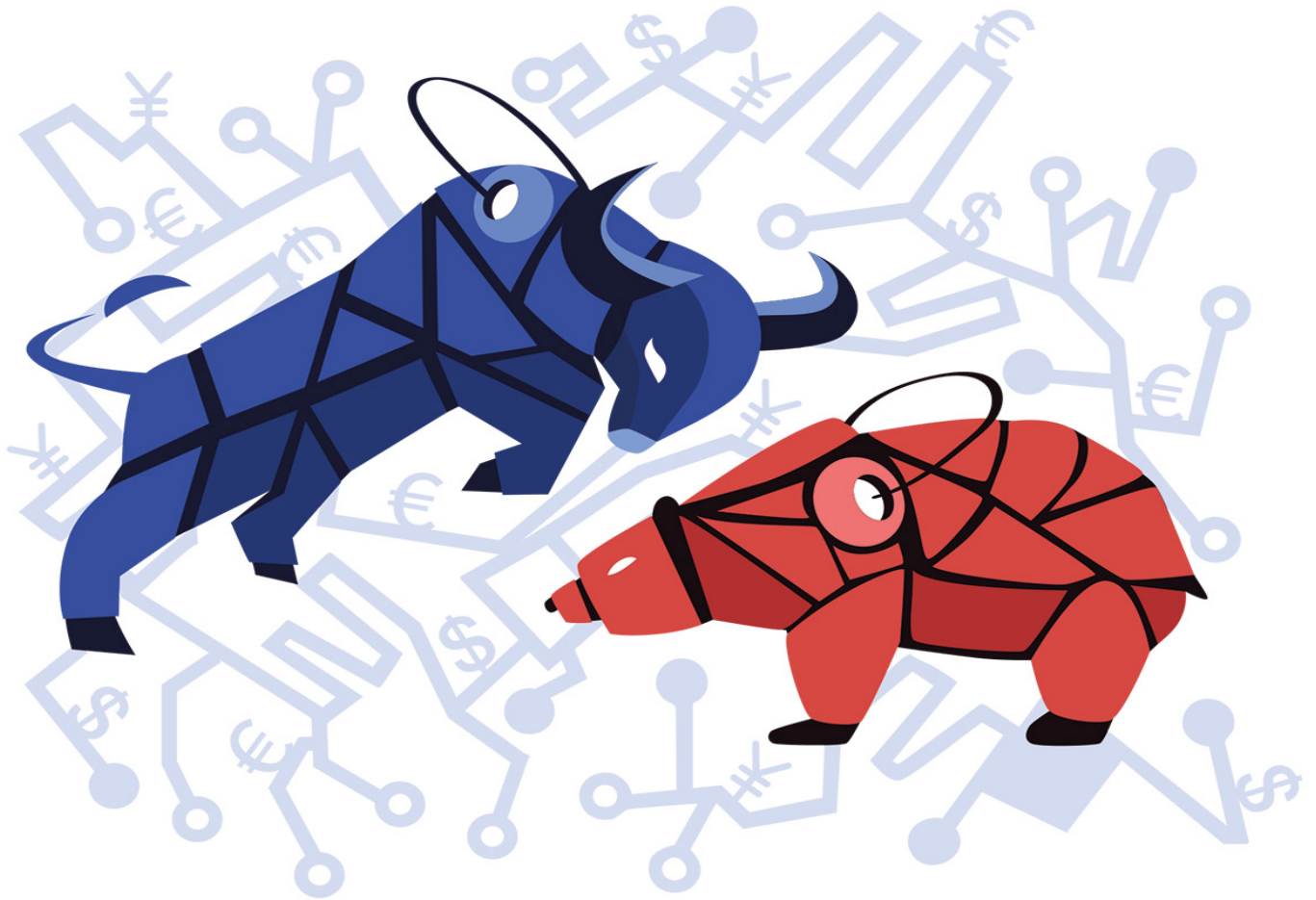


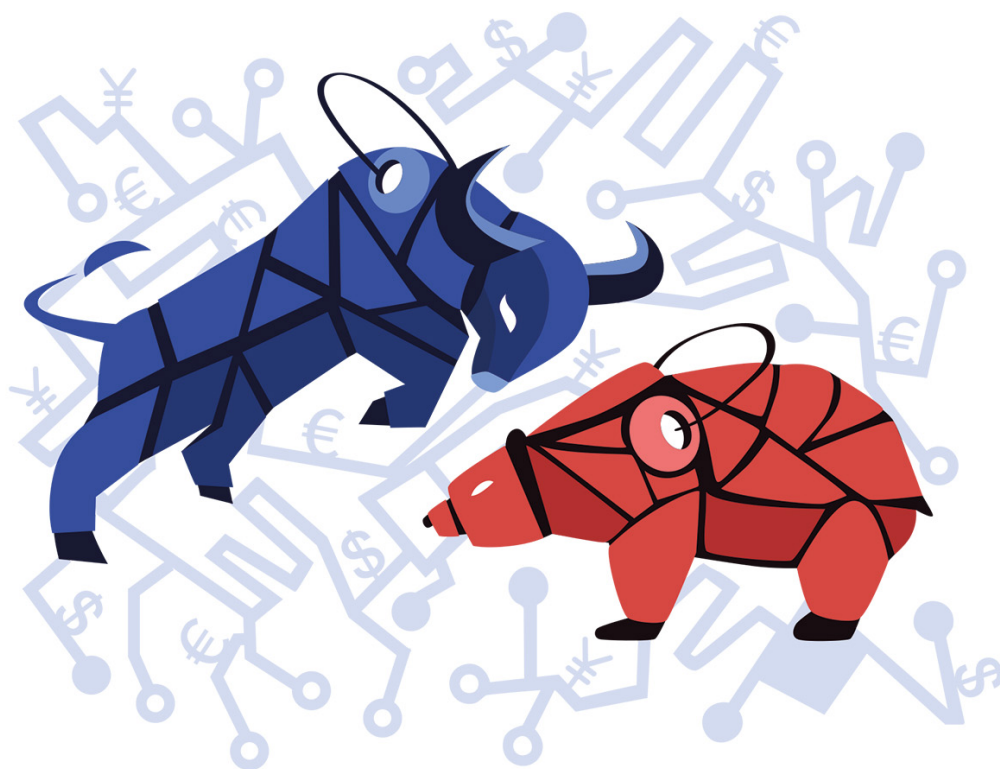
GenAI Trading



**Strategie intelligenti
per gestire patrimoni,
investimenti e portafogli**

H. J. MEDINA RUIZ, E. CHAN

GenAI Trading



**Strategie intelligenti
per gestire patrimoni,
investimenti e portafogli**

APOGEO

GENAI TRADING
STRATEGIE INTELLIGENTI PER GESTIRE PATRIMONI, INVESTIMENTI E
PORTAFOGLI

Hamlet Jesse Medina Ruiz e Ernest P. Chan

APOGEO

© Apogeo - Giangiacomo Feltrinelli Editore S.r.l.

ISBN ebook: 9788850340057

Original English–language edition published with the title “Generative AI for Trading and Asset Management”. Copyright (c) 2025 by John Wiley & Sons, Inc. All rights reserved, including rights for text and data mining and training of artificial intelligence technologies or similar technologies. This translation published under license with the original publisher John Wiley & Sons, Inc.

Giangiacomo Feltrinelli editore srl, gli autori e qualunque persona o società coinvolta nella scrittura, nell’editing o nella produzione (chiamati collettivamente “Realizzatori”) di questo libro (“l’Opera”) non offrono alcuna garanzia sui risultati ottenuti da quest’Opera. Non viene fornita garanzia di qualsivoglia genere, espressa o implicita, in relazione all’Opera e al suo contenuto. L’Opera viene commercializzata COSÌ COM’È e SENZA GARANZIA. In nessun caso i Realizzatori saranno ritenuti responsabili per danni, compresi perdite di profitti, risparmi perduti o altri danni accidentali o consequenziali derivanti dall’Opera o dal suo contenuto.

Il presente file può essere usato esclusivamente per finalità di carattere personale. Tutti i contenuti sono protetti dalla Legge sul diritto d’autore.

Nomi e marchi citati nel testo sono generalmente depositati o registrati dalle rispettive case produttrici.

[L’edizione cartacea è in vendita nelle migliori librerie.](#)

~

Sito web: www.apogeeonline.com

Scopri le novità di Apogeo su [Facebook](#)

Seguici su [X](#)

Collegati con noi su [LinkedIn](#)

Guarda cosa stiamo facendo su [Instagram](#)

Rimani aggiornato iscrivendoti alla nostra [newsletter](#)

Ai miei genitori, Denis e Herinarco, e a mia nonna, Gloria María.

Alla mia famiglia: Ben, Sarah ed Ethan.

Introduzione

Oggi esistono grosso modo tre tipi di modelli di intelligenza artificiale: discriminativi, generativi e per rinforzo. La maggior parte dei gestori di patrimoni ha una certa familiarità con i modelli discriminativi (per esempio, dato il rendimento di ieri, qual è la probabilità che il rendimento di oggi sia positivo?); alcuni hanno familiarità anche con l'apprendimento per rinforzo (per esempio, come possiamo ottimizzare il prezzo di vendita per ottenere un profitto migliore?). Ma l'intelligenza artificiale generativa, cui d'ora in poi faremo riferimento con l'acronimo GenAI, è un'invenzione recente che suscita molto clamore, ma spesso viene considerata un sinonimo di LLM (*Large Language Model*) o come uno strumento per la generazione di immagini. Ma la GenAI può apprendere da qualsiasi cosa, non solo da testo o immagini. In particolare, può apprendere da serie temporali di rendimenti delle attività, e questo è un argomento forse più rilevante per i gestori patrimoniali.

In questo libro, approfondiamo sia le applicazioni sia i fondamenti della GenAI. Il libro si divide in due parti: utilizzo senza codice dell'intelligenza artificiale generativa per i trader e i gestori patrimoniali con *poca* esperienza di programmazione; fondamenti dell'intelligenza artificiale generativa e sue applicazioni nella finanza per coloro che hanno una buona padronanza della programmazione e dell'intelligenza artificiale discriminativa. Di conseguenza, chi sente di appartenere all'una o all'altra categoria è libero di dare solo una scorsa ai capitoli dell'altra parte.

Nei primi due capitoli della Parte I, esamineremo esempi di come utilizzare la versione no-code della GenAI per fare cose che la maggior parte dei trader e degli investitori incontrerà nella propria vita. Per esempio, come recuperare da Internet i prezzi rettificati di un ETF; come calcolare le metriche di performance più comuni basandosi su un foglio di calcolo pieno di prezzi giornalieri; come convertire il codice di backtest di una strategia di trading da Matlab a Python (utilizzando come esempio il codice Matlab del libro *Machine Trading* di Ernie); come ideare codice Python basandosi su una specifica della strategia; come riassumere un articolo su una strategia di trading e trasformarlo in codice di backtest. Ciò che *non* saremo in grado di mostrarvi è “ChatGPT, inventa una strategia di trading redditizia che io possa usare con facilità”. In questa fase di sviluppo della GenAI, questo livello di creatività non è ancora stato raggiunto.

Le strategie per le quali chiediamo un aiuto alla GenAI includono una strategia fattoriale long-short, una strategia di carry sui futures del VIX e una strategia di calendar spread su opzioni SPX. Abbiamo anche chiesto alla GenAI di condurre una ricerca bibliografica sulle tecniche di ottimizzazione del portafoglio basate sul *Deep Reinforcement Learning*.

Per questi due capitoli, utilizzeremo l’interfaccia più comunemente conosciuta per questi esempi: la versione web di ChatGPT GPT-4o che, al momento della stesura, era la più aggiornata, e Microsoft Copilot. Potete eseguire la maggior parte di questi compiti in modo altrettanto efficace utilizzando Google Gemini Pro, X.AI Grok o DeepSeek, che tuttavia non abbiamo provato in questo libro.

Il terzo capitolo della Parte I di questo libro è un tour intensivo sulle tecniche di machine learning comunemente utilizzate nella gestione patrimoniale. Spaziano dall’apprendimento non supervisionato (*unsupervised learning*) all’apprendimento

supervisionato (*supervised learning*) all'apprendimento per rinforzo (*reinforcement learning*). Il capitolo tratta anche alcune tecniche utili per evitare l'overfitting e per la scelta del modello, come la regolarizzazione e l'ottimizzazione degli iperparametri. Tratta inoltre varie sfumature dell'ingegneria dei dati e delle feature, che spesso sono importanti tanto quanto la scelta del modello di machine learning. Il capitolo può essere utilizzato come un'introduzione per i professionisti della finanza alle prime armi con l'intelligenza artificiale, o come un ripasso per coloro che già la usano. Fino a circa il 2022, quando fu lanciato ChatGPT, questa era tutta l'intelligenza artificiale che la maggior parte dei gestori patrimoniali avrebbe mai imparato.

La Parte II del libro approfondisce i fondamenti e i dettagli tecnici della GenAI. Il Capitolo 4 chiarisce la differenza tra AI discriminativa e generativa e introduce le principali famiglie generative: modelli autoregressivi profondi e modelli profondi con variabili latenti, come gli autoencoder variazionali, i modelli di flusso e le GAN (*Generative Adversarial Network*). Ognuna di queste famiglie avrà un proprio capitolo nel resto della Parte II. Ogni capitolo esplora come questi modelli sono stati sviluppati e mostra come adattarli alle dinamiche delle serie temporali finanziarie, con esempi pratici di codice su notebook che utilizzano dati finanziari. Il libro si conclude con alcune applicazioni end-to-end, mostrando come questi modelli possono pre-elaborare dati alternativi, generare segnali di trading ed essere ottimizzati per rendere efficiente l'inferenza. Il Capitolo 9 è interamente dedicato all'applicazione: come sfruttare gli LLM (*Large Language Model*) per l'analisi del sentiment nel trading. Il Capitolo 10 riguarda l'implementazione pratica di questi sistemi – in particolare come ottimizzare questi modelli per un'inferenza efficiente. Questo è un capitolo particolare,

tipicamente assente nelle risorse dedicate alla modellazione generativa, almeno al momento della stesura. L'inferenza efficiente è un ambito su cui Hamlet ha lavorato negli ultimi anni nella sua azienda, dove fattori come la scalabilità e l'economia non erano negoziabili. Ancora una volta, in tutti i casi troverete forniti i notebook Python che implementano la maggior parte di queste tecniche. Concludiamo con il Capitolo 11, che riassume le principali tecniche trattate in questo libro. Sottolineiamo anche il ruolo della conoscenza del dominio per progettare strategie di trading significative, in particolare quando si utilizzano gli LLM come supporto alle decisioni, e discutiamo il motivo per cui la GenAI, nonostante il suo successo in altri domini, nel trading potrebbe richiedere più lavoro empirico per ottenere risultati simili. Infine, delineiamo alcune promettenti direzioni di ricerca in cui la GenAI potrebbe avere un impatto significativo sul trading.

Questo libro non ha lo scopo di dimostrare teoremi matematici. Piuttosto, cerchiamo di offrire al lettore sufficienti conoscenze di base per aiutarlo ad adattare le tecniche ai propri problemi specifici. Hamlet, con la sua esperienza nel risolvere problemi complessi nella sua azienda, nei suoi progetti personali e nelle competizioni di machine learning, ha scoperto che la chiave per ottenere buone soluzioni, consiste nella conoscenza, nel risalire alla formulazione matematica. Questo è particolarmente importante quando le soluzioni più popolari o più ampiamente accettate tendono a fallire su dataset "veri". Questo libro è il suo tentativo di offrire questi principi ai trader e ai gestori patrimoniali. Allo stesso tempo, include molti esempi pratici di codice che il lettore può utilizzare per condurre le proprie ricerche, migliorare i modelli e sviluppare nuove soluzioni.

Laddove possibile, abbiamo utilizzato dataset del mondo reale tratti dal dominio finanziario, limitando l'uso di dataset standard

utilizzati per le pubblicazioni di ricerca e che hanno il solo scopo di spiegare i principi fondamentali. In tutto il libro, utilizziamo anche numerosi esempi che coinvolgono immagini e dati testuali. Questi esempi non solo aiutano a comprendere i concetti, ma illustrano anche tecniche che hanno avuto molto successo in altri campi. Hanno anche implicazioni dirette per la pre-elaborazione di dati alternativi nella finanza e per la loro integrazione nelle strategie di trading. Inoltre, offriamo spunti su come queste tecniche possano catturare le caratteristiche peculiari dei dati finanziari. Alcune delle tecniche possono sembrare lontane dalle applicazioni finanziarie, ma la chiave per trovare conoscenze dove prima non esistevano spesso arriva dal prendere in prestito tecniche da un altro dominio.

La Parte I tratta ciò che Ernie ha appreso impiegando l'intelligenza artificiale e il machine learning negli ultimi trent'anni, iniziando nel gruppo sul machine learning del T. J. Watson Lab di IBM, e finendo con l'applicare l'intelligenza artificiale nell'hedge fund e nella startup che ha fondato (qtscm.com e predictnow.ai).

La Parte II si basa sull'esperienza di Hamlet in Criteo come Chief Data Scientist, dove ha lavorato su svariate applicazioni di intelligenza artificiale su larga scala, sulla sua esperienza nello sviluppare le proprie strategie di trading, sulla sua partecipazione nel raggiungere posizioni top 1% in varie competizioni di machine learning, su temi finanziari e di altro genere, su piattaforme come Kaggle e Numerai, sul suo apprendimento personale e sulla sua formazione formale in ambito finanziario.

Il viaggio di Hamlet nella finanza è iniziato, come per molti altri, da outsider. Ha iniziato leggendo qualsiasi libro trovasse interessante (curiosità: ciò includeva tutti i libri di Ernie). Si rese ben preso conto che, come molte altre persone che cercano di entrare in un nuovo campo, capita spesso di scoprire che metodi simili esistono anche nel

proprio campo, ma con nomi, notazioni e applicazioni differenti. Più tardi, formalizzando il suo percorso con un MicroMasters in Finance al MIT, si rese conto che non esisteva una singola risorsa che raggruppasse tutti gli argomenti che aveva appreso o applicato negli anni. Questo libro è il suo tentativo di colmare tale lacuna.

Se la Parte II di questo libro vi sembra difficile, è perché i concetti che espone sono veramente profondi e rivoluzionari. Leggetela più volte, se lo ritenete necessario. Dopo la quarta lettura, e dopo aver provato alcuni degli esempi di codice, forse potrete applicare le tecniche di deep learning e GenAI ai vostri problemi e scoprire conoscenze che nessuno ha mai trovato prima di voi.

Siamo sicuri che quando questo libro sarà pubblicato, avremo già acquisito una migliore comprensione e anche strumenti migliori per applicare l'intelligenza artificiale alla finanza, e pertanto pubblicheremo le nostre ultime scoperte nei nostri account dei social media (x.com/echanQT, x.com/hamletjmedina, e sul nostro blog substack.com/@gatambook). Inoltre, come disse Blaise Pascal: “Se avessi avuto più tempo, ti avrei scritto una lettera più breve”. Speriamo di espandere ulteriormente i concetti qui trattati tramite nostri account sui social media, e tramite i nostri workshop.

Ringraziamenti

Hamlet Jesse Medina Ruiz

Il percorso di scrittura di questo libro è stato profondamente gratificante, ma, trattandosi del mio primo libro, anche piuttosto impegnativo. Desidero ringraziare le numerose persone che mi hanno supportato in tutto questo percorso.

- A Criteo, per avermi permesso di lavorare sul lato applicativo del business, affrontando per molti anni una grande varietà di applicazioni complesse e su larga scala, supportandomi anche nel mio voler dedicare del tempo al lato della ricerca, in questo campo in rapida evoluzione. Ai miei colleghi: questa combinazione ha plasmato profondamente il modo in cui oggi affronto i problemi e trovo soluzioni.
- A mio fratello, Yanick Medina, attualmente studente magistrale in intelligenza artificiale, che è stato un collaboratore inestimabile per il codice nella Parte II di questo libro. Ha revisionato meticolosamente e sottoposto a test il codice, fornendo feedback che ci hanno aiutato a garantire un buon equilibrio tra modularità e accessibilità del codice, un principio che cerchiamo sempre di mantenere.
- Alla mia famiglia e ai miei amici, il cui supporto e incoraggiamento sono stati fondamentali. Un ringraziamento speciale a mio padre, che fin da quando ero molto piccolo mi ha insegnato la bellezza della matematica.

- A Dio, la cui infinita misericordia ha reso possibile tutto questo.

Ernest Chan

Desidero ringraziare Roger Hunter, CTO di QTS Capital Management (qtscm.com), per la sua partnership nel lavorare a molti progetti di intelligenza artificiale dedicati alla finanza nel corso degli anni, in particolare per aver co-sviluppato il workshop *Lifecycle of Trading Strategy Development with Machine Learning* (epchan.com/workshops) e il workshop *Generative AI for Asset Managers* (predictnow.ai/generative-ai-workshop). Sono anche grato che l'attuale CEO di QTS Capital Management (qtscm.com), Nahid Jetha, abbia continuato nello sviluppo dell'intelligenza artificiale a vantaggio dei nostri investitori.

Desidero anche ringraziare i membri attuali e passati del mio team tecnico di Predictnow.ai per il loro contributo nel portare l'intelligenza artificiale a molti asset manager: Johann Abraham, Sergei Belov, Pavan Dutt, Haoyu Fan, Guillaume Goujard, Andrew Inscore, Nancy Khullar, Nancy Xin Man, Uttej Mannava, Akshay Nautiyal, Sudarshan Sawal, Jean Silva, Jai Sukumar e Quentin Viville.

Naturalmente, questo libro non avrebbe visto la luce senza il supporto di Wiley, in particolare senza i preziosi contributi della nostra editor Stacey Rivera, dell'assistente editoriale Katherine Cording, della redattrice Sheryl Nelson, dello specialista dei contenuti Bala Shanmugasundaram e dell'editor esecutivo Bill Falloon. Lavorare con loro è stato davvero un piacere.

Gli autori

Hamlet Jesse Medina Ruiz ricopre la posizione di Chief Data Scientist presso Criteo. È specializzato in previsioni di serie temporali, machine learning, deep learning e intelligenza artificiale generativa. Esplora attivamente il potenziale delle nuove tecnologie di intelligenza artificiale in diverse applicazioni.

Vanta una laurea in ingegneria elettronica presso la Universidad Rafael Bellosó Chacín in Venezuela, nonché due lauree magistrali con lode in matematica e machine learning presso l'Institut Polytechnique de Paris e l'Université Paris-Saclay. Inoltre, ha conseguito un dottorato di ricerca in fisica presso l'Université Paris-Saclay. Hamlet ha costantemente ottenuto il primo posto e si è classificato tra i primi dieci in concorsi globali di machine learning, guadagnando i titoli di Kaggle Expert e Numerai Expert. Recentemente, ha anche conseguito un MicroMaster in finanza presso la MIT's Sloan School of Management.

Ernest Chan (Ernie) è fondatore e Chief Scientific Officer di Predictnow.ai (<https://www.predictnow.ai>), che offre soluzioni di ottimizzazione adattiva guidate dall'intelligenza artificiale al settore finanziario e non solo. È anche fondatore e presidente non esecutivo di QTS Capital Management (<https://qtscm.com>), un CTA/CPO quantitativo dal 2011. Ha iniziato la sua carriera come ricercatore di machine learning presso il gruppo di modellazione linguistica del T.J. Watson Research Center di IBM, che ha prodotto alcuni dei più noti gestori di fondi quantistici. Ernie è l'acclamato autore di quattro libri

precedenti, *Quantitative Trading* (2a ed.), *Algorithmic Trading*, *Machine Trading* e *Hands-on AI Trading*, tutti pubblicati da Wiley. Maggiori informazioni su questi libri e sui workshop di Ernie su temi di investimento quantitativo e machine learning sono disponibili su <https://epchan.com>. Ha conseguito il dottorato di ricerca in fisica presso la Cornell University e la laurea in fisica presso l'University of Toronto.

Parte I

L'AI generativa per il trading e la gestione degli investimenti: un'introduzione no-code

In questa Parte

- Capitolo 1 - Elementi di AI generativa no-code per la finanza quantitativa
- Capitolo 2 - AI generativa no-code per lo sviluppo di strategie di trading
- Capitolo 3 - Il machine learning nella gestione degli investimenti

Capitolo 1

Elementi di AI generativa no-code per la finanza quantitativa

In questo capitolo vogliamo dimostrare come la GenAI possa essere utilizzata per svolgere quei compiti per i quali i trader e gli investitori assumevano un programmatore professionista. Scopriremo in che misura abbiamo avuto successo. In tutti gli esempi che seguono, abbiamo utilizzato un abbonamento ChatGPT GPT-4o (al costo di 20 dollari al mese al momento della stesura) poiché è il servizio GenAI più noto, ma potete provare Microsoft Copilot (che ha accesso diretto a Internet e spesso fornisce risposte differenti da ChatGPT a causa di un “fine-tuning” differente), Gemini Pro di Google (19,99 dollari al mese), Grok di xAI o DeepSeek per vedere come si comportano.

Vedremo che, sebbene possiamo inserire istruzioni in inglese, spesso l’output migliore di ChatGPT è dato dal codice e non dalle risposte numeriche.

Ernie ha competenze di base di programmazione Python ma è molto più esperto in Matlab. Quanto segue è quindi scritto dal punto di vista di un programmatore Python principiante ma anche di un programmatore Matlab esperto. Vediamo se ChatGPT può aiutare una persona con queste caratteristiche a creare codice Python utile, sia a partire da istruzioni in inglese, sia traducendo listati Matlab.

Chiederemo a ChatGPT di fare quanto segue.

1. Recuperare i prezzi storici *rettificati* di SPY e BIL.
2. Calcolare i rendimenti giornalieri in base a quei prezzi rettificati.
3. Calcolare l'indice di Sharpe da quei rendimenti giornalieri.
4. Analizzare i dati contenuti in un foglio di calcolo di Excel.
5. Tradurre in codice Python dei listati Matlab che utilizzano l'ottimizzazione quadratica per calcolare la frontiera efficiente di un portafoglio di ETF.
6. Utilizzare quel codice Python per tracciare il grafico della frontiera efficiente.

In questo e nel prossimo capitolo, utilizzeremo le seguenti convenzioni per denotare i nostri prompt inviati a ChatGPT, le risposte di ChatGPT e i nostri commenti.

Per i prompt dell'utente, utilizzeremo il seguente formato.

Prompt dell'utente

Questo è un prompt dell'utente.

Per l'output di ChatGPT, incluso il codice, utilizzeremo il seguente formato.

Risposta di ChatGPT

```
# Ecco un fantastico esempio di codice Python.  
import my_genai_trading_system # :)  
% Ed ecco un esempio di codice MATLAB.  
load_system('my_genai_trading_system') % :)
```

Quando includiamo i nostri commenti nella risposta di ChatGPT, li racchiuderemo in “[Nota dell'autore: ...]”, come segue.

Risposta di ChatGPT

```
# Ecco un fantastico esempio di codice Python.  
import scipy # [Nota dell'autore: Dovrebbe essere 'scipy', non 'spicy'.]
```

Il codice, i dati e le risorse che vengono generati o richiesti per questo capitolo si trovano nella cartella del capitolo all'interno del repository del libro: <https://github.com/genai-for-traders/genai-for-trading-and-am>. Per questo capitolo, i file si trovano nella cartella chapter01 e includono gli script, i notebook Jupyter, i file di dati di input e tutto il codice necessario per scaricare o generare i dati.

Recupero dei dati storici di mercato

Come sanno i lettori del primo libro di Ernie (Chan, 2021), quando proviamo una strategia di trading su un'azione, dobbiamo calcolarne i rendimenti giornalieri. Per calcolare correttamente i rendimenti giornalieri, dobbiamo utilizzare i prezzi di chiusura rettificati per split e dividendi. Immaginate se un'azione avesse uno split 1 a 2 prima dell'apertura del mercato nel giorno t . Senza la rettifica per gli split, il rendimento dal giorno $t - 1$ al giorno t sarebbe di circa -50%. Quindi, abbiamo inserito in ChatGPT uno dei prompt più ovvi.

Prompt dell'utente

Puoi recuperare i prezzi di chiusura giornalieri dell'ETF SPY per gli ultimi 100 giorni? Assicurati che siano rettificati per split e dividendi.

Risposta di ChatGPT

Sembra che al momento non abbia accesso alla libreria yfinance, che utilizzo solitamente per recuperare i dati finanziari. Pertanto, non posso recuperare direttamente i prezzi giornalieri dell'ETF SPY per gli ultimi 100 giorni.

Tuttavia, puoi ottenere facilmente queste informazioni utilizzando servizi di dati finanziari come Yahoo Finance, Google Finance o Bloomberg. Queste piattaforme forniscono dati storici completi, inclusi gli aggiustamenti per split e dividendi. Se vuoi, posso guidarti attraverso il processo di recupero di questi dati utilizzando queste piattaforme.

Nonostante la risposta apparentemente poco utile, come assistente no-code, ha fornito uno script Python decente che potrebbe portare a termine il compito, che abbiamo incluso nell'Appendice. Ne parleremo più avanti. Supponiamo di utilizzare quello script per scaricare sul nostro desktop i prezzi elencati di seguito di caricarli come file csv in ChatGPT:

```
2023-08-25: $438.42
2023-08-28: $441.20
2023-08-29: $447.58
...
2023-12-11: $461.99
```

Abbiamo posto questa domanda il 12/12/2023, quindi i prezzi recuperati terminano correttamente l'11/12/2023. Sono identici a quelli che avreste visto su Yahoo Finance se lo aveste fatto anche voi quello stesso giorno. È difficile verificarlo in una data successiva (per esempio ora che state leggendo questa pagina). I prezzi rettificati cambiano a seconda di quando li recuperate, ma dovrebbero produrre gli stessi rendimenti giornalieri indipendentemente da come vi appaiano i prezzi. Quindi, ora calcoliamo i rendimenti giornalieri.

Prompt dell'utente

Calcola i rendimenti giornalieri sulla base dei prezzi sopra indicati.

Risposta di ChatGPT

Per calcolare i rendimenti giornalieri, usiamo la formula:

$$\text{Rendimento giornaliero} = \left(\frac{\text{Prezzo}_{\text{oggi}} - \text{Prezzo}_{\text{ieri}}}{\text{Prezzo}_{\text{ieri}}} \right) \times 100\%$$

Calcolerò i rendimenti giornalieri per l'ETF SPY sulla base dei prezzi di chiusura rettificati forniti:

- 1. 2023-08-28: $((441.20 - 438.42) / 438.42) \times 100\%$.*
- 2. 2023-08-29: $((447.58 - 441.20) / 441.20) \times 100\%$.*

... e così via per ogni giorno successivo.

Calcoliamo questi rendimenti.

Ecco i rendimenti giornalieri per i primi cinque giorni:

1. 2023-08-28: $((441.20 - 438.42) / 438.42) \times 100\% = 0.63\%$.

2. 2023-08-29: $((447.58 - 441.20) / 441.20) \times 100\% = 1.45\%$.

...

Queste percentuali rappresentano il rendimento giornaliero per ogni rispettivo giorno. Vuoi che continui a calcolare i rendimenti giornalieri per i giorni rimanenti?

Sebbene queste formule siano corrette, l'output è difficile da scaricare e verificare. Quindi chiediamo quanto segue.

Prompt dell'utente

Puoi formulare l'output sopra come un file csv di due colonne?

Risposta di ChatGPT

Certamente! Ecco i dati formattati come file CSV con due colonne, "Data" e "Rendimento Giornaliero":

Data, Rendimento Giornaliero

2023-09-12, -0.0055

2023-09-13, 0.0012

...

2023-12-11, 0.0039

Possiamo copiare e incollare questo output in un file di testo, SPY Daily Returns.csv e poi aprirlo in Excel. Controllando alcuni prezzi intorno alla data di stacco del dividendo del 15/09/2023, verifichiamo che i rendimenti sono stati calcolati correttamente, sulla base dei prezzi di chiusura rettificati.

Anticipando che avremo bisogno del tasso privo di rischio per calcolare l'indice di Sharpe e altre metriche di performance per SPY (o altre strategie di trading), chiediamo quanto segue.

Prompt dell'utente

Nello stesso lasso di tempo, recupera il tasso privo di rischio giornaliero in forma percentuale.

Risposta di ChatGPT

Recuperare il tasso privo di rischio giornaliero in un periodo specifico, come quello per i prezzi di chiusura rettificati dell'ETF SPY, può essere un po' più complesso, perché non esiste una fonte diretta che fornisca questi tassi in un formato giornaliero adatto per il download immediato e l'uso nei calcoli finanziari.

(Probabilmente, quando leggerete questa pagina, ChatGPT avrà già integrato diverse fonti di dati che gli permetteranno di scaricare il rendimento dei Treasury USA a tre mesi.)

In altre parole, non può farlo, a meno che i dati non siano disponibili su Yahoo Finance. Ma esiste un proxy che possiamo usare per i rendimenti privi di rischio: l'ETF SPDR Bloomberg 1-3 Month T-Bill ("BIL"). Lo recupereremo con una semplice modifica dello script Python descritto nel paragrafo "Recupero dei prezzi di chiusura rettificati e calcolo dei rendimenti giornalieri" dell'Appendice. Possiamo usare il codice presentato nel paragrafo "Tracciamento del tasso privo di rischio nel corso degli anni" dell'Appendice per calcolare i rendimenti giornalieri di BIL, che useremo in tutto il libro come tasso privo di rischio (giornaliero).

Come ogni data scientist ben sa, uno dei modi migliori per verificare se i risultati di un calcolo sono corretti consiste nel tracciarli su un grafico.

Prompt dell'utente

Traccia il grafico dei rendimenti annualizzati con lo stesso asse x. Ho solo bisogno di avere YYYY come etichette dell'asse x.

Potreste pensare che la risposta a un prompt così semplice sia immediata, ma non è così. Al primo tentativo, ChatGPT ha annualizzato erroneamente i rendimenti moltiplicando i rendimenti

giornalieri per $\sqrt{252}$ invece che per 252. In un altro tentativo, non è riuscito a interpretare correttamente il formato YYYY-MM-DD hh:mm:ss-05:00.

Prompt dell'utente

Che cosa significa il formato YYYY-MM-DD hh:mm:ss-05:00 per la data e l'ora?

La risposta a questo prompt è corretta. L'ultimo hh:mm è l'offset del fuso orario rispetto all'UTC. Per esempio, a dicembre, l'orario YYYY-MM-DD hh:mm:ss a New York avrà -05:00, alla fine.

Dopo ore (eh già!) di prompting, ha finalmente prodotto il grafico corretto. Abbiamo riprodotto il codice Python associato al grafico nel paragrafo “Tracciamento del tasso privo di rischio nel corso degli anni” dell'Appendice. Vi raccomandiamo di controllare bene quel codice, perché vedrete che i dati grezzi non sono utilizzabili. In più, ci dà una visione d'insieme dell'economia statunitense dal 2007.

Calcolo dell'indice di Sharpe

Con questa “fatica” con il prompting (ChatGPT è come un tirocinante mediocre e un po' inesperto), abbiamo finalmente scaricato i dati sui nostri ETF preferiti SPY e BIL, calcolato i rendimenti giornalieri di entrambi, e ora possiamo usare questi dati per calcolarne l'indice di Sharpe. Come suggerito in precedenza, vogliamo che ChatGPT ci fornisca il codice Python per il calcolo, invece di generare direttamente i risultati numerici.

Prompt dell'utente

Supponiamo che una serie di rendimenti giornalieri A sia memorizzata in A.csv e che il tasso privo di rischio giornaliero sia memorizzato in B.csv, e che le date di A e B non siano identiche, sebbene si sovrappongano, come calcoliamo l'indice di Sharpe di A? Presenta il codice Python.

Trovate la risposta (in larga parte corretta) nel paragrafo “Calcolo dell’indice di Sharpe di SPY” nell’Appendice. L’indice di Sharpe calcolato nel paragrafo “Calcolo dell’indice di Sharpe di SPY” nell’Appendice è per il periodo dal 25/08/2023 all’11/12/2023, che è stato molto rialzista. Ovviamente, potete adattare questi prompt e i relativi codici Python ad altre metriche di performance come il massimo drawdown (la “peggior caduta del capitale”), il Calmar ratio e altri simili.

Formattazione e analisi dei dati

Non tutti i fogli di calcolo che ci capitano saranno ben formattati come quelli scaricati da Yahoo. Uno dei compiti più ingrati e che richiede più tempo per un analista o data scientist, è formattare correttamente il file di input in modo da poterlo analizzare. Per esempio, un collega ci ha inviato il file `Indices.xlsx` salvato in <https://github.com/genai-for-traders/genai-for-trading-and-am/chapter01/>. Abbiamo anonimizzato i nomi degli indici in “A”, “B” e “C”, ma abbiamo lasciato tutto il resto invariato. Il file contiene molti “detriti” qua e là, nei nomi degli indici, nelle date di inizio e fine dei dati più valori #N/A per quelle date che non avevano dati. Vediamo se ChatGPT può rendere più facile il compito di analizzare questi dati.

Prompt dell'utente

Questo file contiene quattro colonne di dati sui prezzi. Puoi dirmi i nomi delle colonne?

[Nota dell'Autore: possiamo allegare un file al prompt in ChatGPT.]

Risposta di ChatGPT

Sfortunatamente, ho riscontrato un errore durante il tentativo di accedere e leggere il file che hai appena caricato. Questo potrebbe

essere dovuto a varie ragioni, come problemi di formato del file o di compatibilità.

Non “butta bene”! Quando abbiamo provato a farlo un paio di mesi fa, ChatGPT non aveva avuto problemi ad analizzare questi stessi dati. Dopo diverse ore di tentativi ed errori, ha funzionato una soluzione banale: abbiamo semplicemente fatto logout e login. Spesso questo è frustrante con ChatGPT, data la natura non deterministica e non riproducibile dei risultati. La risposta è stata la seguente.

Risposta di ChatGPT

Il file Excel che hai fornito contiene le seguenti colonne:

- 1. Unnamed: 0*
- 2. Unnamed: 1*
- 3. [Una colonna denominata con una data: 2000-01-03]*
- 4. Unnamed: 3*
- 5. Unnamed: 4*
- 6. Unnamed: 5*
- 7. Unnamed: 6*
- 8. Unnamed: 7*
- 9. Unnamed: 8*
- 10. Unnamed: 9*

...

ChatGPT ha bisogno di un piccolo aiuto.

Prompt dell'utente

Usa la riga 6, colonne da G a J per i nomi delle colonne per i dati sui prezzi.

Risposta di ChatGPT

Indice A

Indice B

Indice C

Indice D

...

Il risultato ora è corretto.

Prompt dell'utente

Calcola i rendimenti giornalieri di questi quattro indici.

Risposta di ChatGPT

Sembra ci sia stato un errore nel calcolo dei rendimenti giornalieri, probabilmente a causa della presenza di dati non numerici nelle colonne.

I NaN stanno mettendo in difficoltà ChatGPT.

Prompt dell'utente

Per quelle celle con NaN, puoi restituire NaN per i rendimenti. Ma altrimenti, visualizza i risultati numerici.

Dopo questo prompt, ChatGPT ha tentato di calcolare i rendimenti giornalieri per diversi sottoinsiemi dei dati, e senza la colonna delle date. Infine, abbiamo inviato il seguente prompt.

Prompt dell'utente

Voglio che l'analisi parta dalla prima riga fino all'ultima riga e restituisca un file .csv per i risultati.

ChatGPT ha calcolato i rendimenti giornalieri correnti per tutti i dati e li ha salvati in un file .csv. Ecco un esempio:

Prompt dell'utente

Quali sono i rendimenti del 06/11/2023?

Risposta di ChatGPT

I rendimenti giornalieri per gli indici il 6 novembre 2023 sono i seguenti:

Indice A: 0.001378

Indice B: -0.004321

Indice C: -0.003129

Indice D: -0.000333

...

Prompt dell'utente

Utilizzando questi rendimenti giornalieri, calcola l'indice di Sharpe annualizzato dell'Indice A dal 07/01/2019 al 03/07/2023, ipotizzando un tasso privo di rischio pari a 0.

Risposta di ChatGPT

L'indice di Sharpe annualizzato dell'Indice A per il periodo dal 7 gennaio 2019 al 3 luglio 2023, ipotizzando un tasso privo di rischio pari a 0, è approssimativamente 0.732.

Una volta controllati alcuni numeri in questo modo, è molto più comodo ed efficiente utilizzare ChatGPT per estrarre altre informazioni statistiche da tutti o da parte dei dati, invece di usare Excel o Python. ChatGPT può essere un grande strumento.

Traduzione di listati Matlab in codice Python

Chi ha letto i precedenti tre libri di Ernie sa che sono scritti in Matlab, per le ragioni spiegate nel suo terzo libro, *Machine Trading* (Chan, 2017). Anche se successivamente ha aggiunto del codice Python ai repository online dei suoi primi due libri, non c'è codice Python per *Machine Trading*. Vediamo se possiamo usare ChatGPT per tradurre i listati Matlab in Python.

Il primo esempio di codice in *Machine Trading*, `ef.m`, esegue il calcolo della frontiera efficiente di un portafoglio di ETF e la ricerca del portafoglio di tangenza (che massimizza storicamente l'indice di Sharpe). Per comodità dei lettori, lo abbiamo riprodotto nel paragrafo

“Codice Matlab per calcolare la frontiera efficiente e trovare il portafoglio di tangenza” nell’Appendice.

Ora chiediamo a ChatGPT di tradurlo in Python. Potete semplicemente caricare `ef.m` e chiederne la traduzione, oppure copiare e incollare l’intero file come testo nel prompt.

Prompt dell’utente

Traduci il seguente codice Matlab in Python:

```
% ef.m
% Trova una frontiera efficiente di un insieme di ETF
% Nota che in questo programma mean e sd sono vettori riga,
clear;
load('inputDataOHLCDaily ETF_20150417', 'stocks', 'tday', 'cl');
...
```

[Nota dell’Autore: vedi il paragrafo “Codice Matlab per calcolare la frontiera efficiente e trovare il portafoglio di tangenza” nell’Appendice.]

Risposta di ChatGPT

Per tradurre questo codice MATLAB in Python, utilizzeremo librerie come NumPy per i calcoli numerici e SciPy per l’ottimizzazione (nello specifico, la programmazione quadratica). Il codice MATLAB viene utilizzato per trovare la frontiera efficiente per un insieme di ETF, calcolare il portafoglio di tangenza e trovare il portafoglio a varianza minima. Ecco il codice Python equivalente:

```
import numpy as np
import pandas as pd
from scipy.optimize import minimize
# Supponendo che 'inputDataOHLCDaily ETF_20150417.csv' sia un file CSV
# con colonne 'stocks', 'tday' e 'cl'
# Carica i dati (dovrai sostituire a questo il tuo meccanismo effettivo
# di caricamento dati)
data = pd.read_csv('inputDataOHLCDaily ETF_20150417.csv')
stocks = data['stocks'].unique()
cl = data.pivot(index='tday', columns='stocks', values='cl')
# Rimuovi 'EWZ' e 'FXI'
cl = cl.drop(columns=['EWZ', 'FXI'])
# Calcola i rendimenti
R = cl.pct_change().dropna()
# Media e covarianza dei rendimenti
mi = R.mean()
C = R.cov()
# Prepara diversi rendimenti medi target del portafoglio per la
```

```

frontiera efficiente
m = np.linspace(mi.min(), mi.max(), 21)
v = np.zeros_like(m)
# Varianza del portafoglio da minimizzare
H = 2 * C.values
# Vincolo di vendita allo scoperto
A = -np.eye(len(mi))
b = np.zeros(len(mi))
# Nessun termine lineare nella minimizzazione
f = np.zeros(len(mi))
# Calcolo della frontiera efficiente
for i in range(len(m)):
    Aeq = np.vstack([mi, np.ones(len(mi))])
    beq = np.array([m[i], 1])
    def objective(F):
        return F @ H @ F
    constraints = ({'type': 'eq', 'fun': lambda F: np.dot(Aeq[0], F) -
    beq[0]},
                  {'type': 'eq', 'fun': lambda F: np.dot(Aeq[1], F) -
    beq[1]})
    bounds = [(0, None)] * len(mi)
    result = minimize(objective, np.ones(len(mi)) / len(mi),
    bounds=bounds, constraints=constraints)
    v[i] = result.fun
sd = np.sqrt(v)

plt.scatter(sd, m)
plt.xlabel('Standard Deviation')
plt.ylabel('Mean Return')
plt.title('Efficient Frontier')
plt.show()
# Trova il portafoglio di tangenza
sharpe_ratio = m / sd
idx = np.argmax(sharpe_ratio)
# Trova il portafoglio a varianza minima
idx_min = np.argmin(sd)
# Per ulteriori dettagli come i pesi del portafoglio di tangenza e di
varianza minima,
# usa 'idx' e 'idx_min' con la funzione 'minimize' come mostrato sopra.

```

ChatGPT prosegue con la sua risposta.

Risposta di ChatGPT

Questo codice fa quanto segue.

Carica i dati degli ETF e calcola i rendimenti giornalieri. Calcola la media e la covarianza dei rendimenti. Utilizza la programmazione quadratica (tramite `scipy.optimize.minimize`) per trovare la frontiera efficiente, il portafoglio di tangenza e il portafoglio a varianza minima.

Nota: il file `inputDataOHLCDaily ETF_20150417.csv` dovrebbe essere sostituito con il nome e il percorso del tuo file. Questo codice

presuppone che i dati siano in un formato lungo con colonne “stocks”, “tday” (data) e “cl” (prezzo di chiusura). Adegua il caricamento e la manipolazione dei dati in base al formato effettivo dei tuoi dati. La parte di plotting (usando matplotlib.pyplot) è minima; potresti volerla migliorare in base alle tue specifiche esigenze di visualizzazione.

Il primo ostacolo nel confrontare l’output del codice Python con i listati Matlab è che il primo si aspetta in input un file .csv in “formato lungo”, cioè presuppone che i dati dei prezzi corrispondenti alle diverse azioni siano impilati in righe un giorno alla volta. Per esempio:

```
20050512 EWC 1412
20050512 EWG 1388
20050512 EWJ 894
...
20150417 EWY 6102
20150417 EWZ 3453
20150417 FXI 5003
```

Supponendo di non essere programmatori Matlab esperti, chiediamo a ChatGPT di creare del codice Matlab che possa salvare le informazioni contenute nel file .mat di input in un file .csv.

Prompt dell'utente

Nel codice Matlab, abbiamo supposto che il file di input inputDataOHLCDaily ETF_20150417.mat abbia un array di celle stocks di lunghezza S, un array di double tday di lunghezza T e un array di double cl di dimensioni TxS. Quale codice Matlab possiamo scrivere per produrre in output un file csv che rispetti il formato previsto nel file .csv di input del codice Python?

Risposta di ChatGPT

```
% Carica i dati dal file .mat
load('inputDataOHLCDaily ETF_20150417.mat', 'stocks', 'tday', 'cl');
% Inizializza una tabella con 'tday' come prima colonna
outputData = array2table(tday, 'VariableNames', {'tday'});
% Aggiunge colonne per ogni stock, con i prezzi di chiusura come valori
for i = 1:length(stocks)
    outputData.(stocks{i}) = cl(:, i);
end
% Riformatta i dati in formato lungo
outputDataLong = stack(outputData, stocks, 'NewDataVariableName', 'cl', 'IndexVariableName', 'stocks');
```

```
% Scrive in CSV
writetable(outputDataLong, 'outputData.csv');
```

Il file di output con i dati sembra ragionevole. Rinominando `outputData.csv` in `inputDataOHLCDaily ETF_20150417.csv` potremo provare il codice Python.

Da questo punto in poi, il nostro viaggio si fa insidioso. Inizialmente, il codice che ChatGPT ha tradotto da Matlab a Python non funzionava. Ecco di seguito il tedioso scambio di prompt e risposte con cui abbiamo guidato ChatGPT a produrre codice funzionante; non perché siamo sadici, ma perché mostra che cosa occorre fare per “spremere” codice corretto da ChatGPT.

L'esecuzione del codice Python produce l'errore `ModuleNotFoundError: No module named 'scipy'`. Chiediamo aiuto a ChatGPT.

Risposta di ChatGPT

L'errore "ModuleNotFoundError: No module named 'scipy'" indica che la libreria SciPy, essenziale per la parte di ottimizzazione del codice (nello specifico, la funzione minimize usata per la programmazione quadratica), non è installata nel tuo ambiente Python.

Per risolverlo, dovrai installare la libreria SciPy. Se stai usando un'installazione standard di Python, puoi installare SciPy usando pip, il gestore di pacchetti Python. Ecco come puoi farlo:

Apri la tua interfaccia a riga di comando ed esegui il seguente comando:

```
pip install scipy
```

Cosa che abbiamo fatto. Eseguendo nuovamente il codice Python abbiamo ottenuto un altro errore:

```
NameError                                Traceback (most recent call
last)
Cell In[1], line 50
     47     v[i] = result.fun
     49     sd = np.sqrt(v)
--> 50 plt.scatter(sd, m)
     51 plt.xlabel('Standard Deviation')
     52 plt.ylabel('Mean Return')
NameError: name 'plt' is not defined
```

Chiediamo di nuovo aiuto a ChatGPT.

Risposta di ChatGPT

L'errore che stai riscontrando, "NameError: name 'plt' is not defined", indica che il modulo matplotlib.pyplot, comunemente importato come plt, non è stato importato nel tuo script Python. Questo modulo è necessario per tracciare la frontiera efficiente.

Per risolvere il problema, devi importare matplotlib.pyplot all'inizio del tuo script. Ecco il segmento corretto di codice Python con la necessaria istruzione di importazione:

```
...  
import matplotlib.pyplot as plt # Importa matplotlib per il plotting  
# [Resto del codice]
```

Gentile da parte di ChatGPT dircelo, ma, allora, perché non l'ha aggiunto fin dall'inizio? Comunque, ora il codice genera la Figura 1.1. Se volete salvarle l'output in un file .png, ricordatevi di aggiungere `plt.tight_layout()` prima di `plt.savefig(plot_file_path)`.

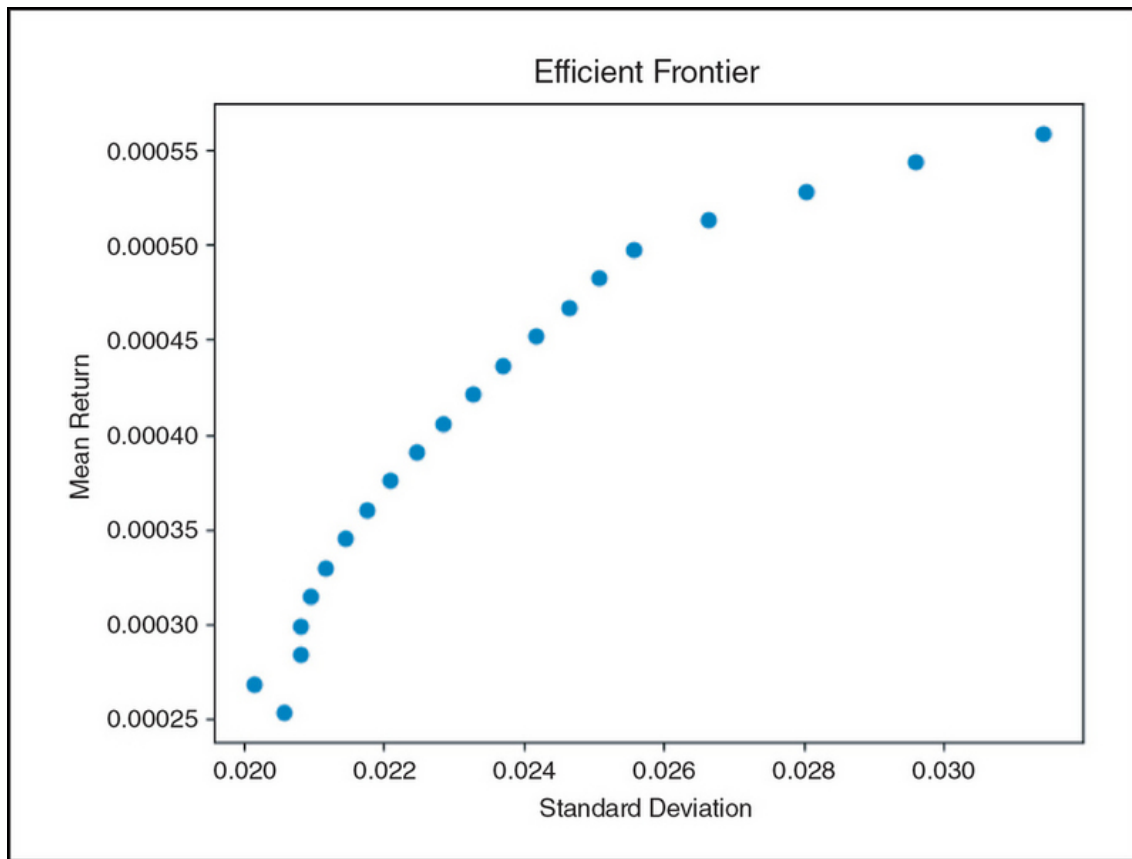


Figura 1.1 Grafico della frontiera efficiente basata sul codice Python generato da ChatGPT.

Confrontando questo grafico con la frontiera efficiente tracciata dal codice Matlab nel paragrafo “Codice Matlab per calcolare la frontiera efficiente e trovare il portafoglio di tangenza” nell’Appendice A, vediamo chiaramente che nel codice Python c’è qualcosa che non va. Le scale degli assi x e y sono corrette. Anche le coordinate del portafoglio a varianza minima (corrispondente al punto verde sul grafico di Matlab) sembrano giuste. Ma che cosa è successo ai portafogli con i due rendimenti medi più bassi? Andando nel dettaglio, i vettori dei rendimenti medi mi di entrambi i programmi coincidono. Anche le matrici di covarianza C di entrambi

i programmi coincidono. Seguendo il suggerimento di ChatGPT, calcoliamo i pesi per il portafoglio di tangenza:

```
Aeq = np.vstack([mi, np.ones(len(mi))])
beq = np.array([m[idx], 1])
def objective(F):
    return F @ H @ F
constraints = ({'type': 'eq', 'fun': lambda F: np.dot(Aeq[0], F) -
beq[0]},
               {'type': 'eq', 'fun': lambda F: np.dot(Aeq[1], F) -
beq[1]})
bounds = [(0, None)] * len(mi)
result = minimize(objective, np.ones(len(mi)) / len(mi), bounds=bounds,
constraints=constraints)
tangency_portfolio_weights = result.x
```

I pesi `tangency_portfolio_weights` risultano essere `[0.19413052, 0.2978653, 0., 0.01713741, 0., 0.49086677]` che sono molto diversi dai valori `F = [0.3815, 0, 0.6040, 0.0001, 0.0143, 0]` trovati da Matlab.

Inserendo questi pesi nell'array per gli indici di Sharpe a pesi differenti, abbiamo scoperto che Matlab ha assegnato un indice di Sharpe di `0.0283` per il portafoglio di tangenza, contro lo `0.0194` di Python. Chiaramente, l'ottimizzazione di Matlab è migliore. Questo, la differenza non è davvero colpa di ChatGPT, poiché un confronto riga per riga del codice non ha rivelato nulla di errato. In effetti, siamo piacevolmente sorpresi che sia stato in grado di trovare un pacchetto di ottimizzazione (`minimize`) in `scipy` che rispecchi ragionevolmente la funzione di ottimizzazione quadratica di Matlab. La colpa ricade quindi completamente sul pacchetto di ottimizzazione di Python. Come Ernie ha sostenuto in *Machine Trading* (Chan, 2017), Python è un “cugino povero” di Matlab: lo usiamo, ma a nostro rischio e pericolo. Mentre il codice per Matlab è sviluppato e curato commercialmente e a tempo pieno da un team di professionisti ed esperti, il codice per Python è sviluppato e curato essenzialmente da un gruppo di volontari part-time.

Piccolo pettegolezzo: è noto che il “padre” del deep learning nonché premio Nobel Geoff Hinton preferiva Matlab a Python per le sue ricerche.

Conclusioni

Abbiamo chiesto a ChatGPT di recuperare dei dati storici dal Web, di calcolare rendimenti e indici di Sharpe, di condurre un’analisi esplorativa dei dati, di applicare l’ottimizzazione quadratica per calcolare la frontiera efficiente e, in generale, di tradurre del codice Matlab in codice Python. Con una quantità di “coaching” (ovvero, prompting) fastidiosamente eccessiva e un attento controllo degli errori, concludiamo che ChatGPT è capace di svolgere questi compiti. Per lo meno, può generare una prima bozza dalla quale possiamo partire per produrre codice funzionante. Questo è particolarmente utile quando chi sta operando non è un programmatore esperto. Per esempio, trovare la sintassi esatta per generare il grafico che vogliamo in Python è noioso e richiederebbe tempo per un programmatore Matlab nativo come Ernie: un compito che ChatGPT può svolgere in modo più efficiente utilizzando istruzioni in linguaggio naturale e con un minimo spazio per risolvere gli errori gravi. La parola chiave qui è “codice”: abbiamo scoperto che è molto meglio chiedere a ChatGPT di fornirci codice che possiamo verificare (vedi Alshahwan et al., 2024) e poi modificarlo, piuttosto che chiedergli di fornirci le risposte numeriche alle nostre domande.

Nel prossimo capitolo, saremo più ambiziosi e vedremo se la GenAI può aiutarci a provare o magari suggerirci strategie di trading quantitative.

Capitolo 2

AI generativa no-code per lo sviluppo di strategie di trading

Nel capitolo precedente, abbiamo scoperto che con un po' di pazienza e un po' di prompting, possiamo far sì che ChatGPT svolga qualche attività di data science finanziaria: recuperare, esplorare e tracciare dati e calcolare su di essi alcune statistiche di performance. Sebbene le istruzioni siano tutte in inglese (o comunque in linguaggio naturale), spesso l'output è costituito da codice (generalmente Python). Lo consideriamo comunque un utilizzo no-code della GenAI.

In questo capitolo, vedremo se la GenAI no-code può gestire anche compiti più impegnativi, come trasformare la specifica di una strategia di trading in inglese in codici di backtest, e, idea ancora più ambiziosa, suggerire una strategia di trading basata su vaghe indicazioni umane. Come promemoria, ricordate che Ernie ha solo competenze di base di programmazione Python, che è molto più esperto in MATLAB e che è il fondatore di un hedge fund. Ciò significa che ha una vasta esperienza e conoscenza del dominio finanziario. Quindi, negli esempi che seguono, l'autore affronta lo sviluppo dei problemi come un esperto nel campo finanziario che non è un programmatore professionista o uno specialista in prompt engineering: in pratica come un "normale" asset manager che ha bisogno di un aiuto per programmare in un nuovo linguaggio.

Vediamo se ChatGPT può aiutarlo. Spoiler: è molto probabile che le prestazioni di ChatGPT migliorino nelle prossime versioni.

Come nel Capitolo 1, utilizziamo ChatGPT GPT-4o per i nostri test. Potete provare i prompt anche su Microsoft Copilot, Google Gemini Pro, xAI Grok o DeepSeek per vedere che cosa riescono a fare. Chiederemo a ChatGPT di fare quanto segue.

1. Creare il codice Python per il backtest di una strategia di investimento long-short su azioni specificata da Ernie, che utilizza i fattori Fama-French per prevedere i rendimenti dei singoli titoli.
2. Creare il codice Python per il backtest di una strategia di carry su futures VIX coperta, tratta da un articolo.
3. Condurre una ricerca bibliografica per trovare una tecnica di ottimizzazione del portafoglio che utilizzi il *Deep Reinforcement Learning* e che possa adattarsi ai cambi di regime.
4. Suggerire e condurre un'analisi esplorativa di una strategia di trading di calendar spread su opzioni SPX.

Ancora una volta, utilizzeremo i seguenti riquadri per denotare i nostri prompt inviati a ChatGPT, le risposte di ChatGPT e i nostri commenti.

Per i prompt dell'utente, utilizzeremo il seguente formato.

Prompt dell'utente

Questo è un prompt dell'utente.

Per l'output da ChatGPT, incluso il codice, utilizzeremo una risposta di ChatGPT.

Risposta di ChatGPT

```
# Ecco un fantastico esempio di codice Python.  
import my_genai_trading_system # :)  
% 0 un esempio di codice MATLAB.  
load_system('my_genai_trading_system') % :)
```

Se dobbiamo incorporare i nostri commenti nella sua risposta, li racchiuderemo in [Nota dell'Autore: ...], come segue.

Risposta di ChatGPT

```
# Ecco un fantastico esempio di codice Python.  
import spacy # [Nota dell'Autore: Dovrebbe essere 'scipy', non  
'spicy'.]
```

Ancora una volta, il codice, i dati e le risorse prodotti o richiesti per questo capitolo si trovano nella cartella del Capitolo 2 all'interno del repository del libro: <https://github.com/genai-for-traders/genai-for-trading-and-am>. Per questo capitolo, i file si trovano quindi nella cartella `chapter02` e includono script, notebook Jupyter, file di dati di input e tutto il codice necessario per scaricare o generare dati.

Creazione di codice da una specifica di strategia

In *Machine Trading* (Chan, 2017), Ernie ha creato una strategia-giocattolo che utilizzava il modello a tre fattori Fama-French per prevedere i rendimenti delle azioni e ha sottoposto a test le sue performance con una strategia di trading basata su quelle previsioni. Chiediamo a ChatGPT di creare tutto il codice Python necessario per il backtest, specificando la strategia in inglese. Ricordiamo che Chan, 2017 contiene solo listati Matlab.

Prompt dell'utente

Nel dialogo seguente, immagina di essere un esperto di finanza quantitativa.

Risposta di ChatGPT

...

[Nota dell'Autore: la risposta è costituita da un sacco di chiacchiere irrilevanti. Alcune ricerche suggeriscono che si ottengano risposte

migliori se prima si “prepara” ChatGPT su una determinata area applicativa. Vedi Kıcıman et al. (2023).]

Prompt dell'utente

I fattori Fama-French vengono molto usati per spiegare i rendimenti di un portafoglio di azioni. Che cosa sono i fattori Fama-French?

Risposta di ChatGPT

I fattori Fama-French sono un insieme di tre fattori (successivamente espansi a cinque) introdotti da Eugene F. Fama e Kenneth R. French. Vengono utilizzati in finanza per spiegare i rendimenti di un portafoglio azionario, andando oltre il tradizionale CAPM (Capital Asset Pricing Model), che spiega i rendimenti utilizzando solo il rischio di mercato (beta). Ecco una suddivisione di questi fattori:

I tre fattori originali

Rischio di Mercato ($R_m - R_f$): è lo stesso del CAPM e rappresenta il rendimento in eccesso di un portafoglio rispetto al tasso privo di rischio, catturando il rischio di mercato. R_m sta per il rendimento del portafoglio di mercato e R_f è il tasso privo di rischio.

Fattore Dimensione (SMB - Small Minus Big): SMB sta per “Small [capitalizzazione di mercato] Minus Big”. Cattura i rendimenti in eccesso storici delle azioni a piccola capitalizzazione rispetto a quelle a grande capitalizzazione. Il fattore si basa sull’osservazione che, sul lungo periodo, le azioni a piccola capitalizzazione tendono a sovraperformare quelle a grande capitalizzazione.

Fattore Valore (HML - High Minus Low): HML sta per “High [rapporto libro/mercato] Minus Low”. Questo fattore cattura i rendimenti in eccesso delle azioni value (con alti rapporti libro/mercato) rispetto alle azioni growth (con bassi rapporti libro/mercato). Le azioni value sono quelle tipicamente sottovalutate dal mercato ma con solidi fondamentali.

...

Fin qui tutto bene: ChatGPT fornisce le informazioni, corrette, che si possono trovare in qualsiasi testo di finanza quantitativa (per esempio, Ruppert e Matteson, 2015) o perfino su Wikipedia. Ora facciamogli produrre i codici.

Prompt dell'utente

Concentriamoci solo sui tre fattori Fama-French originali. Invece di usarli per spiegare i rendimenti contemporanei delle azioni in un portafoglio, usiamoli per prevedere i rendimenti del giorno successivo

per queste azioni. Puoi mostrare l'impostazione matematica per questa previsione?

Risposta di ChatGPT

Certamente! Per utilizzare i tre fattori Fama-French originali per prevedere i rendimenti del giorno successivo delle azioni in un portafoglio, impostiamo un modello di regressione lineare predittivo. In questo modello, i rendimenti futuri delle azioni sono la variabile dipendente e i valori correnti dei fattori Fama-French sono le variabili indipendenti. L'impostazione matematica per questo modello predittivo è la seguente.

Denotiamo: [Nota dell'Autore: abbiamo apportato alcune modifiche al codice LaTeX per renderlo più leggibile.]

$R_{i,t+1}$: il rendimento dell'azione i il giorno successivo (giorno $(t+1)$).

$R_{m,t} - R_{F,t}$: il rendimento in eccesso del mercato rispetto al tasso privo di rischio nel giorno corrente (t).

SMB_t : il fattore small minus big nel giorno corrente.

HML_t : il fattore high minus low nel giorno corrente.

$(\beta_{i0}, \beta_{i1}, \beta_{i2}, \beta_{i3})$: coefficienti da stimare per l'azione i .

$\epsilon_{i,t+1}$: il termine di errore per l'azione i nel mese [Nota dell'Autore: dovrebbe essere giorno, non mese] $t+1$.

Il modello di regressione lineare per ogni azione i può essere rappresentato come:

$$R_{i,t+1} = \beta_{i0} + \beta_{i1}(R_{m,t} - R_{F,t}) + \beta_{i2} SMB_t + \beta_{i3} HML_t + \epsilon_{i,t+1}$$

In questo modello:

β_{i0} è l'intercettazione per l'azione i .

β_{i1} , β_{i2} , β_{i3} sono i coefficienti che misurano le sensibilità dei rendimenti dell'azione i rispetto al rendimento in eccesso del mercato, rispettivamente, SMB e HML .

$\epsilon_{i,t+1}$ cattura gli effetti residui non spiegati dal modello.

Questa impostazione consente la previsione dei rendimenti del giorno successivo per ogni azione del portafoglio basandosi sui valori correnti dei fattori Fama-French. È importante notare che, sebbene questo modello utilizzi relazioni storiche per fare previsioni, l'accuratezza di queste previsioni può variare a causa della volatilità del mercato, dei cambiamenti economici e di altri fattori non catturati dal modello.

Prompt dell'utente

Corretto. Puoi trasformare questo modello matematico in codice Python?

Al che, ChatGPT risponde con del codice Python che rappresenta la regressione lineare dei rendimenti di un singolo titolo rispetto ai fattori Fama-French. Ovviamente, il codice non è completo. Inoltre, come prerequisito, avremo bisogno dei dati. Supponiamo che i prezzi siano contenuti nel file `CRSP_mid.csv`, che contiene dati formattati come segue:

```
tday      syms  mid
20070103  ABT   8.867780
20070103  HON   37.689365

20070103  AA    25.932974
20070103  BEAM  45.437196
20070103  AEP   31.218061
...
20131231  GGP   20.055000
20131231  MHK   148.904995
20131231  ADS   262.945005
20131231  FB    54.655000
20131231  TSCO  NaN
[1309166 righe × 3 colonne]
```

Non possiamo condividere questi dati a causa di restrizioni relative alla licenza, ma potete costruire un file simile partendo da qualsiasi dato giornaliero di azioni, privo di *bias di sopravvivenza*, come quelli di Sharadar (vedi un elenco di altri fornitori di dati in Chan [2017]).

Esercizio #1

Chiedete a ChatGPT di aiutarvi a convertire i dati che avete acquistato nel formato “lungo” mostrato nel paragrafo “Traduzione di listati Matlab in codice Python” del Capitolo 1.

Abbiamo chiesto a ChatGPT il codice per calcolare il rendimento futuro a un giorno, e ha risposto con qualcosa simile al codice che trovate nel paragrafo “Calcolo del rendimento del giorno successivo”, alla fine di questo capitolo. Il codice originale non era perfetto, ma se

conoscete un po' di programmazione Python, non vi sarà difficile sistemarlo.

Successivamente, dobbiamo caricare il file dei dati Fama-French, disponibile su

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html come F-F_Research_Data_Factors_daily.csv. Nuovamente, abbiamo

sollecitato ChatGPT a produrre il codice, e la risposta è passabile. Dopo alcune leggere modifiche, il codice è quello mostrato nel paragrafo “Caricamento dei fattori Fama-French” a fine capitolo. Abbiamo anche chiesto del codice per unire le etichette (rendimento a un giorno futuro) e le feature (fattori Fama-French) in un unico dataframe da utilizzare come dati di addestramento. Li trovate nell'ultimo paragrafo “Combinazione dei fattori Fama-French con i rendimenti del giorno successivo” del capitolo,

Ora siamo pronti per addestrare un modello di regressione o, più precisamente, per addestrare modelli di regressione per tutte le azioni. Abbiamo iniziato con quanto segue.

Prompt dell'utente

Il modello di regressione lineare che hai proposto sopra è corretto, ma è per una sola azione. Puoi produrre del codice generico per tutte le azioni presenti nei dati?

Risposta di ChatGPT

```
...
# Supponendo che 'Stock_ID' e '1_Day>Returns' siano colonne

# nei tuoi dati dei rendimenti del mese successivo
# Addestra un modello per ogni azione
for stock_id in combined_data['Stock_ID'].unique():
    stock_data = combined_data[combined_data['Stock_ID'] == stock_id]
    X = stock_data[['Mkt-RF', 'SMB', 'HML']]    # Sostituisci con i nomi
effettivi                                     # delle colonne Fama-
French
    y = stock_data['1_Day>Returns']
```

```
model = LinearRegression().fit(X, y)
# Salva o utilizza il modello per ogni azione
```

Il codice sembra corretto. Ma sfortunatamente, non può gestire i valori NaN presenti nei dati, e in genere i dati finanziari (o in effetti qualsiasi tipo di dati) sono pieni di valori mancanti. Quindi, riproviamo.

Prompt dell'utente

Corretto, ma che cosa succede se X e y contengono NaN?

Risposta di ChatGPT

Poiché la maggior parte degli algoritmi di machine learning, inclusa la regressione lineare in scikit-learn, non può gestire direttamente i valori NaN. Per affrontare questo problema, hai alcune opzioni: ...

L'affermazione che la maggior parte degli algoritmi di machine learning non può gestire direttamente valori NaN è falsa. Certamente la maggior parte, se non tutti, gli algoritmi presenti nel *Statistics and Machine Learning Toolbox* di Matlab possono gestire valori NaN.

Vogliamo che ChatGPT noti che questa è una “bugia”.

Prompt dell'utente

LightGBM può gestire i NaN nell'input?

Risposta di ChatGPT

Sì, LightGBM, un framework di boosting del gradiente che utilizza algoritmi di apprendimento basati su alberi, può gestire valori NaN (Not a Number) nei suoi dati di input. ...

Il che contraddice la sua precedente affermazione! No, la coerenza non è un pregio della GenAI.

Prompt dell'utente

Esiste qualche pacchetto Python per la regressione lineare che possa gestire i valori NaN presenti nei dati di input?

Risposta di ChatGPT

Statsmodels, un'altra libreria Python di modellazione statistica, può gestire i dati mancanti utilizzando il parametro `missing='drop'` nelle funzioni di fitting del modello, il quale chiede al modello di eliminare automaticamente le osservazioni che hanno NaN.

```
import statsmodels.api as sm
model = sm.OLS(y, X, missing='drop').fit()
```

Nota che questo approccio elimina qualsiasi riga con NaN, un po' come avviene con `dropna()` in pandas, ma questo opera internamente al modello.

Quindi, abbiamo provato a sostituire la riga

```
model = LinearRegression().fit(X, y)
```

con la riga `sm.OLS` (dopo aver eseguito `pip install statsmodels`). Beh, non ha funzionato nemmeno così, perché `sm.OLS` interpreta l'opzione `missing='drop'` come dei pesi. ChatGPT non ha idea del perché e ha suggerito di rimuovere tutte le righe con NaN prima di eseguire

`LinearRegression` in `scikit-learn`.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
# Supponendo che 'combined_data' sia il tuo DataFrame
some_minimum_threshold = 100 # Definisci una soglia in base alla tua
analisi
for stock_id in combined_data['Stock_ID'].unique():
    stock_data = combined_data[combined_data['Stock_ID'] == stock_id]
    # Rimuovi le righe in cui una qualsiasi delle colonne ha valori NaN
    stock_data = stock_data.dropna(subset=['1_Day>Returns', 'Mkt-RF',
'SMB', 'HML'])
    X = stock_data[['Mkt-RF', 'SMB', 'HML']]
    y = stock_data['1_Day>Returns']
    # Assicurati che X e y abbiano abbastanza punti di dati
    if len(X) > some_minimum_threshold: # Definisci una soglia in base
alla tua analisi
        model = LinearRegression().fit(X, y)
        print({
            'Stock_ID': stock_id,
            'Intercept': model.intercept_,
            'Beta_Market': model.coef_[0],
            'Beta_SMB': model.coef_[1],
            'Beta_HML': model.coef_[2]
        })
```

Sembra funzionare. Ma non vogliamo adattare un modello di regressione utilizzando tutti i nostri dati: dobbiamo lasciare alcuni dati per poter verificare la nostra strategia di trading. Quindi usiamo la prima metà per l'addestramento.

Prompt dell'utente

Il programma che hai creato sopra utilizzando la regressione lineare per prevedere il rendimento a un giorno è corretto, ma puoi modificarlo in modo che utilizzi solo la prima metà dei dati per il fitting?

Risposta di ChatGPT

```
...  
  
# Dividi i dati in due metà  
halfway_point = len(combined_data) // 2  
  
training_data = combined_data.iloc[:halfway_point]  
#...  
for stock_id in training_data['Stock_ID'].unique():  
    # Filtra i dati di addestramento per l'azione corrente  
    stock_data = training_data[training_data['Stock_ID'] == stock_id]  
#...
```

Le modifiche sembrano corrette e sono abbastanza facili da incorporare nel programma originale (vedi il frammento di codice finale nella cella `%%Train model` in `ff_1day.py`, disponibile al download dal nostro GitHub).

Dovremo fare previsioni utilizzando il modello precedentemente addestrato (fitted). Chiediamo a ChatGPT come fare.

Prompt dell'utente

Con il modello di regressione addestrato, qui sopra, come si fanno le previsioni utilizzando le "feature" di input?

ChatGPT ha risposto con del codice molto macchinoso che moltiplica esplicitamente i coefficienti di regressione come "Beta_Market" con le feature di input contenute in `X`. Quindi, gli abbiamo chiesto di renderlo più elegante.

Prompt dell'utente

Invece di fare queste previsioni utilizzando esplicitamente i nomi dei coefficienti del modello, puoi richiamare una funzione predict generica per un oggetto "modello di regressione"?

Risposta di ChatGPT

Certamente! Per utilizzare una funzione predict generica comune agli oggetti modello di regressione, tipicamente dovresti salvare ogni oggetto modello fitted per ogni azione durante il processo di addestramento, e poi fare previsioni utilizzando il metodo predict di questi oggetti modello. Questo approccio è più flessibile e si allinea con le pratiche standard di sviluppo dei modelli.

Ecco come puoi modificare il codice per includere il salvataggio degli oggetti modello fitted e poi utilizzare i loro metodi predict:

Passo 1: salva ogni modello fitted

Durante l'addestramento dei modelli, salva ogni oggetto modello fitted in un dizionario dove le chiavi sono gli ID delle azioni:

```
from sklearn.linear_model import LinearRegression
# Dizionario per memorizzare il modello per ogni azione
fitted_models = {}
for stock_id in training_data['Stock_ID'].unique():
    # ... [Passi di preparazione dei dati]
    # Adatta il modello
    model = LinearRegression().fit(X, y)
    # Memorizza il modello fitted
    fitted_models[stock_id] = model
```

Passo 2: prevedi utilizzando i modelli salvati

Utilizza il metodo predict di ogni modello salvato per fare previsioni su nuovi dati:

```
def predict_returns(fitted_models, features):
    """
    Fai previsioni per ogni azione utilizzando i modelli fitted.
    :param fitted_models: Dizionario di oggetti modello fitted per ogni azione
    :param features: DataFrame con nuove feature di input per la previsione
    :return: un DataFrame con i rendimenti previsti per ogni azione
    """
    predictions = []
    for stock_id, model in fitted_models.items():
        # Seleziona le feature per l'azione corrente
        # stock_features = features.loc[features['Stock_ID'] == stock_id,
        #
```

```

['Market_Excess_Return', 'SMB', 'HML']]
    # Modifica questo in
    stock_features = features.loc[features['Stock_ID'] == stock_id,
['Mkt-RF',
'SMB', 'HML']]
    ]
    # Controlla se ci sono feature su cui prevedere
    if not stock_features.empty:
        # Fai la previsione
        predicted_return = model.predict(stock_features)
        # Memorizza le previsioni
        for i, prediction in enumerate(predicted_return):
            predictions.append({'Stock_ID': stock_id,
                              'Predicted_Return': prediction,
                              'Date': stock_features.index[i]
                              })
    return pd.DataFrame(predictions)
# Esempio di utilizzo
# features = pd.DataFrame(...) # Sostituisci con il tuo DataFrame delle
nuove feature di input
predicted_returns = predict_returns(fitted_models, training_data)
print(predicted_returns)

```

In questo codice aggiornato.

- La funzione `predict_returns` itera su ogni modello `fitted` in `fitted_models`.
- Utilizza il metodo `predict` sul sottoinsieme di feature corrispondente a ogni azione.
- La funzione compila le previsioni in un `DataFrame`.

Assicurati che il `DataFrame` `features` sia strutturato correttamente con i fattori Fama-French e gli ID delle azioni. Questo approccio offre maggiore flessibilità ed è più allineato alle tipiche pratiche di sviluppo dei modelli.

Certo, sapremmo porre questa domanda solo se sapessimo che i modelli `fitted` di `scikit-learn` o `statsmodels` hanno una funzione `predict`, quindi chiaramente non andrà in questo modo per chi fosse meno esperto di Python e delle sue librerie di data science. Inoltre, sebbene questo codice sembri utile, perché ci aiuta a salvare i modelli `fitted` da qualche parte, non è del tutto corretto.

Un problema fondamentale con la funzione `predict_returns` è che si aspetta che l'input `stock_features` abbia un indice, `stock_id`. Questo potrebbe essere appropriato per feature “cross-sectional” (vedi

Nautiyal e Chan [2021]), ma certamente non per feature “time-series”. Quando abbiamo detto a ChatGPT che queste sono feature time-series che non usano `stock_id` come indice, ha pensato che intendiamo prevedere i rendimenti del portafoglio nel suo insieme, non i rendimenti delle singole azioni presenti in quel portafoglio. Quindi, sono necessari ulteriori prompt per chiarire questo dettaglio.

Prompt dell'utente

Forse è comune utilizzare fattori time-series per prevedere i rendimenti del portafoglio, ma qui abbiamo addestrato i nostri modelli per prevedere i rendimenti per ogni azione. Modifica la funzione di conseguenza.

Risposta di ChatGPT

...

```
import pandas as pd
import numpy as np
def predict_returns(fitted_models, features_today):
    """
    Prevedi i rendimenti per ogni azione per un singolo giorno
    utilizzando i rispettivi modelli fitted.
    :param fitted_models: Dizionario di oggetti modello fitted per ogni
    azione
    :param features_today: DataFrame o Series con le feature di input
    per un singolo giorno
    :return: una Series con i rendimenti previsti per ogni azione
    """
    predictions = {}
    # Assicurati che features_today sia nel formato corretto (array 2D)
    if isinstance(features_today, pd.Series):
        features_today = features_today.values.reshape(1, -1)
    elif isinstance(features_today, pd.DataFrame):
        features_today = features_today.values
    # Itera su ogni modello e prevedi il rendimento per ogni azione
    for stock_id, model in fitted_models.items():
        # Prevedi il rendimento per l'azione corrente utilizzando le
        feature comuni
        predicted_return = model.predict(features_today)
        # Se il modello restituisce una lista/array,
        # prendi il primo elemento come previsione
        if isinstance(predicted_return, (list, np.ndarray)):
            predictions[stock_id] = predicted_return[0]
        else:
            predictions[stock_id] = predicted_return
    return pd.Series(predictions)
```

In definitiva, nessuno dei programmi Python prodotti da ChatGPT ha funzionato a dovere. Ernie ha dovuto utilizzare le sue limitate competenze in Python per correggere e mettere insieme il codice per poter produrre la funzione finale.

```
# Previsione dei rendimenti
import pandas as pd
import numpy as np
def predict_returns(fitted_models, features_today):
    """
    Prevedi i rendimenti per ogni azione per un singolo giorno
    utilizzando i rispettivi modelli fitted.
    :param fitted_models: Dizionario di oggetti modello fitted per ogni
    azione
    :param features_today: DataFrame o Series con le feature di input
    per un singolo giorno
    :return: una Series con i rendimenti previsti per ogni azione
    """
    predictions = {}
    # Assicurati che features_today sia nel formato corretto (array 2D)
    features_today = features_today[['Mkt-RF', 'SMB', 'HML']]
    features_today = features_today.values.reshape(1, -1)
    # Itera su ogni modello e prevedi il rendimento per ogni azione
    for stock_id, model in fitted_models.items():
        # Prevedi il rendimento per l'azione corrente utilizzando le
        feature comuni
        predicted_return = model.predict(features_today)
        # Se il modello restituisce una lista/array, prendi il primo
        elemento come previsione
        predictions[stock_id] = predicted_return[0]
    return pd.Series(predictions)
```

Eseguendo questo codice con

```
predicted_returns = predict_returns(fitted_models, features_today)
```

dove features_today è una pd.Series che ha il seguente aspetto

Mkt-RF	-0.04
SMB	0.08
HML	-0.11
RF	0.022

e fitted_models è un output del codice di addestramento qui sopra,

l'output sarà simile a questo:

0033A	0.044187
3511B	0.057673
3FMCC	-0.139659
3FNMA	-0.112233

```
5933B      0.045648
...
XTO        0.016649
YH00       -0.012752
YUM        0.015054
ZION       -0.039987
ZMH        -0.007693
Length: 553, dtype: float64
```

Ora dobbiamo incorporare queste previsioni in un programma di backtest per la strategia di trading. Ma vediamo prima se ChatGPT sa che cos'è una strategia di trading.

Prompt dell'utente

Immagina una strategia di trading che acquista le prime 50 azioni con i rendimenti previsti più alti e vende allo scoperto le ultime 50, su base giornaliera, e mantiene quelle posizioni per i successivi 30 giorni. Qual è il numero massimo di posizioni che possiamo detenere per una particolare azione in un dato momento?

Risposta di ChatGPT

Poiché stai operando giornalmente e detieni ogni posizione per 30 giorni, significa che in un dato giorno, puoi avere fino a 30 posizioni sovrapposte per la stessa azione, ciascuna avviata in un giorno diverso negli ultimi 30 giorni. Questo scenario presuppone che l'azione appaia costantemente ogni giorno tra le prime 50 o le ultime 50 per 30 giorni consecutivi.

Impressionante! ChatGPT finalmente mostra un'intelligenza concettuale che non era emersa finora. Tuttavia, quando gli abbiamo chiesto di convertire questa strategia di trading in codice di backtest (con un periodo di detenzione di un giorno, per semplicità e per corrispondere al codice Matlab esistente), ha prodotto versioni sbagliate per un verso o per l'altro. In una versione utilizza come feature tutte le colonne presenti nei dati di input, invece di selezionare solo i tre fattori Fama-French. In un'altra versione, ha dimenticato che i dati di input hanno più righe per la stessa data, che corrispondono a diversi `stock_Id`. Quindi, abbiamo dovuto chiedere quanto segue.

Prompt dell'utente

Questo non risolve il problema. Il problema è che il ciclo for scorre ogni riga dei dati, immaginando che ogni riga si riferisca a dati di una data differente, mentre non è vero. Attualmente calcolerà ripetutamente più volte portfolio_returns per la stessa data, mentre itera sulle righe dei dati.

Al che, purtroppo, ha risposto con codice ancora più errato.

In una terza versione, ChatGPT pensa di essere diventato più intelligente, e considera i dati un giorno avanti rispetto alla data corrente per cercare il valore 31_Day>Returns. Ma questo è inutile, perché abbiamo creato 1_Day>Returns proprio come il rendimento del giorno successivo. Questo è un altro caso in cui chi non conosca Python, non sarà in grado di cogliere l'errore.

Dopo molti giorni (!) di prompting e aggiustamenti manuali, siamo finalmente arrivati alla seguente versione funzionante.

```
# Backtest della strategia di trading long-short
import pandas as pd
import warnings
warnings.filterwarnings('ignore') # Senza soppressione degli avvisi, il
programma
                                     # si bloccherà in Jupyter Notebook a causa di
troppi output
# Funzione per eseguire la strategia di trading
def execute_strategy(data, fitted_models, holding_period=1):
    portfolio_returns = []
    # Ottieni le date uniche dai dati
    unique_dates = data['Date'].unique()
    for day in range(len(unique_dates) - holding_period):
        # Ottieni la data corrente
        current_date = unique_dates[day]
        # Filtra i dati per la data corrente
        data_today = data[data['Date'] == current_date]
        # Ottieni le feature per questo giorno (supponendo che le
feature
        # siano le stesse per tutte le azioni in un dato giorno)
        features_today = data_today[['Mkt-RF', 'SMB', 'HML',
'Mkt-RF']].iloc[0]
        # Prevedi i rendimenti utilizzando la funzione predict_returns
        predicted_returns = predict_returns(fitted_models,
features_today)
        # Implementa la tua strategia: seleziona le prime 50 e le ultime
50 azioni
        top_50 = predicted_returns.nlargest(50)
```

```

        bottom_50 = predicted_returns.nsmallest(50)
        # Calcola i rendimenti per le azioni selezionate in una data
corrente
        current_date = unique_dates[day]
        current_data = data[data['Date'] == current_date]
        #print(current_date)
        long_returns =
current_data[current_data['Stock_ID'].isin(top_50.index)]

['1_Day>Returns'].mean()
        short_returns = -
current_data[current_data['Stock_ID'].isin(bottom_50.index)]

['1_Day>Returns'].mean()
        # Rendimento giornaliero del portafoglio
        daily_portfolio_return = (long_returns + short_returns) / 2
        portfolio_returns.append(daily_portfolio_return)
    return portfolio_returns

```

Per eseguire questa funzione, ChatGPT ha anche suggerito (correttamente):

```
portfolio_returns = execute_strategy(training_data, fitted_models)
```

Questa produce una lista, non una `pd.Series`, il che non è ideale, perché non include i timestamp. Non vogliamo fare i pignoli, quindi gli abbiamo detto quanto segue.

Prompt dell'utente

Corretto! Per favore calcola l'indice di Sharpe.

Risposta di ChatGPT

```

import numpy as np
# Calcola l'indice di Sharpe
def calculate_sharpe_ratio(portfolio_returns, risk_free_rate=0.02):
    excess_returns = np.array(portfolio_returns) - risk_free_rate / 252
    sharpe_ratio = np.mean(excess_returns) / np.std(excess_returns) *
np.sqrt(252)
    return sharpe_ratio

```

Questo solleva un punto importante discusso da Ernie in *Quantitative Trading* (Chan, 2021). Vediamo se ChatGPT coglie questa sfumatura.

Prompt dell'utente

Nella funzione `calculate_sharpe_ratio` definita sopra, il tasso privo di rischio è impostato per default su 0.02. Per un portafoglio dollar-neutral long-short, dovremmo dedurre il tasso privo di rischio dai rendimenti nel calcolo dell'indice di Sharpe? In altre parole, i rendimenti in eccesso sono i rendimenti lordi meno il tasso privo di rischio?

Risposta di ChatGPT

Nel contesto di un portafoglio long-short dollar-neutral, la considerazione del tasso privo di rischio nel calcolo dell'indice di Sharpe può essere un po' complessa. Ecco come considerarlo.

Rendimenti in Eccesso: tipicamente, l'indice di Sharpe viene calcolato utilizzando i rendimenti in eccesso, che sono i rendimenti del portafoglio meno il tasso privo di rischio. Questo serve per valutare quanto rendimento aggiuntivo l'investimento sta fornendo rispetto a un'alternativa priva di rischio (come i Treasury bill).

Portafoglio Dollar-Neutral: in un portafoglio dollar-neutral, sei lungo e corto in uguale misura in termini di dollari. Teoricamente, tale portafoglio potrebbe essere visto come autofinanziante, implicando che la sua performance potrebbe essere valutata senza considerare il tasso privo di rischio. La logica è che qualsiasi interesse che guadagneresti sul lato lungo (dall'investire denaro) potrebbe essere compensato dagli interessi che pagheresti sul lato corto (dal prendere in prestito denaro).

Considerazioni pratiche: praticamente, la scelta se includere o meno il tasso privo di rischio nel calcolo dell'indice di Sharpe per un portafoglio dollar-neutral dipende da diversi fattori:

Strategia e leva finanziaria: se la tua strategia implica costi di leva finanziaria o di finanziamento, questi dovrebbero essere considerati nel calcolo del rendimento.

Benchmarking: considera qual è la pratica standard nel tuo gruppo di benchmark o tra i tuoi colleghi.

Aspettative degli investitori: considera che cosa si aspetterebbero gli investitori del tuo fondo. Potrebbero confrontare la performance del tuo fondo con altri investimenti, che tipicamente sono valutati utilizzando i rendimenti in eccesso rispetto al tasso privo di rischio.

Tasso privo di rischio nell'indice di Sharpe.

Includi il tasso privo di rischio: se vuoi mantenere la coerenza con le pratiche di mercato più ampie e rendere il tuo indice di Sharpe confrontabile con altri, potresti includerlo.

Escludi il tasso privo di rischio: se ritieni che le caratteristiche del tuo portafoglio (come l'essere dollar-neutral) rendano meno rilevante il tasso privo di rischio, potresti escluderlo.

In sintesi, l'uso di rendimenti lordi o rendimenti in eccesso (rendimenti meno il tasso privo di rischio) nel calcolo dell'indice di Sharpe per un portafoglio long-short dollar-neutral dipende dalle specifiche caratteristiche del tuo portafoglio e dai tuoi obiettivi di misurazione della performance. Entrambi gli approcci hanno una loro logica, ma è importante essere coerenti e chiari sul metodo scelto, specialmente quando si comunica con stakeholder o investitori.

Impressionante! Ha colto la maggior parte dei dettagli. Per un portafoglio long-short come il nostro, tipicamente non sottraiamo il tasso privo di rischio (r_f) dai rendimenti lordi nei calcoli dell'indice di Sharpe, per la ragione di autofinanziamento descritta da ChatGPT. In altre parole, gli investitori con questo portafoglio possono ancora guadagnare interessi sul margine depositato nei loro conti, e fintantoché non includete tali interessi di margine nel vostro P&L, non avete bisogno di sottrarre r_f dai rendimenti. Ma se gestite un hedge fund e questo portafoglio è la sua unica attività, i rendimenti del fondo (ignorando le commissioni di gestione e performance) avranno già incluso questi interessi di margine, e quindi nel report del suo indice di Sharpe, occorre sottrarre r_f . Nell'era post-COVID dei tassi di interesse elevati che hanno raggiunto oltre il 5%, questa è una considerazione importante. Poiché non stiamo includendo r_f come interesse attivo nel nostro portafoglio, possiamo anche impostare r_f a 0 nel nostro seguente calcolo dell'indice di Sharpe.

```
# Calcolare l'Indice di Sharpe
def calculate_sharpe_ratio(portfolio_returns, risk_free_rate=0.0):
    excess_returns = np.array(portfolio_returns) - risk_free_rate / 252
    sharpe_ratio = np.mean(excess_returns) / np.std(excess_returns) *
    np.sqrt(252)
    return sharpe_ratio
# Esegui il backtest della strategia
sharpe_ratio = calculate_sharpe_ratio(portfolio_returns)
print("Indice di Sharpe:", sharpe_ratio)
```

che produce 6.5, molto più alto del 3.7 che abbiamo ottenuto da Matlab, sul set di addestramento. Quale dei due è corretto?

Il motivo per cui il programma Python ha sovraperformato quello Matlab è che nel programma Python abbiamo saltato tutti i titoli con dati insufficienti, mentre nel programma Matlab, abbiamo supposto una previsione di rendimenti nulli per quei titoli, il che non era corretto.

Vediamo le performance sul set di test.

Prompt dell'utente

*Diciamo che ho un dataframe combined_data. Voglio dividerlo in due metà: training_data e test_data usando halfway_point = len(combined_data) // 2
training_data = combined_data.iloc[:halfway_point]. Come dovrei
calcolare test_data?*

Risposta di ChatGPT

```
test_data = combined_data.iloc[halfway_point:]
```

che è corretto. Il backtest sui dati di test ha dato come risultato un indice di Sharpe pari a 0.55, che è molto meglio dell'indice di Sharpe negativo del programma Matlab. Il processo completo può essere scaricato da `ff_1day.py`, sul nostro GitHub. Ora vediamo la curva dei rendimenti cumulati ("Equity" o "Growth of \$1") mostrata nella Figura 2.1.

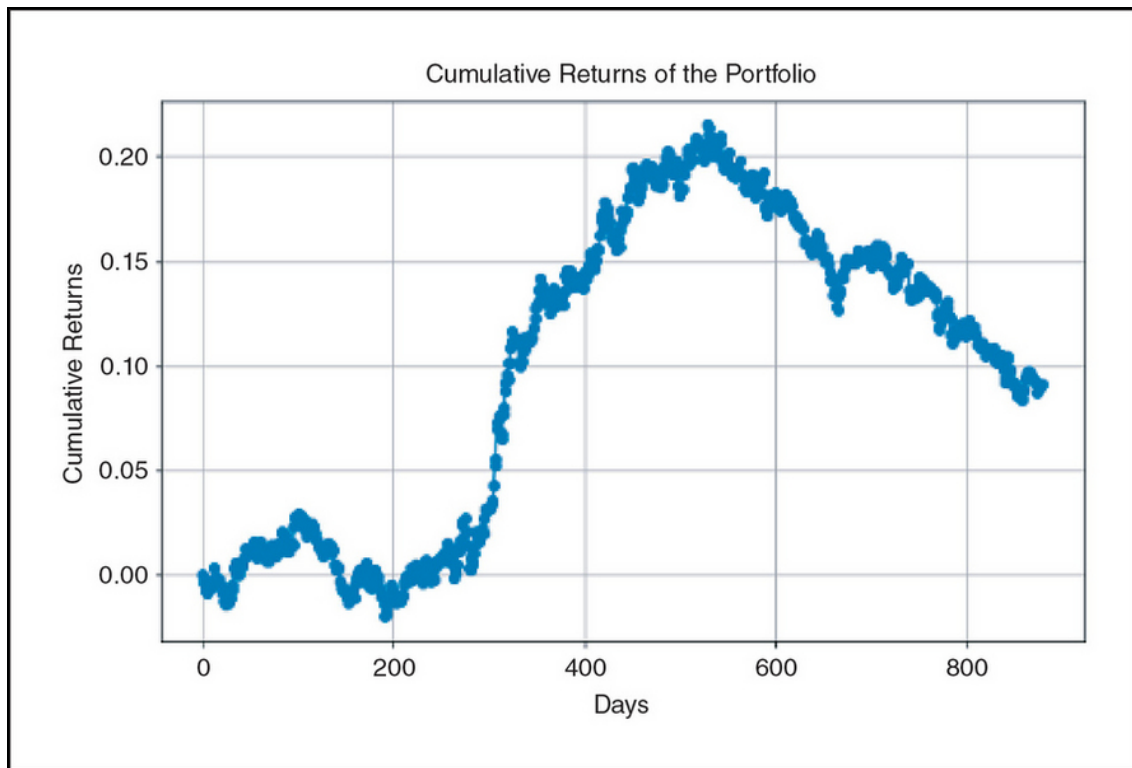


Figura 2.1 Grafico dei rendimenti cumulati della strategia a tre fattori Fama-French.

Prompt dell'utente

Supponi che i rendimenti giornalieri di un portafoglio siano contenuti in una lista `portfolio_returns`. Visualizza la curva dei rendimenti cumulati.

Risposta di ChatGPT

```
import matplotlib.pyplot as plt
import numpy as np
# Lista portfolio_returns di esempio (sostituisci con i tuoi rendimenti effettivi)
portfolio_returns = [0.01, -0.005, 0.007, -0.003, 0.002, 0.01, -0.004]
# Calcola i rendimenti cumulati
cumulative_returns = np.cumprod(1 + np.array(portfolio_returns)) - 1
# Traccia la curva dei rendimenti cumulati
plt.figure(figsize=(10, 6))
plt.plot(cumulative_returns, marker='o', linestyle='-')
plt.title('Rendimenti Cumulati del Portafoglio')
plt.xlabel('Giorni')
plt.ylabel('Rendimenti Cumulati')
```

```
plt.grid(True)
plt.show()
```

Ricordate che tutto questo riguarda i set di test. Se vi state chiedendo, il drawdown è iniziato intorno al settembre 2012, ed è stato inesorabile fino alla fine dei dati, dicembre 2023. Questo non significa che i fattori Fama-French non possano spiegare i rendimenti. Significa solo che i caricamenti dei fattori non sono più persistenti da un giorno all'altro. Come nota a margine, non abbiamo cercato di abbellire le figure generate da ChatGPT, poiché l'obiettivo, qui, era solo quello di mostrare gli output grezzi che ha prodotto. Certo, avremmo potuto chiedergli di creare figure dall'aspetto migliore, ma non era questo il senso dell'esercizio. A proposito, se chiedete a ChatGPT di migliorare lo stile dei grafici in Python usando Matplotlib o Seaborn, in realtà scoprirete che è molto bravo a farlo.

Riassumere un articolo su una strategia di trading e creare codici di backtest

Il secondo libro di Ernie, *Algorithmic Trading* (Chan, 2013, p.143), descrive una strategia di trading di momentum intergiornaliero che coinvolge i futures sul VIX. La strategia si basa su un articolo di Simon e Campasano (2014). Vediamo se ChatGPT può leggere e capire quell'articolo e produrre codici di backtest per esso.

Prompt dell'utente

Immagina di essere un esperto di finanza quantitativa. Per favore riassumi le strategie di trading descritte in questo articolo.

[Nota dell'Autore: ricordate che su ChatGPT-4o possiamo caricare allegati.]

Risposta di ChatGPT

...

L'articolo esplora spread trade di valore relativo, progettati per trarre profitto da deviazioni nell'EH

[Nota dell'Autore: Expectations Hypothesis]

...

I risultati dell'articolo suggeriscono che il mercato dei futures sul VIX presenta opportunità per trade strategici basati sulle anomalie della sua struttura a termine.

...

Un riassunto troppo generico. Gli abbiamo chiesto di essere più specifico.

Prompt dell'utente

Più specificamente, genera pseudocodici per gli spread trade di valore relativo, o per qualsiasi altro trade basato su anomalie della struttura a termine.

Risposta di ChatGPT

L'articolo descrive una strategia di trading beta-neutral (BN) che implica il mantenere una posizione lunga nei futures VIX a scadenza costante con una scadenza lunga (L) e una simultanea posizione corta nei futures VIX a scadenza costante con una scadenza corta (S). L'obiettivo di questa strategia è quello di assicurare che la posizione netta sia VIX-neutral. Ecco uno pseudocodice semplificato per la strategia di trading beta-neutral:

```
# Definisci la strategia di trading beta-neutral (BN)
Define BN_Trading_Strategy(long_maturity_tenor, short_maturity_tenor,
                           constant_maturity_betas):
    # Inizializza i pesi per le posizioni lunghe e corte
    weight_long = (1 / beta_long) / ((1 / beta_long) + (1 / beta_short))
    weight_short = 1 - weight_long
    # Calcola il payoff giornaliero
    daily_payoff = weight_long * return_long - weight_short *
return_short
    return daily_payoff
# Esempio di utilizzo
long_maturity_tenor = L # Specifica la scadenza lunga
short_maturity_tenor = S # Specifica la scadenza corta
constant_maturity_betas = Table_4_data # Recupera i beta a scadenza
costante dalla tabella 4
```

```
# Calcola il payoff giornaliero per la strategia beta-neutral
daily_payoff = BN_Trading_Strategy(long_maturity_tenor,
short_maturity_tenor,
                                constant_maturity_betas)
```

In questo pseudocodice:

long_maturity_tenor e short_maturity_tenor sono le scadenze per le posizioni rispettivamente lunghe e corte sui futures VIX. constant_maturity_betas rappresenta i beta per ogni scadenza, che possono essere recuperati da una tabella specificata (Tabella 4 nell'articolo). I pesi per le posizioni lunghe (weight_long) e corte (weight_short) vengono calcolati basandosi su questi beta. Il daily_payoff è calcolato come la differenza pesata tra i rendimenti delle posizioni lunghe e corte.

Questa, in realtà, *non* è la strategia di trading proposta dall'articolo. La strategia proposta non è una strategia di calendar spread. È una strategia per comprare o vendere allo scoperto futures sul VIX basandosi sul fatto che sia in backwardation o in contango e poi coprire la sua posizione con il future ES con lo stesso segno. Invece, vogliamo che ChatGPT si concentri sul seguente paragrafo dell'articolo:

Le simulazioni esaminano la redditività della vendita allo scoperto o dell'acquisto di contratti futures sul VIX quando la base dei futures sul VIX è, rispettivamente, in contango o in backwardation, e l'efficacia della copertura del rischio di mercato con posizioni sui mini-futures S&P. Le simulazioni poi esaminano l'impatto di modificare le regole di ingresso e poi introdurre regole di uscita basate su strategie di gestione del rischio. Tutte le simulazioni di trading esaminano la redditività della vendita allo scoperto o dell'acquisto del contratto futures sul VIX più frontale che ha almeno dieci giorni lavorativi fino alla regolamentazione quando l'entità del daily roll è maggiore in grandezza di .10 punti VIX futures (\$100) al giorno. Il daily roll è definito come la differenza tra il prezzo del future VIX frontale e il VIX, diviso per il numero di giorni lavorativi fino alla regolamentazione del contratto future VIX, e misura i potenziali profitti supponendo che la base declini linearmente fino alla regolamentazione.

Vediamo come possiamo indurlo a farlo.

Prompt dell'utente

Nello pseudocodice precedente, hai perso un'importante regola di trading. Vedi soprattutto il paragrafo che inizia con "Le simulazioni esaminano la redditività della vendita allo scoperto o dell'acquisto di contratti futures sul VIX quando la base dei futures sul VIX è in contango o in backwardation, ..."

[Nota dell'Autore: abbiamo citato per intero il paragrafo.]

Risposta di ChatGPT

...

```
# Definisci la strategia di trading beta-neutral
# con considerazione contango/backwardation
Define
BN_Trading_Strategy_With_Contango_Backwardation(long_maturity_tenor,
short_maturity_tenor, constant_maturity_betas,
VIX_term_structure_state):
    # Determina la direzione del trade basandosi sullo stato
    # della struttura a termine dei futures VIX
    if VIX_term_structure_state == "contango":
        # In contango, compra scadenza lunga e vendi scadenza corta
        futures VIX
        trade_direction = "buy_long_sell_short"
    elif VIX_term_structure_state == "backwardation":
        # In backwardation, vendi scadenza lunga e compra scadenza corta
        futures VIX
        trade_direction = "sell_long_buy_short"
    else:
        return "No trade"
    # Calcola i pesi per le posizioni lunghe e corte
    weight_long = (1 / beta_long) / ((1 / beta_long) + (1 / beta_short))
    weight_short = 1 - weight_long
    # Calcola il payoff giornaliero basandosi sulla direzione del trade
    if trade_direction == "buy_long_sell_short":
        daily_payoff = weight_long * return_long - weight_short *
return_short
    elif trade_direction == "sell_long_buy_short":
        daily_payoff = weight_short * return_short - weight_long *
return_long
    return daily_payoff
# Esempio di utilizzo
VIX_term_structure_state = Determine_VIX_Term_Structure_State()
# Funzione per determinare se il VIX è in contango o backwardation
daily_payoff =
BN_Trading_Strategy_With_Contango_Backwardation(long_maturity_tenor,
short_maturity_tenor, constant_maturity_betas,
VIX_term_structure_state)
```

In questo pseudocodice riveduto, la funzione `Determine_VIX_Term_Structure_State()` si suppone che analizzi la struttura a termine del VIX e restituisca contango o backwardation. La strategia di trading, quindi, aggiusta le sue posizioni di conseguenza, comprando lungo e vendendo corto in uno stato di contango o il contrario in uno stato di backwardation.

Ancora non è la strategia descritta nel paragrafo, ma almeno ha menzionato il criterio della struttura a termine. ChatGPT sa come gli autori dell'articolo definiscono la backwardation o il contango dei futures sul VIX? Lo scopriremo.

Prompt dell'utente

Basandoti sull'articolo, fornisci la definizione della funzione `Determine_VIX_Term_Structure_State()`. Cioè, definisci backwardation o contango basandoti sulla definizione dell'articolo.

Risposta di ChatGPT

Basandosi sulle informazioni nell'articolo, la funzione `Determine_VIX_Term_Structure_State()` può essere definita come segue.

Contango: la curva dei futures sul VIX è in uno stato di contango quando ha una pendenza rivolta verso l'alto. Ciò significa che i prezzi dei futures sono più alti del prezzo spot dell'indice VIX, e aumentano man mano che aumenta il tempo alla scadenza. In un mercato in contango, l'estremità lunga della curva dei futures è meno volatile, e i range di prezzo (differenza tra letture alte e basse) e la volatilità dei prezzi diminuiscono con l'aumentare del tempo alla scadenza.

Backwardation: la curva dei futures sul VIX è in uno stato di backwardation quando ha una pendenza rivolta verso il basso. Ciò si verifica quando i prezzi dei futures sono più bassi del prezzo spot dell'indice VIX, e diminuiscono man mano che aumenta il tempo alla scadenza. L'estremità lunga della curva in backwardation sarebbe più reattiva ai cambiamenti nel prezzo spot.

Ecco come potrebbe essere definita in pseudocodice la funzione `Determine_VIX_Term_Structure_State()`:

```
# Definisci una funzione per determinare lo stato della struttura a
termine del VIX
Define Determine_VIX_Term_Structure_State(VIX_spot_price,
VIX_futures_prices):
    # Confronta il prezzo spot con i prezzi dei futures per diverse
```

```

scadenze
    is_upward_sloping = True
    for each maturity in VIX_futures_prices:
        if VIX_futures_prices[maturity] < VIX_spot_price:
            is_upward_sloping = False
            break
    # Determina lo stato della struttura a termine basandosi sul pendio
    if is_upward_sloping:
        return "contango"
    else:
        return "backwardation"
# Esempio di utilizzo
VIX_spot_price = Current_VIX_Spot_Price()
# Funzione per ottenere il prezzo spot corrente del VIX
VIX_futures_prices = Get_VIX_Futures_Prices()
# Funzione per ottenere i prezzi dei futures VIX per diverse scadenze
term_structure_state =
Determine_VIX_Term_Structure_State(VIX_spot_price,

VIX_futures_prices)

```

In questo pseudocodice:

il VIX_spot_price è il prezzo spot corrente dell'indice VIX. VIX_futures_prices è un dizionario (o struttura dati simile) che contiene i prezzi dei futures per varie scadenze. La funzione confronta il prezzo spot con i prezzi dei futures per ogni scadenza per controllare se la curva ha una pendenza rivolta verso l'alto o verso il basso, determinando così se la struttura a termine è in contango o in backwardation.

Ha dato la definizione corretta di backwardation e contango. Ma ci mancano ancora le regole di trading per incorporarle.

Prompt dell'utente

Questa è la definizione corretta di backwardation e contango. Ma nota il paragrafo che inizia con "Le simulazioni esaminano....", che contiene una versione della strategia. Per favore riassumi quella versione e fornisci lo pseudocodice.

Risposta di ChatGPT

Grazie per aver fornito il paragrafo dettagliato. Basandoti su questa informazione, la strategia di trading rivista implica quanto segue.

Determinare lo stato della base dei futures VIX: la strategia implica vendere allo scoperto contratti futures sul VIX quando la base dei futures sul VIX è in contango e comprarli quando è in backwardation.

Regole di ingresso (Entry Rules): i trade vengono effettuati quando il contratto future sul VIX più frontale, che ha almeno dieci giorni

lavorativi fino alla regolamentazione, mostra un daily roll maggiore in grandezza di 0.10 punti VIX futures (\$100) al giorno.

Calcolo del Daily Roll: il daily roll è calcolato come la differenza tra il prezzo del future VIX frontale e il VIX, diviso per il numero di giorni lavorativi fino alla regolamentazione del contratto future sul VIX. Questo misura i potenziali profitti supponendo che la base declini linearmente fino alla regolamentazione.

Regole di uscita (Exit Rules): basandosi su strategie di gestione del rischio, la strategia può anche coinvolgere specifiche regole di uscita, sebbene queste non siano dettagliate nel paragrafo che hai fornito.

Soglie alternative: la nota a piè di pagina 13 indica che le simulazioni sono state eseguite anche con soglie di daily roll di +/- 0.15, risultando in meno trade più redditizi.

Ecco una rappresentazione in pseudocodice di questa strategia.

```
# Definisci la strategia di trading basata sulla base dei futures VIX e
sul daily roll
Define Trading_Strategy(VIX_spot_price, front_VIX_futures_price,
days_until_settlement,

daily_roll_threshold = 0.10):
    # Calcola il daily roll
    daily_roll = (front_VIX_futures_price - VIX_spot_price) /
days_until_settlement
    # Determina lo stato della base dei futures VIX
    if daily_roll > daily_roll_threshold:
        basis_state = "contango"
    elif daily_roll < -daily_roll_threshold:
        basis_state = "backwardation"
    else:
        return "No trade"
    # Determina l'azione di trading basandosi sullo stato della base
    if basis_state == "contango":
        trade_action = "short VIX futures"
    elif basis_state == "backwardation":
        trade_action = "buy VIX futures"
    return trade_action

# Esempio di utilizzo
VIX_spot_price = Current_VIX_Spot_Price()
# Funzione per ottenere il prezzo spot corrente del VIX
front_VIX_futures_price = Get_Front_VIX_Futures_Price()
# Funzione per ottenere il prezzo del contratto future VIX frontale
days_until_settlement =
Get_Days_Until_Settlement(front_VIX_futures_contract)
# Funzione per ottenere il numero di giorni lavorativi
# fino alla regolamentazione del contratto future VIX frontale
trade_action = Trading_Strategy(VIX_spot_price, front_VIX_futures_price,
days_until_settlement)
```

Ci stiamo avvicinando. Tuttavia, è strano che abbia completamente ignorato l'esplicito requisito (molto discusso nell'articolo) di coprire il future VIX con il future E-mini.

Prompt dell'utente

Credo che tu abbia dimenticato la specifica "copertura del rischio di mercato con posizioni sui mini-futures S&P". Come includerla nella tua specifica della strategia e nello pseudocodice?

Risposta di ChatGPT

Incorporare la copertura del rischio di mercato con mini-futures S&P nella strategia di trading richiede di aggiungere un componente alla strategia che aggiusti la posizione nei mini-futures S&P basandosi sull'esposizione al mercato della posizione in futures VIX. L'idea, qui, è quella di coprire il rischio di mercato associato al trade in futures VIX prendendo una posizione compensativa nei mini-futures S&P.

Ecco come questo potrebbe essere integrato nella strategia.

Valuta l'esposizione di mercato: dopo aver determinato l'azione di trading per i futures VIX (vendere allo scoperto o comprare), valuta l'esposizione al mercato creata da questa posizione. Per esempio, una posizione lunga in futures VIX potrebbe essere considerata come avente un'esposizione di mercato negativa (poiché spesso il VIX aumenta quando il mercato scende), e una posizione corta in futures VIX come esposizione di mercato positiva.

Copertura con mini-futures S&P: basandosi sull'esposizione di mercato valutata dalla posizione in futures VIX, prendi una posizione compensativa nei mini-futures S&P. Se la posizione in futures VIX ha un'esposizione di mercato negativa, una posizione lunga nei mini-futures S&P coprirebbe questo rischio, e viceversa.

Ecco lo pseudocodice riveduto con la copertura del rischio di mercato:

```
# Definisci la strategia di trading con copertura del rischio di mercato
# usando mini-futures S&P
Define Trading_Strategy_With_Hedging(VIX_spot_price,
front_VIX_futures_price,
    days_until_settlement, S_and_P_position_size,
daily_roll_threshold = 0.10):
    # Calcola il daily roll
    daily_roll = (front_VIX_futures_price - VIX_spot_price) /
days_until_settlement
    # Determina lo stato della base dei futures VIX
    if daily_roll > daily_roll_threshold:
        basis_state = "contango"
    elif daily_roll < -daily_roll_threshold:
        basis_state = "backwardation"
```

```

else:
    return "No trade"

# Determina l'azione di trading basandosi sullo stato della base
if basis_state == "contango":
    VIX_trade_action = "short VIX futures"
    S_and_P_trade_action = "long mini-S&P futures"
    # Copertura contro l'esposizione di mercato positiva
elif basis_state == "backwardation":
    VIX_trade_action = "buy VIX futures"
    S_and_P_trade_action = "short mini-S&P futures"
    # Copertura contro l'esposizione di mercato negativa
# Aggiusta la dimensione della posizione in mini-futures S&P
# per coprire il rischio di mercato
Adjust_Mini_SP_Futures_Position(S_and_P_position_size,
S_and_P_trade_action)
return VIX_trade_action, S_and_P_trade_action
# Esempio di utilizzo
VIX_spot_price = Current_VIX_Spot_Price()
front_VIX_futures_price = Get_Front_VIX_Futures_Price()
days_until_settlement =
Get_Days_Until_Settlement(front_VIX_futures_contract)
S_and_P_position_size = Determine_Position_Size()
# Funzione per determinare la dimensione della posizione in mini-futures
S&P
VIX_trade_action, S_and_P_trade_action =
Trading_Strategy_With_Hedging(VIX_spot_price,
front_VIX_futures_price, days_until_settlement,
S_and_P_position_size)

```

In questo pseudocodice, Adjust_Mini_SP_Futures_Position è una funzione che aggiusta la posizione nei mini-futures S&P basandosi sull'azione di trading calcolata e sulla dimensione della posizione. Questa funzione terrebbe conto delle specifiche della copertura, come il grado di leva o il dimensionamento preciso della posizione mini-S&P relativa alla posizione VIX, per bilanciare efficacemente il rischio di mercato.

Dopo aver, praticamente, “imboccato” le informazioni a ChatGPT (trattandolo come un tirocinante inesperto e mediocre), finalmente ha aggiunto la copertura. Ma aspettate: ogni volta che c'è una copertura, dobbiamo specificare un rapporto di copertura (hedge ratio). ChatGPT non conosce la finanza di base?

Prompt dell'utente

Questo è corretto. L'articolo specifica come dovremmo determinare il rapporto di copertura tra i futures VIX e ES?

Risposta di ChatGPT

Ho esaminato accuratamente l'articolo "VIX Futures Term Structure and the Expectations Hypothesis" per trovare informazioni specifiche riguardanti la determinazione del rapporto di copertura tra VIX e mini-S&P (ES) futures. Tuttavia, i paragrafi dell'articolo cui ho potuto accedere non fornivano dettagli espliciti o formule per calcolare questo rapporto di copertura.

Questo tirocinante mediocre sta diventando sempre più fastidioso. Ma come suoi mentori, non lasceremo che ci faccia perdere la pazienza.

Prompt dell'utente

Vedi il paragrafo che inizia con "Il rapporto di copertura - il numero di contratti di mini-futures S&P da comprare o vendere per posizione in futures VIX - è costruito usando le stime dei parametri out of sample dall'equazione 3". Sei in grado di riassumere il metodo di calcolo del rapporto di copertura e mostrare lo pseudocodice?

Risposta di ChatGPT

..., il rapporto di copertura (HR) è calcolato usando la seguente formula:

$$HR_t = \frac{\beta_1 \times 1000 + \beta_2 \times TTS_{t-1} \times 1000}{0.01 \times ES_{t-1} \times 50}$$

dove:

- HR_t è il rapporto di copertura al tempo t ;
- β_1 e β_2 sono parametri stimati dall'equazione 3 (non forniti nella tua citazione);
- TTS_{t-1} è il valore ritardato di qualche variabile (non specificata nella tua citazione) al tempo $(t - 1)$;
- ES_{t-1} è il valore ritardato del contratto di mini-futures S&P al tempo $(t - 1)$;
- il valore di un cambiamento di prezzo di un punto dei futures VIX è \$1,000;
- il guadagno o la perdita per un cambiamento di prezzo dell'1% dei mini-futures S&P è calcolato come 0.01 volte il valore ritardato del contratto di mini-futures S&P volte \$50, che è il valore di un punto dei mini-futures S&P.

Il rapporto di copertura medio è approssimativamente un contratto di mini-futures S&P per contratto future VIX, con un range da 1/2 a 2

contratti, influenzato principalmente dalle fluttuazioni nel numero di giorni lavorativi fino alla regolamentazione dei contratti futures VIX rilevanti e dal livello dei prezzi dei futures S&P.

Ecco lo pseudocodice per questo calcolo del rapporto di copertura.

```
# Definisci una funzione per calcolare il rapporto di copertura
Define Calculate_Hedge_Ratio(beta1, beta2, TTS_lagged, ES_lagged):
    # Calcola il valore di un cambiamento di prezzo di un punto dei
    futures VIX
    VIX_price_change_value = 1000
    # Calcola il guadagno o la perdita per un cambiamento di prezzo
    # dell'uno per cento dei mini-futures S&P
    mini_SP_price_change_gain_loss = 0.01 * ES_lagged * 50
    # Calcola il rapporto di copertura
    HR = (beta1 * VIX_price_change_value + beta2 * TTS_lagged *
    VIX_price_change_value) /

mini_SP_price_change_gain_loss
    return HR
# Esempio di utilizzo
beta1 = Parameter_Estimate1 # Stimato dall'equazione 3
beta2 = Parameter_Estimate2 # Stimato dall'equazione 3
TTS_lagged = Lagged_Value_TTS # Valore ritardato della variabile TTS
ES_lagged = Lagged_Value_ES # Valore ritardato del contratto di mini-
futures S&P
hedge_ratio = Calculate_Hedge_Ratio(beta1, beta2, TTS_lagged, ES_lagged)
```

In questo pseudocodice, *Parameter_Estimate1* e *Parameter_Estimate2* rappresentano le stime dei parametri β_1 e β_2 dall'equazione 3. *Lagged_Value_TTS* e *Lagged_Value_ES* sono, rispettivamente, i valori ritardati della variabile TTS e del contratto di mini-futures S&P. La funzione *Calculate_Hedge_Ratio* calcola il rapporto di copertura basandosi su questi input e sulla formula fornita.

Apparentemente, ChatGPT non riusciva a ricordare l'intero articolo, quindi non sa cosa siano β_1 e β_2 , né cosa sia TTS_{t-1} . Per non prolungare questa agonia, citiamo semplicemente quello che diceva l'articolo.

I rapporti di copertura sono costruiti da regressioni dei cambiamenti di prezzo dei futures VIX su una costante e su cambiamenti percentuali contemporanei del contratto di mini-futures S&P frontale sia da solo che moltiplicato per il numero di giorni dalla regolamentazione del contratto future VIX, come mostrato sotto.

$$\Delta VIX_t^F = \beta_0 + \beta_1 * SPRET_t + \beta_2 * SPRET_t * TTS_t + \mu_t$$

Questa è la misteriosa equazione 3 che ChatGPT non riesce a ricordare. Ma lasciamo da parte questo problema e passiamo a un problema più grande: e la regola di uscita? L'articolo specificava che:

Queste simulazioni presumono anche che i trade vengano chiusi alla fine di 5 giorni lavorativi...

Vediamo se ChatGPT l'ha notato!

Prompt dell'utente

Nella specifica della strategia di trading Define Trading_Strategy_With_Hedging(VIX_spot_price, front_VIX_futures_price, days_until_settlement, S_and_P_position_size, daily_roll_threshold = 0.10), sembra che tu abbia specificato solo quando dovremmo entrare in un trade. Qual è la regola di uscita?

Risposta di ChatGPT

La regola di uscita per la strategia di trading che coinvolge futures VIX e copertura con mini-futures S&P, come descritto nella tua citazione precedente dall'articolo, non è dettagliata esplicitamente nelle informazioni fornite...

Prompt dell'utente

Questo non è vero! Vedi il paragrafo che contiene la frase, "Queste simulazioni ipotizzano anche che i trade vengano chiusi alla fine di ..."

Risposta di ChatGPT

...

```
# Definire la strategia di trading con copertura del rischio di mercato
# e una regola di uscita a durata fissa
Define
Trading_Strategy_With_Hedging_And_Fixed_Duration_Exit(VIX_spot_price,
front_VIX_futures_price, days_until_settlement,
S_and_P_position_size,
daily_roll_threshold = 0.10, holding_period = 5):

    # Regole di ingresso come prima
    ...
    # Regola di uscita basata su un periodo di detenzione fisso di 5
    giorni lavorativi
    Define Fixed_Duration_Exit_Rule(current_holding_days,
holding_period):
        if current_holding_days >= holding_period:
```

```

        return "exit"
    else:
        return "hold"
    # Monitorare il periodo di detenzione e decidere se uscire
    current_holding_days = Count_Holding_Days()
    # Funzione per contare il numero di giorni lavorativi dall'apertura
    della posizione
    exit_decision = Fixed_Duration_Exit_Rule(current_holding_days,
holding_period)
    return exit_decision
# Esempio di utilizzo
...
exit_decision = Trading_Strategy_With_Hedging_And_Fixed_Duration_Exit(...)

```

In questo pseudocodice:

holding_period è impostato a 5 giorni lavorativi, come specificato nella tua frase. La funzione Fixed_Duration_Exit_Rule verifica se il periodo di detenzione corrente ha raggiunto o superato la durata specificata di 5 giorni lavorativi. Count_Holding_Days è una funzione ipotetica che conta il numero di giorni lavorativi da quando la posizione è stata avviata.

ChatGPT ha finalmente trovato la frase rilevante in questione! Ora convertiamo il risultato in codice Python.

Prompt dell'utente

Bene. Ora converti lo pseudocodice in un programma Python completo per il backtest dell'intera strategia di completa.

Risposta di ChatGPT

```

import pandas as pd
import numpy as np
from datetime import timedelta
# Funzione per simulare il recupero dei dati storici
# (Sostituire con una fonte dati reale)
def get_historical_data():
    # Dati fittizi per illustrazione
    data = pd.DataFrame({'date': pd.date_range(start='2020-01-01',
periods=100,
                                                    freq='B'),
                        'VIX_spot': np.random.uniform(20, 30, 100),
                        'VIX_futures': np.random.uniform(20, 30, 100),
                        'ES_futures': np.random.uniform(3000, 4000,
100)
    })
    return data

# Funzione per calcolare il rollio giornaliero
def calculate_daily_roll(VIX_futures, VIX_spot, days_until_settlement):

```

```

    return (VIX_futures - VIX_spot) / days_until_settlement
# Implementazione della strategia di trading
def trading_strategy(data, daily_roll_threshold=0.10, holding_period=5):
    positions = []
    entry_days_remaining = None
    for index, row in data.iterrows():
        if entry_days_remaining is None:
            # Logica di ingresso
            daily_roll = calculate_daily_roll(row['VIX_futures'],
row['VIX_spot'],
                                                    days_until_settlement=10)
            if abs(daily_roll) > daily_roll_threshold:
                position = 'short' if daily_roll > 0 else 'long'
                entry_days_remaining = holding_period
            else:
                # Detenzione della posizione
                entry_days_remaining -= 1
                if entry_days_remaining <= 0:
                    # Logica di uscita dopo il periodo di detenzione
                    positions.append(position)
                    position = None
                    entry_days_remaining = None
    return positions
# Funzione principale per eseguire il backtest
def backtest_strategy():
    data = get_historical_data()
    positions = trading_strategy(data)
    print("Posizioni di trading:", positions)
# Esegui il backtest
backtest_strategy()

```

Questo script include:

Una funzione fittizia get_historical_data per simulare dati storici. Dovresti sostituirla con i dati effettivi, tratti da una fonte affidabile. Le funzioni calculate_daily_roll e trading_strategy implementano la logica di ingresso e uscita basata sulle variazioni giornaliere e sul periodo di detenzione fisso. La funzione backtest_strategy esegue il backtest e mostra le posizioni di trading. Nota bene:

Questo script è molto semplificato e non include fonti di dati reali, costi di transazione, scostamento di esecuzione, funzionalità di gestione del rischio o metriche di performance.

Questo script non include il rapporto di copertura e la copertura con i futures mini-S&P. Dovresti integrarli in base alla specifica formula del rapporto di copertura e ai dati di mercato. Per un ambiente di backtesting realistico, considera di utilizzare un framework come backtrader o zipline. Questo script è un punto di partenza e dovresti espanderlo e affinarlo per ottenere un backtesting più completo e realistico.

Stranamente, ChatGPT non ha tradotto in codice Python tutto lo pseudocodice che aveva generato precedentemente nel corrispondente. Questo è piuttosto imperdonabile.

Prompt dell'utente

Perché non includi le posizioni di copertura con i futures E-mini, visto che hai già lo pseudocodice per Define Calculate_Hedge_Ratio? Aggiungilo nel codice Python.

ChatGPT ha risposto aggiungendo la definizione della funzione `calculate_hedge_ratio` e così via. Il codice finale è corretto, e lo trovate in `VX_ES.py` sul nostro GitHub.

Nel complesso, troviamo che ChatGPT sia davvero scadente nel leggere e riassumere contenuti quando sono coinvolti dettagli tecnici. Trascura molti dettagli di un tipico articolo di finanza di sole 40 pagine. Anche quando gli vengono presentate citazioni specifiche del documento, ha richiesto molto lavoro, competenza e prompting umani per generare lo pseudocodice di backtest per l'articolo. Questo è ben lontano dal paradiso (o dalla distopia) dell'intelligenza artificiale generale che ci era stato promesso. Immaginateci con un'espressione un po' delusa.

Ricerca di un algoritmo di ottimizzazione del portafoglio basato sul machine learning

Adesso basta col dare in pasto a ChatGPT le specifiche strategie di cui vogliamo fare il backtest. E se lo usassimo per trovare algoritmi? Abbiamo sviluppato un metodo di ottimizzazione del portafoglio basato sul machine learning chiamato CPO (Chan et al., 2023). Il metodo CPO utilizza un gran numero di feature di input che abbracciano diversi mercati, inclusi dati fondamentali,

macroeconomici, tecnici e perfino microstrutturali. Poi un algoritmo di machine learning viene addestrato su questi dati insieme a molte allocazioni di capitale ipotetiche storiche, per determinare quale allocazione di capitale darà il miglior indice di Sharpe per un qualsiasi periodo futuro.

Tuttavia, questo non è l'unico modo in cui il machine learning può essere applicato all'ottimizzazione del portafoglio. Un approccio più "convenzionale" consiste nell'applicare il DRL (*Deep Reinforcement Learning*), impiegando l'attuale allocazione del portafoglio come "stato" corrente, i rendimenti futuri come "ricompensa" e le decisioni di riallocazione come "azioni" che portano allo stato successivo. I tipici input per il DRL sono solo rendimenti o prezzi: non abbiamo visto esempi che utilizzino input di dati fondamentali e altri big data, specialmente quelli che utilizzano feature di serie temporali. Quindi, chiediamo tali riferimenti a Copilot. Abbiamo anche provato ChatGPT, ma non ci ha fornito risposte soddisfacenti.

Ancora una volta, Kıcıman et al. (2023) hanno detto che dovremmo "preriscaldare" e orientare gli LLM verso un dominio specifico per pensare di ottenere risultati migliori.

Prompt dell'utente

Immagina di essere un esperto di finanza quantitativa e deep learning per le domande seguenti.

Risposta di Copilot

[Nota dell'Autore: alcune chiacchiere].

Prompt dell'utente

Intendiamo applicare il Deep Reinforcement Learning al problema dell'ottimizzazione del portafoglio. Che tipi di input hanno utilizzato i ricercatori per questo problema?

Risposta di Copilot

L'applicazione del DRL (Deep Reinforcement Learning) all'ottimizzazione del portafoglio è un approccio all'avanguardia che combina la complessità dei mercati finanziari con tecniche avanzate di intelligenza artificiale. In questo contesto, i ricercatori hanno utilizzato vari tipi di input per informare i modelli DRL. Ecco alcuni tipi comuni di input.

1. *Dati basati sui prezzi:* questo è il tipo di input più comune e include prezzi storici, rendimenti e altre statistiche derivate come medie mobili, volatilità (misurata come deviazione standard dei rendimenti) e indicatori di momentum. Questi dati possono essere a diverse frequenze (giorni, ore, minuti e così via).
2. *Dati sul volume:* il volume di scambio e i suoi derivati possono fornire conoscenze sul sentiment e sulla liquidità del mercato, che sono fattori importanti nell'ottimizzazione del portafoglio.
3. *Dati fondamentali:* per le azioni, questo potrebbe includere dati finanziari aziendali come utili, ricavi, livelli di debito, rapporti P/E e così via. Per altre classi di attività, verrebbero utilizzati dati fondamentali rilevanti (per esempio, indicatori economici per i forex o le materie prime).
4. *Dati alternativi:* questo può spaziare dalle immagini satellitari per prevedere i rendimenti delle colture nei mercati delle materie prime all'analisi del sentiment dei social media per valutare il sentiment del mercato.
5. *Indicatori di mercato:* vari indici di mercato, spread intermercato e indicatori macroeconomici (come tassi di interesse, tassi di inflazione, tassi di crescita del PIL) possono essere importanti per comprendere il contesto più ampio del mercato.
6. *Indicatori tecnici:* spesso i trader usano indicatori come l'Indice di Forza Relativa (RSI), le Bande di Bollinger, il MACD (Moving Average Convergence Divergence) e altri, che possono essere input utili per un modello DRL.
7. *Metriche di rischio:* anche misure come il Value at Risk (VaR), il Conditional Value at Risk (CVaR) o l'indice di Sharpe possono essere input per guidare l'algoritmo DRL non solo verso la massimizzazione del rendimento ma anche verso l'ottimizzazione del rendimento corretto per il rischio.
8. *Dati di notizie ed eventi:* feed di notizie in tempo reale, report sugli utili e altri dati di evento possono avere un impatto significativo sulla dinamica del mercato e, quindi, sulla performance del portafoglio.
9. *Dati del libro ordini:* in scenari di trading ad alta frequenza, i dati del libro ordini (come spread bid-ask, profondità degli ordini e

così via) possono essere fondamentali.

10. Segnali macro: nei modelli più sofisticati possono essere presi in considerazione anche segnali da tendenze economiche più ampie o eventi geopolitici.

...

Sembra incoraggiante: menziona una grande varietà di input, esattamente quello che abbiamo in mente. Quindi, approfondiamo.

Prompt dell'utente

Puoi fornirmi riferimenti a ricerche che hanno utilizzato dati fondamentali come input per una soluzione DRL dell'ottimizzazione del portafoglio?

Risposta di Copilot

...

Uno studio, "Deep Reinforcement Learning for Portfolio Optimization Using Latent Feature State Space (LFSS) Module", esplora l'uso del reinforcement learning in un contesto finanziario...

...

Un altro studio rilevante è intitolato "A New Deep Reinforcement Learning Model for Dynamic Portfolio Optimization"...

...

In realtà conosciamo un articolo (Cong et al., 2021) che utilizza il DRL con feature fondamentali, sebbene sia più un algoritmo di selezione di titoli che di ottimizzazione del portafoglio, in generale. È comunque deludente che Copilot non lo abbia menzionato. Vediamo di cosa parlano questi altri articoli.

Prompt dell'utente

In "Deep Reinforcement Learning for Portfolio Optimization Using Latent Feature State Space (LFSS) Module", che hai citato, che tipo di dati di input sono stati utilizzati?

Risposta di Copilot

...

L'attenzione principale era sui dati dei prezzi e dei rendimenti, che tipicamente derivano da dati Open, High, Low, Close (OHLC).

...

Non è ciò che volevamo.

Prompt dell'utente

Nel nuovo modello di Deep Reinforcement Learning per l'ottimizzazione dinamica del portafoglio che hai citato, che tipo di dati di input sono stati utilizzati?

Risposta di Copilot

L'articolo "A New Deep Reinforcement Learning Model for Dynamic Portfolio Optimization" non ha fornito dettagli specifici nei paragrafi cui ho avuto accesso riguardo ai tipi esatti di dati di input utilizzati...

Beh, piuttosto deludente. A che cosa serve l'intelligenza artificiale se dobbiamo leggerci l'articolo originale per trovare quelle informazioni?

Prompt dell'utente

Quindi, nessuno dei due articoli che hai citato ha effettivamente utilizzato dati fondamentali come input, eppure hai affermato che i dati fondamentali sono un tipo di dati utilizzati per il DRL. Su quali prove basi questa affermazione?

Risposta di Copilot

Hai ragione nel farmi notare che i due articoli che ho citato in precedenza non menzionavano specificamente l'uso di dati fondamentali come input nei loro modelli di Deep Reinforcement Learning per l'ottimizzazione del portafoglio. Tuttavia, l'affermazione che i dati fondamentali vengono utilizzati in tali modelli si basa su tendenze e pratiche più ampie nel campo della finanza quantitativa e del deep learning.

...

In sintesi, sebbene i specifici articoli che ho citato non abbiano utilizzato dati fondamentali, questi rimangono un tipo di input importante per l'ottimizzazione del portafoglio basata su DRL. Apprezzo la tua diligenza e ti ringrazio per aver segnalato l'oversight!

Per ulteriori esplorazioni, puoi fare riferimento a articoli di ricerca che si concentrano esplicitamente sull'integrazione di dati fondamentali

nei modelli DRL per la gestione del portafoglio.

Prompt dell'utente

Sì, quindi quali articoli di ricerca dovrei esplorare che trattano esattamente quell'argomento?

Risposta di Copilot

Certamente! Ecco alcuni articoli di ricerca che esplorano specificamente l'integrazione di dati fondamentali nei modelli di Deep Reinforcement Learning (DRL) per l'ottimizzazione del portafoglio:

"Deep Reinforcement Learning for Portfolio Optimization Using Latent Feature State Space (LFSS) Module" Abstract: questo articolo presenta un framework DRL per l'ottimizzazione autonoma del portafoglio. Include un modulo Latent Feature State Space (LFSS) per il filtraggio e l'estrazione di caratteristiche dai dati finanziari. Lo studio vanta risultati robusti e vantaggi prestazionali rispetto a diversi benchmark.

"Deep Reinforcement Learning in Portfolio Management" Abstract: questo articolo implementa due algoritmi all'avanguardia di reinforcement learning continuo (Deep Deterministic Policy Gradient e Proximal Policy Optimization) nella gestione del portafoglio. Sebbene non si concentri esplicitamente sui dati fondamentali, fornisce approfondimenti sulle tecniche DRL per l'ottimizzazione del portafoglio.

"A New Deep Reinforcement Learning Model for Dynamic Portfolio Optimization" Abstract: questo modello integra metodi di elaborazione dati, deep learning e reinforcement learning per migliorare il processo decisionale degli investitori. Sebbene non utilizzi direttamente dati fondamentali, esplora l'ottimizzazione dinamica del portafoglio.

"Deep Reinforcement Learning for Optimizing Finance Portfolio Management" Abstract: questo studio esamina come il DRL può essere adottato per massimizzare la gestione del portafoglio nel settore finanziario. Sebbene copra vari aspetti, inclusi gli indicatori tecnici, fornisce approfondimenti sulle applicazioni del DRL.

...

È fantastico che Copilot abbia finalmente citato articoli rilevanti per la nostra richiesta, ma quando abbiamo chiesto chiarimenti sulle specifiche funzionalità utilizzate da questi articoli, non è stato in grado di rispondere. In effetti, potete verificare che questi articoli non utilizzano affatto dati fondamentali o feature di serie temporali. Ancora una volta, immaginateci piuttosto delusi.

Esplorare le strategie di arbitraggio sulla struttura a termine delle opzioni

Nel mondo ideale del famoso modello di pricing delle opzioni Black-Scholes che ogni studente di finanza quantitativa è costretto a imparare, la volatilità è costante e i prezzi possono essere descritti da un moto browniano gaussiano (cioè, un andamento casuale dei logaritmi dei prezzi). Nessuna di queste ipotesi è vera nel mondo reale. Questa è una buona cosa per i trader: se queste ipotesi fossero corrette, le volatilità implicite sarebbero costanti per qualsiasi scadenza, e così sarebbe anche il valore tempo annualizzato di queste opzioni. In altre parole, la struttura a termine dei prezzi delle opzioni sarebbe piatta, e l'arbitraggio sulla struttura a termine sarebbe impossibile. Per arbitraggio sulla struttura a termine, intendiamo la possibilità di guadagnare sulle opzioni acquistando e vendendo allo scoperto opzioni di scadenze differenti senza assumere rischi direzionali sul sottostante. La strategia di arbitraggio sulla struttura a termine più semplice è il trading di spread di calendario in cui tutte le opzioni coinvolte hanno lo stesso prezzo di esercizio.

Vediamo se ChatGPT può aiutarci a costruire una strategia di arbitraggio sulla struttura a termine. Notate che questo compito è più impegnativo di qualsiasi altro che abbiamo assegnato a ChatGPT finora. Qui non abbiamo specificato una strategia di trading, e stiamo chiedendo a ChatGPT di costruirne una basandosi solo su suggerimenti concettuali.

Prompt dell'utente

Immagina di essere un esperto di finanza quantitativa e, in particolare, di trading di opzioni.

Prompt dell'utente

Per favore, riassumi le diverse strategie di trading che coinvolgono l'arbitraggio sulla struttura a termine delle opzioni.

Risposta di ChatGPT

L'arbitraggio sulla struttura a termine nel trading di opzioni implica lo sfruttamento di inefficienze nella quotazione di opzioni con diverse date di scadenza. Questo tipo di arbitraggio sfrutta le discrepanze nella curva della volatilità implicita o nella struttura a termine dei tassi di interesse, che possono influenzare i prezzi delle opzioni in modo differente a seconda del loro tempo alla scadenza. Ecco diverse strategie che i trader potrebbero utilizzare per sfruttare queste inefficienze.

1. Spread di calendario (Spread orizzontali)

Descrizione: implica l'acquisto e la vendita di due opzioni dello stesso tipo (call o put) e stesso strike, ma con diverse date di scadenza. Il trader potrebbe vendere un'opzione a breve termine e acquistare un'opzione a lungo termine, scommettendo sulla differenza di volatilità nel tempo.

Obiettivo: trarre profitto dalla differenza nel decadimento temporale o dai cambiamenti nella struttura a termine della volatilità.

2. Spread diagonali

Descrizione: simile agli spread di calendario ma coinvolge opzioni con strike e date di scadenza differenti. È una combinazione di uno spread verticale e uno spread di calendario.

Obiettivo: sfruttare le differenze nella volatilità e nel decadimento temporale tra due opzioni.

3. Spread farfalla nel tempo

Descrizione: questa strategia implica l'impostazione di uno spread farfalla (acquistando un'opzione in-the-money, vendendo due opzioni at-the-money e acquistando un'opzione out-of-the-money) su diverse date di scadenza.

Obiettivo: trarre profitto dalla convergenza della struttura a termine verso uno stato normale se attualmente è in uno stato anormale.

4. Variance Swap

Descrizione: stipulare un contratto variance swap permette ai trader di speculare o coprirsi contro la variabilità futura del prezzo di un'attività, che può essere correlata alle differenze nella struttura a termine della volatilità implicita nel tempo.

Obiettivo: trarre profitto dalla differenza tra la varianza realizzata dei rendimenti di un'attività e la varianza implicita dalla struttura a termine.

5. Arbitraggio su opzioni sui tassi di interesse

Descrizione: utilizzando opzioni su futures sui tassi di interesse o direttamente su obbligazioni, i trader possono sfruttare le

inefficienze nella struttura a termine dei tassi di interesse.

Obiettivo: trarre profitto dalla errata quotazione delle opzioni basata su cambiamenti attesi nei tassi di interesse che influenzano scadenze differenti in modo differente.

6. Arbitraggio sulla superficie di volatilità

Descrizione: questo implica l'analisi dell'intera superficie di volatilità (attraverso sia prezzi di esercizio sia date di scadenza) per identificare errate quotazioni.

Obiettivo: sfruttare discrepanze nella volatilità implicita per opzioni con strike e scadenze differenti.

7. Strategie di copertura dinamica

Descrizione: implica l'aggiustamento frequente della composizione di un portafoglio di opzioni con scadenze differenti per mantenere una posizione delta-neutral o gamma-neutral.

Obiettivo: trarre profitto dai cambiamenti nella struttura a termine della volatilità o dei tassi di interesse gestendo il rischio.

Considerazioni di implementazione

- *Analisi di mercato: richiede una profonda comprensione dei fattori che influenzano i prezzi delle opzioni, inclusi la volatilità, i tassi di interesse e le loro strutture a termine.*

- *Gestione del rischio: le strategie comportano rischi significativi, incluso il potenziale di perdita se la struttura a termine si muove contro la posizione o se l'opportunità di arbitraggio anticipata non si materializza.*

- *Esecuzione: esecuzione efficiente e bassi costi di transazione sono cruciali a causa dei potenziali margini ridotti delle opportunità di arbitraggio.*

Queste strategie sono sofisticate e comportano un alto livello di rischio, richiedendo una solida comprensione dei modelli di pricing delle opzioni, delle dinamiche di mercato e di rigorose pratiche di gestione del rischio. I trader impiegano queste strategie nei mercati in cui hanno identificato specifiche inefficienze o in cui si aspettano determinati spostamenti nella struttura a termine della volatilità o dei tassi di interesse.

È positivo che ChatGPT elenchi una pletora di strategie possibili, la più semplice delle quali utilizza solo spread di calendario. Quindi, approfondiamo.

Prompt dell'utente

Approfondisci la strategia dello spread di calendario.

Risposta di ChatGPT

Quella dello spread di calendario, nota anche come spread temporale o spread orizzontale, è una strategia di trading di opzioni versatile che implica l'entrare simultaneamente in due posizioni di opzioni dello stesso tipo (due call o due put), con lo stesso prezzo di esercizio ma con diverse date di scadenza. Il trader, tipicamente, vende un'opzione a breve termine (il mese frontale) e acquista un'opzione a lungo termine (il mese posteriore) sullo stesso sottostante. Questa strategia è progettata per sfruttare le differenze nel decadimento temporale (theta) e potenzialmente nella volatilità implicita tra le due opzioni.

Come funziona

- Opzione a breve termine (Vendita): vendendo un'opzione a breve termine, il trader incassa un premio grazie al theta, o decadimento temporale, più alto dell'opzione, poiché le opzioni perdono valore più velocemente man mano che si avvicinano alla scadenza.*
- Opzione a lungo termine (Acquisto): acquistando un'opzione a lungo termine, il trader paga per l'opzione che ha un tasso di decadimento temporale più lento. Questa posizione beneficia di qualsiasi aumento della volatilità implicita o movimenti favorevoli nel prezzo del sottostante nel tempo.*

Obiettivo

L'obiettivo primario di uno spread di calendario è quello di trarre profitto dalla differenza nel decadimento temporale tra le opzioni a breve scadenza e quelle a scadenza più lunga. Lo scenario ideale è che l'opzione a breve termine scada senza valore o quasi, permettendo al trader di trattenere il premio incassato, mentre detiene ancora l'opzione a lungo termine che conserva un significativo valore tempo.

...

Questa non sembra affatto una strategia praticabile, poiché, tipicamente, un'opzione a lunga scadenza avrà un valore tempo più alto di una a breve scadenza. Se non ci aspettiamo movimenti direzionali del sottostante, allora dopo la scadenza dell'opzione a breve termine, perderemo semplicemente valore (tempo) dell'opzione a lungo termine fino alla sua scadenza. Forse se continuiamo a rinnovare l'opzione a breve termine al prossimo periodo di scadenza breve questo funzionerebbe, ma ChatGPT non lo ha suggerito.

E la strategia opposta, di vendere allo scoperto l'opzione a lunga scadenza e acquistare quella a breve scadenza? In questo caso, dovremmo continuare a rinnovare (acquistare) la posizione lunga a breve termine come copertura della posizione corta a lunga scadenza.

In entrambe le strategie, dobbiamo confrontare il premio annualizzato (prezzo) delle opzioni a breve termine con il premio annualizzato della vendita allo scoperto dell'opzione a lunga scadenza, supponendo che non ci siano cambiamenti nel prezzo del sottostante né nella volatilità.

Prompt dell'utente

Hai suggerito che dovremmo vendere allo scoperto un'opzione a breve termine e acquistare un'opzione a lungo termine. Supponiamo che questa sia una put sull'SPX. Puoi tracciare i valori tempo annualizzati di queste due put in funzione del loro tempo alla scadenza?

Risposta di ChatGPT

...

[Nota dell'Autore: vedi il grafico nella Figura 2.2.]

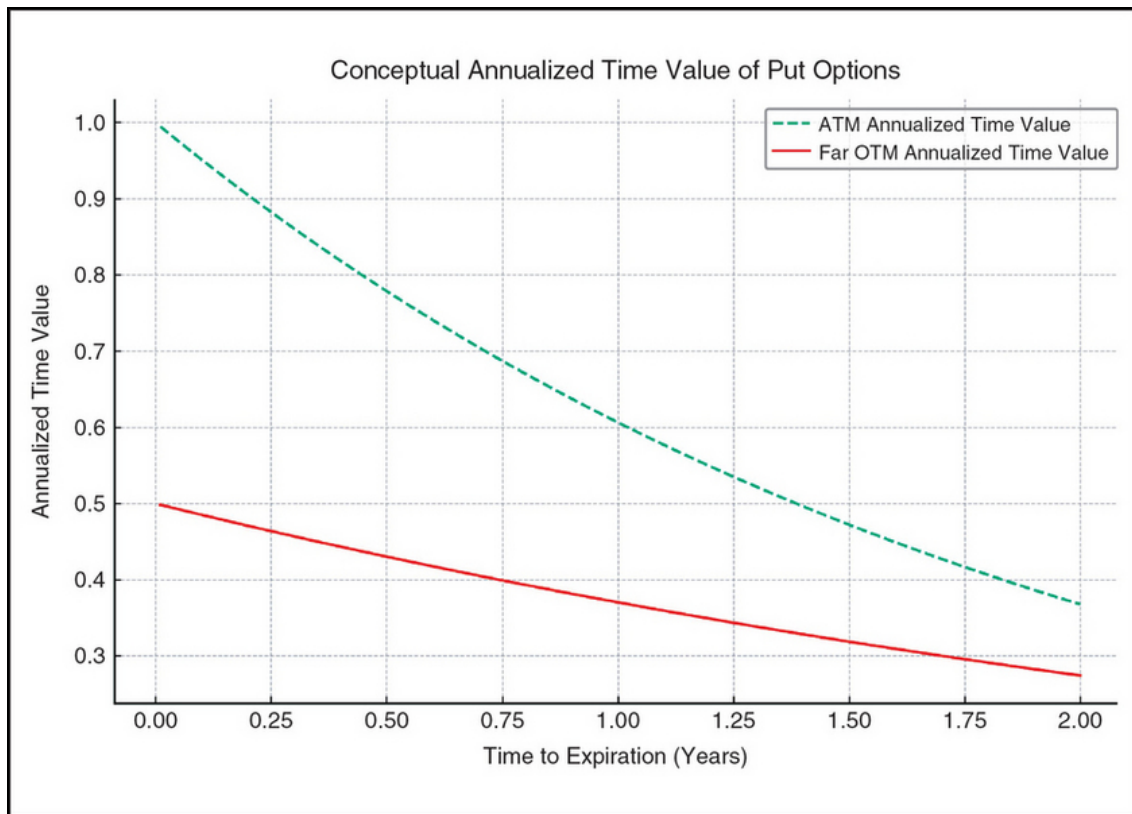


Figura 2.2 Grafico errato del valore tempo annualizzato delle opzioni put in funzione del tempo alla scadenza.

Prompt dell'utente

Puoi tracciare lo stesso grafico per una put molto out-of-the-money?

Il grafico nella Figura 2.2 non è corretto: abbiamo confrontato queste curve con i prezzi reali delle opzioni osservati dai dati di mercato live, e sono enormemente fuori target per le opzioni OTM. Esploriamo ulteriormente.

Prompt dell'utente

Per favore, questa volta traccialo per le volatilità implicite.

Risposta di ChatGPT

...

La Figura 2.3 mostra che la volatilità implicita si comporta in modo opposto ai prezzi delle opzioni! È piuttosto controintuitivo: sta avendo un'allucinazione? Come abbiamo imparato, è meglio chiedere a ChatGPT del codice, piuttosto che l'output dei modelli che utilizza internamente. Per produrre codice, in questo caso, abbiamo bisogno di un modello di pricing delle opzioni, ma il modello Black-Scholes è inadatto.

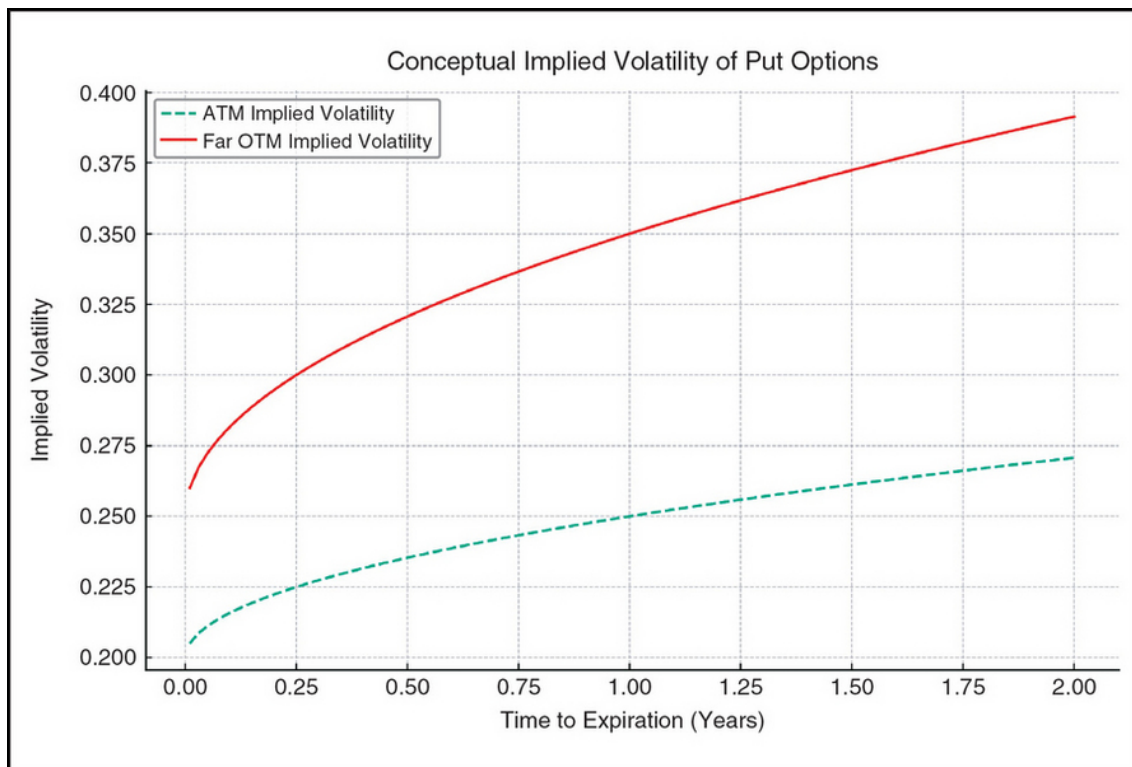


Figura 2.3 Grafico errato della volatilità implicita delle opzioni put in funzione del tempo alla scadenza.

Prompt dell'utente

Conosci un modello teorico di pricing delle opzioni che mostri una superficie di volatilità non banale con caratteristiche come il "sorriso di volatilità" e il fatto che le opzioni a breve scadenza hanno una volatilità implicita diversa dalle opzioni a lunga scadenza dello stesso strike, mentre le opzioni a breve scadenza hanno un theta più negativo di quelle a lunga scadenza?

Risposta di ChatGPT

Sì, i fenomeni che stai descrivendo, incluso il sorriso di volatilità e i modelli osservati nella volatilità implicita e nel theta attraverso diverse date di scadenza, sono affrontati da diversi modelli avanzati di pricing delle opzioni che vanno oltre il framework di Black-Scholes. Questi modelli incorporano feature per tenere conto delle imperfezioni del mercato e dei comportamenti osservati che il modello Black-Scholes, con le sue ipotesi semplificatrici (come la volatilità costante, la distribuzione log-normale dei prezzi azionari), non può catturare appieno. Ecco alcuni modelli teorici noti per la loro capacità di riflettere una superficie di volatilità più complessa e realistica.

...

2. Esempi di modelli di volatilità stocastica: modello di Heston, modello SABR. Feature principali: nei modelli di volatilità stocastica, la volatilità del sottostante è essa stessa un processo stocastico. Questi modelli possono catturare il sorriso di volatilità, perché consentono cambiamenti nella volatilità nel tempo, indipendentemente dai movimenti di prezzo del sottostante. Caratteristiche: questi modelli possono spiegare varie osservazioni empiriche, incluso il sorriso di volatilità e i cambiamenti nelle volatilità implicite attraverso diverse date di scadenza. Consentono anche la correlazione tra il prezzo dell'attività e la sua volatilità, che può influenzare il pricing delle opzioni a breve termine rispetto a quelle a lungo termine.

...

ChatGPT ha elencato quattro tipi di modelli di pricing delle opzioni: modelli di volatilità locale che utilizzano i prezzi delle opzioni per calibrare la superficie di volatilità a diverse scadenze e strike; modelli di volatilità stocastica (come Heston) che utilizzano alcuni parametri per produrre la forma generale della superficie di volatilità; modelli di salto-diffusione che consentono salti nel prezzo del sottostante in aggiunta al moto browniano gaussiano; e i modelli di processo di Levy, che modellano il movimento del sottostante come un processo di Levy. Il modello di Heston sembra il più semplice e appropriato per avere solo una comprensione concettuale della struttura a termine senza ottenere una corrispondenza esatta con i dati reali.

Prompt dell'utente

Scegliamo il modello di Heston. Puoi approfondirlo?

Risposta di ChatGPT

Il modello di Heston è un noto modello di volatilità stocastica per il pricing delle opzioni, introdotto da Steven Heston nel 1993. Affronta alcune delle limitazioni del modello Black-Scholes, in particolare il suo presupposto di una volatilità costante. Il modello di Heston consente alla volatilità del sottostante di essere stocastica, il che significa che può cambiare nel tempo secondo uno specifico processo stocastico. Questo permette al modello di catturare più accuratamente fenomeni di mercato come il sorriso di volatilità.

Caratteristiche principali del modello di Heston

- 1. Volatilità stocastica: il modello descrive l'evoluzione del prezzo del sottostante e della sua varianza (volatilità al quadrato) utilizzando due equazioni differenziali stocastiche. La prima modella la dinamica del prezzo dell'attività in modo simile a Black-Scholes, ma incorpora un termine di volatilità stocastica. La seconda modella la dinamica della varianza stessa.*
- 2. Mean Reversion: il processo di varianza nel modello di Heston è mean-reverting, il che significa che tende a tornare a un livello medio a lungo termine nel tempo. Ciò è coerente con il comportamento osservato del mercato, dove di solito i picchi di volatilità sono temporanei.*
- 3. Correlazione tra rendimenti dell'attività e volatilità: il modello consente una correlazione tra i rendimenti dell'attività e i cambiamenti di volatilità. Questa correlazione è cruciale per catturare accuratamente l'effetto leva, dove i prezzi delle attività e la loro volatilità tendono a essere inversamente correlati.*

Formulazione matematica

La dinamica del prezzo dell'attività sottostante (S) e della sua varianza (v) nel modello di Heston sono descritte dalle seguenti equazioni differenziali stocastiche:

$$\begin{aligned}dS_t &= \mu S_t dt + \sqrt{v_t} S_t dW_t^S \\dv_t &= \kappa(\theta - v_t) dt + \sigma \sqrt{v_t} dW_t^v\end{aligned}$$

dove:

S_t è il prezzo dell'attività al tempo t ;

v_t è la varianza del prezzo dell'attività al tempo t ;

μ è il tasso di rendimento privo di rischio. [Nota dell'Autore: questo non è corretto! Dovrebbe essere il drift istantaneo deterministico (cioè non stocastico) dei rendimenti del prezzo]

dell'attività! Vedi Gatheral (2006).]

κ è il tasso di mean reversion della varianza;

θ è il livello medio della varianza a lungo termine;

σ è la volatilità della volatilità (vol of vol), che determina la volatilità del processo di varianza.

dW_t^S e dW_t^V sono due processi di Wiener per il prezzo dell'attività e la varianza, rispettivamente, con un coefficiente ρ .

Prezzo delle opzioni con il modello di Heston

Il modello di Heston consente di derivare una formula semi-analitica per il prezzo di opzioni di tipo europeo. La formula di pricing delle opzioni implica integrazioni complesse a causa della natura stocastica del modello, ma può essere calcolata in modo efficiente utilizzando metodi numerici come la trasformata di Fourier veloce.

Applicazioni e limitazioni

Applicazioni: il modello di Heston è ampiamente utilizzato nei mercati finanziari per il pricing di opzioni europee, catturando la superficie di volatilità in modo più accurato rispetto al modello di Black-Scholes, ed è anche esteso per prezzare opzioni americane ed esotiche in determinate condizioni.

Limitazioni: nonostante le sue caratteristiche avanzate, il modello di Heston può essere difficile da calibrare ai dati di mercato, a causa della complessità delle sue dinamiche e della sensibilità dei suoi parametri. Inoltre, come tutti i modelli, applica semplificazioni e ipotesi che potrebbero non valere in tutte le condizioni di mercato.

La capacità del modello di Heston di incorporare la volatilità stocastica e la mean reversion lo rende uno strumento potente per comprendere e prezzare le opzioni in modo più realistico, riflettendo la natura dinamica dei mercati finanziari.

Prompt dell'utente

Puoi citare pubblicazioni che mostrino la struttura a termine della volatilità implicita di un'opzione in funzione del tempo alla scadenza.

Risposta di ChatGPT

Libri e articoli

1. *Option Volatility and Pricing* di Sheldon Natenberg – Questo libro è una risorsa ampiamente riconosciuta nel campo del trading di opzioni. Natenberg fornisce spiegazioni dettagliate sul comportamento della volatilità implicita e sul suo impatto sul pricing delle opzioni, incluse discussioni sulla struttura a termine della volatilità.

2. *Dynamic Hedging* di Nassim Nicholas Taleb – Il lavoro di Taleb si concentra sugli aspetti pratici della gestione di posizioni in opzioni, includendo approfondimenti sulla struttura a termine della volatilità e su come può essere sfruttata attraverso strategie di hedging dinamico.

3. *The Volatility Surface: A Practitioner's Guide* di Jim Gatheral – Gatheral fornisce un'analisi approfondita della superficie di volatilità, che include discussioni sulla struttura a termine della volatilità implicita e su come varia con il tempo alla scadenza e il prezzo di esercizio.

...

Scegliamo il libro di Jim Gatheral (Gatheral, 2006) come riferimento, in quanto sembra trattare l'argomento in questione con molti grafici. Sfortunatamente, nessuno di questi è esattamente ciò che vogliamo: un confronto della struttura a termine dei prezzi di opzioni ATM e OTM basato sul modello di Heston. Per farlo, dobbiamo istanziare un modello di Heston. Ma prima, dobbiamo sapere quali input sono ragionevoli.

Prompt dell'utente

Quali sono stime ragionevoli per μ quando l'attività in questione è l'SPX?

Risposta di ChatGPT

Nei modelli finanziari, il parametro μ rappresenta tipicamente il rendimento atteso o il tasso di drift dell'attività sottostante nel tempo. Per l'S&P 500 Index (SPX), stime ragionevoli per μ dovrebbero riflettere il rendimento medio a lungo termine dell'indice, tenendo conto dei dividendi, dell'inflazione e della crescita generale dell'economia.

Prospettiva storica su μ per l'SPX

L'S&P 500 ha storicamente reso annualmente circa dal 7% al 10% tenendo conto dell'inflazione. Questo intervallo tiene conto dei dividendi reinvestiti e rappresenta una prospettiva a lungo termine su più decenni. La cifra esatta può variare a seconda dello specifico periodo temporale considerato, di come vengono trattati i dividendi e del metodo di rettifica per l'inflazione.

...

Prompt dell'utente

Quali sono stime ragionevoli per il tasso di mean reversion della varianza quando l'attività in questione è l'SPX?

Risposta di ChatGPT

Il tasso di mean reversion κ nel modello di Heston è un parametro chiave che descrive quanto rapidamente la varianza dei rendimenti dell'attività ritorna alla sua media a lungo termine θ . Per l'S&P 500 Index (SPX), come per altre attività finanziarie, il valore specifico di κ può variare in base alle condizioni di mercato, al periodo analizzato e alla metodologia di calibrazione utilizzata. Tuttavia, studi empirici e la pratica di mercato forniscono alcune linee guida su stime ragionevoli per questo parametro.

Stime ragionevoli per κ

- Intervallo tipico: per indici azionari come l'SPX, spesso i valori di κ si trovano nell'intervallo da 1 a 3 per anno. Questo intervallo suggerisce che la varianza tende a ritornare alla sua media a lungo termine entro pochi mesi e fino a un anno.

...

Prompt dell'utente

Quali sono stime ragionevoli per il livello medio della varianza a lungo termine per l'SPX?

Risposta di ChatGPT

Il livello medio della varianza a lungo termine θ nel modello di Heston rappresenta il livello al quale ci si aspetta che la varianza dei rendimenti dell'attività ritorni nel tempo. Per l'S&P 500 Index (SPX), come per altre attività, questo parametro è cruciale per modellare il comportamento a lungo termine della volatilità di mercato. Stimare θ implica analizzare i dati storici di volatilità e calibrare il modello ai prezzi di mercato correnti delle opzioni.

Stime ragionevoli per θ

Dato che θ è la media a lungo termine della varianza, non della volatilità, è importante distinguere tra le due quando si discutono le stime. La varianza v_t è il quadrato della volatilità σ [Nota dell'Autore: questo è sbagliato. Qui σ è la volatilità della volatilità, non la volatilità], quindi quando i trader si riferiscono

ai livelli di volatilità, spesso discutono di σ [Nota dell'Autore: stesso errore di sopra] , mentre θ riguarda v_t .

- Conversione da volatilità a varianza: per pensare a θ in termini di livelli di volatilità familiari, è prima necessario elevare al quadrato la volatilità annualizzata. Per esempio, se l'aspettativa di volatilità a lungo termine per l'SPX è circa il 20% (o 0.20 in forma decimale), la varianza corrispondente θ sarebbe ($0.20^2 = 0.04$).

- Intervallo tipico per SPX: storicamente, la volatilità annualizzata dell'SPX ha fluttuato, ma un intervallo dal 15% al 30% (da 0.15 a 0.30 in forma decimale) è comune per stimare la volatilità a lungo termine, a seconda del periodo temporale e delle condizioni di mercato. Elevando al quadrato questi valori si ottiene un intervallo di varianza approssimativamente da 0.0225 a 0.09 per θ .

Prompt dell'utente

Quali sono delle stime ragionevoli per σ per l'SPX?

Risposta di ChatGPT

Nel contesto del modello di Heston, σ rappresenta la "volatilità della volatilità", che quantifica la variabilità nella volatilità implicita dell'attività sottostante. Questo parametro è cruciale per catturare le dinamiche della superficie di volatilità, inclusa la curvatura e il comportamento dello skew della volatilità implicita. Per l'S&P 500 Index (SPX), le stime ragionevoli per σ possono variare in base alle condizioni di mercato storiche, al periodo di calibrazione e allo specifico approccio di calibrazione del modello. [Nota dell'Autore: in questa descrizione, ha indicato correttamente i parametri σ , che, come descritto, sta per la volatilità della volatilità.]

Stime ragionevoli per σ

- Intervallo tipico: per l'SPX, le stime ragionevoli per σ (volatilità della volatilità) spesso ricadono nell'intervallo da 0.1 a 0.4 per anno. Questo intervallo riflette la variabilità osservata nella volatilità implicita di mercato, ma può variare a seconda dello specifico ambiente di mercato e del periodo storico analizzato.

...

Prompt dell'utente

Quali sono stime ragionevoli per il coefficiente di correlazione ρ per l'SPX?

Risposta di ChatGPT

Il coefficiente di correlazione ρ nel contesto del modello di Heston rappresenta la correlazione tra i rendimenti dell'attività e la sua volatilità. Per l'S&P 500 Index (SPX), così come per altre attività, questa correlazione tende a essere negativa, riflettendo l'osservazione comune del mercato che i prezzi delle attività spesso si muovono inversamente alla loro volatilità. Questo fenomeno è a volte indicato come "effetto leva", dove i cali del mercato sono tipicamente associati ad aumenti della volatilità.

Stime ragionevoli per ρ - Intervallo tipico: per l'SPX, le stime ragionevoli per il coefficiente di correlazione ρ spesso ricadono nell'intervallo da (-0.5) a (-0.8) . Questo intervallo è coerente con studi empirici che hanno esaminato la relazione tra i rendimenti azionari e la volatilità.

Generalmente sceglieremo il valore medio di un dato intervallo di stime appena fornito, ma utilizziamo l'estremo inferiore della stima di θ (varianza a lungo termine dell'SPX), poiché il livello di volatilità è calato negli anni, e una stima all'estremo superiore di σ (volatilità della volatilità dell'SPX), poiché questa è aumentata negli anni (vedi Chan (2013), Figura 5.10). Scegliamo la stima all'estremo superiore anche per μ (rendimento a lungo termine dell'SPX) perché ha avuto una tendenza al rialzo negli anni recenti.

Prompt dell'utente

Ipotizziamo $\kappa = 2$, $\theta = 0.152$, $\sigma = 0.4$, $\rho = -0.65$, $\mu = 0.1$, varianza iniziale $v_0 = 0.152$. Mostra il codice Matlab per calcolare il prezzo dell'opzione ATM utilizzando il suo Financial Toolbox.

Risposta di ChatGPT

Per calcolare il prezzo di un'opzione At-The-Money (ATM) sull'SPX utilizzando il modello di Heston in MATLAB, tipicamente si utilizzerebbero le funzioni fornite dal Financial Toolbox di MATLAB. I parametri del modello di Heston che hai specificato sono:

- Tasso di mean reversion, $\kappa = 2$
- Livello medio della varianza a lungo termine, $\theta = 0.152$
- Volatilità della volatilità, $\sigma = 0.4$
- Coefficiente di correlazione, $\rho = -0.65$
- Rendimento atteso o tasso di drift, $\mu = 0.1$
- Varianza iniziale, $v_0 = 0.152$

Ecco uno script MATLAB concettuale per calcolare il prezzo di un'opzione ATM utilizzando il modello di Heston. Nota che il Financial Toolbox di MATLAB potrebbe richiedere funzioni specifiche per il pricing delle opzioni, come `optByHestonNI` (utilizzando l'integrazione numerica) per calcolare i prezzi delle opzioni europee.

```
% Parametri del modello di Heston
kappa = 2;
theta = 0.15^2;
sigma = 0.4;
rho = -0.65;
v0 = 0.15^2;
r = 0.1; % Tasso privo di rischio, supponendo sia lo stesso di mu per
semplicità
    % [Nota dell'Autore: questo è sbagliato! r dovrebbe essere
0.03.]
% Definisce il percorso dell'attività
S0 = 100; % Livello iniziale SPX di esempio, modificare secondo
necessità
    % [Nota dell'Autore: dovrebbe essere circa 5.000 a febbraio
2024.]
K = S0; % Opzione ATM
T = 1; % Tempo alla scadenza in anni
% Definisce il tipo di opzione
OptType = {'call', 'put'}; % Scegliere secondo necessità
% Prezzo dell'opzione utilizzando il modello di Heston
for i = 1:length(OptType)
    price = optByHestonNI(kappa, theta, sigma, rho, v0, r, T, S0, K,
'OptSpec',
                        OptType{i});
    fprintf('%s option price using Heston model: %f\n', OptType{i},
price);
end
```

Questo script utilizza la funzione `optByHestonNI` per calcolare i prezzi delle opzioni basati sui parametri del modello di Heston. L'opzione `OptSpec` consente di specificare il tipo di opzione ('call' o 'put'). Modifica `S0`, `K` e `T` secondo necessità per riflettere, rispettivamente, il prezzo corrente dell'SPX, il prezzo di esercizio per l'opzione ATM e il tempo desiderato alla scadenza.

Assicurati di aver installato il Financial Toolbox e di consultare la documentazione MATLAB per conoscere le nuove funzioni e la loro sintassi, poiché il nome delle funzioni e i parametri possono cambiare. Questo esempio ipotizza uno scenario semplificato a scopo illustrativo.

Non ci fidiamo del tutto del codice generato, ma siamo lieti che ChatGPT abbia indicato la funzione `optByHestonNI()` del Matlab Financial Instruments Toolbox (sebbene non il Financial Toolbox che ha suggerito). Faremo riferimento alla documentazione ufficiale

all'indirizzo https://www.mathworks.com/help/fininst/optbyhestonni.html?s_tid=doc_ta#d126e204312 per correggere questo script. Una versione finalizzata potrà dare un grafico per le “put” simile a quello rappresentato nella Figura 2.4. Quello per le “call” sarà simile a quello rappresentato nella Figura 2.5.

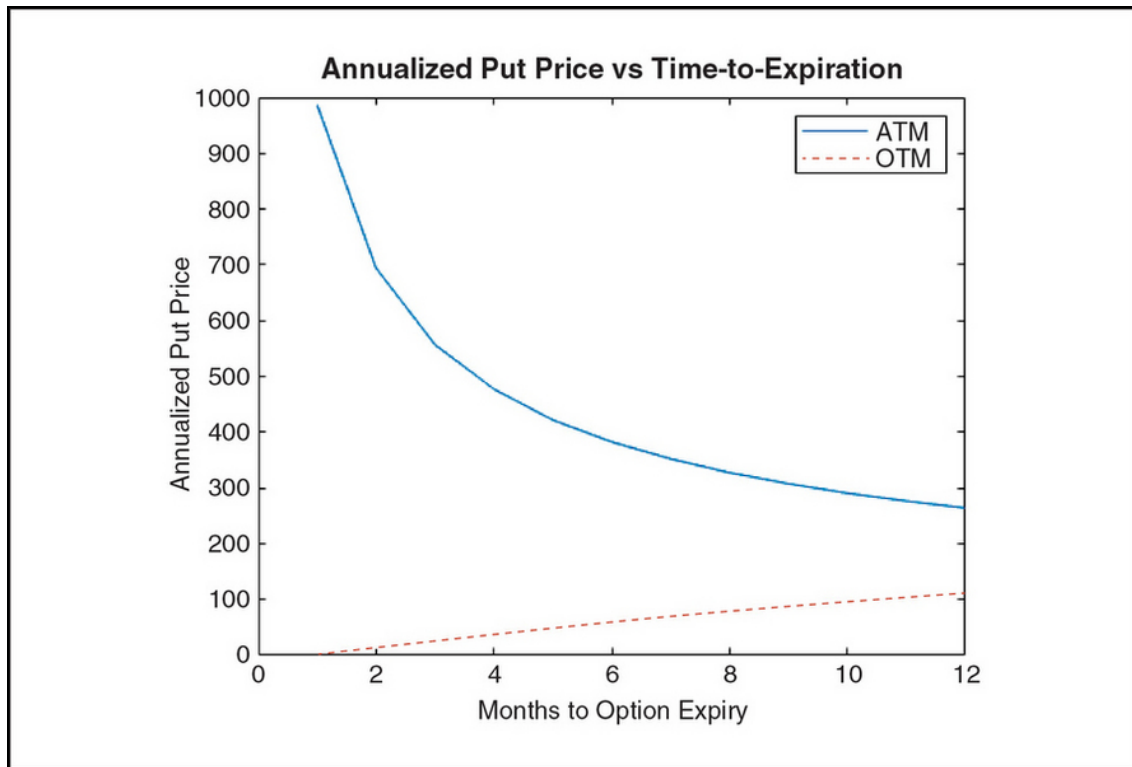


Figura 2.4 Grafico dei prezzi annualizzati delle opzioni put in funzione del tempo alla scadenza basati sul modello di Heston.

Esercizio #2

Chiedete a ChatGPT una versione equivalente in Python.

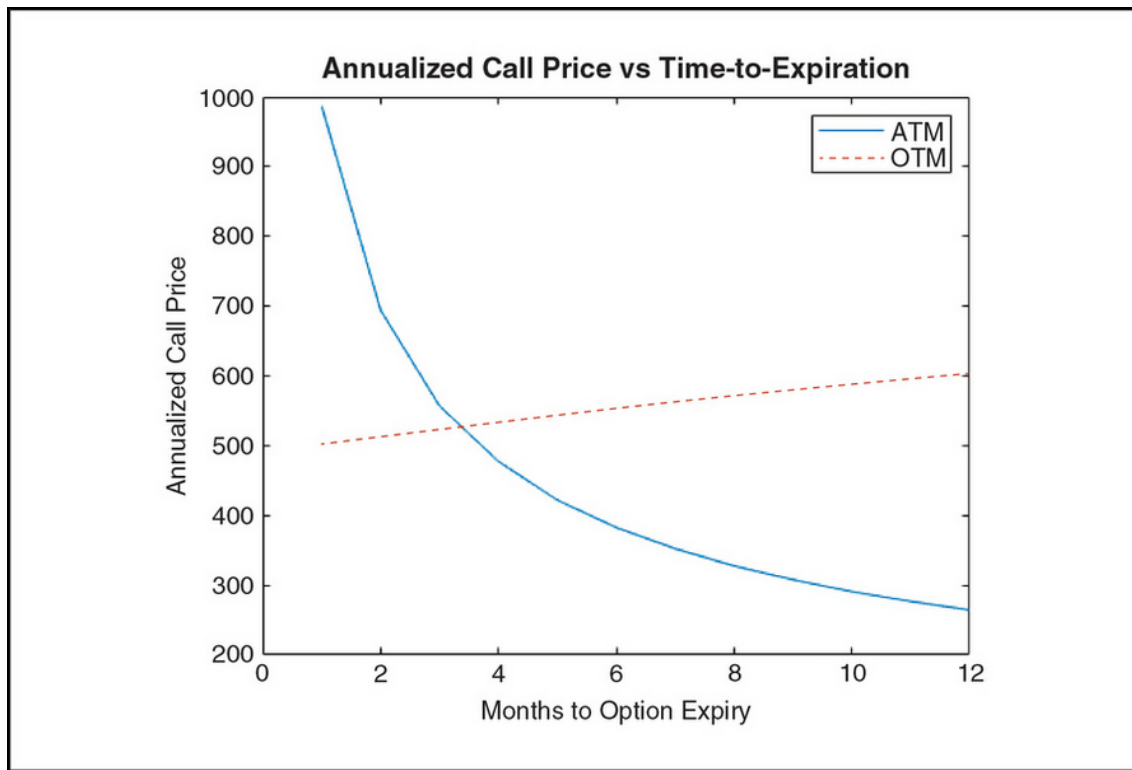


Figura 2.5 Grafico dei prezzi annualizzati delle opzioni call in funzione del tempo alla scadenza basati sul modello di Heston.

Sulla base di queste curve, sembra che possiamo vendere allo scoperto un'opzione OTM a lunga scadenza e coprirla acquistando un'opzione OTM a breve scadenza che viene continuamente rinnovata in avanti prima della sua scadenza, a meno che a un certo punto non diventi un'opzione ATM a causa di cambiamenti nel livello dell'SPX. A quel punto, dovremmo liquidare le posizioni, perché il costo della copertura dell'opzione a lunga scadenza supererà il valore temporale annualizzato di quell'opzione. La strategia opposta, acquistare un'opzione ATM a lunga scadenza e vendere allo scoperto un'opzione a breve scadenza, non funzionerebbe, perché quando l'opzione a breve scadenza scade, l'opzione a lunga scadenza molto probabilmente non sarà più ATM e la struttura a termine notata non si applicherà (a meno che non ci aspettiamo un catalizzatore nel

mercato che potrebbe produrre un movimento significativo nel sottostante dopo la scadenza della prima opzione, tale per cui l'opzione a lunga scadenza diventa ATM). C'è un'ampia gamma di livelli dell'SPX che rendono le opzioni OTM, ma un intervallo ristretto che le rende ATM.

E la struttura a termine della volatilità implicita? È uguale o opposta al valore annualizzato dell'opzione, come suggerito in precedenza da ChatGPT? Ma... un attimo: nel modello di Heston, la volatilità è stocastica e cambia costantemente. Che cosa significa addirittura la struttura a termine della volatilità implicita (nota anche come superficie di volatilità)?

Prompt dell'utente

Dato che il modello di Heston utilizza come base un modello di volatilità stocastica, ha senso chiedere la volatilità implicita di un'opzione, se è stocastica nel tempo?

Risposta di ChatGPT

La tua domanda tocca un aspetto delicato della teoria del pricing delle opzioni e dell'interpretazione della volatilità implicita nel contesto di modelli di volatilità stocastica come il modello di Heston.

Volatilità implicita nel framework Black-Scholes: nel modello di Black-Scholes, la volatilità implicita è un valore costante che, quando inserito nel modello, produce un prezzo dell'opzione che corrisponde al prezzo di mercato. Rappresenta una misura media singola della volatilità attesa durante la vita dell'opzione. Questo modello ipotizza che la volatilità sia nota e costante, il che semplifica il calcolo ma non cattura la complessità delle condizioni reali del mercato.

Modelli di volatilità stocastica: i modelli di volatilità stocastica, incluso il modello di Heston, riconoscono che la volatilità non è costante ma varia nel tempo in modo casuale. Questi modelli forniscono un framework più realistico per il pricing delle opzioni incorporando un processo stocastico per la volatilità.

Volatilità implicita all'interno dei modelli di volatilità stocastica: anche se il modello di Heston e altri modelli di volatilità stocastica adottano una visione dinamica della volatilità, i trader calcolano ancora le volatilità implicite da questi modelli per diverse ragioni.

1. Benchmark di mercato: la volatilità implicita serve come benchmark standardizzato, consentendo a trader e analisti di confrontare i prezzi

delle opzioni tra diversi strike e scadenze in modo coerente, anche nel contesto di modelli che riconoscono la natura dinamica della volatilità.

2. Calibrazione del modello: la volatilità implicita viene utilizzata per calibrare i modelli di volatilità stocastica ai dati di mercato. Modificando i parametri del modello per far corrispondere le volatilità implicite di mercato osservate su una gamma di strike e scadenze, i trader si assicurano che il modello rifletta il più fedelmente possibile le condizioni di mercato correnti.

3. Superficie di volatilità: il concetto di volatilità implicita è fondamentale per costruire la superficie di volatilità, che traccia le volatilità implicite attraverso diversi strike e scadenze. Anche nel contesto dei modelli di volatilità stocastica, la superficie di volatilità fornisce conoscenze preziose sulle aspettative e sul sentiment del mercato.

4. Strumento di comunicazione: nonostante i suoi limiti, la volatilità implicita rimane uno strumento utile per comunicare le condizioni di mercato e le percezioni del rischio tra trader, analisti e altri partecipanti al mercato.

Conclusione: potrebbe sembrare controintuitivo discutere di volatilità implicita nel contesto di un modello che riconosce intrinsecamente la natura stocastica della volatilità, ma farlo permette l'applicazione pratica di questi modelli nel trading e nella gestione del rischio nel mondo reale. La volatilità implicita, derivata dai prezzi di mercato delle opzioni, colma il divario tra modelli teorici e pratica di mercato, servendo come linguaggio comune per i partecipanti al mercato e come strumento per la calibrazione dei modelli.

A differenza di ciò che fanno spesso i teorici accademici, non ci preoccuperemo di tracciare la superficie di volatilità. Non è empiricamente osservabile e non ci offre alcun vantaggio diretto. Come trader, preferiamo cose su cui possiamo effettivamente fare trading.

Conclusioni

Abbiamo chiesto a ChatGPT di creare del codice di backtest per una strategia di trading long-short equity utilizzando i fattori Fama-French e una strategia coperta di carry su futures VIX. Inoltre, abbiamo chiesto a ChatGPT di suggerire una strategia di calendar spread su opzioni e di esplorare la struttura a termine dei prezzi delle opzioni. Abbiamo posto a Copilot una domanda meno impegnativa:

scoprire se qualcuno ha pubblicato una tecnica di DRL (*Deep Reinforcement Learning*) per l'ottimizzazione del portafoglio. Da questi esperimenti, concludiamo quanto segue.

1. Non possiamo usare ChatGPT per automatizzare l'intero processo. Ciò che serve è una combinazione di intelligenza, conoscenza, esperienza umana e di suggerimenti dell'LLM. La situazione è per certi versi analoga alle auto a guida autonoma. È raro trovare veicoli completamente autonomi, tranne in poche aree urbane sperimentali, ma la tecnologia di guida assistita si trova ormai incorporata già da anni in quasi tutte le auto nuove (Chan, 2022). Siamo però ancora piuttosto lontani da una vera intelligenza artificiale generale.
2. Le risposte di ChatGPT spesso non sono riproducibili: lo stesso prompt può portare a risposte differenti. Questo è un problema per la maggior parte dei progetti di "ingegneria", che richiedono una rigorosa riproducibilità.
3. Potrebbe essere più efficiente usare ChatGPT come un coach per accelerare i propri compiti di programmazione che chiedergli di scrivere del codice ogni volta che se ne ha bisogno.

Queste conclusioni non sono dissimili da quelle di Simon Willison (creatore di Django e Datasets) come riportato in Claburn (2024). Abbiamo bisogno di un tocco umano nel "loop" per le applicazioni più serie dell'intelligenza artificiale.

Questo non vuol dire che non ci siano occasionali lampi di genialità o prove di comprensione concettuale della finanza quantitativa, come quando ChatGPT capisce che una strategia di trading che detiene una posizione per n giorni può avere fino a n posizioni in un singolo giorno, o se quando si calcola l'indice di Sharpe sia sempre necessario sottrarre il tasso privo di rischio dai rendimenti lordi. Questi "lampi di intelligenza artificiale generale"

sono ben documentate nell'articolo di Bubeck et al. (2023). Ma tale genialità è sembrata svanire quando abbiamo chiesto a ChatGPT di analizzare il documento sul VIX e produrre lo pseudocodice per la strategia di trading in esso contenuta, così accuratamente descritta. Anche dopo aver indicato a ChatGPT il paragrafo specifico con le informazioni rilevanti, non è riuscito a catturare l'intera strategia di trading. Forse è solo una limitazione relativa alla lunghezza di un documento che GPT-4o può gestire in una sola volta, ma possiamo solo dire che lo stato dell'arte oggi disponibile per un trader professionista semplicemente non è ancora adeguato (al momento della stesura di questo testo). Come mostra il nostro sforzo di individuare un articolo sull'applicazione del *Deep Reinforcement Learning* all'ottimizzazione del portafoglio, Copilot non è molto bravo a individuare informazioni che soddisfino le nostre esigenze professionali. Possiamo forse migliorare questa capacità affinando il modello su un database di finanza quantitativa, o con un prompt engineering particolarmente esperto. Per esempio, è noto che il "chain-of-thought prompting", ovvero "chiedere ai modelli linguistici di generare soluzioni passo-passo ha consentito ai modelli di risolvere problemi che in precedenza sembravano irraggiungibili" (Brukbaker, 2024). Ma se all'utente è richiesto tale livello di competenze, altamente tecniche, difficilmente si può parlare di intelligenza artificiale generale. Naturalmente, GPT migliorerà a ogni nuova generazione (qui stiamo usando GPT-4o), le sue capacità miglioreranno. Ciò per cui ChatGPT e altri strumenti di GenAI sono bravi è fornire una prima bozza di "qualcosa" che poi può essere corretta. In quanto tale, è utile come un "collaboratore junior", che può motivare il collaboratore senior a provare cose nuove. Spesso, scrivere la prima bozza è la cosa più difficile. Per noi esseri umani è più facile rivedere e migliorare qualcosa. Quindi, se ci chiedeste

“Dopo aver scritto questi due capitoli, usereste ancora la GenAI per farvi aiutare con la scoperta di strategie di investimento e il backtest?”, la risposta sarebbe “Sì”.

Potreste chiedervi, dato lo stato della GenAI, perché dovremmo approfondirla ulteriormente. La ragione è questa: la maggior parte dei compiti che abbiamo assegnato alla GenAI finora sono quelli che hanno una risposta chiara, e gli esperti umani possono eseguirli altrettanto bene, se non meglio. Naturalmente, non tutti noi siamo esperti in ogni materia, e quindi possiamo risparmiare tempo facendo fare all’intelligenza artificiale ciò che sappiamo fare, se può farlo bene. Questa era la ragione per cui abbiamo esplorato l’intelligenza artificiale no-code. Tuttavia, ci sono compiti per i quali non ci sono risposte chiare, e perfino gli esperti umani potrebbero non essere d’accordo su quali siano le migliori risposte. Nella diagnostica medica, potremmo chiedere: “Dati i sintomi, qual è la probabile malattia che affligge il paziente?”. Nella gestione patrimoniale, “NVDA aumenterà i suoi utili il prossimo trimestre?”, “Il prezzo del petrolio scenderà il prossimo mese?” o “Ciò che il presidente della Fed ha appena detto in conferenza stampa è positivo o negativo per l’S&P 500?” (e la risposta della GenAI a quest’ultima domanda è un esempio che troverete più avanti in questo libro). Queste sono tutte domande senza risposte chiare, se non col senno di poi. Questo è il regno del trading discrezionale o fondamentale. Poiché macchine ed esseri umani sono su un piano più equo per queste domande, potremmo scoprire che la GenAI può aggiungere valore. Vogliamo quindi scoprire se è possibile costruire un “George Soros on-chip”.

Il prossimo capitolo preparerà il terreno per questo ambizioso progetto. Ciò comporterà inevitabilmente l’utilizzo di API di intelligenza artificiale generativa.

Esercizi

1. Chiedete a ChatGPT di creare del codice di backtest per l'articolo Zhang et al. (2019) su *Deep Reinforcement Learning for Trading*. ChatGPT utilizza Tensorflow? Potete creare unit test per vedere se il codice è corretto?
2. Cos'è la RAG (*Retrieval Augmented Generation*) e come correggerebbe alcune delle allucinazioni di cui possono essere vittima gli LLM?
3. Chiedete all'ultima versione di ChatGPT di vedere se ora riesce a riassumere correttamente la strategia VIX dall'articolo citato.
4. Chiedete a ChatGPT di creare del codice per calcolare il profitto atteso di un iron condor per opzioni SPX con un periodo di detenzione di un mese, ipotizzando una distribuzione normale dei rendimenti e stime di volatilità ragionevoli. Osservate come esegue con disinvoltura l'integrazione numerica. Domanda bonus: chiedetegli di generare il codice per calcolare anche il suo indice di Sharpe.

Calcolo del rendimento del giorno successivo

Ecco il programma Python che calcola i rendimenti del giorno successivo (o, in generale, dei prossimi n giorni). Questi rendimenti saranno utilizzati come “etichette” (variabile y) per il modello di regressione Fama-French.

```
import pandas as pd
def calculate_N_day_ahead_returns(file_path, output_file_path, N=1):
    try:
        # Carica i dati
        stock_prices = pd.read_csv(file_path)
        # Rinomina le colonne
```

```

        stock_prices=stock_prices.rename(columns={'tday': 'Date',
'syms': 'Stock_ID',
                                                'mid': 'Price'})
        # Converte la colonna 'Date' dal formato intero YYYYMMDD al
formato datetime
        stock_prices['Date'] = pd.to_datetime(stock_prices['Date'],
format='%Y%m%d')
        # Ordina i dati per 'Stock_ID' e 'Date'
        stock_prices.sort_values(by=['Stock_ID', 'Date'], inplace=True)
        # Sposta i prezzi di -N giorni per allinearsi con il prezzo tra
N giorni
        stock_prices['Price_in_N_days'] =
stock_prices.groupby('Stock_ID')['Price'].shift(-N)
        # Calcola i rendimenti a N giorni
        stock_prices[str(N)+'_Day>Returns'] =
(stock_prices['Price_in_N_days'] -
                                                stock_prices['Price']) /
stock_prices['Price']
        # Elimina la colonna utilizzata per il calcolo
        stock_prices.drop(columns=['Price_in_N_days'], inplace=True)
        # Salva i risultati in un nuovo file CSV
        stock_prices.to_csv(output_file_path, index=False)
        return "Successfully computed 30-day returns and saved to " +
output_file_path
    except Exception as e:
        return f"An error occurred: {e}"
# Esempio di utilizzo
file_path = 'CRSP_mid.csv' # Sostituisci con il percorso effettivo del
file
output_file_path = 'CRSP_nextDayReturns.csv'
# Sostituisci con il percorso desiderato per il file di output
result = calculate_N_day_returns(file_path, output_file_path)
print(result)

```

Caricamento dei fattori Fama-French

Il file dei fattori Fama-French è stato scaricato da

https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.htm

ml. Dobbiamo solo creare un dataframe Pandas per contenere i dati con le date appropriate.

```

import pandas as pd
# Percorso del file (sostituisci con il tuo percorso file)
file_path = 'F-F_Research_Data_Factors_daily.CSV'
# Primo tentativo: salta le righe iniziali che potrebbero contenere

```

```

descrizioni o intestazioni
ff_data_1 = pd.read_csv(file_path, skiprows=3)

# Ipotizzando una lettura riuscita, filtra le voci non date
ff_data_filtered = ff_data_1[ff_data_1['Unnamed: 0'].apply(lambda x:
str(x).isdigit())]
# Rinomina la prima colonna in 'Date'
ff_data_filtered.rename(columns={ff_data_filtered.columns[0]: 'Date'},
inplace=True)
# Cambia il tipo della prima colonna in datetime
ff_data_filtered['Date'] = pd.to_datetime(ff_data_filtered['Date'])
# Query di esempio: Trovare il fattore SMB in una determinata data
smb_factor = ff_data_filtered[ff_data_filtered['Date'] == '20151106']
['SMB'].iloc[0]
# Questo restituisce il valore numerico corretto 0.87

```

L’inclusione di un “unit-test” come ultima riga di questo codice è importante in qualsiasi programma, ma soprattutto per quelli generati da ChatGPT.

Ora, dobbiamo combinare correttamente i dati dei rendimenti giornalieri e i dati dei fattori, utilizzando i primi come variabile “etichetta” o “Y” nel nostro addestramento di regressione e i secondi come variabile “feature” o “X”. Chiedendo a ChatGPT di farlo, otteniamo una risposta che somiglia a quella nel paragrafo “Combinazione dei fattori Fama-French con i rendimenti del giorno successivo” che segue, che contiene alcune modifiche umane.

Combinazione dei fattori Fama-French con i rendimenti del giorno successivo

Abbiamo creato il dataframe che conteneva i fattori Fama-French come feature per i nostri dati di addestramento. Ora dobbiamo combinarli con le etichette, che sono i rendimenti del giorno successivo, per formare il dataframe completo per l’addestramento e il test di un modello di regressione lineare.

```

import pandas as Pd
from sklearn.linear_model import LinearRegression

```

```

# Presupponendo che ff_data_filtered e next_day_returns_data siano i
# tuoi DataFrame caricati
# %% Carica e analizza i rendimenti giornalieri dei titoli
next_day_returns_data = pd.read_csv('CRSP_nextDayReturns.csv')
next_day_returns_data['Date'] =
pd.to_datetime(next_day_returns_data['Date'],
               format='%Y-%m-%d')

next_day_returns_data['Date'] =
pd.to_datetime(next_day_returns_data['Date'],
               format='%Y%m%d')

# %% Carica e analizza i dati Fama-French
file_path = 'F-F_Research_Data_Factors_daily.CSV'
# Primo tentativo: saltare le righe iniziali che potrebbero contenere
# descrizioni o intestazioni
ff_data_1 = pd.read_csv(file_path, skiprows=3)

# Presupponendo una lettura riuscita, filtrare le voci non-date
ff_data_filtered = ff_data_1[ff_data_1['Unnamed: 0'].apply(lambda x:
str(x).isdigit())]
# Rinomina la prima colonna in 'Date'
ff_data_filtered.rename(columns={ff_data_filtered.columns[0]: 'Date'},
inplace=True)
# Cambia il tipo della prima colonna in datetime
ff_data_filtered['Date'] = pd.to_datetime(ff_data_filtered['Date'])
# Esempio di query: Trovare il fattore SMB in una determinata data
smb_factor = ff_data_filtered[ff_data_filtered['Date'] == '20151106']
['SMB'].iloc[0]
# Questo restituisce il valore numerico corretto 0.87
# Unisci i dataset sulla colonna 'Date'
combined_data = pd.merge(ff_data_filtered, next_day_returns_data,
on='Date')
# Unit test
# Funzione per eseguire lo unit test
def test_factors_on_date(data, date, expected_smb, expected_hml):
    # Estrazione della riga per la data data
    data_on_date = data[data['Date'] == date]
    # Verifica se la data esiste nei dati
    if data_on_date.empty:
        return "Date not found in the data."
    # Estrazione dei valori SMB e HML
    smb_value = data_on_date['SMB'].iloc[0]
    hml_value = data_on_date['HML'].iloc[0]
    # Test delle condizioni
    smb_test = abs(smb_value - expected_smb) < 0.01 # Controllo
approssimativo
    hml_test = abs(hml_value - expected_hml) < 0.01 # Controllo
approssimativo
    return {"SMB Test Passed": smb_test,
            "HML Test Passed": hml_test,
            "Actual SMB Value": smb_value,
            "Actual HML Value": hml_value
    }

# Esecuzione del test per il 20151106
test_results = test_factors_on_date(combined_data, '20100423', 0.37,

```

```

0.62)
test_results
# Funzione per eseguire lo unit test per il rendimento a un giorno di
MSFT
def test_msft_return_on_date(data, date, expected_return):
    # Estrazione della riga per la data data
    data_on_date = data[(data['Date'] == date) & (data['Stock_ID']==
'MSFT')]
print(data_on_date)
    # Verifica se la data esiste nei dati
    if data_on_date.empty:
        return "Date not found in the data."
    # Estrazione del valore del rendimento a un giorno di MSFT
    msft_return_value = data_on_date['1_Day_Returns'].iloc[0]

    # Test della condizione
    return_test = abs(msft_return_value - expected_return) < 0.01 #
Controllo approssimativo
    return {"MSFT Return Test Passed": return_test,
            "Actual MSFT 1 Day Return Value": msft_return_value
    }
test_msft_return_on_date(combined_data, '20100423', 0.00508690122)
{'MSFT Return Test Passed': True,
 'Actual MSFT 1 Day Return Value': 0.0035501048894635
}
test_msft_return_on_date(combined_data, '20081021', -0.078)
{'MSFT Return Test Passed': True,
 'Actual MSFT 1 Day Return Value': -0.0777468408652821
}

```

Questo listato ha superato tutti gli unit test.

Capitolo 3

Il machine learning nella gestione degli investimenti

Forse non ve ne siete resi conto, ma probabilmente state applicando ormai da molti anni il machine learning alla gestione degli investimenti. Forse avete usato la regressione lineare per trovare il beta di un portafoglio rispetto a un indice di mercato? Forse avete provato l'analisi delle componenti principali (PCA – *Principal Component Analysis*) come alternativa al modello fattoriale di Fama-French descritto nel Capitolo 2? Anche se possono sembrare antiquati, questi sono tutti esempi di machine learning. D'altra parte, potreste pensare che il machine learning nella gestione degli investimenti consista solo nell'usare un robot per prevedere i rendimenti azionari. Come spiegheremo nel paragrafo sull'apprendimento supervisionato, questa potrebbe essere l'applicazione di maggior successo dell'intelligenza artificiale nella finanza.

In questo capitolo, faremo un tour guidato sull'uso del machine learning nella gestione degli investimenti, per concludere scoprendo come l'intelligenza artificiale generativa può aiutarci oltre gli usi no-code che abbiamo esaminato nei due capitoli precedenti (lascieremo le sfumature del Deep Learning e dell'intelligenza artificiale generativa al resto del libro). Questo tour includerà quanto segue.

- *Apprendimento non supervisionato*: clustering con applicazione all'allocazione del capitale tramite HRP (*Hierarchical Risk Parity*), PCA (*Principal Component Analysis*), cMDA (*Cluster-based Feature Selection*) e HMM (*Hidden Markov Model*).
- *Apprendimento supervisionato*: regressioni lineari e logistiche con regolarizzazioni, modelli basati su alberi e reti neurali. Varie metriche di performance, ottimizzazione degli iperparametri e validazione (incrociata). Applicazioni alla CAI (*Corrective AI*) e alla CPO (*Conditional Parameter/Portfolio Optimization*), ma enfaticamente non alla previsione dei rendimenti delle attività.
- Commenti sulle reti neurali ricorrenti e sul *Deep Reinforcement Learning*.
- Ingegneria di dati e delle feature; classificazione e selezione dell'importanza delle feature; applicazione della intelligenza artificiale generativa all'ingegneria delle feature.

L'intenzione di questo capitolo è quella di evidenziare alcuni dei concetti più importanti del machine learning che sono anche rilevanti per i gestori di investimenti, senza approfondire troppo i dettagli. Potete trovare libri molto migliori, nei quali troverete tutti i dettagli. Per i concetti fondamentali del machine learning, vedi Murphy (2012) e Bishop (2006). Per il machine learning per i gestori di investimenti, vedi López de Prado (2018, 2020). Per le reti neurali, vedi Goodfellow et al. (2016).

Seguiremo questa convenzione: $\mathbf{X}$ è una matrice, $\mathbf{x}$ è un vettore, x è uno scalare. In particolare, $\mathbf{x}$ è un vettore di feature, y e $\mathbf{y}$ sono rispettivamente un'etichetta scalare e vettoriale inferita (prevista), mentre $\hat{y}$ e $\hat{\mathbf{y}}$ sono rispettivamente l'etichetta scalare e vettoriale effettiva. $\mathbf{X}$ spesso denota una matrice di dati o di progetto con n righe di campioni e m colonne di feature. Spiegheremo che cosa solo

le feature, le etichette e le matrici di progetto nel paragrafo sull'apprendimento supervisionato.

Apprendimento non supervisionato

A volte ci aspettiamo che una macchina faccia cose che anche noi sapremmo fare, magari col senno di poi. Decidere se acquistare o vendere un'attività e mantenerla per un giorno è un esempio, perché con i dati storici sapremmo immediatamente se è stata una buona decisione o meno. Altre volte, vogliamo che una macchina faccia cose di cui non possiamo dire se la soluzione fornita sia quella giusta o la migliore disponibile. Decidere quali azioni dovrebbero essere raggruppate in cluster per l'ottimizzazione del portafoglio tramite HRP (*Hierarchical Risk Parity*) è un esempio. Non abbiamo dati storici che mostrino quali siano i cluster ottimali, né un metodo analitico per dimostrare che un dato cluster è ottimale. Il problema della ricerca di questi cluster, storici o attuali, è esattamente quello cui vogliamo applicare il machine learning non supervisionato. Allo stesso modo, se vogliamo scoprire quali sono i fattori comuni che guidano i rendimenti delle attività basandoci solo sui dati storici dei rendimenti tramite la PCA (*Principal Component Analysis*), non possiamo usare alcuna variabile osservabile, eccetto i rendimenti stessi. Infine, potremmo voler usare un HMM (*Hidden Markov Model*) per modellare i cambi di regime nel mercato finanziario quando costruiamo modelli probabilistici dei rendimenti delle attività. Come suggerisce l'aggettivo *Hidden*, nascosto, gli stati discreti di Markov che ipotizziamo non sono osservabili, e pertanto dobbiamo inferirli dalle “emissioni” – come i rendimenti di un'azione – per stimare i parametri. Tutti questi compiti rientrano

nella categoria dell'apprendimento non supervisionato, che riguarda la cattura di modelli da dati non etichettati. Quindi ora discuteremo ciascuna di queste tecniche di apprendimento non supervisionato, per illustrare il concetto generale.

HRP (Hierarchical Risk Parity)

L'ottimizzazione del portafoglio (quanto denaro dovremmo allocare a ciascuna attività che compone un portafoglio) è un classico problema finanziario. Harry Markowitz ha vinto un Premio Nobel per l'Economia per aver inventato nel 1952 l'approccio di ottimizzazione media-varianza (vedi il paragrafo "Traduzione di listati Matlab in codice Python" nel Capitolo 1 sul trovare la frontiera efficiente di un portafoglio). Tuttavia, è risaputo che l'ottimizzazione media-varianza funziona solo col senno di poi: di solito non produce un portafoglio ottimale per il futuro. Questo perché il portafoglio ottimale media-varianza è altamente sensibile ai rendimenti in input. Ang (2014) lo ha illustrato bene in un paragrafo dal titolo decisamente efficace: *Garbage In, Garbage Out* (p. 95). Un portafoglio allocato su cinque indici azionari di paesi sviluppati cambierà la sua allocazione sull'indice USA dal -9% al 41% se cambiamo la stima del suo rendimento medio annuo dal 10.3% al 13%. Davvero pensate di potere prevedere l'indice azionario USA entro il 2,7% per il prossimo anno? Quindi, molti gestori di investimenti si disperano (inclusi i potenti Bridgewater Associates) e adottano invece un approccio apparentemente semplicistico: allocare semplicemente un "rischio" uguale a ciascuna attività, con il rischio misurato dalla volatilità dei rendimenti. Questo è chiamato metodo di allocazione *risk parity*. Se l'attività A ha un rischio del 20% e l'attività B ha un rischio del 40%, allora allocheremo il capitale ad A:B nel rapporto 2:1 (cioè, inversamente proporzionale alla volatilità). Questo approccio

semplificistico, pur non esente da problemi, in realtà funziona meglio della sofisticata ottimizzazione media-varianza. La risk parity ipotizza che tutti i rendimenti delle attività siano statisticamente indipendenti (e quindi non correlati). Ma se le due attività sono altamente correlate (pensate alle azioni F vs. GM) e hanno una volatilità simile, la risk parity allocherà a esse su una base pari approssimativamente a 1:1, ma insieme le due attività contribuiscono al portafoglio $\sqrt{2}$ volte il rischio di due attività non correlate. La *Hierarchical Risk Parity* (López de Prado, 2020) è un metodo per affrontare questo problema. L'idea di base è quella di raggruppare in cluster le attività all'interno di un portafoglio in base alle loro correlazioni. Le attività altamente correlate finiranno quindi nello stesso cluster. La struttura dell'algoritmo è la seguente.

1. Applicare il clustering gerarchico (Hastie et al., 2009) alla matrice di covarianza per trovare cluster di attività altamente correlate. Ci sono varie tecniche per decidere il numero ottimale di cluster, ma questo è di importanza secondaria. Potreste considerare questo numero come un iperparametro da ottimizzare, un concetto che discuteremo nel paragrafo dedicato all'apprendimento supervisionato.
2. Applicare un metodo di ottimizzazione del portafoglio (il preferito) alle attività all'interno di ciascun cluster, per trovare le allocazioni intra-cluster. Poiché tutte le attività presenti nel cluster sono simili, possiamo semplicemente applicare la consueta allocazione risk parity, che in questo caso non eccederà nel pesare il rischio di alcuna attività. Notate che la correlazione ~ 1 non significa che le volatilità delle attività siano le stesse all'interno del cluster. Chiamate questa allocazione $w_c(s)$ per l'attività s nel cluster c , dove
$$\sum_s w(s) = 1.$$

3. Applicare questo metodo di ottimizzazione del portafoglio al rendimento di ciascun cluster, per trovare l'allocazione del capitale tra i diversi cluster. Poiché le correlazioni tra questi cluster dovrebbero essere molto basse, la matrice di covarianza dei rendimenti dei cluster è quasi diagonale. Quindi l'ottimizzazione media-varianza applicata qui non soffrirà dell'instabilità e della sensibilità agli input. In effetti, se $W(c)$ è il peso dato al cluster c , con $\sum_c W(c) = 1$, e m_c e σ_c sono la media e la deviazione standard dei rendimenti del cluster c , allora $W(c) \sim m_c / \sigma_c^2$, che molti lettori riconosceranno come l'allocazione basata sulla formula di Kelly (Chan, 2017).
4. L'allocazione finale alle attività presenti nel cluster c è $\omega(s) = W(c) * w_c(s)$. Notate che, se tutti i cluster hanno rendimenti e volatilità simili, e tutte le attività all'interno di un cluster c hanno volatilità simili, allora

$$\omega(s) = \frac{1}{n(c) \cdot N}$$

dove $n(c)$ è la dimensione del cluster c , e N è il numero di cluster. Questo, ovviamente, è molto diverso da un'allocazione ingenua, uguale, che darebbe

$$\omega(s) = \frac{1}{\sum_c n(c)}.$$

Esercizio #3

Chiedete a ChatGPT di tracciare il dendrogramma di queste cinque azioni in base alle correlazioni dei loro rendimenti giornalieri negli ultimi cinque anni: F, GM, TSLA, BAC, JPM. Fermatevi a un massimo di tre cluster. Il risultato somiglia a quello rappresentato nella Figura 3.1?

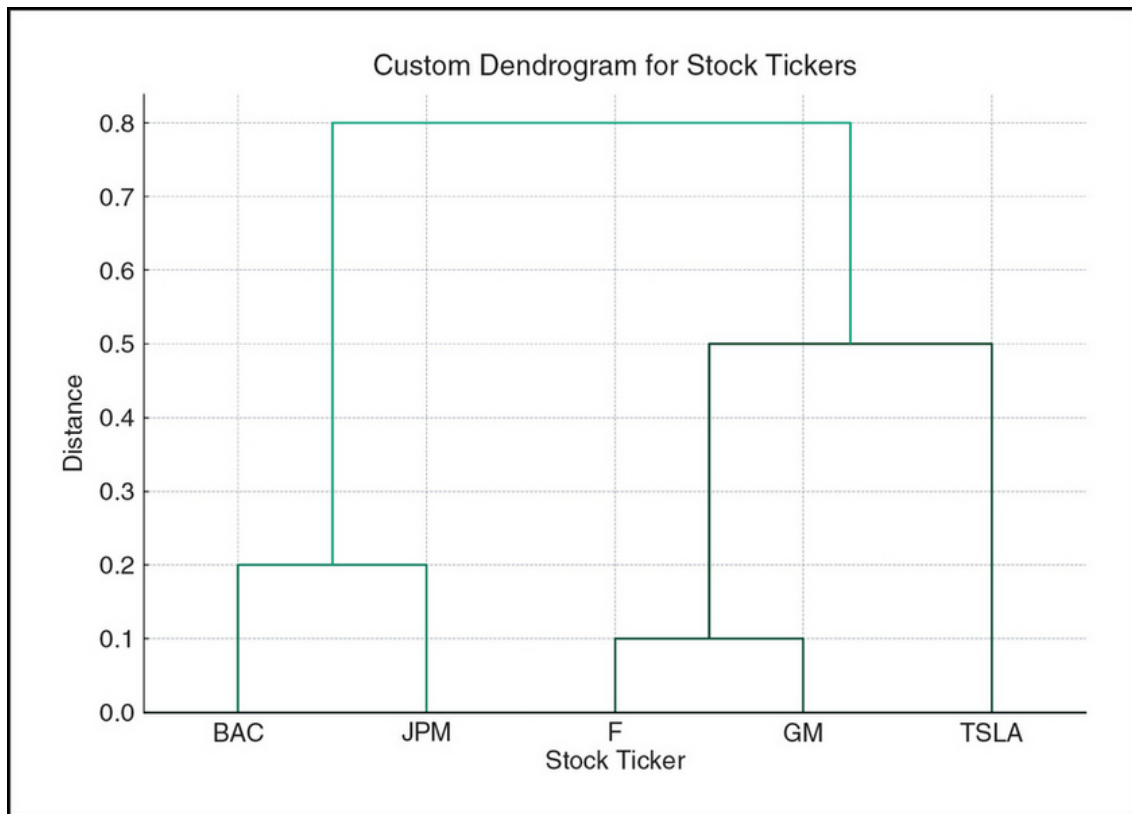


Figura 3.1 Dendrogramma di cinque azioni basato sulle correlazioni dei loro rendimenti giornalieri negli ultimi cinque anni.

PCA (Principal Component Analysis)

Nel Capitolo 2 abbiamo visto come possiamo usare i fattori Fama-French per prevedere i rendimenti delle azioni. Questi fattori sono osservabili: per esempio, chiunque può calcolare il fattore SMB raccogliendo i rendimenti e le capitalizzazioni di mercato delle azioni. La PCA è un metodo per trovare fattori che sono “statistici” – interamente costruiti solo dai dati dei rendimenti delle azioni – e poi usare questi fattori per analizzare o prevedere questi stessi rendimenti. Quindi è un altro esempio di apprendimento non supervisionato. I fattori PCA (più precisamente, le esposizioni o i carichi fattoriali) vengono creati trovando gli autovettori principali

(classificati in base ai loro autovalori) della matrice di covarianza dei rendimenti, o trovando i “vettori singolari destri” principali applicando la decomposizione ai valori singolari (SVD – *Singular Value Decomposition*) direttamente alla matrice dei dati dei rendimenti (per esempio, righe di vettori di rendimenti giornalieri di varie azioni). Per maggiori dettagli, vedi Chan (2021), Esempio 7.1. Qui, per “principali” intendiamo gli autovettori associati agli autovalori più grandi, o i vettori singolari destri associati ai valori singolari più grandi. Il valore di cutoff di ciò che viene considerato principale è un dettaglio che potete trovare discusso in López de Prado (2018) e può anche essere considerato un iperparametro da ottimizzare quando i risultati verranno poi usati a valle in un processo di investimento.

Oltre al loro uso come fattori statistici che guidano i rendimenti delle attività, la PCA o più in generale la SVD possono spesso essere usate per il denoising (togliere il “rumore”) o per la selezione delle feature nelle applicazioni di machine learning. Nello specifico, se abbiamo un insieme di feature di input che possono essere usate nell’apprendimento supervisionato (vedi il prossimo paragrafo), e queste feature sono altamente correlate, possiamo applicare la PCA per trovare le “feature ortogonali” principali che sono non correlate e significative. Significative significa che queste feature sono classificate in alto come importanza statistica rispetto a tutte le altre feature (il che non significa necessariamente che siano statisticamente significative in senso assoluto). Per maggiori dettagli, vedi López de Prado (2018). Nel Capitolo 6, discuteremo ulteriori approcci di apprendimento non supervisionato, come la PPCA (*Probabilistic Principal Component Analysis*), che è la versione probabilistica della PCA, così come l’analisi fattoriale (FA –

Factorial Analysis), che può essere considerata un'estensione della PPCA.

Esercizio #4

Chiedete a ChatGPT di generare due serie casuali con 100 campioni ciascuna, e con matrice di covarianza

$$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 2 \end{bmatrix}$$

Create un grafico a dispersione (*scatter plot*) e sovrapponetevi gli autovettori basati sulla PCA. Somiglia a quello rappresentato nella Figura 3.2?

Parlando di selezione delle feature e di classificazione dell'importanza, c'è un altro modo per sfruttare l'apprendimento non supervisionato.

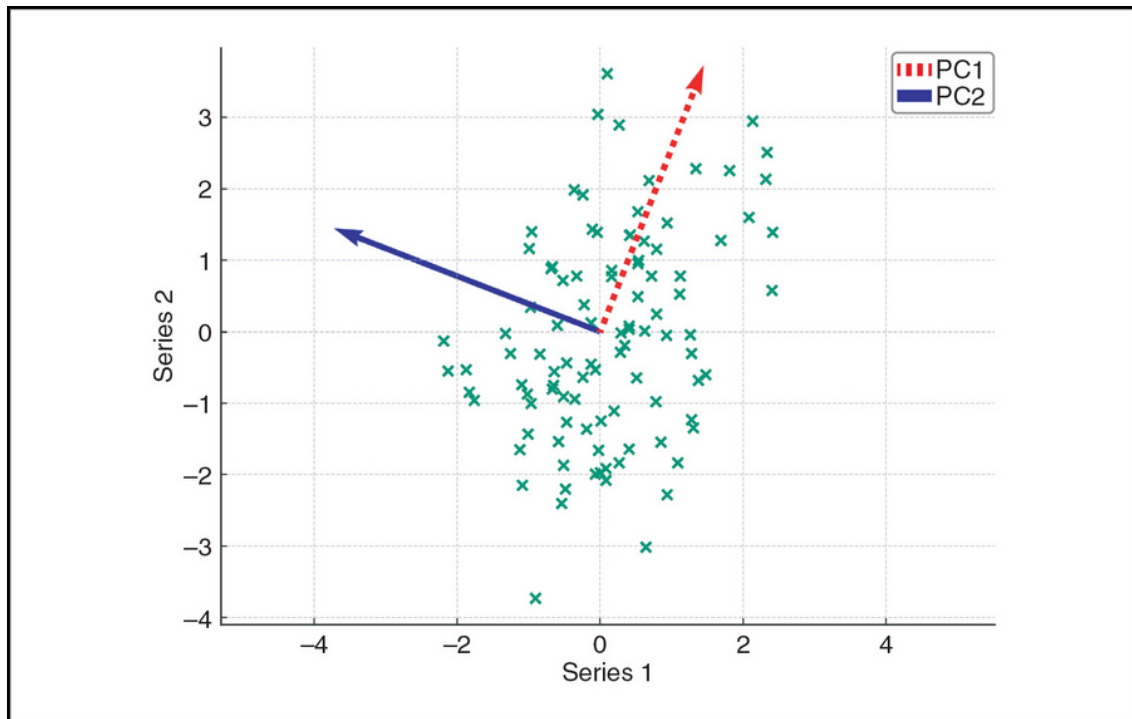


Figura 3.2 Grafico dei componenti principali di due serie correlate.

cMDA (Cluster-based Feature Selection)

Come discuteremo nel contesto dell'apprendimento supervisionato, la classificazione e selezione dell'importanza delle feature è un metodo per regolarizzare un processo di machine learning in modo che non apprenda da feature di input statisticamente insignificanti e rumorose. La classificazione dell'importanza delle feature migliora anche l'interpretabilità di un modello di machine learning, poiché l'utente può comprendere più facilmente come il modello ottiene il suo potere predittivo da un piccolo numero di feature importanti invece di un gran numero di input possibilmente irrilevanti. Il fatto che una feature sia significativa (importante) dipende, ovviamente, dall'etichetta (variabile target) durante la fase di addestramento nell'apprendimento supervisionato. Dettaglieremo alcuni metodi di selezione delle feature quando arriveremo all'apprendimento supervisionato, ma molti di quei metodi hanno un problema. Se molte delle feature sono correlate o, in generale, altamente dipendenti statisticamente tra loro, questi metodi di selezione spesso selezionano casualmente una ma non le altre come feature importante. Due feature molto simili possono ricevere due ranghi di importanza molto differenti. Questo è arbitrario e non aiuta l'interpretabilità, poiché un cluster di feature simili spesso racconta una storia migliore di una singola feature. López de Prado (2020) ha descritto un metodo di selezione delle feature basato su cluster per affrontare questo problema. In Man and Chan (2021b), abbiamo applicato il clustering gerarchico per trovare questi cluster e presentato prove che il fatto di applicare la classificazione e selezione dell'importanza delle feature basata su cluster invece che sulle singole feature produce risultati predittivi migliori e una migliore interpretabilità. Da quel documento, abbiamo raggruppato alcune feature utilizzate per prevedere i rendimenti dello SPX e trovato due grandi cluster che

possono essere facilmente interpretati come feature fondamentali vs tecniche, come mostrato nella Tabella 3.1.

Tabella 3.1 Risultati della classificazione dell'importanza delle feature basata su cluster in base alla loro importanza nella previsione dei rendimenti dello SPX.

Tipo	Punteggi del cluster	Feature
Fondamentale	0.667	d/p, d/y, e/p, b/m, ntis, tbl, lty, dfy, dfr, infl
Tecnico	0.333	d/e, svar, ltr, tms

HMM (Hidden Markov Model)

L'*Hidden Markov Model* è una tecnica molto in uso (Chan, 2017) tra gli investitori, perché la suggestione tra “stati nascosti” e regimi di mercato è davvero seducente. Chi non vorrebbe modellare le transizioni tra i regimi di mercato come transizioni su una catena di Markov? Il problema è che, se i regimi di mercato fossero davvero osservabili e ben definiti, non avremmo bisogno di un “modello nascosto di Markov”. Per esempio, potete definire l’inizio di un regime di mercato “bear” quando il prezzo scende del 20% o più al di sotto del massimo storico (*high watermark*), e la fine di questo mercato bear quando il prezzo sale del 20% o più al di sopra del prezzo minimo dall’inizio del mercato bear (la fine di un mercato bear è l’inizio di un mercato bull, e viceversa). Tuttavia, prevedere questi regimi sarebbe un problema di apprendimento supervisionato, non un problema da HMM. Se non possiamo osservare i regimi di mercato, tutto ciò che fa l’HMM consiste nell’introdurre più parametri in un modello probabilistico dei rendimenti dove, per esempio, una semplice miscela statica di gaussiane, nota come GMM

(*Gaussian Mixture Model*), non si adatta abbastanza bene ai dati. La corrispondenza con regimi di mercato nascosti è più un'aspirazione o, più schiettamente, solo fittizia. Naturalmente, più parametri avete, migliore è l'adattamento per il set di addestramento. Se due stati di Markov nascosti non si adattano abbastanza bene ai rendimenti osservati, perché non usarne duecento? La domanda è: l'HMM prevede bene, di per sé, i rendimenti? Qui le prove sono deboli. Chan (2017) ha scoperto che fallisce miseramente per SPY, ma Catello et al. (2023) hanno scoperto che ha funzionato su alcune azioni per alcuni periodi di tempo. Il giudizio finale è ancora in sospeso. Nel Capitolo 6, approfondiremo il GMM e forniremo un esempio illustrativo di rilevamento del regime di mercato.

Apprendimento supervisionato

La maggior parte delle applicazioni del machine learning negli investimenti rientra nella categoria dell'apprendimento supervisionato. Prevedere i rendimenti azionari utilizzando indicatori tecnici? Trovare il beta di un portafoglio rispetto a un indice di mercato? Prevedere se una decisione di investimento sarà redditizia? Tutti questi sono esempi familiari di apprendimento supervisionato.

Apprendimento supervisionato è quando conosciamo la verità, almeno storicamente, e vogliamo addestrare un modello di machine learning con quella conoscenza, in modo che possa fare buone inferenze (previsioni) su dati che non ha mai visto prima. La verità può essere chiamata in molti modi: etichetta (label), target, variata, variabile dipendente o, semplicemente y . Per addestrare il modello di machine learning, abbiamo anche bisogno di caratteristiche (feature), ovvero di predittori, regressori, covariate, fattori, variabili indipendenti o, semplicemente, x . Quando l'etichetta è discreta (per esempio, il fatto che un'azione salga o scenda), il problema è detto di

classificazione. Quando l'etichetta è continua (per esempio, il fatto che un'azione avrà un rendimento dell'11,2% il prossimo anno), il problema è detto di *regressione*.

Abbiamo detto del prevedere i rendimenti azionari come se quello fosse il miglior caso d'uso dell'apprendimento supervisionato; in realtà, è il peggiore. Quando usiamo dei dati storici per prevedere qualcosa, perché la previsione abbia successo è imperativo che ciò che cerchiamo di prevedere non cambi il comportamento della variabile target. Per esempio, possiamo anche cercare di prevedere quando avverrà il prossimo terremoto, ma ciò non cambierà le cose. Ma quando si costruisce un modello di previsione azionaria basato sull'apprendimento da dati passati, il nostro non sarà l'unico modello predittivo, poiché i dati sono pubblici. Le attività di arbitraggio faranno sì che la precisione della previsione dei rendimenti diminuisca gradualmente: questo è noto nel settore come decadimento dell'alfa (*alpha decay*). Il fatto di agire in base alla previsione, in effetti, cambia il comportamento della variabile che si sta cercando di prevedere. Questo effetto può essere considerato un ciclo di feedback (le nostre azioni cambiano il sistema osservato e previsto) e non è considerato nella formulazione standard dei problemi di apprendimento supervisionato, mentre invece è l'oggetto dei sistemi di controllo o dell'apprendimento per rinforzo (RL, *Reinforcement Learning*), che verrà descritto più avanti in questo stesso capitolo. Ci sono casi d'uso molto più efficaci del machine learning nella gestione patrimoniale: *Corrective AI* (Belov et al., 2023) e *Conditional Portfolio Optimization* (Chan et al., 2023). Il primo applica la classificazione e il secondo applica la regressione. Entrambi sono più efficaci della previsione diretta dei rendimenti delle attività, perché ciò che cercano di prevedere non è soggetto ad arbitraggio. Vedremo il perché.

Il nome CAI (*Corrective AI*) è il nome dato al “metalabeling” da López de Prado (2018). Significa che trattiamo un sistema di investimento quantitativo convenzionale come un “sistema esperto”, con regole di investimento definite da esseri umani basate sulle loro intuizioni di mercato/matematiche e con una quantità di input limitata. Quindi applichiamo il machine learning come “livello correttivo” utilizzando l’apprendimento supervisionato per determinare la “probabilità di profitto” (PoP, *Probability of Profit*) di ogni decisione di investimento o di ogni posizione in un certo periodo di tempo. Se la PoP è superiore a una soglia, accettiamo la decisione/posizione. In caso contrario, la rifiutiamo e rimaniamo “piatti” o “privi di rischio”. Le etichette per la CAI sono *redditizio* (la classe positiva, con etichetta 1) o non redditizio (la classe negativa, con etichetta 0). Per un esempio di applicazione trading della CAI, vedi Belov et al. (2023). Questo uso della previsione tramite machine learning non è soggetto ad arbitraggio, perché nessun altro dovrebbe conoscere la prossima previsione di rendimenti delle attività del vostro sistema esperto. Quindi, anche se tutti possono prevedere con un’elevata accuratezza se il vostro sistema esperto sarà redditizio, non possono intraprendere alcuna azione che possa cambiare il risultato.

La CPO (*Conditional Portfolio Optimization*) è stata inventata da Predictnow.ai. Utilizza il machine learning per prevedere le performance (per esempio, i rendimenti, l’indice di Sharpe, il massimo drawdown) di un portafoglio sotto varie allocazioni di attività, e classificare tali performance in modo da poter scegliere la migliore allocazione di attività, con la performance prevista più alta. È un metodo innovativo di ottimizzazione del portafoglio che può utilizzare come input i big data per adattarsi ai cambiamenti di regime di mercato. Per un’esposizione più completa della CPO e dei

suoi molti successi, vedi Chan et al. (2023). Anche la CPO non è soggetta ad arbitraggio, perché pochi investitori hanno lo stesso identico obiettivo di ottimizzazione sullo stesso identico portafoglio. L'efficacia dell'ottimizzazione non dipende nemmeno dalla previsione accurata dei rendimenti dei singoli asset, né dalla previsione accurata dei rendimenti del portafoglio, rendendo questo metodo altamente fault-tolerant.

Esistono numerose tecniche di apprendimento supervisionato, che vanno dalle più semplici e note, alle più complesse e arcane. La maggior parte di queste tecniche può essere applicata sia alla classificazione sia alla regressione, con piccoli aggiustamenti. Molte di queste tecniche possono essere ottimizzate regolando gli iperparametri. Alcune tecniche per migliorare le prestazioni di un sistema di apprendimento supervisionato su dati mai visti prima (fuori campione, di test) rientrano in una categoria chiamata *regolarizzazione*, che è un insieme ricco e vario di metodi. Qui esamineremo le varie tecniche di apprendimento supervisionato, così come i metodi di ottimizzazione degli iperparametri e le regolarizzazioni.

Regressioni lineari e logistiche

Confidiamo nel fatto che chiunque abbia studiato finanza o statistica abbia visto l'adattamento della regressione $y = \beta \cdot X$, dove $\cdot$ è il prodotto scalare (*dot product*) di β (il vettore dei coefficienti di regressione) e X (il vettore delle feature) che restituisce l'etichetta scalare y . Questo funziona per inferire (prevedere) un'etichetta alla volta. Per addestrare questo modello, cioè per trovare i valori migliori di β che permettano di fare inferenze (previsioni) accurate, abbiamo bisogno di molti dati storici con coppie (feature, etichetta) di (X, y) che formano la matrice dei dati X (dove ogni riga

rappresenta un dato campione con molte colonne di diverse feature, aumentata con una colonna di tutti 1 per tenere conto di un possibile offset costante) e il vettore delle etichette y (un vettore colonna delle corrispondenti etichette univariate). X è chiamata anche “matrice di disegno” (*design matrix*). Possiamo usare il metodo dei minimi quadrati ordinari (OLS, *Ordinary Least Squares*) per ottenere $\beta = (X^T X)^{-1} X^T y$. Vedi Murphy (2012).

Esercizio #5

Chiedete a ChatGPT di generare una matrice di disegno ipotetica di quattro campioni con tre feature. Denominate queste feature “Rendimento Oil 30d”, “Crescita PIL” e “HML”. Tutte le feature dovrebbero essere distribuite normalmente con media e deviazione standard dello 0.5%. La matrice somiglia alla Tabella 3.2?

Tabella 3.2 Matrice di disegno ipotetica per tre feature con quattro campioni.

Campione	Rendimento Oil 30d	Crescita PIL	HML
0	0.013820	0.007001	0.009894
1	0.016204	0.014338	0.000114
2	0.009750	0.004243	0.004484
3	0.007053	0.005720	0.012271

Un altro modo per ottimizzare i parametri del modello di regressione lineare consiste nel massimizzare la log-verosimiglianza (*log likelihood*) dei dati di addestramento tramite la stima di massima verosimiglianza (MLE, *Maximum Likelihood Estimation*). In altre parole, massimizziamo la log-verosimiglianza congiunta $\sum_i \log(P(y_i))$ di ottenere un insieme di osservazioni y_i modificando i valori di β , dati i valori di X_i . Nota: qui usiamo l’indice i in quanto è l’indice di riga convenzionale per una matrice di disegno. Si evince che, se ipotizziamo che y abbia una distribuzione gaussiana,

massimizzare la log-verosimiglianza equivale a minimizzare l'errore quadratico medio (MSE, *Mean Squared Error*) delle etichette inferite rispetto alle etichette effettive sul set di addestramento (Goodfellow et al., 2016, paragrafo 5.5.1). La soluzione è la stessa dell'adattamento con il metodo OLS.

Ma nel trading, spesso, vogliamo solo prevedere se l'attività salirà o scenderà il giorno successivo, o se una strategia di trading sarà redditizia il giorno successivo. In altre parole, abbiamo un problema di classificazione, non di regressione. Quindi, spesso y è una variabile discreta o anche binaria, 0 o 1. In questo caso, non vogliamo solo prevedere se y sarà 0 o 1, vogliamo calcolare la probabilità $P(y)$ di ciascuno. Come la regressione lineare, ipotizziamo che $P(y)$ sia una funzione lineare $\beta \cdot \mathbf{x}$ delle feature $\mathbf{x}$. Ma non è così semplice qui, perché $P(y)$ può assumere solo valori tra 0 e 1, e una funzione lineare dà valori tra $-\infty$ e ∞ . Per sistemare le cose, possiamo far passare questa funzione lineare prima attraverso una funzione sigmoide $P(y) = 1 / [1 + \exp(-\beta \cdot \mathbf{x})]$.

Esercizio #6

Chiedete a ChatGPT di tracciare una funzione sigmoide. Il grafico somiglia a quello rappresentato nella Figura 3.3?

Ora abbiamo un modello probabilistico parametrico quasi-lineare della variabile binaria. Per trovare il miglior β basato sulle osservazioni y , possiamo applicare a questo modello probabilistico nuovamente la MLE, ma non esiste una soluzione in forma chiusa come l'adattamento OLS per la regressione lineare. È necessario un metodo di ottimizzazione numerica come la discesa del gradiente (*gradient descent*). Questo ci dà un assaggio di ciò che verrà: gran parte dell'addestramento delle reti neurali e del deep learning si basa sulla discesa del gradiente, sebbene nelle reti neurali i coefficienti di regressione β siano chiamati "pesi" $\mathbf{W}$. Ne discuteremo presto.

Questo schema è chiamato “regressione logistica”, nonostante sia un problema di classificazione. E la funzione sigmoide $1/[1 + \exp(\dots)]$ qui sopra è chiamata funzione logistica.

Una volta addestrato $P(y|\mathbf{x})$, possiamo impostare una soglia per decidere quando dovremmo inferire/prevedere $y = 0$ o 1 per un dato input $\mathbf{x}$. La scelta convenzionale è 0.5. Ma immaginando di volere che la previsione dei rendimenti generi un bias long per il modello di trading, si può sempre impostare la soglia a meno di 0.5. Quello della migliore soglia non è un problema di machine learning, ma di costruzione della strategia di trading. Si può eseguire il backtest e ottimizzare la migliore soglia per ottenere il CAGR (*Compound Annual Growth Rate*) più alto, l'indice di Sharpe migliore o il massimo drawdown più basso e così via. In altre parole, è un iperparametro da ottimizzare in base alle metriche finanziarie.

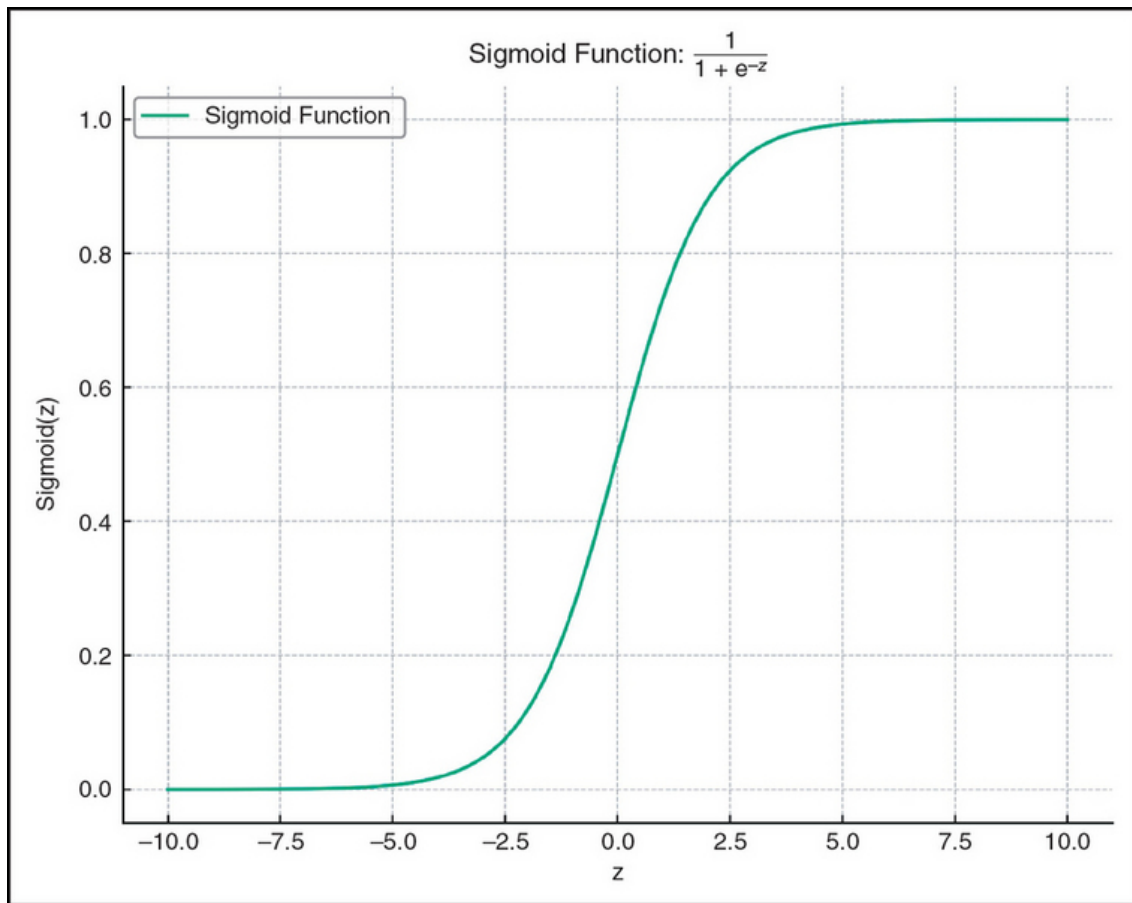


Figura 3.3 Grafico di una funzione sigmoide.

Regolarizzazioni L1 e L2

In teoria, le regressioni lineari e logistiche sembrano semplici da addestrare (abbiamo addirittura una soluzione in forma chiusa per la regressione lineare). Ma in pratica, se abbiamo un gran numero di feature e non sappiamo se sono significative o ridondanti (cioè altamente correlate, nel qual caso si parla di *multicollinearità*), scopriremo che β può essere molto instabile (sensibile a piccoli cambiamenti in X) e produrrà grandi errori. Inoltre, la matrice $X^T X$ può essere singolare e non invertibile e non può generare alcun β . Inoltre, il fatto di avere troppe feature di input porta al problema

dell'overfitting nel machine learning: il modello addestrato potrebbe comportarsi molto bene sui dati di addestramento, ma generalizzare molto male su dati di test che non ha mai visto. Tutti questi problemi richiedono procedure chiamate “regolarizzazione” o “selezione delle feature”. La prima riduce la complessità del modello riducendo la “dimensione” o il numero dei parametri del modello stesso; la seconda riduce in particolare il numero delle feature. Alcune tecniche di regolarizzazione potrebbero anche fare una selezione automatica delle feature, come vedremo più avanti. Nel contesto delle regressioni lineari o logistiche, due note tecniche di regolarizzazione e selezione delle feature sono le regolarizzazioni L1 e L2, due modi diversi per “misurare” o quantificare la complessità del modello.

La regolarizzazione L1 consiste semplicemente nell'aggiungere una penalità, come $-\lambda \sum_j |\beta_j|$, alla funzione di log-verosimiglianza da massimizzare. Nota: qui usiamo l'indice j in quanto è l'indice di colonna convenzionale per una matrice di disegno che enumera le singole feature. Potreste riconoscere $\sum_j |\beta_j|$ come la norma L1 per un vettore β . Poi, $\lambda > 0$ è un iperparametro di penalità regolabile, che potete ottimizzare in un “set di validazione” (parleremo più avanti dell'ottimizzazione degli iperparametri e dei set di validazione). Il punto è che questo termine di penalità diminuirà la log-verosimiglianza se si hanno troppi β_j diversi da zero, e quindi costringerà il programma di addestramento a ridurre a un valore prossimo a zero il β di alcune feature insignificanti o correlate, applicando una selezione automatica delle feature.

Allo stesso modo, la regolarizzazione L2 consiste nell'aggiungere un termine di penalità come $-\lambda \sum_j \beta_j^2$ alla funzione di log-verosimiglianza da massimizzare. Potreste riconoscere $\sum_j \beta_j^2$ come la norma L2 per un vettore β . Questo termine di penalità ha un effetto simile a $\sum_j |\beta_j|$. Ma qual è la regolarizzazione migliore? L1 o L2? I

ricercatori hanno scoperto che la regolarizzazione L1 spesso produce più β_j a zero, e quindi il modello risultante è più sparso e meglio regolarizzato. L1 è quindi più una tecnica di selezione delle feature che di regolarizzazione. Murphy (2012, Figura 13.3) lo spiega molto bene: il vincolo $\sum_j |\beta_j| \leq B$ forma un'area a forma di rombo B che ha molti vertici dove un β_j si annulla, e la superficie di contorno dove la funzione di log-verosimiglianza è una costante e che cerchiamo di massimizzare può soddisfare più facilmente il vincolo, toccando uno di questi vertici. Abbiamo riprodotto l'effetto nella Figura 3.4.

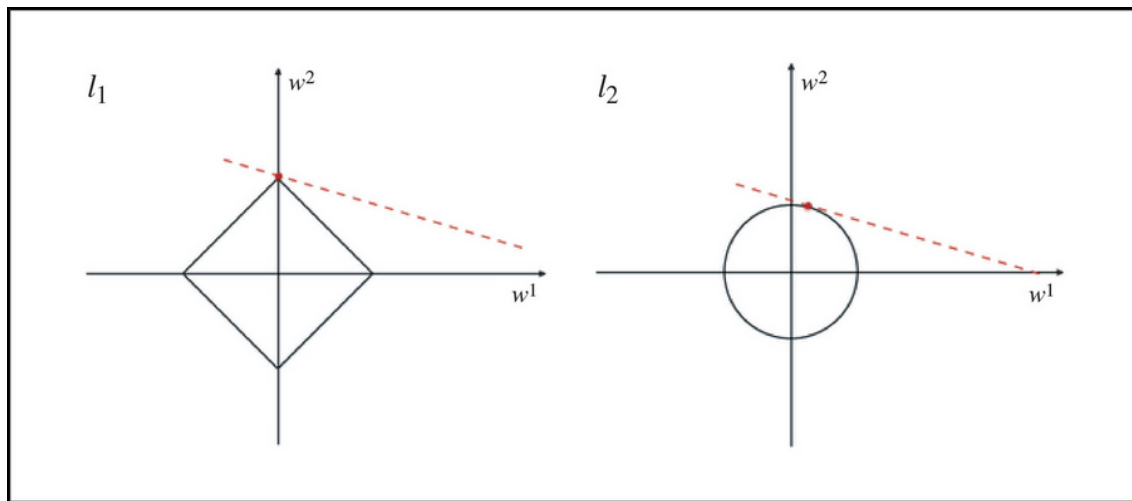


Figura 3.4 Illustrazione del motivo per cui la regolarizzazione L1 può più facilmente portare a pesi zero. Le linee nere indicano dove i vincoli sono soddisfatti (un rombo per L1, un cerchio per L2). Le linee rosse tratteggiate rappresentano le superfici di verosimiglianza costante (i contorni), che cerchiamo di massimizzare spostandole verso il basso. I punti in cui le superfici di verosimiglianza toccano i vincoli rappresentano i pesi ottimizzati. Si può vedere che L1 dà $w^1 = 0$ ma L2 no. Credit: Mxwsn – Opera propria, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=45672058>.

Le linee tratteggiate nella Figura 3.4 indicano il contorno di log-verosimiglianza costante, mentre il rombo e il cerchio e la loro area

interna indicano rispettivamente i vincoli L1 e L2. Il contorno probabilmente intersecherà il rombo in un vertice con β_j (o w^j nella Figura 3.4) nullo.

Le regolarizzazioni L1 e L2 sono chiamate anche, rispettivamente, *regolarizzazione Lasso* e *regolarizzazione ridge*. Si possono anche applicare contemporaneamente, con un termine di penalità $-\lambda_1 \sum_j |\beta_j| - \lambda_2 \sum_j \beta_j^2$, nel qual caso si parla di regolarizzazione *Elastic Net*. Ernie trova che questi nomi siano terribilmente difficili da ricordare e preferisce usare L1, L2 e L1+L2.

Proprio come ottimizzare i parametri di una regressione logistica utilizzando la discesa del gradiente è una tecnica generale che si estende all'ottimizzazione di una rete neurale, anche le regolarizzazioni L1 e L2 durante l'addestramento sono tecniche generali per regolarizzare le reti neurali. Tuttavia, per reti neurali molto profonde, nella pratica si applicano tecniche di regolarizzazione più specifiche.

Ottimizzazione degli iperparametri, validazione e cross-validazione

Ogni modello di machine learning ha i suoi parametri e iperparametri. Nei modelli di regressione lineare e logistica che abbiamo visto, β contiene i parametri, e se si applicano regolarizzazioni L1 o L2 o entrambe, allora λ_1 e λ_2 sono gli iperparametri. Quando più avanti parleremo di algoritmi di apprendimento più complessi, come le foreste casuali, alberi con boosting del gradiente o reti neurali, troverete un insieme più ampio di iperparametri. Potete perfino considerare come iperparametri la lunghezza del set di addestramento o anche quale algoritmo di machine learning applicare. Alcuni iperparametri, come la soglia di probabilità (*decision boundary*) per la classificazione, riguardano il

risultato finanziario che si desidera ottenere, e non è una questione che debba essere decisa dal machine learning. Abbiamo descritto come potremmo addestrare i parametri nel caso della regressione lineare e logistica, ma questo presuppone che abbiamo un insieme fisso di iperparametri. Come li ottimizziamo? La risposta è: validazione o cross-validazione. Questo processo è chiamato ottimizzazione degli iperparametri (HPO, *Hyperparameter Optimization*). Nella validazione, dividiamo i dati in tre sottoinsiemi non sovrapposti: il set di addestramento, il set di validazione e il set di test. Assegnando alcuni valori iniziali ragionevoli per gli iperparametri (per esempio, λ_1), usiamo il set di addestramento per trovare i parametri (per esempio, β). Quindi vediamo come si comporta il modello nel set di validazione. La metrica di performance può essere l'errore quadratico medio (MSE) nel caso della regressione, o la log-verosimiglianza nel caso della classificazione, o può essere qualcosa di più complicato, come l'F1 score o l'AUC score, di cui parleremo più avanti, o si potrebbero perfino usare le metriche di performance finanziaria per scegliere i parametri migliori. Dopo la valutazione delle performance, possiamo variare gli iperparametri, riaddestrare il modello e ripetere il processo fino a trovare il set "migliore" di iperparametri, che genera le performance più alte sul set di validazione. Questo "migliore" ha limiti determinati dalla pazienza e dalle risorse di calcolo disponibili. Esistono metodi fantasiosi per campionare in modo più intelligente lo spazio degli iperparametri (come utilizzare un ottimizzatore bayesiano), ma spesso viene utilizzata una ricerca a griglia (*grid search*) a forza bruta, se gli iperparametri non sono troppi. Per esempio, se la lunghezza del set di addestramento va considerata come un iperparametro, la si aumenta fino a quando le performance sul set di validazione smettono di migliorare. Una volta trovato il

miglior set di iperparametri e il corrispondente modello addestrato, quest'ultimo può essere applicato al set di test per vedere se tutto questo sforzo produce buone performance al di là del campione. In caso contrario, non si dovrebbe tornare indietro per modificare gli iperparametri e riprovare (per esempio, aggiungendo feature, provando altri algoritmi di machine learning, provando altri set di addestramento, ...). Si dovrebbe ammettere la sconfitta e abbandonare il progetto (López de Prado, 2015). Tuttavia, nulla vieta di cambiare le etichette e iniziare un nuovo progetto.

C'è un piccolo problema nel prendere una parte di dati da usare come set di validazione: abbiamo ridotto la quantità di dati che può essere utilizzata per l'addestramento, il che riduce la capacità del modello di generalizzare. Tuttavia, come vedremo più in dettaglio nel Capitolo 10, le tecniche più moderne consentono di addestrare un modello utilizzando piccoli dataset ma ottenendo una buona generalizzazione, sfruttando modelli pre-addestrati. Inoltre, specialmente per le serie temporali finanziarie, che tendono a violare l'ipotesi standard delle osservazioni indipendenti e identicamente distribuite (IID) della maggior parte dei modelli di machine learning, chi può dire che per scegliere gli iperparametri ottimali debba essere utilizzato un determinato periodo e non un altro? Un modo semplice per rimediare a ciò consiste nell'applicare la “cross-validazione”: suddividere ogni dato *che non è nel set di test* in K sottoinsiemi (chiamati “fold di validazione”), addestrare il modello k sui dati *non* nel k -esimo fold e fare inferenze sul k -esimo fold. Poi si media la performance del set di validazione su tutti i k -fold e si ottimizzano gli iperparametri in base a questa performance. In questo modo, per ottimizzare gli iperparametri abbiamo utilizzato tutti i dati al di fuori del set di test. Con “mediare le performance”, intendiamo fare inferenze/previsioni su ogni campione nei k -fold e poi calcolare la

metrica di performance desiderata su quelle etichette inferite. Non intendiamo solo calcolare la media dell'indice di Sharpe o di altre metriche non lineari sui vari fold.

Ci sono ulteriori problemi quando la cross-validazione viene applicata a dati a serie temporali, come quando le feature utilizzate per addestrare il modello k si sovrappongono nel tempo con il periodo del k -esimo fold, inflazionando così possibilmente le performance del modello k sul k -esimo fold a causa del *bias di anticipazione (look-ahead)*. López de Prado (2018) ha discusso ampiamente i rimedi a questo problema.

Metriche di performance

Per i gestori patrimoniali, le migliori metriche di performance di qualsiasi sistema di machine learning sono quelle finanziarie, come gli indici di Sharpe o Calmar. Ma se vogliamo una metrica di performance intermedia solo per il componente di machine learning di un sistema di investimento, ne possiamo applicare altre standard, che possono essere utilizzate per la validazione, il confronto o l'ottimizzazione degli iperparametri del modello di machine learning.

Per problemi di regressione, le metriche di performance sono il noto errore quadratico medio (MSE, “più è vicino a zero e meglio è”) e l' R^2 (“più è vicino a 1 e meglio è”). Come promemoria, R^2 misura la proporzione della varianza nell'etichetta che può essere spiegata dalle feature. Tuttavia, prima di entusiasmarci troppo per un buon MSE o R^2 , dovremmo eseguire alcuni controlli. A titolo illustrativo, vedi il quartetto di Anscombe

(https://it.wikipedia.org/wiki/Quartetto_di_Anscombe) nella Figura 3.5.

Questi quattro diagrammi di dispersione (x, y) , molto differenti, risultano avere lo stesso R^2 , alto, pari a 0.67 e molte altre statistiche descrittive uguali, ma nessuno penserebbe che la regressione lineare

sia un buon adattamento per il secondo e il quarto set di dati. Nel frattempo, il coefficiente di regressione è distorto per il terzo set a causa di un valore anomalo (*outlier*). L'ispezione visiva racconta una storia molto diversa e più accurata rispetto a una statistica numerica come R^2 . Tuttavia, quando abbiamo un set di dati a elevata dimensionalità, è molto difficile rilevare queste anomalie tramite un'ispezione visiva.

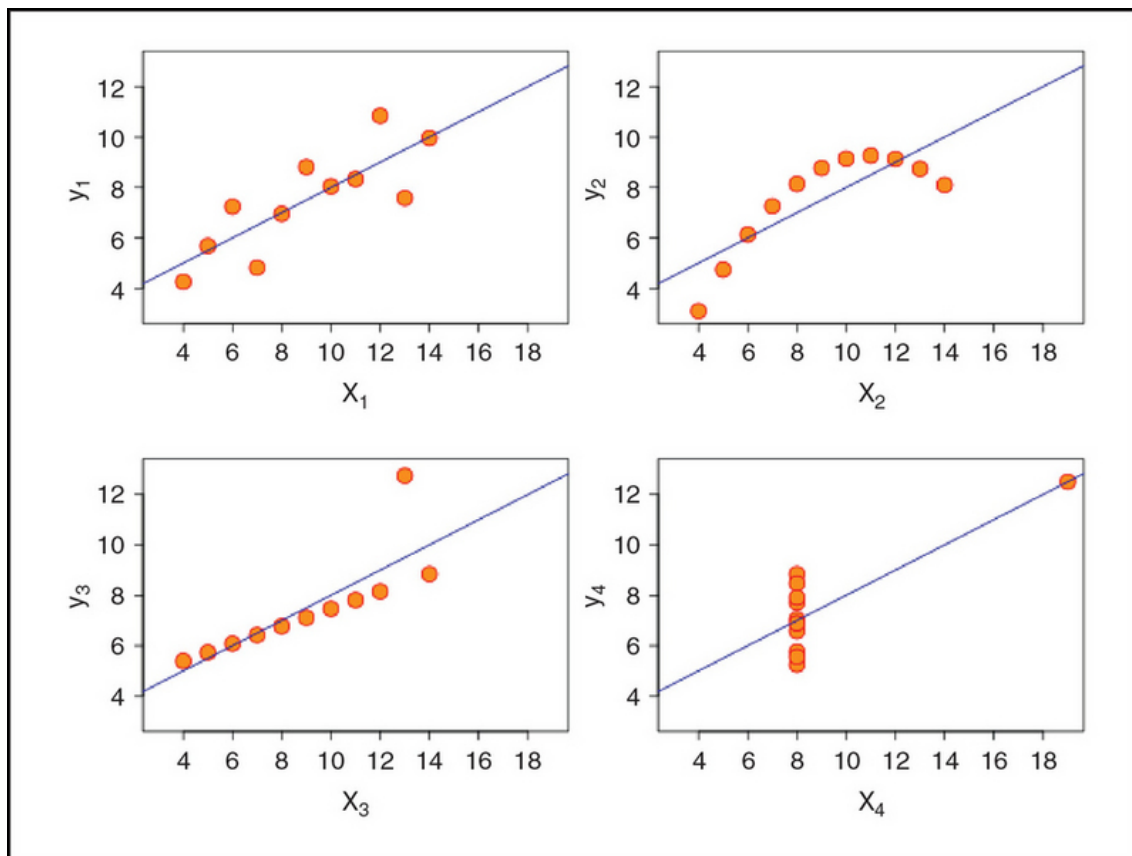


Figura 3.5 Rappresentazione grafiche del “quartetto di Anscombe”: tutti e quattro i set di dati hanno gli stessi valori per medie, varianze, correlazioni, coefficienti di regressione e R^2 . Fonte: Anscombe / <https://commons.wikimedia.org/wiki/File:Anscombe.svg#/media/File:Anscombe.svg>, ultimo accesso 13 gennaio 2025 / CC BY 3.0.

Quello che possiamo fare in quei casi è eseguire un'analisi esplorativa dei dati (EDA, *Exploratory Data Analysis*) per determinare se nei dati ci sono valori anomali e se questi sono dovuti a errori, e quindi dovrebbero essere esclusi. Quindi possiamo applicare la classificazione dell'importanza delle feature (*Feature Importance Ranking*) per trovare le feature meglio classificate. Ancora meglio, possiamo usare la MDA basata su cluster (cMDA) (vedi López de Prado [2020]) per trovare i cluster di feature meglio classificati e vedere se hanno senso. A volte, si scopre che feature particolarmente importanti che hanno un elevato potere predittivo incorporano un bias di anticipazione (*look-ahead*) e dovrebbero essere scartate. Affronteremo l'ingegneria delle feature più avanti in questo capitolo.

Per problemi di classificazione binaria, e per la maggior parte degli approcci probabilistici al machine learning, abbiamo già visto che la metrica usuale da ottimizzare è la log-verosimiglianza (LL). Questa è anche la metrica per confrontare il potere inferenziale di diversi classificatori (o magari dello stesso classificatore ma con iperparametri differenti) date le etichette nel set di test. Tuttavia, il punteggio LL non dà una comprensione intuitiva di quanto sia “buono” un classificatore, proprio come il MSE non dà un'idea intuitiva di quanto sia “buona” una regressione, perché il suo valore dipende dalla varianza intrinseca delle etichette. Si usano altre metriche di performance della classificazione.

La metrica di classificazione più comune è l'accuratezza della previsione: $\text{previsioni corrette} / \text{previsioni}$. Si potrebbe pensare che qualsiasi accuratezza superiore al 50% significhi che il modello funziona meglio del caso, e sia efficace. Ma immaginate se le etichette sono molto sbilanciate: il 90% di esse è a 0 e solo il 10% è a 1. Potremmo tranquillamente scommettere ciecamente su 0 e ottenere

automaticamente un tasso di accuratezza del 90%. Abbiamo bisogno di misure aggiuntive per dimostrare la bontà del nostro modello di machine learning. Si utilizzano la precisione e il recall. Tipicamente, queste vengono applicate alla classe minoritaria, o alla classe con etichetta 1 che consideriamo i risultati “positivi”. Per esempio, in *Corrective AI*, questo sarebbe il caso in cui la decisione di investimento è redditizia. La classe negativa è quella che si presume sia in maggioranza con etichetta 0. La precisione è $\text{precisioni } 1 \text{ corrette} / \text{previsioni } 1$ e il recall è $\text{previsioni } 1 \text{ corrette} / 1 \text{ effettivi}$. Le previsioni 1 corrette sono i veri positivi (TP, *True Positives*), mentre le previsioni 1 errate (note anche come errore di Tipo I) sono i falsi positivi (FP, *False Positives*). Le previsioni 0 corrette sono i veri negativi (TN, *True Negatives*), mentre le previsioni 0 errate (note anche come errore di Tipo II) sono i falsi negativi (FN, *False Negatives*). Quindi, si può anche scrivere $\text{precisione} = \text{TP} / \text{TP} + \text{FP}$ e $\text{recall} = \text{TP} / \text{TP} + \text{FN}$. Tipicamente, sia il recall sia la precisione dovrebbero essere superiori al 50% per indicare un certo qual potere predittivo. Nel nostro esempio, se il modello scommettesse sempre su 0, il recall sarebbe 0, e ciò indicherebbe che il modello non funziona. Se il modello scommettesse sempre su 1, la precisione sarebbe solo del 10%, indicando di nuovo che il modello non funziona. Per tenere conto sia della precisione sia del recall, spesso si calcola l’F1 score che è la media armonica di recall e precisione: $2 * (\text{precisione} * \text{recall}) / (\text{precisione} + \text{recall})$.

Si può vedere che l’F1 score è ovviamente 0 se precisione o recall sono 0. Sarà anche molto più piccolo di 0.5 se la precisione o il recall sono molto più piccoli di 0.5. Se vogliamo scendere nei dettagli, possiamo anche calcolare la matrice di confusione per ciascun quadrante (Tabella 3.3).

Tabella 3.3 La matrice di confusione per la classificazione binaria.

#	Previsto positivo	Previsto negativo
Effettivo positivo	TP	FN
Effettivo negativo	FP	TN

Troviamo utile questo riferimento mnemonico: nel trading, gli errori di Tipo I (FP) sono più gravi degli errori di Tipo II (FN) perché un FP può farci prendere una decisione di investimento sbagliata e farci perdere denaro, mentre un FN può farci rinunciare a una decisione di investimento che poi si rivela corretta e redditizia, però è anche vero che teniamo i nostri soldi al sicuro in banca. Quindi, un FP, un errore di Tipo I, ha maggior peso. La matrice di confusione dà anche la priorità alla prima riga e alla prima colonna per la classe positiva.

Quando abbiamo un problema di classificazione multi-classe (per esempio, il rendimento è etichettato come molto negativo, leggermente negativo o 0, leggermente positivo o molto positivo) i punteggi di precisione e recall devono essere calcolati considerando ogni singola classe come la classe positiva e raggruppando le altre classi insieme come classe negativa. L’F1 score viene poi calcolato come una sorta di media su questi punteggi di recall e precisione. Copilot lo spiega molto bene.

Risposta di Copilot

Micro-media: questo metodo aggrega i contributi di tutte le classi per calcolare la media. Calcola l’F1 score globalmente contando i totali dei veri positivi, dei falsi negativi e dei falsi positivi.

Macro-media: questo metodo calcola l’F1 score per ogni classe in modo indipendente e poi ne calcola la media. Questo tratta tutte le classi allo stesso modo, indipendentemente dalle loro dimensioni.

Media ponderata: questo metodo calcola l’F1 score per ogni classe in modo indipendente, ma quando calcola la media, utilizza un peso che dipende dal numero di istanze vere per ogni classe.

Una matrice di confusione per un problema di classificazione a tre classi sarebbe simile alla Tabella 3.4. Naturalmente, ciascuno dei TP e FN avrà un numero differente in ogni cella.

Tabella 3.4 Matrice di confusione per la classificazione a tre classi.

#	Classe prevista 1	Classe prevista 2	Classe prevista 3
Classe reale 1	TP	FN	FN
Classe reale 2	FN	TP	FN
Classe reale 3	FN	FN	TP

Esercizio #7

Chiedete a Copilot di generare una matrice di confusione casuale 3×3 e chiedetegli di calcolare recall e precisione per ogni classe, e di calcolare i tre tipi di F1 score. Verificate manualmente se questi numeri sono corretti.

C'è un iperparametro particolare che di solito vogliamo ottimizzare in un problema di classificazione binaria: la soglia di probabilità p_0 che decide se dovremmo inferire 0 o 1. Questo è chiamato anche il “confine decisionale” nel machine learning più in generale. Quali metriche di performance dovrebbero essere usate per valutarlo?

La solita funzione obiettivo da ottimizzare, la log-verosimiglianza (LL), non include p_0 come iperparametro, in quanto richiede come input solo la distribuzione di probabilità e le etichette reali. Se usiamo la precision o la recall come obiettivo, aumentarne una ottimizzando p_0 tipicamente porterà alla diminuzione dell'altra. Questo può essere visto nella curva PR ipotetica nella Figura 3.6 (generata da ChatGPT), che traccia la precision rispetto alla recall con p_0 decrescente verso il lato destro.

Possiamo vederlo anche dalla curva ROC (*Receiver Operating Characteristics*) della Figura 3.7, che traccia il tasso di veri positivi (TPR, ovvero il recall) rispetto al tasso di falsi positivi ($FPR = FP / \text{negativi reali} = FP / (FP + TN)$) ancora con p_0 decrescente verso il lato destro. Man mano che p_0 diminuisce, otteniamo naturalmente più predizioni positive, alcune giuste e alcune sbagliate, e quindi sia TPR sia FPR aumentano. È ovvio che la recall può essere al massimo 1, e anche l’FPR può essere al massimo 1, entrambi i casi si verificano quando etichettiamo erroneamente tutti i negativi come positivi.

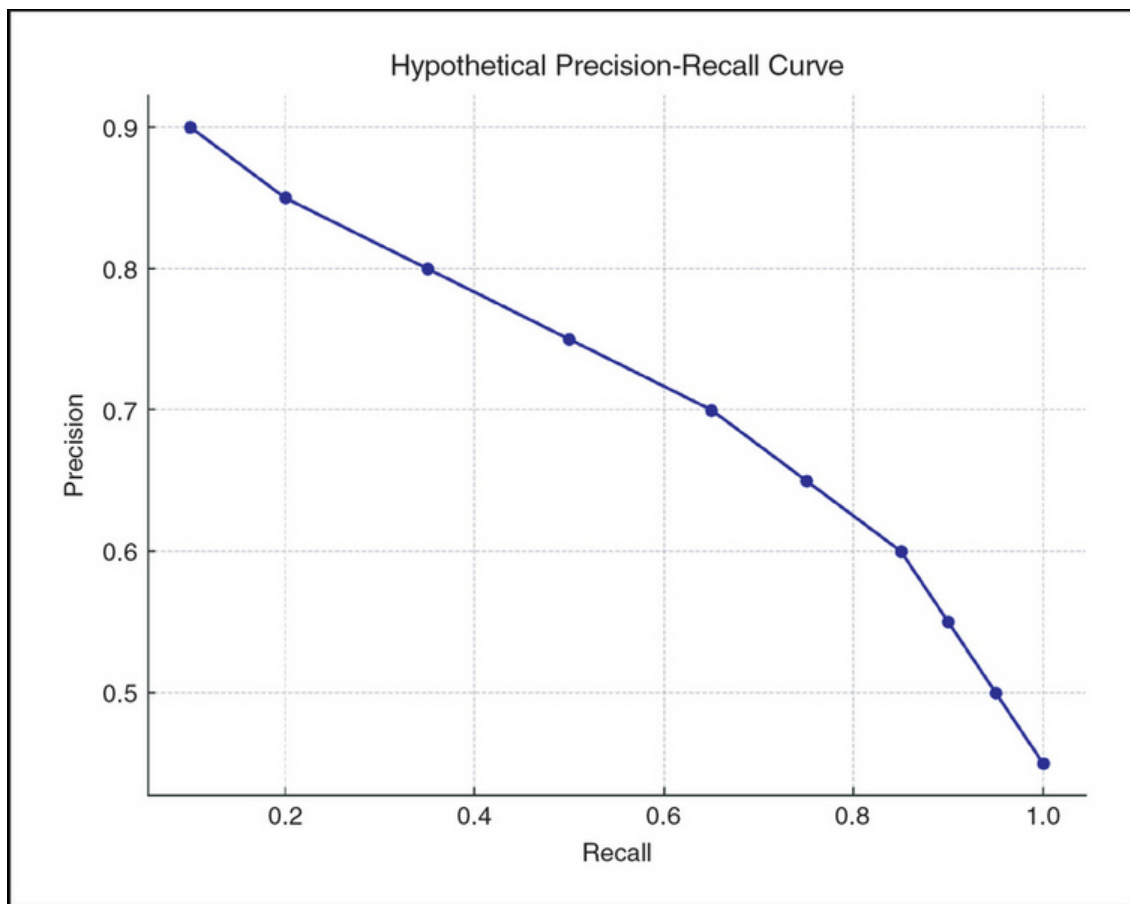


Figura 3.6 Grafico di una tipica curva precision-recall.

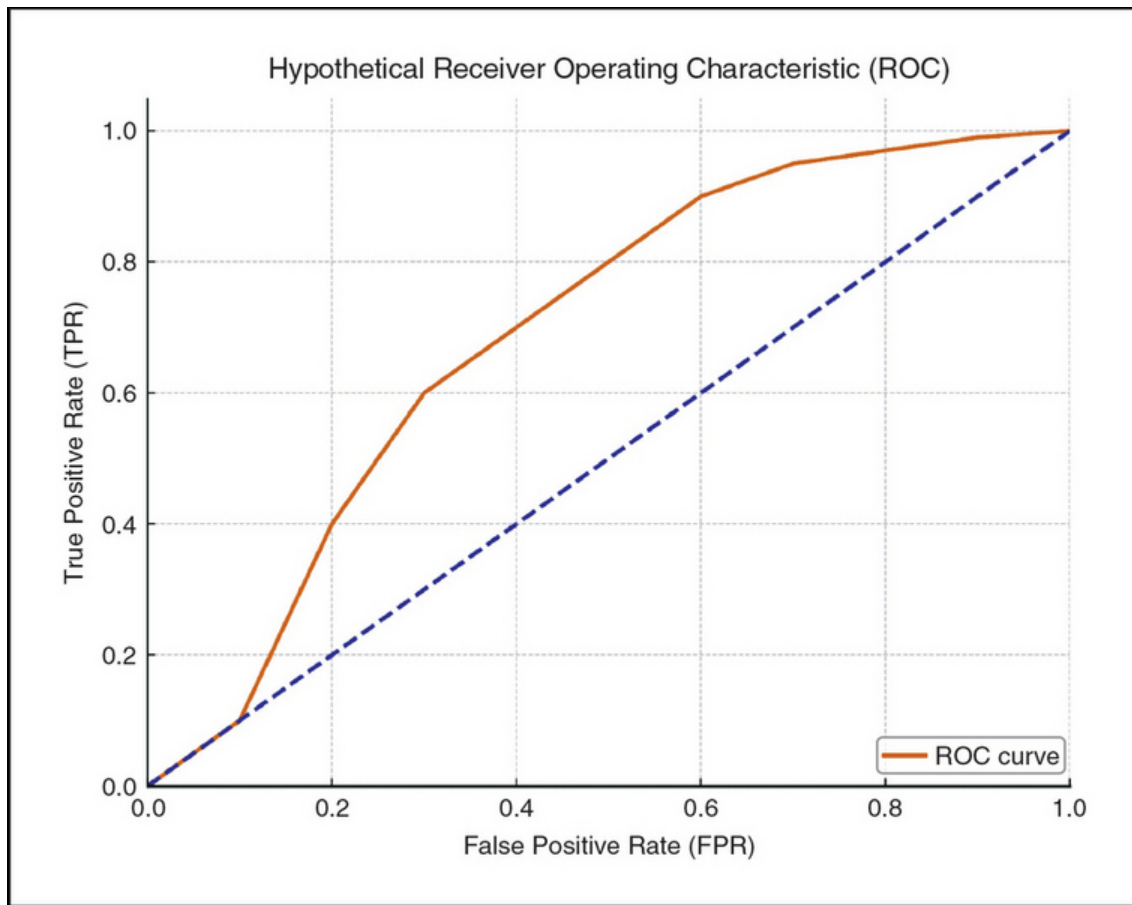


Figura 3.7 Grafico di una tipica curva ROC (Receiver Operating Characteristics).

Un classificatore binario che assegna probabilità casualmente con $p(0) = p(1) = 0.5$ darà una linea diagonale dritta come curva ROC, e più potere discriminativo ha il classificatore, più negativa è la sua curvatura. Ciò significa che in qualsiasi punto della curva tranne nei punti terminali, $TPR > FPR$. Se una curva ROC ha curvatura positiva, dobbiamo solo inferire 0 quando $p(1) > p_0$ e recupereremo lo stesso potere discriminativo. Cioè, un falso profeta che è consistentemente sbagliato diventa un vero profeta. Questo significa anche che possiamo usare come statistica l'area sotto la curva ROC, che descrive il potere discriminativo del classificatore. Quest'area è

chiamata banalmente AUC (*Area Under Curve*), e dovrebbe essere maggiore di 0.5 per avere un buon classificatore. Questa statistica è utile per confrontare più classificatori, specialmente durante la fase di modellazione, durante la quale vogliamo valutare la performance del modello indipendentemente dalla soglia di probabilità scelta. C'è anche una misura correlata chiamata AUCH (*Area Under Convex Hull*), che è molto simile all'AUC ma si applica all'involuppo convesso adattato ai dati reali di precisione e recall.

Esercizio #8

Chiedete a ChatGPT di generare 1001 campioni casuali di $y = 0$ oppure 1 basati sulla distribuzione binomiale, dove la probabilità della classe 1 è determinata da una funzione sigmoide di una feature scalare x che assume valori da -500 a 500 in incrementi di 1. Chiedetegli di adattare una regressione logistica su questi dati simulati e di calcolare le varie metriche di performance appena descritte. Calcola esattamente queste metriche? Il modello si adatta bene ai dati? Suggerimento: deve riuscire a farlo! Questo è un semplice adattamento in-sample.

Alberi di classificazione e regressione, foreste casuali e alberi potenziati

I modelli basati su alberi sono ampiamente considerati la tecnica di machine learning più efficace per feature tabulari e disomogenee (Shwartz-Ziv e Armon, 2021). La disomogeneità significa che le feature possono avere caratteristiche statistiche molto differenti. In finanza, una feature potrebbe essere la crescita annualizzata del PIL, un'altra potrebbe essere una feature categorica che indica se l'annuncio del tasso d'interesse della Fed è stato fatto il giorno precedente, e un'altra ancora potrebbe essere il rendimento a un giorno dello SPY. Al contrario, le reti neurali sono ampiamente considerate la tecnica di machine learning più efficace per feature

omogenee, come immagini, video, audio (incluso il parlato) e testi. Le feature omogenee sono esemplificate dalle parole in un file di testo: ogni parola, di solito, è estratta dallo stesso vocabolario. Come vedremo, tipicamente i modelli ad albero selezionano una feature alla volta, quindi il fatto che feature differenti abbiano proprietà statistiche differenti non rappresenta un problema. I semplici alberi di classificazione e regressione sono anche più spiegabili/interpretabili delle reti neurali, sebbene le versioni più elaborate, come gli alberi potenziati con gradiente o le foreste casuali lo siano molto meno.

Il metodo più noto e semplice per addestrare un albero è chiamato CART (*Classification And Regression Trees*, Breiman et al., 1984). A seconda del contesto, a volte anche l'albero stesso è chiamato CART. Per un problema di classificazione, l'algoritmo inizia con l'intero set di addestramento, scorre la lista delle feature e ne sceglie una che separa i campioni in due sottoinsiemi ("nodi figli") tali per cui ciascuno contenga "prevalentemente" campioni della stessa etichetta. Per essere più tecnici, ciò significa che scegliamo la feature tale per cui la somma ponderata degli indici di Gini dei due sottoinsiemi sia la più bassa. Se la feature scelta è continua e non categorica, allora la scelta della feature e della soglia c (che denota il cutoff) per dividere l'intervallo di valori della feature in due vengono fatte contemporaneamente. Ripetete questo processo ricorsivamente con ogni nodo figlio, fino a quando il numero di campioni di un nodo figlio raggiunge o scende al di sotto del minimo, o quando la profondità del ramo raggiunge un certo massimo. Come potete intuire, è un algoritmo *greedy*, perché sceglie sempre la feature migliore una alla volta, senza considerare la possibilità che scegliere consecutivamente più feature subottimali possa alla fine produrre un indice di Gini più basso. È facile vedere come possiamo inferire

l'etichetta dato un nuovo set di feature: basta seguire le regole decisionali fino al nodo foglia e vedere a quale classe appartiene.

La probabilità di finire in quella classe y basata sulle feature $x_1, \dots, x_j$ selezionate prima del nodo foglia $n + 1$ è semplicemente

$$p(y | x_1 \geq c_1, x_2 \geq c_2, \dots, x_j \geq c_j) = \frac{\text{campioni di addestramento di classe } y \text{ nel nodo foglia } (j + 1)}{\text{campioni di addestramento nel nodo foglia } (j + 1)}$$

Non c'è perdita di generalità nell'ipotizzare che valga la regola decisionale $\geq$, perché possiamo sempre applicare un segno meno alle feature continue, e il $=$ si occupa delle feature categoriche.

Per un problema di regressione, l'algoritmo CART è molto simile, tranne per il fatto che il criterio per scegliere una feature è determinato minimizzando il MSE ponderato dei nodi figli. Per l'inferenza, possiamo semplicemente calcolare il valore medio delle etichette nei campioni di addestramento in un nodo foglia

$$p(y | x_1 \geq c_1, x_2 \geq c_2, \dots, x_j \geq c_j) = \frac{\sum_{s \in \text{nodo } j+1} y_s}{\text{campioni di addestramento in nodo foglia } (j + 1)}$$

Le tecniche di regolarizzazione più semplici per gli alberi consistono nell'impostare un minimo per il numero di campioni in qualsiasi nodo e/o impostare una profondità massima (cioè, il numero massimo di feature utilizzate) di qualsiasi ramo di un albero. I data scientist hanno anche scoperto che è utile calcolare la media delle inferenze di un insieme di K alberi, dove ogni albero viene addestrato utilizzando campioni estratti casualmente con reinserimento dal set di addestramento originale. Questo insieme è chiamato foreste casuali. Per un albero di classificazione, ciò significa calcolare la media della probabilità di inferire la classe y e scegliere la y che ha la probabilità più elevata, dati i valori delle feature. Per un albero di regressione, si può semplicemente calcolare la media dei valori

inferiti dell'etichetta attraverso tutti gli alberi, dati i valori delle feature.

Le prestazioni di qualsiasi modello di machine learning possono essere potenziate tramite una tecnica chiamata *boosting*. L'implementazione più nota del boosting è il boosting del gradiente, un algoritmo ricorsivo. Ipotizziamo di avere già un modello potenziato M_k (inizializzato con il modello originale non potenziato $k = 0$). Il modello fa un'inferenza $\mathbf{y}_k$ (usiamo un vettore per indicare i molti campioni di etichette). Calcoliamo gli errori di queste inferenze $\mathbf{y}_k - \hat{\mathbf{y}}$ e usiamo questi errori come nuove etichette per addestrare un modello di predizione/correzione dell'errore m_k . Un nuovo modello di insieme $M_{k+1} = M_k - m_k$ ora è pronto per la prossima iterazione (siamo questa notazione per indicare che l'output di M_{k+1} è l'output di M_k meno l'output di m_k). Gli errori di inferenza (almeno sul set di addestramento) di M dovrebbero diminuire a ogni iterazione (vedi <https://developers.google.com/machine-learning/decision-forests/intro-to-gbdt>). Ci fermeremo quando gli errori diventeranno abbastanza piccoli (nel caso della classificazione, quando il numero di casi positivi diventa abbastanza piccolo), o quando il miglioramento del tasso di errore non è sufficiente, o quando superiamo il numero massimo di iterazioni. Notate che, sebbene colleghiamo insieme modelli differenti in un modello/albero potenziato, li aggiungiamo sequenzialmente, e i parametri della sequenza di modelli non vengono modificati negli addestramenti successivi. Come vedremo, questo è simile alle reti neurali in quanto anch'esse collegano insieme più modelli elementari, ma in una rete neurale i parametri dei modelli elementari cambiano durante l'addestramento.

Un modo per regolarizzare un modello potenziato per ridurre l'overfitting consiste nell'introdurre un tasso di apprendimento $0 \leq \epsilon \leq 1$ tale che $M_{k+1} = M_k - \epsilon m_k$. In altre parole, non ci fidiamo della

piena predizione dell'errore per correggere il modello. I tipici iperparametri da ottimizzare sono il tasso di apprendimento ϵ e il numero massimo di alberi $n_estimators$ nell'insieme. Quando ϵ è vicino a 0, $n_estimators$ deve essere alto per minimizzare gli errori. Il concetto di tasso di apprendimento è rilevante per l'addestramento delle reti neurali, che affronteremo di seguito.

Ci sono molti tentativi di migliorare il classico algoritmo CART per la crescita degli alberi. Agli alberi sono applicate anche tecniche di deep learning (specificamente usando l'*attention*, vedi Zhuang et al., 2024; Arik e Pfister, 2020). Nelle applicazioni finanziarie, Cong et al. (2024) hanno descritto l'approccio *Panel Tree* per classificare intere serie di rendimenti azionari, che offre una performance di investimento superiore. Questi sforzi sono fatti a causa della particolare interpretabilità del CART e del suo potere predittivo sui dati tabulari. C'è ancora un dibattito in corso sul potere predittivo dei modelli basati su alberi rispetto al deep learning per dati tabulari. Un eccellente articolo che mostra le prestazioni dei modelli basati su albero in questo contesto è quello di Grinsztajn et al. (2022), che raccomandiamo ai lettori.

Reti neurali

Potreste aver notato che gli algoritmi di machine learning che abbiamo descritto finora, regressioni lineari o logistiche o CART, restituiscono sempre un'etichetta scalare y , a valori continui o categorica (potremmo aver scritto $\mathbf{y}$ come un vettore, a volte, ma solo perché impiliamo tutti i campioni insieme durante l'addestramento). Questi non possono essere usati per produrre un vettore. Potete certamente costruire modelli multipli usando lo stesso input per produrre un vettore, come nella regressione multivariata, ma quei modelli vengono addestrati separatamente e hanno parametri

indipendenti. Tuttavia, a volte troviamo che addestrare un modello su etichette che sono vettori genera inferenze migliori. Per esempio, in Kolm et al. (2021), gli autori hanno usato una rete neurale per predire la struttura a termine dei rendimenti basata sul libro degli ordini. Il fatto che conosciamo (come etichette, non come feature) i rendimenti futuri su diversi orizzonti temporali ci fornisce un set di informazioni più ricco per addestrare un modello migliore. Le reti neurali sono un algoritmo di machine learning molto più flessibile per produrre tali output vettoriali. Questo beneficio può superare lo svantaggio menzionato in precedenza che i modelli ad albero funzionano meglio per feature tabulari ed eterogenee; quando le etichette sono vettori, non avete scelta se non usare le reti neurali.

Esercizio #9

Quali sono gli input esatti usati nell'articolo di Kolm appena citato? Quanti diversi orizzonti temporali vengono usati per le etichette? Che tipo di architettura di rete neurale viene usata? Chiedete a Copilot di aiutarvi, o addirittura di generare del codice basato su questo articolo.

Per motivare la scelta delle reti neurali, abbiamo visto prima come le macchine potenziate combinino sequenzialmente più modelli elementari (per esempio, ognuno un singolo albero) per potenziare le prestazioni. Per estendere questo concetto, perché non connettere questi modelli sia in parallelo sia sequenzialmente? Perché non prendere gli output dei vari modelli in modo che l'output globale sia un vettore o addirittura un tensore? Perché non permettere a tutti i parametri di queste macchine di essere regolabili durante l'addestramento invece di fissarli sequenzialmente? Se concediamo tutte queste possibilità, finiamo con una rete neurale feed-forward, nota anche come perceptron multi-layer (MLP, *Multilayer Perceptron*), che è il tipo più semplice di rete neurale (Goodfellow et al., 2016), vedi Figura 3.8.

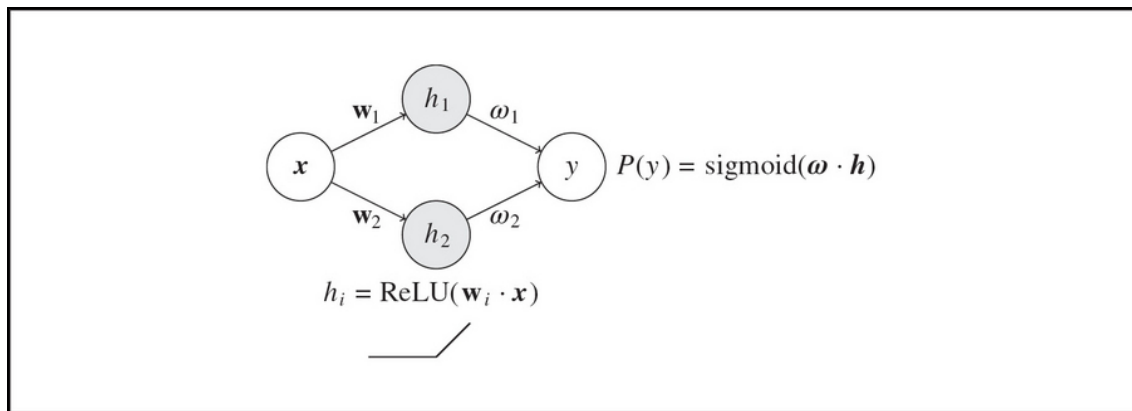


Figura 3.8 Diagramma di un MLP a un layer nascosto che prende un vettore di feature $\mathbf{x}$ per classificare un'etichetta binaria scalare y . Ogni freccia denota una funzione lineare (prodotto scalare) con pesi (coefficienti di regressione) e bias (offset). Per un compito di regressione, possiamo omettere la funzione sigmoide al nodo di output. In generale, possiamo anche predire un vettore di etichette binarie $\mathbf{y}$, con pesi ω differenti. Possiamo anche avere più nodi nel layer nascosto e/o aggiungere più layer nascosti. Per semplificare la notazione, abbiamo usato $\mathbf{w}$ e ω per rappresentare sia i pesi sia i bias.

Il MLP più semplice prevede tre layer.

1. Un layer di input che è solo un insieme di regressori lineari i che accettano tutti le stesse feature $\mathbf{x}$ e ognuno restituisce uno scalare $w_i \cdot \mathbf{x}$. Usiamo $\mathbf{w}_i$ invece di β_i perché è tipico nella letteratura riguardante le reti neurali. Per semplificare le notazioni, abbiamo usato anche $\mathbf{w}$ invece di $\mathbf{w}$ e $\mathbf{b}$ per includere sia i pesi (*weight*) sia i bias, che sono gli offset costanti nelle regressioni lineari. Questo significa che abbiamo ipotizzato un “1” da aggiungere al vettore di input $\mathbf{x}$. Questo è per pura comodità notazionale, perché, in pratica, per ragioni di tempo di calcolo e di complessità di memoria, è più efficiente aggiungere i bias direttamente piuttosto che aggiungerne uno all'input,

specialmente quando il numero di regressioni diventa grande. Questi sono i parametri fondamentali di una rete neurale.

2. Questi scalari $\mathbf{w}_i \cdot \mathbf{x}$ sono ciascuno iniettati in un nodo nascosto. Qui introduciamo una funzione non lineare che è la più vicina possibile a un modello lineare: come l'unità ReLU $h(z) = \max(0, z)$. L'output di ogni unità ReLU è ancora uno scalare, $h_i = \max(0, \mathbf{w}_i \cdot \mathbf{x})$, e noto con il nome di *attivazione*.
3. Questi scalari h_i , insieme, formano un vettore che è iniettato nel nodo (o nei nodi) di output, che è un altro regressore lineare $y = \boldsymbol{\omega} \cdot \mathbf{h}$ per un problema di regressione, o la nostra ormai nota funzione sigmoide $y = 1 / [1 + \exp(\boldsymbol{\omega} \cdot \mathbf{h})]$ per un problema di classificazione binaria. Quest'ultima rappresenta la probabilità di una classe, come nella regressione logistica. Ancora una volta, abbiamo semplificato le notazioni usando $\boldsymbol{\omega}$ per rappresentare sia i pesi sia il bias, e abbiamo ipotizzato di aver aggiunto "1" al vettore $\mathbf{h}$.

Poiché abbiamo usato lo stesso simbolo per rappresentare sia i pesi sia i bias, ci riferiremo a entrambi come semplicemente "pesi", nel testo.

Possiamo estendere l'MLP aggiungendo più layer nascosti, dove gli output di un layer nascosto vengono usati come input del successivo layer nascosto. Possiamo usare come input, pesi e output anche tensori di dimensione superiore. Per la classificazione multi-classe, possiamo sostituire la funzione sigmoide con la funzione

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Non preoccupatevi troppo per tutte queste moltiplicazioni di matrici. Le librerie per reti neurali, come PyTorch o TensorFlow se ne occuperanno per noi, purché specifichiamo le dimensioni degli input

e degli output, il numero di layer nascosti e il numero di nodi per ogni layer.

Una ricerca mostra che un MLP con un solo layer nascosto può approssimare qualsiasi funzione di complessità ragionevole (Hornik et al., 1989). Ma ciò che questa ricerca non dice è che in questo layer avremmo bisogno di $\exp(\text{numero di feature})$ nodi per avere una buona approssimazione. Come si sa, generalmente è più efficiente addestrare una rete con più layer nascosti, ognuno con pochi nodi, che un singolo layer nascosto con molti nodi.

Per addestrare una rete neurale, dobbiamo decidere quale funzione obiettivo utilizzare. Come nelle regressioni lineari e logistiche e quasi ogni altro problema di machine learning: ottimizzare la log-verosimiglianza è la scelta giusta la maggior parte delle volte. Ipotizzando una distribuzione di probabilità gaussiana per il layer di output, questo è ancora equivalente a minimizzare il MSE per un problema di regressione, come menzionato per le regressioni lineare e logistica. Per massimizzare la log-verosimiglianza per la regressione logistica, possiamo ancora ricorrere alla discesa del gradiente dei parametri, ora chiamati pesi. Il problema dell'applicare la discesa del gradiente al set molto più grande di parametri che sono tipici di una rete neurale e a un grande set di addestramento è il tempo di calcolo, che diventa proibitivo. Quindi c'è un algoritmo più efficace, chiamato discesa stocastica del gradiente (SGD – *Stochastic Gradient Descent*) che per calcolare il gradiente campiona un piccolo sottoinsieme del set di addestramento. Questo sottoinsieme è chiamato mini-batch e, al massimo, è costituito da poche centinaia di campioni. Più piccole sono le dimensioni di un mini-batch, più la rete è regolarizzata. Proprio come nell'algoritmo di boosting descritto in precedenza, aggiorneremo i parametri $\theta_{i+1} = \theta_i - \epsilon \mathbf{g}$ dove θ rappresenta tutti i pesi della rete, $\mathbf{g}$ è il gradiente, e ϵ è il tasso di

apprendimento (ci sono diverse tecniche per accelerare l'addestramento del modello modificando queste equazioni, come vedremo più tardi). Il calcolo di $\mathbf{g}$ può essere eseguito in modo efficiente da un algoritmo forward-backward chiamato *backpropagation*.

La *backpropagation* si basa sulla regola della catena del calcolo per calcolare ricorsivamente $\mathbf{g}$, che è la derivata parziale prima della log-verosimiglianza negativa L (o, in generale, qualsiasi funzione di costo J) rispetto a un parametro θ_i . Per calcolare la funzione di costo può essere usato il nodo di output inferito y (insieme all'etichetta reale $\hat{y}$) di una rete neurale. Usando la regola della catena del calcolo,

$$g_i = \left. \frac{\partial J}{\partial \theta_i} \right|_{\hat{y}} = \sum_j \left. \frac{\partial J}{\partial h_j} \right|_{\hat{y}} \frac{\partial h_j}{\partial \theta_i}$$

Notate che la somma è su tutti i nodi nascosti j che portano in avanti al nodo di output y , e che h_j è l'output dal nodo j .

In altre parole, su tutti i nodi genitori di y . Per calcolare $\partial h_j / \partial \theta_i$, possiamo invocare di nuovo la regola della catena ed esprimerla in termini di tutte le derivate parziali $\partial h_j / \partial u_k$ rispetto a tutti gli output da tutti i nodi genitori k che portano in avanti al nodo j . E così via fino a raggiungere il nodo con il parametro i . Eccetto che per il nodo di output, tutte le altre derivate parziali $\partial h_j / \partial h_k$ sono solo semplici funzioni di tutti gli output dei nodi genitori di j così come dei pesi che prendono l'output del nodo genitore k e lo trasformano in un input per il nodo figlio j . Per esempio, per il nodo nascosto $j = 3$, diciamo $h_3 = \text{ReLU}(0.1 * h_1 + 0.2 * h_2 - 0.2)$, e $h_1 = 1$ e $h_2 = 1$, allora $\partial h_3 / \partial h_1 = 0.1$. Se $h_1 = 1$ e $h_2 = 0$, allora $\partial h_3 / \partial h_1 = 0$. Per il nodo di output, diciamo che J è la funzione MSE, e $J = \sum_s (0.1 * h_3 + 0.2 * h_4 + 0.3 - \hat{y}_s)^2$ dove s è su tutti i campioni nel mini-batch, e $h_3 = 1$ e $h_4 =$

0.5. Ipotizziamo inoltre di avere un solo campione nel mini-batch e $\hat{y} = 1$, allora $\partial J / \partial h_3 = 2 * (0.1 * 1 + 0.2 * 0.5 + 0.3 - 1) * 0.5$.

Esercizio #10

Nell'esempio precedente, che cosa sarebbe $\partial h_3 / \partial h_1$, dati due set alternativi h_1 e h_2 ? Se il mini-batch ha due campioni, e $h_3 = -1$, $h_4 = 1$ e $\hat{y} = 0.5$ per il secondo campione, che cosa sarebbe J ?

Per iniziare il processo di addestramento (*backprop*), prendiamo campioni da un mini-batch di dati di addestramento, inizializziamo i parametri θ_1 ad alcuni valori (consiglio: inizializzate i pesi a piccoli valori casuali nell'intervallo

$$\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right)$$

dove m è il numero di feature, e inizializzate i bias a zero) e li eseguiamo *in avanti* attraverso la rete per calcolare tutti gli output u_i di tutti i nodi i . Cioè, andando layer per layer e link per link attraverso tutte le funzioni lineari e ReLU o sigmoide, attraverso tutte le moltiplicazioni di matrici di pesi, e poi finalmente arrivare ai nodi di output $\mathbf{y}$. Tutti gli $\mathbf{y}$ dai diversi campioni vengono combinati con le loro corrispondenti etichette reali $\hat{\mathbf{y}}$ per produrre

$$\left. \frac{\partial J}{\partial u_n} \right|_{\hat{\mathbf{y}}}$$

aggregato su tutti i campioni nel mini-batch. Questa è la propagazione in avanti che produce i valori di output di tutti i nodi. Ora siamo pronti per calcolare il gradiente $\mathbf{g}$ usando la regola della catena descritta in precedenza, partendo dal nodo di output e andando *all'indietro*, come nell'esempio numerico precedente. Dopo aver calcolato $\mathbf{g}$, possiamo usarlo per aggiornare $\theta_2 = \theta_1 - \epsilon_g$. Questo conclude un'iterazione della discesa stocastica del gradiente, e

possiamo ricominciare con un altro mini-batch. Parleremo più avanti di quando dovremmo smettere di iterare. Spesso non è necessario aspettare che il gradiente sia 0 per fermarsi, né ci si dovrebbe preoccupare se abbiamo raggiunto un minimo locale o globale nella funzione di costo.

In questo algoritmo forward-backward, nessuna derivata parziale viene mai calcolata più di una volta, perché ogni volta che ne viene calcolata una, viene memorizzata per poter essere utilizzata in un altro percorso di propagazione all'indietro. Per qualsiasi rete con η nodi, non ci possono essere più di η^2 archi e quindi più di $O(\eta^2)$ operazioni per calcolare i gradienti. Molto spesso, la rete neurale non sarà pienamente connessa e avrà solo $O(\eta)$ archi, con un risultato di avere $O(\eta)$ operazioni.

Se l'addestramento di una rete neurale vi preoccupa, considerate che in realtà non avrete mai bisogno di implementare nessuno di questi passaggi complicati: PyTorch, TensorFlow e così via fanno uso della differenziazione automatica (chiamata anche autograd) e si occuperanno di tutto. Ma è bene conoscere i principi di alto livello, per sapere quali iperparametri provare.

Per regolarizzare una rete neurale, abbiamo già menzionato al fatto che un iperparametro è il numero di campioni in un mini-batch. Più piccolo è, più la rete è regolarizzata. Un altro iperparametro per la regolarizzazione è il tasso di apprendimento ϵ durante la discesa del gradiente. Più piccolo è ϵ , più la rete è regolarizzata. Tuttavia, si è scoperto che l'addestramento funziona meglio se ϵ diminuisce linearmente nel tempo. Goodfellow et al. (2016) suggeriscono che dopo poche centinaia di iterazioni, ϵ dovrebbe scendere all'1% del suo valore iniziale e rimanere stabile. Come valore iniziale, si può ottimizzarlo sul set di validazione con le prime 100 iterazioni e impostarlo appena un po' più alto del valore ottimale. Un'altra

tecnica consiste nell'usare un tasso di apprendimento ciclico, che varia all'interno di un intervallo di valori invece di diminuirlo passo a passo (che, tra l'altro, è già implementato in PyTorch, vedi https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CyclicLR.html). Maggiori dettagli su questo approccio possono essere trovati in Smith (2017).

Possiamo anche usare regolarizzazioni L1 o L2. La funzione di costo J presentata in precedenza può essere impostata alla log-verosimiglianza negativa più $\lambda_1 \sum_i |W_i|$ o $\lambda_2 \sum_i W_i^2$ o a entrambe. Notate che tipicamente regolarizziamo solo i pesi ma non i bias. Come al solito, più grandi sono i λ , più la rete è regolarizzata. Tuttavia, come menzionato in precedenza, queste tecniche di regolarizzazione sono meno comunemente usate in pratica per le reti neurali profonde.

Un'altra nota tecnica di regolarizzazione è l'arresto anticipato: essenzialmente trattare il numero di passi di addestramento (un'iterazione forward-backward) come un iperparametro da ottimizzare sul set di validazione. Una volta che il costo scende a un plateau nel set di validazione in funzione dei passi di addestramento, fermeremo l'addestramento anche se un ulteriore addestramento diminuirebbe ulteriormente il costo sul set di addestramento.

Ultima, ma non meno importante, c'è la potente tecnica del dropout (Srivastava et al., 2014): rimuovere casualmente nodi non di output di una rete per ogni iterazione del mini-batch (tipicamente i nodi di input sono inclusi con probabilità 0.8, i nodi nascosti sono inclusi con probabilità 0.5). Tuttavia, i ricercatori hanno scoperto che il dropout non funziona bene quando i dati di addestramento hanno meno di 5.000 campioni.

Mentre la discesa stocastica del gradiente è una tecnica di addestramento canonica, i ricercatori hanno cercato di migliorare il suo tasso di convergenza. C'è un metodo chiamato *momentum* che usa

una media mobile esponenziale dei gradienti calcolati su diverse iterazioni. Questa è chiamata “velocità” v , e la regola di aggiornamento diventa $\theta_2 = \theta_1 - \epsilon v$. Sono stati proposti anche altri miglioramenti, come RMSProp, Adam e BFGS. C’è anche la “normalizzazione del batch”, che cerca di mantenere fisse la media e la varianza dell’output delle unità di input o nascoste durante l’addestramento, normalizzandole rispettivamente a quasi 0 e 1. Solo le correlazioni tra unità nascoste e statistiche di ordine superiore sono autorizzate a cambiare. I dettagli possono essere trovati in Goodfellow et al. (2016).

Esercizio #11

Costruite una rete neurale con un layer nascosto dove il layer ha due nodi nascosti per la classificazione. Collegate l’output del primo nodo nascosto sia al nodo di output sia al secondo nodo nascosto. Notate che questo significa che non è più una rete neurale feed-forward né un MLP. Etichetta = Il rendimento del giorno successivo è positivo? Feature = ultimo rendimento un giorno SPY, ultimi 2 giorni dei rendimenti SPY. Con l’aiuto di ChatGPT, scrivete un programma Python per addestrare questa rete senza usare librerie per reti neurali come Tensorflow. Qual è la sua performance? Riuscite a sintonizzare gli iperparametri per migliorare il set di validazione?

Reti neurali ricorrenti

Ipotizziamo di avere un classico problema di previsione di serie temporali nel trading: vogliamo predire il rendimento del periodo successivo utilizzando un indicatore tecnico noto nel periodo corrente. Per essere più concreti, ipotizziamo di voler utilizzare un MLP (o, in effetti, un qualsiasi sistema di machine learning come CART) per predire il rendimento $y(t)$ del periodo successivo utilizzando come feature il rendimento dell’indice di mercato $x(t)$. Ogni campione, cioè ogni periodo, ha un’etichetta scalare e una

feature scalare. Possiamo utilizzare il semplice MLP della Figura 3.9 per rappresentare questo scenario.

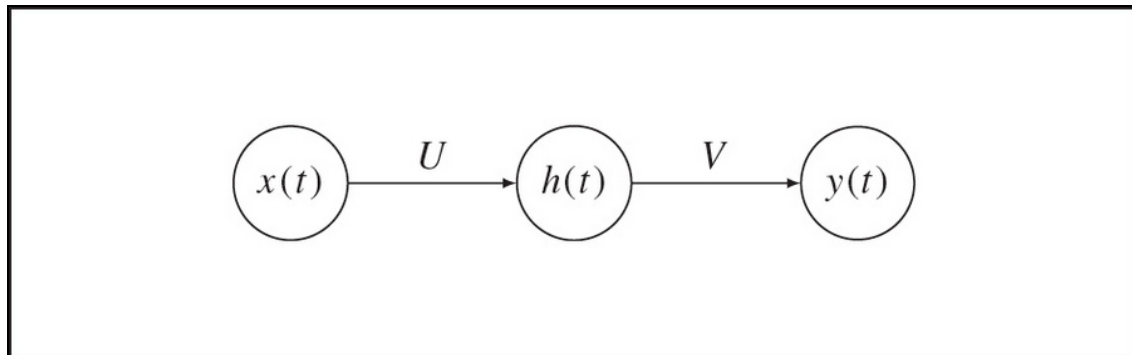


Figura 3.9 Rappresentazione grafica di un semplice MLP per la previsione di serie temporali.

Esercizio #12

Chiedete a ChatGPT di aiutarvi ad addestrare questo MLP utilizzando come feature i rendimenti giornalieri di SPY e come etichette i rendimenti giornalieri di MSFT del giorno successivo. Regularizzate il MLP e ottimizzate gli iperparametri su un set di validazione. Che aspetto hanno gli indici di Sharpe sui set di addestramento, validazione e test?

U indica i pesi che collegano il nodo di input al nodo nascosto h , e V indica quelli che collegano il nodo nascosto al nodo di output y . Tuttavia, per un modello di machine learning, potrebbe essere vantaggioso ricordare quali erano le feature e le etichette passate quando si predice l'etichetta successiva, invece di trattarle come campioni indipendenti, specialmente durante le inferenze. Un modo semplice per passare informazioni dalle feature e dalle etichette precedenti dal tempo $t - 1$ al tempo t è iniettare l'output di $h(t - 1)$ come input di $h(t)$. Possiamo immaginare una rete che in t ha l'aspetto rappresentato nella Figura 3.10.

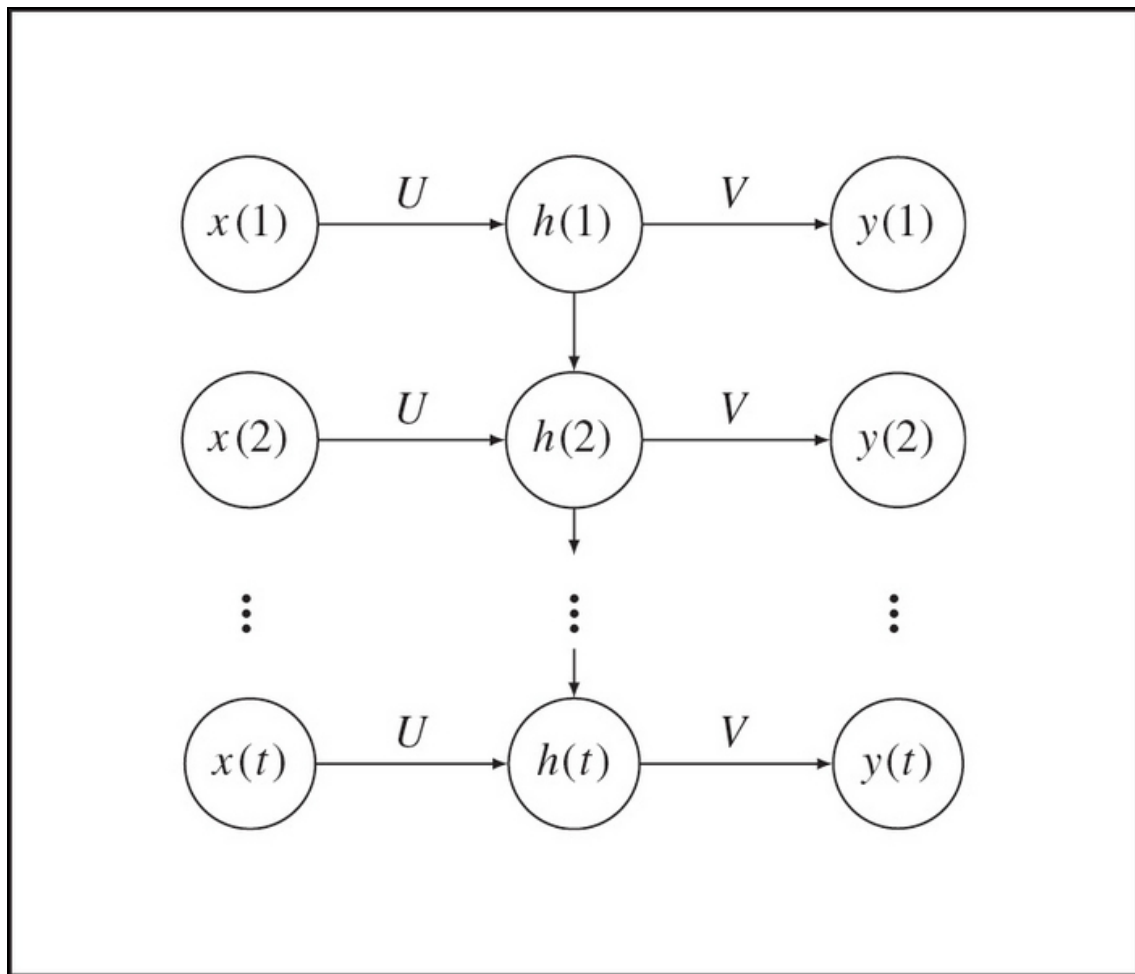


Figura 3.10 Rappresentazione grafica di una rete di MLP in cui leghiamo i pesi agli stessi valori e colleghiamo i nodi nascosti per trasmettere informazioni da $t - 1$ a t .

Poiché si ipotizza che la serie temporale sia stazionaria (se non lo è, il machine learning non funzionerà), possiamo forzare tutti i pesi U , V e W (i pesi che collegano i nodi nascosti) a essere gli stessi per ogni t . Durante l'addestramento, tutte le feature e le etichette fino a $t - 1$ sarebbero disponibili e saranno state utilizzate per determinare indirettamente $h(t - 1)$ in quanto influenzeranno i valori di U , V e W . Ogni volta che avanziamo di un passo temporale a t , la rete aggiunge un altro nodo di input, un nodo nascosto e un nodo di output. Poiché i

pesi si presume siano gli stessi in tutta la rete, non c'è problema nel fare una previsione a t . Una volta che abbiamo l'etichetta effettiva $\hat{y}(t)$, possiamo riaddestrare la rete e regolare tutti i pesi (se vogliamo). Abbiamo appena descritto una tipica rete neurale ricorrente (RNN). La maggior parte dei libri visualizza una rete neurale ricorrente ruotando la nostra figura di 90° in senso antiorario, in modo che il tempo proceda sull'asse orizzontale (come è comune nei grafici dei prezzi azionari) e l'output sia in alto, come nella Figura 3.11.

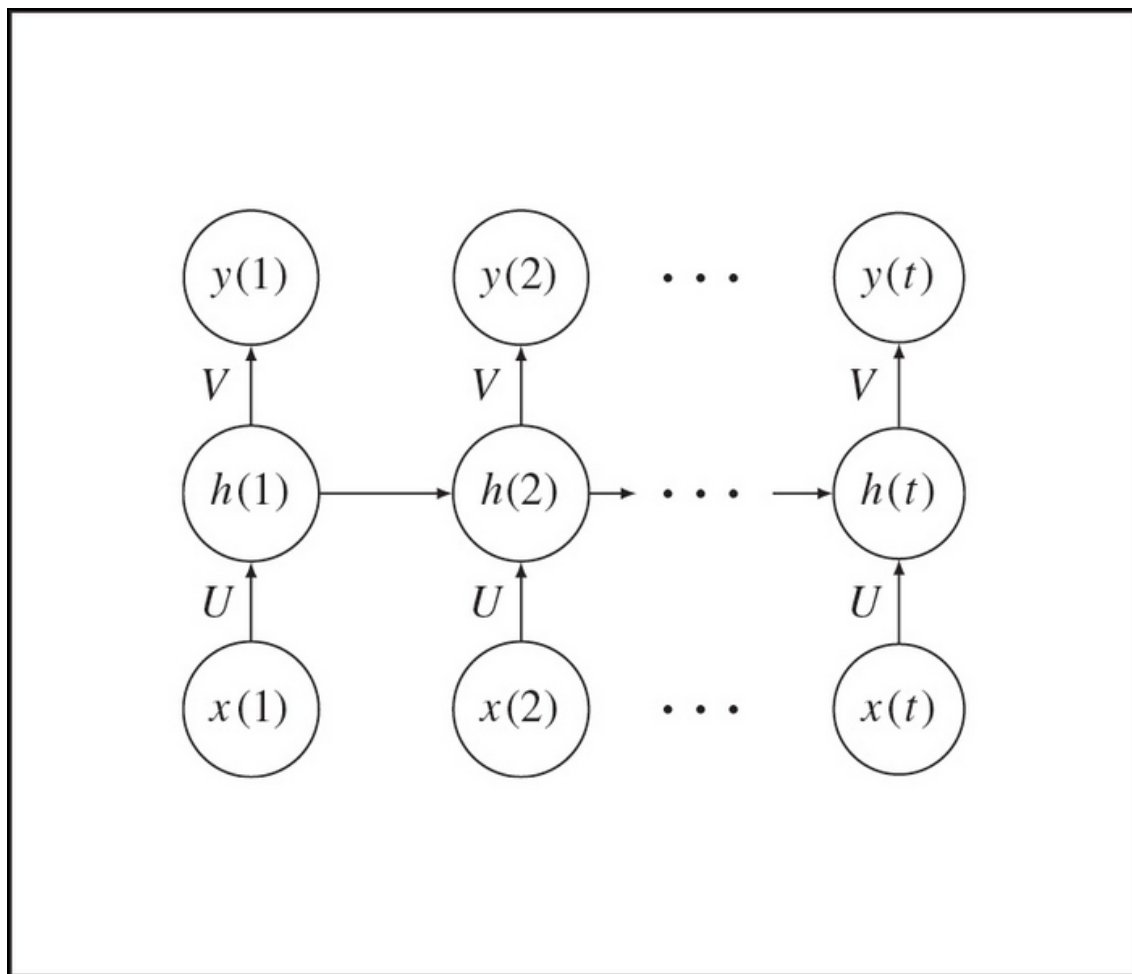


Figura 3.11 La rappresentazione grafica della Figura 3.10, ma ruotata di 90° in senso antiorario in modo che il tempo proceda da sinistra a destra.

Notate la somiglianza con un *Hidden Markov Model*. Durante la previsione, potete pensare che gli input precedenti x da 1 a $t - 1$ modifichino uno stato nascosto $h(t - 1)$ che serve come altro input per $h(t)$ e influisce sulla previsione $y(t)$, proprio come lo stato nascosto in un HMM determina la distribuzione di probabilità delle emissioni $x(t)$. Alcune differenze chiave tra gli HMM standard e le reti neurali ricorrenti sono che le reti neurali ricorrenti possono modellare mappature più complesse per l'evoluzione dello stato e le previsioni (dagli stati alle osservazioni) grazie all'uso delle reti neurali. Inoltre, l'evoluzione dello stato nelle reti neurali ricorrenti è deterministica, e non stocastica come negli HMM.

Ci sono altre architetture per le reti neurali ricorrenti; per esempio, possiamo collegare l'output $y(t - 1)$ al nodo nascosto $h(t)$. Progettare un'architettura di rete è come progettare un edificio, l'architetto può dare libero sfogo alla propria immaginazione, sempre nei vincoli della ragionevolezza. L'addestramento di una rete neurale ricorrente, tuttavia, non è un problema banale. È afflitto dai problemi del gradiente che svanisce e che esplode (Goodfellow et al., 2016). Il nocciolo di questi problemi è che nella backpropagation usando la regola della catena, dobbiamo moltiplicare insieme molti gradienti. Se la rete è molto profonda, essenzialmente state calcolando g^n dove g è il gradiente e n è il numero di layer (la profondità) della rete. Se $g < 1$, $g^n \rightarrow 0$ quando $n \rightarrow \infty$, cioè svanisce. Se $g > 1$, $g^n \rightarrow \infty$ quando $n \rightarrow \infty$, cioè esplode. In entrambi i casi, la discesa stocastica del gradiente fallisce. Le soluzioni a questi problemi hanno dato origine a LSTM (*Long Short-Term Memory*), GRU (*Gated Recurrent Unit*), meccanismo di attenzione e transformer. Quest'ultimo è diventato la chiave per costruire grandi modelli linguistici come GPT-4. Quindi, ora abbiamo chiuso il cerchio. Per un esempio pratico dell'uso di LSTM per previsioni finanziarie, vedi Sirignano e Cont (2018) e

Kolm et al. (2021). Maggiori dettagli sul meccanismo di attenzione e sui modelli transformer saranno forniti nel Capitolo 6.

Esercizio #13

- Chiedete a ChatGPT di utilizzare Tensorflow o PyTorch per creare una rete neurale ricorrente di questo tipo e per addestrarla su SPY e MSFT come nell'esercizio precedente. L'addestramento ha successo? Se sì, quali sono le sue performance sul set di test? Sono migliori di quelle del precedente semplice MLP?
- Chiedete a ChatGPT, "Puoi trovare un articolo che descriva la previsione di serie temporali finanziarie usando LSTM e che includa un repository GitHub?". Poi provate a implementare il modello.

Apprendimento per rinforzo profondo

Nell'apprendimento non supervisionato, non abbiamo la risposta corretta (l'etichetta) di un problema (come "Qual è il modo migliore per raggruppare le azioni?"). Nell'apprendimento supervisionato, abbiamo la risposta a posteriori a un problema (come "Quale sarà il rendimento di MSFT?"). Esiste, però, anche una situazione intermedia: non conosciamo la risposta corretta, ma sappiamo che una risposta è migliore di un'altra per un problema (come "Quale allocazione di asset è la migliore tra queste 100 azioni?").

L'apprendimento per rinforzo (RL, *Reinforcement Learning*) è stato progettato proprio per gestire questa situazione. Il problema tipico è una mossa negli scacchi: non si sa se una mossa è la "migliore" possibile, ma si sa che una mossa è cattiva se porta immediatamente allo scacco matto. Allo stesso modo, il problema tipico, nel trading, è l'ottimizzazione del portafoglio. Non si sa quale sia l'allocazione ottimale di asset per un portafoglio di attività (sebbene il portafoglio

di tangenza dall'ottimizzazione media-varianza, a posteriori, possa avvicinarsi molto), ma si sa che un'allocation è migliore di un'altra se a posteriori genera un indice di Sharpe migliore. Quindi, l'apprendimento per rinforzo è stato applicato a molti problemi di ottimizzazione finanziaria, come l'ottimizzazione del portafoglio e le esecuzioni ottimali (Dixon et al., 2020; Sood et al., 2023). Negli ultimi anni, i migliori schemi di apprendimento per rinforzo spesso utilizzano reti neurali profonde per apprendere automaticamente le azioni ottimali, invece di crearle manualmente o aggiornare una tabella di azioni, e allora parliamo di apprendimento per rinforzo profondo (DRL, *Deep Reinforcement Learning*).

Il DRL può sembrare complicato, ma possiamo implementarlo in un modo molto simile all'apprendimento supervisionato. Ricordiamo che addestriamo una rete neurale tramite apprendimento supervisionato presentandole dei campioni (mini-batch) dotati di etichette, e le etichette vengono utilizzate per calcolare la log-verosimiglianza (negativa) del campione, dati i parametri della rete (pesi). La log-verosimiglianza negativa può essere vista come una funzione di costo J da minimizzare tramite discesa del gradiente e backpropagation, aggiustando iterativamente i parametri della rete mentre le presentiamo i campioni di addestramento. Nel caso del DRL, non abbiamo né le etichette, né la loro log-verosimiglianza, ma possiamo comunque calcolare il costo J dato l'output di una rete. Per esempio, nell'ottimizzazione del portafoglio, l'output di una rete possono essere i pesi degli asset in un portafoglio, e il costo può essere il negativo dell'indice di Sharpe per i prossimi 12 mesi di quel portafoglio, dati tali pesi. Seguendo Cong et al. (2021), se stiamo pianificando il ribilanciamento mensile di un portafoglio, possiamo campionare ogni fine mese t nel set di addestramento e alimentare le feature ottenibili al tempo t come input in una rete, dove il costo J è

impostato al negativo dell'indice di Sharpe da t a $t + 12$ mesi. Una volta che i parametri della rete sono addestrati, essa può produrre in output i pesi ottimali per ogni mese futuro, dati nuove feature di input. Naturalmente, la rete può essere riaddestrata periodicamente con nuovi dati anche in avanti. Cong et al. (2021) hanno applicato questo schema a un portafoglio azionario e hanno scoperto che genera un indice di Sharpe prossimo a 2 nel set di test.

Nell'apprendimento supervisionato, il risultato del modello predittivo è chiamato etichetta o target, ma nell'apprendimento per rinforzo, il risultato del modello di ottimizzazione è chiamato "ricompensa". In generale, il costo J è la somma di tutte le ricompense guadagnate su tutti i campioni. Potreste aver sentito che c'è una differenza tra apprendimento per rinforzo online e offline. L'apprendimento per rinforzo offline è necessario quando non possiamo simulare quale ricompensa avremmo guadagnato con un'azione differente, e tutto ciò su cui possiamo eseguire l'addestramento sono le azioni storiche effettivamente intraprese e le loro ricompense (per esempio, non sapremmo se un consumatore acquisterebbe effettivamente un prodotto se ne abbassassimo il prezzo nell'uso dell'apprendimento per rinforzo per l'ottimizzazione dei prezzi). Tuttavia, nelle applicazioni finanziarie come l'ottimizzazione del portafoglio, possiamo sicuramente simulare ciò che sarebbe successo se avessimo preso una decisione di trading differente, poiché le ricompense si basano sui prezzi storici. Quindi, quasi tutto l'apprendimento per rinforzo finanziario è apprendimento online.

Notate che ci sono altri modi, più complicati, per implementare il DRL, come il *Q-learning* o la PPO (*Proximal Policy Optimization*). Questi metodi affrontano la possibilità che a volte vogliamo esplorare azioni meno immediatamente gratificanti per ottimizzare le

ricompense a lungo termine (il compromesso fra sfruttamento ed esplorazione), e a volte vogliamo evitare che vi siano comportamenti instabili durante l'addestramento. In particolare, la funzione che produce in output un'azione sulla base dello stato corrente (chiamata "funzione di policy") potrebbe invece produrre in output una distribuzione di probabilità delle possibili azioni (o, più spesso, i parametri della distribuzione di probabilità). Per ulteriori dettagli, vedi l'eccellente corso online gratuito sull'apprendimento per rinforzo di Hugging Face (<https://huggingface.co/learn/deep-rl-course>). Inoltre, nel caso avessimo dato l'impressione che l'apprendimento per rinforzo basato sul machine learning possa essere implementato solo tramite reti neurali profonde, l'articolo di Fuhrer et al. (2024), che utilizza alberi potenziati con gradiente, dovrebbe risolvere il dubbio.

Ingegneria dei dati

Senza dati, non c'è nulla da cui il machine learning possa imparare. Anche con i dati, è necessaria una notevole competenza per verificarne la correttezza. Evidenzieremo quindi alcuni problemi comuni con i dati finanziari. La maggior parte di questi problemi è relativa all'addestramento, non all'inferenza/previsione.

Identificatori univoci delle società

Questo è principalmente un problema relativo ai dati associati a singole società e ai dati a esse correlati (per esempio, i prezzi delle loro azioni, obbligazioni e opzioni, informazioni fondamentali sulla società stessa e così via). Fornitori di dati differenti spesso utilizzano identificatori differenti per queste società. Anche all'interno di un unico file di dati, l'identificatore di una società potrebbe cambiare

nel tempo, e due società potrebbero utilizzare lo stesso identificatore in momenti diversi. Questo è particolarmente vero se l'identificatore è solo il simbolo ticker. Dovremmo utilizzare dati che hanno un file "securities master": dovrebbe avere un identificatore univoco per un titolo per tutta la sua storia, e incrociare altri identificatori comunemente usati come simboli ticker, cusip e così via per diversi periodi, presenti in vari altri dataset. L'obiettivo è quello di collegare tutte le informazioni relative a questo titolo da più dataset per la stessa data, e usarle come feature o etichette per il machine learning.

Un problema correlato è come dovremmo gestire i titoli altamente correlati. Con questo non intendiamo azioni Exxon vs Chevron. Intendiamo società che emettono azioni multi-classe come BF.A e BF.B. Sono quasi identici nei loro rendimenti giornalieri. Costruendo le feature dai loro rendimenti, otterremmo feature altamente ridondanti. Naturalmente, le feature ridondanti non sarebbero necessariamente un problema se utilizziamo algoritmi di machine learning come CART, o tecniche di selezione delle feature come cMDA. Ma comunque, è sempre opportuno eliminare tali duplicazioni, per evitare confusione a valle nel processo di investimento.

Aggiustamenti per dividendi e split

In generale, dovremmo trattare come feature o etichette i rendimenti di un asset, non i prezzi. Il livello di prezzo è arbitrario e non aggiunge molta informazione. Inoltre, non è stazionario e le feature legate a un prezzo in un dato momento generalmente non possono essere utilizzate per prevedere le etichette in un altro momento futuro. Potreste pensare che se c'è uno split azionario, potrebbe essere un segnale rialzista, e uno split inverso un segnale ribassista, ma potete codificare tali eventi come feature categoriche,

separatamente. Allo stesso modo, potreste pensare che un grande aggiustamento dei prezzi dovuto a un grande dividendo sia rialzista. Ma, ancora, potete usare i dividendi yield come feature autonoma. Non c'è bisogno di inferirlo dal livello di prezzo. Abbiamo già discusso nel Capitolo 1 come calcolare i rendimenti tenendo conto di tali azioni societarie. L'argomento viene ulteriormente trattato in Chan (2021).

Bias di sopravvivenza

Il bias di sopravvivenza è un problema dei dati che non potete scoprire esaminando i dati stessi. Come sapere che cosa manca? Per esempio, per mettere alla prova una strategia di trading su un portafoglio di azioni con capitalizzazione di mercato superiore a 1 miliardo di dollari, avreste bisogno di un database storico della capitalizzazione di mercato di tutte le azioni esistenti in un certo momento (un database “Point-in-Time”), non solo un database della capitalizzazione di mercato delle azioni che esistono ora. Allo stesso modo, se volete mettere alla prova questa strategia sulle azioni S&P 500, avrete bisogno di un database storico dei componenti dell'indice S&P 500. Le conseguenze dell'avere bias di sopravvivenza nei dati di addestramento e quali fornitori di dati forniscono dati privi di bias di sopravvivenza sono discusse in Chan (2021).

Bias di anticipazione

Se qualcuno ha sbagliato a specificare il timestamp di una *feature* ottenuta al tempo t indicando t' , precedente a t , abbiamo un bias di anticipazione se la usiamo per predire un'etichetta al tempo τ tale che $\tau < t$. Analogamente, se qualcuno ha sbagliato a specificare il timestamp di un'etichetta ottenuta al tempo t indicando t' , successivo a t , abbiamo un bias di anticipazione se usiamo una feature ottenuta

al tempo τ tale che $t < \tau$. Ci sono modi sottili in cui il bias di anticipazione può insinuarsi nei dati, anche quelli di fornitori di dati rinomati. Per esempio, se una società ha annunciato utili e al tempo t , ma successivamente ha revisionato gli utili a e' al tempo $t' > t$, i dati non devono contenere la voce (t, e') , sebbene alcuni dataset sicuramente lo facciano. Allo stesso modo, se i dati *non-farm payroll* p sono stati annunciati dal Bureau of Labor Statistics al tempo t , ma successivamente revisionati a p' al tempo $t' > t$, i dati non devono contenere solo la voce revisionata (t, p') .

C'è un altro caso interessante di bias di anticipazione nei dati. Alcune strategie di trading guidate da eventi si basano sulla data di annuncio degli utili t (non sull'utile annunciato stesso). Per esempio, la strategia può specificare che “Alla chiusura del mercato prima dell'annuncio degli utili al tempo t , compra l'azione se ha un rendimento giornaliero positivo”. Se eseguite il backtest di tale strategia basata sul calendario degli annunci degli utili che potete scaricare da Yahoo Finance o simili, probabilmente avrete introdotto un bias di anticipazione. Il motivo è che la maggior parte dei database cattura solo la data effettiva di annuncio, non la data di annuncio prevista, mentre la data prevista è ciò che serve per il backtest di una strategia guidata da eventi come quella descritta. Spesso le società cambiano la data o ora di annuncio prevista, ed è cruciale catturare esattamente la data/ora di annuncio prevista. Per quanto ne sappiamo, tali dati sono disponibili solo da Wall Street Horizon.

Un altro caso interessante di bias di anticipazione si verifica nei dati dei prezzi futures. Ma in questo caso, la colpa risiede nell'interpretazione dei dati, non nei dati stessi. Per esempio, il nostro fornitore di dati futures preferito CSIdata fornisce un prezzo di liquidazione dei futures così come il prezzo spot del sottostante. Per essere specifici, diciamo che il 2019-06-06, il contratto giugno 2019

del future a 6M ha prezzo di liquidazione = 0.05082 e prezzo spot = 0.05086. È corretto concludere che questo future è in backwardation (vedi Chan, 2013 per la definizione) in quella data? No! Il prezzo di liquidazione dei futures è ottenuto alle 15:00 ET al CME, mentre il prezzo spot è ottenuto (da CSIdata) alle 17:00 ET.

Forse pensate che utilizzando dati intraday, possiamo superare in sicurezza eventuali problemi di bias di anticipazione con futures o altri dati di prezzo. Potrebbe non essere sufficiente. Ipotizziamo di voler fare il backtest di una strategia di spread di calendario futures utilizzando barre di trading OHLC a 1 minuto. Ancora non funzionerà, perché l'ultimo trade di una "gamba" di solito non avviene allo stesso tempo dell'ultimo trade dell'altra "gamba". Fare il backtest di tali barre di trading in uno spread di calendario ipotizza essenzialmente che conosciate il prezzo del trade successivo quando eseguite il trade precedente. Per superare questo problema, dobbiamo utilizzare le migliori offerte invece dei prezzi di trade per formare la barra minuto su cui è calcolato lo spread, poiché l'ultimo quote di ogni gamba rimane valido fino alla fine di quella barra minuto.

Tali sottigliezze sfuggiranno quasi sicuramente a un data scientist, a meno che non sia esperto nella scienza dei dati finanziari. Anche se il bias di anticipazione è un problema riconosciuto, spesso è difficile trovare dati privi di bias. E anche se li trovate, la loro storia tende a essere breve. Tuttavia, sono gli unici dati adatti a essere utilizzati per il machine learning.

Ingegneria delle feature

Abbiamo già scritto che, a differenza di quanto avviene con le applicazioni del machine learning nel campo di immagini, video, parlato e testi, i dati grezzi dei prezzi non funzionano come feature per le applicazioni, nel trading. A volte è necessaria una pesante

ingegneria delle feature. In primo luogo, dobbiamo assicurarci che una feature sia stazionaria. Questo, di solito, significa differenziare temporalmente una serie temporale invece di utilizzare i suoi livelli, come prendere le differenze del logaritmo di una serie di prezzi per creare una serie di rendimenti logaritmici, prendere le differenze percentuali di una serie del PIL per creare una serie di variazioni del PIL e così via. In secondo luogo, dobbiamo unire correttamente serie temporali che presentano frequenze di campionamento differenti. In terzo luogo, ci sono diversi modi per gestire feature simili da un ampio pool di asset simili. Infine, se acquistiamo tali feature da un fornitore e non le creiamo noi stessi dai dati grezzi, sono necessarie ulteriori precauzioni. Quindi approfondiremo questi problemi.

Stazionarietà

È ovvio che non possiamo utilizzare i livelli di prezzo delle azioni come feature o etichette. Il prezzo potrebbe essere 10 dollari nel 2000 e 10.000 dollari nel 2024 se aggiustiamo per gli split, e semplicemente non possiamo addestrare un modello di machine learning basato sui prezzi del 2000 e aspettarci che le inferenze funzionino con i prezzi di oggi. Dovremmo convertire il prezzo in serie di rendimenti. Ma ci sono serie temporali più subdole, per le quali non è chiaro se siano stazionarie. Per esempio, lo spread tra i futures sul petrolio greggio WTI e Brent è stazionario? Lo spread tra MSFT e GOOG è stazionario? Per scoprirlo, dobbiamo eseguire un test di stazionarietà, come il test *Dickey-Fuller aumentato* (ADF). Se non lo sono, possiamo differenziare temporalmente questi spread o costruire uno spread che sia stazionario fin dall'inizio. Ernie ha discusso molti modi per costruire una serie temporale stazionaria da più serie temporali non stazionarie in Chan (2013).

Unione di serie temporali con frequenze differenti

La frequenza con cui il modello di machine learning dovrebbe fare previsioni spetta all'utente. Per esempio, vuole negoziare a intervalli giornalieri o settimanali? Una volta determinata questa frequenza, dobbiamo convertire i dati delle feature che potrebbero avere frequenza inferiore o superiore per conformarli alla frequenza delle etichette. Il primo problema, di solito, viene risolto riempiendo le feature con `NaN` quando non hanno un valore aggiornato, o riempiendole in avanti con il loro ultimo valore aggiornato. La prima soluzione potrebbe essere appropriata se una feature è di un evento categorico come “Il non-farm payroll è stato annunciato ieri?”; la seconda potrebbe essere appropriata se è una feature numerica come “Qual era la più recente variazione percentuale del non-farm payroll?”. Quale metodo adottare deve essere determinato caso per caso. Quando invece la frequenza delle feature è superiore alla frequenza delle etichette, dobbiamo utilizzare un metodo di down-sampling. Per default si utilizza semplicemente l'ultimo valore aggiornato della feature appena prima del timestamp dell'etichetta. Tuttavia, in alcune situazioni potrebbe essere meglio utilizzare una sorta di media tra tutti i valori durante gli intervalli tra due etichette consecutive. Per esempio, se come dati grezzi stiamo utilizzando una serie di prezzi, la prima scelta significa campionare gli ultimi prezzi, mentre la seconda scelta potrebbe significare prendere il prezzo medio ponderato per volume (VWAP). Dopo aver effettuato il down-sampling di questi prezzi, possiamo procedere a calcolare le feature di rendimento. Oltre a calcolare le medie, altre tecniche di down-sampling includono il prendere statistiche d'ordine, come il max o il min (il massimo o il minimo di una barra dei prezzi). Possono essere applicate molte altre tecniche di elaborazione del segnale, vedi López

de Prado (2018) per maggiori dettagli. Ma come eseguiamo il down-sampling delle feature categoriche? Per eseguire il down-sampling di feature categoriche binarie con valori 0 o 1, possiamo trattarle come feature a valori reali e applicare le tecniche appena esposte oppure possiamo applicare gli operatori logici AND, OR, XOR. Per eseguire il down-sampling di una feature categorica multi-classe, possiamo prima convertirla in una feature binaria tramite la codifica one-hot. Per esempio, se abbiamo una feature categorica a tre classi, può essere convertita in tre feature binarie, la prima indica se la feature originale è di classe A, la seconda se è di classe B e la terza se è di classe C. Una feature categorica di k classi può quindi essere rappresentata da k feature binarie.

Feature di serie temporali vs feature cross-sectional

Se vogliamo costruire un modello di machine learning per prevedere i rendimenti o altri attributi di diversi asset nella stessa categoria (per esempio, azioni S&P 500), possiamo sicuramente utilizzare feature specifiche per ciascuno di questi asset (per esempio, l'earning yield passato di un'azione). Queste feature sono chiamate feature cross-sectional, perché attraversano la *sezione trasversale* (appunto) dell'universo degli asset. Tuttavia, che cosa succede se vogliamo costruire un modello di machine learning per prevedere i rendimenti o altri attributi del portafoglio di asset nel suo insieme? Mentre potreste avere ragione nel pensare che gli utili di MSFT prefigurino i rendimenti di SPY, in generale ciascuna di queste feature cross-sectional sarà un predittore estremamente debole, ed è molto inefficiente usarne così tante come feature. È meglio aggregare queste feature trasversali in qualcosa come una feature di sintesi, simile al modo in cui eseguiamo il down-sampling delle feature ad

alta frequenza in feature a bassa frequenza (vedi paragrafo precedente). Il metodo di aggregazione non è nuovo: si basa sul documento seminale di Fama-French Fama e French (1995). Prima ordiniamo gli asset in base alla feature cross-sectional in questione (per esempio, earning yield). Poi formiamo un portafoglio long-short basato su questa classifica (per esempio, il portafoglio long contiene azioni nel decile superiore di earning yield, e il portafoglio short contiene azioni nel decile inferiore). Questo è chiamato portafoglio “hedge”. La feature della serie temporale risultante è semplicemente il rendimento di questo portafoglio su un qualche periodo passato. In questo modo, abbiamo compresso 500 feature cross-sectional (per un portafoglio S&P 500) in una sola feature della serie temporale. Le più famose di queste feature di serie temporali sono nientemeno che i fattori Fama-French discussi nel Capitolo 2. Per maggiori dettagli, vedi Pik et. al. (2025).

Validazione di feature di terze parti

A volte una serie di feature è creata da un fornitore di terze parti e non da noi stessi, dai dati grezzi. Come facciamo a sapere se la feature è stata calcolata correttamente? Caso emblematico: una volta abbiamo acquistato da un fornitore dei dati di sentiment storici e in tempo reale su articoli di notizie finanziarie. I punteggi sembravano ragionevoli, e abbiamo proceduto ad addestrare un modello di machine learning utilizzando questi dati. Stranamente, dopo un ulteriore esame dei dati delle serie temporali prendendo il loro min e max, abbiamo trovato il grafico mostrato nella Figura 3.12.

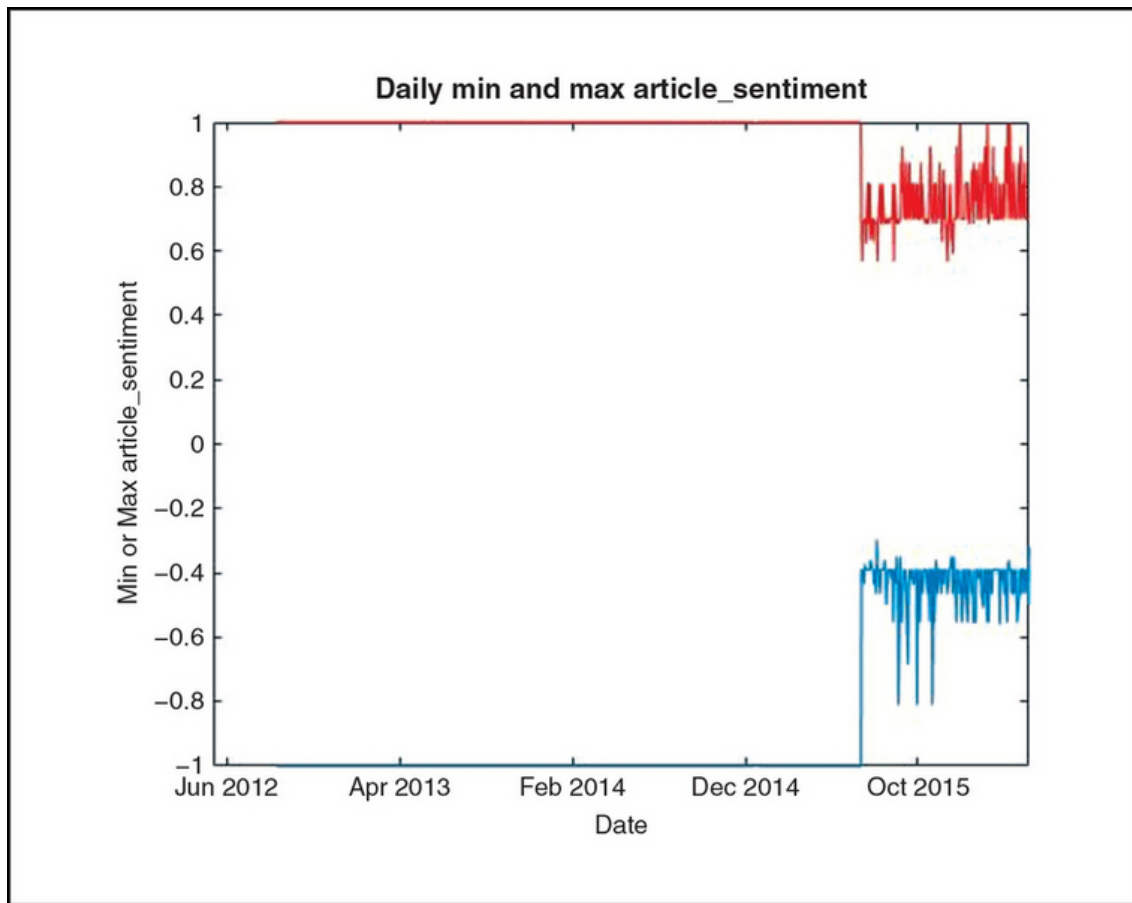


Figura 3.12 Grafico dei punteggi di sentiment minimi e massimi giornalieri che mostrano break strutturali nel processo di generazione dei dati.

C'è un chiaro break strutturale nei dati nel 2015. Il fornitore ha ammesso di aver cambiato metodologia per calcolare i punteggi del sentiment. Naturalmente, ciò ha reso inutili tutti i dati storici precedenti al 2015. Ma ancor più pericolosamente, come facciamo a sapere che non cambieranno nuovamente la metodologia in futuro?

La lezione è che è necessaria un'attenta e ampia analisi esplorativa dei dati (EDA) quando si acquistano feature da terze parti, idealmente prima di impegnarsi in un contratto annuale.

L'intelligenza artificiale generativa come generatore di feature

L'esempio che abbiamo citato sopra, sull'affidarsi a fornitori di terze parti per le feature, è illuminante. Ai vecchi tempi, abbiamo acquistato quei dati di sentiment perché non era facile elaborare dei dati non strutturati, come il testo, e trasformarli in dati numerici/strutturati, come i punteggi di sentiment. Tuttavia, con l'intelligenza artificiale generativa, tutto è diventato molto più facile. Usiamo questo come esempio di intelligenza artificiale generativa nel Capitolo 9: come calcolare i punteggi di sentiment, e se la GenIA aggiunge valore a un processo di investimento.

Naturalmente, che le feature siano create da noi stessi o da un fornitore, possiamo sempre utilizzare l'intelligenza artificiale generativa per condurre un'analisi esplorativa dei dati senza ricorrere al codice. Inoltre, come dimostrato nei capitoli precedenti, possiamo chiedere all'intelligenza artificiale generativa anche di generare il codice di machine learning che permetta di unire le feature e le etichette in un file, di costruire un modello di machine learning e di valutarne le performance.

Classificazione e selezione dell'importanza delle feature

Uno dei benefici della regolarizzazione L1 è che alcune feature saranno eliminate del tutto come input di un modello di machine learning. Allo stesso modo, tipicamente i modelli basati su alberi selezionano solo un piccolo sottoinsieme di feature per i loro nodi, prima di fermarsi alle foglie. Se siamo solamente preoccupati della performance sul set di test, tipicamente non abbiamo bisogno di ridurre ulteriormente il numero di feature per l'algoritmo CART.

Tuttavia, a volte c'è bisogno di classificare e selezionare l'importanza delle feature, non per migliorare le performance, ma per migliorare l'interpretabilità del modello per gli esseri umani e per la validazione delle feature (cioè, per i controlli di sanità mentale). La potenza del machine learning consiste nel fatto che possiamo iniziare con un set di feature ampio, senza pensare al fatto che queste feature siano insignificanti o ridondanti. Ma questa inclusività va a scapito dell'interpretabilità, perché gli esseri umani non avranno idea di come il modello giunge alle sue inferenze. La classifica dell'importanza delle feature allevia tale problema. Per un esempio finanziario reale, vedi la Tabella 7 di Chan et al. (2023). A volte, quando la feature “prima classificata” non sembra avere alcun senso, dovremmo investigare se essa ha incorporato un bias di anticipazione o se è corrotta in altro modo. Quindi, la classifica dell'importanza delle feature serve anche come metodo di validazione.

Come abbiamo discusso in precedenza, il cMDA è il nostro metodo preferito per la classificazione dell'importanza delle feature. Abbiamo spiegato la parte c, *clustering*; la parte MDA (*Mean Decrease in Accuracy*) è più nota. L'idea è che, se permutiamo casualmente i campioni di una feature (cioè permutiamo casualmente le righe di una colonna di una matrice di dati), il modello dovrebbe generare una minore accuratezza predittiva su un set di validazione (abbiamo specificato set di validazione, perché è quasi garantito che si generi una minore accuratezza predittiva se corrompiamo i dati del set di addestramento, indipendentemente dall'effettiva importanza di quella feature). Se eseguiamo più permutazioni e calcoliamo la media della riduzione dell'accuratezza (o della log-verosimiglianza), possiamo utilizzare questa media come metrica per classificare l'importanza delle feature. Maggiore è la riduzione, più importante è la feature.

Parte II

Modelli generativi profondi per il trading e la gestione degli asset

In questa Parte

- [Capitolo 4 - L'intelligenza artificiale generativa](#)
- [Capitolo 5 - Modelli autoregressivi profondi per la modellazione di sequenze](#)
- [Capitolo 6 - Modelli profondi a variabili latenti](#)
- [Capitolo 7 - Modelli di flusso](#)
- [Capitolo 8 - Reti generative antagoniste](#)
- [Capitolo 9 - Sfruttare gli LLM per l'analisi del sentiment nel trading](#)
- [Capitolo 10 - Efficienza dell'inferenza](#)
- [Capitolo 11 - Postfazione](#)

Capitolo 4

L'intelligenza artificiale generativa

I modelli generativi rappresentano un'affascinante frontiera nell'avanzamento dell'intelligenza artificiale. Questi modelli hanno la capacità di comprendere e generare dati complessi e a elevata dimensionalità. Questa capacità non solo apre la porta a numerose applicazioni pratiche, ma offre anche una visione più profonda della struttura sottostante dei dati. Questo capitolo esplora l'essenza dei modelli generativi, le loro applicazioni e le differenze che presentano con i modelli discriminativi con cui molti professionisti della finanza quantitativa hanno familiarità. Forniremo anche una tassonomia per orientarci nel panorama dei modelli generativi (Figura 4.1).

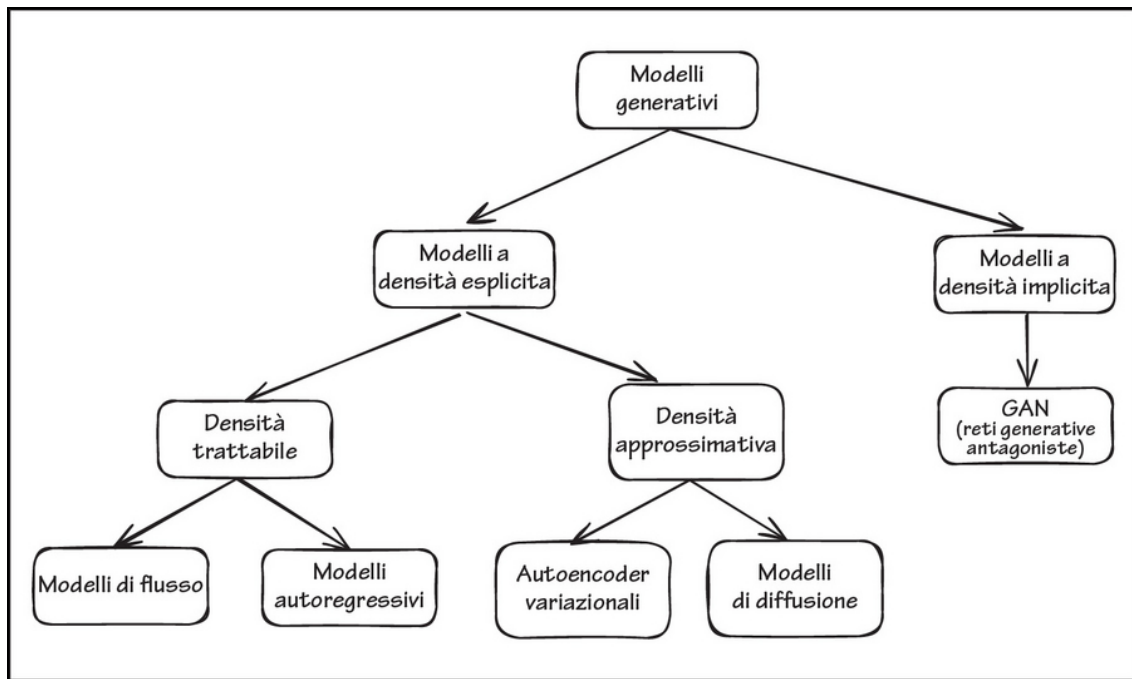


Figura 4.1 Grafico ad albero che rappresenta la tassonomia dei modelli.

Perché usare i modelli generativi

I modelli generativi sono fondamentali per diverse ragioni. In primo luogo, consentono alle macchine di generare nuove istanze di dati che somigliano ai dati incontrati durante l'addestramento, apprendendo efficacemente la distribuzione del dataset. Questa capacità va oltre la mera replicazione dei dati osservati; si tratta di comprendere la struttura sottostante dei dati, per generare nuove istanze. In secondo luogo, i modelli generativi facilitano i compiti di apprendimento non supervisionato, in cui la macchina apprende modelli comuni nei dati senza bisogno di dati etichettati. Questo aspetto è cruciale per sfruttare le grandi quantità di dati non etichettati disponibili oggi ed evitare il costoso e lento processo di etichettatura manuale dei dati.

Infine, essi aiutano a comprendere la distribuzione dei dati, rendendo possibili applicazioni per il rilevamento di anomalie, la compressione dei dati e altro.

Vogliamo dissuadere il lettore dall'idea che l'intelligenza artificiale generativa sia utile solo per comprendere dati non strutturati come testi o immagini, o che sia utile solo per generare dati sintetici per scopi di simulazione. Come illustra il prossimo paragrafo, può essere cruciale per le consuete attività di apprendimento supervisionato che utilizzano modelli discriminativi, cui la maggior parte dei professionisti della finanza quantitativa è principalmente interessata.

Differenze rispetto ai modelli discriminativi

Al cuore del machine learning vi è la capacità di modellare e fare previsioni dai dati. I modelli discriminativi e generativi rappresentano due approcci a questa ricerca, differenziati dal tipo di distribuzione di probabilità che mirano ad apprendere dai dati.

I modelli discriminativi si concentrano sull'apprendere la probabilità di un'etichetta o target y dato un input x , ovvero $p(y|x)$. Questi modelli eccellono in compiti di classificazione o regressione, dove l'obiettivo è quello di prevedere correttamente il target. Sono strettamente associati a compiti di apprendimento supervisionato, dove i modelli vengono addestrati su un dataset etichettato, imparando a mappare gli input su determinati target. Nonostante la loro potente capacità di determinare il target più probabile per un dato input, una limitazione critica dei modelli discriminativi è la loro incapacità di valutare la verosimiglianza $p(x)$ dei dati di input stessi.

Questo limite di fondo li rende suscettibili ad attacchi avversari o da interpretazioni errate di dati nuovi o anomali.

I modelli generativi, al contrario, mirano a modellare la distribuzione dei dati stessi, apprendendo la distribuzione di probabilità congiunta $p(y,x)$ o la distribuzione di probabilità incondizionata $p(x)$. I modelli generativi incondizionati tentano di apprendere la distribuzione di probabilità dell'input $p(x)$, cosa che consente loro di generare nuove istanze di dati che somigliano ai dati di addestramento. Come esempio, ipotizziamo di voler modellare i dati dei rendimenti dell'ETF SPY e di decidere di approssimarne la distribuzione usando un GMM (*Gaussian Mixture Model*), uno dei modelli generativi più semplici. Questo modello non solo ci permette di generare nuove istanze di dati, ma, come vedremo nel Capitolo 6, cattura anche alcune strutture sottostanti dei dati che possono essere utilizzate per applicazioni come il rilevamento dei regimi. In generale, i modelli generativi forniscono una comprensione più profonda della struttura dei dati, rendendo possibili applicazioni come la generazione di dati e il rilevamento delle anomalie (outlier).

I modelli generativi condizionali rappresentano un approccio ibrido, imparando a generare dati x condizionati su valori specifici di y , come etichette di classe, denotati come $p(x|y)$. Questi modelli combinano la capacità generativa di produrre nuove istanze con il potere discriminativo di condizionare queste istanze su determinati attributi o classi, rendendoli più robusti agli attacchi avversari o alle interpretazioni errate di dati nuovi o anomali. Come esempio tipico, si possono considerare modelli come ChatGPT, che possono generare nuovi dati come testi, immagini o codice sorgente, condizionati dai prompt di input forniti dall'utente.

Per illustrare la differenza tra modelli discriminativi e generativi, consideriamo il classico esempio del classificatore cane-gatto. Un

modello discriminativo apprende $p(y|x)$: la probabilità di un'etichetta y (cane o gatto) data un'immagine x . Al momento dell'inferenza, prende un'immagine di input e prevede la probabilità che appartenga a ciascuna delle classi. Un modello generativo condizionale, al contrario, apprende $p(x|y)$, modellando la probabilità di un'immagine x data una specifica classe y .

Questo modello può anche essere usato come classificatore, un “classificatore generativo”, confrontando la probabilità di un'immagine x sotto $p(x|y = \text{cane})$ rispetto a $p(x|y = \text{gatto})$. Tuttavia, a differenza dei modelli discriminativi, può anche generare nuove immagini condizionate su un'etichetta. Per esempio, se condizionato sul cane ($y = \text{cane}$), il modello può generare un'immagine di un cane campionando da $p(x|y = \text{cane})$.

In questo senso, un modello discriminativo mappa l'immagine di input x sull'etichetta della classe di output y , mentre un modello generativo opera al contrario, generando x da un dato y . Inoltre, nella modellazione generativa, y non è ristretto alle etichette di classe: può anche rappresentare attributi o feature.

La distinzione tra modelli generativi e discriminativi, che inizialmente può sembrare un po' avulsa dal discorso, è radicata nella regola di Bayes della probabilità. Questa relazione è illustrata dall'equazione

$$p(x|y) = \frac{p(x)p(y|x)}{p(y)}$$

(4.1)

dove $p(x)$ rappresenta il modello generativo incondizionato, $p(y|x)$ corrisponde al modello discriminativo, $p(y)$ è la probabilità marginale sulle etichette, e $p(x|y)$ è un modello condizionale di x dato y . Combinando i punti di forza dei modelli generativi incondizionati e

dei modelli discriminativi, possiamo costruire modelli generativi condizionali, che possono essere utilizzati non solo per la generazione condizionale di dati, ma anche per compiti di apprendimento supervisionato, come la classificazione.

Come possiamo utilizzarli?

Questi modelli sono progettati con alcune capacità o obiettivi chiave in mente, che aprono un mondo di possibilità per l'innovazione, la creatività e l'efficienza in vari domini. Di seguito riportiamo un riepilogo di queste caratteristiche, che esploreremo ulteriormente nei prossimi capitoli.

Stima della densità di probabilità

Questi modelli possono approssimare distribuzioni di dati a elevata dimensionalità e multimodali. Una volta che abbiamo accesso a questo oggetto così versatile, possiamo sviluppare e risolvere un'ampia gamma di applicazioni. Esempi notevoli includono la previsione di serie temporali multivariate, come illustrato nel Capitolo 3, applicazioni del Value at Risk (VaR) per la gestione del rischio, rilevamento di anomalie e compiti di classificazione.

I professionisti della finanza hanno già familiarità con le distribuzioni di probabilità a elevata dimensionalità dei rendimenti, che tuttavia sono tipicamente modellate come versioni multivariate di modelli parametrici semplici come la gaussiana multivariata o la distribuzione t . Modelli più avanzati come le copule tentano di catturare la natura non lineare della co-dipendenza tra le variabili, ma fondamentalmente sono ancora modelli parametrici semplici. L'intelligenza artificiale generativa, al contrario, offre distribuzioni

molto più complesse, non lineari, che non potrebbero essere espresse in forme parametriche semplici.

Generare nuovi dati

Al cuore delle capacità dei modelli generativi c'è il loro potere di “creare”. Possono produrre oggetti altamente complessi e a elevata dimensionalità, come immagini realistiche, video, testi, codice, serie temporali multivariate e così via. Questo li rende preziosi in applicazioni come l'intrattenimento, il marketing e strumenti di produttività e gestione del rischio. La loro capacità di generare dati è spesso indistinguibile da quella umana. Questa capacità va oltre il semplice creare contenuti.

I nuovi dati generati possono essere utilizzati per molti scopi.

- *Migliorare la qualità e la disponibilità dei dati:* i dati generati possono risolvere problemi relativi alla scarsità e alla qualità dei dati. Questi modelli possono estendere i dataset esistenti, specialmente in campi in cui la raccolta di dati è costosa o addirittura impraticabile, creando nuove istanze di dati e migliorando i dataset per prepararli ai successivi compiti di apprendimento.
- *Facilitare la scoperta e la presa di decisioni:* una delle applicazioni più promettenti dei modelli generativi risiede nella loro capacità di simulare e prevedere strutture e fenomeni complessi e a elevata dimensionalità. Nell'industria farmaceutica, per esempio, questi modelli possono prevedere delle strutture molecolari, che potenzialmente possono portare alla scoperta di farmaci rivoluzionari. Inoltre, possono simulare ambienti per l'addestramento dell'intelligenza artificiale, per applicazioni di apprendimento per rinforzo che notoriamente sono avidi di dati, riducendo così la necessità di raccogliere dati

reali, che è costosa e dispendiosa in termini di tempo, accelerando la ricerca e lo sviluppo in campi come la guida autonoma e la pianificazione urbana.

Scoprire nuove rappresentazioni dei dati

L'obiettivo di trovare nuove rappresentazioni dei dati, o “apprendimento delle rappresentazioni”, consiste nello scoprire un modo efficace per rappresentare i dati. Idealmente, questa nuova rappresentazione dovrebbe possedere proprietà desiderabili, come una dimensionalità inferiore rispetto allo spazio dei dati o assi di variazione “indipendenti”. Questa nuova rappresentazione serve come forma compressa dei dati, che può essere utilizzata per risolvere compiti successivi come la previsione, il rilevamento di anomalie o la generazione di dati (vedi Torralba et al., [2024]).

Molti professionisti della finanza hanno già familiarità con l'apprendimento delle rappresentazioni, travestito con altri nomi: modelli fattoriali, PCA e così via. Ne abbiamo già discusso nel Capitolo 3, ma li inquadreremo nuovamente all'interno del framework dell'intelligenza artificiale generativa. Questo ci permette di migliorarli, potenzialmente, a livello fondamentale.

Il potere dell'apprendimento delle rappresentazioni consente di combinare più modelli come “mattoni” per sviluppare applicazioni complesse. Esploreremo vari esempi del loro utilizzo, in particolare nei Capitoli 9 e 10, dove vengono impiegati per addestrare un modello specializzato nell'analisi del sentiment per dati finanziari.

I modelli generativi e ChatGPT

Per capire meglio come possiamo utilizzare i modelli generativi per calcolare le densità di probabilità e per generare nuovi dati,

esaminiamo un esempio concreto di elaborazione del linguaggio naturale (NLP, *Natural Language Processing*). Per semplicità, metteremo da parte i dettagli specifici del modello e ci concentreremo su un modello pre-costruito, ChatGPT, esaminando le sue capacità di apprendimento in termini di stima, campionamento e rappresentazione.

Consideriamo una sequenza di variabili casuali, dove ogni variabile rappresenta una parola o token in una frase (per questo esempio, ignoreremo la distinzione tra parole e token, che sarà trattata nel Capitolo 5). Ogni elemento nella sequenza prende valori da un vocabolario finito: un insieme discreto. In ChatGPT-4o, per esempio, il vocabolario è costituito da 16.384 token unici (vedi la documentazione OpenAI su

<https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>), che includono parole comuni, frammenti di parole e caratteri.

Un modello probabilistico per una sequenza è definito dalla sua distribuzione di probabilità congiunta: $p(x_1, \dots, x_N)$.

Modellazione del linguaggio

Modellare questa probabilità congiunta su una sequenza di parole/token è un compito di modellazione del linguaggio, nell'elaborazione del linguaggio naturale. Uno dei modi più diretti per esprimere questa distribuzione congiunta consiste nell'applicare la regola del prodotto della probabilità:

$$\begin{aligned}
 p(x_1, \dots, x_N) = & p(x_1) \\
 & \times p(x_2 | x_1) \\
 & \times p(x_3 | x_1, x_2) \\
 & \dots \\
 & \times p(x_N | \mathbf{x}_{1:N-1})
 \end{aligned}$$

(4.2)

I modelli che sfruttano il lato destro dell'Equazione 4.2 per rappresentare le distribuzioni congiunte come una sequenza di probabilità condizionali come questa sono noti come *modelli autoregressivi*. ChatGPT è un'istanza di un modello autoregressivo, per la precisione un modello basato su transformer, che esploreremo nel Capitolo 5.

Dall'Equazione 4.2, vediamo che ogni probabilità condizionale modella la verosimiglianza della parola/token successiva date quelle precedenti. In altre parole, ChatGPT esegue previsioni probabilistiche “a un passo avanti”, dove ogni previsione si basa sui valori passati o sul contesto passato.

Una domanda chiave è: quante parole precedenti può considerare ChatGPT quando fa le sue previsioni?

Questo è definito da quella che, nell'elaborazione del linguaggio naturale, è chiamata *finestra di contesto*. Al momento della stesura di questo testo, la finestra di contesto di ChatGPT-4o è di 128.000 token, il che significa che può analizzare fino a 128.000 token precedenti per stimare la probabilità del successivo

(<https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>). Questo migliora significativamente la sua capacità di catturare le dipendenze a lungo raggio nel testo.

Campionamento: in che modo i modelli generativi creano nuovi dati

I modelli che si occupano di linguaggio naturale sono modelli generativi, il che significa che possono generare nuove sequenze campionando dalla distribuzione di probabilità appresa. Il processo segue questi passaggi.

- Genera il primo token, $x_1 \sim p(x_1)$.
- Genera il secondo token, $x_2 \sim p(x_2 | x_1)$.
- Continua a generare token, $x_N \sim p(x_N | x_1, \dots, x_{N-1})$.

Poiché il campionamento è sequenziale, ogni token appena generato dipende da quelli generati in precedenza, rendendo il processo relativamente lento per i modelli autoregressivi.

Generazione condizionale del linguaggio: una domanda a ChatGPT

Proprio come possiamo modellare una sequenza di parole/token con un modello per il linguaggio naturale, possiamo anche generare testo condizionatamente. Cioè, possiamo condizionare l'output generato su un dato contesto, come può essere una domanda o un prompt, che denoteremo nel seguente esempio come y . Questo processo viene utilizzato nelle risposte ai prompt, dove il modello genera risposte condizionate dall'input dell'utente.

Matematicamente, la versione condizionale della regola del prodotto è data da:

$$\begin{aligned}
p(x_1, \dots, x_N | y) &= p(x_1 | y) \\
&\times p(x_2 | x_1, y) \\
&\times p(x_3 | x_1, x_2, y) \\
&\dots \\
&\times p(x_N | \mathbf{x}_{1:N-1}, y)
\end{aligned}$$

(4.3)

Il campionamento segue lo stesso processo sequenziale descritto nell'Equazione 4.3.

- Genera il primo token $x_1 \sim p(x_1 | y)$.
- Genera il secondo token $x_2 \sim p(x_2 | x_1, y)$.
- Continua a generare token $x_N \sim p(x_N | x_1, \dots, x_{N-1}, y)$

Per illustrare questo concetto, chiediamo a ChatGPT di generare risposte umoristiche ai nostri prompt. Ora, consideriamo il seguente prompt dell'utente.

Prompt dell'utente

Che cosa hanno in comune Mark Zuckerberg ed Elon Musk?

Quando ChatGPT genera una risposta a un dato prompt, segue il processo delineato sopra, prevedendo una parola alla volta in base alle probabilità che assegna ai possibili token successivi condizionati ai token precedenti e al prompt (y).

Ecco una risposta di esempio generata da ChatGPT-4o.

Risposta di ChatGPT

Hanno entrambi abbastanza soldi per comprare un piccolo paese ... o almeno una pizzeria davvero niente male!

Ora, come ha fatto ChatGPT-4o a generare questa risposta?

Per comprendere il suo output, abbiamo utilizzato l'API OpenAI per accedere sia alla sequenza di token generati da ChatGPT-4o sia

alle probabilità associate a ciascun token presente nel vocabolario. Poiché ChatGPT-4o ha una dimensione del vocabolario di 16.384 token unici, sarebbe impossibile visualizzare l'intera distribuzione di probabilità sulla parola successiva. Invece, visualizziamo solo i primi 20 token più probabili in base alle loro probabilità.

Esecuzione passo-passo: in quale modo il modello genera una risposta.

1. Calcolo delle probabilità condizionate della prima parola.
 - Condizionato dal prompt, il modello calcola le probabilità della prima parola.
 - Come mostra la Figura 4.2, le scelte migliori includono “They”, “They’re” e “Both”.
 - Il modello campiona “They” (che in questo caso risulta essere la parola più probabile).

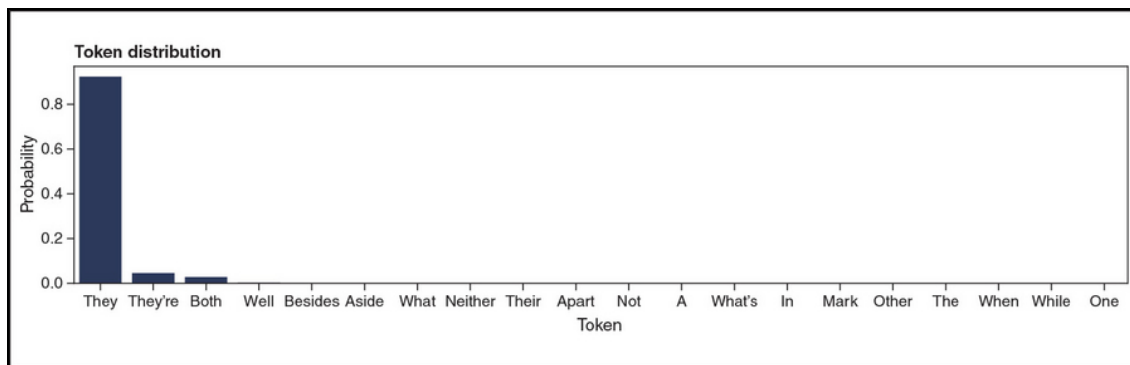


Figura 4.2 Grafico della probabilità condizionata del primo token dato il prompt y : $p(x_1|y)$.

2. Calcolo delle probabilità condizionate della seconda parola.
 - Successivamente, il modello prevede le probabilità della seconda parola, condizionate dal prompt e dalla prima parola (“They”).

- Come mostra la Figura 4.3, le scelte migliori includono “both”.
- Il modello campiona “both”.

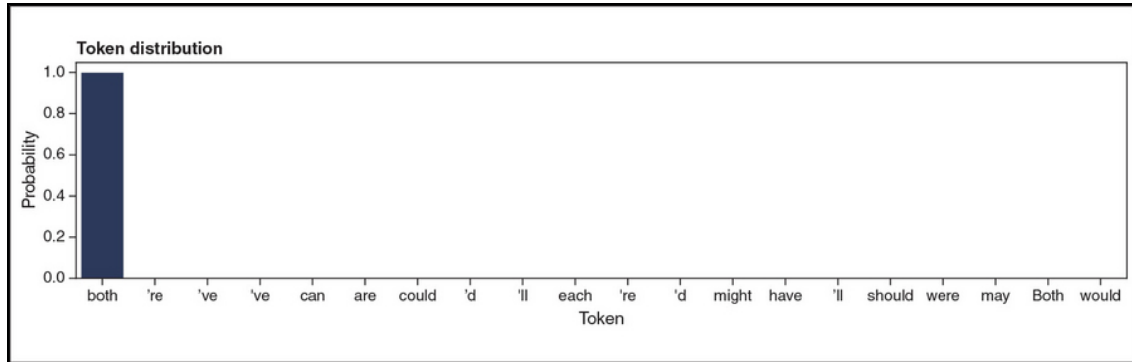


Figura 4.3 Grafico della probabilità condizionata del secondo token dato il prompt y e il token precedente x_1 : $p(x_2 | x_1, y)$.

- Calcolo delle probabilità condizionate della terza parola.
 - Ora, il modello prevede la parola successiva, condizionata dal prompt, da “They” e da “both”.
 - Come mostra la Figura 4.4, le parole più probabili sono “have”, “now” e “think.”
 - Il modello campiona “have”.
- Calcolo delle probabilità condizionate della quarta parola.
 - Il modello poi prevede la parola successiva, dato il prompt e poi “They”, “both” e “have”.
 - Come mostra la Figura 4.5, le scelte migliori includono “enough”, “more”, “a” e “billions”.
 - Il modello campiona “enough”.

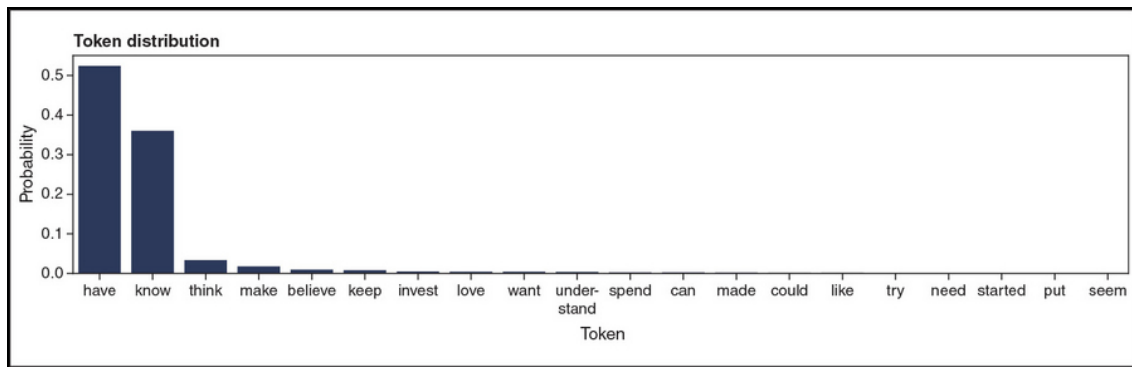


Figura 4.4 Grafico della probabilità condizionata del terzo token dato il prompt y e i token precedenti x_1, x_2 : $p(x_3 | x_1, x_2, y)$.

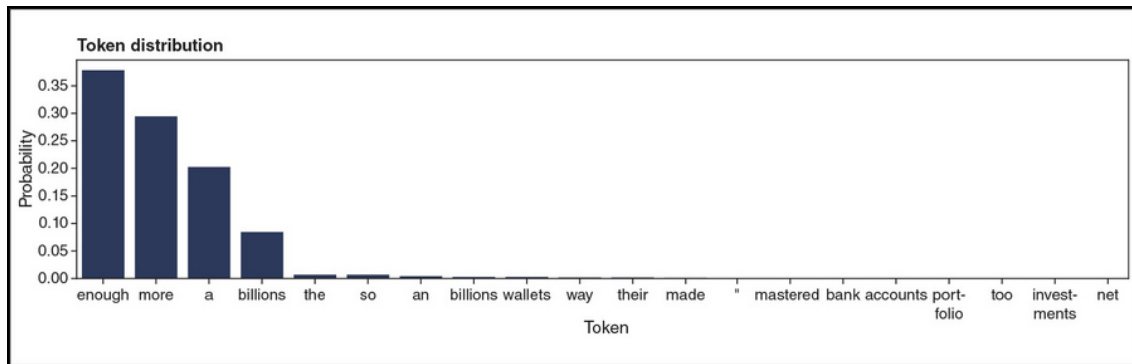


Figura 4.5 Grafico della probabilità condizionata del quarto token dato il prompt y e i token precedenti x_1, x_2, x_3 : $p(x_4 | x_1, x_2, x_3, y)$.

5. Calcolo delle probabilità condizionate della quinta parola.
 - Il modello continua, prevedendo la parola successiva dato il prompt e “They”, “both”, “have” ed “enough”.
 - Come mostra la Figura 4.6, le parole più probabili includono “money”, “wealth” e “cash”.
 - Il modello campiona “money.”

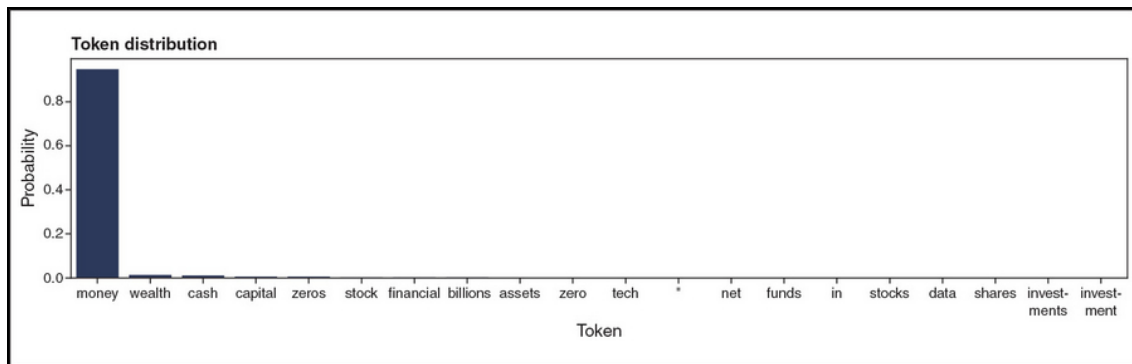


Figura 4.6 Grafico della probabilità condizionata del quarto token dato il prompt y e i token precedenti x_1, x_2, x_3, x_4 : $p(x_5 | x_1, x_2, x_3, x_4, y)$.

6. Il processo continua.

- Il modello prevede sequenzialmente ogni parola in base a quelle precedenti e al prompt, fino a formare una risposta coerente.

Ogni parola generata dipende dalla distribuzione di probabilità stimata dal modello in quel passo. In questo caso, la generazione dei dati viene eseguita utilizzando la funzione di densità di probabilità. Questo processo illustra come i modelli generativi autoregressivi eseguono il campionamento in uno scenario reale.

Oltre che alla generazione di dati, la stima della densità di probabilità può essere applicata anche alla previsione del token successivo più probabile o per rispondere a domande sulla probabilità di parole rare in un testo. Questo diventa ancora più rilevante quando si passa dal modellare una sequenza di parole al modellare una sequenza di rendimenti finanziari, un argomento che esploreremo nel Capitolo 5. Queste tecniche possono essere utilizzate per applicazioni come il rilevamento di anomalie o per calcolare metriche finanziarie chiave come il VaR (*Value at Risk*).

Come potete vedere, i modelli generativi possono funzionare come simulatori, che sono ampiamente utilizzati in finanza. Un esempio noto è il pricing delle opzioni, specialmente per payoff di opzioni utilizzando tecniche Monte Carlo.

Nell'esempio qui sopra, il processo di campionamento seleziona la parola/token dal vocabolario che ha la probabilità più elevata a ogni passo. Tuttavia, in pratica, ChatGPT può utilizzare strategie di campionamento più complesse (come *temperature scaling* o il campionamento top-k/top-p) per generare risposte più diversificate e contestualmente rilevanti.

Se preferite le videolezioni, su YouTube potete trovare un tutorial basato su questo paragrafo, presentato dagli Autori del libro. Cercate: *Trading Using LLM | Generative AI & Sentiment Analysis for Finance | Webinar* sul canale YouTube QuantInsti Quantitative Learning.

Come potreste aver notato con questo esempio, anche se questi modelli sono altamente efficaci nel generare nuovi dati, sorge una domanda chiave: come possiamo valutare la qualità dei dati generati se ogni esecuzione produce risultati differenti? Come abbiamo visto nel Capitolo 3, la risposta è chiara per applicazioni come la previsione, dove potete confrontare i valori reali realizzati con le previsioni utilizzando una metrica scelta, come l'errore quadratico medio.

Nella nostra esperienza nello sviluppo di applicazioni, uno dei modi migliori per valutare la qualità dei dati generati consiste nell'utilizzarli per addestrare un modello a valle per risolvere un compito specifico. Combinando i dati reali con i nuovi dati generati, potete addestrare il modello e valutare se le sue prestazioni migliorano. Tuttavia, la letteratura offre molte metriche per valutare

la qualità dei campioni generati, e ne useremo diverse in tutto il libro per valutare le prestazioni di questi modelli nella generazione di dati.

Nel notebook allegato [*Synthetic Data Evaluation – Notebook*], esamineremo un esempio per spiegare le diverse metriche che useremo nei nostri esperimenti, nel libro.

Questo esempio evidenzia come ChatGPT, un modello autoregressivo, genera il testo modellando distribuzioni di probabilità su sequenze di token.

Qualche parola sull'apprendimento delle rappresentazioni con ChatGPT

Come accennato in precedenza, l'apprendimento delle rappresentazioni riguarda la scoperta di modi efficaci per codificare i dati che facilitano la risoluzione di altri compiti a valle, come la previsione, il rilevamento di anomalie, la generazione di dati e altri. Sorprendentemente, ChatGPT ha anche la capacità di calcolare nuove rappresentazioni dei dati di input sotto forma di vettori, i vettori di embedding nella letteratura dedicata all'elaborazione del linguaggio naturale. Questi embedding catturano informazioni semantiche su parole e concetti, permettendoci di manipolare e “comprendere il linguaggio” eseguendo operazioni matematiche con questi oggetti. Esploreremo gli embedding in maggior dettaglio nel Capitolo 5.

OpenAI dà accesso agli embedding di ChatGPT attraverso la sua API, consentendo agli utenti di utilizzare queste rappresentazioni per risolvere applicazioni.

Come caso d'uso, immaginate di essere un investitore o un trader discrezionale che traccia un insieme di società in cui detiene posizioni nel portafoglio o di voler considerare un'allocazione. Diciamo che analizzate frequentemente documenti 10-K e 10-Q, trascrizioni degli utili e report di analisi finanziaria per prendere

decisioni informate. Tuttavia, recuperare informazioni rilevanti da più documenti può richiedere molto tempo.

Non sarebbe utile avere un modo per estrarre conoscenze specifiche da queste fonti? Per esempio, immaginate di porre una domanda come “Quali sono i rischi associati alla Società XYZ?”, e di avere un sistema che recuperi documenti rilevanti dove sono menzionati questi rischi. Per risolvere questo tipo di problema, potrebbero venire in soccorso le rappresentazioni testuali.

Per affrontare questo problema, il primo passo consiste nel costruire un database di conoscenze utilizzando vettori di embedding. Ecco come.

1. *Suddividere i documenti*: suddividete ogni report finanziario o documento in parti gestibili (per esempio frasi, paragrafi o pagine).
2. *Generare i vettori di embedding*: usate l’API Embeddings di OpenAI per generare le rappresentazioni vettoriali per ogni parte del documento. Memorizzate questi embedding in un database.
3. *Eseguire la ricerca semantica*: quando un utente invia una query (per esempio “Quali sono i rischi associati alla Società XYZ?”), convertite la query in un embedding. Questo embedding viene poi confrontato con gli embedding memorizzati nel database utilizzando una misura di similarità, come la similarità del coseno. Quindi, potete recuperare le prime K parti di testo più rilevanti, in modo da ottenere le informazioni contestualmente più rilevanti per la vostra query.

Questo approccio di recupero basato su embedding è un blocco costitutivo fondamentale per tecniche più avanzate come la RAG (*Retrieval-Augmented Generation*), vedi

<https://help.openai.com/en/articles/8868588-retrieval-augmented->

[generation-rag-and-semantic-search-for-gpts](#). La RAG consente ai modelli LLM (*Large Language Model*) di generare risposte basate su conoscenze esterne, arricchendo la query di input, cosa che vi permette di personalizzare un LLM per il vostro specifico dominio senza addestrarlo. Maggiori dettagli su questo argomento possono essere trovati nel Capitolo 10.

Modellazione ibrida: combinare modelli generativi e discriminativi

Per illustrare i vantaggi derivanti dal combinare questi due approcci di modellazione, consideriamo un esempio dal trading di materie prime.

Il trading di materie prime implica l'acquisto e la vendita di materiali grezzi come prodotti agricoli, petrolio, gas naturale e così via. Il successo in questo campo spesso dipende dalla nostra capacità di prevedere con precisione l'offerta e la domanda. Per esempio, nel trading energetico, gli investitori e i trader vorrebbero monitorare la produzione globale di petrolio e gas, che è distribuita in diverse regioni del mondo.

Un'applicazione particolarmente utile del machine learning in questo campo consiste nello sfruttare le immagini satellitari per monitorare le operazioni di fracking; nello specifico, rilevare i *well pad* (i siti nei quali avviene il processo di perforazione). Questo caso d'uso è menzionato nel blog di SkyFi, all'indirizzo

<https://skyfi.com/en/blog/synmax-case-study>. SkyFi è un'azienda che fornisce dati di immagini satellitari e analisi.

Ipotizziamo di avere accesso a questi dati e di voler valutare come possono essere utilizzati per prendere decisioni informate di trading. Un'idea potrebbe essere quella di costruire un classificatore che

prende come input delle immagini satellitari e restituisce la probabilità della presenza di un well pad. Questo modello potrebbe essere utilizzato per stimare il numero di well pad attivi, fornendo informazioni sulla produzione di gas naturale e consentendo a investitori e hedge fund di migliorare il loro processo decisionale. Nel caso di strategie completamente sistematiche, per esempio, l'output di tale modello potrebbe essere incorporato in strategie di trading più complesse, basate su fattori.

Ora, immaginiamo il processo di costruzione effettiva di questo modello. Indicheremo l'immagine di input come x e l'etichetta binaria come y , che indica se è presente un well pad. Un approccio comune consiste nell'utilizzare un modello discriminativo che stima direttamente $p(y|x)$: la probabilità della presenza di un well pad data l'immagine satellitare.

Possiamo supporre di aver raccolto un dataset di alta qualità di immagini satellitari, tutte etichettate con precisione da annotatori umani o con un altro metodo affidabile. Ipotizziamo di risolvere questo problema utilizzando un modello a rete neurale profonda (*Deep Neural Network*). Il modello prende un'immagine di input x e restituisce direttamente la probabilità y , che indica la presenza di un well pad. Se il modello è efficace sui dati di test (utilizzando metriche di classificazione comuni come AUC, precisione o richiamo) potremmo essere tentati di pensare che il problema sia risolto.

Tuttavia, l'impiego del modello nel mondo reale presenta delle problematiche. Un problema chiave è che del rumore inatteso negli input, causato da molteplici fattori, potrebbe influenzare le prestazioni. Consideriamo quello che succede quando il nostro modello incontra un'immagine che è significativamente diversa da qualsiasi cosa abbia visto prima. Che cosa restituirà $p(y|x)$?

E se un'immagine di input fosse stata influenzata da condizioni meteorologiche avverse o altri fattori ambientali? Per esempio, le nuvole, che, secondo l'*Earth Observatory* della NASA (<https://earthobservatory.nasa.gov/features/CloudsGallery>), coprono in qualsiasi momento circa il 70% della superficie terrestre, possono nascondere alcuni elementi chiave. Che cosa succede se un'immagine di input è completamente coperta da nuvole? Oppure, invece delle nuvole, che cosa succede se una crisi geopolitica distrugge importanti condutture o se disastri naturali alterano drasticamente il paesaggio? In ambienti di produzione che operano in tempo reale, le immagini di input possono differire drasticamente da quelle nel nostro set di addestramento. Che cosa restituirà $p(y|x)$ in questi casi?

Idealmente, in situazioni di estrema incertezza, vorremmo che il nostro modello dicesse semplicemente qualcosa come “Non lo so” e restituisse $p(y = \text{presenza-di-well-pad}|x) = 0.5$. In realtà, spesso non è così, e il modello potrebbe comunque produrre una previsione eccessivamente confidente.

Questo problema è ben documentato nel deep learning, nel quale l'incertezza rimane un'area di ricerca molto attiva. Questi modelli spesso raggiungono prestazioni impressionanti, ma sono incapaci di riconoscere “quando non sanno”. Per esempio, nell'articolo '*Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*' di Nguyen, Yosinski e Clune (2015), gli autori hanno condotto esperimenti in cui a classificatori all'avanguardia sono state presentate immagini di input costituite da rumore casuale, senza alcun senso per gli esseri umani. Nonostante l'input fosse rumore casuale, le reti neurali profonde addestrate su ImageNet hanno classificato questi input con una confidenza $\geq 99.6\%$ come appartenenti a categorie di oggetti familiari, vedi Nguyen, Yosinski e Clune (2015). Questo esperimento fornisce prove empiriche su come

i moderni modelli di deep learning possano essere eccessivamente confidenti nelle loro previsioni, anche quando l'input è palesemente puro rumore. Idealmente, vorremmo che il nostro modello desse risposte coerenti a tali input.

Questo problema sorge quando la distribuzione nei dati di addestramento differisce significativamente dalla distribuzione incontrata al momento dell'inferenza, un problema noto in letteratura come problema OOD (*Out-Of-Distribution*) o *dataset shift*. Nella letteratura delle serie temporali e della finanza, questo è comunemente indicato come problema di *non stazionarietà*. Senza entrare troppo nei dettagli, due scenari comuni incontrati nella pratica sono il *covariate shift*, dove la distribuzione delle feature di input $p(x)$ cambia al momento dell'inferenza, e l'*open set recognition*, che si verifica quando al momento dell'inferenza appaiono nuove classi, non viste in precedenza e che non erano presenti durante l'addestramento.

Una limitazione fondamentale del modellare solo $p(y|x)$ è che, poiché rappresenta una distribuzione di probabilità su y , il modello è costretto ad assegnare masse di probabilità sulle classi conosciute, indipendentemente dall'input. In uno scenario di classificazione binaria, per esempio, il modello deve soddisfare:

$$p(y = \text{presenza-di-well-pad}|x) + p(y = \text{assenza-di-well-pad}|x) = 1$$

Come nota a margine riguardo alle immagini satellitari e alle crisi geopolitiche, SkyFi menziona nel suo blog come le immagini satellitari siano state utilizzate nel 2022 per monitorare le infrastrutture energetiche in Ucraina, incluse le centrali elettriche, le condutture e le strutture di stoccaggio del carburante.

Bene, torniamo al nostro problema. La domanda è: come possiamo risolvere o mitigare problemi come questo?

Una soluzione comune utilizzata nella pratica consiste nell'introdurre una nuova classe, tipicamente chiamata *classe sconosciuta (unknown)*: le immagini che non appartengono a nessuna delle categorie target vengono etichettate come sconosciute. Tuttavia, questo introduce un'altra sfida: quante classi che non appartengono alle categorie target dovremmo etichettare come "sconosciute"? Quanti esempi dovremmo includere per ognuna di queste classi per ottenere una buona copertura di tutti i possibili input OOD (*Out-Of-Distribution*)? Affidandoci solo a questo metodo, potremmo ritrovarci ad aggiungere infinite nuove categorie senza però risolvere veramente il problema. Quindi, possiamo affrontare questo problema utilizzando un approccio differente?

Un approccio per risolvere questo problema potrebbe essere combinare modelli generativi e discriminativi. Il problema principale con l'approccio precedente è che, al momento dell'addestramento, la nuova immagine non era presente nel set di addestramento. Ma che cosa succederebbe se avessimo un modo per quantificare la probabilità che x provenga dalla distribuzione di addestramento? E se, invece di modellare $p(y|x)$, modellassimo $p(x, y)$?

Applicando la regola del prodotto, sappiamo che: $p(x, y) = p(x)p(y|x)$.

Ciò significa che, se potessimo costruire un modello per $p(x)$ (un modello generativo per le feature di input) e un modello discriminativo distinto per $p(y|x)$, potremmo potenzialmente trovare una soluzione al nostro problema. Poiché il nostro modello $p(x)$ mira ad approssimare la distribuzione di probabilità dei dati, dovrebbe assegnare un'elevata densità di probabilità alle immagini (input) familiari e una bassa densità di probabilità a quelle che sono *Out Of Distribution*. Di conseguenza, gli input assenti dal set di

addestramento dovrebbero avere una bassa densità di probabilità sotto $p(x)$.

Per un input OOD x durante l'addestramento, $p(x)$ dovrebbe essere basso, il che a sua volta attenua il prodotto $p(y|x)p(x)$, rendendo basso anch'esso. Possiamo quindi utilizzare una soglia di probabilità per decidere se fidarci della previsione. Per esempio, se $p(y|x)p(x)$ scende al di sotto di un certo valore, l'incertezza diventa eccessiva, e il modello dovrebbe evitare di fare una previsione confidente.

Pertanto, questo approccio di modellazione ibrida ci consente di combinare il nostro modello discriminativo preferito con un modello generativo, migliorando la robustezza agli input OOD e la stima dell'incertezza, offrendo così un migliore processo decisionale.

Un altro caso d'uso, distinto dall'esempio delle immagini satellitari, è la nota applicazione per prevedere i regimi di mercato. Ipotizziamo di aver raccolto un dataset costituito da coppie di input x e target y , dove y è una variabile binaria che indica se il prossimo stato del mercato corrisponde a un mercato rialzista (*bull*) o ribassista (*bear*). Gli input x rappresentano feature o “caratteristiche” dello stato attuale del mercato che riteniamo forniscano informazioni utili per prevedere il prossimo stato del regime di mercato.

In modo simile al caso precedente sulle immagini satellitari, se raccogliamo dati rilevanti solo per i mercati bull e bear – ciascuno con caratteristiche specifiche codificate in x – che cosa succede quando incontriamo un nuovo input che è molto diverso da quelli osservati in entrambi i regimi? Come prima, se modelliamo $p(y|x)$, il sistema fornirà comunque una previsione ad alta confidenza per bull o bear, anche se l'input non individua bene nessuna delle due categorie.

Tuttavia, modellare $p(x)$ potrebbe offrire ulteriori informazioni sulla novità di un dato input. Questo può essere utile in due modi.

1. Come sistema autonomo di rilevamento delle anomalie, identificando quegli input che si discostano significativamente dalla distribuzione di addestramento.
2. In combinazione con un modello discriminativo, migliorando le previsioni tenendo conto degli input OOD e dell'incertezza.

Ora, la domanda è: se prendiamo questo approccio, come possiamo costruire $p(x)$ e $p(y|x)$? Ci sono molti modi per farlo, ma di seguito descriviamo i due approcci principali.

Una soluzione semplice consiste nel costruirli in modo indipendente e poi tentare di combinarli come $p(y|x)p(x)$. Tuttavia, non c'è garanzia che ciò funzioni. Nella pratica, a volte funziona molto bene, ma altre volte no. Questo approccio potrebbe anche richiedere numerose iterazioni su entrambi i modelli, poiché per definizione vengono addestrati in modo indipendente e disaccoppiati l'uno dall'altro.

Possiamo considerare un esempio legato alla finanza. Potreste costruire un modello di trading basato su machine learning per prevedere i rendimenti di alcuni futures sulla volatilità. Ma sorge un grande rischio di overfitting, a causa dei dati storici limitati. Potrebbe quindi essere vantaggioso apprendere dalla vasta quantità di dati storici provenienti da tutte le serie finanziarie, futures, azioni, FX o altro, e usarli come probabilità a priori dei rendimenti dei futures sulla volatilità. Come input per questo modello di apprendimento supervisionato potremmo anche utilizzare alcuni indicatori tecnici. Calcolare la probabilità marginale di questi indicatori potrebbe informarci se certi input sono anomalie, outlier, e potrebbero non essere generalizzabili.

Un altro modo per affrontare questo problema consiste nel rendere i modelli dipendenti l'uno dall'altro. Questa idea è esplorata nell'articolo *Hybrid Models with Deep and Invertible Features* di

Nalisnick et al. (2019). L'idea chiave consiste nell'utilizzare un modello a variabili latenti che non solo cattura la distribuzione dei dati, ma serve anche come input per il modello discriminativo. In questo modo, entrambi i modelli sono strettamente accoppiati e dipendono dalla stessa rappresentazione latente. Ora, l'addestramento di $p(x)$ e $p(y|x)$ è accoppiato.

Nell'articolo, si utilizza un modello di flusso (*flow model*) per $p(x)$ (un tipo di modellazione generativa che approfondiremo nel Capitolo 7) e un GLM (*Generalized Linear Model*) per $p(y|x)$. Questa configurazione consente di applicare l'approccio a problemi generali sia di classificazione sia di regressione.

Tassonomia dei modelli generativi

I modelli generativi possono essere ampiamente categorizzati in due tipi principali in base a come modellano la distribuzione dei dati: modelli a densità esplicita e modelli a densità implicita. I *modelli a densità esplicita* definiscono e calcolano esplicitamente la distribuzione di probabilità dei dati. Ecco alcuni esempi di modelli a densità esplicita.

- *Modelli autoregressivi*: questi modelli generano dati in sequenza, prevedendo ogni parte dei dati in base a ciò che è stato generato finora.
- *Autoencoder variazionali*: i VAE (*Variational Autoencoders*) imparano a codificare i dati di input in uno spazio a dimensionalità inferiore e poi li decodificano nuovamente, consentendo una generazione efficiente dei dati.
- *Modelli di flusso (Flow Models)*: questa famiglia di modelli trasforma distribuzioni semplici in distribuzioni complesse, mantenendo la capacità di calcolare l'esatta probabilità dei dati.

- *Modelli di diffusione*: questi modelli aggiungono gradualmente rumore ai dati e imparano a invertire questo processo, generando dati tramite *denoising* (in questo libro non abbiamo spazio per discutere i modelli di diffusione, ma citeremo dei riferimenti che li trattano).

Al contrario, i *modelli a densità implicita* mirano a generare dati catturando la distribuzione dei dati senza definirla esplicitamente. Un esempio ben noto di questi modelli è il seguente.

- Reti generative antagoniste: le GAN (*Generative Adversarial Networks*) prevedono un'architettura a doppia rete, in cui una rete genera dati e l'altra li valuta, portando a output altamente realistici.

La Figura 4.1 a inizio capitolo presenta una categorizzazione dei principali tipi di modelli generativi.

Ogni famiglia di modelli, come specificato in questa tassonomia, ha i suoi vantaggi e svantaggi per le abilità menzionate nel paragrafo precedente. Nel notebook di accompagnamento [*Model Family Comparison – Notebook*], presentiamo un elenco delle principali famiglie di modelli e confrontiamo la loro efficienza. Questa tabella è qualcosa che speriamo di aggiornare man mano che verranno rilasciati nuovi modelli o verranno sviluppate tecniche migliori.

Conclusioni

I modelli generativi rappresentano un cambiamento profondo nelle capacità dell'intelligenza artificiale. Apprendendo con lo scopo di comprendere e manipolare i dati, ci consentono di categorizzare il mondo così com'è e di immaginare e creare ciò che potrebbe essere. Dal creare arte al far avanzare le scoperte scientifiche, l'intelligenza artificiale generativa non è solo uno strumento per l'innovazione, ma

una base per i futuri progressi. Nel prossimo capitolo, approfondiremo ciascuna delle principali famiglie di modelli generativi.

Capitolo 5

Modelli autoregressivi profondi per la modellazione di sequenze

I modelli autoregressivi (AR, *Autoregressive*) sono una presenza familiare nella cassetta degli attrezzi dell'investitore quantitativo. Dopotutto, il dato finanziario più basilare è una serie di prezzi (una sequenza), e chi non vorrebbe prevedere il prezzo successivo? In Chan (2017), Ernie ha discusso i vari modelli predittivi autoregressivi lineari, come AR(p), ARMA(p,q), VAR, VEC e modelli a spazio di stati, come il filtro di Kalman. In questo capitolo, scopriremo se possiamo spingerci oltre, adottando tecniche non lineari del deep learning per un tale problema di “modellazione di sequenze”. Oltre alle previsioni, l'autoregressione può essere utilizzata anche per applicazioni di rilevamento delle anomalie e di generazione dei dati. Rientra nella categoria dei modelli a densità esplicita, come illustrato nella Figura 5.1. Ciò significa che propone un modello di densità di probabilità trattabile per approssimare la distribuzione dei dati p_{data} .

Con “dati sequenziali” si intende che i dati appaiono in sequenze in cui è presente un ordine naturale o artificiale. Questo principio consente la costruzione di modelli probabilistici per qualsiasi variabile nella sequenza, condizionandola ai suoi valori precedenti.

Questa caratteristica li rende utili per applicazioni che coinvolgono dati intrinsecamente ordinati, come i segnali audio, i fenomeni meteorologici (per esempio, i modelli di precipitazioni), i dati

testuali per applicazioni di elaborazione del linguaggio naturale o serie temporali create dall'uomo, come quelle presenti nei mercati finanziari. Va notato che i modelli autoregressivi si applicano anche a dataset privi di ordine intrinseco, come le immagini.

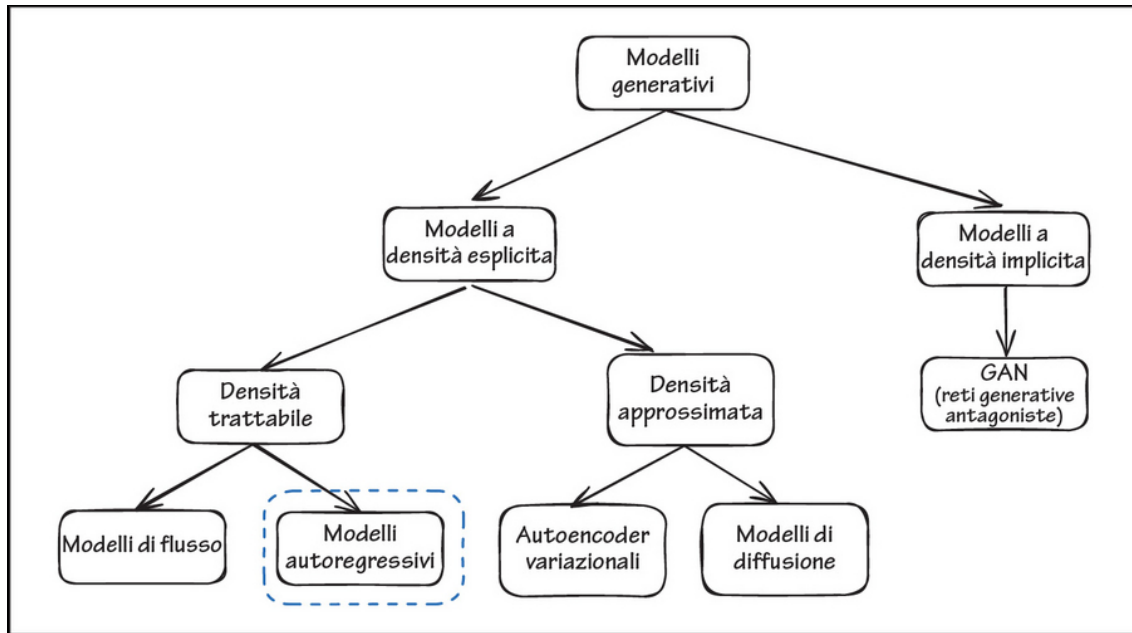


Figura 5.1 Grafico ad albero: i modelli autoregressivi nella tassonomia dei modelli.

Imponendo un ordine tra i pixel all'interno dell'immagine, i modelli autoregressivi possono generare immagini complete un pixel alla volta, come dimostrato con l'architettura PixelCNN, vedi Oord et al. (2016b) per maggiori dettagli.

Complessità della rappresentazione

Considerate una sequenza di variabili casuali $x_1, x_2, \dots, x_N$. Un modello probabilistico della sequenza è specificato dalla sua distribuzione di probabilità congiunta,

$$p_{X_1, X_2, \dots, X_N}(x_1, x_2, \dots, x_N).$$

Attraverso l'applicazione della regola del prodotto di probabilità, possiamo scomporre la distribuzione congiunta in un prodotto di condizionate:

$$\begin{aligned} p_{\mathbf{X}}(\mathbf{x}) = p_{X_1, X_2, \dots, X_N}(x_1, \dots, x_N) &= p_{X_1}(x_1) \\ &\times p_{X_2|X_1}(x_2|x_1) \\ &\times p_{X_3|X_1, X_2}(x_3|x_1, x_2) \\ &\dots \\ &\times p_{X_N|X_1, \dots, X_{N-1}}(x_N|\mathbf{x}_{1:N-1}) \end{aligned}$$

(5.1)

I modelli autoregressivi semplificano la modellazione di distribuzioni a elevata dimensionalità sfruttando il lato destro dell'Equazione 5.1, modellando una serie di distribuzioni condizionate.

Queste distribuzioni forniscono descrizioni probabilistiche per ogni variabile basate sui suoi predecessori nella sequenza, trasformando una complessa modellazione a elevata dimensionalità in una serie di probabilità condizionate monodimensionali più gestibili: più facili da rappresentare e stimare.

Nonostante la semplificazione dei modelli autoregressivi del problema a elevata dimensionalità in distribuzioni condizionate monodimensionali più gestibili, il costo computazionale dell'inferenza e dello spazio di memorizzazione rimane un problema. Per esempio, in serie temporali finanziarie come l'S&P500, la necessità di rendere discreti i valori in un numero finito di stati, diciamo cinque stati – una pratica comune nelle applicazioni finanziarie, come la competizione dell'hedge fund NUMERAI –

illustra la complessità del problema. Come esempio, considerate la modellazione della distribuzione condizionata $p_{x_{n+1}|x_{1:n}}(x_{n+1}|x_{1:n})$ dati i precedenti $n = 21$ valori nella sequenza. Pensate a questo esempio come a prevedere i rendimenti del giorno successivo di un titolo basandosi sull'ultimo mese di rendimenti giornalieri, come nelle applicazioni di trading, dove per la previsione vengono usati i dati dell'ultimo mese. Questo approccio richiederebbe un'enorme tabella di probabilità con 5^{21} voci, il che è chiaramente impraticabile. Invece di specificare le probabilità condizionate attraverso ricerche in tabelle, è necessario impiegare un modello parametrizzato. Tuttavia, anche con un approccio parametrizzato, modellare le interazioni tra tutte le variabili può produrre in un numero proibitivo di parametri. Pertanto, dobbiamo trovare altri modi per ridurre il numero di parametri.

Rappresentazione e riduzione della complessità

Un modo per ridurre il numero di parametri nel modello consiste nel fare ipotesi sulla struttura della distribuzione congiunta nell'Equazione 5.1. Queste ipotesi potrebbero includere specifiche sulle interazioni tra variabili, forme delle densità e così via.

Un approccio per ridurre la complessità consiste nell'imporre ipotesi di probabilità condizionata, tali per cui per modellare x_N non abbiamo bisogno di conoscere l'intera sequenza di valori precedenti $x_1, \dots, x_{N-1}$. Per esempio, potremmo dire che per modellare x_N , tutto ciò di cui abbiamo bisogno è il valore di x_{N-1} , suggerendo che tutta l'informazione rilevante è contenuta in questa variabile (non c'è perdita di generalità nell'ipotizzare che x_N dipenda solo dallo stato precedente x_{N-1} , perché potremmo semplicemente ridefinire uno stato

per includere informazioni da un periodo di lookback più lungo).
Matematicamente, esprimiamo la cosa come:

$$p_{x_n|x_1,\dots,x_{n-1}}(x_n|x_1,\dots,x_{n-1}) = p_{x_n|x_{n-1}}(x_n|x_{n-1})$$

nota come proprietà di Markov. Ne abbiamo accennato nel Capitolo 3 quando abbiamo parlato dell'*Hidden Markov Model*.

Un'altra tecnica implica la condivisione dei pesi (*weight sharing*), dove, nel caso dell'esempio precedente, la probabilità condizionata

$$p_{x_n|x_{n-1}}(x_n|x_{n-1})$$

è indipendente dalla posizione nella sequenza n , imponendo una sorta di invarianza o stazionarietà attraverso la sequenza.

Queste semplificazioni riducono anche l'espressività del modello, restringendo i tipi di distribuzioni di probabilità che possono essere rappresentate.

Per illustrare il concetto di indipendenza condizionata e stazionarietà del modello nella modellazione di sequenze, consideriamo un esempio pratico che coinvolge un modello autoregressivo di ordine 2, denotato come AR(2). Un modello AR(2) fattorizza la probabilità congiunta di una sequenza $x_1, \dots, x_N$ come segue:

$$p(x_1, \dots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_2, x_3)\dots p(x_N|x_{N-2}, x_{N-1}) \quad (5.2)$$

In questo scenario, ipotizziamo che x assuma valori nello spazio $\chi = \mathbb{R}$ e che possiamo modellare ulteriormente ogni probabilità condizionata usando una distribuzione gaussiana. Nello specifico, per ogni x_N , la probabilità condizionata $p(x_N|x_{N-2}, x_{N-1})$ è espressa come:

$$p(x_n|x_{n-2}, x_{n-1}) = \mathcal{N}(w_0 + w_1x_{n-1} + w_2x_{n-2}, \sigma^2) \quad (5.3)$$

Qui $\mathcal{N}(\dots, \sigma^2)$ denota una distribuzione gaussiana dove la media è determinata dalla combinazione lineare dei due valori precedenti nella sequenza e di un termine costante, e con varianza costante σ^2 . Come esempio di utilizzo di modelli autoregressivi lineari, il notebook di accompagnamento [*AR Notebook*] dimostra la previsione del tasso di crescita trimestrale del prodotto nazionale lordo (GNP) statunitense.

Questa ipotesi gaussiana è solo a scopo illustrativo. Non abbiamo bisogno di vincolarci a nessuna forma particolare della funzione di densità $p_{\theta}(\dots|\dots)$. Potremmo anche scegliere di modellare variabili che assumono valori in spazi continui o discreti. La scelta della funzione di densità può determinare se inquadrare il compito come un problema di classificazione o di regressione, ciascuno con le sue specifiche caratteristiche. Queste rappresentazioni avranno i loro vantaggi e svantaggi in termini di addestramento efficiente, rappresentazione del modello, espressività, generalizzazione, qualità, velocità di campionamento e tasso di compressione.

In questo capitolo, esploreremo varie classi di modelli comunemente usati per modellare probabilità condizionate, partendo da architetture tradizionali fino a quelle più all'avanguardia. Inizieremo con famiglie di modelli come la regressione logistica per gettare le basi e chiarire i concetti essenziali. Successivamente esploreremo le reti neurali e le reti neurali ricorrenti, che portano a famiglie più complesse. Poi avanzeremo verso innovazioni che servono come pietra angolare delle attuali architetture, includendo reti neurali con mascheramento causale, in particolare le WaveNet, e i transformer. La discussione sulle reti neurali con mascheramento causale si ispira alle intuizioni offerte dalle slide delle lezioni di Abbeel et al. [2020]).

Per illustrare i concetti principali e le tecniche per applicazioni nel campo delle immagini, useremo il dataset standard Binary MNIST (Larochelle e Murray, 2011), un esempio canonico della visione artificiale. Anche se usiamo dataset semplici, questi modelli hanno potenzialità per estendersi ben oltre queste semplici applicazioni.

Questi modelli possono essere usati per sviluppare strategie di trading utilizzando dati alternativi. Per esempio, l'analisi di immagini satellitari di campi coltivati può aiutare a prevedere i prezzi delle materie prime (Guida, 2019; Denev e Amen, 2020), mentre le osservazioni dei parcheggi dei negozi al dettaglio possono prevedere gli utili delle società di vendita al dettaglio (Guida, 2019), tra le tante altre applicazioni. Naturalmente, possiamo applicare questi modelli anche alle serie di prezzi, o più appropriatamente, di rendimenti.

Entro la fine del capitolo, forniremo esempi di come questi modelli possono essere applicati alla gestione patrimoniale e al trading, esplorando anche come i modelli transformer possono essere adattati per compiti di previsione delle serie temporali. Nel Capitolo 9 vedremo come costruire una semplice strategia di trading ad alta frequenza utilizzando modelli generativi autoregressivi per elaborare i segnali vocali dagli annunci della Federal Reserve per informare le decisioni di trading.

Le principali famiglie di modelli

Modello di regressione logistica

Il dataset binary MNIST (Larochelle e Murray, 2011) è una versione semplificata del dataset MNIST originale, che è stato ampiamente utilizzato come benchmark nella computer vision.

Questo dataset è costituito da immagini di 28×28 pixel, dove ogni pixel appartiene a $\{0, 1\}$, per rappresentare il colore nero o bianco in quella posizione. Il vincolo binario fa sì che ci siano $D = 2^{784}$ immagini possibili, evidenziando il fatto che qualsiasi set di addestramento pratico può coprire solo una piccolissima porzione di tutte le potenziali configurazioni di immagini. Infatti, il dataset contiene solo 60.000 immagini per l'addestramento e 10.000 immagini per il test.

In questo contesto, un'immagine $\mathbf{x}$ è un elemento dello spazio $\mathcal{X} = \{0, 1\}^D$.

Come esempi tipici di cifre nel dataset, vedi la Figura 5.2.



Figura 5.2 Campione dal dataset Binarized MNIST. Larochelle e Murray (2011).

Per una data immagine $\mathbf{x}$, possiamo scrivere la sua probabilità congiunta come segue:

$$\begin{aligned}
p_{\mathbf{X}}(\mathbf{x}) = p_{X_1, X_2, \dots, X_{768}}(x_1, \dots, x_{768}) &= p_{X_1}(x_1) \\
&\times p_{X_2|X_1}(x_2 | x_1) \\
&\times p_{X_3|X_1, X_2}(x_3 | x_1, x_2) \\
&\dots \\
&\times p_{X_{768}|X_1, \dots, X_{767}}(x_{768} | \mathbf{x}_{1:767})
\end{aligned}
\tag{5.4}$$

Questa fattorizzazione è presentata in modo prolisso per evidenziare due aspetti chiave di questo modello:

- la mancanza di condivisione dei parametri;
- nessuna ipotesi di indipendenza condizionata.

In questo modello, ogni probabilità condizionata è modellata usando un modello logistico definito come:

$$p(x_d | \mathbf{x}_{1:d-1}) = f(\mathbf{x}_{1:d-1}; \theta^{(d)}) = \text{sigmoid}(\mathbf{x}_{1:d-1}^T \mathbf{w}^{(d)} + b^{(d)})
\tag{5.5}$$

Dal Capitolo 3 potreste ricordare che il modello logistico è definito in termini della funzione sigmoide. Qui, a ogni condizionata è assegnato un insieme unico di parametri $\theta^d = \{\mathbf{w}^d, b^d\}$. Questo modello è noto come FVSBN (*Fully Visible Sigmoid Belief Network*); vedi Frey et al. (1995).

Finora, abbiamo esaminato diverse rappresentazioni di modelli e ne introdurremo alcune altre. Tenete presente, tuttavia, che l'addestramento del modello e la stima dei parametri saranno coperti in dettaglio nel prossimo paragrafo di questo capitolo.

Campionamento da FVSBN. Il processo di campionamento implica il generare un pixel alla volta, sequenzialmente. Per esempio, il primo pixel è campionato dalla sua distribuzione, $x_1 \sim p_{x_1}$. I pixel successivi vengono generati condizionati dai valori dei pixel realizzati

precedentemente, per esempio, $x_2 \sim p_{x_2|x_1}(\cdot, x_1)$, continuando in questo modo per il resto dei pixel. Questo metodo è noto come *Ancestral Sampling* (Bishop, 2006).

Invece di modellare le condizionate usando un modello logistico, potremmo esplorare l'uso di una funzione differente per migliorare l'espressività del modello, come una rete neurale. Qui,

$$p(x_d | \mathbf{x}_{1:d-1}) = f(\mathbf{x}_{1:d-1}; \theta^{(d)})$$

(5.6)

dove ora $f = \text{NeuralNetwork}(\mathbf{x}_{1:d-1}; \theta^{(d)})$. Naturalmente, modellare con una rete neurale aumenterà il numero di parametri da stimare.

MADE (Masked Autoencoder for Density Estimation)

Masked Autoencoder for Density Estimation, vedi Germain et al. (2015), rappresenta un progresso significativo nel campo dell'intelligenza artificiale generativa, migliorando le capacità degli autoencoder tradizionali, i quali comprimono i dati di input in una rappresentazione latente e poi tentano di ricostruirla, introducendo e rimuovendo rumore. Tuttavia, sono limitati nella loro capacità di generare nuovi punti di dati o di modellare la distribuzione di probabilità dei dati.

MADE affronta queste limitazioni introducendo una dipendenza sequenziale tra le variabili di input, seguendo la regola della catena per le probabilità. Questo metodo garantisce che la generazione di qualsiasi punto di dati nella sequenza non dipenda da punti di dati futuri, aderendo a un ordinamento predefinito. Questa innovazione permette a MADE di restituire una probabilità e di campionare nuovi punti di dati in un modo che rispetta la distribuzione dei dati sottostante, segnando un distacco dagli autoencoder tradizionali.

Le caratteristiche chiave di MADE includono la sua flessibilità nel gestire variabili di input di dimensionalità variabile e il suo uso di un output softmax per ogni variabile, per rappresentare una distribuzione di probabilità dalla quale possono essere campionati nuovi punti di dati. La struttura del modello, che incorpora connessioni mascherate tra i nodi, gli permette di mantenere la proprietà autoregressiva, essenziale per un'accurata stima della densità.

Illustrando l'applicazione di MADE, gli esperimenti su dataset come MNIST dimostrano la sua capacità di generare immagini di cifre che somigliano sempre a punti di dati autentici con un addestramento prolungato. La Figura 5.3 mostra un'immagine generata, a sinistra, e il suo vicino più prossimo, a destra, che è una buona pratica per verificare che il modello non stia solo imparando l'aspetto delle immagini del dataset, ma che stia effettivamente generandone di nuove.

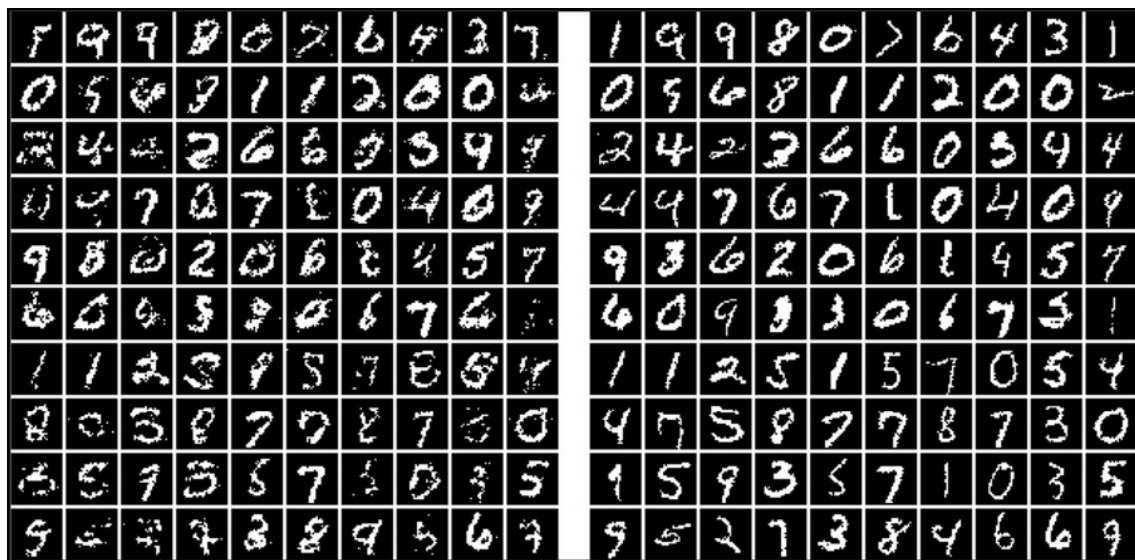


Figura 5.3 Generazione di MADE sul MNIST. A sinistra: campioni da un modello MADE.

A destra: il vicino più prossimo nel MNIST binarizzato. Fonte: Germain et al. (2015).

In conclusione, l'approccio innovativo di MADE nel combinare framework di autoencoder con tecniche di modellazione probabilistica lo ha fatto emergere come un modello fondante nell'intelligenza artificiale generativa. Il suo contributo risiede nella sua capacità di generare nuovi punti di dati e di modellare più accuratamente le distribuzioni di dati, spianando la strada a ulteriori progressi nel campo.

Modelli a reti neurali con mascheramento causale

Le reti neurali con mascheramento causale sono emerse come un approccio oggi leader nel campo dell'intelligenza artificiale generativa, sviluppando ulteriormente i concetti fondanti introdotti da modelli come MADE. Questo approccio ha fatto avanzare lo stato della modellazione generativa implementando un'efficiente condivisione dei parametri e introducendo la codifica delle coordinate per incorporare informazioni sulla posizione nelle sequenze di dati.

Al centro delle reti neurali con mascheramento causale c'è il principio di parametrizzare le condizionate con una rete neurale, simile a MADE, ma con un'innovazione cruciale: l'uso degli stessi parametri della rete neurale attraverso le diverse fasi del processo di generazione. Questo approccio è simile ad applicare una "finestra scorrevole" sui dati, che permette il riutilizzo ripetuto di un insieme di parametri, aumentando significativamente l'efficienza del modello e riducendo il numero totale di parametri necessari, pur modellando dati a elevata dimensionalità.

Per affrontare le problematiche associate alla condivisione dei parametri, in particolare la potenziale perdita di informazioni posizionali in dati come le immagini, queste reti incorporano la

codifica delle coordinate. Questa tecnica richiede di alimentare il modello con le coordinate di posizione dei punti di dati (per esempio, i pixel in un'immagine) insieme ai dati stessi, permettendo alla rete di mantenere la consapevolezza spaziale nonostante l'uso uniforme dei parametri in più posizioni. La codifica delle coordinate può essere implementata in varie forme, come codifica one-hot o codifica di posizione relativa, aggiungendo un livello di flessibilità al design del modello.

WaveNet

Fra le implementazioni di questo approccio, una delle prime e di maggior successo è stata WaveNet (vedi Oord et al., 2016a), un modello progettato principalmente per generare parlato. La sua influenza si estende oltre gli ambienti accademici, con varianti del modello implementate in sistemi del mondo reale (come gli assistenti personali sugli smartphone). WaveNet è riconosciuto per la sua superiore qualità di generazione dell'audio, rendendolo una pietra miliare nell'evoluzione delle tecnologie di sintesi audio.

WaveNet utilizza layer convoluzionali 1D e introduce tecniche aggiuntive come la dilatazione nelle convoluzioni per catturare informazioni da posizioni poste più indietro nella sequenza, vedi Figura 5.4. Ciò significa che WaveNet può catturare in modo più efficiente dipendenze a più lungo raggio all'interno dei dati audio, migliorando la qualità e la naturalezza dell'audio generato. Questo modello ha mostrato le potenzialità di questo approccio non solo per i dati audio ma anche per un'ampia gamma di tipi di dati sequenziali.

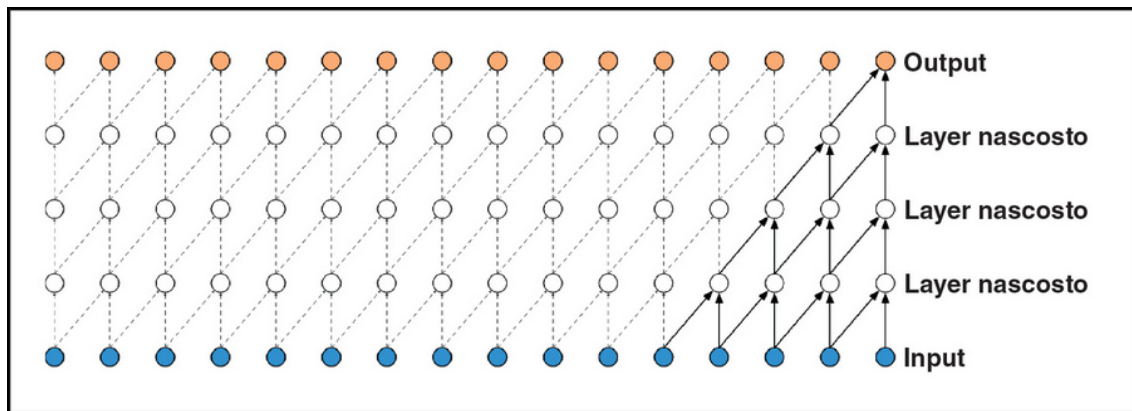


Figura 5.4 Rappresentazione grafica di una serie di layer convoluzionali causali. Figura 2 da Oord et al. (2016a).

In pratica, WaveNet utilizza una serie di layer convoluzionali causali dilatati per permettere alle reti di avere campi recettivi molto grandi, permettendo così alla rete di considerare input situati più lontano nella sequenza. Per esempio, la Figura 5.5 illustra delle convoluzioni causali dilatate con dilatazioni di 1, 2, 4 e 8, come mostrato nell'articolo di Oord et al. (2016a).

Come esempio di utilizzo di WaveNet, il notebook di accompagnamento [*WaveNet Notebook*] ne dimostra l'applicazione per la previsione di dati finanziari, fornendo un'opportunità per l'apprendimento pratico con questo tipo di modelli.

Nonostante l'efficienza e l'espressività dei modelli a reti neurali con mascheramento causale, permangono dei problemi, in particolare nel contesto del campionamento.

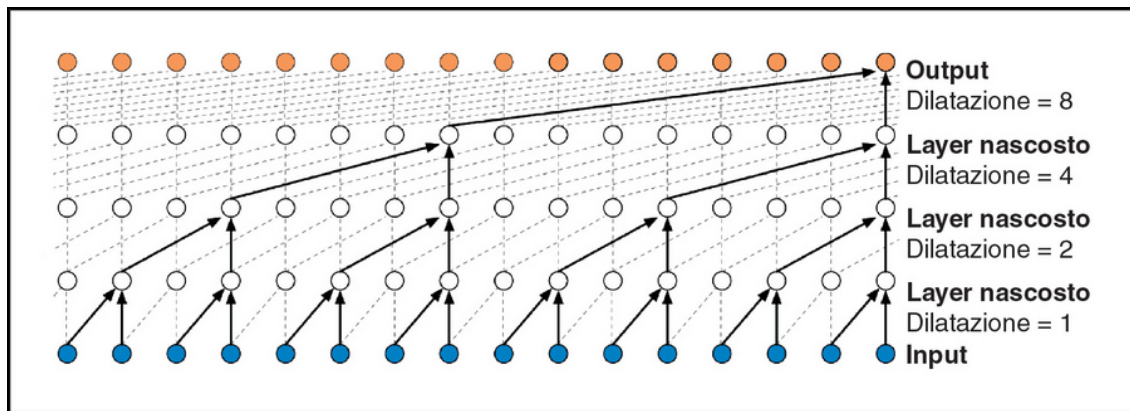


Figura 5.5 Rappresentazione grafica di una serie di layer convoluzionali causali dilatati. Figura 3 da Oord et al. (2016a).

La generazione di nuovi punti di dati richiede l’elaborazione sequenziale di ogni elemento, il che può essere lento. Tuttavia, il processo di addestramento beneficia del parallelismo, che accelera significativamente l’apprendimento del modello. Inoltre, le preoccupazioni riguardanti la finestra di contesto finita – ovvero quanto indietro il modello può guardare quando genera la parte successiva di una sequenza – sono state mitigate dai progressi nei meccanismi di attenzione.

Reti neurali ricorrenti

Come discusso nel Capitolo 3, le reti neurali ricorrenti (RNN, *Recursive Neural Network*) sono ben note per la loro capacità di elaborare dati sequenziali con una capacità di look-back effettivamente infinita. Questa caratteristica le distingue da altri modelli limitati a input di dimensione fissa, cosa che permette alle reti neurali ricorrenti di incorporare dinamicamente tutte le informazioni precedenti di una sequenza. Questa capacità le rende eccezionalmente adatte per compiti complessi come il

riconoscimento vocale, l'elaborazione del linguaggio naturale e altre applicazioni relative alle serie temporali.

Per ricapitolare, al centro di una rete neurale ricorrente c'è il concetto di uno stato nascosto della rete, spesso chiamato semplicemente “stato”. Questo stato agisce come memoria della rete, contenendo teoricamente tutte le informazioni passate necessarie per fare previsioni future. In una tipica RNN, lo stato nascosto a ogni indice della sequenza, t , denotato da h_t , viene aggiornato basandosi sullo stato precedente h_{t-1} e sull'input corrente x_t , secondo una funzione g . L'output y_t a ogni timestep dipende dallo stato corrente h_t usando una tipica funzione non lineare f :

$$\begin{aligned}h_t &= g(h_{t-1}, x_t) \\ y_t &= f(h_t)\end{aligned}\tag{5.7}$$

Qui, f e g sono tipici layer di reti neurali non lineari che sono indipendenti dalla specifica posizione dell'indice t . Di conseguenza, le reti neurali ricorrenti impiegano efficacemente la condivisione dei parametri attraverso la sequenza.

Questa loro natura permette alle reti neurali ricorrenti di gestire lunghezze di input variabili e fornisce loro la capacità teorica di gestire una finestra di contesto infinita, che è ideale per generare sequenze coerenti e contestualmente ricche.

Tuttavia, l'applicazione pratica delle reti neurali ricorrenti è spesso afflitta da problemi relativi alla loro struttura profonda e ricorsiva. In particolare, l'addestramento delle reti neurali ricorrenti può essere soggetto ai fenomeni dei gradienti, che possono svanire ed esplodere. Questi si verificano quando i segnali del gradiente, essenziali per l'apprendimento attraverso la retropropagazione, si riducono a livelli insignificanti (e svaniscono) o crescono troppo (ed esplodono) man mano che si propagano all'indietro attraverso ogni timestep nella

sequenza. Tali problemi possono limitare la capacità della rete neurale ricorrente di apprendere dipendenze a lungo termine all'interno dei dati, influenzando l'efficacia complessiva del modello nelle applicazioni del mondo reale.

In compiti generativi, le reti neurali ricorrenti operano prevedendo l'elemento successivo in una sequenza basandosi sulla storia che ha elaborato, dove ogni output dipende dallo stato computazionale precedente. Per esempio, se addestrate su dataset specifici come le opere di Shakespeare o testi codificati di Wikipedia, le reti neurali ricorrenti possono generare un testo che imita lo stile dei dati di addestramento, catturando sfumature come la punteggiatura, la grammatica e perfino la formattazione specifica del testo.

Nel notebook di accompagnamento [*RNN Notebook*], dimostriamo un esperimento che usa reti neurali ricorrenti per prevedere i movimenti di prezzo e i rendimenti dell'S&P 500.

Considerazioni pratiche e limiti

Sebbene teoricamente capaci di gestire sequenze di lunghezza e complessità arbitraria, le reti neurali ricorrenti, nella pratica, soffrono di sfide operative significative.

- *Velocità di addestramento*: l'addestramento delle reti neurali ricorrenti è lento a causa della natura sequenziale dei loro calcoli, che impedisce l'elaborazione parallela. Ogni passo dipende dal completamento del passo precedente, portando a tempi di addestramento più lunghi.
- *Problemi del gradiente*: l'addestramento delle reti neurali ricorrenti è notoriamente difficile a causa dei problemi del gradiente, che può svanire ed esplodere, i quali possono compromettere l'efficacia dell'apprendimento. Tecniche come le

unità LSTM (*Long Short-Term Memory*) permettono di mitigare questi problemi, ma non li eliminano del tutto.

- *Requisiti di memoria:* a causa della loro natura ricorsiva, le reti neurali ricorrenti spesso richiedono una notevole quantità di memoria per l'addestramento, in quanto necessitano di mantenere informazioni di molti passi temporali.

I transformer

Negli ultimi anni, i modelli transformer hanno rivoluzionato sia l'industria sia il mondo accademico. Da quando sono stati introdotti nell'influentissimo articolo *Attention Is All You Need* di Vaswani et al. (2023), sono stati applicati con successo a un'ampia varietà di tipi di dati, inclusi testi, audio, serie temporali, immagini e video. In quanto modelli autoregressivi, i transformer offrono diversi vantaggi rispetto alle tradizionali reti neurali ricorrenti. Uno dei più significativi è la loro facilità di addestramento e la capacità di sfruttare la parallelizzazione, che accelera sia l'addestramento sia l'inferenza. Inoltre, i transformer mitigano problemi come la sparizione e l'esplosione dei gradienti, prevalenti nei modelli precedenti, come le reti RNN e LSTM.

In questo paragrafo, esploreremo in dettaglio l'architettura transformer, scomponendo i suoi componenti fondamentali. Alla fine di questo paragrafo, avremo coperto tutti i blocchi costitutivi essenziali necessari per implementare due dei modelli transformer più influenti: BERT, introdotto da (Devlin et al. 2019) e poi sviluppato da Google, e GPT, introdotto da Radford et al. (2018) e sviluppato da OpenAI. BERT (*Bidirectional Encoder Representations from Transformers*) ha rivoluzionato l'elaborazione del linguaggio naturale, ridefinendo il modo in cui sia l'industria sia il mondo accademico affrontano un'ampia gamma di compiti di elaborazione

del linguaggio naturale. Allo stesso modo, la famiglia di modelli GPT (*Generative Pre-trained Transformers*) di OpenAI è stata pionieristica, spingendo in avanti i confini di ciò che è possibile fare in termini di generazione e comprensione del linguaggio naturale.

Per chiarezza espositiva, concentriamoci sui dati testuali, poiché aiutano la comprensione di questi modelli. Tuttavia, tenete presente che lo stesso framework potrebbe essere applicabile alle serie temporali numeriche, in quanto sia il testo sia le serie temporali sono “sequenze”. Approfondiremo ulteriormente questo punto nel paragrafo successivo, che si occupa del passaggio dei transformer dalla NLP alle serie temporali. Tratteremo brevemente i passi essenziali coinvolti in una tipica pipeline di pre-elaborazione per le applicazioni di elaborazione del linguaggio naturale (NLP, *Natural Language Processing*), fornendo giusto le informazioni sufficienti per comprendere i transformer. Per fare ciò, utilizzeremo una combinazione di frammenti di codice e notazione matematica, così da illustrare come funzionano questi modelli.

Il meccanismo di attenzione

Per comprendere il concetto di attenzione, consideriamo le seguenti due frasi.

- *The startup achieved a new **round** of funding to expand operations.*
- *He invited his friends for another **round** of drinks at the bar.*

Nella prima frase, la parola “round” si riferisce a una fase di finanziamento per un’azienda, mentre nella seconda frase si riferisce a una sequenza di drink ordinati. Intuitivamente, questi diversi significati suggeriscono che la parola “round” debba essere rappresentata da vettori differenti, anche noti in letteratura come

embedding vettoriali, *vector embedding* o, semplicemente, *embedding*, a seconda del contesto.

Tuttavia, nei passi iniziali della pipeline NLP – quando il testo grezzo viene trasformato in embedding dei token – il modello assegna la stessa rappresentazione vettoriale alle parole “round” di entrambe le frasi, indipendentemente dal contesto. Questo perché gli embedding dei token, a questo punto, sono ancora *indipendenti dal contesto* e si basano esclusivamente sull’ID del token assegnato alla parola.

La pipeline NLP: dai dati grezzi agli embedding dei token.

1. *Tokenizzazione*: il primo passo nelle attuali pipeline NLP prevede la *conversione del testo grezzo in unità più piccole, chiamate token*. I token possono essere parole intere, parti di parole o perfino caratteri, a seconda del tokenizzatore. Ogni token viene mappato su un ID univoco.
2. *Embedding dei token*: il modello *assegna un embedding iniziale a ogni token* in base all’ID del token stesso. In questa fase, gli embedding sono ancora *indipendenti dal contesto*: la stessa parola avrà sempre lo stesso embedding, indipendentemente dal testo circostante.

Per illustrare ciò, useremo la libreria Python `transformers` di Hugging Face (<https://huggingface.co>), che semplifica il lavoro con i modelli transformer, come BERT. Nel codice seguente, esaminiamo il processo di tokenizzazione:

```
from transformers import BertTokenizer, BertModel
import torch
# Inizializza il tokenizer e il modello BERT
model_ckpt = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(model_ckpt)
model = BertModel.from_pretrained(model_ckpt)
# Due frasi con significati diversi per "round"
sentence_1 = "The startup achieved a new round of funding to expand operations."
```

```

sentence_2 = "He invited his friends for another round of drinks at the
bar."
# Tokenizza le frasi
inputs_1 = tokenizer(sentence_1, return_tensors='pt')
inputs_2 = tokenizer(sentence_2, return_tensors='pt')
# Ottieni l'ID del token per "round"
token_id = tokenizer.convert_tokens_to_ids('round')
print(f"Token ID of word round: {token_id}")
# Mostra gli ID dei token per mostrare il processo di tokenizzazione
print("Token ids for sentence 1:", inputs_1.input_ids)
print("Token ids for sentence 2:", inputs_2.input_ids)
# Accedi direttamente agli embedding dei token dalla matrice di
embedding di BERT
# La matrice di embedding è model.embeddings.word_embeddings
token_embedding_matrix = model.embeddings.word_embeddings.weight
# Estrai l'embedding del token per "round"
token_embedding_matrix[token_id]
# Mostra i risultati per verificare che gli embedding dei token siano
identici
print(f"Token embedding for 'round':
{token_embedding_for_round_1[:5]}...") # Primi 5 valori
>>>
Token ID of word round: 2461
Token ids for sentence 1: tensor([[
  101,  1996, 22752,  4719,
  1037,  2047,  2461,  1997,
  4804,  2000,  7818,  3136,
  1012,   102]])
Token ids for sentence 2: tensor([[
  101,  2002,  4778,  2010,
 2814,  2005,  2178,  2461,
 1997,  8974,  2012,  1996,
 3347,  1012,  102]])

```

Qui, l'ID del token per la parola “round” è 2461, che si trova in posizione 6 e posizione 7 (considerate l'indicizzazione a base 0) rispettivamente in `inputs_1.input_ids` per la frase `sentence_1` e in `inputs_2.input_ids` per la frase `sentence_2`.

Per determinare gli embedding dei token assegnati alla parola “round” accediamo alla matrice di embedding dei token di BERT:

```

# Accedi direttamente agli embedding dei token dalla matrice di
embedding di BERT
# La matrice di embedding è model.embeddings.word_embeddings
token_embedding_matrix = model.embeddings.word_embeddings.weight
# Estrai l'embedding del token per "round"
token_embedding_for_round = token_embedding_matrix[token_id]
print(f"Embedding matrix size: {token_embedding_matrix.size()}")
>>>

```

```

Embedding matrix size: torch.Size([30522, 768])
print(f"Token embedding for 'round': {token_embedding_for_round[:5]}...")
>>
Token embedding for 'round': tensor([0.0109, 0.0577, 0.0310, -0.0615,
0.0571],
                                grad_fn=<SliceBackward0>)
...

```

Le dimensioni della matrice di embedding per il modello BERT sono `torch.Size([30522, 768])`, il che significa che ogni ID di token viene mappato su una rappresentazione vettoriale 768-dimensionale, e il vocabolario – il numero di token unici per questo specifico modello – è costituito da 30.522 token. Maggiori dettagli su come vengono create e apprese queste rappresentazioni di embedding saranno forniti più avanti. Ulteriori informazioni su questo processo e sulla tipica pipeline NLP possono essere trovate nel tutorial di Hugging Face, situato nella cartella `chapter10` del repository del libro.

Lo scopo del meccanismo di attenzione è quello di raffinare la rappresentazione di ogni parola in una frase, rendendola più dipendente dal contesto. Dopo aver applicato l'attenzione, otteniamo un embedding più raffinato, che cattura il contesto in cui appare la parola. L'importanza del meccanismo di attenzione è così profonda che gli autori dell'influente articolo *Attention Is All You Need* (Vaswani et al., 2023) l'hanno enfatizzata fin dal titolo. Ma come funziona, esattamente, l'attenzione? Come fa a raffinare questi embedding?

Prima di immergerci nei dettagli matematici, esploriamo alcune rappresentazioni illuminanti usando BertViz, uno strumento introdotto da Vig (2019), che ci aiuta a capire come modelli come BERT, GPT2 o T5 aggiornano la rappresentazione delle parole attraverso l'attenzione. Questa visualizzazione ci aiuterà a cogliere il concetto prima di passare alle equazioni.

La maggior parte dei meccanismi di attenzione raffina gli embedding calcolando una combinazione lineare di tutti gli

embedding di input usando specifici “pesi di attenzione”. Il meccanismo di attenzione rende questi “pesi di attenzione” o “punteggi di attenzione” dipendenti dall’input, permettendo al modello di creare rappresentazioni dipendenti dal contesto. Per chiarire questo concetto, useremo BertViz per vedere come BERT assegna i pesi di attenzione tra le diverse parole in una frase.

Nelle rappresentazioni generate dal codice seguente, vedi Figura 5.6, a e b, vengono mostrati i punteggi di attenzione per due frasi di input:

```
from transformers import AutoTokenizer
from transformers import AutoModel
from bertviz import head_view
model_ckpt = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = AutoModel.from_pretrained(model_ckpt, output_attentions=True)
sentence_a = 'The startup achieved a new round of funding to expand
operations'

sentence_b = 'He invited his friends for another round of drinks at the
bar'
viz_inputs = tokenizer(sentence_a, sentence_b, return_tensors='pt')
attention = model(**viz_inputs).attentions
sentence_b_start = (viz_inputs.token_type_ids == 0).sum(dim=1)
tokens = tokenizer.convert_ids_to_tokens(viz_inputs.input_ids[0])
head_view(attention, tokens, sentence_b_start, heads=[8])
```

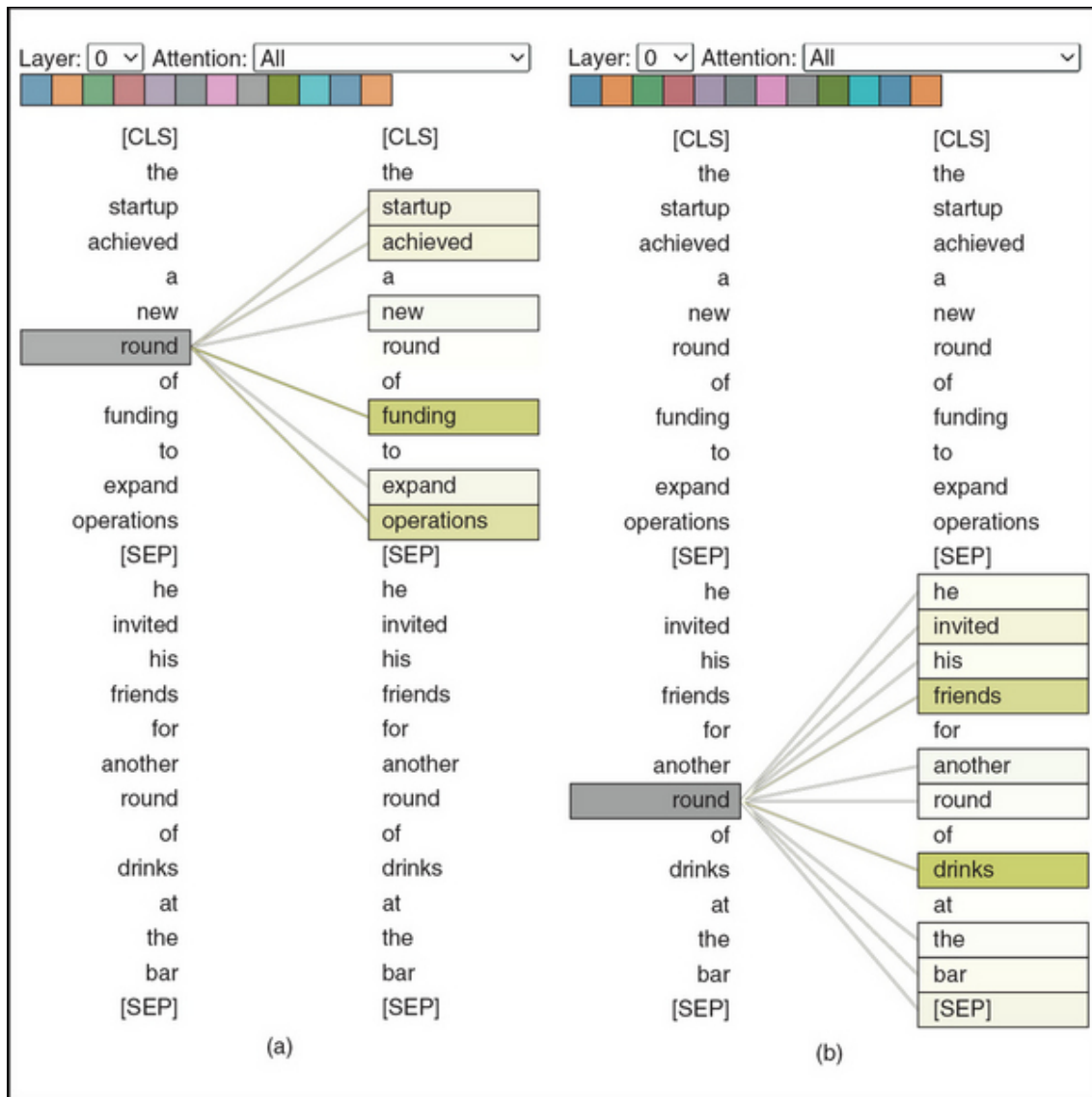


Figura 5.6 Rappresentazione grafica del meccanismo di attenzione.

Qui, abbiamo chiesto a un BERT di elaborare simultaneamente le due sequenze di input. Le colonne a sinistra e a destra rappresentano entrambe le frasi, separate dal token speciale [SEP].

In queste figure, la colonna a destra rappresenta gli embedding di input iniziali, mentre la colonna a sinistra mostra gli embedding di output dopo aver applicato il meccanismo di attenzione. Sebbene le rappresentazioni non mostrino gli embedding finali delle parole,

mostrano i punteggi di attenzione tra le parole. Selezionando qualsiasi parola nella colonna a sinistra, evidenziata con uno sfondo grigio, possiamo vedere le linee che collegano questa parola a tutte le parole nella colonna a destra. L'intensità di colore della colonna a destra rappresenta l'entità del punteggio di attenzione, che varia da 0 a 1, indicando quanto ogni parola nella colonna a destra influenza la parola selezionata nella colonna a sinistra.

Esaminiamo bene la Figura 5.6a, dove ci concentriamo sulla parola “round” della prima frase, “The startup achieved a new *round* of funding to expand operations”. La rappresentazione mostra che le parole “funding”, “operations”, “startup” e “achieved” hanno i punteggi di attenzione più alti, cosa che contribuisce maggiormente a raffinare la rappresentazione di “round”. Significativamente, nessuna delle parole della seconda frase è considerata importante nell'aggiornare questa specifica rappresentazione.

Al contrario, nella Figura 5.6b, selezioniamo la parola “round” della seconda frase, “He invited his friends for another *round* of drinks at the bar”. In questo contesto, le parole “drinks”, “friends” e “invited” sono le più significative nell'aggiornare la rappresentazione di “round”. Allo stesso modo, nessuna significativa attenzione è assegnata a parole della prima frase.

Queste rappresentazioni illustrano come l'attenzione aiuti il modello a concentrarsi sulle parti rilevanti del contesto quando raffina le rappresentazioni delle parole. Ora vediamo come viene calcolato questo aggiornamento dell'attenzione.

Se avessimo a che fare con una serie di rendimenti azionari, potremmo immaginare che un rendimento al tempo t potrebbe dipendere significativamente da qualche rendimento passato $t' = t - \Delta$ che era particolarmente significativo, magari perché al tempo t' c'è

stato un annuncio di utili o determinate notizie economiche (Guijarro-Ordóñez et al., 2021).

Ora, vediamo come viene calcolato questo aggiornamento dell'attenzione. Ipotizziamo di avere una rappresentazione di embedding per ogni token nella sequenza di input, denotata come $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N$, ciascuna di dimensione $\mathbb{R}^{d_v}$. Nel caso di una serie di rendimenti, ogni $\mathbf{v}_t$ potrebbe essere una rappresentazione di embedding dei rendimenti azionari al tempo t in un portafoglio. Come accennato, la maggior parte dei meccanismi di attenzione crea embedding raffinati calcolando una combinazione lineare di tutti gli embedding di input, come gli $\mathbf{v}_s$ del nostro esempio, che, come vedremo dopo, sono combinazioni lineari degli embedding di input $\mathbf{X}$. Per trovare una rappresentazione raffinata per ogni vettore di input, calcoliamo una somma pesata. Per esempio, la rappresentazione raffinata, o rappresentazione nascosta, $\mathbf{h}_1$ per $\mathbf{v}_1$ viene calcolata come:

$$\begin{aligned}\mathbf{h}_1 &= a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + \dots + a_N \mathbf{v}_N \\ &= \sum_n a_n \mathbf{v}_n \\ &= \begin{bmatrix} | & \dots & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_N \\ | & \dots & | \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix} \\ &= \begin{bmatrix} | & \dots & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_N \\ | & \dots & | \end{bmatrix} \begin{bmatrix} | \\ \mathbf{a}_1 \\ | \end{bmatrix}\end{aligned}$$

(5.8)

dove $\mathbf{a} = [a_1, a_2, \dots, a_N]^T$ è il vettore di attenzione, noto anche come punteggi di attenzione o pesi di attenzione, usato per calcolare $\mathbf{h}_1$. Allo stesso modo, per la rappresentazione raffinata o rappresentazione nascosta del terzo vettore, abbiamo:

$$\mathbf{h}_3 = \begin{bmatrix} | & \cdots & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_N \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} | \\ \mathbf{a}_3 \\ | \end{bmatrix}$$

(5.9)

In generale, la rappresentazione raffinata per l'input i -esimo è:

$$\mathbf{h}_i = \begin{bmatrix} | & \cdots & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_N \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} | \\ \mathbf{a}_i \\ | \end{bmatrix}$$

(5.10)

Questo può essere rappresentato in forma matriciale come:

$$\begin{bmatrix} \text{--}\mathbf{h}_1^T\text{--} \\ \text{--}\mathbf{h}_2^T\text{--} \\ \vdots \\ \text{--}\mathbf{h}_N^T\text{--} \end{bmatrix} = \begin{bmatrix} \text{--}\mathbf{a}_1^T\text{--} \\ \text{--}\mathbf{a}_2^T\text{--} \\ \vdots \\ \text{--}\mathbf{a}_N^T\text{--} \end{bmatrix} \begin{bmatrix} \text{--}\mathbf{v}_1^T\text{--} \\ \text{--}\mathbf{v}_2^T\text{--} \\ \vdots \\ \text{--}\mathbf{v}_N^T\text{--} \end{bmatrix}$$

$$\mathbf{H} = \mathbf{A}\mathbf{V}$$

(5.11)

dove $\mathbf{A}$ è la matrice dei pesi di attenzione. In breve, aggiorniamo ogni token attraverso una combinazione lineare di tutti gli altri token. In un'applicazione finanziaria di serie temporali per la previsione dei rendimenti, per evitare bias di anticipazione, dovremmo includere solo i rendimenti passati quando calcoliamo il rendimento corrente. Questo approccio è noto nella terminologia NLP come attenzione causale mascherata (*causal masked attention*). La domanda ora è: come determiniamo questa matrice $\mathbf{A}$? Bene, questi pesi vengono appresi dai dati durante l'addestramento.

Prima di spiegare come viene calcolata la matrice di attenzione, chiariamo la differenza tra attenzione *hard* e *soft*.

- *Attenzione hard*: ogni vettore di attenzione $\mathbf{a}_N$ soddisfa la proprietà $\|\mathbf{a}_N\|_0 = 1$, il che significa che è un vettore one-hot dove solo una componente è 1 e le altre sono 0. Ciò significa che la nuova rappresentazione sarà esattamente uno dei vettori di input.
- *Attenzione soft*: ogni vettore di attenzione $\mathbf{a}_N$ soddisfa $\|\mathbf{a}_N\|_1 = 1$, il che significa che la nuova rappresentazione è una combinazione pesata dei vettori di input.

Sebbene ci siano diversi metodi per calcolare la matrice di attenzione e la matrice dei vettori di embedding $\mathbf{V}$, in questo libro ci concentreremo sull'approccio più noto, *Scaled Dot-Product Attention*, come introdotto nell'influente articolo di Vaswani et al. (2023).

Scaled Dot-Product Attention

I punteggi di attenzione $\mathbf{A}$ vengono calcolati come:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)$$

(5.12)

e quindi, la rappresentazione nascosta viene calcolata come:

$$\begin{aligned}\mathbf{H} &= \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \\ \mathbf{H} &= \mathbf{A}\mathbf{V}\end{aligned}$$

(5.13)

Le operazioni della Scaled Dot-Product Attention sono illustrate nella Figura 5.7, che serve come utile riferimento per comprendere il

processo.

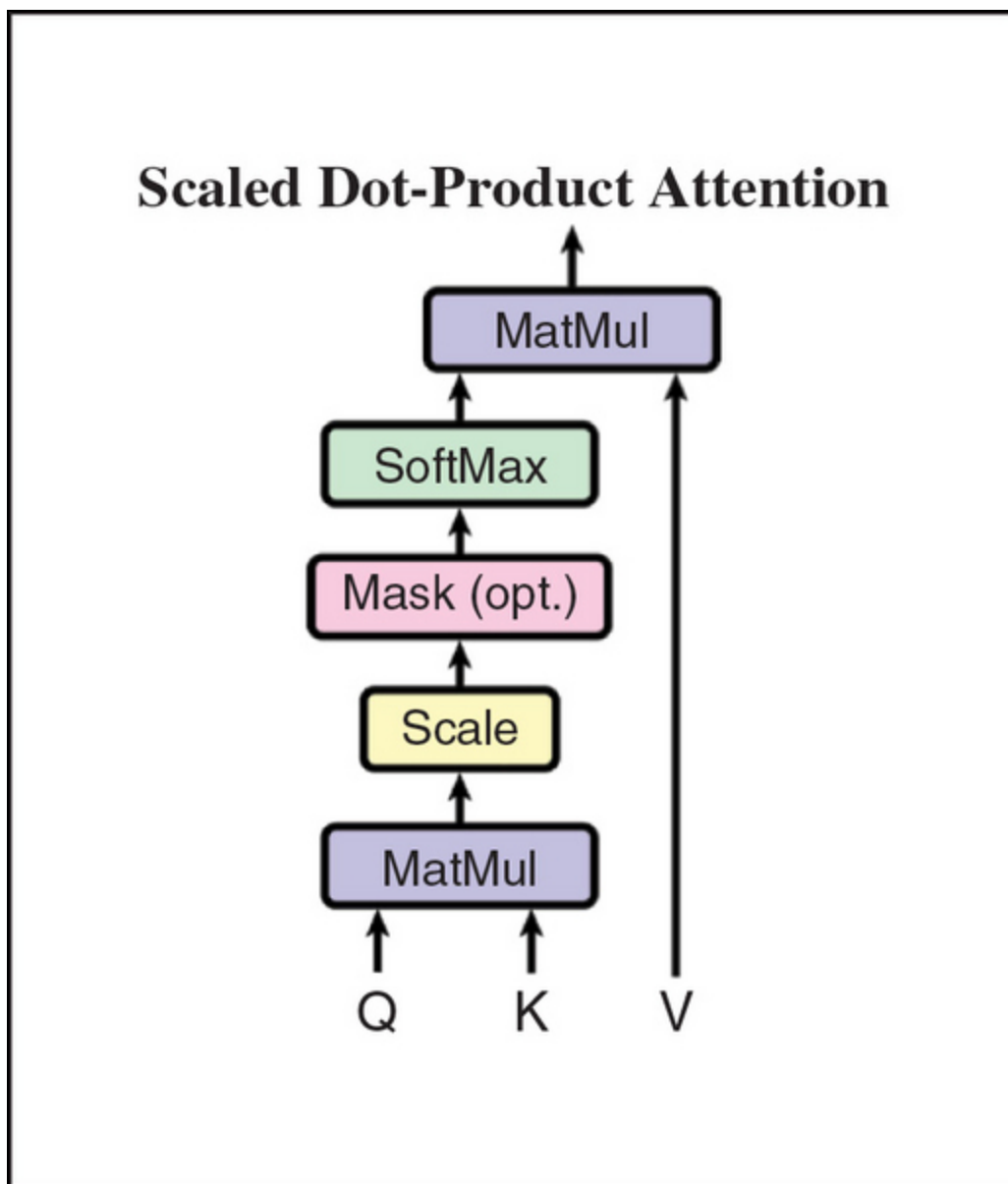


Figura 5.7 Rappresentazione grafica della Scaled Dot-Product Attention.

Figura 2 (a sinistra) da Vaswani et al. (2023).

Analizziamo ora l'equazione per la Scaled Dot-Product Attention.

- **Q (Query Matrix):** questa matrice Query viene calcolata dagli embedding di input $\mathbf{X}$ come $\mathbf{Q} = \mathbf{X} \mathbf{W}_q$. Per ora, considerate gli embedding di input $\mathbf{X}$ come quelli generati dalla pipeline NLP di

base spiegata in precedenza, dove vengono assegnati in base all'ID del token. Più avanti vedremo come negli embedding dei token vengono incorporate le informazioni aggiuntive sulla sequenza, per creare completamente $\mathbf{X}$. La matrice $\mathbf{Q}$ contiene N vettori riga, ognuno dei quali rappresenta un vettore della query $\mathbf{q}_N^T$ di dimensione d_q . La matrice $\mathbf{W}_q$ è un parametro apprendibile di dimensioni $d \times d_q$.

- **K (Key Matrix)**: allo stesso modo, la matrice Key viene calcolata come $\mathbf{K} = \mathbf{X} \mathbf{W}_k$. Ogni riga di $\mathbf{K}$ corrisponde a un vettore di chiavi $\mathbf{k}_N^T$ di dimensione d_k . La matrice $\mathbf{W}_k$ è anch'essa un parametro apprendibile di dimensioni $d \times d_k$.
- **V (Value Matrix)**: la matrice Value viene calcolata come $\mathbf{V} = \mathbf{X} \mathbf{W}_v$. Contiene N vettori riga, ognuno dei quali rappresenta un vettore dei valori $\mathbf{v}_N^T$ di dimensioni d_v . La matrice $\mathbf{W}_v$ è un parametro apprendibile di dimensioni $d \times d_v$.

Affinché la Scaled Dot-Product Attention funzioni, le dimensioni d_q e d_k devono coincidere, e una scelta comune è $d_q = d_k = d$.

Il meccanismo di attenzione utilizzato qui è di attenzione soft, imposto dall'applicazione della funzione softmax per garantire che le righe di $\mathbf{A}$ abbiano somma 1. Il fattore di scala $1/\sqrt{d_k}$ è stato introdotto dagli autori dell'articolo *Attention Is All You Need* per prevenire che l'argomento della softmax diventi troppo grande, evitando così che sorgano regioni della funzione con gradienti molto piccoli (Vaswani et al., 2023).

Per comprendere meglio la Scaled Dot-Product Attention, concentriamoci sul calcolo dei punteggi di attenzione non normalizzati:

$$\tilde{\mathbf{A}} = \mathbf{Q}\mathbf{K}^T$$

(5.14)

Possiamo chiamare questa matrice $\tilde{\mathbf{A}}$ i punteggi di attenzione non normalizzati. Per ogni vettore di input nella sequenza $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, calcoliamo la sua rappresentazione query $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_N$, la sua rappresentazione key $\mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_N$ e la sua rappresentazione value $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N$ e, usando queste informazioni, calcoliamo i pesi di attenzione.

Per esempio, per calcolare i pesi di attenzione $\tilde{\mathbf{a}}_1$ necessari per la rappresentazione nascosta $\mathbf{h}_1$ di $\mathbf{X}_1$, prendiamo il prodotto scalare del suo vettore query $\mathbf{q}_1$ con i vettori key nella sequenza:

$$[-\tilde{\mathbf{a}}_1^T -] = [-\mathbf{q}_1^T -] \begin{bmatrix} | & | & | & | \\ \mathbf{k}_1 & \mathbf{k}_2 & \dots & \mathbf{k}_N \\ | & | & | & | \end{bmatrix}$$

(5.15)

La terminologia e i concetti utilizzati nell'articolo *Attention Is All You Need* di Vaswani et al. (2023) traggono spunto dall'*information retrieval*, dove abbiamo un database o un archivio chiave-valore indicizzato da chiavi con i corrispondenti valori. Per recuperare un valore, calcoliamo la similarità tra una query e tutte le chiavi (*key*) del database. Il valore (*value*) recuperato può essere quello con il punteggio massimo o con una somma pesata di valori basata su questi punteggi, che è l'essenza della self-attention.

Dopo aver calcolato i punteggi di attenzione non normalizzati $\tilde{\mathbf{a}}_1$, li normalizziamo usando la funzione softmax:

$$\mathbf{a}_1 = \text{softmax}\left(\frac{\widetilde{\mathbf{a}}_1}{\sqrt{d_k}}\right)$$

(5.16)

Ora, possiamo calcolare la rappresentazione nascosta $\mathbf{h}_1$ prendendo una combinazione lineare di tutti i valori nell'archivio chiavi-valori, seguendo la stessa analogia dell'archivio chiavi-valori:

$$\begin{bmatrix} - & - & \mathbf{h}_1^T & - & - \end{bmatrix} = \begin{bmatrix} - & - & \mathbf{a}_1^T & - & - \end{bmatrix} \begin{bmatrix} - & - & \mathbf{v}_1^T & - & - \\ - & - & \mathbf{v}_2^T & - & - \\ & & \vdots & & \\ - & - & \mathbf{v}_N^T & - & - \end{bmatrix}$$

(5.17)

Questo produce una rappresentazione raffinata $\mathbf{h}_i$ per ogni input $\mathbf{x}_i$, che incorpora informazioni sull'intera sequenza.

La combinazione della Scaled Dot-Product Attention con le tre trasformazioni lineari per i vettori *query*, *key* e *value* è ciò che è chiamato *Attention Head* (“testa di attenzione”) nei modelli transformer (e possono essere utilizzate anche più “teste” per catturare le diverse relazioni tra le parole, come vedremo più avanti). Implementiamo questo concetto in PyTorch per comprenderlo meglio. La seguente implementazione è adattata dal libro *Natural Language Processing with Transformers*, di Tunstall et al. (2022).

Per implementare la Scaled Dot-Product Attention, dobbiamo passare le matrici *query*, *key* e *value*. Ecco una semplice implementazione della funzione in PyTorch:

```
import torch
from math import sqrt
import torch.nn.functional as F
def scaled_dot_product_attention(query, key, value):
    dim_k = query.size(-1)
    unnormalised_scores = torch.bmm(query, key.transpose(1, 2)) /
```

```

sqrt(dim_k)
scores = F.softmax(unnormalised_scores, dim=-1)
return torch.bmm(scores, value)

```

In questa funzione:

- usiamo `torch.bmm` per la moltiplicazione di matrici in batch, che gestisce in modo efficiente le dimensioni del batch durante il calcolo;
- i punteggi vengono scalati dividendoli per la radice quadrata delle dimensioni dei vettori *key* (`sqrt(dim_k)`) per stabilizzare i prodotti scalari e prevenirne una crescita eccessiva;
- applichiamo ai punteggi la funzione `softmax` per ottenere i pesi di attenzione e poi usiamo questi pesi per calcolare una somma pesata dei vettori *value*.

Successivamente, implementeremo la testa di attenzione completa. Ciò richiede la creazione delle matrici $\mathbf{W}_k$, $\mathbf{W}_q$ e $\mathbf{W}_v$, rispettivamente per le trasformazioni *key*, *query* e *value*. Ognuna di queste matrici trasforma gli embedding di input nelle dimensioni richieste per gli spazi dimensionali *query*, *key* e *value*.

Definiamo queste matrici nella seguente classe `AttentionHead`. Per semplicità, ipotizziamo che $d_k = d_q = d_v =$ dimensioni della testa, cui ci riferiamo nel codice come `head_dim`:

```

from torch import nn
class AttentionHead(nn.Module):
    def __init__(self, embed_dim, head_dim):
        super().__init__()
        self.q = nn.Linear(embed_dim, head_dim)
        self.k = nn.Linear(embed_dim, head_dim)
        self.v = nn.Linear(embed_dim, head_dim)
    def forward(self, hidden_state):
        attn_outputs = scaled_dot_product_attention(
            self.q(hidden_state),
            self.k(hidden_state),
            self.v(hidden_state)
        )
        return attn_outputs

```

Descrizione di questa implementazione.

- Il metodo `__init__` inizializza tre trasformazioni lineari per le matrici *query*, *key* e *value*. Queste trasformazioni proiettano gli embedding di input nello spazio dimensionale `head_dim`.
- Nel metodo `forward`, gli stati nascosti vengono passati attraverso le trasformazioni lineari per ottenere le matrici *query*, *key* e *value*.
- Viene richiamata la funzione `scaled_dot_product_attention` con queste matrici per calcolare gli `attn_outputs`.

Dalla self-attention ai transformer

In questo paragrafo, spiegheremo come si passa dal meccanismo di self-attention discusso in precedenza all'architettura transformer introdotta nell'influente articolo di Vaswani et al. (2023).

L'architettura del modello transformer è rappresentata nella Figura 5.8.

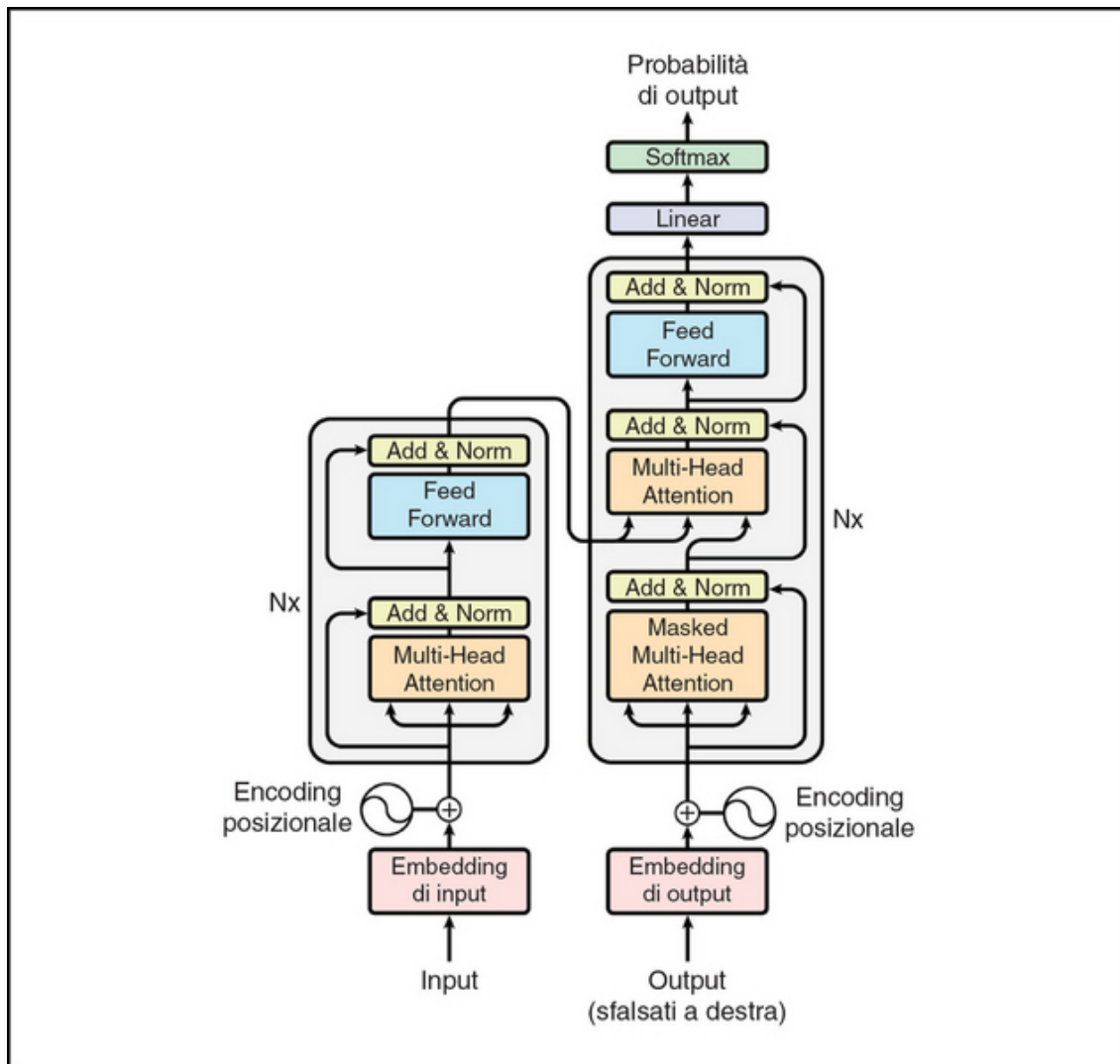


Figura 5.8 Rappresentazione grafica dell'architettura encoder-decoder del transformer, sviluppata per la traduzione automatica, elabora tramite l'encoder una sequenza di input – una frase nella lingua sorgente – in rappresentazioni continue. Il decoder genera poi la sequenza di output tradotta, producendo un token alla volta. Gli embedding di input corrispondono alla rappresentazione della lingua sorgente, mentre gli embedding di output corrispondono alla rappresentazione della lingua target. Gli embedding di output sono sfalsati di una posizione, per garantire che la previsione per la posizione i dipenda solo

dagli output noti alle posizioni precedenti a i . Figura 1 tratta da Vaswani et al. (2023).

Il modello transformer è stato originariamente sviluppato per compiti di traduzione automatica, che tradizionalmente seguono una struttura encoder-decoder. Il ruolo dell'encoder è quello di mappare la sequenza di input – la frase nella lingua sorgente – in una sequenza di rappresentazioni nascoste. Queste rappresentazioni nascoste vengono poi utilizzate dal decoder per generare la sequenza di output tradotta, un token alla volta. Questo processo è autoregressivo, ovvero il decoder genera ogni token considerando come input aggiuntivi i token generati precedentemente. L'encoder è raffigurato sul lato sinistro della Figura 5.8, mentre il decoder è mostrato a destra.

Come si evince dalla figura, sono necessari diversi componenti per costruire il modello transformer completo, inclusi gli embedding di input, la *Multi-Head Attention* e i blocchi *Feed-Forward*. Questi componenti sono fondamentali per le sezioni sia encoder sia decoder. Per comprendere appieno il modello transformer, esploreremo ciascuno di questi componenti in dettaglio, fornendo un'implementazione in codice della sezione encoder.

Inoltre, è importante notare che la self-attention consente al modello di prestare attenzione a tutti i token simultaneamente, il che non è ideale per compiti come la generazione di sequenze, dove il modello deve distinguere tra token passati e futuri, come nelle serie temporali finanziarie. Per affrontare questo problema, il meccanismo di self-attention può essere modificato per creare quello che è noto come *causal masked self-attention* (auto-attenzione mascherata causale), che garantisce che ogni token presti attenzione solo ai token precedenti nella sequenza. Questa modifica preserva la struttura causale necessaria per generare testo in maniera autoregressiva.

Le codifiche posizionali

La codifica posizionale consente alla self-attention di considerare l'ordine dei token nella sequenza, affrontando una limitazione del meccanismo di self-attention, che non tiene conto delle informazioni posizionali nella sequenza.

Le codifiche posizionali vengono combinate con gli embedding dei token in modo che il meccanismo di self-attention possa incorporare nella sequenza l'ordine dei token. Queste codifiche possono essere create utilizzando una formula matematica fissa o apprese dai dati. I metodi più comuni per integrare gli embedding dei token con le codifiche posizionali includono la concatenazione e l'addizione. L'approccio più ampiamente utilizzato consiste nell'aggiungere direttamente le codifiche posizionali agli embedding dei token. È importante notare che le codifiche posizionali dipendono solo dalla posizione all'interno della sequenza.

Con le codifiche posizionali, l'input all'AttentionHead non è più costituito solo dagli embedding dei token, ma dalla somma degli embedding dei token e delle informazioni posizionali.

Il seguente codice implementa una classe che crea sia gli embedding dei token sia le informazioni posizionali e li somma insieme.

```
class Embeddings(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.token_embeddings = nn.Embedding(config.vocab_size,
config.hidden_size)
        self.position_embeddings =
nn.Embedding(config.max_position_embeddings,
config.hidden_size)
        self.layer_norm = nn.LayerNorm(config.hidden_size, eps=1e-12)
        self.dropout = nn.Dropout()
    def forward(self, input_ids):
        # Crea gli ID di posizione per la sequenza di input
        seq_length = input_ids.size(1)
        position_ids = torch.arange(seq_length,
dtype=torch.long).unsqueeze(0)
```

```
# Crea gli embedding dei token e delle posizioni
token_embeddings = self.token_embeddings(input_ids)
position_embeddings = self.position_embeddings(position_ids)
# Combina gli embedding dei token e delle posizioni
embeddings = token_embeddings + position_embeddings
embeddings = self.layer_norm(embeddings)
embeddings = self.dropout(embeddings)
return embeddings
```

In questa classe:

- il metodo `__init__` crea sia gli embedding dei token sia le informazioni posizionali, così come un layer di normalizzazione e un layer di dropout;
- il metodo `forward` combina gli embedding dei token e le informazioni posizionali mediante l'addizione, li normalizza e applica il dropout.

L'attenzione multi-head

Il meccanismo di self-attention coinvolge un singolo insieme di proiezioni per generare le rappresentazioni *query*, *key* e *value*. In pratica, gli autori di Vaswani et al. (2023) hanno ritenuto utile impiegare molteplici insiemi di queste trasformazioni, noti come *attention head* (“teste di attenzione”). Queste “teste” vengono calcolate in parallelo e poi combinate mediante concatenazione, che unisce i vettori per creare una rappresentazione arricchita dell'input. La Figura 5.9 mostra una rappresentazione schematica di questo processo. L'attenzione multi-head consente al modello di concentrarsi simultaneamente su più aspetti dell'input.

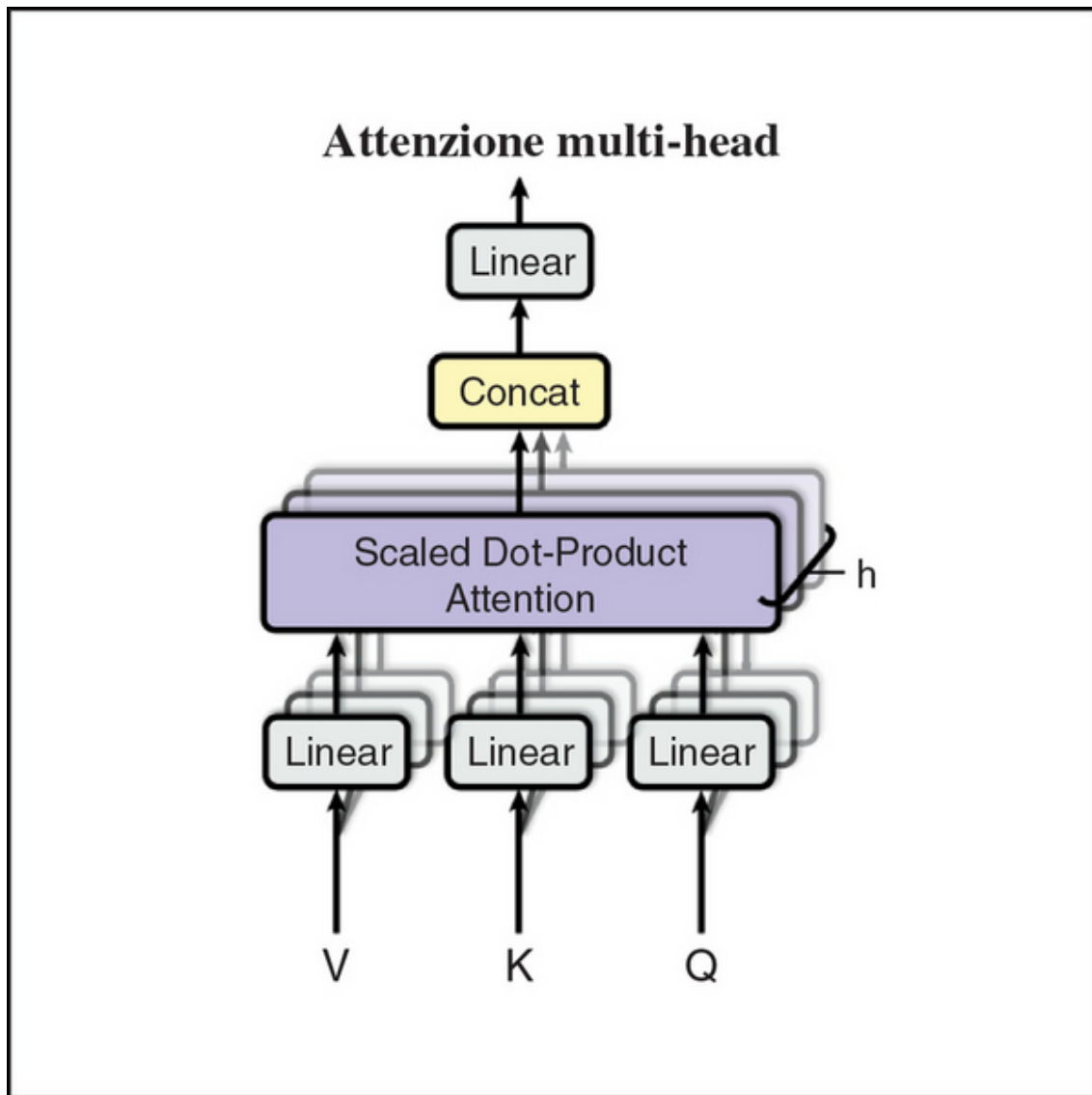


Figura 5.9 Rappresentazione grafica dell'attenzione multi-head. Figura 2 (a destra)
tratta da Vaswani et al. (2023).

L'attenzione multi-head consente di avere dimensioni differenti per le proiezioni *query*, *key* e *value*. Tuttavia, in pratica, ogni “testa di attenzione” ha le stesse dimensioni. Oltre alle dimensioni di queste matrici di proiezione, dobbiamo specificare il numero di teste di attenzione. Per esempio, nell'architettura transformer originale, le

dimensioni degli embedding di input sono di 512 e ci sono 8 teste di attenzione. Questa configurazione imposta le dimensioni per ogni testa come $d_v = d_k = d_q = d_{\text{head}} = 512 / 8 = 64$.

Un'implementazione della seguente classe `MultiHeadAttention` illustra questo concetto:

```
class MultiHeadAttention(nn.Module):
    def __init__(self, config):
        super().__init__()
        embed_dim = config.hidden_size
        num_heads = config.num_attention_heads
        head_dim = embed_dim // num_heads
        self.heads = nn.ModuleList([
            AttentionHead(embed_dim, head_dim) for _ in range(num_heads)
        ])
        self.output_linear = nn.Linear(embed_dim, embed_dim)
    def forward(self, hidden_state):
        x = torch.cat([h(hidden_state) for h in self.heads], dim=-1)
        x = self.output_linear(x)
        return x
```

In questa implementazione:

- il metodo `__init__` crea più teste di attenzione utilizzando la classe `AttentionHead`, basandosi sul numero specificato di teste;
- il metodo `forward` calcola gli output delle teste di attenzione e li concatena, il risultato viene poi passato attraverso un layer lineare, finale.

Il layer feed-forward

Il layer feed-forward aggiunge non linearità al modello e viene applicato all'output del livello di attenzione multi-head. È costituito da due trasformazioni lineari con in mezzo una funzione di attivazione non lineare, come mostrato qui:

```
class FeedForward(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.linear_1 = nn.Linear(config.hidden_size,
config.intermediate_size)
        self.linear_2 = nn.Linear(config.intermediate_size,
```

```

config.hidden_size)
    self.gelu = nn.GELU()
    self.dropout = nn.Dropout(config.hidden_dropout_prob)
    def forward(self, x):
        x = self.linear_1(x)
        x = self.gelu(x)
        x = self.linear_2(x)
        x = self.dropout(x)
        return x

```

Questa classe esegue i seguenti passi:

- il metodo `__init__` definisce due layer lineari, una funzione di attivazione GELU e un dropout per la regolarizzazione;
- il metodo `forward` applica la prima trasformazione lineare, l'attivazione GELU, la seconda trasformazione lineare e infine il dropout prima di restituire l'output.

Nel documento originale, la dimensionalità dell'input e dell'output è pari a 512, indicata nel codice come `hidden_size`, mentre il layer interno ha una dimensionalità pari a 2048, indicata nel codice come `intermediate_size`.

L'attenzione multi-head, i layer feed-forward e le codifiche posizionali, insieme, formano i mattoni fondamentali del modello transformer, consentendogli di catturare dipendenze complesse e di apprendere rappresentazioni ricche dai dati di input.

I blocchi Add e Norm

Come si vede nella Figura 5.8 (pagine precedenti), l'architettura transformer integra i blocchi Add e Norm. Questi blocchi eseguono due operazioni chiave: *Add* applica una connessione residua (cioè, aggiunge l'input di un layer al suo output per migliorare il flusso del gradiente e facilitare l'addestramento); *Norm* implica la normalizzazione a layer (vedi Goodfellow et al., 2016, per maggiori dettagli). Come mostrato in figura, gli autori utilizzano una

connessione residua attorno ai blocchi sia di attenzione multi-head sia feed-forward.

Il layer encoder del transformer

Ora che abbiamo trattato i componenti di base dell'architettura transformer, siamo pronti per implementare il layer encoder del transformer, mostrato a sinistra nella Figura 5.8. Il layer encoder, o blocco encoder, è costituito da un layer di embedding, un meccanismo di attenzione multi-head e una rete feed-forward. Come abbiamo detto, sono inclusi anche la normalizzazione a batch e le connessioni residue.

Segue l'implementazione del `TransformerEncoderLayer`, che incorpora la normalizzazione a layer e le connessioni residue per un migliore flusso del gradiente:

```
class TransformerEncoderLayer(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.layer_norm_1 = nn.LayerNorm(config.hidden_size)
        self.layer_norm_2 = nn.LayerNorm(config.hidden_size)
        self.attention = MultiHeadAttention(config)
        self.feed_forward = FeedForward(config)
        def forward(self, x):
            # Applica la normalizzazione a layer e poi copia l'input in query, key,
            value
            hidden_state = self.layer_norm_1(x)
            # Applica l'attenzione con una connessione residua
            x = x + self.attention(hidden_state)
            # Applica il layer feed-forward con una connessione residua
            x = x + self.feed_forward(self.layer_norm_2(x))
            return x
```

Una nota sulla regolarizzazione: come descritto dagli autori dell'articolo sui transformer, e come potreste aver notato nell'implementazione del codice, per la regolarizzazione viene utilizzato il dropout (vedi Capitolo 3). Il dropout viene applicato attorno ai blocchi *Add* e *Norm*, così come alla somma degli

embedding e delle codifiche posizionali nello stack sia dell'encoder sia del decoder (Vaswani et al., 2023).

L'encoder del transformer, completo

Il modello transformer viene costruito impilando più moduli `TransformerEncoderLayer`. Ogni layer elabora gli embedding di input e li raffina ulteriormente, consentendo al modello di apprendere relazioni complesse presenti nei dati.

Ecco come possiamo implementare l'encoder del transformer completo:

```
class TransformerEncoder(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.embeddings = Embeddings(config)
        self.layers = nn.ModuleList([TransformerEncoderLayer(config)
                                     for _ in range(config.num_hidden_layers)])
    def forward(self, x):
        x = self.embeddings(x)
        for layer in self.layers:
            x = layer(x)
        return x
```

Obiettivo del modello

Ora che abbiamo definito l'architettura dell'encoder del transformer, la domanda successiva è: per quali compiti possiamo utilizzarlo? Come strutturiamo l'output del modello per risolvere questi compiti e come addestriamo il modello?

L'approccio tipico consiste nell'utilizzare il modello encoder del transformer come corpo principale del nostro modello e collegargli un decoder, come nei compiti di traduzione linguistica, oppure una "testa" (*head*) adatta al compito in questione. Per esempio, per un compito di classificazione, possiamo collegare una testa appropriata, un layer di rete che prenda l'output del transformer e generi una distribuzione di probabilità sulle possibili classi. Questo ci consentirà

di utilizzare un algoritmo di ottimizzazione per trovare i parametri del modello che spiegano al meglio la relazione tra input e target nel nostro dataset.

Consideriamo alcuni esempi per comprendere meglio questo concetto.

- *Analisi del sentiment*: per questo compito, useremmo una testa di classificazione che elabora l'output del modello transformer e predice il sentiment (per esempio, positivo, negativo o neutro) di un dato testo di input. Il modello impara ad associare determinati pattern presenti nel testo a specifici sentiment.
- *Modellazione linguistica*: il modello BERT, introdotto da (Devlin et al. 2019), è un buon esempio di questo approccio. BERT (*Bidirectional Encoder Representations from Transformers*) utilizza l'architettura encoder del transformer. Uno degli obiettivi primari di addestramento di BERT è il compito MLM (*Masked Language Model*), dove certe parole di una frase sono mascherate casualmente, e il modello viene addestrato a predire le parole mancanti. Nello specifico, viene mascherato il 15% delle parole della frase. Di queste, l'80% è sostituito dal token [MASK], il 10% rimane invariato e il restante 10% è sostituito con una parola scelta a caso. Per risolvere questo compito, useremmo una testa di classificazione che elabora l'output del transformer e genera una distribuzione di probabilità in base al vocabolario dell'input.

Per adattare l'encoder del transformer a un compito di classificazione come l'analisi del sentiment, aggiungiamo una testa di classificazione sopra l'encoder. Come esempio, la testa implementata qui è costituita da un layer di dropout seguito da un layer pienamente connesso che mappa la rappresentazione codificata sul numero desiderato di etichette:

```

class TransformerForSequenceClassification(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.encoder = TransformerEncoder(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size,
config.num_labels)
    def forward(self, x):
        x = self.encoder(x)[: , 0, :] # seleziona lo stato nascosto del
token [CLS]
        x = self.dropout(x)
        x = self.classifier(x)
        return x

```

Qui:

- `self.encoder` è l'encoder del transformer che elabora la sequenza di input;
- `self.dropout` è il layer utilizzato per prevenire l'overfitting durante l'addestramento;
- `self.classifier` è il layer pienamente connesso che mappa l'output dell'encoder sul numero desiderato di classi.

Come abbiamo appena mostrato, il modello di classificazione sfrutta l'encoder del transformer per creare potenti rappresentazioni del testo di input. Collegando una testa di classificazione, possiamo addestrare questo modello per vari compiti, come l'analisi del sentiment, la modellazione linguistica, il *question answering* e altro. Tuttavia, il modo tipico in cui questi modelli vengono addestrati per risolvere compiti, come l'analisi del sentiment, coinvolge un processo di addestramento in due fasi. Nella prima fase, di pre-addestramento (*pre-training*), l'encoder del transformer viene addestrato su un compito di modellazione linguistica. In un secondo step, noto come *fine-tuning*, il modello viene ulteriormente addestrato su un dataset specifico relativo al compito, e gli viene aggiunta una testa di classificazione o di regressione per risolvere il

problema specifico. Maggiori dettagli su questo processo, insieme a esempi pratici, possono essere trovati nei Capitoli 9 e 10.

In effetti, la struttura descritta qui è essenzialmente un'implementazione del modello BERT, che è diventato uno strumento fondamentale per molte applicazioni di elaborazione del linguaggio naturale grazie alla sua versatilità e alle sue potenti prestazioni su un'ampia gamma di compiti.

Per scopi complementari, il notebook allegato [*DecoderOnly Transformers – Notebook*] mostra l'implementazione di un modello transformer solo decoder, in particolare GPT, per complementare la discussione sui transformer solo encoder presentata qui.

Dal transformer per la NLP ai transformer per le serie temporali

Dato il grande successo dei modelli transformer nel catturare le dipendenze a lungo termine nei dati sequenziali – come testi, audio e video – è naturale chiedersi se possano essere applicati anche ad applicazioni relative a serie temporali, poiché i dati a serie temporali, come i testi, sono intrinsecamente sequenziali.

I transformer sono noti per avere diversi vantaggi, che discuteremo più in dettaglio nel Capitolo 10. Per menzionarne alcuni, questi modelli migliorano le loro prestazioni nei compiti di elaborazione del linguaggio naturale all'aumentare delle dimensioni del modello, del volume dei dati di addestramento e del tempo di addestramento (vedi Kaplan et al., 2020). Inoltre, sono altamente parallelizzabili e adattabili per molteplici compiti, come dimostrato da Devlin et al. (2019). Queste qualità stanno motivando l'industria a esplorare se i transformer possano raggiungere un livello simile di successo anche nella modellazione di problemi di serie temporali.

Per i compiti di modellazione linguistica, i transformer vengono addestrati con l'obiettivo di predire il token successivo in una sequenza basandosi sui token precedenti. Questo è concettualmente simile al compito di prevedere le serie temporali, dove vogliamo predire l'elemento successivo della serie basandoci sui valori precedenti, o eseguire quella che viene chiamata "previsione one-step-ahead". Nella terminologia delle serie temporali, potremmo dire che il compito di modellazione linguistica è simile alla previsione di una serie temporale discreta, dove gli elementi della sequenza assumono valori tratti da un vocabolario finito, con un orizzonte temporale pari a 1. Data questa similarità di obiettivi, potremmo chiederci quali modifiche siano necessarie per applicare i transformer alle serie temporali.

Come accennato, una differenza chiave consiste nel fatto che, la maggior parte delle volte, i valori di una serie temporale sono quantità continue e illimitate, a differenza di una sequenza di testi, dove gli elementi (i token) assumono valori tratti da un vocabolario finito. Un'idea semplice per adattare i transformer ai dati a serie temporali consiste nel convertire i valori continui della serie in un insieme finito di valori, ancora token, sfruttando l'analogia con le applicazioni testuali, un po' come le parole vengono rappresentate nei compiti linguistici. Questo approccio è adottato in *Chronos: Learning the Language of Time Series* (Ansari et al., 2024).

Discretizzazione dei dati a serie temporali: l'approccio Chronos

Nell'articolo *Chronos: Learning the Language of Time Series* (Ansari et al., 2024), gli autori propongono un approccio per utilizzare i transformer sui dati a serie temporali tokenizzando i valori delle serie temporali stesse. Questo è ottenuto prima

normalizzando e poi quantizzando la serie temporale, per trasformare i valori continui in “token” discreti che possono essere elaborati da architetture transformer standard.

1. *Normalizzazione*: ogni serie temporale viene normalizzata per mitigare le differenze di scala tra le serie e per migliorare l’ottimizzazione del modello. Gli autori utilizzano il *mean scaling*, in cui ogni serie viene normalizzata in base al valore assoluto medio storico all’interno del “contesto storico”, il numero di input del transformer.
2. *Quantizzazione*: i valori normalizzati vengono poi quantizzati in un numero fisso di bin, convertendo i valori continui in un insieme discreto di valori, o token. Questi token discreti rappresentano la serie temporale in un modo simile a come il testo viene rappresentato da una sequenza di token.

Una volta che ogni valore è stato tokenizzato, gli *embedding dei token* e le *informazioni posizionali* vengono creati nello stesso modo dei compiti di modellazione linguistica e combinati mediante addizione, per formare gli embedding di input per il transformer. Al momento dell’inferenza, poiché il modello restituisce token, viene poi applicato un processo di *dequantizzazione* e *denormalizzazione*, per convertire i token discreti delle previsioni nuovamente in valori continui.

Questo metodo consente l’uso di architetture transformer standard, come T5 (Raffel et al., 2020; encoder-decoder) e GPT-2 (Radford et al., 2019; solo decoder), senza modifiche all’architettura. Gli autori di Chronos hanno dimostrato che questo approccio è efficace su un’ampia gamma di dati a serie temporali, raggiungendo risultati interessanti su dati mai visti prima. Per addestrare i loro modelli, gli autori hanno compilato dati a serie temporali da vari domini, inclusi il retail, l’energia, la finanza, la sanità e la climatologia, migliorando

la generalizzazione tramite l’incorporamento di dati sintetici generati attraverso tecniche di *data augmentation*.

Nel notebook allegato [*Chronos Notebook*], esploriamo l’uso di Chronos sul dataset *Exchange Rate*, che contiene i tassi di cambio giornalieri per le valute di otto paesi (Australia, Regno Unito, Canada, Svizzera, Cina, Giappone, Nuova Zelanda e Singapore) dal 1990 al 2016 (dati *Exchange Rate*). Parte di questi dati è stata utilizzata per addestrare Chronos, e secondo i riferimenti forniti nel loro repository e nel loro articolo, vengono valutate le prestazioni di Chronos sulla parte non vista dei dati, vengono evidenziate alcune delle sue limitazioni e viene confrontato con alcuni altri semplici esempi.

Input continuo per i transformer: l’approccio Lag-Llama

Un approccio alternativo alla discretizzazione delle serie temporali consiste nel trattare i dati così come sono, utilizzando valori continui in modo che la rete prenda input continui e restituisca output continui. Questo approccio non solo richiede la pre-elaborazione dell’input per creare gli “embedding dei token”, ma anche di eseguire adattamenti all’output, poiché ora stiamo risolvendo un compito di regressione e non di classificazione, come nella modellazione linguistica. Questo è l’approccio adottato nell’articolo *Lag-Llama: Towards Foundation Models for Probabilistic Time Series Forecasting* (Rasul et al., 2024).

In questo articolo, gli autori adattano il modello Llama (Touvron et al., 2023), un’architettura transformer solo decoder, perché restituisca valori continui piuttosto che token discreti, introducendo un layer finale che predice i parametri di distribuzioni continue, specificamente una distribuzione t , piuttosto che i parametri di una

distribuzione categorica tipica nei problemi di classificazione multi-classe.

Le modifiche in Lag-Llama includono quanto segue.

1. *Livello di output continuo probabilistico*: invece di predire i parametri di una distribuzione categorica che assegna la massa di probabilità su un vocabolario finito di token, Lag-Llama restituisce i parametri di una distribuzione t , consentendo al modello di fare previsioni probabilistiche su valori continui.
2. *Feature ritardate (lag) e temporali*: oltre a utilizzare il passo temporale corrente, Lag-Llama incorpora feature ritardate – valori precedenti a vari intervalli (per esempio, trimestrali, mensili, settimanali) – per arricchire l’input e catturare stagionalità e pattern periodici. Il modello include anche feature come l’ora, il giorno e il mese e così via, derivate dal timestamp, che forniscono informazioni contestuali aggiuntive su ogni punto temporale. Gli autori includono anche statistiche di riepilogo della serie per fornire maggiore contesto.
3. *Normalizzazione e controllo delle anomalie (outlier)*: Lag-Llama normalizza ogni serie temporale e mitiga l’effetto dei valori anomali utilizzando l’intervallo interquartile (IQR, *Interquartile Range*).

Arricchendo l’input corrente con feature ritardate e temporali, gli autori forniscono al modello Llama più informazioni contestuali per catturare efficacemente le dipendenze a lungo e a breve termine. Nella nostra esperienza con compiti di previsione che coinvolgono grandi dataset di serie temporali multivariate, questo tipo di ingegneria delle feature aiuta i modelli a catturare caratteristiche stagionali e migliora le prestazioni di previsione.

Nell’approccio Lag-Llama, i valori correnti e ritardati della serie, insieme alle feature temporali ingegnerizzate, vengono concatenati in

un vettore (vedi Figura 5.10). Questo vettore viene poi proiettato nello spazio di “embedding dei token” del modello, creando una “rappresentazione dei token” analoga a quella delle applicazioni di elaborazione del linguaggio naturale, che serve come input per il modello transformer, come mostrato nella Figura 5.11.

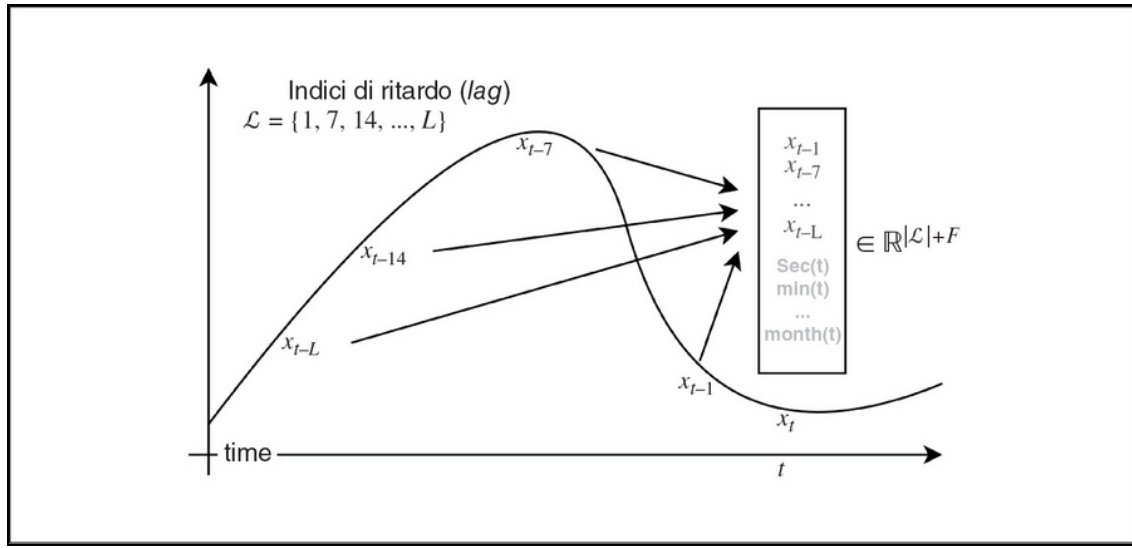


Figura 5.10 Un grafico che mostra come i valori correnti e ritardati della serie, combinati con feature temporali ingegnerizzate, vengono concatenati in un vettore. Ogni componente del vettore rappresenta $|\mathcal{L}|$ valori passati di x_t (scuri), insieme a F covariate temporali (feature temporali) derivate dal timestamp (chiaro). Figura 1 tratta da Rasul et al. (2024).

Nel notebook allegato [*Lag-Llama Notebook*], esploriamo l’uso di Lag-Llama sullo stesso dataset *Exchange Rate*. Per Lag-Llama, poiché secondo gli autori questo dataset non è stato utilizzato per l’addestramento del modello, valuteremo la sua performance “zero-shot”. Come esercizio aggiuntivo, addestreremo ulteriormente (*fine-tuning*) il modello su questo dataset e osserveremo come vengono influenzate le sue prestazioni. In modo simile a Chronos, evidenzieremo alcune delle sue limitazioni e confronteremo le sue prestazioni con semplici esempi.

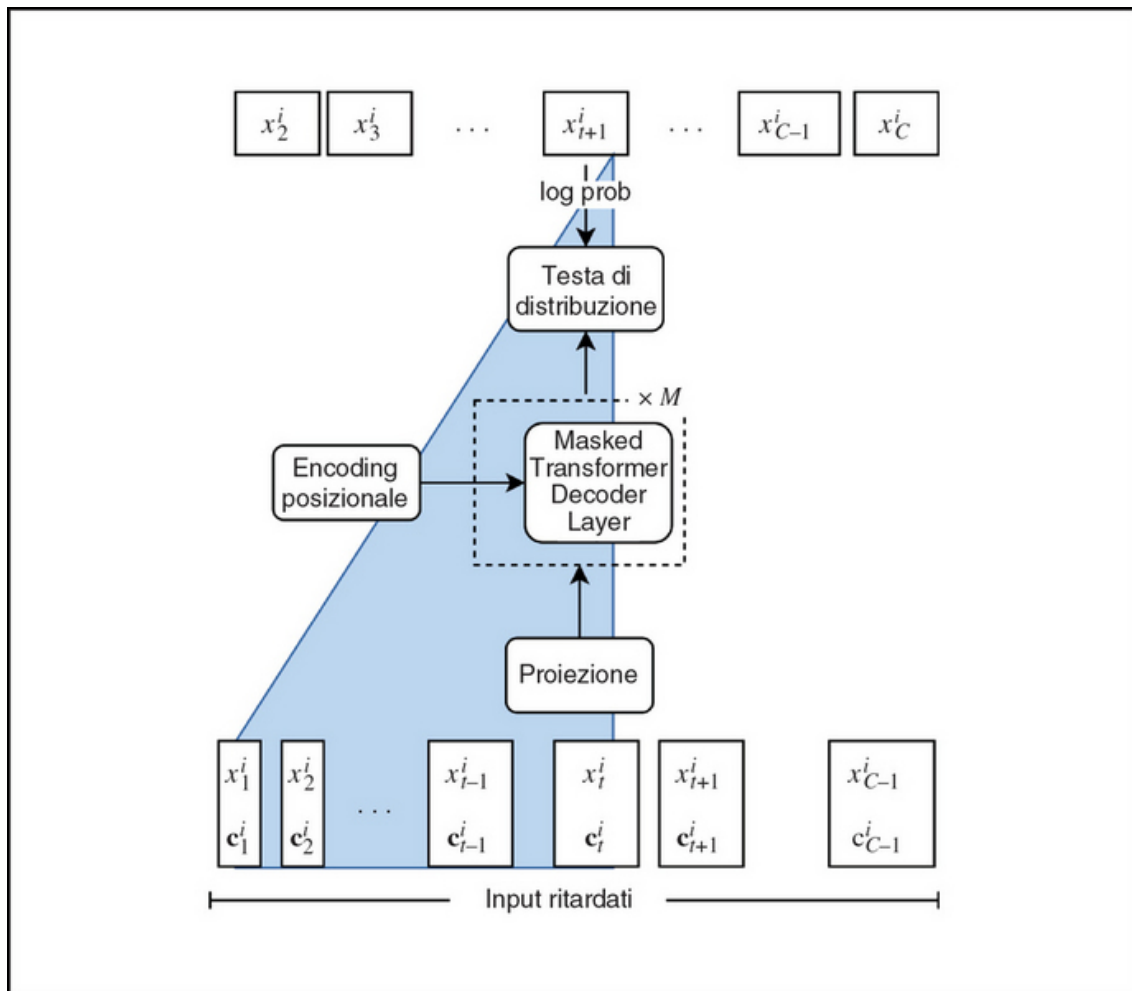


Figura 5.11 Un grafico che mostra come l'input del modello, comprendente il valore della serie temporale al timestep t con ritardo $|\mathcal{L}|$, F feature temporali e statistiche di riepilogo, viene proiettato nello spazio di "embedding dei token" del transformer, creando una "rappresentazione dei token" simile alle applicazioni di elaborazione del linguaggio naturale. Figura 2 tratta da Rasul et al. (2024).

Innovazioni e approcci

Gli esempi Chronos e Lag-Llama illustrano alcune delle tecniche più comuni per adattare le architetture transformer per applicazioni a serie temporali, che in questi casi hanno richiesto poche o nessuna modifica alle architetture transformer stesse. In questi casi, gran

parte del lavoro ruota attorno alle tecniche di pre-elaborazione dei dati a serie temporali e l'arricchimento delle informazioni contestuali, utilizzando metodi come la normalizzazione, la quantizzazione e l'aggiunta di valori ritardati e feature temporali. Abbiamo anche visto come certe scelte di pre-elaborazione, come applicare o meno la quantizzazione, influenzino il modo in cui il problema viene inquadrato: come un problema di classificazione (come in Chronos, che non richiede modifiche alle architetture transformer standard e tratta il problema in modo simile ai compiti testuali) o come un problema di regressione (come in Lag-Llama, dove gli autori hanno modificato l'ultimo layer dell'architettura per modellare una distribuzione di probabilità su valori continui).

Sebbene questi esempi mostrino alcuni adattamenti interessanti, non sono affatto rappresentativi dell'intera gamma di approcci che possono essere applicati per migliorare le prestazioni del modello. In particolare, non abbiamo ancora discusso degli approcci che si concentrano sul migliorare la complessità spaziale e temporale dei transformer.

Negli ultimi anni, sono state proposte numerose architetture transformer per le serie temporali. Per quanto ne sappiamo, le principali aree di lavoro riguardano quanto segue.

1. *Normalizzazione dell'input e informazioni contestuali:*
quest'area include il ridimensionamento (*scaling*) dei dati a serie temporali per aiutare l'ottimizzazione del modello, l'applicazione di varie tecniche di normalizzazione, la gestione dei valori anomali (outlier) e la quantizzazione della serie. Il contesto può essere arricchito incorporando valori ritardati o creando feature temporali per catturare importanti effetti stagionali e di calendario, oppure incorporando indicatori macroeconomici, nonché indicatori da mercati o indici che non

fanno parte della serie temporale target – alcuni dei quali sono stati delineati nel Capitolo 3. Queste informazioni vengono tipicamente proiettate per creare “embedding dei token”, che servono come input del transformer. Per esempio, alcuni modelli utilizzano come layer di proiezione dei layer convoluzionali unidimensionali, come nell’architettura Informer (Zhou et al., 2021).

2. *Encoding posizionali*: i transformer sono per natura invarianti alla permutazione, il che pone un problema quando si gestiscono dati sequenziali come le serie temporali. La codifica posizionale è cruciale per trasmettere informazioni sull’ordine nei compiti nell’elaborazione del linguaggio naturale e diventa ancora più vitale per le applicazioni a serie temporali, dove l’ordine della sequenza è essenziale. Le soluzioni spaziano dalle codifiche posizionali fisse, come nel transformer originale (Vaswani et al., 2023), agli embedding posizionali apprendibili, come quello discusso nel Paragrafo 5.3.5.4.
3. *Efficienza del layer di attenzione*: migliorare l’efficienza del meccanismo di attenzione è un’area di ricerca attiva, specialmente per i dati a serie temporali, che spesso coinvolgono sequenze storiche lunghe e previsioni su orizzonti estesi. Migliorare la complessità computazionale di questo layer può aiutare a ridurre la latenza di inferenza e può aiutarci a soddisfare i requisiti di sistema di specifiche applicazioni. Ricordiamo dalla nostra discussione precedente nel paragrafo “Scaled Dot-Product Attention” che il meccanismo di attenzione del transformer “puro” (Vaswani et al., 2023), calcola un prodotto scalare tra ogni query e tutte le chiavi (*keys*), cosa che produce una complessità temporale di $\mathcal{O}(N^2)$ per N input. Per ridurre questa complessità, sono stati proposti diversi metodi.

Per esempio, il modello Informer (Zhou et al., 2021) riduce il costo computazionale del layer di attenzione calcolando l'attenzione per un sottoinsieme di $\log N$ “query attive”, piuttosto che per tutte le N query, raggiungendo così una complessità temporale $\mathcal{O}(N \log N)$.

4. *Nuove varianti di architetture transformer*: i ricercatori hanno sviluppato varianti per tenere conto della natura multi-risoluzione delle serie temporali. Per esempio, l'architettura Informer effettua un downsampling della serie inserendo layer di max-pooling tra i blocchi di attenzione (Zhou et al., 2021). Il modello Pyraformer (Liu et al., 2021) introduce un modulo di attenzione piramidale che riassume le feature a varie risoluzioni, cosa che gli consente di modellare efficacemente le dipendenze temporali su diverse scale temporali.

Per una panoramica più dettagliata delle architetture transformer più note per le serie temporali al momento della stesura di questo testo, raccomandiamo l'eccellente risorsa *Transformers in Time Series: A Survey* (Wen et al., 2023). Inoltre, per un'applicazione dei transformer come mattoni nella costruzione di portafogli, vedi Cong et al. (2021).

Adattamento del modello (Model Fitting)

I modelli autoregressivi spesso sono definiti modelli di massima verosimiglianza (*Maximum Likelihood*), poiché si basano principalmente sulla stima di massima verosimiglianza per la stima dei parametri. Tutti i modelli discussi in questo capitolo utilizzano la MLE (*Maximum Likelihood Estimation*) come metodo principale per l'apprendimento dei parametri. Una volta definita una classe di

modelli parametrizzata da θ , denotata come p_θ , diventa essenziale stabilire una metrica che valuti la capacità del modello di approssimare la distribuzione di probabilità dei dati, denotata come p_{data} . Naturalmente, abbiamo già discusso l'adattamento MLE nel Capitolo 3, ma ecco un nuovo modo di considerarlo.

Considerando che i nostri obiettivi includono il calcolo delle densità di probabilità, la determinazione di strutture nascoste e la generazione di nuovi campioni, determinare una singola metrica di punteggio del modello che soddisfi tutti questi obiettivi può essere difficile. Intuitivamente, se un modello può approssimare la distribuzione p_{data} , teoricamente dovrebbe comportarsi bene anche nelle altre attività. Pertanto, il nostro obiettivo è allineare il più possibile la distribuzione del modello p_θ con p_{data} .

Per quantificare la vicinanza tra due distribuzioni, utilizziamo la divergenza di Kullback-Leibler, $KL(P||Q)$, definita come:

$$KL(P||Q) = \mathbb{E}_{\mathbf{x} \sim P} \left[\log \frac{P(\mathbf{x})}{Q(\mathbf{x})} \right]$$

(5.18)

Il nostro obiettivo è quello di minimizzare la divergenza di Kullback-Leibler tra la distribuzione empirica dei dati e la distribuzione del nostro modello:

$$\begin{aligned} KL(p_{\text{data}}||p_\theta) &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})} \right] \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log p_{\text{data}}(\mathbf{x}) - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log p_\theta(\mathbf{x}) \end{aligned}$$

(5.19)

Il termine

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log p_{\text{data}}(\mathbf{x})$$

non dipende da θ e quindi è irrilevante per il processo di ottimizzazione. Pertanto, la funzione obiettivo, funzione di costo o punteggio da minimizzare è data da:

$$\operatorname{argmin}_{\theta} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} -\log p_{\theta}(\mathbf{x})$$

(5.20)

Dato che abbiamo accesso solo a campioni da p_{data} , la stima empirica è:

$$\operatorname{argmin}_{\theta} \mathcal{L}(\theta) = \frac{1}{N} \sum_{\mathbf{x}_n \in \mathcal{D}} -\log p_{\theta}(\mathbf{x}_n)$$

(5.21)

Questa formulazione corrisponde alla stima di massima verosimiglianza, o a minimizzare la log-verosimiglianza negativa (*Negative Log Likelihood*), che mira ad aumentare la massa di probabilità nei punti presenti nel nostro dataset di addestramento mentre assegna probabilità più basse a quelli assenti.

Per minimizzare questa funzione obiettivo, l'algoritmo di ottimizzazione tipicamente utilizzato è la discesa stocastica del gradiente (*Stochastic Gradient Descent* - SGD):

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \tilde{\mathcal{L}}(\theta)$$

(5.22)

Qui

$$\tilde{\mathcal{L}}(\theta)$$

rappresenta la loss calcolata utilizzando un batch o sottoinsieme dei dati, cosa che facilita il calcolo efficiente del gradiente e gli aggiornamenti del modello.

Conclusioni

Come abbiamo esplorato in questo capitolo, i modelli autoregressivi sono strumenti potenti per modellare sequenze a elevata dimensionalità. Abbiamo iniziato con approcci classici, come i modelli autoregressivi lineari, siamo progrediti verso le reti neurali ricorrenti e abbiamo concluso con i rivoluzionari transformer, che hanno rimodellato i campi dell'elaborazione del linguaggio naturale, del riconoscimento vocale, della computer vision e altro. Tuttavia, al momento della stesura di questo testo, i transformer per applicazioni a serie temporali non hanno ancora raggiunto il livello di performance osservato in altri domini, come i compiti sul testo e sulla visione.

Adattare i transformer per le serie temporali potrebbe richiedere ulteriori ottimizzazioni e lavoro empirico per raggiungere lo stesso livello di successo. In tutto questo capitolo, abbiamo evidenziato quei punti che potrebbero fornire percorsi di miglioramento. Affrontando le problematiche uniche dei dati a serie temporali, la ricerca e lo sviluppo futuri possono aiutare a colmare l'attuale divario di performance. Sulla base dei risultati tratti dalla letteratura, certe soluzioni personalizzate semplici, come quelle basate su modelli autoregressivi lineari, possono ancora superarli, in particolare quando si prevedono serie temporali finanziarie. Tuttavia, i modelli transformer offrono molti vantaggi, e il loro utilizzo dipende dal nostro obiettivo principale: la performance, la scalabilità per gestire più serie temporali eterogenee, soluzioni personalizzate o generali, o altri fattori.

Quanto all'uso dei transformer per lo sviluppo di strategie di trading, essi sono strumenti potenti per pre-elaborare i dati, fornendo spunti utili per strategie sistematiche o per gestori di investimento discrezionali. Come caso d'uso, nel Capitolo 9, mostreremo come

sfruttare questo potente modello per elaborare dati alternativi e sviluppare strategie di trading sistematiche.

Capitolo 6

Modelli profondi a variabili latenti

Nel capitolo precedente, abbiamo visto come i modelli autoregressivi sfruttino i valori precedenti in una sequenza per rappresentare distribuzioni di probabilità altamente complesse e a elevata dimensionalità. Al contrario, i *modelli a variabili latenti* adottano un approccio distinto per approssimare la distribuzione di probabilità p_{data} : non utilizzano i valori precedenti nella sequenza, ma ipotizzano l'esistenza di variabili nascoste non osservabili che influenzano ogni punto di dati nel dataset.

In questo capitolo, ci concentriamo su modelli in cui la conoscenza delle variabili nascoste o latenti *spiega solo parzialmente i dati osservati*. Qui, la relazione tra le variabili nascoste e le osservazioni non è completamente deterministica – introduciamo un certo grado di incertezza nel passo di modellazione riguardo a come le variabili latenti generano i dati osservati. I modelli in cui, invece, le variabili latenti determinano completamente i dati osservati – dove quindi esiste una mappatura deterministica e invertibile tra le variabili nascoste e i dati – saranno l'argomento del Capitolo 7. Questi sono noti come modelli flow o modelli invertibili. Nel Capitolo 8, introdurremo un altro tipo di modello a variabili latenti: le GAN (*Generative Adversarial Networks*). A differenza dei modelli discussi

qui o nel prossimo capitolo, le GAN non modellano esplicitamente la distribuzione di probabilità dei dati.

In questo capitolo, ci concentriamo su modelli in cui la dimensionalità di queste variabili latenti è significativamente minore di quella dei dati osservati. Il fatto di introdurre variabili nascoste nei modelli aiuta a modellare una distribuzione multimodale a elevata dimensionalità come l'integrale del prodotto di due distribuzioni più semplici. Comunemente, queste distribuzioni più semplici appartengono alla famiglia esponenziale, il che semplifica il processo di modellazione complessivo, pur mantenendo la flessibilità nel catturare la struttura sottostante dei dati. L'*Hidden Markov Model* che abbiamo introdotto nel Capitolo 3 è ovviamente un semplice esempio di tale modello a variabili latenti (dove c'è una certa dinamica tra le variabili latenti), ma vedremo che anche la PCA (*Principal Component Analysis*) può essere vista come un esempio di modello a variabili latenti. I modelli profondi a variabili latenti, come i noti autoencoder variazionali discussi in questo capitolo, appartengono alla categoria dei modelli espliciti che stimano le densità in modo approssimato, come mostra la Figura 6.1. Questo capitolo sfrutterà l'intera strumentazione del deep learning per creare una varietà molto più ricca di questi modelli.

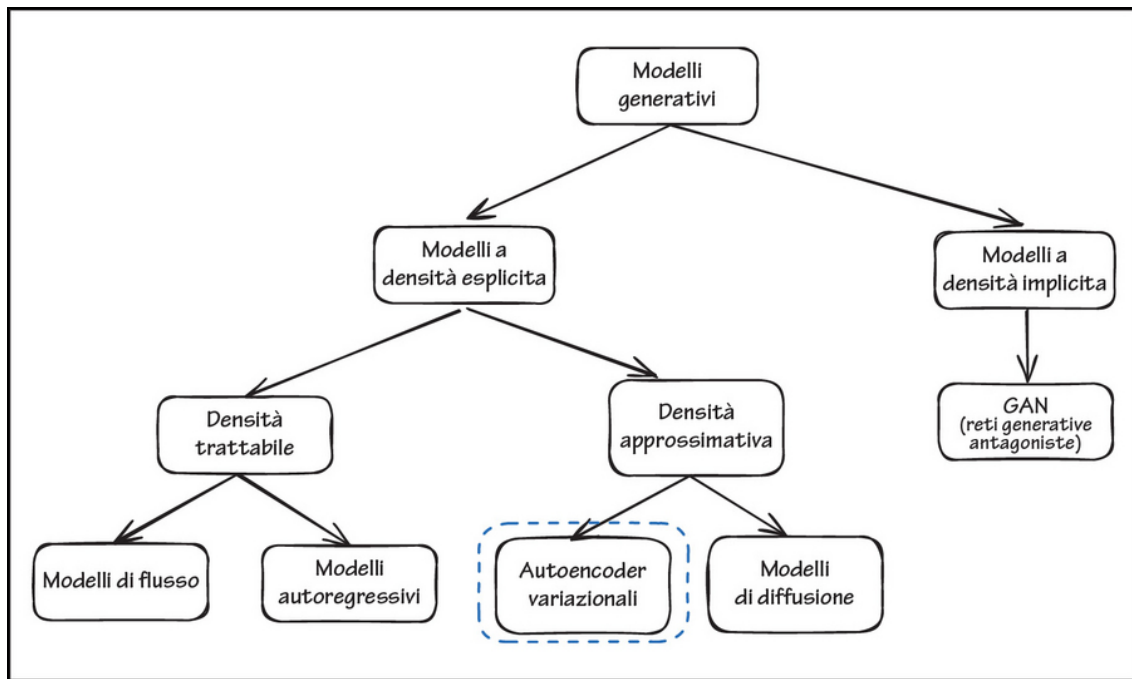


Figura 6.1 Grafico ad albero della tassonomia dei modelli: gli autoencoder variazionali.

Introduzione

Per illustrare l'utilità dei modelli a variabili latenti, considerate la loro applicazione nella finanza, in particolare per modellare e stimare la matrice di covarianza di serie storiche multivariate. Le matrici di covarianza sono fondamentali in finanza, utilizzate per compiti come l'analisi di regressione, la stima del rischio, l'ottimizzazione del portafoglio e la simulazione di scenari attraverso metodi Monte Carlo (López de Prado, 2020).

Ipotizziamo di avere una lunga storia di rendimenti da D serie temporali, ciascuna delle quali rappresenta i rendimenti storici di un asset. Un approccio semplice per modellare queste serie temporali multivariate consiste nell'ignorare la correlazione seriale delle singole serie temporali e supporre una covarianza costante tra gli

asset, nel tempo. A ogni indice temporale t , i rendimenti potrebbero essere modellati come un'estrazione da una distribuzione di probabilità costante, per esempio, $\mathbf{x}_t \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, dove $\mathbf{x}_t \in \mathbb{R}^D$ rappresenta il rendimento all'indice temporale t , $\boldsymbol{\mu}$ è il vettore dei rendimenti medi e $\boldsymbol{\Sigma}$ è la matrice di covarianza dei rendimenti.

Per fornire un contesto, considerate di modellare i rendimenti giornalieri dei componenti dell'indice S&P 500, impostando così il numero di serie temporali a $D = 500$. Qui, lo stimatore più semplice per la matrice di covarianza richiederebbe il calcolo di $D(D + 1) / 2 = 500(501) / 2 = 125.250$ valori di covarianza. Questo grande numero di parametri può diventare problematico se i dati giornalieri disponibili sono limitati.

Un altro modo di pensare alla modellazione di $\mathbf{x}_t$ consiste nel supporre che sia influenzato da un insieme più piccolo di fattori sottostanti e non osservati, che spiegano parzialmente le osservazioni.

Questi fattori possono somigliare ai tradizionali modelli multi-fattore, come il modello a tre fattori Fama-French di cui abbiamo parlato nel Capitolo 2, che spiega i rendimenti degli asset utilizzando fattori come il mercato, il fattore SMB (*Small Minus Big*) e il fattore HML (*High Minus Low*). Tuttavia, una differenza chiave tra questi modelli tradizionali e quelli che esploreremo in questo capitolo è che, nel caso tradizionale, i fattori sono osservabili – possono essere misurati e, spesso, possono anche essere negoziati, fornendo un valore significativo per i trader. Al contrario, i modelli introdotti in questo capitolo trattano fattori non osservabili. Queste sono le variabili nascoste, o latenti, termini che useremo come sinonimi in questo capitolo e che sono comuni in letteratura.

Per esempio, l'analisi fattoriale (FA, *Factorial Analysis*), un tipo di modello a variabili latenti, scompone la matrice di covarianza come $\boldsymbol{\Sigma} = \boldsymbol{\Lambda} \boldsymbol{\Lambda}^T + \boldsymbol{\Sigma}_0$, dove $\boldsymbol{\Lambda} \in \mathbb{R}^{D \times M}$ è chiamata matrice dei carichi, M è il

numero di fattori (un iperparametro nell'analisi fattoriale) con $M \ll D$, e Σ_0 è una matrice diagonale $D \times D$. Nella terminologia finanziaria, possiamo pensare a $\Lambda\Lambda^T$ come alla matrice di covarianza sistematica, che rappresenta la parte della varianza dei rendimenti dell'asset spiegata dai fattori, e a Σ_0 come alla matrice di covarianza idiosincratICA – la varianza dei rendimenti dell'asset non spiegata dai fattori. Questo tipo di scomposizione della varianza è particolarmente utile per la modellazione del rischio nelle applicazioni di costruzione del portafoglio, dove potremmo essere interessati a ridurre l'esposizione della nostra strategia a certi fattori o a porre un limite superiore alla varianza sistematica e così via. Questa formulazione della scomposizione della varianza permette di incorporare facilmente vincoli come questi nel processo di ottimizzazione del portafoglio.

Questo metodo riduce significativamente il numero di parametri da stimare, da $D(D + 1) / 2$ a $D \times M + D$, da $\mathcal{O}(D^2)$ a $\mathcal{O}(D \times M)$. Per esempio, se $M = 10$, il conteggio dei parametri scende da 125.250 a 5.010, una riduzione di 25 volte. Poiché questo approccio di modellazione ipotizza che le osservazioni siano spiegate da un insieme più piccolo di fattori, la matrice di covarianza è intrinsecamente di rango basso. Ciò non solo riduce il numero di parametri da stimare, ma diminuisce anche la memoria richiesta per memorizzare la matrice di covarianza rispetto ad approcci di modellazione più semplici.

Ciò che affascina dei modelli a variabili latenti è che, anche se i fattori non sono direttamente osservati, facendo ipotesi probabilistiche su come si comportano, possiamo inferire i loro valori e stimare i parametri del modello. Hamlet ricorda quando studiò per la prima volta questo tipo di modelli; ne rimase assolutamente affascinato. Nel contesto dell'analisi fattoriale, le cose diventano

ancora più interessanti quando introduciamo delle dinamiche nelle variabili latenti, portando a modelli come il noto Filtro di Kalman.

Come vedremo in questo capitolo, quando consideriamo i modelli profondi a variabili latenti, gli algoritmi utilizzati per stimare questi modelli diventano ancora più interessanti. I concetti e gli strumenti richiesti per svilupparli sono applicati in molti campi, fra cui la statistica, il machine learning, l'ottimizzazione, la teoria dell'informazione e così via, e hanno applicazioni in molte aree.

Modelli a variabili latenti

I modelli a variabili latenti mirano a rappresentare distribuzioni di dati a elevata dimensionalità utilizzando due distribuzioni più semplici. Il modello coinvolge una variabile latente $\mathbf{z}$ e una variabile osservata $\mathbf{x}$, dove $\mathbf{z}$ è un vettore casuale M -dimensionale e $\mathbf{x}$ è un vettore casuale D -dimensionale. Per definizione, scegliamo $M < D$ per garantire una dimensionalità inferiore per lo spazio latente.

Un modello a variabili latenti è definito dalla distribuzione di probabilità congiunta delle variabili latenti e osservate come $p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$. Il modello è specificato da due componenti: una distribuzione di probabilità sulle variabili latenti, $p(\mathbf{z})$, chiamata distribuzione a priori, e una distribuzione condizionata sui dati osservati date le variabili latenti, $p(\mathbf{x}|\mathbf{z})$, che mappa $\mathbf{z}$ dallo spazio latente allo spazio dei dati. Il campionamento nei modelli a variabili latenti è relativamente semplice, poiché segue il processo di generazione dei dati descritto da:

$$\begin{aligned}\mathbf{z}_t &\sim p(\mathbf{z}_t) \\ \mathbf{x}_t | \mathbf{z}_t &\sim p(\mathbf{x}_t | \mathbf{z}_t) \\ (6.1)\end{aligned}$$

La sfida sta nello scegliere le distribuzioni a priori e condizionate per rappresentare le nostre ipotesi sul processo di generazione dei dati. Queste scelte determinano anche i metodi specifici utilizzati per eseguire l'inferenza sulle variabili latenti e la stima dei parametri.

Come esempi di queste scelte, esamineremo brevemente modelli a variabili latenti tradizionali come l'analisi fattoriale (FA, *Factorial Analysis*), la PPCA (*Probabilistic Principal Component Analysis*) e i GMM (*Gaussian Mixture Models*) prima di passare all'argomento principale di questo capitolo: i modelli profondi a variabili latenti.

Esempi di modelli tradizionali a variabili latenti

Analisi fattoriale

Come abbiamo già visto, l'analisi fattoriale (FA, *Factorial Analysis*), vedi Bishop (2006), è un modello fondamentale che esprime l'ipotesi che i dati possono essere spiegati da un numero ridotto di fattori. Questo approccio ha applicazioni interessanti, come la stima della matrice di covarianza e la riduzione della dimensionalità. Allo stesso modo, anche la PPCA, che esamineremo più avanti, si basa su questo framework probabilistico. In questo paragrafo, approfondiremo la formulazione probabilistica di questi metodi.

Nell'analisi fattoriale, ipotizziamo che le distribuzioni sia marginali sia condizionate siano gaussiane e siano definite come segue:

$$p(\mathbf{z}_t) = \mathcal{N}(\mathbf{z}_t; \mathbf{0}, \mathbf{I})$$
$$p(\mathbf{x}_t | \mathbf{z}_t) = \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu} + \boldsymbol{\Lambda} \mathbf{z}_t, \boldsymbol{\Sigma}_0)$$

(6.2)

dove $p(\mathbf{z}_t)$ è la distribuzione a priori sulla variabile latente, $\mathbf{z}_t$, e $p(\mathbf{x}_t|\mathbf{z}_t)$ rappresenta la probabilità condizionata sulla variabile latente $\mathbf{z}_t$ dati i dati osservati $\mathbf{x}_t$.

In letteratura, la matrice $\Lambda \in \mathbb{R}^{D \times M}$ è chiamata matrice dei carichi (*loading matrix*), e cattura le correlazioni tra le variabili osservate. Il vettore delle medie $\boldsymbol{\mu} \in \mathbb{R}^D$ può essere considerato un termine di bias. La matrice $\boldsymbol{\Sigma}_0$ è diagonale con D elementi e rappresenta la matrice di covarianza del rumore. I valori lungo la diagonale sono le varianze del rumore indipendenti per ogni variabile. In finanza, questa è comunemente chiamata varianza idiosincratca, come descritto in Bodie et al. (2018).

Per ottenere la distribuzione marginale su $\mathbf{x}_t$, integriamo su tutti i possibili valori della variabile latente $\mathbf{z}_t$:

$$p(\mathbf{x}_t) = \int p(\mathbf{x}_t, \mathbf{z}_t) d\mathbf{z}_t = \int p(\mathbf{x}_t | \mathbf{z}_t) p(\mathbf{z}_t) d\mathbf{z}_t$$

Nell'analisi fattoriale, dove sia $p(\mathbf{x}_t)$ sia $p(\mathbf{x}_t|\mathbf{z}_t)$ sono gaussiane, questo integrale ha una soluzione in forma chiusa. Tuttavia, un modo più semplice per calcolare la distribuzione marginale ed evitare il precedente integrale consiste nell'esprimere $\mathbf{x}_t$ come:

$$\mathbf{x}_t = \boldsymbol{\mu} + \Lambda \mathbf{z}_t + \boldsymbol{\epsilon}_t$$

dove $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$.

Usando la proprietà che le gaussiane sono chiuse sotto l'addizione e la moltiplicazione, la distribuzione marginale su $\mathbf{x}_t$ è anch'essa gaussiana. Pertanto, dobbiamo solo calcolare la media e la covarianza di $\mathbf{x}_t$, che sono date da:

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \boldsymbol{\mu} \\ \text{Cov}[\mathbf{x}] &= \boldsymbol{\Sigma} = \Lambda \Lambda^T + \boldsymbol{\Sigma}_0 \end{aligned}$$

Pertanto, la distribuzione marginale su $\mathbf{x}_t$ è:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

dove $\boldsymbol{\Sigma} = \boldsymbol{\Lambda} \boldsymbol{\Lambda}^T + \boldsymbol{\Sigma}_0$, recuperando la formulazione presentata nell'introduzione di questo capitolo.

La stima dei parametri in questo modello segue un approccio ben noto: i parametri $\boldsymbol{\mu}$, $\boldsymbol{\Lambda}$ e $\boldsymbol{\Sigma}_0$ sono stimati dai dati utilizzando l'algoritmo EM (*Expectation-Maximization*). Per maggiori dettagli sulla stima dei parametri per l'analisi fattoriale, vedi Bishop (2006).

Probabilistic Principal Component Analysis

La PPCA (*Probabilistic Principal Component Analysis*) è la versione probabilistica della PCA, proposta da Bishop e Tipping (2001). La PCA è una tecnica ampiamente utilizzata nell'analisi ed elaborazione dei dati, che tuttavia non si basa su un modello probabilistico. In particolare, nella finanza la PCA è impiegata per identificare quei fattori statistici che possono essere utilizzati per la previsione degli asset (Chan, 2017) o per la modellazione del rischio, dove questi fattori hanno mostrato performance comparabili a quelli fondamentali (vedi Paleologo [2021]).

Questo modello è strettamente correlato all'analisi fattoriale. Nella PPCA, la distribuzione, sia a priori, sia condizionata, sono gaussiane e sono definite dalle stesse equazioni dell'Equazione 6.2. Tuttavia, la differenza chiave tra i due metodi è che la PPCA ipotizza che $\boldsymbol{\Sigma}_0$ sia una matrice diagonale costante, denotata come $\boldsymbol{\Sigma}_0 = \sigma_2^2 \mathbf{I}$.

Questa ipotesi fornisce un vantaggio computazionale preservando l'interpretabilità dell'analisi fattoriale.

Esempio: confronto tra PCA e analisi fattoriale per il recupero dello spazio Latente

Come caso d'uso illustrativo per l'analisi fattoriale rispetto alla PCA probabilistica (PPCA), esploriamo un esempio adattato dalla documentazione di sklearn (https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_fa_model_selection.html). Come abbiamo visto, analisi fattoriale e PCA probabilistica sono strettamente correlate: entrambe ipotizzano distribuzioni gaussiane a priori e condizionata. La differenza chiave sta nel modo in cui trattano la *varianza idiosincratICA*.

- PPCA ipotizza la presenza di rumore omoschedastico, ovvero la varianza è la stessa per tutte le variabili (matrice di covarianza identità).
- L'analisi fattoriale ipotizza la presenza di rumore eteroschedastico, dove ogni variabile ha la propria varianza (matrice di covarianza diagonale).

Questo esempio illustra la capacità della PCA (il caso limite della PPCA con varianza zero) e dell'analisi fattoriale di recuperare la corretta dimensione latente in un dataset simulato a rango basso con rumore additivo. Esaminiamo due casi, uno con rumore omoschedastico, con la stessa varianza per ogni variabile (covarianza identità), e un secondo caso con rumore eteroschedastico, con varianza diversa per ogni variabile (covarianza diagonale).

Per l'esperimento, generiamo dati 100-dimensionali con una struttura sottostante a rango basso (dimensione latente = 10).

I risultati riguardanti la selezione del modello per il rumore omoschedastico sono illustrati nella Figura 6.2. In questo caso, i dati a rango basso sono stati corrotti con rumore che ha varianza uguale per tutte le variabili. Sia la PCA (linea continua nera) sia l'analisi

fattoriale (linea tratteggiata grigio scuro) recuperano con successo la corretta dimensione latente di 10, quando eseguono la selezione del modello.

Al contrario, la Figura 6.3 illustra i risultati riguardanti la selezione del modello per rumore eteroschedastico. Qui, i dati a rango basso sono corrotti dal rumore, che ha varianza variabile tra le variabili. Questo porta a un risultato più interessante.

- L'analisi fattoriale (linea tratteggiata in grigio scuro) identifica correttamente la dimensione latente come 10, allineandosi con la sua ipotesi di rumore eteroschedastico.
- La PCA (linea continua nera), al contrario, sovrastima la dimensione latente.

Questi risultati dimostrano come le ipotesi alla base di ciascun metodo influenzino la loro capacità di catturare la vera struttura latente in presenza di diverse caratteristiche del rumore.

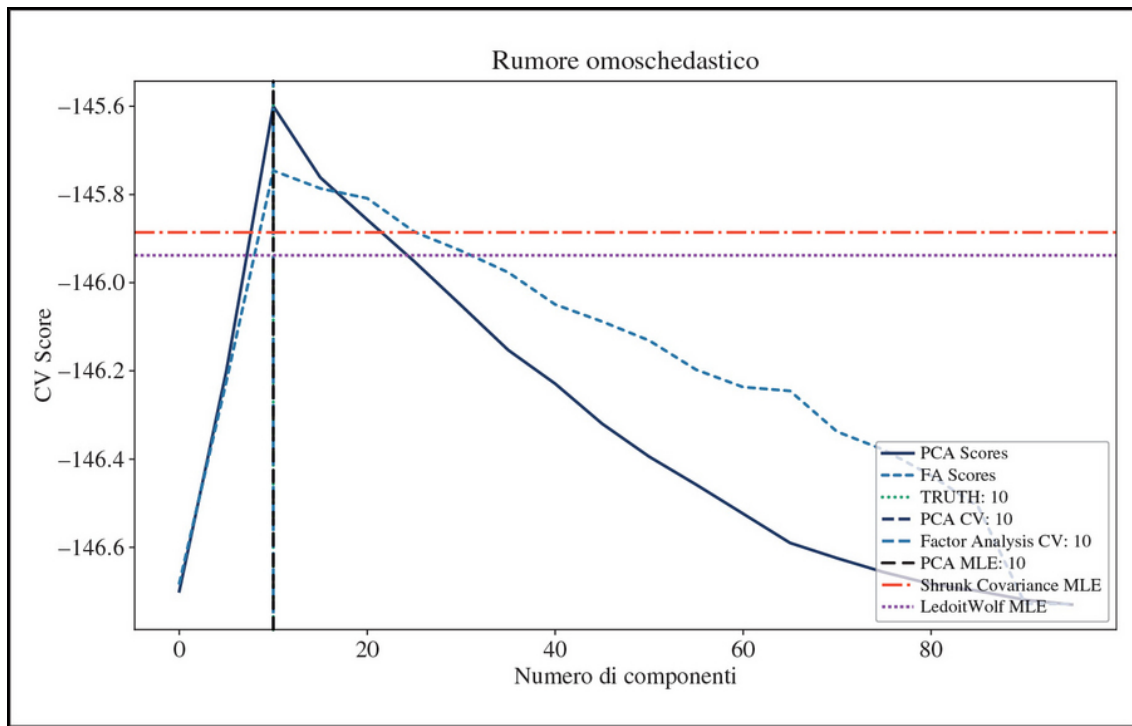


Figura 6.2 Grafico della selezione del modello per rumore omoschedastico.

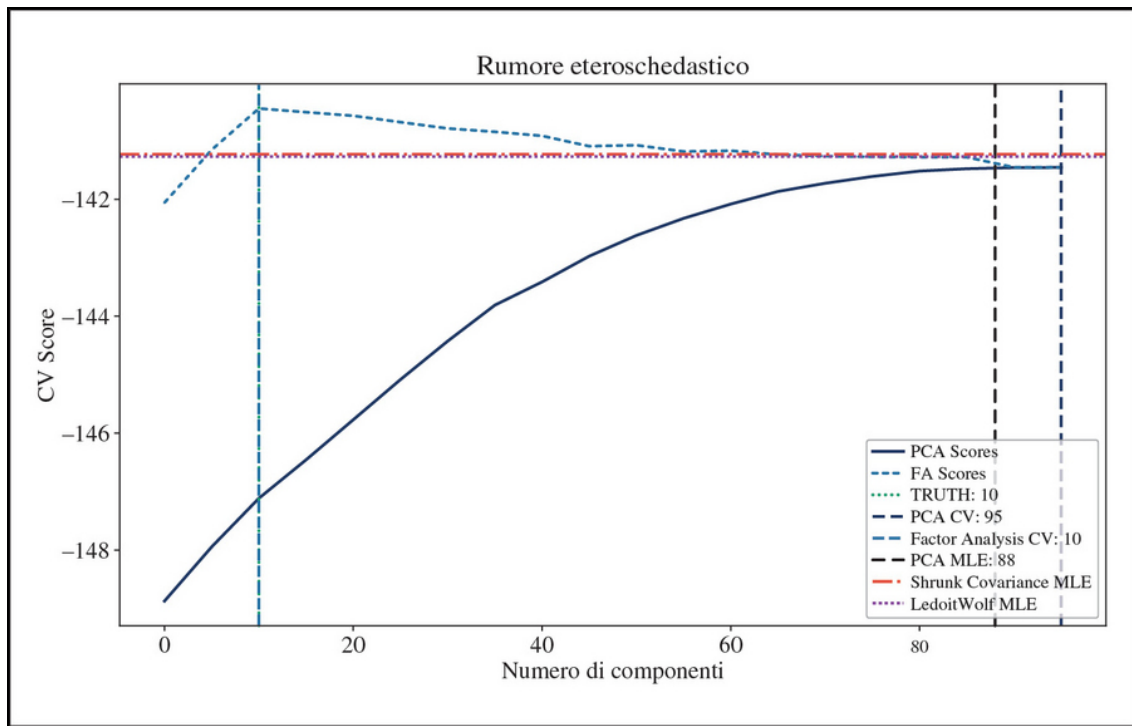


Figura 6.3 Grafico della selezione del modello per rumore eteroschedastico.

Vantaggi degli approcci probabilistici (PPCA/FA) rispetto alla PCA

Ci sono diversi vantaggi nell'usare l'approccio probabilistico, sia PPCA sia FA (analisi fattoriale), rispetto alla PCA (chiamata anche PCA non probabilistica). Alcuni dei principali sono i seguenti.

- *Stima con Dati Mancanti*: PPCA e FA forniscono un framework naturale per la stima dei parametri quando alcune voci di un vettore nel dataset $\mathbf{X}^n \in \mathbb{R}^D$ sono mancanti. Al contrario, con la PCA non probabilistica la stima dei parametri con dati mancanti potrebbe richiedere tecniche di imputazione dei dati.
- *Mix di modelli PPCA*: a volte, una singola proiezione PCA è insufficiente per spiegare i dati. Perché non usare un mix di modelli PCA? Questa idea è analoga ai GMM (*Gaussian Mixture*

Models), trattati nel prossimo paragrafo, dove la distribuzione dei dati viene approssimata usando un mix di gaussiane piuttosto che una singola gaussiana. Allo stesso modo, il framework probabilistico permette che quel mix di modelli PPCA possa essere implementato facilmente, per catturare meglio la struttura dei dati, il che è un vantaggio rispetto all'approccio non probabilistico.

- Generazione di dati e calcolo della verosimiglianza: se il vostro compito coinvolge la generazione di dati o il calcolo delle densità (per esempio, trovare la verosimiglianza di un punto nel dataset), le versioni probabilistiche (PPCA/FA) modellano esplicitamente la distribuzione dei dati.

Per maggiori dettagli su questi vantaggi e sulle procedure di stima per FA, PPCA e PCA utilizzando l'algoritmo EM (*Expectation-Maximization*), fate riferimento all'eccellente articolo di Bishop e Tipping (2001) sulla PPCA.

All'atto pratico, la scelta del metodo appropriato può essere fatta usando la cross-validazione o la conoscenza del dominio. Inoltre, potreste fare le seguenti considerazioni.

- Usare la varianza spiegata per determinare le dimensioni dello spazio latente.
- Decidere tra PPCA e FA in base al fatto che la varianza idiosincratICA (il rumore residuo) di ogni variabile sia uguale (omoschedastica), richiedendo quindi la stima di un singolo σ^2 , o differente (eteroschedastica), richiedendo la stima di D varianze, $\sigma_1^2, \dots, \sigma_d^2$.
- Per l'esplorazione iniziale dei dati o per determinare il modello, potreste usare la PCA, che è computazionalmente più efficiente da stimare rispetto alla sua controparte probabilistica. A seconda delle vostre esigenze, la PCA potrebbe offrire un buon

compromesso tra performance e complessità computazionale, specialmente quando la quantità di dati da elaborare è estremamente grande.

Comprendendo o facendo ipotesi sulla struttura del rumore nei dati possiamo prendere una decisione meglio informata sulle scelte di modellazione.

Gaussian Mixture Models

Con il modello GMM (*Gaussian Mixture Model*), la variabile latente z è discreta, e assume valori nell'insieme $\{1, \dots, K\}$, mentre il valore osservato $\mathbf{x}$ è continuo, e assume valori in $\mathbb{R}^D$. La variabile latente segue una distribuzione categorica, denotata come $p(z) = \text{Categorical}(1, \dots, K; \boldsymbol{\pi})$, dove $\boldsymbol{\pi} = [\pi_1, \dots, \pi_K]$ e, per esempio, la probabilità che la variabile z assuma il valore k è data da $p(z = k) = \pi_k$. La distribuzione condizionata di $\mathbf{x}$ data z è gaussiana, definita da $p(\mathbf{x}|z = k) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Il mix di gaussiane diventa apparente quando si calcola il marginale sulla variabile osservata, marginalizzando la probabilità congiunta tra le variabili latenti e osservate, rispetto alla variabile latente, come segue:

$$\begin{aligned} p(\mathbf{x}) &= \sum_{k=1}^K p(z = k, \mathbf{x}) \\ &= \sum_{k=1}^K p(z = k) p(\mathbf{x} | z = k) \\ &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$

(6.3)

Il risultato è semplicemente una somma pesata di gaussiane (da qui il termine “*Mixture*”). Possiamo creare diversi modelli mix alterando la massa di probabilità sulle variabili latenti.

La distribuzione di probabilità a posteriori delle variabili latenti date le osservazioni quantifica la probabilità che il punto di dati $\mathbf{x}$ sia stato generato dalla k -esima componente del mix. È data dalla distribuzione condizionata:

$$p(z = k | \mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

(6.4)

Per un esempio illustrativo di GMM applicato al clustering, vedi la Figura 6.4. Per maggiori dettagli sui GMM vedi gli eccellenti riferimenti di Bishop (2006) e Murphy (2022).

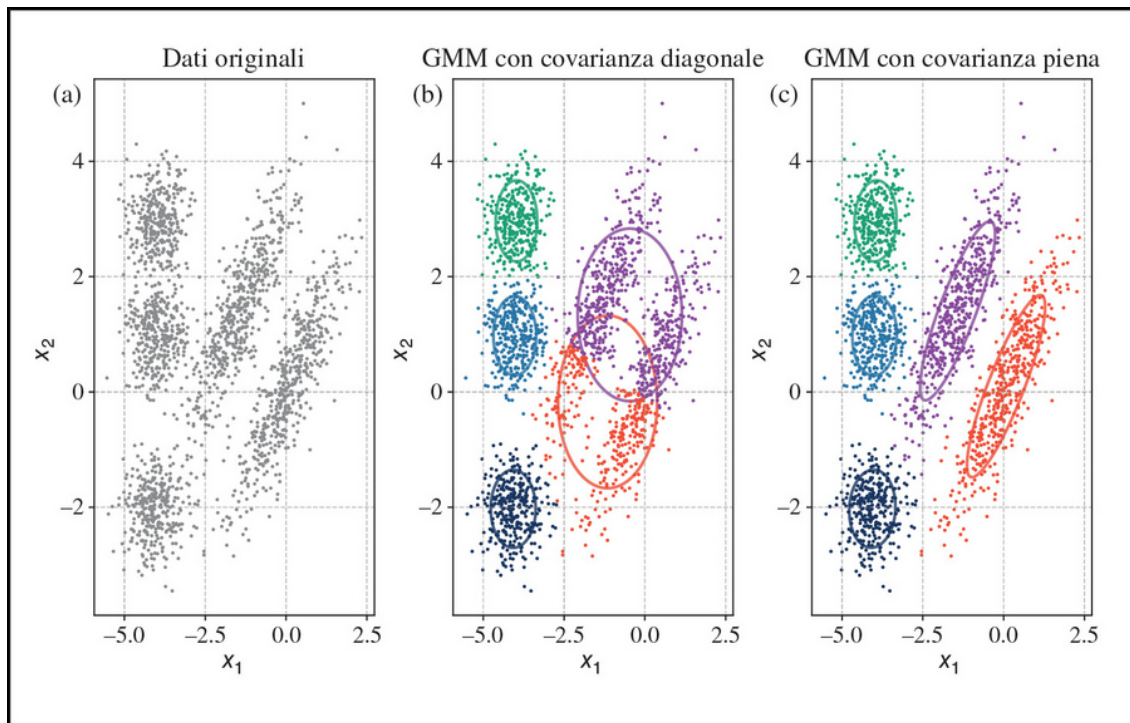


Figura 6.4 Illustrazione del clustering utilizzando modelli GMM (Gaussian Mixture Models).

(a) Campioni di dati 2D generati. (b) Un possibile clustering degli stessi dati con $K = 5$ utilizzando un GMM con una matrice di covarianza diagonale. (c) Un possibile clustering con $K = 5$ utilizzando un GMM con una matrice di covarianza piena. Le assegnazioni dei cluster (indicate dai colori) sono determinate selezionando il cluster con la più alta probabilità a posteriori per ogni punto. Illustrazione basata sulla Figura 3.12 in Murphy (2022), con dettagli aggiuntivi.

Modelli GMM per il rilevamento dei regimi di mercato

Esploriamo come i modelli GMM (*Gaussian Mixture Models*) possono essere applicati a un problema ben noto: il rilevamento dei regimi di mercato. Come sappiamo, i mercati cambiano nel tempo, e trader e investitori spesso usano la conoscenza del dominio per

etichettare certi periodi. Etichette comuni che sentiamo nella stampa, nelle notizie o nelle interviste includono termini come “mercato rialzista” (*bull*), “mercato ribassista” (*bear*), “mercato in trend” o “mercato a ritorno alla media (*mean-reversion*) e così via. In questo caso, tuttavia, per stimare i regimi di mercato adotteremo un approccio guidato dai dati.

Modellare i regimi di mercato è cruciale per trader e investitori, poiché per esempio, potrebbe aiutarli a implementare la strategia più appropriata da un pool di strategie potenziali in base alle condizioni di mercato correnti o incorporare informazioni sul regime nei modelli di costruzione del portafoglio per la gestione del rischio. Come vedremo, il GMM funge da potente elemento costitutivo per sviluppare modelli più sofisticati che possono essere adattati alle vostre specifiche conoscenze del mercato.

Esempio: regimi di bassa e alta volatilità

In questo scenario ipotetico, basato sulla conoscenza del dominio, diciamo che siamo interessati a identificare due cluster distinti che possono corrispondere a due diversi regimi di mercato: uno con bassa e l'altro con alta volatilità. Questi regimi potrebbero essere identificati interamente basandosi sulla conoscenza del dominio o attraverso l'analisi esplorativa dei dati. Per esempio, è improbabile che i rendimenti di un asset come lo SPY possano essere ben spiegati su lunghi periodi utilizzando una singola distribuzione gaussiana con media costante e varianza costante. Considerate periodi recenti della storia, come il periodo prima, durante e dopo la pandemia di COVID-19. Durante la pandemia, la volatilità era eccezionalmente alta, mentre i mesi prima e dopo mostravano una volatilità molto più bassa. Per semplicità, potremmo costruire un modello per catturare separatamente questo periodo di alta volatilità rispetto ai periodi di

bassa volatilità, che potremmo poi visualizzare come “cluster” nella serie temporale dei rendimenti dello SPY. Dal punto di vista applicativo, identificare i regimi di volatilità può aiutare i trader a prendere decisioni come aggiustare le allocazioni del portafoglio, modificare i livelli di leva o implementare strategie di trading specifiche per la volatilità.

Nel seguente esempio, utilizziamo i dati dei rendimenti dell’ETF SPY per il rilevamento dei regimi di mercato. In questa configurazione, la variabile latente z rappresenta il regime di mercato e può assumere due possibili valori: “bassa volatilità” o “alta volatilità”. In base al regime di mercato (allo stato di z), ipotizziamo che i rendimenti seguano una distribuzione gaussiana con medie e varianze specifiche del regime.

In questo caso, il processo di generazione dei dati può essere descritto come segue. Poiché abbiamo solo due stati per la variabile latente, 0 o 1, al tempo t estraiamo il valore dello stato da una distribuzione di Bernoulli con parametro π , dove $P(z = 1) = \pi$:

$$z_t \sim \text{Bernoulli}(\pi)$$

Una volta determinato il valore della variabile latente, al tempo t generiamo dati dalla corrispondente distribuzione gaussiana, con medie e varianze specifiche del regime, come segue:

$$x_t \sim (1 - z_t)\mathcal{N}(\mu_0, \sigma_0) + z_t\mathcal{N}(\mu_1, \sigma_1)$$

Qui, lo stato $z = 0$ corrisponde al primo cluster, il regime di bassa volatilità, mentre $z = 1$ rappresenta il secondo cluster, il regime di alta volatilità. Nel notebook di accompagnamento [*GMM-Market-Regime-Detection*], mostreremo come stimare i parametri di questo modello, cioè $\theta = \{\pi, \mu_0, \mu_1, \sigma_0, \sigma_1\}$, utilizzando dati dall’ETF SPY.

La Figura 6.5 è un grafico di serie temporali dei rendimenti di SPY da gennaio 2000 a marzo 2024, insieme ai regimi di mercato più

probabili stimati dalla distribuzione di probabilità a posteriori (vedi Equazione 6.4) per questo periodo. In questo grafico, i singoli punti rappresentano i rendimenti di SPY, mentre le cornici verticali indicano il regime più probabile in ogni punto temporale. Nella Figura 6.5, abbiamo fasce ben visibili di Regime 1 ad alta volatilità, in mezzo ad ampi spazi di Regime 0 a bassa volatilità. Questo semplice modello identifica ben noti periodi di alta volatilità lungo la linea temporale, come la crisi finanziaria del 2008 e il periodo pandemico del 2020. Vale la pena di notare che in questo esempio, stiamo affrontando il problema del rilevamento o identificazione, non della previsione o predizione del regime di mercato.

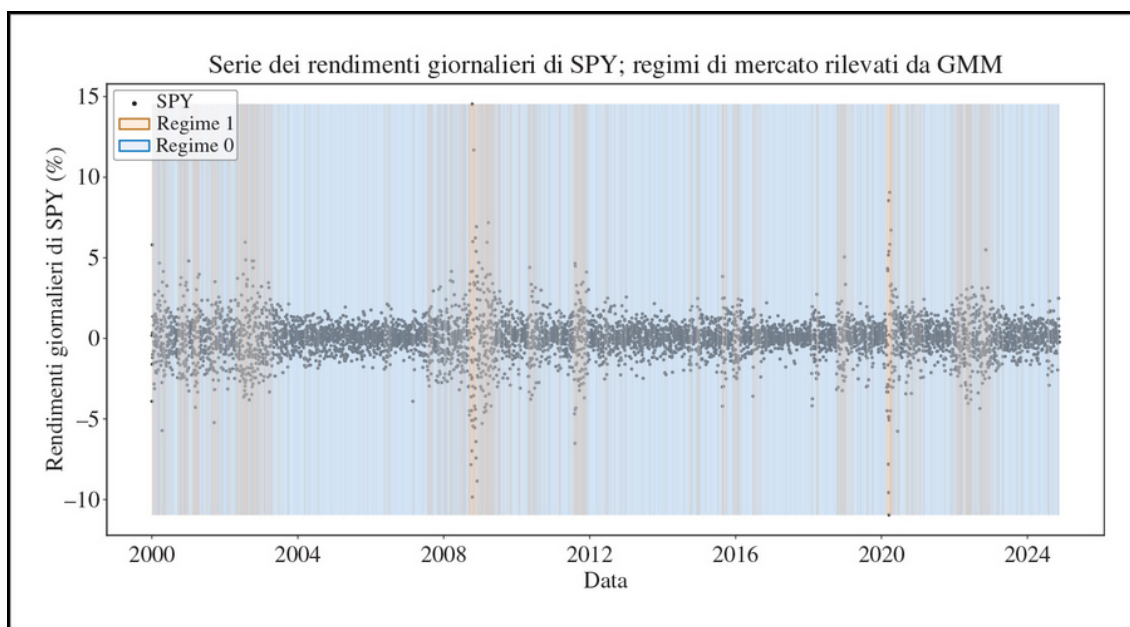


Figura 6.5 Grafico del modello a mix gaussiano per il rilevamento del regime di mercato.

Two Sigma, uno dei più grandi hedge fund al mondo, classificatosi quarto nella *Top 10 US Hedge Funds of December 2024* di Forbes Advisor (Baldrige, 2024) con 67,47 miliardi di dollari di asset in gestione al momento della stesura di questo capitolo, ha condiviso un

approccio simile nel suo post sul blog, *A Machine Learning Approach to Regime Modeling*, vedi Botte e Bao (2021). Hanno utilizzato GMM per rilevare i regimi di mercato e hanno identificato quattro cluster che hanno collegato a diverse condizioni di mercato: “crisi”, “stato stazionario”, “inflazione” e “camminare sul ghiaccio”. Hanno supportato queste etichette con analisi storiche e hanno applicato il metodo ad alcuni fattori personalizzati sviluppati internamente.

Anche se il loro esempio utilizza più cluster e dei dati proprietari, il metodo di fondo è molto simile. Il GMM è un solido punto di partenza per costruire modelli più potenti, che potete personalizzare con le vostre specifiche conoscenze del mercato.

Ci sono molti modi per migliorare l’approccio modellistico per il rilevamento del regime. Per esempio, un limite del setup GMM di base è che ipotizza che le variabili latenti siano indipendenti nel tempo. Questa ipotesi ignora il noto fatto stilizzato del clustering della volatilità, dove periodi di alta volatilità tendono a essere seguiti da altri periodi di alta volatilità, e lo stesso vale per i periodi di bassa volatilità.

Per catturare questo fatto, potete aggiungere delle dinamiche alle variabili latenti. Questo, essenzialmente, trasforma il GMM in un HMM (*Hidden Markov Model*), che è stato ampiamente utilizzato per il rilevamento del regime di mercato (vedi Chan [2017] Kinlaw et al., [2021]). A differenza del GMM, l’HMM aggiunge “memoria”, cosa che gli permette di riflettere meglio l’evoluzione nel tempo delle condizioni di mercato.

Come nota storica interessante, Leonard Baum (https://en.wikipedia.org/wiki/Leonard_E._Baum), uno dei primi dipendenti di Monemetrics (il precursore di Renaissance Technologies, considerato l’hedge fund quantitativo di maggior successo di sempre), ha co-sviluppato l’algoritmo Baum-Welch.

Questo algoritmo, un caso speciale di EM (*Expectation-Maximization*), viene utilizzato per stimare i parametri di un HMM.

Secondo il libro di Gregory Zuckerman (Zuckerman, 2019), *The Man Who Solved the Market*, James Simons, fondatore di Renaissance Technologies, riguardo ai suoi primi giorni in Monometrics disse: “Quando coinvolsi Lenny (Leonard), ho cominciato a vedere le possibilità di costruire modelli”.

Ora, nel resto di questo capitolo, ci concentreremo sui casi in cui i parametri della distribuzione condizionata delle osservazioni date le variabili latenti sono funzioni delle variabili latenti e sono modellati utilizzando reti neurali profonde.

Modelli latenti profondi

In questo libro, ci concentreremo su rappresentazioni specifiche per il modello a variabili latenti discusso finora, cioè:

$$\begin{aligned}\mathbf{z} &\sim p(\mathbf{z}) \\ \mathbf{x}|\mathbf{z} &\sim p(\mathbf{x}|\mathbf{z})\end{aligned}$$

Qui, $p(\mathbf{z})$ rappresenta l’a priori sulla variabile latente $\mathbf{z}$. A differenza degli esempi precedenti, i parametri della probabilità condizionata di $\mathbf{x}$ dato $\mathbf{z}$ sono determinati da una funzione a rete neurale profonda (DNN) di $\mathbf{z}$. Tipicamente, come prima, la relazione è rappresentata come $p(\mathbf{x}|\mathbf{z}) = \text{ExponentialFamily}(\mathbf{x}; \eta)$, ma ora η è definito come $\eta = \text{DNN}_{\theta}(\mathbf{z})$.

Notate che $p(\mathbf{x}|\mathbf{z})$ è chiamata anche *decoder*, un termine che troverete ovunque nella letteratura sul deep learning e sugli LLM. Tali modelli sono indicati come DLVM (*Deep Latent Variable Model*). Quando l’a priori sulla variabile latente è gaussiano, il modello è noto come DLGM (*Deep Latent Gaussian Model*); vedi Murphy (2023).

Contrariamente agli esempi discussi in precedenza, il calcolo dell'a posteriori $p(\mathbf{z}|\mathbf{x})$ è intrattabile. Allo stesso modo, anche la marginale su $\mathbf{x}$ è intrattabile:

$$\begin{aligned} p(\mathbf{x}) &= \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\ &= \int p(\mathbf{z}) p(\mathbf{x} | \mathbf{z}) d\mathbf{z} \\ &= \int p(\mathbf{z}) \text{ExponentialFamily}(\mathbf{x}; \text{DNN}_{\boldsymbol{\theta}}(\mathbf{z})) d\mathbf{z} \end{aligned}$$

Avremo bisogno di approssimare questa quantità per calcolare la densità di $\mathbf{x}$ e, a sua volta, usarla per stimare i parametri del modello. Nel prossimo paragrafo, esploreremo metodi per approssimare questo integrale per addestrare efficacemente i modelli latenti profondi.

Apprendimento

Obiettivo dell'addestramento

Come abbiamo discusso nel capitolo precedente sui modelli autoregressivi, una volta definita una famiglia di distribuzioni di probabilità parametrizzata da $\boldsymbol{\theta}$, denotata da $\{p_{\boldsymbol{\theta}}\}_{\boldsymbol{\theta}}$, il nostro obiettivo è quello di stabilire una funzione obiettivo che ci permetta di selezionare il miglior modello da questa famiglia, che approssimi più accuratamente la distribuzione dei dati, p_{data} .

In modo simile ai modelli autoregressivi, useremo la log-verosimiglianza per definire la nostra funzione obiettivo:

$$\begin{aligned} \mathcal{L} &= \frac{1}{N} \sum_n \log p_{\boldsymbol{\theta}}(\mathbf{x}_n) \\ &= \frac{1}{N} \sum_n \log \int p(\mathbf{x}_n | \mathbf{z}_n) p(\mathbf{z}_n) d\mathbf{z}_n \end{aligned}$$

Tuttavia, come abbiamo osservato, questo calcolo è intrattabile per la maggior parte dei casi interessanti.

L'approssimazione dell'inferenza variazionale

Dato che la verosimiglianza esatta è intrattabile, ne deriveremo un'approssimazione e sceglieremo il miglior modello ottimizzando questa quantità. Le seguenti derivazioni sono state ispirate dallo stile di esposizione presentato in Levine (2021). Concentriamoci su una singola osservazione $\mathbf{x}$

$$\begin{aligned}\log p(\mathbf{x}) &= \log \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\ &= \log \int \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} q(\mathbf{z}) d\mathbf{z} \\ &= \log \mathbb{E}_{\mathbf{z} \sim q} \left[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right]\end{aligned}$$

(6.5)

dove $\mathbb{E}_{\mathbf{z} \sim q}$ indica il valore atteso rispetto alla distribuzione $q(\mathbf{z})$. In questa derivazione, abbiamo introdotto una distribuzione arbitraria $q(\mathbf{z})$ applicando un'identità – moltiplicando e dividendo l'integrando per questa quantità. Questa tecnica ci ha permesso di riscrivere l'integrale come un'aspettativa rispetto a una distribuzione arbitraria $q(\mathbf{z})$. Questa tecnica è chiamata *importance sampling* (letteralmente “campionamento dell'importanza”) e fornisce un framework per approssimare le aspettative rispetto a distribuzioni difficili da campionare ma facili da valutare. Lo fa utilizzando una distribuzione più facile da campionare, come $q(\mathbf{z})$, che è una scelta progettuale (vedi Bishop [2006]). Nella letteratura sull'importance sampling, $q(\mathbf{z})$ è indicata come distribuzione proposta.

Applicando la disuguaglianza di Jensen all'Equazione 6.5, $\log \mathbb{E} [x] \geq \mathbb{E} [\log x]$, possiamo spingere il log all'interno dell'aspettativa,

ottenendo:

$$\begin{aligned}\log p(\mathbf{x}) &= \log \mathbb{E}_{\mathbf{z} \sim q} \left[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \\ &\geq \mathbb{E}_{\mathbf{z} \sim q} \log \left[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \\ &\geq \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z})] \\ &\geq \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z}) + \log p(\mathbf{z}) - \log q(\mathbf{z})]\end{aligned}$$

(6.6)

Qui, l'ultima espressione nell'Equazione 6.6 è chiamata ELBO (*Evidence Lower Bound*), un limite definito come segue:

$$\mathcal{L}(q, p) = \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z}) + \log p(\mathbf{z}) - \log q(\mathbf{z})]$$

(6.7)

L'ELBO può essere scritto o interpretato in due modi (Levine, 2021). Il primo utilizza la formulazione entropica:

$$\begin{aligned}\mathcal{L}(q, p) &= \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z}) + \log p(\mathbf{z})] - \mathbb{E}_{\mathbf{z} \sim q} \log q(\mathbf{z}) \\ &= \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z}) + \log p(\mathbf{z})] + \mathbb{H}[q]\end{aligned}$$

(6.8)

qui, $\mathbb{H}[q]$ rappresenta l'entropia della distribuzione di probabilità q , definita $\mathbb{H}[q] = \mathbb{E}_{\mathbf{z} \sim q} [-\log q(\mathbf{z})]$.

La seconda interpretazione comunemente usata coinvolge la divergenza di Kullback-Leibler:

$$\begin{aligned}
\mathcal{L}(q, p) &= \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z}) + \log p(\mathbf{z})] - \mathbb{E}_{\mathbf{z} \sim q} \log q(\mathbf{z}) \\
&= \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z})] - \mathbb{E}_{\mathbf{z} \sim q} \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z})} \right] \\
&= \mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{x} | \mathbf{z})] - \text{KL}(q(\mathbf{z}) \| p(\mathbf{z}))
\end{aligned}$$

(6.9)

Questa formulazione evidenzia l'interpretazione comune data a questa funzione obiettivo nella letteratura sull'inferenza variazionale: il primo termine rappresenta l'errore di ricostruzione (log-verosimiglianza), e il secondo termine funge da termine di regolarizzazione. Il vantaggio di questo approccio è che la divergenza KL tra $q(\mathbf{z})$ e $p(\mathbf{z})$ spesso ha una soluzione in forma chiusa, specialmente quando entrambe le distribuzioni sono gaussiane – una scelta progettuale comune che semplifica anche il calcolo del gradiente.

Come scegliere la distribuzione proposta

Una decisione critica in questa formulazione consiste nello scegliere la distribuzione proposta, $q(\mathbf{z})$. Potremmo dire che ha senso allineare la distribuzione proposta vicino alla distribuzione a posteriori della variabile latente $\mathbf{z}$ dati i dati osservati $\mathbf{x}$, rappresentata come $p(\mathbf{z} | \mathbf{x})$. Pertanto, possiamo optare per minimizzare la divergenza KL tra entrambe le distribuzioni per avvicinare $q(\mathbf{z})$ a $p(\mathbf{z} | \mathbf{x})$, che implica:

$$\begin{aligned}
\text{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{\mathbf{z} \sim q} \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} \right] \\
&= \mathbb{E}_{\mathbf{z} \sim q} \log q(\mathbf{z}) - \mathbb{E}_{\mathbf{z} \sim q} \log p(\mathbf{z}|\mathbf{x}) \\
&= \mathbb{E}_{\mathbf{z} \sim q} \log q(\mathbf{z}) - \mathbb{E}_{\mathbf{z} \sim q} \log p(\mathbf{x}, \mathbf{z}) + \mathbb{E}_{\mathbf{z} \sim q} \log p(\mathbf{x}) \\
&= \mathbb{E}_{\mathbf{z} \sim q} \log q(\mathbf{z}) - \mathbb{E}_{\mathbf{z} \sim q} \log p(\mathbf{z}) - \mathbb{E}_{\mathbf{z} \sim q} \log p(\mathbf{x}|\mathbf{z}) + \mathbb{E}_{\mathbf{z} \sim q} \log p(\mathbf{x}) \\
&= -\mathbb{E}_{\mathbf{z} \sim q} [\log p(\mathbf{z}) + \log p(\mathbf{x}|\mathbf{z}) - \log q(\mathbf{z})] + \log p(\mathbf{x}) \\
&\quad - \mathcal{L}(q, p) + \log p(\mathbf{x})
\end{aligned}
\tag{6.10}$$

qui, minimizzare $\text{KL}(q(\mathbf{z})||p(\mathbf{x}|\mathbf{z}))$ rispetto a q è equivalente a massimizzare $\mathcal{L}(q, p)$ rispetto a q , poiché $\log p(\mathbf{x})$ è indipendente da $\mathbf{z}$:

$$\text{argmin}_q \text{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) = \text{argmax}_q \mathcal{L}(q, p)
\tag{6.11}$$

Hamlet ricorda la prima volta che ha incontrato questa derivazione e quanto fosse stupito dal suo risultato elegante. Essa ci permette di progettare una distribuzione proposta utilizzando quantità trattabili, anche quando le quantità coinvolte nella derivazione, come la distribuzione a posteriori, sono intrattabili. Nella maggior parte dei casi pratici, in cui le reti neurali profonde vengono utilizzate per parametrizzare le distribuzioni condizionate, è infattibile calcolare direttamente l'a posteriori $p(\mathbf{z}|\mathbf{x})$. Se avessimo accesso diretto a questo a posteriori, potremmo usarlo, invece di fare affidamento su $q(\mathbf{z})$.

È degno di nota il fatto che la derivazione mostra che in realtà non abbiamo bisogno di accedere all'a posteriori $p(\mathbf{z}|\mathbf{x})$ per trovare una distribuzione proposta $q(\mathbf{z})$ che le sia vicina e che minimizzi la divergenza KL tra entrambe le distribuzioni. Sviluppando il problema di ottimizzazione, scopriamo che rendere $q(\mathbf{z})$ il più vicino possibile all'a posteriori è equivalente a massimizzare l'Equazione 6.7,

l'ELBO rispetto a q . Questo è un risultato potente, perché l'ELBO è costituito da quantità che sono trattabili e possono essere calcolate in modo efficiente.

Ciò che ha sorpreso di più Hamlet è il rendersi conto che, ottimizzando lo stesso obiettivo, l'ELBO, rispetto a p , possiamo simultaneamente trovare i migliori parametri per approssimare la distribuzione dei dati. In altre parole, trovare la q ottimale che meglio approssima l'a posteriori e la p ottimale che meglio modella i dati implica alternare tra due step.

1. Massimizzare l'ELBO rispetto a q per rendere $q(\mathbf{z})$ il più vicino possibile all'a posteriori $p(\mathbf{z}|\mathbf{x})$ in termini di divergenza KL.
2. Massimizzare l'ELBO rispetto a p per aumentare la verosimiglianza dei dati osservati $\mathbf{x}$.

Questo approccio di ottimizzazione alternante unifica gli obiettivi dell'approssimazione dell'a posteriori e della massimizzazione della verosimiglianza dei dati sotto la stessa funzione obiettivo.

Ricordate che abbiamo costruito tutte queste equazioni per una singola osservazione fissa $\mathbf{x}$. Una delle problematiche di questa formulazione è che per ogni $\mathbf{x}_N$ nel nostro dataset $\mathcal{D}$, dobbiamo risolvere un problema di inferenza come specificato dall'Equazione 6.11 per determinare la q per lo specifico $\mathbf{x}^n$, denotata come q^n . Per esempio, se rappresentiamo $q_N(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_N, \boldsymbol{\sigma}_N)$, questo implicherà trovare un unico $\boldsymbol{\mu}^n$ e $\boldsymbol{\sigma}^n$ per ogni punto di dati. Di conseguenza, il numero di parametri da stimare è:

$$|\boldsymbol{\theta}| + N(|\boldsymbol{\mu}| + |\boldsymbol{\sigma}|)$$

Qui, la notazione $|\text{valore}|$ rappresenta il numero di parametri. Come indicato dalla formula precedente, il numero di parametri da stimare è lineare rispetto alle dimensioni del dataset N , che nelle attuali applicazioni pratiche può essere enorme.

Inferenza ammortizzata

L'idea di fondo dietro l'inferenza ammortizzata è che, se affrontiamo ripetutamente lo stesso problema di inferenza, una funzione parametrizzata potrebbe risolverlo per noi? Questo ci porta a considerare l'uso di una famiglia di reti neurali per questo compito.

Invece di determinare una distinta q risolvendo il problema di ottimizzazione nell'Equazione 6.11 per ogni singolo $\mathbf{x}$, potremmo pensare di trovare una singola q_ϕ dove i parametri sono determinati da una rete neurale profonda che prende $\mathbf{x}$ come input, e che minimizza la divergenza KL media su tutto il dataset:

$$\min \sum_n \text{KL}(q_\phi(\mathbf{z}_n | \mathbf{x}_n) || p_\theta(\mathbf{z}_n | \mathbf{x}_n)) = \max \sum_n L(q_\phi(\mathbf{z}_n | \mathbf{x}_n), p_\theta(\mathbf{x}_n | \mathbf{z}_n))$$

(6.12)

Questa formulazione ci permette di usare un'altra rete neurale profonda per definire i parametri di q_ϕ e risolvere il problema in modo ammortizzato. Questa rete, che prende un input $\mathbf{x}$ e restituisce i parametri di una distribuzione di probabilità sulla variabile latente $\mathbf{z}$, è comunemente indicata come *encoder* nella letteratura.

Ottimizzazione

Come abbiamo visto in precedenza, per ottimizzare per ϕ e θ , dobbiamo massimizzare lo stesso obiettivo di addestramento, l'ELBO, rispetto a entrambi θ e ϕ . Ora, considerando il calcolo di:

$$\nabla_\phi \mathbb{E}_{\mathbf{z} \sim p_\phi} [p_\theta(\mathbf{x} | \mathbf{z})]$$

(6.13)

affrontiamo la sfida di calcolare i gradienti rispetto a ϕ , che implica prendere la derivata rispetto ai parametri della distribuzione su cui viene preso il valore atteso. Ci sono due metodi comuni per

affrontare questo problema: l'algoritmo REINFORCE (Williams, 1992) e il trucco di riparametrizzazione (Kingma, 2013).

Il gradiente di verosimiglianza, REINFORCE

L'algoritmo REINFORCE può essere derivato come segue:

$$\begin{aligned}
 \nabla_{\phi} \mathbb{E}_{z \sim q_{\phi}}[p_{\theta}(\mathbf{x} | \mathbf{z})] &= \nabla_{\phi} \int q_{\phi}(\mathbf{z} | \mathbf{x}) p_{\theta}(\mathbf{x} | \mathbf{z}) d\mathbf{z} \\
 &= \int \nabla_{\phi} q_{\phi}(\mathbf{z} | \mathbf{x}) p_{\theta}(\mathbf{x} | \mathbf{z}) d\mathbf{z} \\
 &= \int q_{\phi}(\mathbf{z} | \mathbf{x}) \nabla_{\phi} \log q_{\phi}(\mathbf{z} | \mathbf{x}) p_{\theta}(\mathbf{x} | \mathbf{z}) d\mathbf{z} \\
 &= \mathbb{E}_{z \sim p_{\phi}}[\nabla_{\phi} \log q_{\phi}(\mathbf{z} | \mathbf{x}) p_{\theta}(\mathbf{x} | \mathbf{z})]
 \end{aligned}$$

(6.14)

Questo metodo calcola il gradiente di un'aspettativa sfruttando l'identità

$$\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$$

Quindi, il gradiente di un'aspettativa diventa l'aspettativa di un gradiente. REINFORCE utilizza un approccio Monte Carlo per stimare il gradiente dell'aspettativa nell'Equazione 6.14. Un vantaggio di questo metodo è che funziona con variabili latenti i cui spazi dei valori possono essere sia discreti sia continui. Tuttavia, uno degli svantaggi primari di questo approccio è che tende a soffrire di un'elevata varianza. Per maggiori dettagli sull'algoritmo, vedi Williams (1992).

Il trucco di riparametrizzazione

Un approccio alternativo è il trucco di riparametrizzazione (Kingma, 2013), che può essere utilizzato per il calcolo dell'Equazione 6.13.

Per aspettative che coinvolgono il campionamento di $\mathbf{z}$ da $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\Sigma}_\phi(\mathbf{x}))$, possiamo riparametrizzare $\mathbf{z}$ come:

$$\mathbf{z} = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\Sigma}_\phi(\mathbf{x})^{1/2} \boldsymbol{\epsilon} \quad (6.15)$$

dove $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. Con il trucco di riparametrizzazione, tutta la casualità è assorbita da $\boldsymbol{\epsilon}$, e per calcolare il gradiente rispetto a ϕ , abbiamo solo bisogno di calcolare il gradiente delle funzioni deterministiche $\boldsymbol{\mu}_\phi$ e $\boldsymbol{\Sigma}_\phi$.

$$\nabla_\phi \mathbb{E}_{\mathbf{z} \sim q_\phi} [p_\theta(\mathbf{x}|\mathbf{z})] = \nabla_\phi \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})} [p_\theta(\mathbf{x}|\boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\Sigma}_\phi(\mathbf{x})^{1/2} \boldsymbol{\epsilon})]$$

Questo gradiente può essere calcolato in modo efficiente utilizzando strumenti di differenziazione automatica come PyTorch o TensorFlow.

Attenzione al gap!

Un altro modo per scrivere l'Equazione 6.10 è:

$$\log p(\mathbf{x}) = \mathcal{L}(q, p) + \text{KL}(q(\mathbf{z})||p(\mathbf{x}|\mathbf{z})) \quad (6.16)$$

In questa formulazione, il termine $\text{KL}(q(\mathbf{z})||p(\mathbf{x}|\mathbf{z}))$ è noto come gap o errore che sorge quando approssimiamo la log-verosimiglianza con l'ELBO. Se effettivamente abbiniamo $q(\mathbf{z})$ a $p(\mathbf{x}|\mathbf{z})$, la KL sarà zero e la log-verosimiglianza uguale all'ELBO. Di conseguenza, l'errore nella nostra approssimazione è quantificato dalla divergenza KL tra la nostra approssimazione q_ϕ e $p(\mathbf{z}|\mathbf{x})$.

Autoencoder variazionali

Quello ad autoencoder variazionale (VAE, *Variational Autoencoder*) è un modello a variabile latente in cui la distribuzione marginale sulla variabile latente è gaussiana, e sia l'encoder che il

decoder (di cui si parlerà a breve) sono modellati come gaussiane condizionate, come descritto da Kingma (2013). Le osservazioni possono essere immagini, serie temporali multivariate e così via.

L'encoder prende in input $\mathbf{x}$ e restituisce i parametri di $q_{\phi}(\mathbf{z}|\mathbf{x})$, specificamente il vettore della media e la matrice di covarianza. Analogamente, il decoder prende come input $\mathbf{z}$ e restituisce il vettore della media e la matrice di covarianza per $p_{\theta}(\mathbf{x}|\mathbf{z})$. Il VAE classico è tipicamente addestrato utilizzando il trucco della riparametrizzazione. Una rappresentazione schematica di queste operazioni è illustrata nella Figura 6.6.

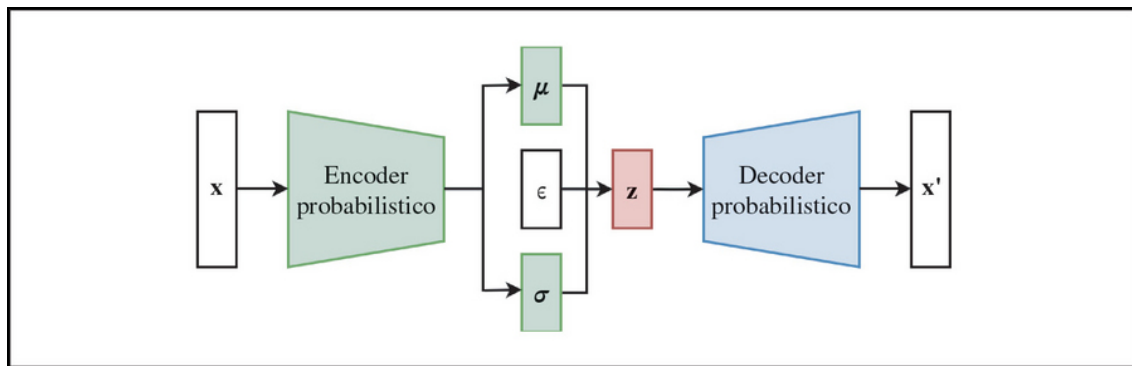


Figura 6.6 Rappresentazione grafica del funzionamento di un VAE. Fonte: EugenioTL,

https://en.m.wikipedia.org/wiki/File:Reparameterized_Variational_Autoencoder.png, ultimo accesso 13 gennaio 2025 / CC BY 30.

Più formalmente, il VAE classico (Kingma, 2013) è definito dalle seguenti rappresentazioni.

- L'a priori su $\mathbf{z}$, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.
- L'encoder, $q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_{\phi}(\mathbf{x}), \boldsymbol{\Sigma}_{\phi}(\mathbf{x}))$.
- Il decoder $p_{\theta}(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_{\theta}(\mathbf{z}), \boldsymbol{\Sigma}_{\theta}(\mathbf{z}))$.

I VAE vengono tipicamente addestrati utilizzando il trucco della riparametrizzazione. Durante l'addestramento, come obiettivo di

addestramento viene utilizzato l'ELBO:

$$\mathcal{L}(q, p) = \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} [\log p_{\theta}(\mathbf{x} | \mathbf{z})] - \text{KL}(q_{\phi}(\mathbf{z} | \mathbf{x}) \parallel p(\mathbf{z}))$$

con $\mathbf{z} = \boldsymbol{\mu}_{\phi}(\mathbf{x}) + \boldsymbol{\Sigma}_{\phi}(\mathbf{x})^{1/2} \boldsymbol{\epsilon}$. Per ottimizzare i parametri ϕ e θ , vengono calcolati i gradienti $\nabla_{\theta} L(q, p)$ e $\nabla_{\phi} L(q, p)$, tipicamente utilizzando gradienti stocastici. I parametri ϕ e θ possono quindi essere aggiornati utilizzando la discesa del gradiente.

Dopo l'addestramento del VAE, è possibile generare nuovi punti campionando un valore dall'a priori e alimentando questo valore nel decoder per generare un nuovo campione:

$$\begin{aligned} \mathbf{z} &\sim p(\mathbf{z}) \\ \mathbf{x} | \mathbf{z} &\sim p_{\theta}(\mathbf{x} | \mathbf{z}) \end{aligned}$$

Nella Figura 6.7, vediamo rappresentazioni tratte dall'articolo di Kingma (2013) dello spazio latente appreso e dello spazio dei dati corrispondente, generati da un VAE dopo l'addestramento sul dataset Frey Faces.

Un aspetto notevole è che, poiché l'a priori dello spazio latente è costituito da variabili gaussiane indipendenti (covarianza identità), l'utilizzo di uno spazio latente bidimensionale illustra il disaccoppiamento nello spazio dei dati. Variare una coordinata può generare volti sorridenti, mentre variare l'altra può alterare la posizione della testa, tutto in modo indipendente, come mostra la Figura 6.7.



Figura 6.7 Illustrazione della varietà di dati appresa per modelli generativi con uno spazio latente gaussiano bidimensionale indipendente. La variazione di una coordinata genera volti sorridenti, mentre la variazione dell'altra altera la posizione della testa, entrambe in modo indipendente. Il modello è stato addestrato sul dataset Frey Faces. Fonte: Kingma (2013).

Il VAE può anche servire come strumento di apprendimento delle rappresentazioni per attività di machine learning a valle. Per esempio,

se $\mathbf{x}$ rappresenta un'immagine, la rappresentazione compressa $\mathbf{z}$ potrebbe essere utilizzata come input per un classificatore di immagini. In effetti, questo approccio di apprendimento delle rappresentazioni viene utilizzato in *Stable Diffusion*, una delle tecniche più potenti per la generazione di immagini attualmente disponibili.

Nel notebook associato a questo capitolo ([*VAE Notebook*]), implementiamo e proviamo il tradizionale modello VAE introdotto da Kingma (2013). Sebbene questo esempio utilizzi dati di immagini, funge da passo importante per costruire una solida comprensione dei VAE, che estenderemo per catturare la dinamica delle serie temporali.

VAE per dati sequenziali e serie temporali

Come abbiamo visto nel paragrafo precedente, basato sul lavoro di Kingma (2013) e illustrato nella Figura 6.7, i VAE sono molto efficaci nel modellare le correlazioni all'interno di un'immagine $\mathbf{x}_N$ in un dataset, dove $\mathbf{x}^n$ è un vettore D -dimensionale di intensità dei pixel. Per applicare i VAE standard ai dati delle serie temporali, potremmo semplicemente reinterpretare il vettore $\mathbf{x}_N$ come una serie temporale multivariata con D serie al tempo n e utilizzare i VAE per campionare, per esempio, serie temporali cross-sectional.

Tuttavia, nel framework standard dei modelli a variabili latenti, il processo di generazione dei dati ipotizza che campioniamo $\mathbf{z}^n$ da $p(\mathbf{z})$ e poi generiamo $\mathbf{x}^n$ dalla distribuzione condizionata $p(\mathbf{x}|\mathbf{z}^n)$. Per definizione, questa struttura presuppone l'indipendenza tra osservazioni consecutive. Questa ipotesi di indipendenza può limitare significativamente le prestazioni dei VAE in scenari in cui le dipendenze temporali tra le osservazioni sono significative.

Ci sono diversi modi interessanti per estendere il VAE standard per catturare le dipendenze temporali tra le osservazioni. Questo è particolarmente importante per modellare serie temporali a determinate scale temporali, dove i prezzi/rendimenti passati, le condizioni di mercato e altri fattori influenzano significativamente i prezzi/rendimenti futuri. Queste estensioni possono trarre ispirazione dai modelli tradizionali di serie temporali, come l'incorporazione di strutture autoregressive nelle osservazioni o nelle variabili latenti, simili ai modelli autoregressivi classici o ai DLM (*Dynamic Linear Models*), vedi Prado e West (2010). Ecco alcune alternative.

Estensioni dei VAE per le serie temporali

Encoder e decoder sequenziali

Un'estensione semplice consiste nel sostituire l'encoder e il decoder VAE standard utilizzati in Kingma (2013) con modelli sequenziali, come le reti neurali ricorrenti o i transformer. Un esempio di questo approccio è il VRAE (*Variational Recurrent Autoencoder*) di Fabius e Van Amersfoort (2014). Di seguito una breve descrizione dell'idea di fondo.

- Il VRAE utilizza lo stato nascosto finale della rete neurale ricorrente encoder come rappresentazione o riepilogo della sequenza di input. Questo stato nascosto viene quindi utilizzato per determinare i parametri della distribuzione di probabilità della variabile latente $\mathbf{z}$. In altre parole, la distribuzione su $\mathbf{z}$ è determinata dall'ultimo stato della rete neurale ricorrente.
- La variabile latente $\mathbf{z}$ viene quindi campionata utilizzando il trucco della riparametrizzazione.

- Infine, la variabile latente $\mathbf{z}$ campionata viene passata nella rete neurale ricorrente decoder, che ricostruisce la sequenza di input.

Sovrapposizione di componenti delle serie temporali

Un altro approccio consiste nel modellare le serie temporali come una sovrapposizione di componenti, un'idea comunemente usata nei DLM (*Dynamic Linear Models*), vedi Prado e West (2010). In questo framework, la serie temporale osservata potrebbe essere composta da una combinazione dei seguenti elementi:

- livello locale o intercettazione (per esempio, un percorso casuale);
- componenti di trend a lungo termine;
- modelli stagionali (per esempio, periodicità settimanali, mensili o annuali);
- processi autocorrelati a breve termine.

Qui, ogni componente della serie temporale può avere il proprio vettore di stato (variabili latenti), che viene poi proiettato linearmente nello spazio dei dati.

Questo approccio fornisce sia interpretabilità sia la flessibilità necessaria per incorporare la conoscenza del dominio, affrontare i limiti relativi ai dati e adattarsi alle esigenze specifiche del problema in questione. Utilizzando singoli componenti delle serie temporali, si possono progettare soluzioni (modelli) su misura per le proprie esigenze specifiche. Nel prossimo paragrafo, descriveremo una soluzione basata su questo principio.

Per saperne di più sui modelli DLM, consultate l'introduzione di *Bayesian Structural Time-series Models* (Medina Ruiz, 2019), che esplora la modellazione delle serie temporali per prevedere il carico del traffico Internet di Criteo nel contesto della pianificazione della

capacità dell'infrastruttura server. Questo esempio evidenzia come l'iniezione di conoscenza del dominio nel processo di modellazione possa affrontare i limiti dei dati e migliorare le prestazioni complessive.

TimeVAE: un VAE Flessibile per la generazione di serie temporali

Un esempio più recente di estensione dei VAE alle serie temporali è TimeVAE, introdotto da Desai et al. (2021). TimeVAE adatta i VAE alla generazione di serie temporali multivariate, combinando layer di deep learning tradizionali con componenti specifici per le serie temporali.

Utilizzando decoder specializzati, TimeVAE consente l'inclusione di strutture temporali, come trend e componenti stagionali, direttamente nel processo di modellazione. Questo approccio combina la flessibilità e l'interpretabilità dei metodi tradizionali come i DLM (*Dynamic Linear Models*), ma con il vantaggio di catturare le relazioni non lineari presenti nei dati a elevata dimensionalità. TimeVAE utilizza sia layer di apprendimento tradizionali (per esempio, layer densi e convoluzionali) sia layer personalizzati specifici per le serie temporali, per catturare componenti come livelli, trend polinomiali multipli e stagionalità.

L'architettura TimeVAE è divisa in due parti principali: *Base TimeVAE*, che non incorpora la conoscenza del dominio per modellare la serie temporale, e *Interpretable TimeVAE*, che consente l'iniezione di conoscenza del dominio ed è personalizzata per modellare specifici componenti delle serie temporali. Di seguito una descrizione dell'architettura.

Architettura di TimeVAE

1. *Encoder e decoder – TimeVAE base.*

- L'encoder elabora la serie temporale di input attraverso una serie di layer convoluzionali con attivazione ReLU.
- L'output dell'encoder viene appiattito e passato attraverso un layer lineare pienamente connesso (denso) per determinare i parametri della distribuzione sulla variabile latente $\mathbf{z}$, che è modellata come una gaussiana multivariata.
- Un'istanza della variabile latente $\mathbf{z}$ viene campionata utilizzando il trucco della riparametrizzazione, e poi viene inviata al decoder.
- Il decoder tenta di invertire il processo di encoding. È composto da un layer lineare pienamente connesso, seguito da un reshaping e da una serie di layer convoluzionali trasposti con attivazione ReLU. Infine, i dati attraversano un layer pienamente connesso time-distributed per produrre la serie temporale ricostruita.

Una rappresentazione schematica del TimeVAE base si trova nella Figura 6.8.

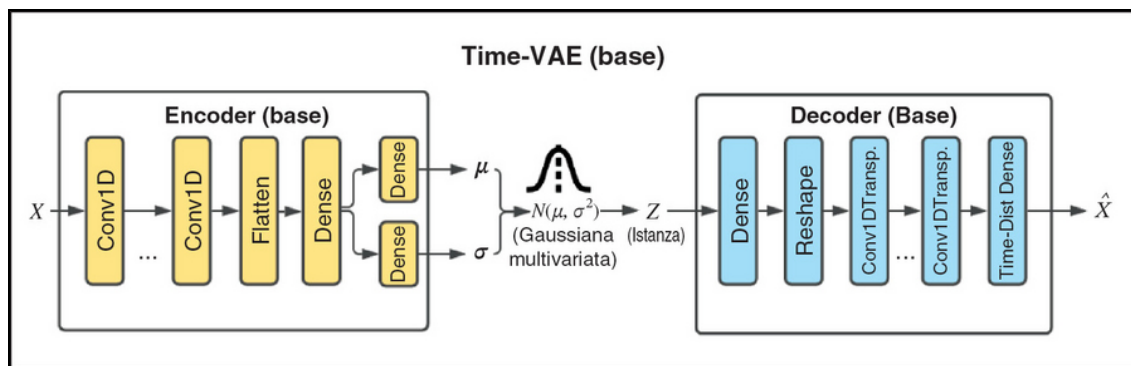


Figura 6.8 Rappresentazione grafica dell'architettura encoder-decoder del TimeVAE base. L'input X viene elaborato attraverso una serie di operazioni, inclusi layer convoluzionali, reshaping e un layer denso, per determinare i parametri della distribuzione della variabile latente Z . Il decoder prende

un'istanza di Z e tenta di ricostruire l'input invertendo il processo di encoding. Figura 1 di Desai et al. (2021).

2. *Interpretable TimeVAE*: componenti specifici per le serie temporali.

Come accennato, TimeVAE introduce decoder specializzati per modellare i singoli componenti di una serie temporale, come i trend e le stagionalità.

Ogni decoder specializzato prende in input la variabile latente z (che può essere prodotta dall'encoder del TimeVAE base). Gli output di questi decoder vengono combinati in parallelo per addizione, per produrre la ricostruzione finale della serie temporale. Una rappresentazione schematica di *Interpretable TimeVAE* si trovata nella Figura 6.9.

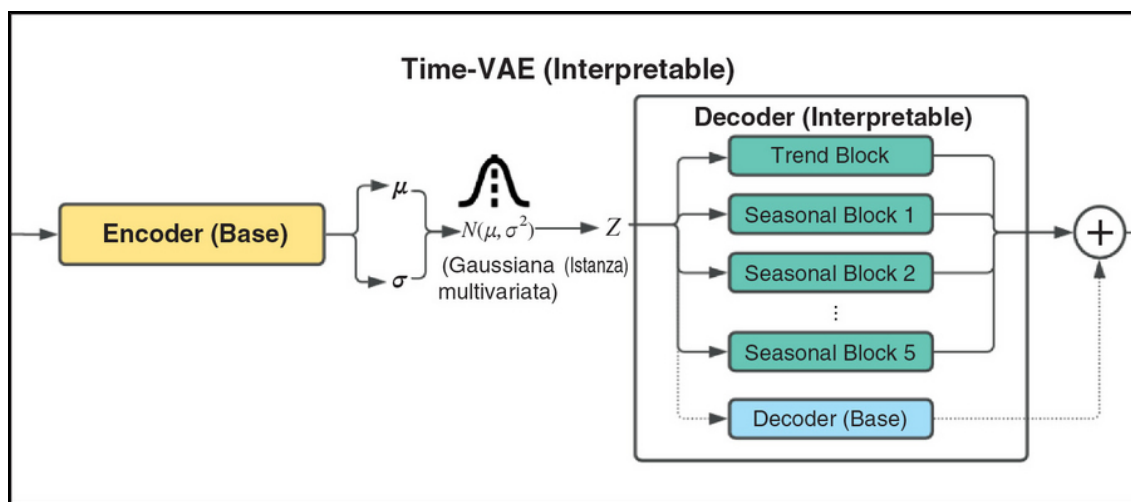


Figura 6.9 Rappresentazione grafica dei componenti principali di Interpretable TimeVAE. Decoder specializzati, come Trend Block, Seasonal Block 1 e Seasonal Block 2, prendono come input la variabile latente Z . Gli output di questi decoder vengono combinati in parallelo tramite addizione per produrre la ricostruzione finale della serie temporale. Seasonal Block 1 e Seasonal Block 2 possono rappresentare diversi componenti stagionali, catturando

anche stagionalità multiple presenti nella serie temporale. Figura 2 di Desai et al. (2021).

TimeVAE offre i seguenti vantaggi.

- *Flessibilità*: è possibile modellare facilmente più componenti temporali, come trend polinomiali di diversi gradi e diverse stagionalità (per esempio, settimanali, mensili, annuali), un aspetto che lo rende altamente adattabile per catturare diverse caratteristiche delle serie temporali.
- *Interpretabilità*: modellando esplicitamente componenti come i trend e le stagionalità, TimeVAE aiuta a comprendere la struttura sottostante della serie temporale.
- *Modellazione non lineare*: a differenza dei modelli tradizionali come DLM, TimeVAE può catturare relazioni non lineari complesse in dati a elevata dimensionalità.

Nel notebook associato a questo capitolo ([*TimeVAE Notebook*]), implementiamo e proviamo TimeVAE su alcuni dati finanziari. Crediamo che il modo migliore per apprendere sia attraverso la pratica, e questo notebook vi dà l'opportunità di esplorare e sperimentare le specifiche architetture decoder per modellare i trend e le stagionalità. Potete anche estendere queste architetture per adattare alle vostre specifiche esigenze.

Se non avete ancora familiarità con l'implementazione standard del VAE, vi consigliamo di iniziare con il notebook VAE, dove le architetture dell'encoder e del decoder sono più facili da seguire. Questo fornirà una solida base prima di immergersi in TimeVAE.

Conclusioni

In questo capitolo, abbiamo esplorato come i VAE forniscano un framework potente per modellare dati complessi, multimodali e a elevata dimensionalità.

Abbiamo visto come i VAE siano altamente efficienti nel campionamento e flessibili per modellare dati a elevata dimensionalità utilizzando una rappresentazione più piccola dello spazio latente. Questo spazio latente fornisce rappresentazioni significative dei dati, che possono essere utilizzate per applicazioni come la compressione, la generazione di dati e altre attività a valle. Tuttavia, un compromesso per questa flessibilità è che il calcolo della verosimiglianza diventa intrattabile per la maggior parte dei casi pratici che coinvolgono modelli profondi. Per affrontare questo problema, ci affidiamo ad approssimazioni della verosimiglianza, che rendono possibili la stima dei parametri e attività come la selezione del modello.

Abbiamo anche discusso come i VAE possano essere estesi per gestire dati sequenziali, in particolare per la dinamica delle serie temporali. Per esempio, abbiamo introdotto approcci come il VRAE e TimeVAE, che combinano il framework VAE con architetture sequenziali come le reti neurali ricorrenti e componenti specifici per le serie temporali. Questi modelli offrono un punto di partenza pratico per costruire soluzioni su misura per applicazioni per la previsione di dati finanziari, per la generazione di dati finanziari o per altri problemi o dati sequenziali. La rappresentazione latente appresa dai VAE è altamente versatile e può essere utilizzata per addestrare modelli a valle per risolvere un'ampia varietà di applicazioni. Abbiamo visto l'uso dei VAE nella generazione di serie temporali, ma possono essere utilizzati anche per attività come la classificazione, il rilevamento di anomalie e la previsione.

Nei prossimi capitoli, continueremo a espandere la nostra “cassetta degli attrezzi” di modelli generativi profondi per dati a elevata dimensionalità, modelli che originariamente hanno avuto successo nell’elaborazione del linguaggio naturale e nella computer vision. Ancora più importante, per i nostri casi d’uso nel trading e nella gestione degli asset, esploreremo tecniche per mostrare come questi modelli possano essere ulteriormente estesi e adattati per applicazioni a serie temporali e dati sequenziali.

Capitolo 7

Modelli di flusso

Nel capitolo precedente, abbiamo esplorato come i modelli profondi a variabili latenti (DLVM, *Deep Latent Variable Models*) modellino distribuzioni di probabilità complesse come l'integrale del prodotto di due distribuzioni più semplici. Questo approccio, sebbene potente, introduce diverse complessità, come l'intrattabilità della probabilità marginale delle osservazioni. Per affrontare ciò, spesso ricorriamo ad approssimazioni come l'ELBO (*Evidence Lower Bound*).

In questo capitolo, esamineremo un approccio più semplice, ma efficace, per modellare distribuzioni complesse: i modelli di flusso (*flow models*). Sono anch'essi modelli a variabili latenti, ma aggirano molte delle problematiche associate ai DLVM descritti nel capitolo precedente. Essi ci permettono di calcolare le probabilità marginali (cioè $p(\mathbf{x})$) in modo esatto, di campionarle in modo efficiente e di utilizzarle per l'apprendimento di rappresentazioni, senza la necessità di ricorrere a complesse approssimazioni della verosimiglianza. I modelli di flusso rientrano nella categoria dei modelli a densità esplicita trattabile, come illustrato nella Figura 7.1.

Introduzione

I modelli a variabili latenti (LVM, *Latent Variable Models*) sono caratterizzati da una distribuzione di probabilità sulla variabile

latente $\mathbf{z}$, denotata da $p(\mathbf{z})$, e da una distribuzione della probabilità condizionata sull'osservazione $\mathbf{x}$ data la variabile latente $\mathbf{z}$, denotata da $p_{\theta}(\mathbf{x}|\mathbf{z})$.

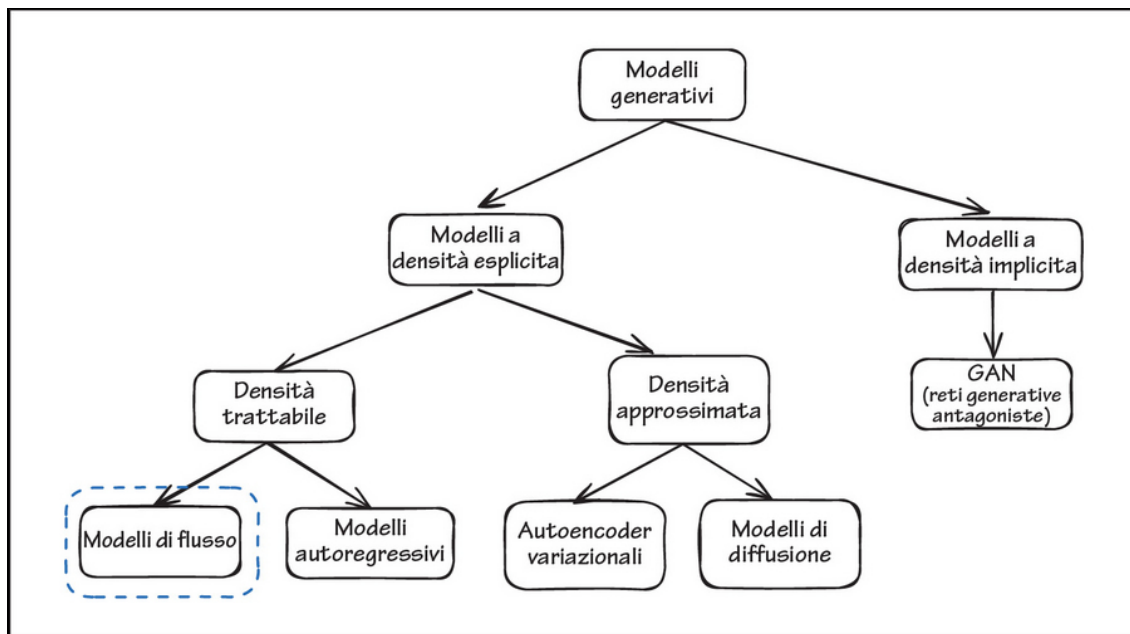


Figura 7.1 Grafico ad albero della tassonomia dei modelli: i modelli di flusso.

Nei modelli profondi a variabili latenti, i parametri di questa probabilità condizionata sono determinati da una rete neurale profonda (DNN, *Deep Neural Network*) chiamata *decoder*. Il decoder prende come input $\mathbf{z}$ e restituisce in output il parametro della distribuzione di probabilità su $\mathbf{x}$. Per esempio, nel caso dell'autoencoder variazionali (VAE, *Variational Autoencoder*), questi parametri sono il vettore della media e la matrice di covarianza di una distribuzione gaussiana su $\mathbf{x}$. In questo contesto, il valore di $\mathbf{z}$ influenza la densità di probabilità sugli stati possibili di $\mathbf{x}$, ma non specifica deterministicamente il suo stato. Pertanto, la mappatura da $\mathbf{z}$ a $\mathbf{x}$ fornita dal decoder è probabilistica.

Nei modelli di flusso, sostituiamo il decoder probabilistico $p_{\theta}(\mathbf{x}|\mathbf{z})$ con una mappatura deterministica dalla variabile latente $\mathbf{z}$

all'osservazione $\mathbf{x}$, denotata da $\mathbf{x} = f(\mathbf{z})$. Nella terminologia dei modelli di flusso, $p_z(\mathbf{z})$ è chiamata misura base o distribuzione base, e f è chiamata flusso, *flow*.

In questi modelli, calcolare la marginale o la densità di $\mathbf{x}$ implica risolvere un problema di *distribuzione derivata*, dove è data la misura base $p_z(\mathbf{z})$, e puntiamo a trovare la distribuzione di probabilità di $\mathbf{x}$ in funzione della misura base. La distribuzione di $\mathbf{x}$ è data dalla formula del cambio di variabili, che ci permette di esprimere la densità di probabilità $p(\mathbf{x})$ in funzione di $p(\mathbf{z})$, che conosciamo.

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z}) \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1}$$

(7.1)

con $\mathbf{z} = f^{-1}(\mathbf{x})$ e

$$\mathbf{J} = \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}$$

rappresenta la matrice jacobiana.

Nei modelli di flusso, tipicamente la misura base è una distribuzione semplice, come una distribuzione normale o uniforme. Quando la misura base è una distribuzione normale, il modello di flusso è noto come *flusso di normalizzazione*. L'obiettivo è quello di progettare il flusso f in modo che sia una funzione altamente complessa capace di mappare una misura base semplice su una distribuzione di dati complessa e multimodale.

Rispetto ai modelli profondi a variabili latenti discussi nel capitolo precedente, i flussi di normalizzazione permettono di calcolare esattamente le marginali senza la necessità di approssimazioni o di imporre limiti inferiori. Nei modelli di flusso, anche il campionamento è più semplice e può essere effettuato come segue:

$$\begin{aligned}\mathbf{z} &\sim p(\mathbf{z}) \\ \mathbf{x} &= f(\mathbf{z}) \\ (7.2)\end{aligned}$$

Pertanto, campioniamo $\mathbf{z}$ dalla misura base e applichiamo la trasformazione di flusso per ottenere l'osservazione $\mathbf{x}$.

La formula del cambio di variabili impone alcuni vincoli sulla famiglia di flussi f che possiamo usare. Nello specifico, affinché questa formula funzioni, f deve essere invertibile e differenziabile. Di conseguenza, la dimensionalità delle osservazioni $\mathbf{x}$ deve essere uguale alla dimensionalità della variabile latente $\mathbf{z}$.

Per soddisfare questi requisiti, dobbiamo progettare architetture speciali di reti neurali che siano invertibili e differenziabili. Questo, spesso, implica la creazione di modelli profondi innovativi e complessi che soddisfino questi vincoli.

Addestramento del modello

Addestriamo i modelli di flusso utilizzando la stima di massima verosimiglianza (*maximum likelihood estimation*). La log-verosimiglianza è data da:

$$\begin{aligned}\mathcal{L} &= \sum_n \log p_{\mathbf{x}}(\mathbf{x}_n) \\ &= \sum_n \log \left[p_{\mathbf{z}}(\mathbf{z}_n) \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1} \right] \\ &= \sum_n \log p_{\mathbf{z}}(\mathbf{z}_n) - \log \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right| \\ (7.3)\end{aligned}$$

dove $\mathbf{z}_N = f^{-1}(\mathbf{x}_N)$. Il problema consiste nel progettare architetture a rete neurale per modellare un flusso f che sia invertibile e abbia un

determinante della jacobiano che possa essere calcolato in modo efficiente. Questo è cruciale per garantire che la log-verosimiglianza possa essere calcolata rapidamente per, potenzialmente, i milioni di punti di dati del dataset. Nei prossimi paragrafi, dettaglieremo ed elaboreremo alcune architetture che soddisfano questi vincoli.

Flussi lineari

Torniamo al modello fattoriale lineare del capitolo scorso, dove l'a priori su $\mathbf{z}$ e il modello condizionato di $\mathbf{x}|\mathbf{z}$ sono dati da:

$$\begin{aligned} p_{\mathbf{z}}(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) \\ p_{\mathbf{x}|\mathbf{z}}(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\mathbf{x}; \boldsymbol{\mu} + \Lambda\mathbf{z}, \boldsymbol{\Sigma}_0) \end{aligned} \quad (7.4)$$

Per convertire questo modello fattoriale lineare in un modello di flusso, un approccio semplice consiste nel sostituire la mappatura probabilistica o il decoder probabilistico da $\mathbf{z}$ a $\mathbf{x}$, definito dalla distribuzione condizionata, con una mappatura deterministica o flusso, rappresentata da f . Se ipotizziamo la stessa misura base $p_{\mathbf{z}}(\mathbf{z})$ (a priori) e il flusso f come $f(\mathbf{z}) = \boldsymbol{\mu} + \Lambda \mathbf{z}$, otteniamo i “flussi lineari normalizzanti”, *Linear Normalizing Flows*, vedi Kobyzev et al. (2021). Questo può essere interpretato come una trasformazione di scala e posizione della variabile $\mathbf{z}$. In questo contesto, Λ deve essere una matrice invertibile $D \times D$ affinché il flusso f sia invertibile.

Il campionamento sotto questo modello è semplice. Campioniamo $\mathbf{z}$ dalla misura base $\mathbf{z} \sim p_{\mathbf{z}}$, e calcoliamo $\mathbf{x}$ usando il flusso e il valore realizzato di $\mathbf{z}$: $\mathbf{x} = f(\mathbf{z})$.

Come esercizio, calcoliamo la distribuzione marginale su $\mathbf{x}$ usando la formula del cambio di variabili. Per applicare questa formula, abbiamo bisogno di conoscere il flusso inverso, che è dato da $\mathbf{z} = f^{-1}(\mathbf{x})$.

$\mathbf{z} = \Lambda^{-1}(\mathbf{x} - \boldsymbol{\mu})$, e la jacobiana, che è uguale a Λ . Quindi, la densità su $\mathbf{x}$ può essere calcolata come segue:

$$\begin{aligned}
 p_{\mathbf{x}}(\mathbf{x}) &= p_{\mathbf{z}}(\mathbf{z}) \left| \det \left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1} \\
 &= \mathcal{N}(\mathbf{z}; 0, \mathbf{I}) \left| \det \left(\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \right) \right|^{-1} \\
 &= (2\pi)^{-D/2} \exp \left(-\frac{1}{2} \mathbf{z}^T \mathbf{z} \right) |\det \Lambda|^{-1} \\
 &= (2\pi)^{-D/2} (\det \Lambda \Lambda^T)^{-1/2} \exp \left(-\frac{1}{2} (\Lambda^{-1}(\mathbf{x} - \boldsymbol{\mu}))^T (\Lambda^{-1}(\mathbf{x} - \boldsymbol{\mu})) \right) \\
 &= (2\pi)^{-D/2} (\det \Lambda \Lambda^T)^{-1/2} \exp \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T (\Lambda \Lambda^T)^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \\
 &= \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Lambda \Lambda^T)
 \end{aligned}$$

dove abbiamo usato il fatto che $|\det \Lambda|^{-1} = (\det \Lambda \Lambda^T)^{-1/2}$. Quindi, $\mathbf{x}$ segue una distribuzione gaussiana con vettore medio $\boldsymbol{\mu}$ e matrice di covarianza $\Lambda \Lambda^T$. Un modo più semplice per calcolare la distribuzione marginale su $\mathbf{x}$ consiste nel riconoscere che una trasformazione lineare di una distribuzione gaussiana produce un'altra distribuzione gaussiana. Quindi, la distribuzione di probabilità su $\mathbf{x}$ può essere derivata calcolando il suo vettore medio e la sua matrice di covarianza:

$$\begin{aligned}
 \mathbb{E}[\mathbf{x}] &= \mathbb{E}[\boldsymbol{\mu} + \Lambda \mathbf{z}] = \boldsymbol{\mu} \\
 \text{Cov}[\mathbf{x}] &= \Lambda \boldsymbol{\Sigma}_{\mathbf{z}} \Lambda^T = \Lambda \mathbf{I} \Lambda^T = \Lambda \Lambda^T
 \end{aligned}$$

Pertanto,

$$p_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \Lambda \Lambda^T)$$

Tuttavia, i flussi lineari sono limitati nella loro espressività, in quanto modificano solo la media e la matrice di covarianza della distribuzione base. Anche se impiliamo più trasformazioni lineari, esse rimangono chiuse rispetto alla composizione. Questo significa che la composizione di trasformazioni lineari è equivalente a una

singola trasformazione lineare, il che limita l'espressività di questo modello, rendendolo insufficiente per modellare distribuzioni di dati molto complesse. In generale, per misure base che sono membri della famiglia esponenziale, una trasformazione lineare rimane all'interno della famiglia esponenziale, eliminando potenzialmente la necessità della formula del cambio di variabili per calcolare la densità di probabilità sulle osservazioni. Tuttavia, nel caso di flussi non lineari, questa formula è essenziale.

Inoltre, il determinante della jacobiana

$$\det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}$$

è uguale a $\det \Lambda$. Se Λ è una matrice piena, il calcolo di questo determinante richiede $\mathcal{O}(D^3)$ operazioni, cosa che può essere proibitivamente costosa per dataset a elevata dimensionalità.

Un modo per ridurre la complessità computazionale del determinante della jacobiana e del suo inverso consiste nell'imporre una qualche struttura alla trasformazione lineare, nel nostro caso alla matrice Λ , che riduca il costo computazionale dei calcoli dell'inverso e del determinante. Per esempio, la Tabella 7.1 mostra strutture utili che possono essere imposte alla matrice usata per la trasformazione lineare, insieme alla complessità computazionale per calcolarne l'inverso e il determinante. Altre strutture utili sono discusse in Kobyzev et al. (2021) e Brubaker e Köthe (2021).

Tabella 7.1 Complessità computazionale dell'inverso e del determinante della jacobiana in base alla struttura della matrice. Qui, D denota le dimensioni della variabile latente, e c rappresenta le dimensioni del blocco per matrici diagonali a blocchi o il numero di canali per convoluzioni 1×1 .

Adattato da Brubaker e Köthe (2021).

Tipo matrice	Complessità inverso	Complessità determinante
--------------	------------------------	-----------------------------

Piena	$\mathcal{O}(D^3)$	$\mathcal{O}(D^3)$
Diagonale	$\mathcal{O}(D)$	$\mathcal{O}(D)$
Triangolare	$\mathcal{O}(D^2)$	$\mathcal{O}(D)$
Diagonale a blocchi	$\mathcal{O}(c^3 D)$	$\mathcal{O}(c^3 D)$
Fattorizzata LU	$\mathcal{O}(D^2)$	$\mathcal{O}(D)$
Convoluzione speciale	$\mathcal{O}(D \log D)$	$\mathcal{O}(D)$
Convoluzione 1×1	$\mathcal{O}(c^3 + c^2 D)$	$\mathcal{O}(c^3)$

Progettazione di flussi non lineari

Come abbiamo visto nell'esempio precedente con un flusso lineare, la sua espressività è molto limitata, poiché una composizione di trasformazioni lineari è equivalente a una singola trasformazione lineare. Inoltre, abbiamo evidenziato alcuni dei requisiti principali che un flusso deve soddisfare affinché la formula del cambio di variabili funzioni, così come alcune proprietà chiave per l'espressività del modello e l'efficienza computazionale. Vogliamo flussi che siano:

- abbastanza espressivi da modellare distribuzioni di dati complesse;
- invertibili;
- differenziabili;
- computazionalmente efficienti da invertire (f^{-1});
- computazionalmente efficienti per il calcolo del determinante della jacobiano.

I *flussi* vengono anche chiamati layer di flusso o biiezioni, vedi Kobyzev et al. (2021).

Dalle reti neurali profonde, sappiamo che un modo semplice per migliorare l'espressività del modello passa attraverso la composizione di funzioni. Un aspetto importante dei modelli di flusso è che la composizione di flussi o layer invertibili rimane invertibile. Quindi, se componiamo K flussi differenti, come $f = f_k \circ f_{k-1} \circ \dots \circ f_2 \circ f_1$, e ogni f_k è invertibile e differenziabile, allora f sarà anch'essa invertibile e differenziabile.

Inoltre, il determinante della jacobiana per la funzione composta ha una forma molto comoda:

$$\begin{aligned}
 \det \mathbf{J} &= \det \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \\
 &= \det \prod_{k=1}^K \frac{\partial f_k(\mathbf{z}_k)}{\partial \mathbf{z}_k} \\
 &= \prod_{k=1}^K \det \frac{\partial f_k(\mathbf{z}_k)}{\partial \mathbf{z}_k} \\
 &= \prod_{k=1}^K \det \mathbf{J}_k
 \end{aligned}
 \tag{7.5}$$

dove

$$\mathbf{J}_k = \frac{\partial f_k(\mathbf{z}_k)}{\partial \mathbf{z}_k}$$

Quindi, sotto la composizione dei flussi, il determinante della jacobiana J è uguale al prodotto dei determinanti delle jacobiane dei singoli flussi, il che offre significativi vantaggi computazionali. Questa forma semplifica il calcolo delle log-verosimiglianze alla

somma dei log-determinanti di ogni funzione coinvolta nella composizione.

Nei prossimi paragrafi, descriveremo un modo per progettare flussi non lineari utilizzati nella pratica: i flussi di accoppiamento (*coupling flows*), i quali si basano sul principio della composizione di layer invertibili, un principio ampiamente applicato alla progettazione di flussi invertibili delle reti neurali profonde. All'interno di questa classe, descriveremo due casi specifici: NICE (*Nonlinear Independent Components Estimation*) e Real-NVP (*Non-volume Preserving Transformation*).

Flussi di accoppiamento

I flussi di accoppiamento (*coupling flows*) sono un approccio ampiamente utilizzato per costruire flussi non lineari (Dinh et al., 2015). L'idea di fondo consiste nel partizionare il vettore $\mathbf{z}$ in due parti:

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_{1:D'} \\ \mathbf{z}_{D'+1:D} \end{bmatrix} \quad (7.6)$$

Per trasformare $\mathbf{z}$ in $\mathbf{x}$, applichiamo il flusso f , tale che $\mathbf{x} = f(\mathbf{z})$. Allo stesso modo, possiamo partizionare $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{1:D'} \\ \mathbf{x}_{D'+1:D} \end{bmatrix} \quad (7.7)$$

Un flusso f è considerato un flusso di accoppiamento se:

$$f(\mathbf{z}) = \begin{bmatrix} \mathbf{x}_{1:D'} \\ \mathbf{x}_{D'+1:D} \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{1:D'} \\ f_1(\mathbf{z}_{D'+1:D}; \Theta(\mathbf{z}_{1:D'})) \end{bmatrix}$$

(7.8)

dove la prima partizione di $\mathbf{z}(\mathbf{z}_{1:d'})$ rimane non influenzata dal flusso f , e f_1 è un'altra trasformazione invertibile (un altro flusso) con parametri Θ , applicata alla seconda partizione di $\mathbf{z}(\mathbf{z}_{d'+1:d})$.

I parametri della funzione f_1 , denotati come Θ , sono definiti da una funzione arbitraria θ . Nella pratica, θ è modellata da una rete neurale, il cui input dipende solo dalla prima partizione di $\mathbf{z}(\mathbf{z}_{1:d'})$ e non dalla seconda $\mathbf{z}(\mathbf{z}_{d'+1:d})$. La funzione θ è chiamata *conditioner*, e non deve essere invertibile. La funzione f_1 è chiamata *funzione di accoppiamento* o *trasformazione di accoppiamento*, e deve essere monotona rispetto al suo argomento, quindi invertibile. La funzione f è chiamata *flusso di accoppiamento* o *layer di accoppiamento*. Come vedremo negli esempi che seguono, diversi flussi di accoppiamento ben noti differiscono nel modo in cui definiscono il conditioner e la funzione di accoppiamento. Nello specifico, esamineremo due casi: uno che utilizza una funzione di accoppiamento additiva e un altro che impiega una funzione di accoppiamento affine.

Questa struttura presenta diversi vantaggi per costruire mappature non lineari complesse, che sono invertibili. Impilando più “layer di accoppiamento”, possiamo creare flussi altamente espressivi, mantenendo efficiente il calcolo del determinante della jacobiano, come vedremo nel prossimo paragrafo.

Come illustrazione del processo, la Figura 7.2 mostra come il vettore $\mathbf{z}$ viene partizionato e poi trasformato usando il flusso f come descritto.

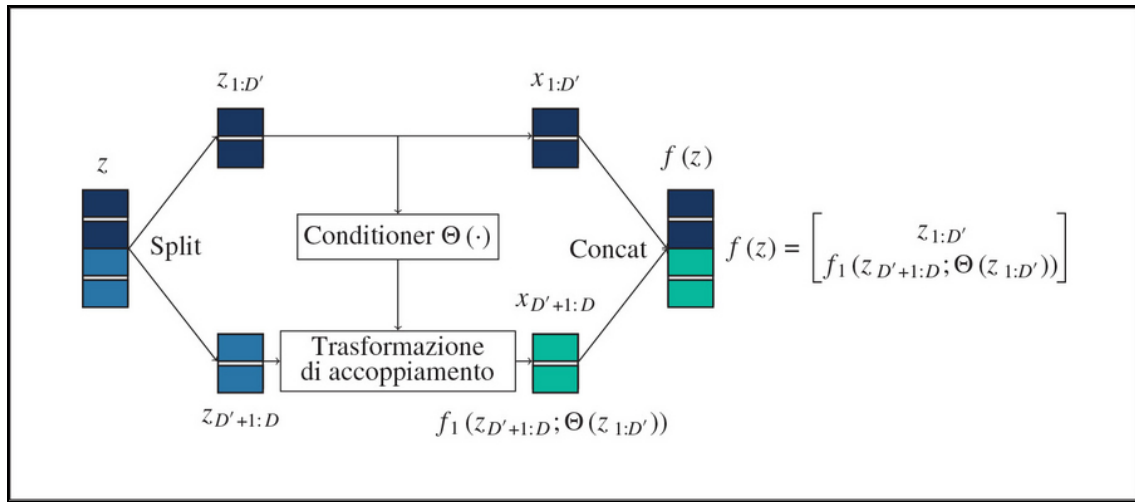


Figura 7.2 Rappresentazione grafica delle operazioni coinvolte nei flussi di accoppiamento, inclusi partizionamento e trasformazione usando un layer di accoppiamento. Adattato da Brubaker e Köthe (2021).

Nel prossimo paragrafo, ispirati dallo stile espositivo di Levine (2021), esploreremo due implementazioni specifiche di layer di accoppiamento: NICE (*Nonlinear Independent Components Estimation*) e Real-NVP (*Non-volume Preserving Transformation*).

NICE: Nonlinear Independent Components Estimation

In questo paragrafo, spieghiamo questo approccio per progettare layer invertibili sulla base del lavoro di Dinh et al. (2015).

Come accennato prima, l'approccio implica il partizionamento del vettore $\mathbf{z}$ in due parti, descritte come $\mathbf{z} = [\mathbf{z}_{1:d'}, \mathbf{z}_{d'+1:d}]^T$. A causa dei vincoli della mappatura invertibile, la dimensione di $\mathbf{x}$ deve anche essere uguale a D . Inoltre, scomponiamo le voci $\mathbf{x}$ in due componenti, cioè $\mathbf{x} = [\mathbf{x}_{1:d'}, \mathbf{x}_{d'+1:d}]^T$. In NICE, il layer invertibile è definito come segue:

$$\begin{aligned}
\mathbf{x}_{1:D'} &= \mathbf{z}_{1:D'} \\
\mathbf{x}_{D'+1:D} &= \mathbf{z}_{D'+1:D} + g_\theta(\mathbf{z}_{1:D'}) \\
(7.9)
\end{aligned}$$

NICE usa una funzione di accoppiamento additiva, che è una trasformazione invertibile con parametri definiti da $g_\theta(\mathbf{z}_{1:D'})$. Nella pratica, g_θ è implementata usando una rete neurale: un MLP, una CNN o altre architetture personalizzate per quello specifico problema. Dato il valore $\mathbf{x}$, possiamo recuperare il vettore $\mathbf{z}$ recuperando prima le sue prime D' voci: $\mathbf{z}_{1:D'} = \mathbf{x}_{1:D'}$, e poi usando questo risultato per recuperare la seconda parte delle voci $\mathbf{z}_{D'+1:D} = \mathbf{x}_{D'+1:D} - g_\theta(\mathbf{z}_{1:D'})$.

La jacobiana di questa mappatura è data da:

$$\begin{aligned}
\mathbf{J} = \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} &= \begin{bmatrix} \frac{\partial(\mathbf{x}_{1:D'})}{\partial \mathbf{z}_{1:D'}} & \frac{\partial(\mathbf{x}_{1:D'})}{\partial \mathbf{z}_{D'+1:D}} \\ \frac{\partial(\mathbf{x}_{D'+1:D})}{\partial \mathbf{z}_{1:D'}} & \frac{\partial(\mathbf{x}_{D'+1:D})}{\partial \mathbf{z}_{(D'+1):D}} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{I}_{D'} & 0 \\ g'_\theta(\mathbf{z}_{1:D'}) & \mathbf{I}_{D-D'} \end{bmatrix} \\
(7.10)
\end{aligned}$$

dove $\mathbf{I}_{D'}$ è una matrice identità con D' elementi diagonali.

Poiché siamo interessati al determinante della jacobiana, non abbiamo bisogno di calcolare la voce (2, 1) di $\mathbf{J}$, poiché è zero, e $g'_\theta(\mathbf{z}_{1:D'})$ non contribuirà al risultato. Quindi, il determinante della jacobiana è $\det \mathbf{J} = 1$.

La log-verosimiglianza per questo modello è data da:

$$\begin{aligned}
\mathcal{L} &= \sum_n \log p(\mathbf{z}_n) - \log \left| \det \left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right) \right| \\
&= \sum_n \log p(\mathbf{z}_n) - \log 1 \\
&= \sum_n \log p(\mathbf{z}_n)
\end{aligned}$$

(7.11)

il che è computazionalmente comodo. Tuttavia, il fatto che il determinante della jacobiana sia costante e uguale a 1 significa che questo flusso trasforma solo la misura base ma non ne cambia la scala. Questo limita l'espressività del modello e rende difficile adattare distribuzioni di dati molto complesse (vedi Levine [2021]). Pertanto, in contesti pratici, spesso nel modello viene inserito un layer di rescaling (vedi Dinh et al. [2015]).

Nella pratica, impiliamo diversi di questi layer invertibili per addestrare efficacemente modelli di flusso generativi. Come esempio illustrativo delle performance di questi modelli su semplici immagini, nella Figura 7.3, le sezioni (a) e (b) mostrano campioni da un modello NICE che è stato addestrato per modellare la distribuzione del semplice dataset di cifre MNIST e immagini base dal dataset TFD. L'architettura utilizzata in entrambi gli esperimenti è costituita da una pila di quattro layer/funzioni NICE. Ogni layer NICE è implementato come una rete profonda rettificata (*deep rectified network*) con unità di output lineari. Gli autori usano la stessa architettura di rete per tutte le funzioni NICE: cinque layer nascosti con 1.000 unità ciascuno per il dataset MNIST e quattro layer nascosti con 5.000 unità ciascuno per il dataset TFD. Come osservato, le immagini di cifre generate (sezione a della figura) somigliano a cifre reali del dataset MNIST, e le immagini di volti semplici generate (sezione b della figura) somigliano in una certa

misura a volti. Questo modello, sviluppato nel 2015, è stato il primo modello di flusso di normalizzazione capace di generare immagini di qualità ragionevole per il suo tempo. Parleremo più avanti di come estendere questi modelli per catturare dipendenze temporali, e vedremo anche un esempio di applicazioni finanziarie.

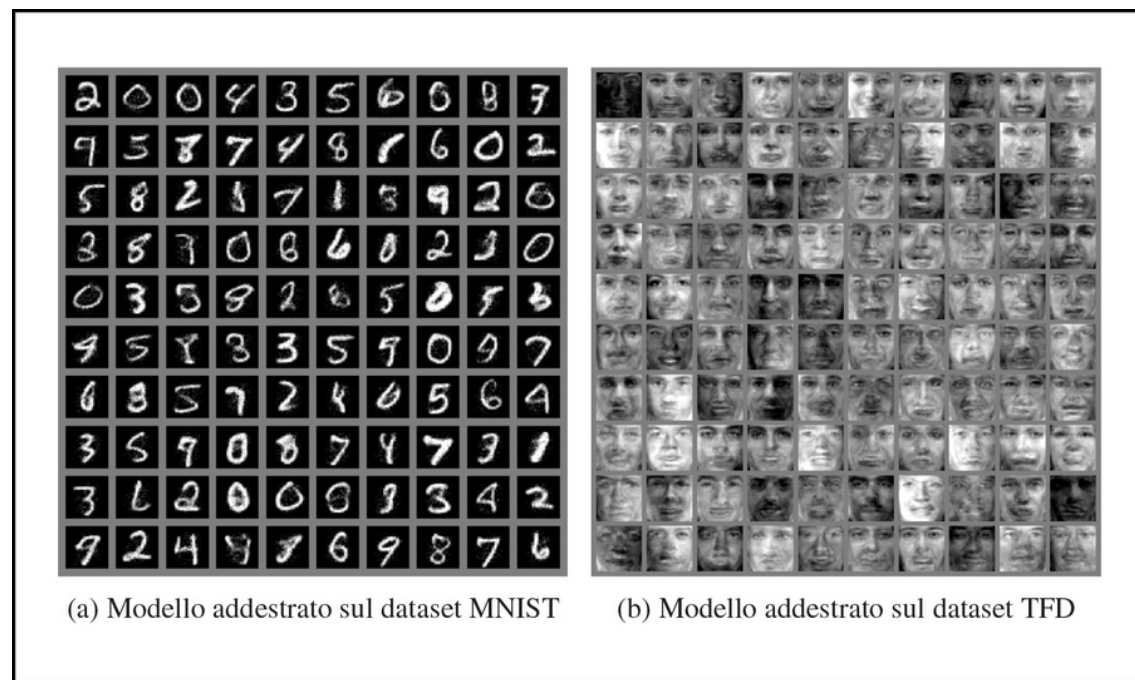


Figura 7.3 Illustrazione di campioni non distorti (unbiased) generati dal modello NICE addestrato sul dataset MNIST (a) e sul dataset TFD (b). Fonte: Dinh, Krueger, and Bengio (2015).

Tuttavia, quando il modello viene addestrato su dataset più complessi, come mostra la Figura 7.4, la sua performance peggiora. Nella sezione (c), vediamo campioni generati da un modello addestrato sul dataset SVHN, costituito da immagini a colori di cifre prese da numeri civici. Il modello fatica a generare questo tipo di cifre a colori. Quando addestrato su CIFAR-10, che è un dataset più complesso contenente un insieme diversificato di immagini realistiche di oggetti, il modello genera immagini di qualità molto

bassa, mostrando i propri limiti nel gestire distribuzioni di dati più complesse.

Nel notebook associato a questo capitolo ([NICE Notebook]), implementiamo e proviamo il flusso di accoppiamento NICE con architetture personalizzate per generare dati finanziari azionari, che possono essere facilmente personalizzate per le vostre applicazioni. Inoltre, includiamo risultati sperimentali per dimostrare le performance e le applicazioni pratiche di questo approccio.

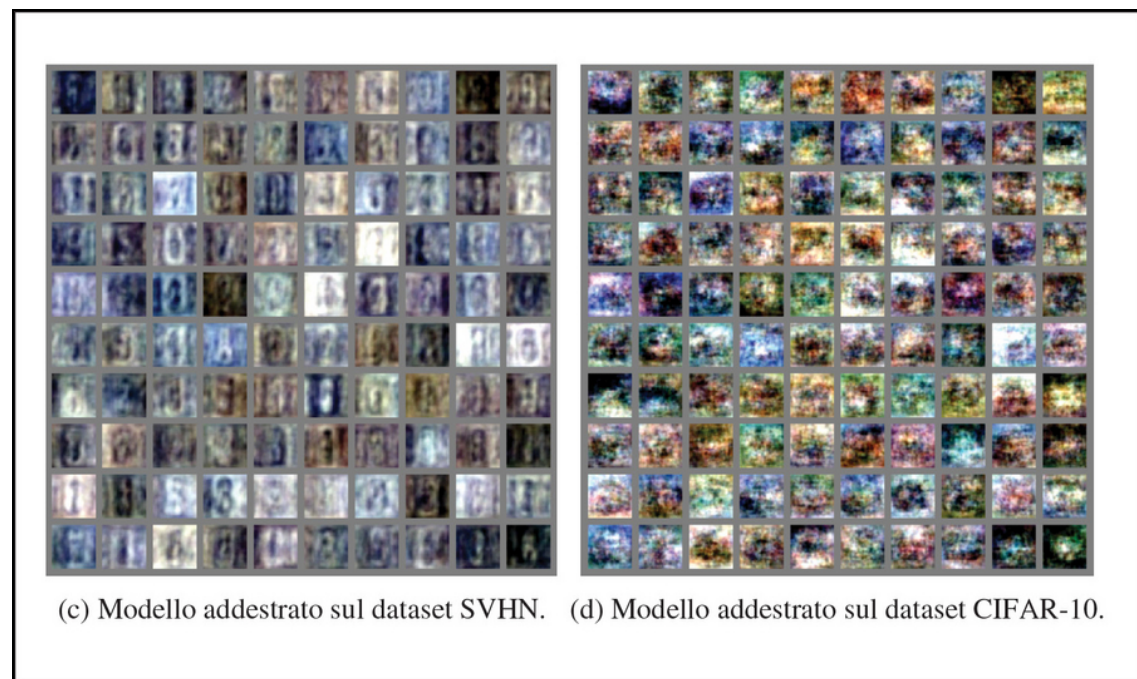


Figura 7.4 Illustrazione di campioni non distorti generati dal modello NICE addestrato sul dataset SVHN (c) e sul dataset CIFAR-10 (d). Fonte: Dinh, Krueger, and Bengio (2015).

Real-NVP: Non-volume Preserving Transformation

La Real-NVP (*Non-volume Preserving Transformation*) introdotta da Dinh et al. (2017) propone un nuovo tipo di layer invertibile che

migliora l'espressività del modello. In particolare, il determinante della jacobiano di questo modello non è costante, cosa che permette una maggiore flessibilità. In modo simile al modello NICE, in Real-NVP partizionano i vettori $\mathbf{z}$ e $\mathbf{x}$ in due parti, cioè $\mathbf{z} = [\mathbf{z}_{1:d'}, \mathbf{z}_{d'+1:d}]^T$ e $\mathbf{x} = [\mathbf{x}_{1:d'}, \mathbf{x}_{d'+1:d}]^T$. Il layer invertibile proposto (funzione di accoppiamento) è definito come segue:

$$\begin{aligned}\mathbf{x}_{1:D'} &= \mathbf{z}_{1:D'} \\ \mathbf{x}_{D'+1:D} &= \mathbf{z}_{D'+1:D} \odot \exp(h_{\theta}(\mathbf{z}_{1:D'})) + g_{\theta}(\mathbf{z}_{1:D'})\end{aligned}\quad (7.12)$$

dove $\odot$ sta per il prodotto elemento per elemento, e h_{θ} e g_{θ} sono layer della rete neurale che possono essere implementati usando architetture come MLP, CNN o altri, personalizzati per vostro specifico problema. Come definito nell'Equazione 7.12, Real-NVP usa una funzione di accoppiamento affine, che per definizione è una trasformazione invertibile con parametri dati da $\{h_{\theta}(\mathbf{z}_{1:d'}), g_{\theta}(\mathbf{z}_{1:d'})\}$. Questa può essere interpretata come una trasformazione di scala e posizione della seconda partizione di $\mathbf{z}(\mathbf{z}_{d'+1:d})$.

Per recuperare il vettore $\mathbf{z}$ da $\mathbf{x}$, applichiamo una trasformazione simile a quella usata nel modello NICE:

$$\begin{aligned}\mathbf{z}_{1:D'} &= \mathbf{x}_{1:D'} \\ \mathbf{z}_{D'+1:D} &= (\mathbf{x}_{D'+1:D} - g_{\theta}(\mathbf{z}_{1:D'}))/\exp(h_{\theta}(\mathbf{z}_{1:D'}))\end{aligned}\quad (7.13)$$

Data questa mappatura, possiamo derivare la jacobiana, che è data da:

$$\begin{aligned}
\mathbf{J} = \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} &= \begin{bmatrix} \frac{\partial(\mathbf{x}_{1:D'})}{\partial \mathbf{z}_{1:D}} & \frac{\partial(\mathbf{x}_{1:D'})}{\partial \mathbf{z}_{D'+1:D}} \\ \frac{\partial(\mathbf{x}_{D'+1:D})}{\partial \mathbf{z}_{1:D'}} & \frac{\partial(\mathbf{x}_{D'+1:D})}{\partial \mathbf{z}_{(D'+1):D}} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{I} & 0 \\ g'_{\theta}(\mathbf{z}_{1:D'}) & \text{diag}(\exp(h_{\theta}(\mathbf{z}_{1:D'}))) \end{bmatrix}
\end{aligned}$$

(7.14)

La jacobiana è triangolare, quindi il suo determinante è il prodotto delle voci diagonali:

$$\begin{aligned}
\det \mathbf{J} &= \prod_{d=D'+1}^D \exp(h_{\theta}(\mathbf{z}_{1:D'})_d) \\
&= \exp\left(\sum_{d=D'+1}^D h_{\theta}(\mathbf{z}_{1:D'})_d\right)
\end{aligned}$$

(7.15)

Qui possiamo vedere che il determinante della jacobiana non è costante, cosa che permette il rescaling della misura base e rende possibile una maggiore espressività del modello, per adattare distribuzioni di dati molto complesse. Questo rende il modello RealNVP più espressivo rispetto al modello NICE.

Ancora una volta, come esempio illustrativo delle sue performance su immagini semplici, la Figura 7.5 mostra campioni generati dal modello RealNVP, vedi Dinh et al. (2017). Gli autori dell'articolo presentano i risultati di un modello RealNVP addestrato su vari dataset: CIFAR-10, ImageNet (32×32), ImageNet (64×64), CelebA e LSUN (bedroom). In ogni coppia di colonne, la colonna a sinistra mostra esempi dal dataset, mentre la colonna a destra visualizza campioni generati dal modello addestrato su quel dataset.

Gli autori di questo lavoro hanno impiegato un'architettura multi-scala, come descritto nel Paragrafo 3.6 di Dinh et al. (2017). Questa architettura incorpora una sequenza di layer *coupling-squeezing-coupling*, utilizzando una rete residua convoluzionale profonda all'interno delle trasformazioni di accoppiamento. Per una spiegazione più dettagliata dell'architettura utilizzata nei loro esperimenti, incoraggiamo i lettori a fare riferimento al lavoro di Dinh et al. (2017).

Come già osservato, questo modello può generare campioni molto più realistici, incluse immagini di vari oggetti, camere, scene naturali e volti umani.

Un altro aspetto interessante dimostrato nell'articolo su RealNVP è la capacità del modello di apprendere rappresentazioni. Come abbiamo detto all'inizio del capitolo, un limite di questi modelli, essendo invertibili, è che le dimensioni della variabile latente $\mathbf{z}$ e della variabile osservata $\mathbf{x}$ devono essere le stesse, dove $\mathbf{x}$ è a elevata dimensionalità. Normalmente, preferiamo che la dimensione latente sia inferiore a quella dei dati, come nei VAE studiati nel Capitolo 6, per fornire una rappresentazione compressa dei dati di input.

Nella Figura 7.6, gli autori mostrano la capacità di Real-NVP di catturare una rappresentazione utile dei dati nello spazio latente. Questa figura mostra nuove immagini generate tramite interpolazioni nello spazio latente tra quattro esempi per diversi dataset. I dataset utilizzati, in senso orario dall'alto a sinistra, sono CelebA, ImageNet (64×64), LSUN (tower) e LSUN (bedroom).



Figura 7.5 Illustrazione con i campioni generati dal modello Real-NVP su quattro dataset: CIFAR-10, ImageNet (32×32), ImageNet (64×64), CelebA e LSUN (bedroom). Per ogni dataset (le varie sottofigure), la colonna a sinistra mostra esempi reali, mentre la colonna a destra visualizza campioni generati dal modello addestrato sul rispettivo dataset. Fonte: Dinh, Sohl-Dickstein and Bengio (2017).

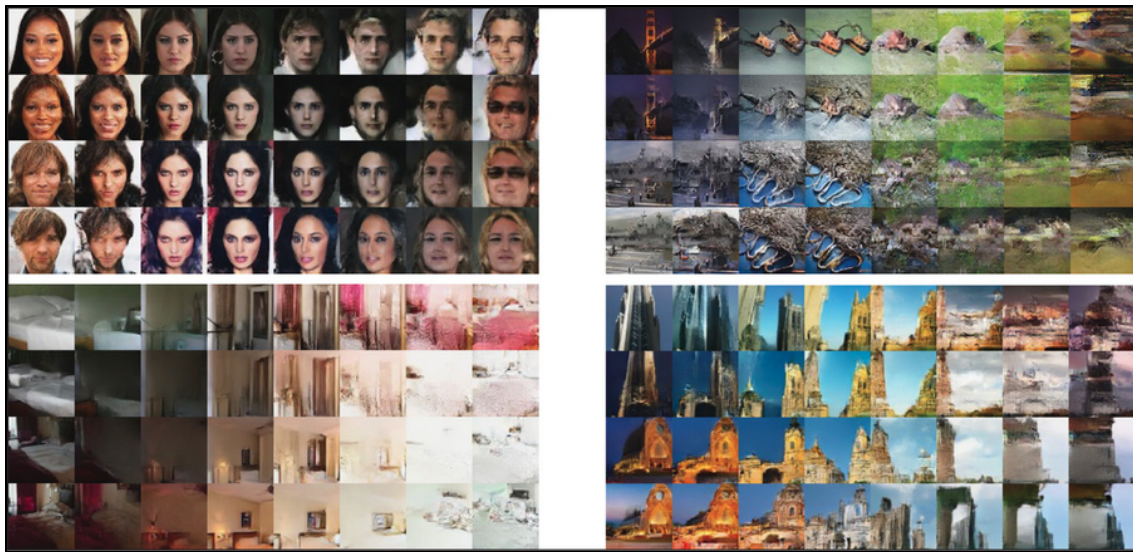


Figura 7.6 Illustrazione di nuove immagini generate attraverso interpolazioni tra quattro esempi di dataset. I dataset utilizzati sono, in senso orario dall'alto a sinistra: CelebA, ImageNet (64×64), LSUN (tower) e LSUN (bedroom). Fonte: Dinh, Sohl-Dickstein e Bengio (2017).

Per ogni dataset, vengono posizionati quattro esempi agli angoli di ogni sotto-immagine, e vengono generate le interpolazioni tra di essi. I risultati suggeriscono che, anche se lo spazio latente è a elevata dimensionalità, il modello apprende comunque una struttura interessante, in cui immagini simili nello spazio dei dati sono vicine anche nello spazio latente. Ciò indica che il modello sta catturando una rappresentazione significativa e semantica che va oltre la semplice interpolazione nello spazio dei pixel.

Per verificare ulteriormente se lo spazio latente ha un'interpretazione semantica coerente, gli autori hanno addestrato sul dataset CelebA un modello condizionato sulla classe. Hanno scoperto che la rappresentazione appresa aveva un significato semantico coerente rispetto alle etichette delle classi, come descritto in Dinh et al. (2017).

Nel notebook allegato a questo capitolo ([`RealNVP Notebook`]), implementiamo e proviamo il flusso ad accoppiamento Real-NVP con architetture personalizzate per generare dati finanziari azionari, che potrete facilmente personalizzare per le vostre applicazioni. Più avanti, vedremo come estendere questi modelli per catturare le dipendenze temporali, insieme a un'applicazione ai dati finanziari.

Nei prossimi paragrafi, introdurremo brevemente due approcci aggiuntivi per la costruzione di flussi: i flussi autoregressivi e i flussi normalizzanti continui.

Flussi autoregressivi

I flussi autoregressivi (*Autoregressive Flows*) sono, in effetti, una delle prime classi di flussi sviluppate (vedi Papamakarios et al. [2021]). Questi flussi sono motivati dall'idea che la distribuzione congiunta su $\mathbf{x}$ possa essere scomposta in un prodotto di distribuzioni di probabilità condizionate unidimensionali usando la regola della catena, come introdotto nel Capitolo 5 sui modelli autoregressivi. Parametrizzando ogni distribuzione condizionata, questi fattori formano la base di un flusso normalizzante noto come MAF (*Masked Autoregressive Flow*) (vedi Papamakarios et al. [2017]; Bishop e Bishop [2023]).

Alcuni dei nostri esperimenti, più avanti, incorporeranno flussi MAF come blocchi costitutivi nell'architettura di un modello. Per quanto riguarda la flessibilità di modellazione dei flussi autoregressivi, un risultato interessante, presentato in Papamakarios et al. (2021), mostra che tutti i modelli autoregressivi per variabili continue possono essere interpretati come flussi autoregressivi con un singolo layer autoregressivo. Per saperne di più su questo risultato, potete fare riferimento a Papamakarios et al. (2021).

Flussi normalizzanti continui

Un altro tipo di modello di flusso è quello dei flussi normalizzanti continui (*Continuous Normalizing Flows*), che adottano un approccio differente per definire la mappatura da $\mathbf{z}$ a $\mathbf{x}$. Nei paragrafi precedenti, abbiamo esaminato la costruzione dei flussi parametrizzando una funzione f e applicando una sequenza finita di trasformazioni (per esempio, K layer di accoppiamento in Real-NVP) per trasformare la misura base $p(\mathbf{z})$ in una più complessa distribuzione $p(\mathbf{x})$.

Al contrario, i flussi normalizzanti continui vedono la mappatura da $\mathbf{z}$ a $\mathbf{x}$ come un processo continuo. Invece di applicare un numero finito di trasformazioni discrete, la trasformazione può essere immaginata, in parole povere, come l'applicazione di un numero infinito di trasformazioni infinitesime. Questo processo è descritto matematicamente da un'equazione differenziale ordinaria neurale (*Neural ODE*). In questo framework, il flusso è definito da un'ODE e, durante il processo di apprendimento, i parametri dell'ODE vengono appresi dai dati, piuttosto che dai singoli flussi discreti, come nei casi precedenti che abbiamo studiato. I flussi normalizzanti continui possono essere utilizzati per generare il complesso processo stocastico continuo dei dati finanziari (vedi Deng et al. [2020]). Per maggiori dettagli sui flussi normalizzanti continui, incoraggiamo il lettore a consultare Chen et al. (2018) e Papamakarios et al. (2021).

Modellazione di serie temporali finanziarie con i modelli di flusso

Transizione dalle immagini alla dinamica delle serie temporali

Nei paragrafi precedenti, abbiamo introdotto due modelli di flusso, NICE e Real-NVP, modelli che creano flussi differenziabili, invertibili e che permettono un calcolo rapido delle log-verosimiglianze. L'efficienza nel calcolo della log-verosimiglianza è principalmente dovuta alla struttura imposta in questi flussi, che semplifica il calcolo del determinante della jacobiana. Finora abbiamo utilizzato immagini come esempi per dimostrare la capacità di questi modelli di approssimare distribuzioni di dati complesse (e anche per evidenziare il campo in cui questi modelli sono stati originariamente sviluppati), ma i flussi non si limitano a questo dominio e possono essere utilizzati ogni volta che abbiamo bisogno di modellare distribuzioni di probabilità complesse. Inoltre possono essere estesi ad altre aree, come la modellazione delle serie temporali, aprendo nuove opportunità di modellazione.

Per passare alle serie temporali, possiamo iniziare semplicemente dando un'altra interpretazione ai dati $\mathbf{x}$. Invece di pensare a $\mathbf{x}^n$ come a un vettore di intensità dei pixel per la n -esima immagine nel nostro dataset, possiamo pensare a $\mathbf{x}^n$ come a un vettore che rappresenta una serie temporale multivariata al tempo n . Ogni componente o “dimensione” potrebbe corrispondere a una variabile, come i rendimenti di un determinato titolo al tempo n , dove il vettore $\mathbf{x}^n$ fornisce un'istantanea dei rendimenti trasversali in quel momento. Per semplicità, useremo t per indicizzare il tempo invece di n , poiché è più intuitivo nel contesto delle serie temporali.

In questa configurazione, flussi come NICE o Real-NVP possono modellare le relazioni trasversali (cioè le correlazioni) tra le serie temporali al tempo t . Tuttavia, per definizione, queste architetture ipotizzano un'indipendenza tra le osservazioni consecutive. Ricordiamo come funziona il campionamento nei modelli di flusso standard: per generare le osservazioni $\mathbf{x}_t$, campioniamo $\mathbf{z}_t$ dalla misura base, gli applichiamo il flusso per produrre $\mathbf{x}_t$, e ripetiamo questo processo per ogni t . Questa ipotesi di indipendenza può limitare le prestazioni di questi modelli a certe scale temporali, dove le dipendenze tra le osservazioni possono essere significative.

Adattamento dei flussi alle serie temporali

I flussi convenzionali, come NICE e Real-NVP, sono efficaci nel modellare le dinamiche trasversali, ma per definizione impongono l'indipendenza tra le osservazioni. Questo potrebbe non essere sufficiente per modellare le serie temporali finanziarie a certe granularità temporali, dove i prezzi/rendimenti passati, le condizioni di mercato e altri fattori influenzano significativamente i prezzi/rendimenti futuri.

Quindi, come possiamo adattare i flussi per catturare le dinamiche temporali (dipendenze nel tempo) presenti nei dati che vogliamo modellare? Ecco alcune idee.

1. *Dinamiche dello stato latente*: introdurre dipendenze temporali nelle variabili latenti, in modo che $\mathbf{z}_t$ dipenda dalle variabili latenti passate $\mathbf{z}_{t-1}$, $\mathbf{z}_{t-2}$, ..., deterministicamente o stocasticamente:

$$\mathbf{z}_t \sim P_{\mathbf{z}_t | \mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \dots}$$

$$\mathbf{x}_t = f(\mathbf{z}_t)$$

Ciò rispecchia l'idea alla base di modelli come il filtro di

Kalman o le reti neurali ricorrenti, dove gli stati latenti dipendono dallo spazio latente precedente.

2. *Dipendenze di tipo autoregressivo (AR)*: estendere il processo generativo per dipendere dalle osservazioni passate, rendendo $\mathbf{x}_t$ una funzione sia della variabile latente $\mathbf{z}_t$ sia delle osservazioni passate $\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \dots$:

$$\mathbf{x}_t = f(\mathbf{z}_t, \phi(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \dots, \mathbf{x}_{t-p}))$$

dove ϕ è una funzione che può codificare l'effetto delle ultime p osservazioni su $\mathbf{x}_t$. Questo approccio rispecchia i modelli autoregressivi.

3. *Approcci ibridi*: combinare i due metodi, permettendo a $\mathbf{x}_t$ di dipendere sia dalle osservazioni passate sia dalle variabili latenti passate:

$$\mathbf{x}_t = f(\mathbf{z}_t, \phi(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \dots, \mathbf{x}_{t-p}), \psi(\mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \dots, \mathbf{z}_{t-q}))$$

dove ϕ codifica l'effetto delle ultime p osservazioni e ψ l'effetto delle ultime q variabili latenti.

4. *Flussi variabili nel tempo*: permettere al flusso di variare con il tempo f_t .

Caso di studio: un esempio pratico – Flussi normalizzanti condizionati

Per illustrare queste idee, esploriamo alcune implementazioni pratiche di flussi per modellare la dinamica delle serie temporali. Un esempio notevole è presentato nell'articolo *Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows* (Rasul et al., 2021). In questo scritto, gli autori utilizzano flussi Real-NVP ma li adattano per gestire dipendenze temporali. L'idea chiave consiste nel condizionare la trasformazione del flusso f su uno stato nascosto $\mathbf{h}_t$, che codifica le informazioni sul passato:

$$\mathbf{x}_t = f(\mathbf{z}_t, \mathbf{h}_t)$$

Quindi, come otteniamo $\mathbf{h}_t$? Gli autori esplorano due approcci.

1. *Stato nascosto basato su reti neurali ricorrenti*: la prima opzione consiste nell'utilizzare una rete neurale ricorrente (come una LSTM o una GRU) per riassumere le osservazioni passate e le covariate in $\mathbf{h}_t$:

$$\mathbf{h}_t = \text{RNN}(\text{concat}(\mathbf{x}_{t-1}, \mathbf{c}_{t-1}), \mathbf{h}_{t-1})$$

Qui $\mathbf{c}_{t-1}$ include eventuali covariate aggiuntive (per esempio, nelle serie temporali finanziarie, pensate a indicatori di mercato aggiuntivi o a variabili macroeconomiche o a variabili categoriche).

2. *Stato nascosto basato su transformer*: la seconda opzione consiste nell'utilizzare un'architettura transformer encoder-decoder. Qui, l'encoder incorpora le osservazioni passate e le covariate, mentre il decoder genera gli stati nascosti per il flusso.

Come sappiamo, sia le reti neurali ricorrenti sia i transformer sono efficaci nel modellare i dati sequenziali, e i transformer si comportano meglio delle reti neurali ricorrenti nel catturare dipendenze a lungo raggio.

A questo punto, potreste notare che il limite è solo la nostra creatività quando progettiamo architetture per catturare dinamiche temporali con i flussi. Questo è un altro esempio di come i mattoni costitutivi che abbiamo trattato nei capitoli precedenti si uniscono, combinando i flussi con i più potenti modelli sequenziali (le reti neurali ricorrenti o i transformer) per raggiungere lo stesso obiettivo.

Esplorare esempi come questo può dare vita a molte idee per personalizzare le soluzioni. Potreste considerare di addestrare il

modello in modo differente, di aggiungere o rimuovere componenti o di adattare l'architettura alle specifiche esigenze del problema .

L'importanza della conoscenza del dominio nelle serie temporali finanziarie

Nella modellazione delle serie temporali finanziarie, è fondamentale incorporare la conoscenza del dominio durante il processo di modellazione. Questo campo è ben noto per il suo basso rapporto segnale-rumore e la sua non-stazionarietà, il che viola le ipotesi fondamentali di indipendenza e distribuzione identica che stanno alla base della maggior parte delle soluzioni di machine learning. Questi problemi non consentono di ottenere buoni risultati con i modelli generici.

Ciò significa che, in questo dominio, la maggior parte delle volte non possiamo semplicemente lanciare un modello generico sui dati e aspettarci grandi risultati. Chissà, forse nel prossimo futuro, magari anche pochi mesi dopo la pubblicazione di questo libro, le cose potrebbero cambiare. Per ora, concentriamoci sul personalizzare le nostre soluzioni per il problema specifico, in modo da poter estrarre informazioni preziose e, si spera, trarre anche qualche profitto.

Nel notebook allegato a questo capitolo, esploriamo queste idee utilizzando il dataset Exchange Rate, che contiene tassi di cambio giornalieri per le valute di otto paesi (Australia, Regno Unito, Canada, Svizzera, Cina, Giappone, Nuova Zelanda e Singapore) dal 1990 al 2016. Tratteremo i componenti principali dell'implementazione e discuteremo variazioni nell'architettura e le sue prestazioni su altri dataset.

Conclusioni

In questo capitolo, abbiamo visto come i modelli di flusso siano una potente aggiunta al nostro toolbox per gestire dati a elevata dimensionalità. Essi combinano le capacità di campionamento veloce dei tradizionali modelli a variabili latenti con il vantaggio chiave del calcolo esatto della verosimiglianza. Progettando flussi con la giusta struttura, possiamo rendere molto efficienti molti calcoli, come quello del determinante del log jacobiano, che è un elemento chiave per il calcolo della verosimiglianza.

Abbiamo anche visto come estendere i flussi per catturare le dipendenze temporali. Questo apre nuove opportunità per combinare i modelli di flusso con le architetture che sappiamo eccellere in questo compito, come i modelli autoregressivi basati su reti neurali ricorrenti e transformer, entrambi ottimi nel modellare dati di natura sequenziale.

Padroneggiando i modelli di flusso, aggiungete uno strumento flessibile ed efficiente ai vostri strumenti di modellazione, con la possibilità di modellare dati multimodali a elevata dimensionalità.

Capitolo 8

Reti generative antagoniste

Nei capitoli precedenti, abbiamo esplorato la modellizzazione di distribuzioni di dati complesse a elevata dimensionalità utilizzando modelli autoregressivi, autoencoder variazionali (VAE) e modelli di flusso. Sebbene ciascuno di questi modelli presenti i suoi vantaggi e svantaggi in termini di calcolo – esatto o approssimato – delle densità, efficienza del campionamento e apprendimento delle rappresentazioni, tutti hanno una caratteristica in comune: sono modelli basati sulla verosimiglianza. Ciò significa che utilizzano la stima della massima verosimiglianza (MLE, *Maximum Likelihood Estimation*) come approccio per apprendere i parametri. Nella letteratura sull'apprendimento generativo, questi modelli sono chiamati *modelli espliciti*.

In questo capitolo, approfondiremo un approccio differente per l'apprendimento di modelli generativi, indicato in letteratura come *modelli impliciti* (vedi Figura 8.1). Questi modelli non richiedono la capacità di calcolare la densità di probabilità. Questo capitolo si concentrerà sulle reti generative antagoniste (GAN, *Generative Adversarial Networks*) come esempio prominente di modelli impliciti, con un processo di addestramento che si focalizza sulla generazione di campioni realistici e di alta qualità in elevata dimensionalità piuttosto che sull'effettuare stime della densità.

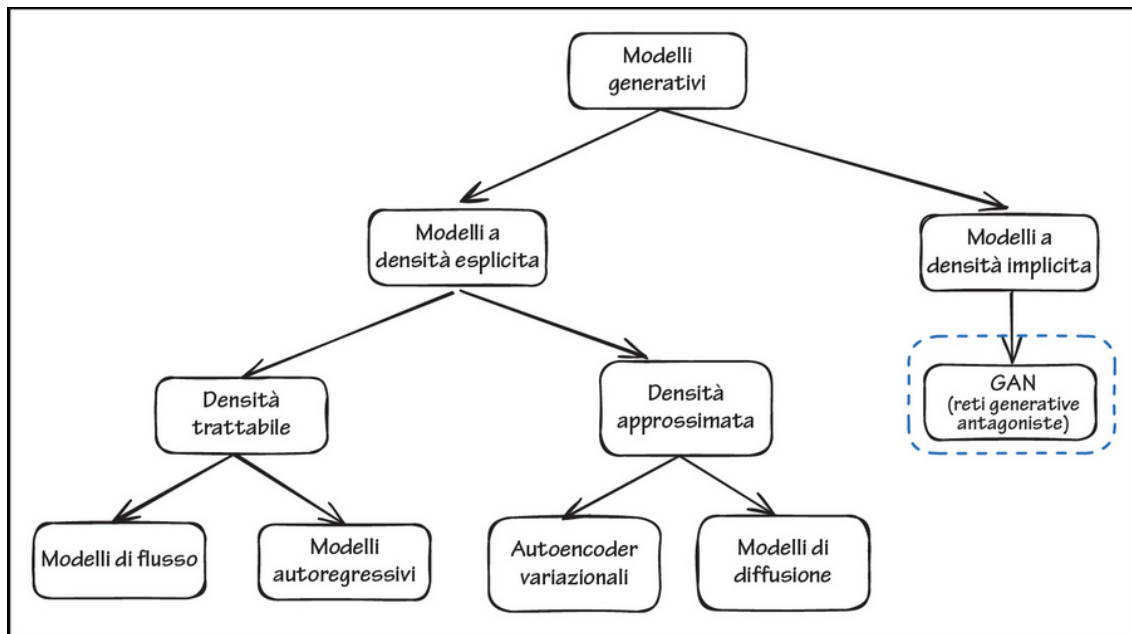


Figura 8.1 Grafico ad albero della tassonomia dei modelli: le reti generative antagoniste (GAN).

Introduzione

Nei capitoli precedenti, abbiamo esaminato gli obiettivi primari dei modelli generativi: costruire modelli per dati a elevata dimensionalità che possano realizzare quanto segue:

- calcolare le densità di probabilità;
- eseguire l'apprendimento delle rappresentazioni (identificando le strutture nascoste);
- generare nuovi campioni.

Scegliere una singola metrica che valuti la qualità di tutti questi obiettivi può essere difficile. Per esempio, una determinata metrica o un certo obiettivo di addestramento potrebbe essere efficace per l'apprendimento delle rappresentazioni ma non per generare campioni di alta qualità. Tuttavia, intuitivamente, se un modello può

approssimare finemente la distribuzione dei dati, in teoria dovrebbe comportarsi bene anche nelle altre attività. Famiglie di modelli come i modelli autoregressivi, i VAE e i modelli di flusso utilizzano questo principio, e durante il processo di apprendimento cercano di allineare il più possibile la distribuzione del modello alla distribuzione dei dati. Questo allineamento assicura che il modello risultante non sia solo efficace nello stimare la densità, ma fornisca anche strutture nascoste utili e campioni di alta qualità. Tutte queste famiglie di modelli propongono una classe di modelli a densità trattabile o approssimata, in modo che l'apprendimento possa basarsi sulla stima della massima verosimiglianza, che abbiamo visto essere equivalente a minimizzare la divergenza di Kullback-Leibler tra la distribuzione dei dati e la distribuzione del modello.

Le reti generative antagoniste (GAN, *Generative Adversarial Network*), al contrario, concentrano il loro processo di addestramento sulla produzione di un modello che possa generare campioni di alta qualità senza la necessità di calcolare esplicitamente la stima della densità. L'inventore delle GAN, Ian Goodfellow, ha descritto la sua soluzione nel suo articolo (Goodfellow et al., 2014), dettagliando il risultato. La sua idea, molto arguta, è stata quella di utilizzare una diversa rete neurale, il *discriminatore*, il cui unico compito è quello di valutare se un campione osservato è stato generato dalla distribuzione dei dati o dalla distribuzione del modello, e poi di utilizzare il segnale fornito dal discriminatore per migliorare la distribuzione del modello. La distribuzione del modello è chiamata *generatore* nella terminologia delle GAN. Idealmente, un buon generatore produrrà campioni indistinguibili dalla distribuzione effettiva dei dati.

Yann LeCun, che ha ricevuto il Premio Turing 2018, insieme a Yoshua Bengio e Geoffrey Hinton, per il loro lavoro sul deep

learning, ha dichiarato che le GAN sono “l’idea più interessante degli ultimi dieci anni nel machine learning”. Come nota storica, Ian Goodfellow concepì le reti generative antagoniste durante un brainstorming sulle tecniche di programmazione con alcuni amici in un bar. Per maggiori dettagli, leggete l’intervista all’indirizzo <https://www.deeplearning.ai/the-batch/ian-goodfellow-a-man-a-plan-a-gan>.

In modo simile a quanto avviene nei VAE o nei modelli di flusso, nelle GAN il processo di campionamento del generatore consiste nel campionare prima una variabile latente $\mathbf{z}$ da una distribuzione a priori p_z , scelta in modo da essere facile da campionare, e poi mappare lo spazio delle variabili latenti sullo spazio delle osservazioni o dei dati utilizzando il generatore, denotato da G_θ e parametrizzato da θ . Questa mappatura può essere probabilistica o deterministica. Per una mappatura deterministica G_θ , il campionamento può essere calcolato in modo molto efficiente, come nei modelli di flusso. Il processo di campionamento nelle GAN è dato da:

$$\begin{aligned}\mathbf{z} &\sim p(\mathbf{z}) \\ \mathbf{x} &= G_\theta(\mathbf{z})\end{aligned}\tag{8.1}$$

Una differenza chiave rispetto ai modelli di flusso è che, poiché per l’apprendimento del modello non stiamo utilizzando la densità, il generatore può essere una qualsiasi rete neurale. Non c’è alcun requisito che la mappatura sia invertibile, e le dimensioni della variabile latente $\mathbf{z}$ e delle osservazioni $\mathbf{x}$ possono essere differenti, come nei VAE, il che può fornire rappresentazioni apprese interessanti.

A differenza dei modelli di flusso, dove addestriamo solo l’encoder o il flusso, le GAN richiedono l’apprendimento dei parametri di due

modelli differenti, simultaneamente: la rete neurale generatrice G_θ e la rete neurale discriminatrice, denotata da D_ϕ e parametrizzata da ϕ .

L'idea generale di ciò che accade durante l'addestramento è la seguente. Addestriamo il discriminatore, che è un classificatore binario che classifica dati reali (generati dalla distribuzione dei dati), rispetto a dati falsi (generati dal generatore), ottimizzando i suoi parametri per diventare sempre più abile nel discriminare queste classi. Per addestrare il discriminatore, costruiamo il “dataset reale”, denotato da $\mathcal{D}_r$, raccogliendo dati reali relativi alla nostra applicazione (per esempio immagini di volti) e assegnando l'etichetta target vero o positivo (o 1) a tutti loro. Costruiamo il “dataset falso”, denotato da $\mathcal{D}_f$, inizializzando casualmente il generatore G_θ , campionando un certo numero di immagini e assegnando l'etichetta target falso o negativo (o 0) a tutte loro. Con il dataset combinato $\mathcal{D} = \mathcal{D}_r \cup \mathcal{D}_f$, addestriamo il discriminatore D_ϕ . Una volta addestrato il discriminatore, utilizziamo il segnale fornito dal discriminatore stesso come una funzione loss per addestrare G_θ , e iteriamo.

Pertanto, il processo di addestramento delle GAN implica un gioco in cui il discriminatore cerca di migliorarsi nel classificare i dati reali rispetto a quelli falsi, e il generatore cerca di migliorarsi nell'ingannare il discriminatore, facendogli classificare come reali i dati falsi. Questa è la parte “antagonista” del gioco, che alla fine può raggiungere un equilibrio chiamato *equilibrio di Nash*.

Le GAN sono note per generare immagini di altissima qualità a elevata dimensionalità. Nella Figura 8.2, vediamo una progressione di campioni generati da GAN in un arco di quattro anni e mezzo, dal lavoro originale di Goodfellow et al. (2014) dal 2014 al 2018. Tuttavia, le GAN sono note per essere difficili da addestrare. Nei prossimi paragrafi, approfondiremo il processo di addestramento e ciò che le GAN stanno facendo effettivamente col loro

apprendimento. La seguente esposizione è stata ispirata dallo stile presentato in Levine (2021).



Figura 8.2 Evoluzione delle immagini generate dalle reti generative antagoniste nel tempo, che illustra il miglioramento progressivo nella generazione di immagini. Questa figura è adattata da un tutorial di Ian Goodfellow intitolato *Ian Goodfellow: Adversarial Machine Learning* all'ICLR 2019. L'immagine è tratta da uno screenshot preso al minuto 6:47 della presentazione. Fonte: Ian Goodfellow: Adversarial Machine Learning (ICLR 2019 invited talk), <https://www.youtube.com/watch?v=sucqskXRkss>, ultimo accesso 17 Gennaio 2025.

Addestramento

L'addestramento delle reti generative antagoniste implica la risoluzione di una partita a due giocatori tra il discriminatore e il generatore, come descritto nella formulazione originale di

Goodfellow et al. (2014). L'obiettivo è quello di risolvere il seguente problema di ottimizzazione min-max:

$$\min_{\theta} \max_{\phi} V(\theta, \phi)$$

(8.2)

dove la funzione obiettivo $V(\theta, \phi)$ è data da:

$$V(\theta, \phi) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D_{\phi}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} [\log(1 - D_{\phi}(G_{\theta}(\mathbf{z})))]$$

(8.3)

In questa formulazione, $V(\theta, \phi)$ rappresenta la funzione loss del discriminatore D_{ϕ} , che prende un input e restituisce la probabilità che appartenga al dataset (di osservazioni reali). Questa funzione loss ha la forma di una tipica loss di entropia incrociata (*negative log likelihood*) utilizzata in problemi di classificazione binaria (vedi Murphy [2022]; López de Prado, 2018), dove l'obiettivo, in questo caso particolare, è quello di distinguere i dati reali da quelli falsi. Il discriminatore mira a massimizzare questo obiettivo, mentre il generatore mira a minimizzarlo, creando una dinamica di addestramento da antagonisti. Il discriminatore cerca di massimizzare la log-verosimiglianza dei dati reali (etichette positive) e la log-verosimiglianza dei dati falsi (etichette negative). Quindi, nell'equazione precedente, la prima aspettativa è presa rispetto alla distribuzione dei dati reali, p_{data} , e la seconda aspettativa è presa rispetto alla distribuzione della variabile latente, $p_{\mathbf{z}}$, che viene utilizzata per generare il dataset falso.

Queste aspettative sono approssimate utilizzando stimatori basati su campioni. La funzione obiettivo può essere approssimata da:

$$\begin{aligned}
V(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \sum_{n=1}^N \log D_{\boldsymbol{\phi}}(\mathbf{x}_n) + \sum_{m=1}^M \log(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z}_m))) \\
&= \sum_{n=1}^N \log D_{\boldsymbol{\phi}}(\mathbf{x}_n) + \sum_{m=1}^M \log(1 - D_{\boldsymbol{\phi}}(\mathbf{x}_m))
\end{aligned}
\tag{8.4}$$

dove $\mathbf{x}_m = G_{\boldsymbol{\theta}}(\mathbf{z}_m)$. Nella maggior parte delle applicazioni pratiche, il numero di campioni negativi M viene impostato uguale al numero di campioni positivi, N , rendendo bilanciato il problema di classificazione binaria del discriminatore.

Per risolvere il problema min-max, dobbiamo trovare i gradienti rispetto a $\boldsymbol{\theta}$ e $\boldsymbol{\phi}$ della funzione obiettivo. Per il discriminatore, abbiamo bisogno del gradiente rispetto a $\boldsymbol{\phi}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} V(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\phi}} \left(\sum_{n=1}^N \log D_{\boldsymbol{\phi}}(\mathbf{x}_n) + \sum_{m=1}^M \log(1 - D_{\boldsymbol{\phi}}(\mathbf{x}_m)) \right)
\end{aligned}
\tag{8.5}$$

Per il generatore, il gradiente rispetto a $\boldsymbol{\theta}$ è:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} V(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\theta}} \left(\sum_{n=1}^N \log D_{\boldsymbol{\phi}}(\mathbf{x}_n) + \sum_{m=1}^M \log(1 - D_{\boldsymbol{\phi}}(\mathbf{x}_m)) \right) \\
&= \nabla_{\boldsymbol{\theta}} \left(\sum_{m=1}^M \log(1 - D_{\boldsymbol{\phi}}(\mathbf{x}_m)) \right)
\end{aligned}
\tag{8.6}$$

Una volta calcolati i gradienti, aggiorniamo i parametri utilizzando l'ascesa stocastica del gradiente per il discriminatore e la discesa stocastica del gradiente per il generatore:

$$\begin{aligned}
\boldsymbol{\phi} &\leftarrow \boldsymbol{\phi} + \alpha_1 \nabla_{\boldsymbol{\phi}} V(\boldsymbol{\theta}, \boldsymbol{\phi}) \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha_2 \nabla_{\boldsymbol{\theta}} V(\boldsymbol{\theta}, \boldsymbol{\phi})
\end{aligned}$$

(8.7)

dove α_1 e α_2 sono, rispettivamente, i tassi di apprendimento per il discriminatore e il generatore.

Un aspetto interessante è il calcolo dei gradienti per il generatore. Mentre il calcolo del gradiente del discriminatore è semplice, il gradiente del generatore richiede la retropropagazione (*backpropagation*) attraverso il discriminatore.

Se denotiamo $\ell = \log(1 - D_\phi(G_\theta(\mathbf{z})))$, allora:

$$\begin{aligned}\nabla_\theta V(\theta, \phi) &= \nabla_\theta \left(\sum_{m=1}^M \log(1 - D_\phi(\mathbf{x}_m)) \right) \\ &= \nabla_\theta \left(\sum_{m=1}^M \log(1 - D_\phi(G_\theta(\mathbf{z}_m))) \right) \\ &= \left(\sum_{m=1}^M \nabla_\theta \log(1 - D_\phi(G_\theta(\mathbf{z}_m))) \right) \\ &= \sum_{m=1}^M \nabla_\theta \ell_m \\ &= \sum_{m=1}^M \frac{\partial \mathbf{x}_m}{\partial \theta} \frac{\partial \ell_m}{\partial \mathbf{x}_m}\end{aligned}$$

(8.8)

Ciò indica che stiamo “retropropagando” i gradienti attraverso il discriminatore verso il generatore. In pratica, concateniamo il generatore e il discriminatore come una funzione composta, $D_\phi(G_\theta(\mathbf{z}))$, e il nostro strumento di differenziazione automatica calcolerà l’intero gradiente in modo efficiente.

La procedura di addestramento tipica per le reti generative antagoniste prevede i seguenti passi.

1. Ottenere un dataset vero $\mathcal{D}_t = \{\mathbf{x}^n\}$ ed etichettare ogni osservazione come reale (etichetta positiva).
2. Inizializzare casualmente il Generatore G_θ .

3. Generare un dataset falso, $\mathcal{D}_F = \{\mathbf{x}_m\}$, campionando dal generatore, $\mathbf{z}_m \sim p_z$, $\mathbf{x}_m = G_\theta(\mathbf{z}_m)$, ed etichettare ogni osservazione come falsa (etichetta negativa).
4. Combinare i dataset vero e falso per creare un dataset combinato $\mathcal{D} = \mathcal{D}_t \cup \mathcal{D}_F$.
5. Ottimizzare il discriminatore $D_\phi(\mathbf{x}) = p(\mathbf{y}|\mathbf{x})$ utilizzando il dataset combinato $\mathcal{D}$ con un passo di ascesa stocastica del gradiente.
6. Ottimizzare il generatore D_θ utilizzando il segnale di feedback dal discriminatore con un passo di discesa stocastica del gradiente.
7. Ripetere i passi da 3 a 6 fino alla convergenza.

Aggiornando alternativamente il discriminatore e il generatore, addestriamo la GAN per migliorare la qualità dei dati generati, rendendo sempre più difficile per il discriminatore distinguere tra dati reali e falsi.

Nella Figura 8.3, mostriamo dei campioni generati come illustrato nel lavoro originale di Goodfellow et al. (2014). A quel tempo, la qualità dei campioni era competitiva rispetto ai migliori modelli generativi disponibili, come notato dagli autori. Nella Figura 8.3, la colonna a destra mostra l'esempio di addestramento più vicino al campione vicino, dimostrando che il modello non ha memorizzato il set di addestramento. Questo è illustrato per i seguenti dataset: (a) MNIST, (b) TFD, (c) CIFAR-10 (modello pienamente connesso) e (d) CIFAR-10 (discriminatore convoluzionale e generatore “deconvoluzionale”). Per maggiori dettagli, vedi Goodfellow et al. (2014).



Figura 8.3 Quattro rappresentazioni grafiche: la colonna più a destra mostra l'esempio di addestramento più vicino a ciascun campione generato corrispondente, mostrando che il modello non ha memorizzato il set di addestramento. Questo è illustrato per i seguenti dataset: (a) MNIST, (b) TFD, (c) CIFAR-10 (modello pienamente connesso) e (d) CIFAR-10 (discriminatore convoluzionale e generatore "deconvoluzionale"). Fonte: I. J. Goodfellow et al. (2014).

Valutazione

La valutazione delle reti generative antagoniste è ancora un problema aperto. Poiché non possiamo calcolare esplicitamente la densità, e il punteggio di probabilità del discriminatore viene utilizzato per guidare il generatore a creare dati simili alla distribuzione dei dati, questo punteggio per una data osservazione

non dice molto sulla qualità dei campioni generati. Questo è particolarmente importante quando vogliamo confrontare più modelli. Una soluzione iniziale era utilizzare le stime di log-verosimiglianza basate su finestre di Parzen, note anche come KDE (*Kernel Density Estimation*), come sostituto delle verosimiglianze. Maggiori dettagli sulla KDE possono essere trovati in Bishop (2006).

La Figura 8.4 mostra la log-verosimiglianza basata su finestre di Parzen nel lavoro originale sulle GAN, che permette un confronto quantificabile tra modelli piuttosto che semplicemente un esame visivo delle immagini generate. Sono riportati numeri sui dataset MNIST e TFD per diversi modelli generati.

Modello	MNIST	TFD
DBN (<i>Deep Belief Network</i>) [3]	138 ± 2	1909 ± 66
Stacked CAE (<i>Convolutional Autoencoder</i>) [3]	121 ± 1.6	2110 ± 50
Deep GSN (<i>Generative Stochastic Network</i>) [6]	214 ± 1.1	1890 ± 29
Reti antagoniste	225 ± 2	2057 ± 26

Figura 8.4 La Tabella 1 da Goodfellow et al. (2014): stime di log-verosimiglianza basate su finestre di Parzen. I valori riportati per MNIST rappresentano la log-verosimiglianza media dei campioni del set di test, dove l'errore standard della media viene calcolato tra i singoli esempi.

Nella letteratura sono state proposte ulteriori metriche di valutazione per tenere meglio conto della copertura dei campioni generati. Una di queste è l'*Inception Score*, vedi Salimans et al. (2016), che utilizza la rete neurale Inception per classificare le immagini e valutare la qualità della generazione condizionale con le reti generative antagoniste. Fra i miglioramenti rispetto all'*Inception Score* vi è la *Frechet Inception Distance*, che punta a migliorare la misurazione della diversità incorporando statistiche delle immagini

in uno spazio di embedding differente; maggiori dettagli possono essere trovati in Heusel et al. (2018).

Alcuni approfondimenti teorici sulle reti generative antagoniste

Abbiamo già visto che l'addestramento delle reti generative antagoniste (GNN) richiede la risoluzione di una partita min-max a due giocatori tra il generatore e il discriminatore. Dalla funzione obiettivo, il compito di ciascun modello è chiaro: il discriminatore massimizza la verosimiglianza dei dati, mentre il generatore cerca di ingannare il discriminatore minimizzando la stessa funzione obiettivo. Tuttavia, poiché siamo interessati alla qualità dei campioni generati, possiamo porci la seguente domanda: le GAN addestrate con la procedura sopra menzionata stanno cercando di ottimizzare una qualche forma di divergenza tra la distribuzione dei dati p_{data} e la distribuzione del generatore p_g ?

Per rispondere a questa domanda, svolgiamo un esercizio teorico per risolvere il problema del discriminatore e del generatore ottimali e per vedere se ci dice qualcosa. Sebbene non addestriamo le GAN utilizzando la procedura esatta che verrà descritta qui, questo esercizio fornisce conoscenze preziose che sono state utilizzate nella pratica per migliorare l'addestramento e le prestazioni delle GAN.

Risolviamo per il classificatore ottimale. Data la nostra funzione obiettivo:

$$\begin{aligned}
& \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} V(\boldsymbol{\theta}, \boldsymbol{\phi}) \\
& \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D_{\boldsymbol{\phi}}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z} [\log(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})))]
\end{aligned}
\tag{8.9}$$

Che cosa possiamo dire su G alla convergenza? Vediamo se possiamo trovare una soluzione in forma chiusa per D . Possiamo riscrivere l'Equazione 8.9 come segue:

$$\begin{aligned}
& \min_G \max_D \mathbb{E}_{\mathbf{x} \sim p} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z} [\log(1 - D(G(\mathbf{z})))] \\
& \min_G \max_D \mathbb{E}_{\mathbf{x} \sim p} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim q} [\log(1 - D(\mathbf{x}))]
\end{aligned}
\tag{8.10}$$

Qui, semplifichiamo la notazione e ottimizziamo sulle funzioni D e G invece di ottimizzare sui parametri $\boldsymbol{\theta}$ e $\boldsymbol{\phi}$. Sostituiamo anche p_{data} con p , la distribuzione del dataset reale, e lasciamo che q rappresenti la distribuzione utilizzata per generare il nostro dataset falso, con $D(\mathbf{x}) = D(G(\mathbf{z}))$.

Risolviamo per il discriminatore ottimale D :

$$D^* = \arg \max_G \mathbb{E}_{\mathbf{x} \sim p} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim q} [\log(1 - D(\mathbf{x}))]
\tag{8.11}$$

Per trovare il valore ottimale di D , calcoliamo il gradiente rispetto a D e lo poniamo uguale a zero:

$$\begin{aligned}\nabla_D &= \mathbb{E}_{\mathbf{x} \sim p} [\nabla_D \log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim q} [\nabla_D \log(1 - D(\mathbf{x}))] \\ &= \mathbb{E}_{\mathbf{x} \sim p} \left[\frac{1}{D(\mathbf{x})} \right] - \mathbb{E}_{\mathbf{x} \sim q} \left[\frac{1}{1 - D(\mathbf{x})} \right]\end{aligned}$$

(8.12)

Ponendo $\nabla_d = 0$ e risolvendo per D , troviamo che il valore ottimale è:

$$\begin{aligned}D^* &= \frac{p(x)}{p(x) + q(x)} \\ D^* &= \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}\end{aligned}$$

(8.13)

dove q è sostituito da p_g , per indicare che le etichette negative sono generate dalla distribuzione del generatore, p_g .

Ora, se nella funzione obiettivo inseriamo il valore ottimale di D , possiamo determinare che cosa sta cercando di ottimizzare il generatore G :

$$\begin{aligned}V(G, D^*) &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_G} [\log(1 - D(\mathbf{x}))] \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)} \right] + \mathbb{E}_{\mathbf{z} \sim p_G} \left[\log \frac{p_G(x)}{p_{\text{data}}(x) + p_G(x)} \right] \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] + \\ &\quad \mathbb{E}_{\mathbf{x} \sim p_G} [\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]\end{aligned}$$

(8.14)

Sia $q(\mathbf{x}) = (p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})) / 2$. Allora

$$\begin{aligned}
V(G, D^*) &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log p_{\text{data}}(\mathbf{x}) - \log q(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim p_G} [\log p_G(\mathbf{x}) - \log q(\mathbf{x})] - \log 4 \\
&= D_{\text{KL}}(p_{\text{data}} \| p_G) + D_{\text{KL}}(p_G \| p_{\text{data}}) - \log 4 \\
&= 2D_{\text{JS}}(p_{\text{data}} \| p_G) - \log 4
\end{aligned}$$

(8.15)

dove D_{JS} è la divergenza di Jensen-Shannon.

Pertanto, alla convergenza, il generatore minimizza un'idea di distanza tra la distribuzione dei dati p_{data} e la distribuzione del generatore p_g . Questa conoscenza ha portato a sviluppare nuovi approcci che affrontano sia gli svantaggi sia i vantaggi della minimizzazione di questa divergenza, proponendo alternative migliori per l'addestramento delle reti generative antagoniste.

Perché l'addestramento delle reti generative antagoniste è così difficile? Tecniche per migliorarlo

Addestrare le reti generative antagoniste è impegnativo. Implementare le GAN come descritto nel lavoro originale di Goodfellow et al. (2014) richiede un enorme lavoro di ottimizzazione degli iperparametri per raggiungere la convergenza e generare campioni di alta qualità.

Per esempio, l'articolo di Salimans et al. (2016) illustra l'ampia ottimizzazione degli iperparametri richiesta e alcune delle tecniche utilizzate per far funzionare efficacemente le GAN. Queste tecniche includono la discriminazione mini-batch, il *feature matching*, la media storica e altre, applicabili al generatore, al discriminatore o a entrambi. Per maggiori dettagli, vedi Salimans et al. (2016).

Una ragione per cui l'addestramento delle GAN è difficile può essere compresa esaminando un semplice esempio, illustrato nella Figura 8.5. Per semplicità, generiamo campioni sia dalla distribuzione dei dati sia dal generatore, sovrapponendo le corrispondenti funzioni di densità e supponendo che questi siano i dati iniziali per l'addestramento. All'inizializzazione, con un generatore casuale, le densità e i campioni generati da p_{data} e p_g sono distanti. Se addestriamo il discriminatore con il dataset combinato $\mathcal{D} = \mathcal{D}_t \cup \mathcal{D}_F$ utilizzando i dati veri $\mathcal{D}_t$ e i dati falsi $\mathcal{D}_F$ da questi campioni, il discriminatore può facilmente ottenere una classificazione perfetta. Ciò si traduce in un confine decisionale molto netto, saturato a 1 per i campioni da p_{data} e a 0 per i campioni generati da p_g .

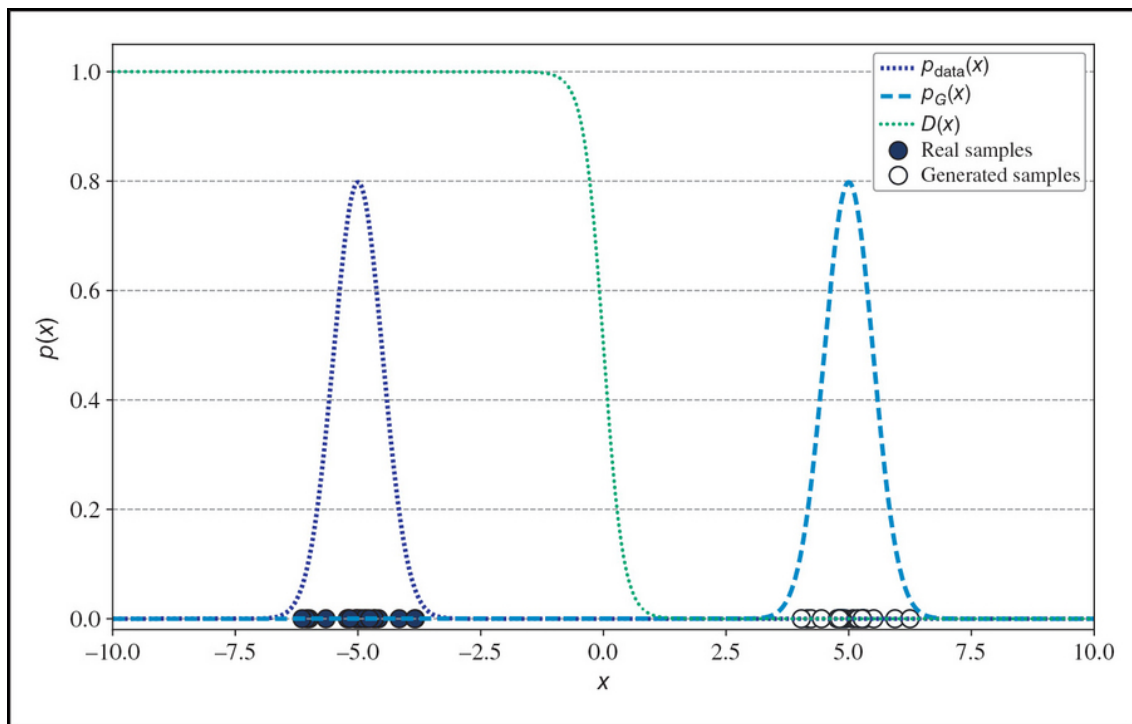


Figura 8.5 Rappresentazione grafica di una delle ragioni per cui l'addestramento delle reti generative antagoniste può essere difficile. In questa illustrazione concettuale, all'inizializzazione, la distribuzione dei dati e quella

del generatore sono distanti, permettendo al discriminatore di ottenere una classificazione perfetta. Ciò produce un confine decisionale netto, dove il discriminatore restituisce 1 per i campioni reali e 0 per quelli generati, rendendo difficile per il generatore ricevere aggiornamenti di gradiente significativi.

Poiché il generatore impara retropropagando i gradienti completi del discriminatore, in situazioni in cui il discriminatore è saturo, fornisce al generatore segnali di gradiente molto scadenti per quasi tutti i punti generati da p_g . Ciò rende quasi impossibile per il generatore migliorare la qualità dei campioni generati. In breve, una delle ragioni principali per cui l'addestramento delle GAN è difficile è che il discriminatore potrebbe fornire un segnale di gradiente molto debole, se la distribuzione dei dati e la distribuzione del generatore sono molto distanti.

Per migliorare questa situazione in cui p_{data} e p_g sono molto distanti, puntiamo ad avere un confine decisionale più liscio per il discriminatore. Ciò fornirebbe segnali significativi per guidare l'addestramento del generatore e migliorare la qualità dei campioni generati.

Le soluzioni a questo punto includono modificare esplicitamente il discriminatore per avere un confine decisionale più sfumato o far sì che i campioni da p_{data} e p_g si sovrappongano, rendendo più difficile per il discriminatore classificarli. L'*instance noise*, come descritto da Mescheder et al. (2018), è un metodo in cui sia ai campioni reali sia a quelli falsi viene aggiunto del rumore, in modo da sovrapporre le loro distribuzioni, complicando il compito di classificazione per il discriminatore e ottenendo così un confine decisionale più sfumato.

Le tecniche per modificare esplicitamente il confine decisionale includono le LSGAN (*Least Squares Generative Adversarial Networks*), descritte da Mao et al. (2017), dove il discriminatore

restituisce un numero reale illimitato invece di una probabilità, un numero reale tra 0 e 1. Altre soluzioni implicano l'uso di diverse funzioni obiettivo per forzare il discriminatore ad avere un confine decisionale più sfumato. La soluzione principale proposta con le WGAN (*Wasserstein GAN*), come descritto in Arjovsky et al. (2017), vincola il discriminatore a essere Lipschitz-continuo. Ulteriori soluzioni per rendere il confine decisionale ancora più sfumato includono le WGAN-GP (*Wasserstein GAN with Gradient Penalty*) di Gulrajani et al. (2017) e SNGAN (*Spectral Normalization GAN*) di Miyato et al. (2018).

Per un'analisi dettagliata della convergenza e delle diverse tecniche di addestramento per le GAN, si rimanda all'articolo di Mescheder et al. (2018).

Wasserstein GAN

Ora sappiamo che il generatore sta cercando di minimizzare la divergenza di Jensen-Shannon (JS) tra la distribuzione dei dati e la distribuzione del generatore. Tuttavia, la divergenza JS non è efficace nel catturare la distanza tra distribuzioni quando queste sono molto distanti o hanno solo poca o addirittura nessuna sovrapposizione.

Per illustrare ciò, consideriamo la Figura 8.6. Nella figura in alto, le distribuzioni p_{data} e p_g sono molto distanti con sovrapposizione zero. Nella figura in basso, p_{data} e p_g sono più vicine ma ancora non si sovrappongono. Idealmente, vorremmo spingere la distribuzione del generatore p_g più vicino a p_{data} , e tra queste due opzioni, preferiremmo il generatore della figura in basso. Tuttavia, la divergenza JS assegna lo stesso punteggio a entrambe le situazioni, poiché non c'è sovrapposizione tra le due distribuzioni, il che rappresenta un problema per l'addestramento efficace del generatore.

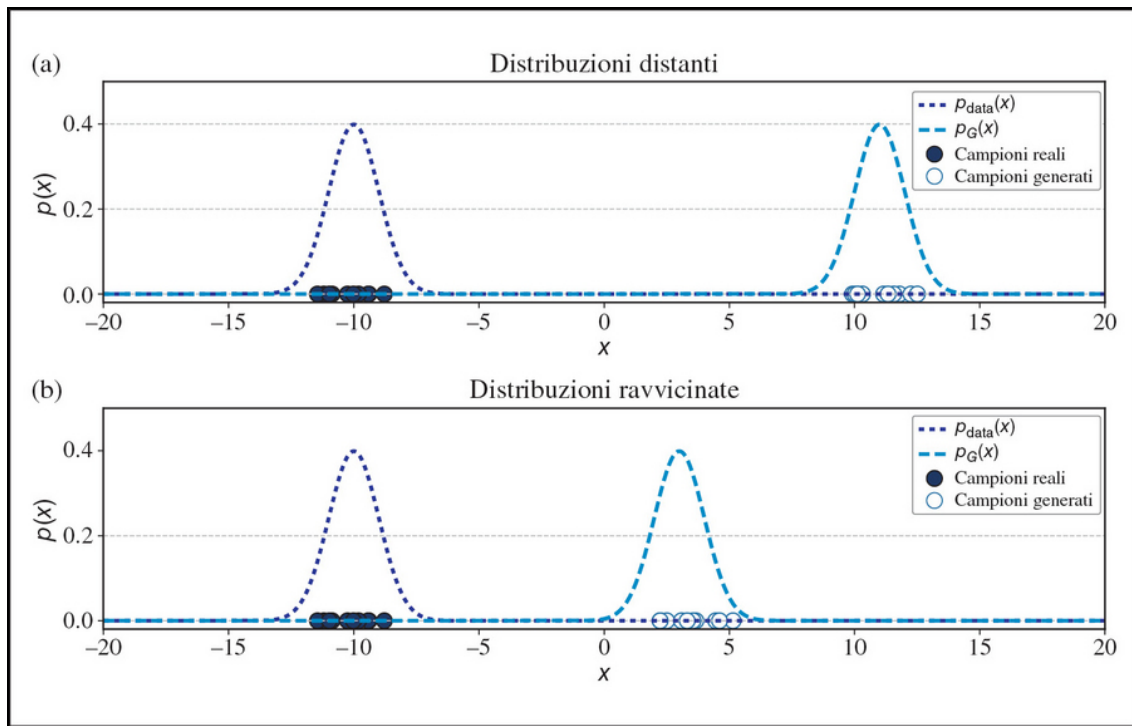


Figura 8.6 Rappresentazione grafica di due casi diversi per la distribuzione dei dati e la distribuzione del generatore. Nella Figura (a), le distribuzioni sono distanti, mentre nella Figura (b) sono più vicine. Nonostante questa differenza, la divergenza JS assegna lo stesso punteggio a entrambi i casi.

Un miglioramento a questa situazione potrebbe essere l'uso di una diversa misura di divergenza, che quantifichi effettivamente la distanza tra due distribuzioni. Un candidato è la distanza di Wasserstein, che viene utilizzata per ottimizzare le reti generative antagoniste come proposto da Arjovsky et al. (2017).

La distanza di Wasserstein è definita come:

$$W(p_{\text{data}}, p_G) = \inf_{\gamma} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma} \|\mathbf{x} - \mathbf{y}\|$$

(8.16)

dove γ è la distribuzione su $\mathbf{x}$ e $\mathbf{y}$ che soddisfa le marginali $\gamma_{\mathbf{x}} = p_{\text{data}}$ e $\gamma_{\mathbf{y}} = p_G$.

Ottimizzare direttamente questa formulazione di $W(p_{\text{data}}, p_G)$ è complicato, perché trovare il valore ottimale di γ non è un compito facile. Tuttavia, esiste una formulazione nota come dualità di Kantorovich-Rubinstein, che rende il problema più trattabile:

$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} f(\mathbf{x}) - \mathbb{E}_{\mathbf{x} \sim p_G} f(\mathbf{x})$$

(8.17)

Questo è uguale all'estremo superiore sull'insieme di tutte le possibili funzioni 1-Lipschitz f , del valore atteso di f sotto p_{data} meno il valore atteso di f sotto p_G . L'insieme di tutte le possibili funzioni scalari 1-Lipschitz f soddisfa:

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{y}\|$$

(8.18)

il che significa, in parole povere, che f dovrebbe avere un gradiente o una pendenza limitati.

Tornando al nostro esempio, se f_{ϕ} è la nostra rete neurale discriminatrice, parametrizzata da ϕ , come possiamo vincolare la rete a essere 1-Lipschitz o, analogamente, ad avere un gradiente limitato? La soluzione originale proposta da Arjovsky et al. (2017) implica il taglio dei pesi della rete (*weight clipping*), per garantire che il gradiente sia limitato. Tuttavia, gli autori dell'articolo hanno chiaramente menzionato che questo probabilmente non è il modo migliore per imporre questo vincolo e hanno illustrato molti problemi

che genera. In effetti, al momento della stesura di questo testo, il *gradient clipping* per ottenere questo vincolo è per lo più obsoleto. Altri lavori si basano su questa idea per imporre una rete neurale Lipschitz-continua, come WGAN-GP (*Wasserstein GAN with Gradient Penalty*) di Gulrajani et al. (2017) e SNGAN (*Spectral Normalization GAN*) di Miyato et al. (2018).

Una volta vincolato lo spazio di ricerca delle funzioni all'insieme delle funzioni 1-Lipschitz, possiamo utilizzare l'obiettivo basato su campioni per trovare il discriminatore e il generatore ottimali:

$$\begin{aligned} W(p_{\text{data}}, p_G) &= \sup_{\|f_\phi\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_G} [f_\phi(\mathbf{x})] \\ &= \sup_{\|f_\phi\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_z} [f_\phi(G_\theta(\mathbf{z}))] \end{aligned} \quad (8.19)$$

Approssimiamo $W(p_{\text{data}}, p_g)$ utilizzando stimatori basati su campioni:

$$\begin{aligned} W(p_{\text{data}}, p_G) &= \sup_{\|f_\phi\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_G} [f_\phi(\mathbf{x})] \\ &= \sup_{\|f_\phi\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_z} [f_\phi(G_\theta(\mathbf{z}))] \end{aligned} \quad (8.20)$$

Possiamo quindi ottimizzare calcolando i gradienti rispetto a ϕ e θ .

Lo pseudocodice per l'addestramento delle WGAN è il seguente (adattato da Levine (2021)).

1. Ottieni un dataset vero $\mathcal{D}_t = \{\mathbf{x}^n\}$ ed etichetta ogni osservazione come reale (etichetta positiva).
2. Inizializza il generatore G_θ in modo casuale.

3. Genera un dataset falso, $\mathcal{D}_F = \{x_m\}$, *campionando dal generatore*, $\mathbf{z}_m \sim p_z$, $x_m = G_\theta(\mathbf{z}_m)$ ed etichetta ogni osservazione come falsa (etichetta negativa).
4. Combina i dataset vero e falso per creare un dataset combinato $\mathcal{D} = \mathcal{D}_t \cup \mathcal{D}_F$.
5. Aggiorna il discriminatore f_ϕ per massimizzare $\hat{W}(p_{\text{data}}, p_G)$. Calcola $\nabla_\phi \hat{W}(p_{\text{data}}, p_G)$ e utilizza l'ascesa stocastica del gradiente.
6. Taglia tutte le matrici dei pesi in ϕ in modo che ogni livello della matrice dei pesi, denotato da ϕ_p , sia all'interno di $[-c, c]$ per una costante c .
7. Aggiorna il generatore G_θ per minimizzare $\hat{W}(p_{\text{data}}, p_G)$. Calcola i gradienti
$$\nabla_\theta \hat{W}(p_{\text{data}}, p_G) = \nabla_\theta \sum_{m=1}^M f_\phi(G_\theta(\mathbf{z}_m))$$
e utilizza la discesa stocastica del gradiente.
8. Ripeti i passaggi da 3 a 7 fino alla convergenza.

Nella Figura 8.7, vediamo i risultati della Figura 2 dell'articolo di Arjovsky et al. (2017). Questa figura illustra l'effetto della distanza di Wasserstein sull'ottimizzazione delle GAN, evidenziando in particolare il confine decisionale ottenuto dal discriminatore. Nella Figura 8.7, il confine decisionale del discriminatore per le tipiche GAN è la linea tratteggiata "GAN Discriminator" (quasi verticale, al centro del grafico), ottimizzato utilizzando il gioco min-max a due. Il confine decisionale per la WGAN, la linea tratteggiata "WGAN Critic" (che attraversa diagonalmente il grafico), mostra un confine molto più sfumato, fornendo migliori segnali di gradiente per il generatore e migliorando il processo di addestramento.

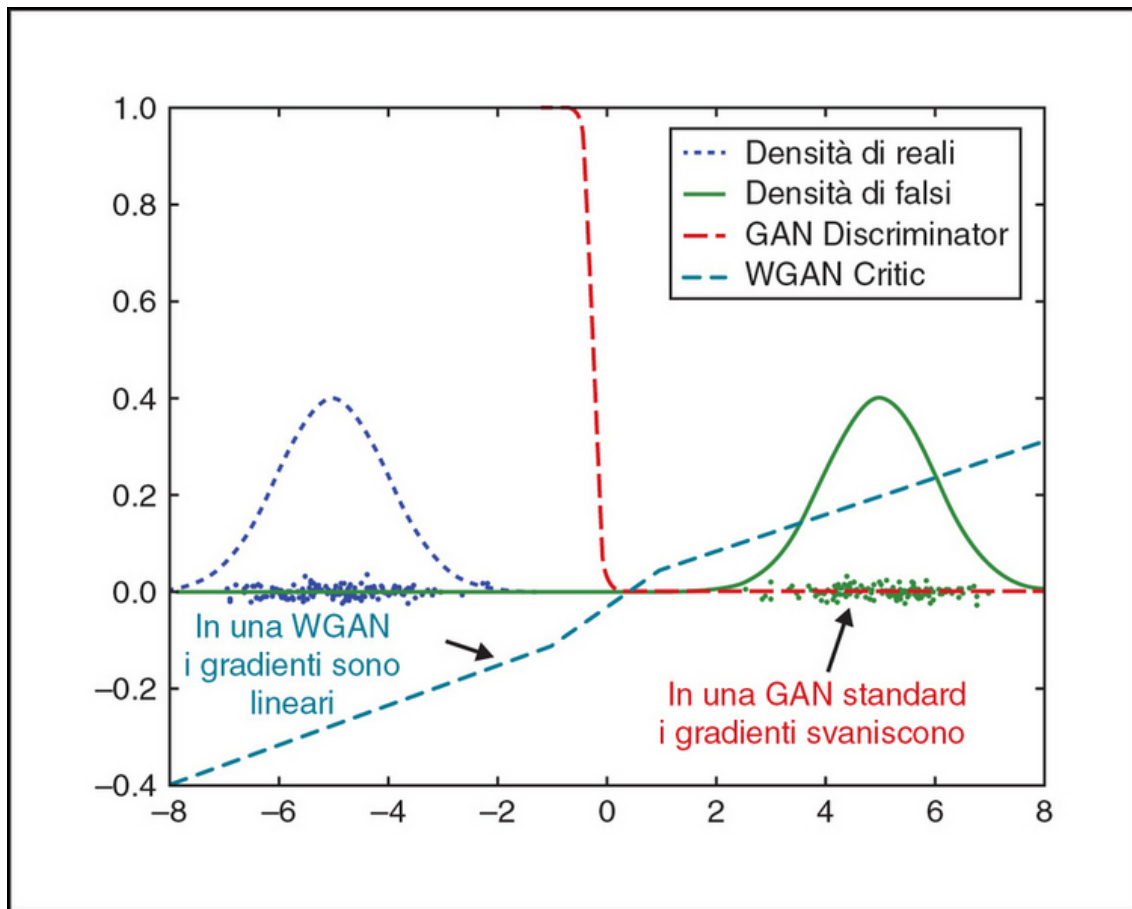


Figura 8.7 Grafico del discriminatore ottimale e del critico quando si distinguono due gaussiane. Il discriminatore GAN minimax va in saturazione, portando a gradienti che svaniscono, mentre il critico WGAN fornisce gradienti più sfumati, migliorando l'addestramento del generatore. Figura 2 da Arjovsky, Chintala e Bottou (2017).

Wasserstein GAN with Gradient Penalty

Come accennato, un altro modo per imporre vincoli di Lipschitz in una rete neurale consiste nel vincolare direttamente il gradiente, che è l'approccio proposto dalle WGAN-GP (*Wasserstein GAN with Gradient Penalty*) da Gulrajani et al. (2017).

Nel loro lavoro, hanno modificato l'obiettivo originale di una WGAN introducendo una penalità del gradiente per la continuità di Lipschitz, data da:

$$W(p_{\text{data}}, p_G) = \sup_{\|f_{\phi}\|_L \leq 1} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [f_{\phi}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_G} [f_{\phi}(\mathbf{x})] - \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}} \left[\left(\|\nabla_{\hat{\mathbf{x}}} f_{\phi}(\hat{\mathbf{x}})\|_2 - 1 \right)^2 \right] \quad (8.21)$$

dove la penalità del gradiente è l'ultimo termine dell'equazione precedente. Qui, la funzione obiettivo penalizza direttamente il discriminatore per avere gradienti con norme diverse da uno, utilizzando il parametro di regolarizzazione Λ . L'aspettativa viene calcolata sotto la distribuzione $p_{\hat{\mathbf{x}}}$, per la quale i campioni vengono generati campionando uniformemente lungo linee rette tra coppie di punti campionati dalla distribuzione dei dati p_{data} e dalla distribuzione del generatore p_G . Per maggiori dettagli, vedi Gulrajani et al. (2017). Nella pratica, le WGAN-GP migliorano la stabilità dell'addestramento delle WGAN ed sono molto facili da implementare.

Un'analisi dettagliata della convergenza di diversi algoritmi di addestramento GAN può essere trovata in Mescheder et al. (2018).

Estendere le reti generative antagoniste per le serie temporali

Allo stesso modo, come abbiamo visto nei capitoli precedenti con gli encoder variazionali e i modelli di flusso, questi, per il loro design strutturale, ipotizzano l'indipendenza tra le osservazioni consecutive. Sorge quindi spontanea la domanda: come possiamo adattarli, per

catturare dinamiche temporali che potrebbero essere importanti a determinate scale temporali?

Una potenziale risposta è simile al modo in cui abbiamo proceduto nei capitoli precedenti. Potremmo provare a combinare le reti generative antagoniste con modelli che eccellono nel gestire il tipo di dinamiche che vogliamo catturare, come i modelli autoregressivi (reti neurali ricorrenti o transformer).

Consideriamo due casi interessanti per la generazione di serie temporali.

- RCGAN (*Recurrent Conditional GAN*), vedi Esteban et al. (2017): utilizza una combinazione di RNN e GAN. Utilizza una rete neurale ricorrente per il generatore e i discriminatori. A causa di questa scelta progettuale, i dati vengono generati sequenzialmente. Maggiori dettagli sull'architettura del modello e sui risultati sperimentali si trovano nel notebook di questo capitolo [RCGAN Notebook].
- TimeGAN (*Time-series GAN*), vedi Yoon et al. (2019): utilizza un generatore di sequenze e un discriminatore per modellare le dinamiche delle serie temporali. Tuttavia, lo fa modellando le dipendenze temporali nello spazio latente. Ciò viene ottenuto combinando gli obiettivi dell'apprendimento supervisionato con gli obiettivi delle reti generative antagoniste in un problema di ottimizzazione congiunto. L'obiettivo dell'apprendimento supervisionato spinge la rete a catturare le dinamiche nello spazio latente. L'obiettivo dell'antagonismo, come nelle GAN standard, spinge la rete a generare campioni realistici. Maggiori dettagli sull'architettura del modello e sui risultati sperimentali sono disponibili nel notebook [TimeGAN Notebook] di questo capitolo.

Conclusioni

In questo capitolo, abbiamo esplorato un altro potente strumento per modellare dati a elevata dimensionalità: le reti generative antagoniste. A differenza dei modelli studiati nei capitoli precedenti, le GAN appartengono alla classe dei modelli generativi impliciti, il che significa che non richiedono una stima esplicita della densità. Si basano invece su un processo di apprendimento “ad avversari” in cui due modelli – il generatore e il discriminatore – vengono addestrati simultaneamente in un framework. A differenza dei modelli di flusso, il generatore di una GAN può essere una qualsiasi rete neurale, senza vincoli di invertibilità o di corrispondenza della dimensionalità tra la variabile latente e le osservazioni. Questa flessibilità consente alle GAN di apprendere rappresentazioni complesse e di generare campioni di alta qualità. Abbiamo anche affrontato le principali problematiche nell’addestramento delle GAN ed esplorato strategie per migliorarne la stabilità e le prestazioni.

Inoltre, abbiamo esaminato come le GAN possano essere estese per modellare i dati sequenziali, in particolare le serie temporali. Integrando le GAN con architetture specializzate per catturare le dipendenze temporali – come le reti neurali ricorrenti – possiamo generare dati sequenziali con dinamiche dipendenti dal tempo. Nei notebook associati a questo capitolo, troverete esempi pratici sia delle GAN tradizionali sia delle GAN adattate per le serie temporali, applicate alla generazione di dati finanziari.

Con questo capitolo, avete aggiunto un altro potente strumento al vostro toolbox di modellazione generativa. Le GAN forniscono un approccio flessibile ed efficace per generare dati a elevata dimensionalità e multimodali.

Capitolo 9

Sfruttare gli LLM per l'analisi del sentiment nel trading

In questo capitolo, ci spostiamo rispetto alle nostre precedenti discussioni, in cui ci siamo concentrati sullo studio di diverse famiglie di modelli generativi in isolamento, per esplorare e costruire un'applicazione finanziaria pratica che combini questi modelli. L'obiettivo è quello di creare un sistema che prenda in input dei dati vocali e restituisca in output dei segnali di trading. Questo sistema sfrutta modelli potenti, come i modelli transformer, per attività sequenza-per-sequenza per convertire segnali audio in testo, e i modelli transformer degli LLM (*Large Language Model*) per assegnare punteggi di sentiment al testo. Applicheremo queste tecniche per analizzare il sentiment dei discorsi alle conferenze stampa della Federal Reserve e per generare segnali di trading potenzialmente redditizi.

Analisi del sentiment nei discorsi alle conferenze stampa della Fed utilizzando Large Language Model

È possibile applicare l'analisi del sentiment sui discorsi alle conferenze stampa della Federal Reserve, calcolata utilizzando gli LLM, per generare segnali di trading predittivi?

Immaginiamo uno scenario: stiamo guardando il presidente della Federal Reserve americana, Jerome Powell, tenere una conferenza stampa il 2 novembre 2022, in cui discute l'ultimo aumento dei tassi della Fed e comunica le prospettive economiche attuali, come mostra la Figura 9.1 (WGN News, 2022).



Figura 9.1 Frame video: il presidente della Fed Powell discute l'ultimo aumento dei tassi. Fonte: WGN News (2022).

Come trader, potremmo chiederci come utilizzare le informazioni esposte nel suo discorso per prendere decisioni informate di trading. Se foste George Soros, noto per le sue eccezionali capacità di trading e la sua notevole track record nel macro trading, potreste essere in grado di filtrare dal discorso le informazioni rilevanti, convertendole in intuizioni che potreste combinare con altri dati, come i dati di mercato, per prevedere facilmente le prossime mosse del mercato.

Tuttavia, per quelli di noi che non hanno l'intuito di Soros, potrebbe essere utile un approccio più quantitativo. Questo approccio richiede di convertire i segmenti del video in valori numerici, dai quali generare segnali di acquisto o vendita. Il nostro approccio è stato ispirato da rapporti sull'analisi dei discorsi della Fed da parte dell'AI di JPMorgan per segnali di trading da *Bloomberg News* (2023) e *Fortune* (2023), così come da un tutorial di *OpenAI Whisper* sulla trascrizione dei discorsi della Fed di Part Time Larry (2022).

Ci sono diversi modi per progettare un sistema con l'obiettivo di tradurre i contenuti video in segnali di trading applicabili. Qui, tentiamo di fornire una soluzione basata su un approccio a cascata, che coinvolge una serie di sottosistemi per l'elaborazione di diversi tipi di dati. L'intero sistema prende come input un segmento del video della conferenza stampa della Fed e restituisce un valore numerico che rappresenta il livello di sentiment di quel segmento. La progettazione di questi sottosistemi coinvolgerà quanto segue.

1. Estrarre il parlato dal video della conferenza stampa della Fed.
2. Convertire i segmenti di parlato in testo.
3. Convertire il testo in valori numerici (punteggi di sentiment) che possono generare segnali rilevanti.

Lo schema a blocchi del sistema che illustra questo processo si trova nella Figura 9.2.



Figura 9.2 Schema a blocchi del sistema.

Per esplorare questo sistema, faremo un passo indietro e simuleremo il prendere decisioni basate sulla conferenza stampa e sui dati di mercato correlati a nostra disposizione.

Ipotizziamo di voler progettare una strategia di trading per negoziare l'SPY in intervalli di 30 secondi. Nella Figura 9.3, illustriamo la serie dei prezzi dell'SPY durante la conferenza stampa della Fed del 2 novembre 2022, dalle 14:30 alle 15:30 in intervalli di 30 secondi. Idealmente, vorremmo trovare una qualche forma di correlazione tra ciò che Jerome Powell sta dicendo in quel particolare momento o fino a quel momento e i successivi movimenti di prezzo dell'SPY.

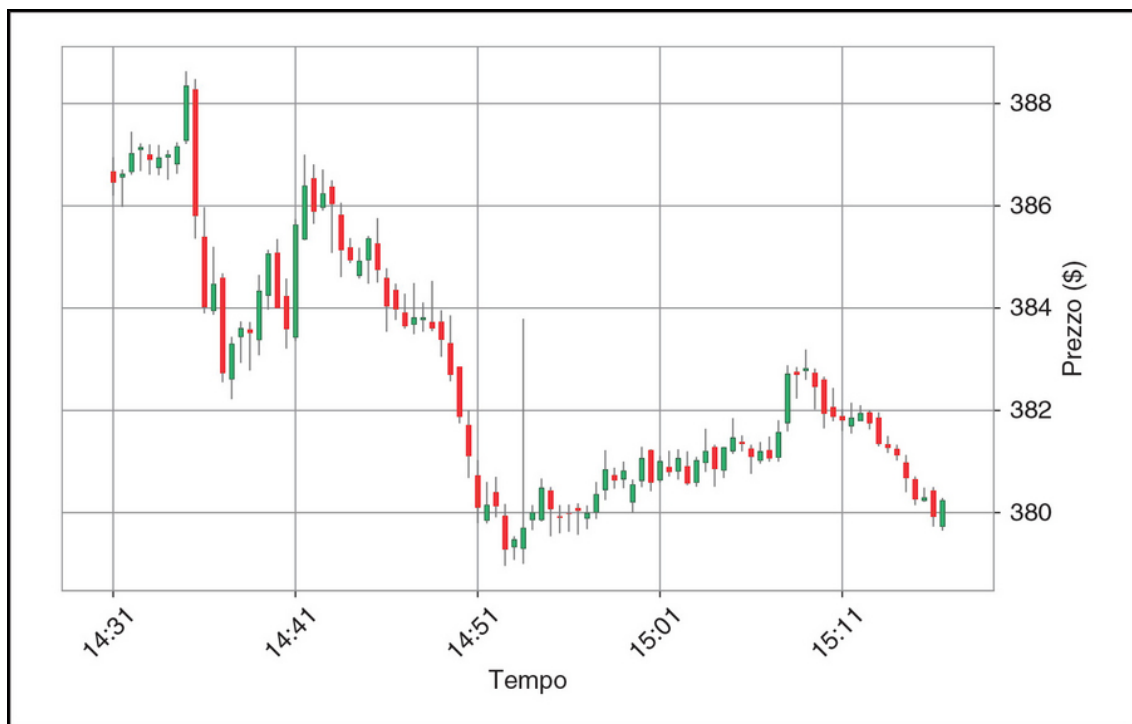


Figura 9.3 Grafico della serie dei prezzi dell'SPY durante la conferenza stampa della Fed.

Per ogni barra di trading di 30 secondi dei dati dell'SPY, faremmo quanto segue.

1. Estrarre l'audio dal video fino a quella specifica barra.
2. Eseguire la conversione da parlato a testo.
3. Eseguire l'analisi del sentiment sulla base del testo.
4. Generare segnali per prendere decisioni di acquisto e vendita.

Questo processo arricchirà la serie dei prezzi OHLC (*Open-High-Low-Close*) dell'SPY con i punteggi di sentiment del discorso di Powell durante ogni intervallo di 30 secondi, aiutando a quantificare il potere predittivo del segnale. Il formato desiderato dei dati tabellari è illustrato nella Figura 9.4.

data_with_sentiment.head(10)									
timestamp	text	video_id	sentiment_score	cst_timestamp	open	high	low	close	return
2022-06-15 14:31:00-04:00	Good afternoon. I will begin with	1	0.116885	2022-06-15 13:31:00-05:00	374.40	374.57	373.65	373.94	0.000053
2022-06-15 14:31:30-04:00	Good afternoon. I will begin with one overarc...	1	0.360220	2022-06-15 13:31:30-05:00	373.80	374.15	373.47	373.96	-0.001685
2022-06-15 14:32:00-04:00	one overarching message. We at the Fed unders...	1	0.313680	2022-06-15 13:32:00-05:00	374.05	374.07	373.00	373.33	-0.001848
2022-06-15 14:32:30-04:00	if we were to have a sustained period of stro...	1	0.115033	2022-06-15 13:32:30-05:00	373.18	373.30	372.55	372.64	-0.000510
2022-06-15 14:33:00-04:00	rate will be appropriate. In addition, we are...	1	0.034392	2022-06-15 13:33:00-05:00	372.49	372.87	372.12	372.45	0.002121
2022-06-15 14:33:30-04:00	with consumption spending remaining strong. I...	1	-0.311607	2022-06-15 13:33:30-05:00	372.41	373.50	372.30	373.24	0.000831
2022-06-15 14:34:00-04:00	have marked down their projections for econom...	1	-0.186132	2022-06-15 13:34:00-05:00	373.06	373.61	372.90	373.55	0.000428
2022-06-15 14:34:30-04:00	robust. Improvements in labor market conditio...	1	0.522128	2022-06-15 13:34:30-05:00	373.59	374.35	373.45	373.71	-0.000722
2022-06-15 14:35:00-04:00	on wages and prices. The median projection in...	1	0.380036	2022-06-15 13:35:00-05:00	373.69	374.01	373.25	373.44	0.001446
2022-06-15 14:35:30-04:00	or prices rose 4.9%. In May, the 12-month cha...	1	-0.127001	2022-06-15 13:35:30-05:00	373.60	374.04	373.34	373.98	0.001230

Figura 9.4 Tabella della serie dei prezzi arricchita.

Con questi dati arricchiti, possiamo scoprire se esiste una relazione statistica tra i punteggi di sentiment calcolati e i rendimenti futuri. Nella Figura 9.5, illustriamo un grafico a dispersione per alcuni video di conferenze stampa della Fed che si sono tenute nel 2022 e nel 2023, dove l'asse x rappresenta i punteggi di sentiment e l'asse y rappresenta i rendimenti forward dell'SPY nei successivi 30 secondi.

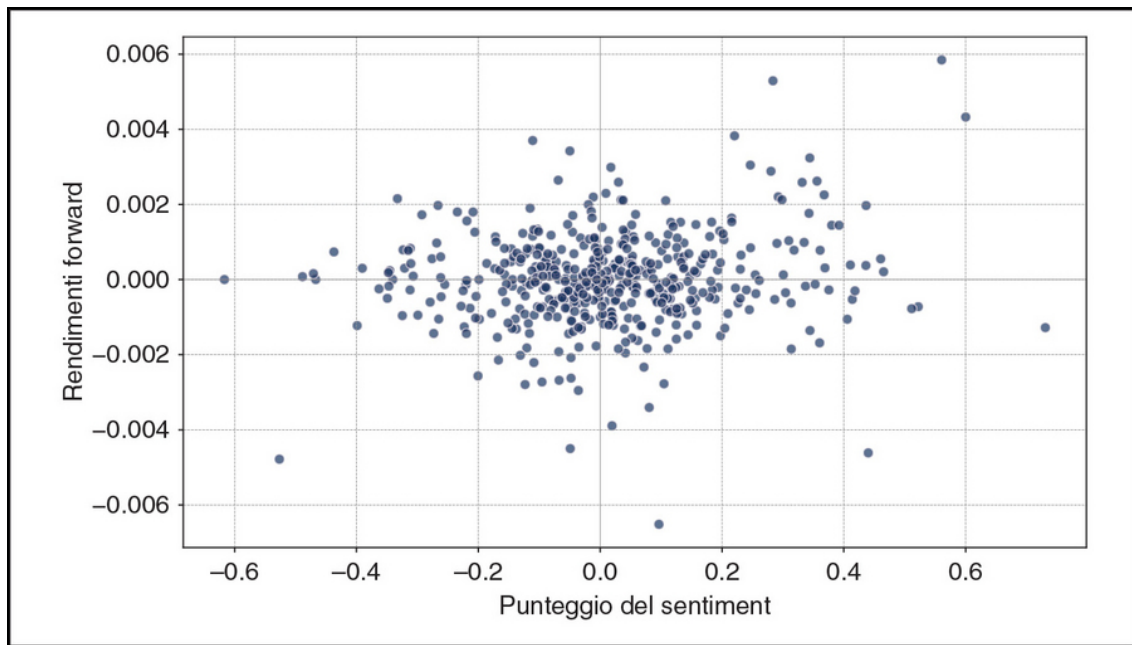


Figura 9.5 Grafico a dispersione del segnale di sentiment vs rendimenti forward.

Nel resto del capitolo, descriveremo più in dettaglio i componenti principali del sistema, come funzionano e come metterli insieme. Per esperienza, utilizzando un campione limitato di video, abbiamo trovato una correlazione di Pearson significativa del 14.14% tra il segnale di sentiment e il rendimento forward dell'SPY, con un p -value dello 0.199%.

Dati: video e prezzi di mercato

In questo caso d'uso, utilizziamo dati video, estraiamo l'audio corrispondente, lo trasformiamo in testo, convertiamo il testo in valori numerici e infine uniamo questi valori numerici ai dati dei prezzi di mercato per la costruzione e valutazione del modello. Questo approccio sfrutta le conoscenze uniche offerte da fonti di dati

alternative per migliorare le strategie di trading e le previsioni di mercato.

Come notato da López de Prado (2018): “I dati alternativi offrono l’opportunità di lavorare con dataset veramente unici e difficili da elaborare. Ricorda: i dati che sono difficili da archiviare, manipolare e utilizzare sono sempre i più promettenti. Riconoscerai che un dataset può essere utile se infastidisce il tuo team che si occupa dell’infrastruttura dati. Magari i tuoi concorrenti non hanno tentato di usarlo per ragioni logistiche, si sono arresi a metà strada o lo hanno elaborato in modo errato”.

Raccolta dei dati audio

Per raccogliere dati audio da YouTube, utilizziamo lo strumento `pytube`, che è un’utility leggera, perfettamente adatta a Python e usabile dalla riga di comando per scaricare video da YouTube.

Il download di video o audio da YouTube può essere facilmente realizzato utilizzando `pytube`. Una volta trovato l’url del video desiderato, potete usare la libreria come mostrato di seguito:

```
from pytube import YouTube
video_url = 'http://youtube.com/watch?v=9bZkp7q19f0'
yt = YouTube(video_url)
yt.streams
    .filter(progressive=True, file_extension='mp4')
    .order_by('resolution')
    .desc()
    .first()
    .download()
```

In questo frammento di codice:

- importiamo la libreria `pytube`;
- specifichiamo l’url del video, `video_url`;
- creiamo un oggetto `yt`;

- filtriamo quindi i flussi di file mp4, li ordiniamo per risoluzione e scarichiamo quello con la risoluzione più alta.

Conversione da parlato a testo

Il modello Whisper

“Abbiamo addestrato e stiamo approntando in open-source una rete neurale chiamata *Whisper* che si avvicina alla robustezza e accuratezza umana nel riconoscimento del parlato inglese.” –

Introducing Whisper (<https://openai.com/index/whisper>).

Whisper, Radford et al. (2022), è un sistema di riconoscimento automatico del parlato (ASR, *Automatic Speech Recognition*) addestrato su 680.000 ore di dati supervisionati multilingue e multi-task raccolti dal Web. Questo dataset su larga scala permette a Whisper di trascrivere in più lingue e tradurre da più lingue all'inglese. OpenAI ha reso Whisper open-source sotto la Licenza MIT. Secondo i risultati dell'articolo, Whisper regge i benchmark standard e spesso è competitivo con modelli che sono stati sottoposti a fine-tuning per specifici dataset.

Nel loro articolo, quelli di OpenAI descrivono vari metodi per valutare e raccogliere dati. I dati raccolti per Whisper erano debolmente supervisionati, il che significa che non sono annotati da trascrittori professionisti, ma invece hanno etichette raccolte dal Web. Per costruire un dataset per l'addestramento del modello consistente in termini di audio e trascrizioni, gli autori hanno eseguito un grande lavoro di data engineering, inclusi molteplici passi di pre-elaborazione e filtraggio euristico per compiti come il rilevamento della lingua di input e l'allineamento delle trascrizioni, che sono dettagliati nell'articolo. A causa dell'enorme quantità di dati

raccolti, il dataset è altamente diversificato, abbracciando un'ampia gamma di parlanti, lingue e ambienti. Maggiori dettagli sull'architettura di Whisper possono essere trovati in Radford et al. (2022).

Whisper elabora l'audio in input suddividendolo in segmenti di 30 secondi e convertendolo in una rappresentazione spettrogramma Mel logaritmico, che serve come input per l'encoder. Il decoder viene addestrato a prevedere la didascalia testuale corrispondente, intervallata da token speciali che dirigono il modello a eseguire compiti come l'identificazione della lingua, i timestamp a livello di frase, la trascrizione del parlato multilingue e la traduzione (vedi *Introducing Whisper* o l'articolo per maggiori dettagli).

Secondo la model card di Whisper, disponibile nel repository GitHub ufficiale di OpenAI (<https://github.com/openai/whisper>), è disponibile in cinque dimensioni, offrendo diversi compromessi tra velocità e accuratezza. La Figura 9.6 elenca i nomi dei modelli disponibili, i loro requisiti approssimativi di memoria e la velocità di inferenza rispetto al modello *large*.

Dimensioni	Parametri	Solo inglese	Multilingue	VRAM richiesta	Velocità relativa
tiny	39 M	<code>tiny.en</code>	<code>tiny</code>	~1 GB	~32x
base	74 M	<code>base.en</code>	<code>base</code>	~1 GB	~16x
small	244 M	<code>small.en</code>	<code>small</code>	~2 GB	~6x
medium	769 M	<code>medium.en</code>	<code>medium</code>	~5 GB	~2x
large	1550 M	N/A	<code>large</code>	~10 GB	1x

Figura 9.6 Tabella dei modelli disponibili e lingue.

Secondo gli esperimenti di OpenAI, i modelli `.en` per applicazioni solo in inglese tendono a comportarsi meglio, specialmente i modelli

`tiny.en` e `base.en`. Per maggiori dettagli, vedi il loro GitHub, citato precedentemente.

Utilizzo in Python

Eseguiamo la trascrizione utilizzando la libreria Python Whisper. Di seguito è riportato un esempio di come trascrivere l'audio utilizzando Whisper:

```
import whisper
audio_input = "audio.mp3"
model_name = "base"
model = whisper.load_model(model_name)
result = model.transcribe(audio_input)
print(result["text"])
```

In questo esempio:

- importiamo la libreria `whisper`;
- specifichiamo il file di input, `audio.mp3`, e il modello che vogliamo usare, `base`;
- carichiamo il modello Whisper specificato, usando `whisper.load_model(model_name)`;
- trascriviamo il file audio usando `model.transcribe(audio_input)`;
- infine, mostriamo il testo trascritto usando `print(result["text"])`.

Internamente, il metodo `transcribe` legge l'intero file ed elabora l'audio con una finestra scorrevole di 30 secondi, eseguendo su ogni finestra previsioni autoregressive sequenza-per-sequenza.

Whisper sui dati audio dei discorsi della Fed

Come abbiamo detto, i modelli `whisper` vengono addestrati su segmenti di input di 30 secondi e non possono elaborare input audio più lunghi in una volta sola. Questo limite può rappresentare un problema per le applicazioni nel mondo reale. Gli autori dell'articolo

propongono una soluzione per trascrivere sequenze audio più lunghe, trascrivendo consecutivamente finestre audio di 30 secondi e spostando questa finestra in base ai timestamp previsti dal modello. Sottolineano inoltre l'importanza di prendere decisioni corrette a riguardo delle euristiche di decodifica per trascrivere in modo affidabile audio lunghi, come dettagliato nel paragrafo 4.5 del loro articolo, Radford et al. (2022).

Questo ha diverse implicazioni per il nostro caso d'uso. Le conferenze “Fed speech” tipicamente durano tra i 45 e i 60 minuti. Utilizzare input audio così lunghi significa che i timestamp previsti da Whisper potrebbero non essere accurati, il che potrebbe potenzialmente introdurre problemi di allineamento dei dati e di bias di anticipazione (*look-ahead*) quando si uniscono i dati per timestamp tra audio e dati dei prezzi – una questione importante nel backtesting delle strategie di trading. L'inaffidabilità dei timestamp previsti per file audio lunghi è qualcosa che abbiamo osservato anche nelle nostre sperimentazioni.

Per garantire che i segnali audio in input siano correttamente allineati con i corrispondenti dati a serie temporali di mercato, evitiamo di affidarci ai timestamp di trascrizione prodotti da Whisper, la cui accuratezza diminuisce all'aumentare della lunghezza dell'audio. Invece, eseguiamo preventivamente una segmentazione dell'audio (descritta di seguito), creando segmenti di input di 60 secondi campionati ogni 30 secondi. Questo processo genera “barre audio” di 60 secondi ogni 30 secondi. Questo metodo ci permette di determinare i timestamp di inizio e fine dell'audio corrispondente, e quindi della sua trascrizione, senza fare affidamento su Whisper per questo scopo. Inoltre, abbiamo introdotto una pausa nella serie audio. Ciò significa unire i timestamp dei dati dei prezzi con i timestamp audio/trascrizione con 15 secondi di anticipo, riducendo così

ulteriormente il rischio di bias di anticipazione. Sebbene tutte queste scelte progettuali limitino la quantità di informazioni utilizzate per calcolare il punteggio di sentiment nei sottosistemi successivi, riducono significativamente anche il rischio di introdurre bias di anticipazione.

Segmentazione dell'audio

Per la segmentazione dell'audio, utilizziamo FFmpeg (<https://ffmpeg.org>), che è una soluzione multiplatforma per registrare, convertire e trasmettere audio e video.

Potete usare ffmpeg-python (<https://github.com/kkroening/ffmpeg-python>): wrapper Python per FFmpeg. Per esempio, nel frammento di codice seguente possiamo vedere come eseguire conversioni tra i diversi formati audio:

```
import ffmpeg
# Definisci i file di input e di output
input_file = 'input_video.mp4'
output_file = 'output_audio.wav'
# Estrai l'audio dal video
ffmpeg.input(input_file).output(output_file).run()
```

Per la segmentazione dell'audio, potete specificare l'ora di inizio e la durata. Questo vi permette di creare specifiche “barre audio” di lunghezze definite, contenenti puri dati. Queste possono poi essere combinate con i dati di mercato per creare dataset arricchiti per l'addestramento e la valutazione del modello.

```
# Definisci il tempo di inizio e la durata in secondi
start_time = 30
duration = 60
# Estrai un determinato segmento
ffmpeg.input(input_file, ss=start_time,
t=duration).output(output_file).run()
```

I risultati sui dati effettivi della Fed sono mostrati nella Figura 9.7.

Analisi del sentiment

Dopo aver effettuato la conversione da parlato a testo utilizzando *whisper*, il passo successivo consiste nel calcolare il sentiment del discorso basandosi sul testo trascritto. Per questo compito, utilizziamo il modello *FinBERT*, Araci (2019), una variante di BERT Devlin et al. (2019) addestrata specificamente su dati finanziari.

```
# load the whisper model
model = whisper.load_model(WHISPER_MODEL_NAME)

100%|██████████| 139M/139M [00:01<00:00, 74.6MiB/s]

If you are running locally with a recent model GPU, the following call may give the message that the installed version of PyTorch does not support the CUDA level of your GPU. You can fix the problem by installing the appropriate version of PyTorch using these instructions: https://pytorch.org/get-started/locally/

transcript = transcribe(model, cut_details, seed=MAIN_SEED)
print(transcript.head())

started at 2023-09-29 14:57:46.030204
0%|          | 0/105 [00:00<?, ?it/s]
ended at 2023-09-29 15:03:34.321123
time elapsed: 0:05:48.290919

                                text                        timestamp
0                Good afternoon. I will begin with 2022-06-15 13:31:00
1    Good afternoon. I will begin with one overarc... 2022-06-15 13:31:30
2    one overarching message. We at the Fed unders... 2022-06-15 13:32:00
3    if we were to have a sustained period of stro... 2022-06-15 13:32:30
4    rate will be appropriate. In addition, we are... 2022-06-15 13:33:00

transcript.head()

                                text                        timestamp
0                Good afternoon. I will begin with 2022-06-15 13:31:00
1    Good afternoon. I will begin with one overarc... 2022-06-15 13:31:30
2    one overarching message. We at the Fed unders... 2022-06-15 13:32:00
3    if we were to have a sustained period of stro... 2022-06-15 13:32:30
4    rate will be appropriate. In addition, we are... 2022-06-15 13:33:00
```

Figura 9.7 Screenshot dell'output di Whisper sui dati della Fed.

Potreste chiedervi: perché non usare il più potente ChatGPT? Una semplice ragione è il costo, sia economico sia computazionale. Utilizzare un modello basato su BERT per questa demo vi permette di lavorare con un modello ampiamente utilizzato nel settore, che può essere eseguito sul vostro computer con risorse minime o con risorse cloud gratuite rese disponibili da Google Cloud. Oltre alle considerazioni sui costi, nel prossimo capitolo, esploreremo come personalizzare questo modello. Nello specifico, vedremo come addestrarlo e ottimizzarlo per l’inferenza, rendendolo più efficiente in termini di memoria e più veloce in esecuzione – qualcosa che

sarebbe molto più impegnativo fare con ChatGPT. Con un modello proprietario di grandi dimensioni come ChatGPT, dovrete fare affidamento interamente sull'infrastruttura di OpenAI, cosa che limiterebbe le possibilità di personalizzazione e l'apprendimento pratico.

BERT

Il modello BERT (*Bidirectional Encoder Representations from Transformers*), è stato introdotto nel 2019 da Devlin et al. (2019).

BERT è stato molto influente sia nella ricerca accademica sia nelle applicazioni industriali. Nell'industria, i risultati di Google Search erano alimentati da un modello BERT, il sistema di raccomandazione di eBay utilizza BERT e varianti di BERT sono state applicate con successo nel lavoro di Hamlet in Criteo. Accademicamente, è stato fatto un grande lavoro per migliorarne le prestazioni, la scalabilità e l'efficienza. A oggi, l'articolo su BERT è stato citato più di 119.678 volte, secondo Google Scholar.

Una ragione per cui BERT è stato così influente è la sua trasformazione del tipico flusso di lavoro di elaborazione del linguaggio naturale, in ambito sia industriale sia accademico. L'articolo ha dimostrato la potenza del pre-addestramento di BERT più il fine-tuning per una varietà di compiti di elaborazione del linguaggio naturale, superando costantemente i modelli all'avanguardia dell'epoca e dimostrandosi inestimabile per scenari con dati di addestramento limitati. A differenza degli approcci tradizionali (all'epoca dell'introduzione di BERT) che spesso richiedevano l'addestramento di modelli separati per ogni task, l'approccio di pre-addestramento e fine-tuning ha reso possibile un flusso di lavoro più generalizzato ed efficiente.

Il concetto di pre-addestramento e fine-tuning, evidenziato nell'articolo tramite risultati sperimentali all'avanguardia in tutti i campi, all'epoca, ha spinto l'industria e il mondo accademico a costruire modelli migliori aumentando la quantità di dati di addestramento e le dimensioni del modello. Questo approccio ha portato allo sviluppo di modelli fondativi che possono poi essere sottoposti a fine-tuning per task più specifici. Le tecniche di fine-tuning saranno trattate nel prossimo capitolo.

Per questo capitolo, descriveremo gli aspetti principali del modello BERT, poiché FinBERT, che usiamo per l'analisi del sentiment, è una versione di BERT sottoposta a fine-tuning specificamente per dati finanziari.

Panoramica su BERT

BERT si basa su un'architettura transformer solo-encoder, che abbiamo trattato nel Capitolo 5 nel paragrafo dedicato ai transformer. A differenza di modelli precedenti, come GPT-1 (Radford et al., 2018) o ELMo (Peters et al., 2018), che utilizzavano il contesto da sinistra a destra o una concatenazione di contesti da sinistra a destra e da destra a sinistra calcolati in modo indipendente, BERT ha introdotto il contesto bidirezionale. Ciò significa che BERT considera simultaneamente sia il contesto sinistro sia il contesto destro di una parola. I risultati empirici hanno mostrato che questo contesto bidirezionale fornisce embedding più adattati al contesto circostante. Le differenze nel modo in cui questi modelli gestiscono le informazioni contestuali sono illustrate nella Figura 9.8.

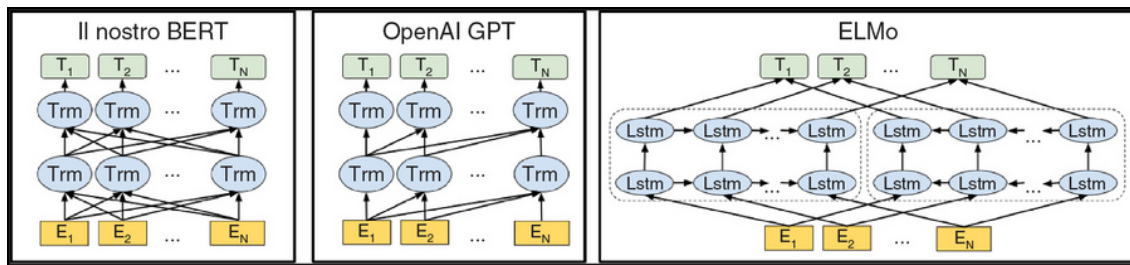


Figura 9.8 Figura 3 di Devlin et al. (2019): illustrazione delle differenze nelle architetture dei modelli di pre-addestramento, confrontando BERT, OpenAI-GPT ed ELMo. OpenAI GPT utilizza un transformer da sinistra a destra. ELMo utilizza la concatenazione di LSTM addestrati indipendentemente da sinistra a destra e da destra a sinistra per generare le feature per i compiti. Tra i tre, solo le rappresentazioni di BERT sono condizionate congiuntamente sul contesto sia sinistro sia destro in tutti i livelli. Oltre alle differenze in termini di architettura, BERT e OpenAI GPT sono approcci di fine-tuning, mentre ELMo è un approccio basato su feature.

Rappresentazioni di input/output

Per permettere a BERT di gestire una varietà di compiti, Devlin et al. (2019) hanno progettato una speciale rappresentazione di input, la quale permette a BERT di rappresentare in modo non ambiguo sia frasi singole sia coppie di frasi con una singola sequenza di token, cosa che lo rende adatto a vari compiti. Esploriamo come sono rappresentati gli input in BERT.

Rappresentazioni degli input

Come discusso nel Capitolo 5, il primo passo nella pipeline di elaborazione del linguaggio naturale coinvolge un processo di tokenizzazione, che prende in input una sequenza di parole grezze e restituisce una sequenza di ID di token. Una volta che abbiamo questa sequenza di ID di token, BERT estende la rappresentazione di input.

Illustriamo questo processo con due frasi, Frase A e Frase B, e vediamo come BERT calcola la loro rappresentazione iniziale e le elabora, utilizzando la notazione di Devlin et al. (2019).

1. Tokenizzazione.

- La Frase A viene tokenizzata in una sequenza di token: $Tok_1, Tok_2, \dots, Tok^N$, dove N è il numero di token contenuti nella Frase A.

- La Frase B viene tokenizzata in una sequenza di token: $Tok_1, Tok_2, \dots, Tok_m$, dove M è il numero di token contenuti nella Frase B.

2. Uso dei token speciali [SEP] e [CLS].

- Le sequenze tokenizzate della Frase A e della Frase B vengono concatenate e separate da un token speciale, [SEP]. Questo token aiuta BERT a distinguere le due frasi quando vengono combinate in un'unica sequenza di token.

- Uno speciale token di classificazione [CLS] viene aggiunto all'inizio della sequenza. Questo token svolge un ruolo critico nei compiti successivi, poiché la sua rappresentazione finale è trattata come la "rappresentazione aggregata della sequenza" dell'input; vedi Devlin et al. (2019).

3. Embedding iniziali: ogni token viene mappato su un embedding iniziale (gli embedding sono discussi più in dettaglio nel Capitolo 5).

- Il token [CLS] viene mappato su $E_{[CLS]}$.

- Il token [SEP] viene mappato su $E_{[SEP]}$.

- I token della Frase A sono associati agli embedding $E_1, \dots, E^N$.

- I token della Frase B sono associati agli embedding $E'_1, \dots, E'_m$.

m .

Questi passaggi fanno parte del processo di rappresentazione dell'input di BERT, che è illustrato nella Figura 9.9. Come esempio per comprendere le dimensioni associate a ciascun embedding, tipicamente gli embedding per ogni token risiedono in uno spazio nascosto a 768 dimensioni per la configurazione BERT-Base o in uno spazio a 1024 dimensioni per la configurazione BERT-Large (maggiori dettagli su questo sono forniti nel prossimo paragrafo).

Rappresentazioni dell'output

Dopo che i layer transformer elaborano l'input, gli embedding iniziali dei token vengono aggiornati per produrre gli stati nascosti finali o le rappresentazioni per ogni token. Nell'output:

- lo stato nascosto finale per il token [CLS] è denotato come C ;
- lo stato nascosto finale per il token [SEP] è denotato come $T_{[SEP]}$;
- gli stati nascosti finali per i token nella Sequenza A e Sequenza B sono $T_1, \dots, T^N$ e $T'_1, \dots, T'_m$, rispettivamente.

Anche questo si trova visualizzato nella Figura 9.9.

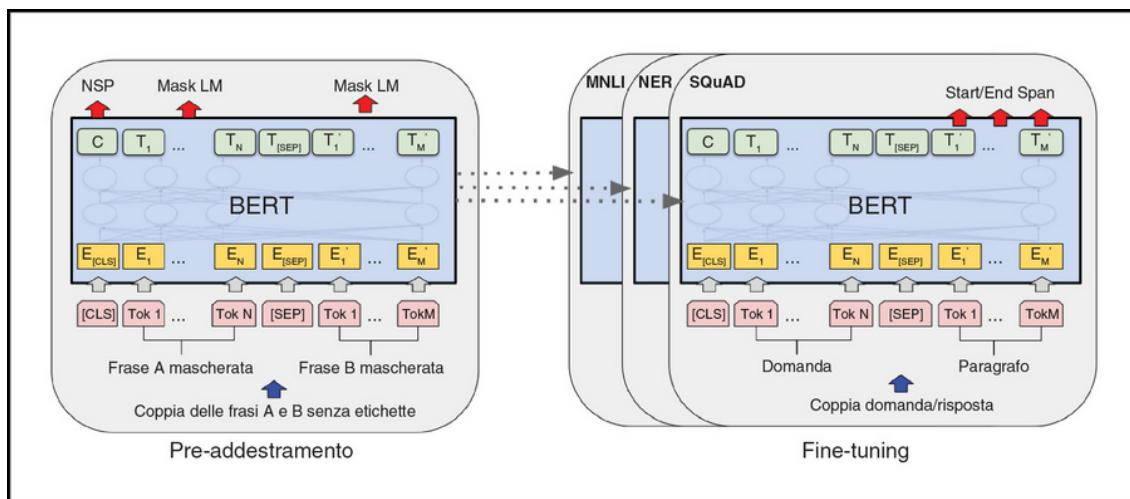


Figura 9.9 Figura 1 di Devlin et al. (2019): illustrazione della rappresentazione di input/output di BERT. A parte i layer di output, viene utilizzata la stessa

architettura sia per il pre-addestramento sia per il fine-tuning. Lo stesso modello pre-addestrato può essere utilizzato come inizializzazione e adattato a molteplici compiti, come mostrato nell'immagine a destra. Nell'articolo, durante il fine-tuning, tutti i parametri sono stati aggiornati. [CLS] è un simbolo speciale aggiunto all'inizio di ogni esempio di input e [SEP] è un token speciale utilizzato per separare le coppie di frasi.

Il ruolo del token speciale [CLS] (*classification*) è di particolare importanza, poiché viene utilizzato come *rappresentazione aggregata* dell'intera sequenza di input. Devlin et al. (2019) hanno utilizzato *C* in esperimenti relativi a compiti di classificazione dimostrandone l'efficacia. Infatti, anche FinBERT utilizza *C* per addestrare il modello a svolgere l'analisi del sentiment. In tale task, *C* viene passato a un classificatore, spesso indicato come “testa di classificazione” (*classification head*), per prevedere il sentiment della sequenza di input (una testa di classificazione è una rete neurale aggiunta in cima all'encoder che mappa *C*, la rappresentazione codificata, sul numero desiderato di classi).

Un'altra caratteristica chiave di BERT è l'introduzione di un terzo tipo di embedding per codificare l'input, noto come *segment embedding*. Nell'introduzione ai transformer nel Capitolo 5, tipicamente gli embedding di input per i transformer vengono calcolati come una combinazione di *embedding dei token* e *informazioni posizionali*, che di solito vengono combinati per addizione. In BERT, gli *embedding di segmento* (*segment embedding*) vengono appresi durante il processo di addestramento e utilizzati per indicare se un token appartiene alla Frase A o alla Frase B.

In sintesi, gli embedding di input sono la somma di tre componenti: embedding dei token, embedding posizionali e embedding di segmento. Il tutto è illustrato nella Figura 9.10.

Questa rappresentazione di input permette a BERT di rappresentare in modo non ambiguo sia una singola frase sia coppie di frasi in una sequenza di token, il che permette di utilizzare BERT per task come *Question answering*, dove tipicamente forniamo il dataset come una collezione di coppie (Domanda, Risposta). Inoltre, questa rappresentazione rende possibile l'obiettivo di pre-addestramento NSP (*Next Sentence Prediction*), che tratteremo in seguito.

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Embedding dei token	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{##ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Embedding di segmento	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Informazioni posizionali	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

Figura 9.10 Figura 2 da Devlin et al. (2019): illustrazione della rappresentazione di input di BERT. Gli embedding di input per il modello transformer sono la somma degli embedding dei token, degli embedding di segmento e delle informazioni posizionali.

Obiettivi del pre-addestramento

Gli obiettivi di pre-addestramento di BERT includono due task.

1. *Task MLM (Masked Language Model)*: in questo task, le parole di una frase vengono mascherate casualmente e il modello viene addestrato a prevedere le parole mancanti. Nell'articolo su BERT, viene mascherato il 15% delle parole. Di queste, l'80% delle volte viene utilizzato il token $[MASK]$, il 10% delle volte viene utilizzata la parola corretta e il restante 10% delle volte viene utilizzata una parola scelta a caso.

2. *Task NSP (Next Sentence Prediction)*: questo task implica prevedere se una data frase B segue logicamente la frase A. Il modello viene addestrato su coppie di frasi tratte da un corpus.

Il contesto bidirezionale e gli embedding di segmento introdotti da BERT, insieme ai suoi due obiettivi di pre-addestramento, sono considerate le principali innovazioni del lavoro su BERT (Devlin et al., 2019).

I dati utilizzati per il pre-addestramento di BERT includevano il BookCorpus (900 milioni di parole) (Zhu et al., 2015) e la Wikipedia in inglese (2.500 milioni di parole), con alcune pre-elaborazioni aggiuntive descritte nell'articolo.

BERT è disponibile in due configurazioni.

- *BERT-Base*: 12 layer (blocchi transformer), dimensione nascosta 768, 12 teste di attenzione, con un numero totale di parametri pari a 110 milioni.
- *BERT-Large*: 24 layer (blocchi transformer), dimensione nascosta 1024, 16 teste di attenzione, con un numero totale di parametri pari a 340 milioni.

Il processo di pre-addestramento più fine-tuning di BERT per specifiche applicazioni di elaborazione del linguaggio naturale ha superato i modelli all'avanguardia dell'epoca, incluso GPT-1 di OpenAI, in tutti i compiti considerati.

Fine-tuning del BERT per un'analisi del sentiment finanziario: la creazione di FinBERT

Il fine-tuning del BERT su dati finanziari è cruciale, perché i modelli per l'analisi del sentiment generici potrebbero non catturare accuratamente le sfumature del vocabolario finanziario e del sentiment del mercato. Per esempio, la frase "Mark ha lasciato

Facebook” potrebbe essere considerata neutra dalla maggior parte dei modelli di sentiment, ma nel contesto finanziario, potrebbe avere un impatto significativo sul mercato. Effettuando il fine-tuning del BERT su testi finanziari, permettiamo al modello di apprendere nuovi concetti, nomi ed entità specifici del mondo della finanza, ottenendo un’analisi del sentiment più accurata. Il fine-tuning del BERT per molteplici compiti è semplice, come dimostrato da Devlin et al. (2019) in molteplici esperimenti.

Come discusso nel paragrafo precedente, il fine-tuning del BERT per un compito di classificazione può essere effettuato utilizzando la rappresentazione nascosta finale del token di classificazione [CLS], che funge da “rappresentazione aggregata” dell’intera sequenza. Questa rappresentazione viene poi inviata a una testa di classificazione (*classification head*), che restituisce le probabilità associate a ogni classe di sentiment.

Per addestrare l’intero modello, abbiamo semplicemente bisogno di una collezione di coppie (input, output). Durante l’addestramento, tutti i parametri del modello vengono aggiornati, inclusi quelli del BERT e della testa di classificazione. In letteratura, BERT è spesso indicato come il “corpo” della rete, per distinguerlo dal layer aggiunto per svolgere un compito specifico, come la classificazione, che viene chiamato “testa di classificazione”.

Nell’esperienza di Hamlet, ci sono variazioni pratiche su quali parametri addestrare o sottoporre a fine-tuning. Per certe applicazioni, il fine-tuning della sola testa di classificazione può produrre buoni risultati. Tuttavia, questo dipende dal caso d’uso specifico e richiede una validazione empirica. Come vedremo nel prossimo capitolo, il fine-tuning di tutti i parametri potrebbe essere molto dispendioso in termini di spesa e tempo, ma spesso migliora le prestazioni. Per affrontare questo problema, si possono utilizzare

metodi di fine-tuning più efficienti, che verranno discussi più avanti in questo stesso capitolo.

Per il fine-tuning di compiti come la classificazione del testo (per esempio, l'analisi del sentiment), il testo in input viene inviato a BERT, e la rappresentazione finale del token [CLS] viene passata a un layer di output per la classificazione. Questo è l'approccio seguito dagli autori di Araci (2019) per produrre il modello FinBERT per la classificazione del testo. Gli autori hanno sperimentato anche un FinBERT per la regressione, dove la differenza principale rispetto al caso della classificazione è l'uso di un target continuo, un layer di output per la regressione e una funzione loss per la regressione, in particolare l'errore quadratico medio. Questo punteggio può servire come rappresentazione che riassume l'elaborazione di dati alternativi relativamente complessi in una strategia di trading multi-fattoriale, per esempio. Per saperne di più sull'applicazione di modelli basati su transformer alle serie temporali, consultate il Capitolo 5.

I dataset utilizzati per effettuare il fine-tuning di BERT in FinBERT sono i seguenti.

1. *TRC2-financial*, un sottoinsieme del TRC2 di Reuters. È costituito da 1.8 milioni di articoli di news pubblicati da Reuters tra il 2008 e il 2010. Questi dati vengono utilizzati per il pre-addestramento di BERT. Il corpus può essere ottenuto per scopi di ricerca facendo richiesta qui:
<https://trec.nist.gov/data/reuters/reuters.html>.
2. *Financial PhraseBank*, vedi Malo et al. (2014), che è il principale dataset per l'analisi del sentiment utilizzato in questo articolo.
3. *FiQA Sentiment*, vedi *WWW '18: Companion Proceedings of the Web Conference 2018* (2018), un dataset creato per la challenge di opinion mining e question-answering finanziario della

conferenza WWW '18. Include 1.174 titoli di news finanziarie e tweet con il corrispondente punteggio di sentiment.

Per maggiori dettagli sui dataset, vedi Araci (2019).

Ora, esploriamo l'uso del modello FinBERT per la nostra applicazione pratica.

Utilizzo di FinBERT

In questo esperimento, utilizziamo FinBERT, che è stato addestrato da Araci (2019) e reso disponibile su ProsusAI/finbert (<https://huggingface.co/ProsusAI/finbert>). FinBERT si basa sul fine-tuning di BERT-Base, quindi ha una dimensione del modello di 110M parametri. Abbiamo anche scritto del codice aggiuntivo sulla base di ProsusAI/finbert per semplificare la valutazione del sentiment.

Qui, vediamo come possiamo utilizzare questo modello per calcolare il sentiment delle frasi. Cominciamo con un semplice esempio.

Per calcolare il sentiment della frase “Mark has left Facebook”, possiamo usare il seguente codice:

```
# negativa
finbert_sa.predict_overall_sentiment('Mark has left facebook',
finbert_model)
>>> -0.45569813
```

Il punteggio di sentiment di -0.45569813 indica un sentiment negativo. Allo stesso modo, possiamo calcolare il sentiment di altre frasi. Per esempio, considerate la frase “Jeff Bezos acquires Netflix”:

```
# sembra neutra
finbert_sa.predict_overall_sentiment('Jeff bezos acquires Netflix',
finbert_model)
>>> -0.0069163926
```

Questa frase ha un punteggio di sentiment di -0.0069163926, che è piuttosto neutro. Osserviamo un altro esempio, “Business magnate

Elon Musk initiated an acquisition of American social media company Twitter”:

```
# sembra neutra
finbert_sa.predict_overall_sentiment(''Business magnate Elon Musk
    initiated an acquisition of American social media company
    Twitter'',
    finbert_model)
>>> 0.090285115
```

La frase ha un punteggio di sentiment leggermente positivo, 0.090285115. Infine, valutiamo il sentiment di una frase più complessa relativa alla Federal Reserve:

```
# negativa
finbert_sa.predict_overall_sentiment(''The Federal Reserve held
    interest rates steady,
    while also indicating it still expects one more hike before the end
    of the year
    and fewer cuts than previously indicated next year.'',
    finbert_model)
>>> -0.13109717
```

Questa frase ha un punteggio di sentiment negativo, -0.13109717.

Possiamo anche elaborare più frasi in una volta sola. Per esempio:

```
sentences = [
    'Mark has left facebook',
    'Jeff bezos acquires Netflix',
    ''Business magnate Elon Musk initiated an
    acquisition of American social media company Twitter'',
    ''The Federal Reserve held interest rates steady, while also
    indicating
    it still expects one more hike before the end of the year and fewer
    cuts
    than previously indicated next year.''
]
sentiments = finbert_sa.process_sentences(finbert_model, sentences)
sentiments
>>> array([-4.5569813e-01, -1.7964840e-04,
           9.0285115e-02, -1.3109717e-01], dtype=float32)
```

Questo array di punteggi di sentiment aiuta ad analizzare il sentiment di più frasi in modo efficiente.

Risultati sperimentali

In questo paragrafo, descriviamo i risultati della pipeline del modello utilizzata per convertire i dati audio dei discorsi della Fed in segnali di trading. Come promemoria, la pipeline del modello è costituita dai seguenti passi.

1. Raccolta di audio da YouTube utilizzando `pytube`.
2. Segmentazione dell'audio utilizzando `ffmpeg`.
3. Trascrizione delle barre audio in testo utilizzando il modello Whisper.
4. Calcolo dei punteggi di sentiment per il testo trascritto utilizzando FinBERT.
5. Generazione di segnali di trading basati sui punteggi di sentiment, con decisioni di acquisto o vendita prese in base a determinate soglie ricavate dalla distribuzione dei punteggi di sentiment.

I dati dei prezzi SPY sono campionati a intervalli (barre) di 30 secondi e, in base al sentiment, tentiamo di prendere una decisione di trading ogni 30 secondi.

Nel nostro test, abbiamo raccolto un totale di 5 ore di dati vocali dalla Fed utilizzando `pytube`, corrispondenti a cinque diverse conferenze. L'audio è stato scaricato con un bit rate di 160kbps per garantire un'alta qualità.

Come spiegato in precedenza, dopo il download dei dati, l'audio è stato segmentato per creare “barre audio” di 60 secondi, campionate a intervalli di 30 secondi, con timestamp ritardati di 15 secondi per ridurre il rischio di introdurre bias di anticipazione (*look-ahead*). Per la trascrizione, abbiamo utilizzato il modello base di Whisper per il suo compromesso tra velocità e accuratezza. Nei nostri esperimenti

relativi alla parte di elaborazione del parlato, abbiamo notato che i risultati erano sensibili all'uso di versioni più grandi di Whisper nel produrre le trascrizioni, ma questo non ha influenzato significativamente i risultati. Invece, la qualità dei dati di input ha avuto un impatto più importante. Pertanto, abbiamo scaricato la qualità più elevata disponibile e comune a tutti i video.

Una volta create le barre audio insieme alle corrispondenti trascrizioni, abbiamo utilizzato FinBERT per calcolare il punteggio di sentiment di ogni barra audio. I segnali di trading sono stati generati in base ai punteggi di sentiment calcolati. La distribuzione dei punteggi di sentiment è stata analizzata e le soglie sono state impostate in base ai suoi quantili. Se il punteggio di sentiment era al di sopra della soglia superiore, generavamo un segnale di acquisto per SPY; se era al di sotto della soglia inferiore, veniva generato un segnale di vendita allo scoperto (*short*). Altrimenti, non veniva intrapresa alcuna azione, a meno che non fosse già aperta una posizione, nel qual caso la chiudevamo.

Ecco un riepilogo dei principali passi eseguiti da ogni sottosistema.

1. Raccolta e pre-elaborazione dei dati.
 - Raccolte 5 ore di dati vocali della Fed.
 - Audio scaricato con bit rate di 160 kbps.
 - Audio segmentato in barre di 60 secondi con un intervallo di campionamento di 30 secondi e un ritardo di 15 secondi.
2. Trascrizione e analisi del sentiment.
 - Utilizzato il modello base di Whisper per la trascrizione.
 - Applicato FinBERT per calcolare i punteggi di sentiment per ogni barra audio.
3. Generazione del segnale di trading.
 - Analizzata la distribuzione dei punteggi di sentiment per impostare le soglie.

- Generati segnali di acquisto o vendita sulla base dei punteggi di sentiment.

La Figura 9.11 mostra la serie temporale dei rendimenti forward dello SPY e del punteggio di sentiment prodotto dal sistema.

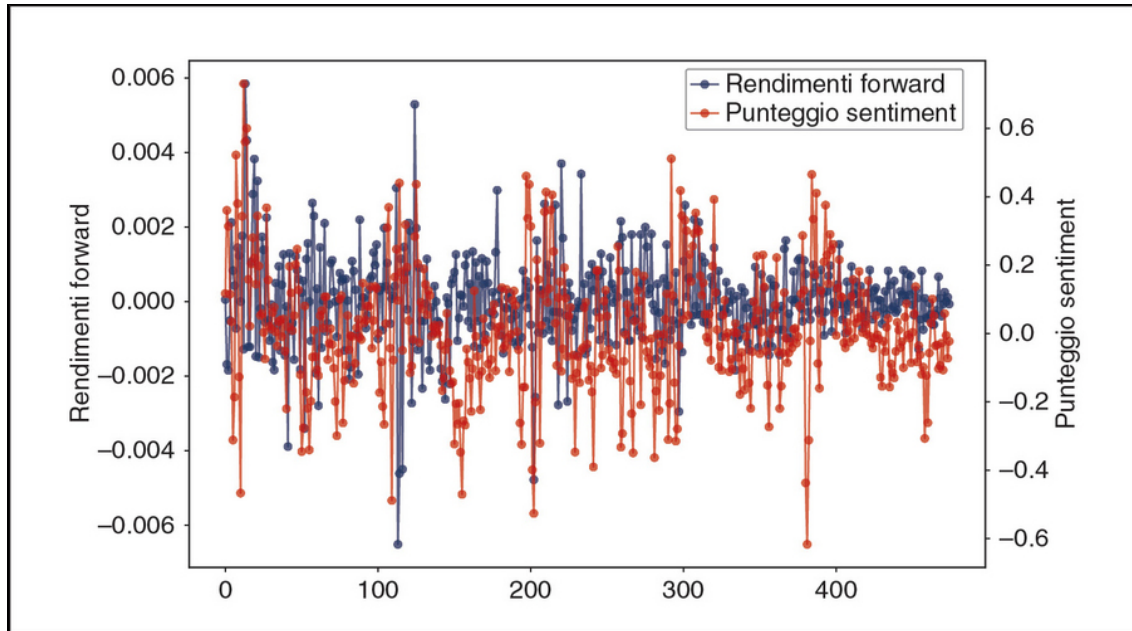


Figura 9.11 Grafico del segnale di sentiment e rendimenti forward nella serie temporale.

Nella Figura 9.12 presentiamo il grafico a dispersione tra il punteggio di sentiment e i rendimenti forward dello SPY, che rivela una correlazione positiva.

Infine, la Tabella 9.1 riassume i principali risultati di performance della strategia. Questi risultati includono un semplice modello costante per i costi di transazione, supponendo 5 punti base per transazione. Complessivamente, in tutti i video, il segnale di trading predice il segno dei rendimenti forward dello SPY con un'accuratezza del 53.15%. La correlazione tra il punteggio di sentiment e i rendimenti forward dello SPY è del 14.14% con un p -value dello 0.199%.

Tabella 9.1 Tabella delle performance.

Rapporto di Sharpe non normalizzato	Accuratezza	Correlazione di Pearson del segnale di sentiment vs SPY	p-value
11.3844%	53.1532%	14.1458%	0.1999%

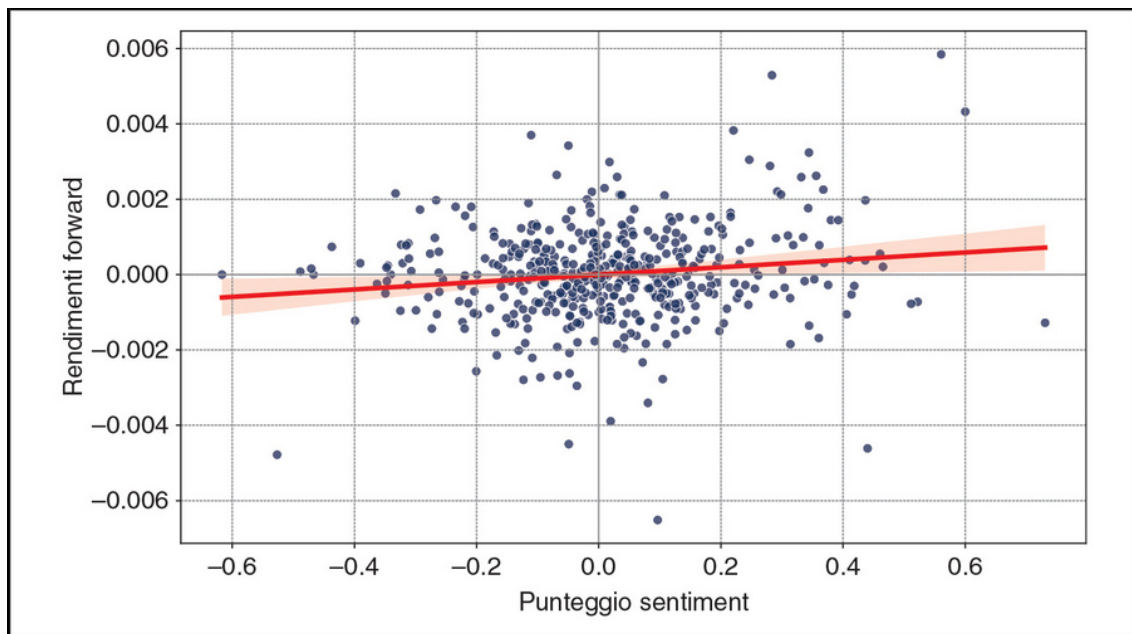


Figura 9.12 Grafico a dispersione del segnale sentiment vs rendimenti forward.

Conclusioni

I risultati sperimentali dimostrano come possiamo elaborare efficacemente dati alternativi, in questo caso audio, per generare segnali di trading attuabili. Raccogliendo audio di alta qualità da Internet, utilizzando un'appropriata segmentazione dell'audio e impiegando modelli complessi per la trascrizione e il calcolo del punteggio di sentiment, siamo stati in grado di generare segnali di

trading efficaci ad alta frequenza per fare trading nei successivi 30 secondi dello SPY. Questa applicazione evidenzia la potenza dei modelli profondi autoregressivi transformer per modellare l'audio e il testo, come descritto nel Capitolo 5. I nostri esperimenti hanno dimostrato il ruolo critico di FinBERT, un modello BERT ottimizzato per i dati finanziari. A differenza dei modelli generici per l'analisi del sentiment, che spesso trascurano le sfumature finanziarie, FinBERT ha superato gli altri modelli testati non addestrati su dati finanziari, migliorando significativamente le nostre metriche di performance.

Il prossimo capitolo esplora metodi per personalizzare gli LLM per task specializzati. Questo approccio evidenzia la possibilità di comporre modelli di deep learning molto complessi per affrontare applicazioni finanziarie articolate.

Capitolo 10

Efficienza dell'inferenza

Introduzione

In passato, il tipico flusso di lavoro del machine learning era relativamente semplice: raccogliere dati per il problema in questione, addestrare un modello su tali dati e infine distribuire il modello addestrato mettendolo in produzione per rispondere ai nuovi input.

Tuttavia, con l'avvento di modelli sempre più grandi, questo approccio tradizionale è diventato meno fattibile. Le grandi dimensioni dei modelli attuali spesso rendono impraticabile la loro distribuzione in produzione così come sono, almeno per la maggior parte delle aziende o istituzioni. Ciò può essere dovuto a vincoli come i requisiti di latenza per le specifiche applicazioni in cui questi modelli vengono utilizzati, i costi proibitivi dell'infrastruttura associati all'esecuzione di questi modelli su larga scala.

Di conseguenza, il tipico flusso di lavoro del machine learning nel settore si è evoluto. Invece di distribuire il modello direttamente dopo l'addestramento, ora viene comunemente introdotto un passo di compressione che prende il modello prodotto dopo l'addestramento e ne produce una versione più piccola – in termini di numero di parametri o di memoria richiesta per memorizzarli – cercando di mantenere o addirittura migliorare le prestazioni del modello originale. Al giorno d'oggi, la compressione dei modelli è essenziale

per soddisfare i requisiti di sistema di certe applicazioni, come ridurre la latenza o semplicemente ridurre i costi dell'infrastruttura.

Un esempio dal mondo industriale è il caso di studio di eBay (vedi Xue et al. [2023]). Hanno migliorato il loro sistema di raccomandazione effettuando il fine-tuning di BERT sui nomi degli articoli di eBay, che spesso includono parole specifiche dei prodotti, affiancato al corpus di Wikipedia, che è adatto alle esigenze dei compiti di comprensione del linguaggio di eBay. In valutazioni offline, questo modello eBERT ha superato i modelli BERT standard su tutta una serie di compiti di tagging specifici per eBay. Tuttavia, il modello eBERT era troppo grande per raggiungere le specifiche di bassa latenza richieste al momento dell'inferenza, rendendo difficile la fornitura di raccomandazioni in tempo reale. Per affrontare questo problema, in eBay hanno utilizzato tecniche di compressione dei modelli per produrre una versione più piccola che soddisfacesse i loro requisiti inferenziali. Hanno anche ottimizzato il modello per l'inferenza su CPU, per ridurre i costi. Per maggiori dettagli sul loro lavoro, vedi Xue et al. (2023).

Nell'esperienza professionale di Hamlet in Criteo, ha visto che le tecniche di compressione sono cruciali non solo per la riduzione dei costi, che è certo una preoccupazione significativa, ma anche per soddisfare i vincoli di progettazione del sistema. Sebbene si possano impiegare tecniche di parallelizzazione massiccia per distribuire i modelli e, potenzialmente, soddisfare i requisiti di latenza, in alcuni casi, gli alti costi associati a queste soluzioni possono ridurre i potenziali benefici del sistema.

In questo capitolo, introduciamo le tecniche più comuni utilizzate nell'industria per rendere efficiente l'inferenza, come la distillazione della conoscenza (*knowledge distillation*) e la quantizzazione dei modelli. Per illustrare queste tecniche, come caso di studio

utilizziamo FinBERT – il modello applicato nella nostra applicazione pratica nel capitolo precedente –, mostrando come renderlo più veloce nell’inferenza. Concludiamo il capitolo mostrando alcune tecniche per personalizzare gli LLM, per permettervi di adattare i modelli alle vostre specifiche esigenze.

Regolare la scala degli LLM: prestazioni, costo computazionale e abilità emergenti

Come abbiamo visto nel capitolo precedente, dove abbiamo esplorato BERT, uno dei primi, più utilizzati e di successo LLM (*Large Language Model*), l’approccio attuale nel risolvere tutta una varietà di compiti di elaborazione del linguaggio naturale (NLP, *Natural Language Processing*) coinvolge un processo di addestramento in due fasi: pre-addestramento e fine-tuning. Nella prima fase, un modello di grandi dimensioni viene pre-addestrato su enormi quantità di dati, per poi essere sottoposto a fine-tuning per risolvere specifici compiti.

Al giorno d’oggi, ci sono prove evidenti che il miglioramento delle prestazioni del modello durante la fase di pre-addestramento offra miglioramenti in molteplici compiti NLP. Per esempio, nel loro articolo, Devlin et al. (2019) hanno dimostrato empiricamente che la versione più grande di BERT, *BERT-Large*, ha superato la versione più piccola, *BERT-Base*, durante il pre-addestramento e l’ha costantemente superata in tutti i compiti NLP considerati.

Date queste prove, sorge spontanea una domanda: se è cruciale che le prestazioni siano elevate durante il pre-addestramento, come dovremmo affrontare efficacemente questa fase? Tra le molte variabili e strategie disponibili, quali sono le più critiche?

Nell'articolo di Kaplan et al. (2020), gli autori hanno scoperto che un modo per migliorare le prestazioni risiede nel cambiare la scala dei modelli: nello specifico, aumentare il numero di parametri, la quantità di dati di addestramento e la durata del tempo di addestramento. Estendendo questi fattori, si notano significativi vantaggi prestazionali. Gli autori hanno riscontrato empiricamente che le prestazioni seguono una relazione a legge di potenza rispetto a questi fattori, come mostra la Figura 10.1.

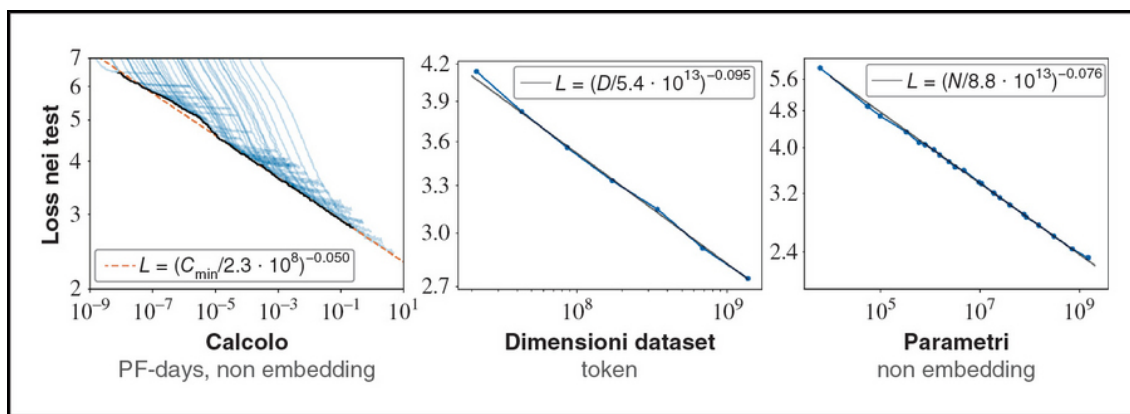


Figura 10.1 Figura 1 da Kaplan et al. (2020): illustrazione di come le prestazioni di modellazione del linguaggio migliorano con l'aumentare delle dimensioni del modello, delle dimensioni del dataset e della quantità di calcoli utilizzate per l'addestramento. Per prestazioni ottimali, occorre scalare tutti e tre questi fattori. Le prestazioni empiriche mostrano una relazione a legge di potenza per ogni singolo fattore quando gli altri due non sono limitanti.

Per maggiori dettagli su quali altri parametri influenzano le prestazioni, fate riferimento all'articolo di Kaplan et al. (2020).

Queste leggi empiriche, che suggeriscono che l'aumento di uno qualsiasi dei fattori menzionati possa portare a miglioramenti quasi prevedibili delle prestazioni del modello, hanno ulteriormente motivato l'industria a estendere la scala dei modelli, aspetto cruciale

anche per sbloccare le *abilità emergenti* dei modelli LLM, che trattiamo brevemente nel prossimo paragrafo.

Sebbene questa via dello scaling sia molto efficace per migliorare le prestazioni, porta anche ad alcuni svantaggi: l'aumento dei costi di inferenza e anche dell'infrastruttura necessaria per servire questi grandi modelli.

Abilità emergenti

Sebbene le leggi di scaling forniscano un percorso potente per migliorare le prestazioni del modello, recenti prove suggeriscono che sono fondamentali anche per sbloccare le abilità emergenti dei modelli LLM. Le abilità emergenti sono definite da Wei et al. (2022) nel seguente modo: “Un’abilità è emergente se non è presente in modelli più piccoli ma è presente in modelli più grandi”.

Ci sono due aspetti principali delle abilità emergenti che è importante conoscere:

- non possono essere direttamente previste estrapolando dalle leggi di scaling (come quelle discusse in precedenza);
- iniziano a emergere solo dopo che il modello raggiunge una certa scala.

Nel prossimo paragrafo, utilizzeremo il lavoro di Wei et al. (2023) sulle abilità emergenti dei modelli di grandi dimensioni. In particolare, illustreremo come le dimensioni del modello e il tempo di addestramento contribuiscano all'emergere di queste abilità.

L'impatto delle dimensioni del modello

Per illustrare l'impatto delle dimensioni del modello sulle abilità emergenti, esaminiamo le prestazioni dei modelli LLM (*Large*

Language Model) nel risolvere problemi di matematica verbale “da scuole medie”.

Nel lavoro di Wei et al. (2023), il dataset GSM8K viene utilizzato per questa valutazione, dove si chiede agli LLM di risolvere problemi di matematica utilizzando il prompting sia standard sia chain-of-thought. Secondo Wikipedia, “Il prompting chain-of-thought è una tecnica che permette ai modelli LLM di risolvere un problema come una serie di passaggi intermedi prima di dare una risposta finale” (vedi https://en.wikipedia.org/wiki/Prompt_engineering).

Nella Figura 10.2, potete vedere le prestazioni misurate come tasso di risoluzione (%) rispetto alla scala del modello, rappresentata dal numero di parametri del modello (in miliardi).

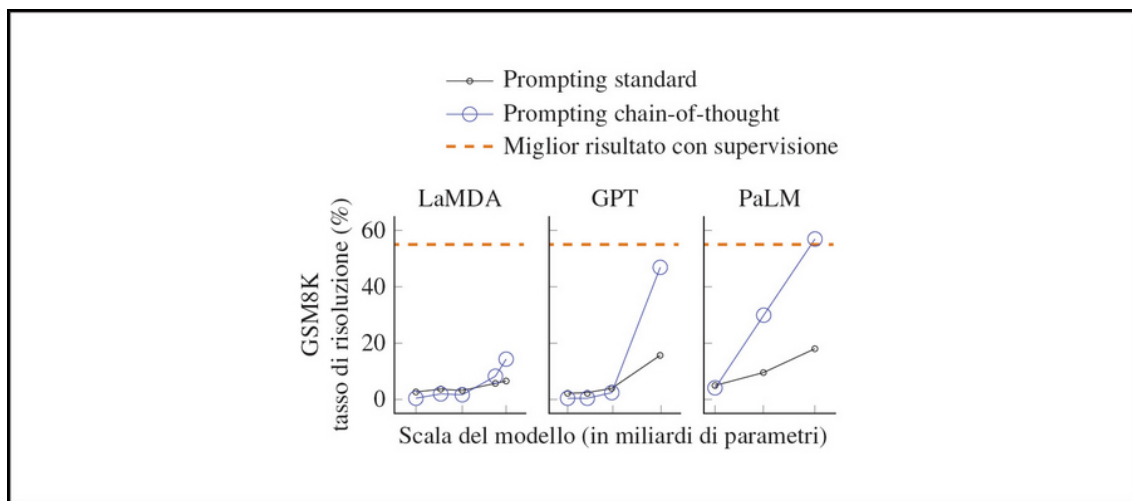


Figura 10.2 Figura 4 da Wei et al. (2023): illustrazione di come le prestazioni (tasso di risoluzione) migliorano all’aumentare della scala del modello, misurata in miliardi di parametri, per il prompting standard e chain-of-thought. Quest’ultimo permette agli LLM di risolvere problemi matematici complessi, con il ragionamento a catena che emerge come un’abilità con l’aumentare della scala del modello.

Il grafico mostra le prestazioni di tre diversi modelli: LaMDA, GPT e PaLM. Emerge uno schema chiaro per ognuno di questi modelli: all'aumentare del numero di parametri, migliorano anche le prestazioni. Per esempio, concentrandosi sui risultati con il prompting chain-of-thought, che tende a produrre risultati migliori, LaMDA, con 137 miliardi di parametri, raggiunge prestazioni inferiori al 20%. Al contrario, GPT, con 175 miliardi di parametri, raggiunge prestazioni superiori al 40%. PaLM, con 540 miliardi di parametri, raggiunge prestazioni di circa il 50%.

L'effetto del tempo di addestramento

Le prove suggeriscono che le abilità emergenti compaiono solo dopo un gran numero di iterazioni di addestramento. Non basta semplicemente avere un modello grande; per sbloccare queste capacità deve anche essere addestrato in modo esteso.

Nella Figura 10.3, tratta dall'articolo di Wei et al. (2022), vediamo le prestazioni di vari modelli – LaMDA, GPT-3, Gopher, Chinchilla, PaLM e un modello scelto a caso – su otto diversi compiti di NLP, come l'aritmetica modulare, la comprensione del linguaggio naturale multi-task, la parola-in-contesto e altri.

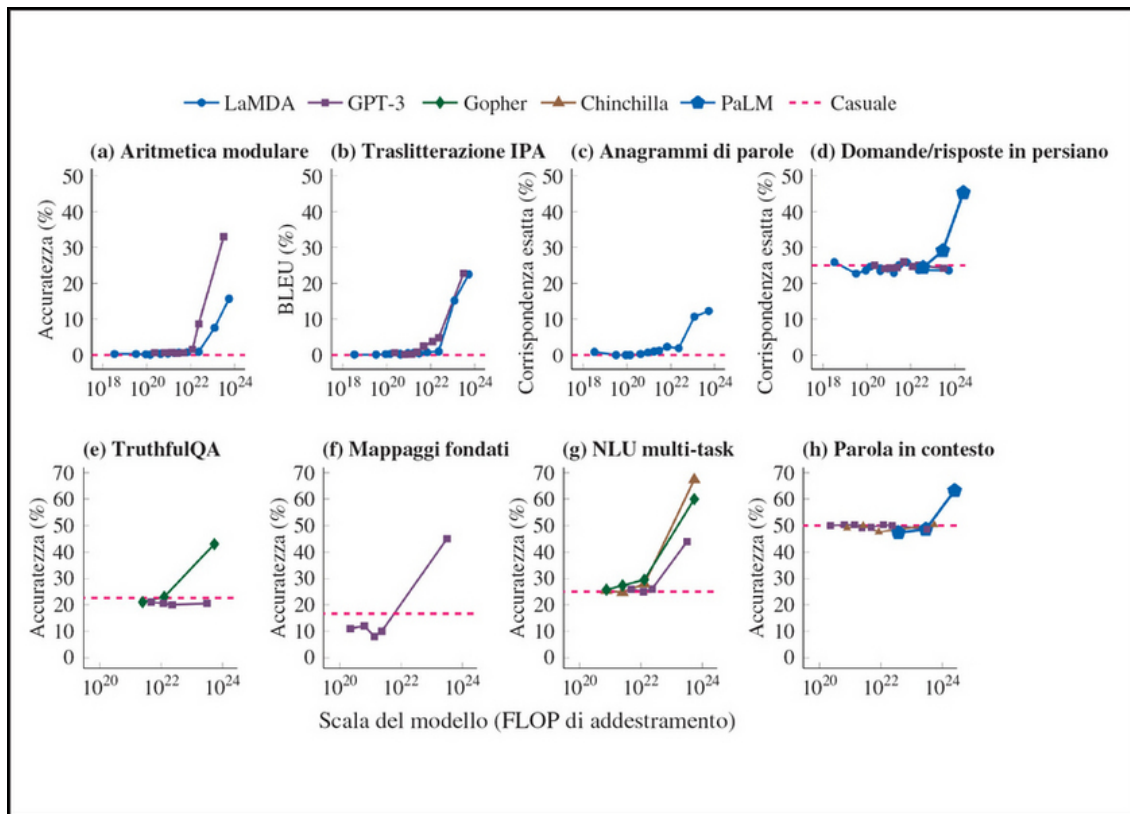


Figura 10.3 Figura 2 da Wei et al. (2022): illustrazione delle prestazioni, misurate in accuratezza, rispetto alla scala del modello (misurata in FLOP di addestramento) per otto diversi compiti. La figura mostra come le prestazioni passano da livelli casuali a significativamente sopra il caso dopo aver raggiunto una certa soglia di scala. Questi compiti mostrano esempi di abilità emergenti nell'ambito del prompting few-shot, che compaiono solo dopo un certo livello di addestramento.

Ogni grafico mostra le prestazioni del modello (misurate in accuratezza) in funzione della scala del modello (FLOP di addestramento, o operazioni in virgola mobile al secondo durante l'addestramento). Per esempio, guardando il compito di Aritmetica modulare, grandi modelli come GPT-3 e PaLM mostrano un significativo aumento delle prestazioni solo quando i loro FLOP di addestramento superano 10^{22} . Analogamente, per il compito NLU

multi-task, modelli come Chinchilla, Gopher e GPT-3 iniziano a migliorare solo una volta che i loro FLOP di addestramento superano la stessa soglia. Osserviamo lo stesso comportamento anche per il compito Parola in contesto.

Il punto chiave è che non solo abbiamo bisogno di modelli di grandi dimensioni, ma dobbiamo anche addestrarli per molte iterazioni per migliorare le loro prestazioni e sbloccare le abilità emergenti.

Tuttavia, è importante notare che anche dopo un ampio addestramento, i modelli di grandi dimensioni possono essere lenti nell'inferenza, a causa del loro enorme numero di parametri. Ciò crea la necessità di impiegare metodi per accelerare l'inferenza, come quelli descritti in questo capitolo.

Infine, sebbene non mostrato esplicitamente nelle figure, anche la quantità di dati di addestramento è un fattore critico nelle prestazioni del modello. I modelli di grandi dimensioni beneficiano di enormi quantità di dati da cui apprendere, migliorando ulteriormente le loro capacità.

Inferenza efficiente per i modelli profondi

Per sviluppare un'inferenza efficiente per modelli profondi, i lavori recenti si sono concentrati sulle tecniche di compressione e accelerazione del modello, come le seguenti.

1. *Knowledge distillation* (distillazione della conoscenza).
2. *Quantizzazione*.
3. *Parameter pruning* (potatura dei parametri).
4. *Neural Architecture Search*.

In questo capitolo, ci concentreremo solo sulle tecniche 1 e 2.

Rendere più veloce FinBERT

Distillazione della conoscenza (Knowledge Distillation)

La distillazione della conoscenza è una tecnica utilizzata per creare un modello più veloce (lo *studente*) – ovvero con un tempo di inferenza inferiore – che approssimi le prestazioni di un modello più lento ma più potente (il *docente*). In questo processo, il modello studente è per definizione più piccolo del modello docente, con meno parametri, cosa che lo rende più efficiente al momento dell’inferenza. La distillazione della conoscenza implica un processo di apprendimento per il modello studente, in cui non impara solo dai dati, come in un addestramento tipico, ma ha anche la conoscenza trasferitagli dal modello docente.

Nel nostro caso di studio, il modello docente che vogliamo approssimare è FinBERT; si veda Araci (2019). Per ricapitolare, FinBERT è una versione fine-tuned di BERT che è stata addestrata in modo specifico per la previsione dell’analisi del sentiment nel dominio finanziario.

Se la nostra unica preoccupazione fosse la velocità di inferenza, potremmo semplicemente addestrare il modello studente utilizzando i dati forniti (che, per esempio, potrebbero essere una combinazione di pre-addestramento sul corpus BERT seguito da un addestramento fine con i dati finanziari utilizzati da FinBERT). Tuttavia, poiché il modello studente è più piccolo del docente, è probabile che anche le sue prestazioni siano inferiori. La distillazione della conoscenza offre un approccio differente: implica non solo l’apprendimento dai dati ma anche l’allineamento del modello studente con certi aspetti del modello docente. Nella sua forma più semplice, questo allineamento

potrebbe coinvolgere la corrispondenza della distribuzione di probabilità che il modello docente assegna a un dato input, come descritto da Hinton et al. (2015) nel loro articolo sulla distillazione della conoscenza (sì, è lo stesso Hinton che ha condiviso il Premio Nobel per la Fisica 2024 con John Hopfield ed è considerato uno dei padri del deep learning, vedi

https://en.wikipedia.org/wiki/Geoffrey_Hinton).

Un'analogia può aiutare a chiarire questo processo. Pensate a come impariamo una materia a scuola: immaginiamo di avere la possibilità di studiare direttamente da un libro di testo o di frequentare lezioni tenute da un insegnante esperto in quella materia. La maggior parte delle volte, una combinazione di entrambe le cose – frequentare lezioni e leggere il libro di testo – porta a una comprensione migliore rispetto a uno dei due approcci, da solo. In questa analogia, il libro di testo rappresenta i dati utilizzati nell'addestramento, mentre il docente rappresenta il modello che guida il processo di apprendimento. Allo stesso modo, nella distillazione della conoscenza, il modello studente trae beneficio dall'apprendere sia dai dati sia dalla guida fornita dal modello docente.

Per permettere al modello studente di apprendere sia dai dati sia dal modello docente, la funzione di costo utilizzata durante l'addestramento deve tenere conto non solo di quanto il modello studente si adatta ai dati, ma anche di quanto corrisponde agli specifici aspetti del modello docente che vogliamo catturare.

Esploreremo come applicare la distillazione della conoscenza a FinBERT, mostrando come questa tecnica possa essere utilizzata per creare un modello più veloce senza sacrificare troppo le prestazioni.

Quale aspetto del modello docente vogliamo trasferire

Nella distillazione della conoscenza, ci sono vari aspetti del modello docente con cui il modello studente può cercare di allinearsi. Questi diversi aspetti sono noti nella letteratura sulla distillazione della conoscenza come diversi tipi di conoscenza. Per esempio, nella forma più basilare di distillazione della conoscenza, l'attenzione è sull'allineamento della distribuzione di probabilità che il modello docente assegna alle diverse classi, dato l'input, come nel caso di un compito di classificazione. Questo approccio è stato reso popolare da Hinton et al. (2015).

Tuttavia, ci sono altre forme di conoscenza che possono dover essere trasferite dal docente allo studente. Per esempio, le rappresentazioni intermedie calcolate dalla rete sono una potenziale fonte di conoscenza trasferibile. Alcuni metodi utilizzano le feature di attivazione dei layer intermedi nel modello docente per guidare l'apprendimento del modello studente, come descritto da Romero et al. (2015) nel loro lavoro sulle FitNet. Inoltre, anche i parametri o i pesi del modello docente possono essere una diversa fonte di conoscenza.

Questi diversi tipi di conoscenza da trasferire rientrano in tre categorie: conoscenza basata sulla risposta (*response-based*), basata sulle feature (*feature-based*) e basata sulle relazioni (*relation-based*). Un'ottima panoramica di questa classificazione si trova nell'eccellente lavoro sulla distillazione della conoscenza di Gou et al. (2021).

Per lo scopo di questo libro e nel nostro caso di studio, ci concentreremo sulla conoscenza basata sulla risposta, che è il metodo più semplice, ma altamente efficace, per la compressione dei modelli.

Conoscenza basata sulla risposta (response-based knowledge)

La conoscenza basata sulla risposta si riferisce a uno scenario in cui il modello studente tenta di imitare o di rispondere come l'ultimo layer del modello docente, il che significa che cerchiamo di far corrispondere la sua previsione finale.

Nei tipici compiti di classificazione, la forma più nota di conoscenza basata sulla risposta è chiamata *soft target*, come descritto in Hinton et al. (2015). In questo contesto, i soft target si riferiscono alla distribuzione di probabilità allocata dal modello docente sulle diverse classi per un dato input. Questo contrasta con la verità “ground truth” o *hard target*, dove la massa di probabilità è completamente allocata alla classe corretta.

Nelle reti neurali per compiti di classificazione, le probabilità delle classi sono solitamente generate applicando una funzione “softmax” ai logit (gli output dell'ultimo layer pienamente connesso della rete). Seguendo la notazione introdotta in Hinton et al. (2015), denotiamo i logit di output come il vettore $\mathbf{z}$, la cui dimensione dipende dal numero di classi specificate dal problema di classificazione. I logit sono gli output di una rete neurale prima che venga applicata la funzione di attivazione softmax. Sono punteggi dell'appartenenza dell'input a una certa classe. Ricordare la discussione sulla regressione logistica nel Capitolo 3. La distribuzione di probabilità viene quindi calcolata applicando la funzione softmax a $\mathbf{z}$:

$$p(z_i, T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

dove z_i è l'output corrispondente alla classe i -esima, e T è un parametro noto come *temperatura*. Tipicamente, nei problemi di classificazione, T è impostata a 1, ma nella distillazione della

conoscenza, l'uso di valori più grandi di T tende a produrre una distribuzione di probabilità più “sfumata” sulle classi.

Per esempio, nella Figura 10.4, mostriamo come cambia l'output della funzione softmax applicata al vettore di logit $\mathbf{z} = [30, 20, 10]^T$ all'aumentare della temperatura. A una temperatura pari a 1, l'output si avvicina a un hard target, con quasi tutta la massa di probabilità allocata a una singola classe. All'aumentare della temperatura, la distribuzione di probabilità diventa progressivamente più sfumata. Per temperature molto grandi – come nel nostro caso $T = 1.000$ – la distribuzione si avvicina a una distribuzione uniforme su tutte le classi. I fisici riconosceranno la distribuzione di Boltzman, che fornisce la probabilità che un sistema si trovi in un certo stato in funzione dell'energia di quello stato e della temperatura del sistema.

Come abbiamo già detto, l'obiettivo nella distillazione della conoscenza è che il modello studente impari sia dai dati sia dal modello docente (in particolare nella conoscenza basata sulla risposta dei soft target) dai soft target prodotti dal docente. Per raggiungere questo obiettivo, abbiamo bisogno di fornire allo studente una funzione obiettivo che tenga conto di quanto bene individua la distribuzione empirica dei dati (che tipicamente, nei problemi di classificazione, è misurata dalla loss di entropia incrociata) e di quanto bene corrisponde ai soft target del docente. Un approccio consiste nell'utilizzare una media ponderata di entrambi gli obiettivi:

$$\mathcal{L}_{\text{student}} = \alpha \mathcal{L}_{\text{CE}}(y, p(\mathbf{z}_s, T = 1)) + (1 - \alpha) \mathcal{L}_{\text{KD}}(p(\mathbf{z}_t, T), p(\mathbf{z}_s, T))$$

dove $\mathcal{L}_{\text{student}}$ rappresenta la loss totale per lo studente, $\mathcal{L}_{\text{CE}}$ rappresenta la loss di entropia incrociata tra il modello studente e l'etichetta di verità y , α è l'iperparametro di peso tra 0 e 1, e $\mathcal{L}_{\text{KD}}$ è la loss di distillazione, che misura quanto la distribuzione dello studente $p(\mathbf{z}_s, T)$ si allinea con la distribuzione del docente $p(\mathbf{z}_t, T)$.

La loss di distillazione della conoscenza è tipicamente proporzionale alla divergenza di Kullback-Leibler (KL)

$$\mathcal{L}_{\text{KD}}(p(\mathbf{z}_t, T), p(\mathbf{z}_s, T)) = T^2 D_{\text{KL}}(p(\mathbf{z}_t, T), p(\mathbf{z}_s, T))$$

dove T è la temperatura e D_{KL} è la divergenza KL, che misura la differenza tra le distribuzioni del docente e dello studente. Il fattore T^2 assicura che i contributi relativi degli hard target e dei soft target rimangano bilanciati. Per spiegazioni più dettagliate sul fattore di proporzionalità, vi incoraggiamo a consultare Hinton et al. (2015)

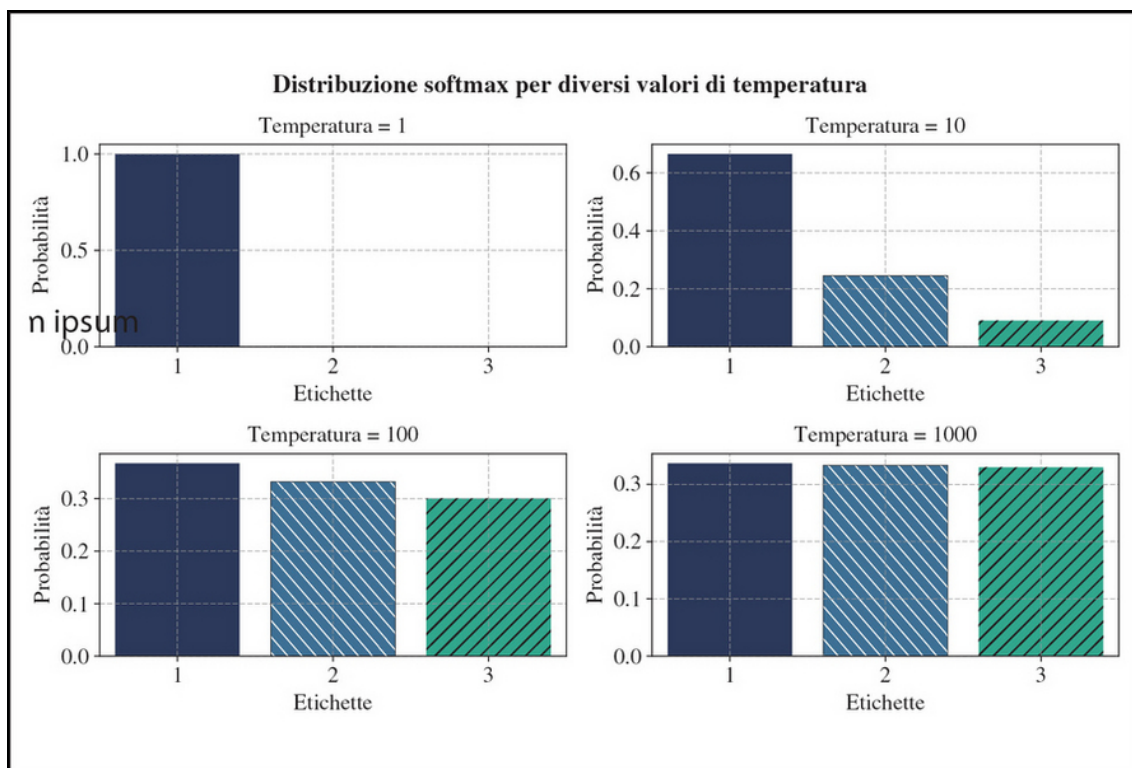


Figura 10.4 Istogrammi che rappresentano la distribuzione softmax.

Dettagli di implementazione

Per eseguire la distillazione della conoscenza utilizzeremo la libreria Python transformers di Hugging Face (<https://huggingface.co>) grazie alla sua potente API e utility, che semplifica notevolmente

l'addestramento e l'inferenza con i modelli transformer. Nel repository che accompagna questo libro, troverete un breve tutorial che tratta le basi di questa libreria, per fornirvi le conoscenze necessarie per comprendere le componenti rilevanti discusse nel libro.

Nei frammenti di codice forniti di seguito, ci concentriamo sui passaggi più importanti per implementare la distillazione della conoscenza. Molti passaggi, come la creazione del dataset, la tokenizzazione, la divisione addestramento-test, tra gli altri, sono comuni nei flussi di lavoro del machine learning e dell'elaborazione del linguaggio naturale e non specifici di questo progetto, quindi non è il caso di descriverli. Tuttavia, nel notebook di questo capitolo potete trovare il processo completo per costruire questo caso d'uso.

Il nostro obiettivo è quello di trovare un equilibrio tra teoria e implementazione specifica, per permettervi di comprendere i concetti di base dell'argomento e di applicarli utilizzando anche altre librerie, come PyTorch Lightning (<https://lightning.ai/docs/pytorch/stable>), o perfino implementando tutto in Pytorch o TensorFlow. Speriamo che questo approccio vi dia una solida comprensione della teoria, mentre fornisce esempi di implementazione dettagliati nel notebook, che potrà essere aggiornato man mano che si renderanno disponibili nuovi strumenti.

Ora che sappiamo che cos'è la loss dello studente – una combinazione lineare della loss di entropia incrociata (comunemente usata per compiti di classificazione) e della loss di distillazione della conoscenza (proporzionale alla divergenza KL tra le distribuzioni del docente e dello studente) – possiamo procedere a implementarla.

Per le procedure di addestramento standard, utilizziamo tipicamente la classe `Trainer` della libreria `transformers`, che fornisce un'API per addestrare modelli in PyTorch, offrendo molte

funzionalità pronte all'uso, come l'addestramento distribuito su più GPU/TPU e così via. Maggiori informazioni possono essere trovate nella documentazione della libreria

https://huggingface.co/docs/transformers/v4.15.0/main_classes/trainer.

Poiché il nostro obiettivo è quello di addestrare il modello utilizzando la loss di distillazione $\mathcal{L}_{\text{student}}$ e la classe `Trainer` non supporta questa loss, dobbiamo estenderne le capacità. Fortunatamente, la cosa è molto semplice.

Estendiamo la classe `Trainer` standard per creare un `DistillationTrainer`, come suggerito in Tunstall et al. (2022), che supporti la funzione loss dello studente necessaria per la distillazione della conoscenza. Il `DistillationTrainer` può essere istanziato come mostrato nel seguente codice:

```
distilbert_trainer = DistillationTrainer(  
    model_init=student_init,  
    teacher_model=teacher_model,  
    args=student_training_args  
    train_dataset=YOUR_TRAINING_SET,  
    eval_dataset=YOUR_EVALUATION_SET,  
    compute_metrics=compute_metrics,  
    tokenizer=student_tokenizer  
)
```

L'oggetto `DistillationTrainer` accetta i seguenti input.

- `student_init`: una funzione utilizzata per l'inizializzazione del modello studente.
- `teacher_model`: il modello docente utilizzato per la distillazione.
- `student_training_args`: un'istanza della classe `TrainingArguments`.
- `YOUR_TRAINING_SET`: un `torch.utils.data.Dataset`, `torch.utils.data.IterableDataset` o `datasets.Dataset`, il dataset utilizzato per addestrare il modello.
- `YOUR_EVALUATION_SET`: un `torch.utils.data.Dataset`, `torch.utils.data.IterableDataset` o `datasets.Dataset`, il dataset

utilizzato per valutare il modello.

- `compute_metrics`: una funzione utilizzata per le metriche sul set di valutazione, come precisione, recall, accuratezza e così via.
- `student_tokenizer`: il tokenizer utilizzato per pre-elaborare i dati.

Una differenza chiave, dal punto di vista degli argomenti di input, tra il `DistillationTrainer` e la classe `Trainer` standard è l'inclusione dell'argomento `teacher_model`, che è essenziale per la distillazione della conoscenza.

Dal punto di vista funzionale, `DistillationTrainer` deve calcolare la corretta funzione loss. Per fare questo, ha bisogno di accedere agli iperparametri α e temperatura T . Ecco l'implementazione del metodo `compute_loss`:

```
import torch.nn as nn
import torch.nn.functional as F
from transformers import Trainer
class DistillationTrainer(Trainer):
    def __init__(self, *args, teacher_model=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.teacher_model = teacher_model
    def compute_loss(self, model, inputs, return_outputs=False):
        device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
        inputs = inputs.to(device)
        outputs_stu = model(**inputs)
        # Estrai la loss di entropia incrociata e i logit dallo studente
        loss_ce = outputs_stu.loss
        logits_stu = outputs_stu.logits
        # Estrai i logit dal docente
        with torch.no_grad():
            outputs_tea = self.teacher_model(**inputs)
            logits_tea = outputs_tea.logits
        # Ammorbidisci le probabilità e calcola la loss di distillazione
        loss_fct = nn.KLDivLoss(reduction="batchmean")
        loss_kd = self.args.temperature ** 2 * loss_fct(
            F.log_softmax(logits_stu / self.args.temperature, dim=-1),
            F.softmax(logits_tea / self.args.temperature, dim=-1)
        )
        # Restituisci la loss ponderata dello studente
        loss = self.args.alpha * loss_ce + (1. - self.args.alpha) *
loss_kd
        return (loss, outputs_stu) if return_outputs else loss
```

In questa classe `Trainer` personalizzata:

- salviamo il modello docente nel metodo `__init__`;
- calcoliamo la loss dello studente nel metodo `compute_loss`, dove:
 - `loss_ce` corrisponde alla loss di entropia incrociata;
 - `loss_fct` è un'istanza dell'oggetto `nn.KLDivLoss`, che calcola la loss per divergenza KL;
 - `loss_kd` corrisponde alla loss di distillazione della conoscenza tra le distribuzioni del docente e dello studente;
 - `loss` è la combinazione lineare di entrambe le loss.

Come mostrato nel codice, la classe accede a α e T come `self.args.alpha` e `self.args.temperature` attraverso l'oggetto `student_training_args`. Per renderli disponibili durante

l'addestramento, estendiamo la classe `TrainingArguments` come segue:

```
from transformers import TrainingArguments
class DistillationTrainingArguments(TrainingArguments):
    def __init__(self, *args, alpha, temperature, **kwargs):
        super().__init__(*args, **kwargs)
        self.alpha = alpha
        self.temperature = temperature
```

Dopo aver creato il trainer personalizzato e gli argomenti di addestramento personalizzati, possiamo addestrare il nostro modello studente utilizzando le funzionalità fornite dalla libreria `transformers` semplicemente richiamando `distilbert_trainer.train()`. Come menzionato in precedenza, il processo dettagliato per la distillazione di FinBERT è fornito nel notebook di accompagnamento, mentre questo paragrafo si concentra solo sui concetti chiave e sui dettagli implementativi.

Risultati del caso di studio. Distillazione di FinBERT

Nel documento originale di Araci (2019) su FinBERT, gli autori hanno utilizzato due diversi dataset per il fine-tuning del modello. Uno di questi dataset è *TRC2-financial*, che è stato impiegato per migliorare la pertinenza del corpus rispetto alle parole chiave finanziarie. Potete ottenere questo dataset per scopi di ricerca facendo domanda qui: <https://trec.nist.gov/data/reuters/reuters.html>.

Il secondo dataset, *Financial PhraseBank* creato da Malo et al. (2014), è il dataset principale utilizzato nel documento FinBERT per il compito specifico dell'analisi del sentiment. Può essere ottenuto gratuitamente qui:

https://www.researchgate.net/publication/251231364_FinancialPhraseBank-v10. Nei nostri esperimenti, questo è il dataset che abbiamo utilizzato per eseguire la distillazione della conoscenza su FinBERT.

Per garantire un confronto equo, abbiamo utilizzato la stessa suddivisione addestramento-test specificata dagli autori, come indicato nel loro repository GitHub, *FinBERT: Financial Sentiment Analysis with BERT* (<https://github.com/ProsusAI/finBERT>). Ciò ci ha permesso di ottenere metriche di valutazione paragonabili a quelle del documento originale. Le metriche di valutazione che abbiamo utilizzato per valutare le prestazioni sono le stesse dello studio originale, ovvero accuratezza e media macro F1.

Il nostro primo passo è stato riprodurre i risultati riportati nel documento originale, cosa che abbiamo fatto con successo sulla parte di test del dataset *TRC2-financial*.

Per il modello studente, abbiamo selezionato `distilbert/distilbert-base-uncased`, fornito da Hugging Face all'indirizzo

<https://huggingface.co/distilbert/distilbert-base-uncased>. La libreria

`transformers` facilita il download dei modelli da HuggingFace.

Ulteriori informazioni possono essere trovate nel notebook di accompagnamento di questo capitolo e nel tutorial di HuggingFace nel repository del libro. In termini di prestazioni, il nostro *DistilledFinBERT* supera leggermente il modello docente, FinBERT, il che è un risultato eccellente. I risultati dettagliati sono riportati nella Tabella 10.1.

Tabella 10.1 Metriche di prestazione dei modelli docente vs. studente.

Modello	Precisione	Recall	F1-Score
FinBERT	0.85	0.84	0.84
DistilledFinBERT	0.86	0.85	0.85

Dato che siamo riusciti a mantenere o addirittura migliorare le prestazioni, cosa dire in termini di velocità di inferenza? I risultati sono riportati nella Tabella 10.2.

Tabella 10.2 Velocità di inferenza dei modelli docente vs. studente.

Modello	Latenza media (ms)
FinBERT	0.034
DistilledFinBERT	0.016

DistilledFinBERT raggiunge una velocità di inferenza 2.125 volte migliore del FinBERT originale. In conclusione, il nostro DistilledFinBERT non solo eguaglia le prestazioni del modello originale sullo stesso set di test, ma offre anche un'inferenza significativamente più veloce, migliorando quindi anche la distribuzione.

Quantizzazione del modello

La quantizzazione si riferisce al processo di riduzione dell'insieme di valori che una variabile può assumere, in modo che la variabile possa essere rappresentata utilizzando meno bit, riducendo così la sua precisione. La domanda chiave, alla base della quantizzazione è: le reti neurali profonde possono funzionare efficacemente pur con una precisione inferiore?

Nel deep learning, siamo particolarmente interessati ad applicare la quantizzazione ai pesi, ai bias e alle attivazioni dei nostri modelli. La riduzione della precisione in bit di questi elementi offre vantaggi in termini di spazio di memoria, velocità computazionale e consumo energetico. L'hardware attuale è dotato di istruzioni specializzate per l'aritmetica intera, che migliorano ulteriormente la velocità computazionale quando si opera con una precisione inferiore. Tuttavia, questa riduzione di precisione può avvenire a scapito dell'accuratezza, a causa della potenziale perdita di informazioni.

L'addestramento e l'inferenza nelle reti neurali profonde hanno requisiti di precisione differenti. Durante l'addestramento – un processo altamente dinamico che coinvolge un'ampia gamma di valori di peso – è importante che vi sia un'ampia gamma dinamica. Al contrario, spesso l'inferenza privilegia la precisione su una piccola gamma dinamica, poiché il modello fa previsioni basandosi su pesi pre-addestrati che spesso si concentrano attorno a un determinato valore. Comprendendo i diversi requisiti di addestramento e inferenza, possiamo progettare tipi di dati ottimizzati per ogni fase.

Al momento della stesura di questo testo, esistono due tecniche di quantizzazione: K-Means e lineare. In questo paragrafo, ci concentreremo sulla quantizzazione lineare e vedremo come può essere applicata per ottenere un'inferenza efficiente del modello.

Quantizzazione lineare

La quantizzazione lineare applica una mappatura affine dallo spazio degli interi allo spazio dei numeri reali (o spazio in virgola mobile) utilizzando la seguente formula: $r = S(q - Z)$, come descritto in Jacob et al. (2017). Qui:

- S è la scala, un valore in virgola mobile;
- Z è il punto zero, un intero che rappresenta l'offset o il bias;
- q è il numero quantizzato, un intero;
- r è il valore reale, un numero in virgola mobile.

La formula, di ricostruzione, mappa i valori quantizzati q nuovamente ai valori reali r . In pratica, però, di solito partiamo dal valore reale r e dobbiamo trovare la sua controparte quantizzata, q .

Possiamo vedere questa equazione come una mappatura dall'intervallo intero $[q_{\min}, q_{\max}]$ sull'intervallo in virgola mobile $[r_{\min}, r_{\max}]$. L'intervallo della rappresentazione intera dipende dal numero di bit utilizzati. Per esempio, nelle rappresentazioni intere con segno, l'intervallo di valori per diverse larghezze di bit è mostrato nella Tabella 10.3:

Tabella 10.3 Intervallo di interi per diverse larghezze di bit.

Lunghezza in bit	$q_{\min}$	$q_{\max}$
2	-1	1
3	-4	3
4	-8	7
N	-2^{N-1}	$2^{N-1} - 1$

L'intervallo in virgola mobile $[r_{\min}, r_{\max}]$ è determinato dalle statistiche dei valori reali in input da quantizzare. Per determinare la

scala S , imponiamo che $q_{\min}$ mappi su $r_{\min}$ e che $q_{\max}$ mappi su $r_{\max}$, risolvendo il seguente sistema di equazioni:

$$r_{\max} = S(q_{\max} - Z)$$

$$r_{\min} = S(q_{\min} - Z)$$

Sottraendo queste due equazioni otteniamo il valore di S :

$$S = \frac{r_{\max} - r_{\min}}{q_{\max} - q_{\min}}$$

Il punto zero Z si trova risolvendo per Z usando $r_{\min} = S(q_{\min} - Z)$, che risulta in:

$$Z = \text{round}\left(q_{\min} - \frac{r_{\min}}{S}\right)$$

Poiché Z è un intero, lo arrotondiamo. Per calcolare il valore quantizzato q , usiamo:

$$q = r/S + Z$$

Infine, q viene costretto nell'intervallo $[q_{\min}, q_{\max}]$ e arrotondato all'intero più vicino. Sebbene esistano diversi metodi di arrotondamento, questo esempio utilizza il caso più semplice, ma esistono anche metodi di arrotondamento adattativi.

Esempio di quantizzazione lineare

Illustriamo la quantizzazione lineare con un piccolo esempio per dare un'idea delle operazioni coinvolte. Consideriamo una matrice di pesi che vogliamo quantizzare usando 2 bit:

```
weights = np.array([
    [-1.0856306  0.99734545  0.2829785 ]
    [-1.50629471 -0.57860025  1.65143654]
    [-2.42667924 -0.42891263  1.26593626]
])
```

Innanzitutto, calcoliamo la scala S . L'intervallo dinamico della matrice dei pesi, definito dai suoi valori minimo e massimo, è $r_{\min} = -2.42667924$ e $r_{\max} = 1.65143654$.

Utilizzando una rappresentazione a 2 bit, l'intervallo intero è $[-2, 1]$. Da questo, calcoliamo la scala come $S = 1.35937$. Possiamo quindi trovare il punto zero $Z = 0$. Infine, calcoliamo i pesi quantizzati utilizzando l'equazione fornita:

```
quantized_weights = (weights / scale + zero_point).clamp(qmin,
qmax).round()
>>> quantized_weights
np.array([
  [-1.  1.  0.]
  [-1. -0.  1.]
  [-2. -0.  1.]]
])
```

Se tentiamo di ricostruire la matrice originale dalla versione quantizzata, otteniamo:

```
>>> reconstructed_weights
np.array([
  [-1.3594  1.3594  0.],
  [-1.3594 -0.  1.3594],
  [-2.7187 -0.  1.3594],
])
```

L'errore di ricostruzione è la differenza tra i pesi originali e quelli ricostruiti:

```
>>> reconstruction_error
np.array([
  [0.2737 -0.362  0.283 ],
  [-0.1469 -0.5786  0.2921],
  [0.2921 -0.4289 -0.0934]
])
```

Quantizzazione di un layer di attenzione nel Distilled FinBERT

Ora, applichiamo questo processo a uno dei layer di attenzione nel nostro modello *Distilled FinBERT* utilizzando 8 bit. La Figura 10.5 è

una distribuzione dei parametri del layer, seguendo una configurazione simile a quella riportata in Tunstall et al. (2022).

Come mostra la Figura 10.5, i pesi sono concentrati all'interno di un intervallo ristretto di valori, mostrando una gamma dinamica limitata. Poiché stiamo utilizzando interi con segno a 8 bit, l'intervallo è $[-127, 128]$. Il fattore di scala e il punto zero vengono calcolati in modo simile all'esempio precedente.

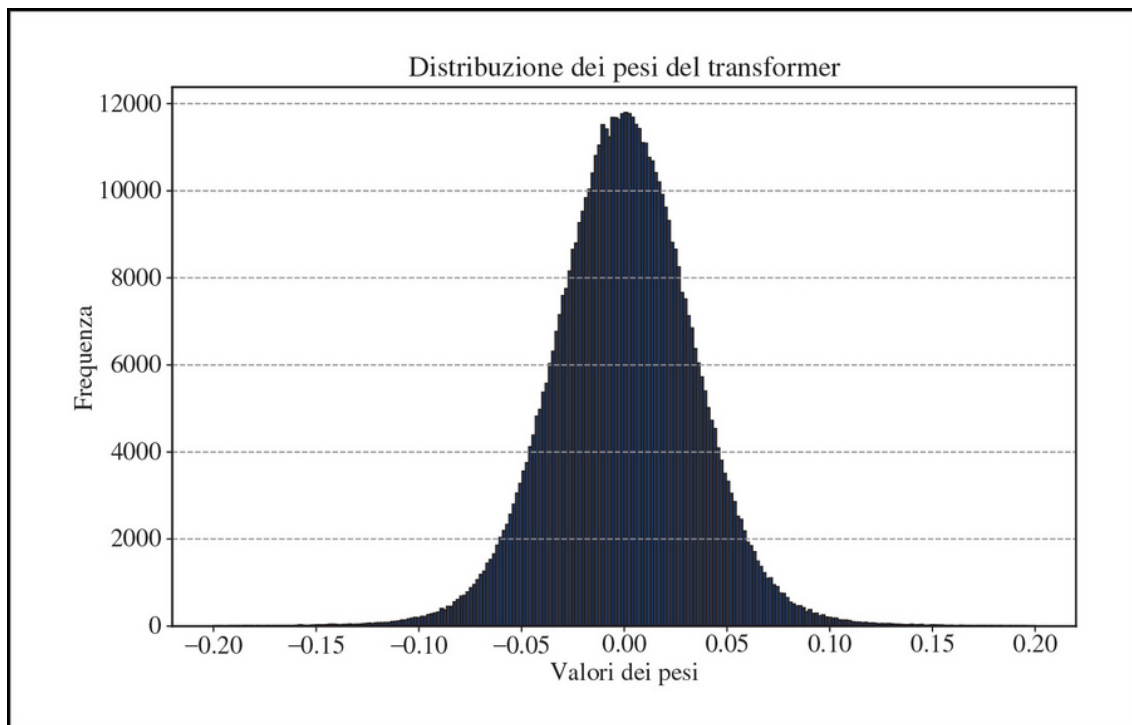


Figura 10.5 Grafico della distribuzione dei pesi.

Successivamente, calcoliamo i pesi quantizzati utilizzando la formula $q = r/S + Z$, vincoliamo i valori, li arrotondiamo all'intero più vicino e li memorizziamo nel tipo di dati `torch.int8`:

```
manually_quantized_weights = (weights / scale + zero_point).clamp(qmin,
qmax).round().char()
>>> manually_quantized_weights
tensor([[ -5,  -8,  0, ..., -6,  -3,  8],
        [ 8,  3,  1, ..., -4,  7,  1],
        [-9, -5,  5, ...,  0,  6, -3],
        ...,
        [ 5,  0, 13, ...,  0,  6, -1],
```

```

        [0, -2, -12, ..., 12, -8, -13],
        [-13, -1, -9, ..., 8, 2, -2]],
        dtype=torch.int8
    )

```

Torch fornisce anche una funzione integrata,

`torch.quantize_per_tensor`, per svolgere l'operazione. Ecco il risultato per il confronto:

```

from torch import quantize_per_tensor
# torch ha una funzione che esegue questa operazione per noi
dtype = torch.qint8
quantized_weights = quantize_per_tensor(weights, scale, zero_point,
dtype)
quantized_weights.int_repr()
tensor([[ -5,  -8,  0, ...,  -6,  -3,  8],
        [ 8,  3,  1, ...,  -4,  7,  1],
        [-9, -5,  5, ...,  0,  6, -3],
        ...,
        [ 5,  0, 13, ...,  0,  6, -1],
        [ 0, -2, -12, ..., 12, -8, -13],
        [-13, -1, -9, ...,  8,  2, -2]],
        dtype=torch.int8
)

```

Infine, confrontiamo i risultati:

```

# assicuriamoci che siano equivalenti
assert abs(quantized_weights.int_repr() -
manually_quantized_weights).sum().item() <= 1e-5

```

La Figura 10.6 mostra la distribuzione dei pesi quantizzati.

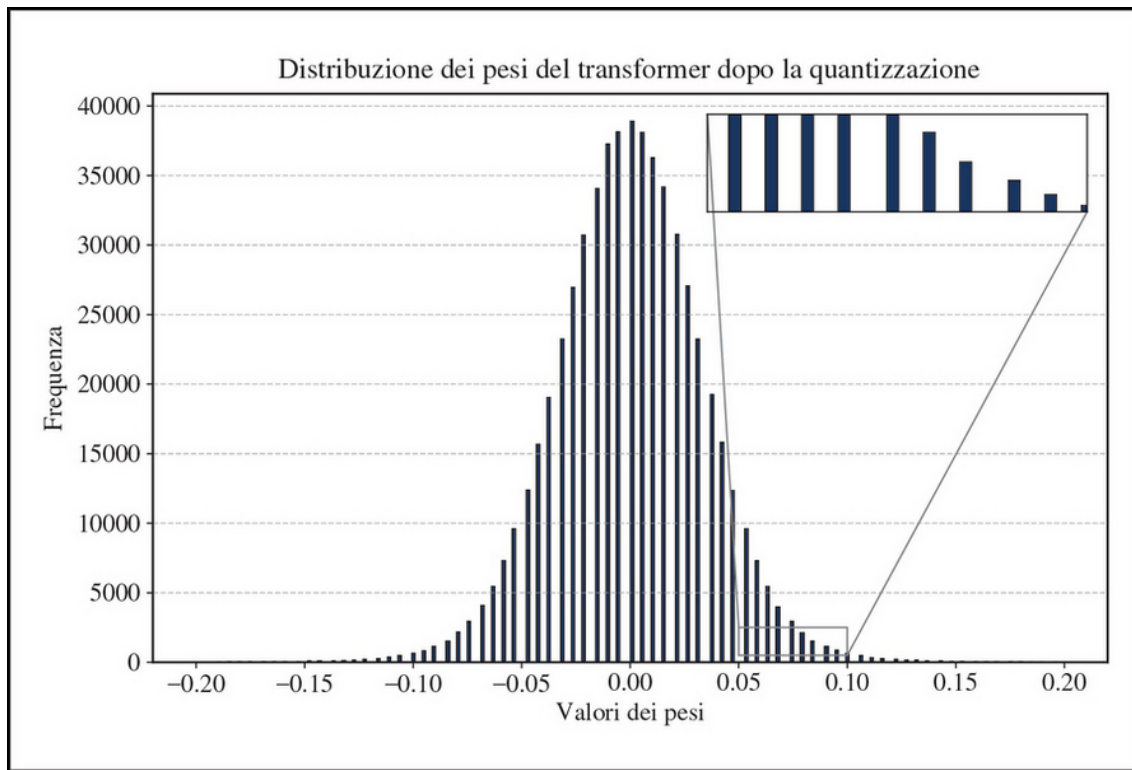


Figura 10.6 Grafico della distribuzione dei pesi quantizzati.

Per verificare il guadagno in termini di efficienza nell'uso della memoria passando da FP32 a INT8, possiamo usare direttamente il codice:

```
import sys
# compressione di 4 volte!
sys.getsizeof(weights.storage()) /
sys.getsizeof(quantized_weights.storage())
>>> 3.999755879241598
```

Misuriamo anche l'aumento di velocità nell'eseguire l'operazione `weights*weights`, come mostrato nella Tabella 10.4.

Tabella 10.4 Tabella delle prestazioni: aumenti di velocità dovuti alla quantizzazione.

Tipo di dati	Tempo (s)
float32	0.006926

int8	0.000266
Accelerazione	26.06x

Come vediamo, la quantizzazione fornisce miglioramenti significativi sia nell'impronta di memoria sia nella velocità di calcolo. Per maggiori dettagli, potete fare riferimento al notebook associato a questo capitolo.

Risultati sperimentali con la quantizzazione lineare su Distilled FinBERT

In questo paragrafo, applichiamo la quantizzazione lineare all'intero modello *Distilled FinBERT* e valutiamo i miglioramenti in termini di spazio di memoria e velocità computazionale, insieme a qualsiasi potenziale perdita di accuratezza dovuta alla rappresentazione dei pesi del modello con una precisione inferiore (come nota a margine, in questi esperimenti, la quantizzazione viene applicata solo durante il processo di inferenza). Dato che al momento della stesura PyTorch supporta la quantizzazione lineare solo su CPU, i risultati qui presentati sono per l'inferenza basata su CPU. I risultati sono riassunti nella Tabella 10.5.

Tabella 10.5 Velocità di inferenza dopo la quantizzazione.

Modello	Accuratezza (%)	Latenza media per frase (s)
FinBERT	83.9175	0.1533
DistilledFinBERT	85.3608	0.1060
QuantizedDistilledFinBERT	83.7113	0.0458

Come vediamo, il processo di quantizzazione introduce una certa perdita di accuratezza: le prestazioni del modello scendono

dall'85,36% nella versione distillata all'83,71% nella versione quantizzata. Tuttavia, il *Distilled FinBERT* quantizzato corrisponde ancora all'accuratezza del FinBERT originale (83,92%), con solo una leggera differenza (0.2%). Tuttavia, il modello quantizzato raggiunge un significativo incremento di velocità, eseguendo l'inferenza 3,35 volte più velocemente del modello originale.

Questo risultato dimostra la potenza della ricerca dell'efficienza dell'inferenza: mantenere prestazioni quasi identiche fornendo al contempo previsioni significativamente più veloci. I dettagli su come applicare la quantizzazione al modello sono forniti nel notebook del capitolo.

Personalizzare un LLM: adattare i modelli a esigenze specifiche

Gli LLM sono strumenti incredibilmente potenti e che possono accelerare molti tipi di compiti. Nei capitoli precedenti, abbiamo mostrato come possono aiutare a generare codice per le strategie di trading o assistere nello sviluppo di idee di trading. Tuttavia, come abbiamo visto, per ottenere risposte soddisfacenti spesso occorre ricorrere a un'intensa ingegneria dei prompt e all'aggiunta di maggior contesto, per guidare il modello nel produrre output rilevanti.

A volte, però, potreste avere già molte informazioni utili, come una base di codice o una documentazione interna, che vorreste che l'LLM potesse consultare direttamente. Personalizzando l'LLM per lavorare con questa conoscenza, potete ridurre l'impegno sui prompt e ottenere risposte molto più adatte alle vostre specifiche esigenze. Per esempio, ciò consentirebbe agli ingegneri del software, ai ricercatori o ai trader discrezionali di interagire con una base di conoscenza in

modo più efficace, ottenendo risposte più pertinenti e allineate con il loro lavoro.

Al momento della stesura, ci sono due modi per personalizzare un LLM per fargli utilizzare i propri dati. Uno implica il fine-tuning, noto anche come post-addestramento che, come abbiamo visto, comporta il riaddestramento del modello con i propri dati per renderlo più efficace per i compiti che gli sottoponiamo. L'altro approccio non richiede il riaddestramento dell'LLM ma arricchisce il prompt aggiungendo più informazioni come parte dell'input, per consentire all'LLM di utilizzare tali dati per rispondere a domande specifiche.

Un metodo molto in voga in questa seconda categoria è la RAG (*Retrieval Augmented Generation*), dove le informazioni contestuali vengono recuperate automaticamente da un database esterno e aggiunte al prompt dell'utente, per aiutare l'LLM a generare risposte più rilevanti o “fondate”.

Le informazioni contestuali vengono recuperate dinamicamente dal database misurando la similarità tra il prompt di input dell'utente e i file del database. L'idea di fondo consiste nel rappresentare sia i file del database sia il prompt dell'utente come oggetti matematici, come vettori (embedding). Una volta ottenute queste rappresentazioni, può essere applicata una misura di similarità per identificare e recuperare i file più rilevanti per il prompt di input. Per maggiori informazioni sulla RAG, si rimanda all'articolo del blog di OpenAI *Retrieval-augmented Generation (RAG) and Semantic Search for GPTs* all'indirizzo <https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts>.

Nel prossimo paragrafo, forniremo una breve panoramica delle tecniche per il fine-tuning (o post-addestramento) per personalizzare gli LLM. Anche se non entreremo nei dettagli tecnici – cosa che

andrebbe ben oltre gli scopi di questo libro – tratteremo i concetti chiave e ci concentreremo su una tecnica per costruire un esempio pratico che poi potrete provare voi stessi.

Tecniche di fine-tuning

Fine-tuning tradizionale

Il fine-tuning implica il riaddestramento di tutti i parametri del modello, cosa che può essere computazionalmente costosa e richiedere tempo, in base alla scala dei modelli. Inoltre, comporta il rischio dell'oblio catastrofico: il modello perde una parte considerevole della conoscenza acquisita durante il pre-addestramento, portando a un degrado della performance nella generalizzazione.

Per affrontare queste problematiche, il mondo accademico e l'industria hanno sviluppato tecniche che implicano l'adattamento solo di un sottoinsieme dei parametri di un modello, durante l'addestramento. Queste tecniche rientrano nella categoria del PEFT (*Parameter-Efficient Fine-Tuning*), rendendo i modelli più veloci da adattare, più efficienti in termini di memoria e capaci di raggiungere performance paragonabili o addirittura superiori rispetto al fine-tuning tradizionale.

PEFT

Il PEFT (*Parameter-Efficient Fine-Tuning*) si concentra sull'adattamento di un sottoinsieme dei parametri del modello durante il fine-tuning, il che riduce i requisiti computazionali, le esigenze di archiviazione e il tempo di addestramento rispetto al fine-tuning tradizionale. Di seguito sono riportati alcuni metodi di PEFT.

BitFit

BitFit aggiorna solo i termini di bias della rete piuttosto che tutti i parametri del modello; vedi Zaken et al. (2021). Gli autori hanno dimostrato che l'applicazione di BitFit ad architetture simili a BERT – modelli che abbiamo esplorato in questo e nel precedente capitolo – offre performance paragonabili al fine-tuning completo su dataset piccoli e medi. Cosa degna di nota, per modelli come BERT-Base e BERT-Large, i termini di bias rappresentano meno dello 0.1% dei parametri del modello, eppure un intervento su questa piccola percentuale dei parametri sembra essere sufficiente per tali dataset. Per maggiori dettagli, si rimanda a Zaken et al. (2021).

Adapter

Gli Adapter (Houlsby et al., 2019) introducono nella rete dei layer addestrabili. Durante l'adattamento, vengono aggiornati solo questi layer appena aggiunti, mentre i parametri originali del modello rimangono congelati. Gli autori hanno dimostrato che gli Adapter offrono performance quasi ottimali (al momento della pubblicazione). Tuttavia, l'aggiunta di nuovi layer alla rete aumenta sia i requisiti di archiviazione sia la latenza di inferenza. Maggiori dettagli possono essere trovati in Houlsby et al. (2019).

Prompt-tuning

Il Prompt-tuning implica l'arricchimento del prompt di input con un prompt informativo, per aiutare l'LLM a comportarsi bene sul successivo compito, come calcolare il sentiment di un testo, vedi l'articolo *The Power of Scale for Parameter-efficient Prompt Tuning* di Lester et al. (2021). Gli autori di questo articolo mostrano che il Prompt-tuning raggiunge un'accuratezza paragonabile al fine-tuning tradizionale, man mano che aumentano le dimensioni del modello.

Un'estensione del Prompt-tuning è il Prefix-tuning, introdotto nell'articolo *Prefix-tuning: Optimizing Continuous Prompts for Generation* di Li e Liang (2021), che aggiunge prompt informativi a ogni layer del transformer. Tuttavia, questo metodo introduce un'ulteriore latenza di inferenza a causa dei prompt informativi e riduce la lunghezza disponibile del prompt di input.

LoRA

LoRA (*Low-Rank Adaptation*), vedi Hu et al. (2021), è simile agli Adapter, nel senso che introduce nuovi layer apprendibili, ma con la differenza che lo fa in un ramo parallelo (vedi Figura 10.7), il che fornisce il vantaggio di non aumentare la latenza di inferenza. Come gli Adapter, LoRA mantiene fissi i parametri originali della rete e addestra ulteriori parametri, che vengono poi combinati con la rete originale tramite addizione.

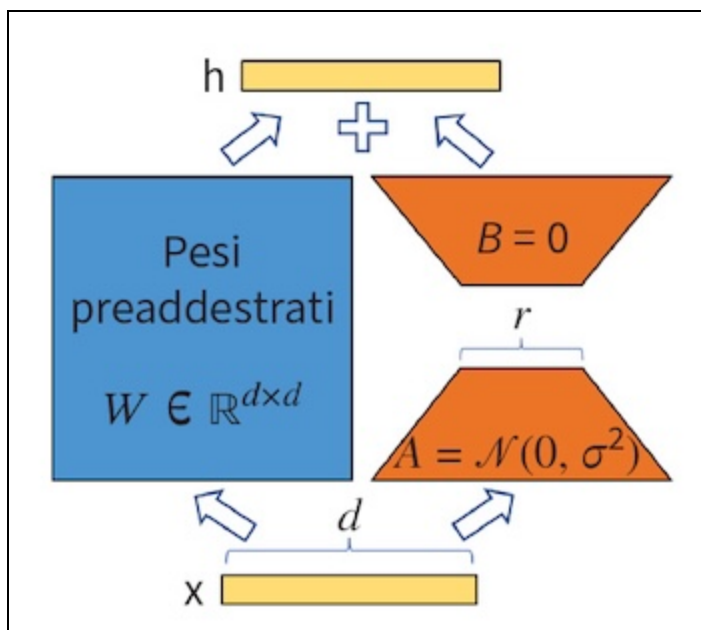


Figura 10.7 Figura 1 da Hu et al. (2021): questa figura illustra la riparametrizzazione dei nuovi layer aggiunti come un ramo parallelo usando LoRA. Solo le matrici A e B vengono addestrate, mentre i pesi pre-addestrati

W non vengono aggiornati. La figura mostra anche l'inizializzazione delle matrici A e B .

Per esempio, seguendo la terminologia del documento, considerate un layer denso nella rete con parametri W . Se l'input di questo layer è x , l'output è denotato come $h = Wx$. In LoRA, il layer aggiuntivo, ΔW viene introdotto come un ramo parallelo. L'output di LoRA diventa quindi:

$$h = Wx + \Delta Wx$$

Durante l'addestramento, i parametri di W rimangono fissi (congelati), mentre vengono adattati i parametri di ΔW .

L'aspetto chiave di LoRA è che la matrice ΔW è vincolata a essere una matrice a basso rango, definita come $\Delta W = BA$. Per esempio, se W ha dimensioni $d \times k$, anche ΔW deve avere dimensioni $d \times k$. Tuttavia, B e A hanno, rispettivamente, dimensioni $d \times r$ e $r \times k$, dove $r \ll \min(d, k)$. Quindi, in LoRA, il passo forward viene modificato come segue:

$$h = Wx + \Delta Wx = Wx + BAx = (W + BA)x = W'x$$

Quindi, una volta che i pesi B e A vengono appresi e combinati con la rete originale per formare W' , non viene introdotta alcuna ulteriore latenza al momento dell'inferenza, perché W e W' hanno le stesse dimensioni. Gli autori di Hu et al. (2021), propongono di inizializzare A con valori gaussiani casuali e B con zeri, in modo da assicurare che $\Delta W = BA$ parta da zero all'inizio dell'addestramento (vedi Figura 10.7).

Efficienza di LoRA

Gli autori hanno dimostrato che per modelli di grandi dimensioni come GPT-3 (175 miliardi di parametri), un rango r molto basso, come 1 o 2, è sufficiente anche quando il rango completo d è alto,

come 12.288. Ciò rende LoRA altamente efficiente in termini di archiviazione e calcolo.

Nei loro esperimenti, gli autori hanno dimostrato che, rispetto al fine-tuning di GPT-3 175B con ottimizzatore Adam, LoRA riduce il numero di parametri addestrabili di 10.000 volte e i requisiti di memoria GPU di tre volte. Vedi Hu et al. (2021) per maggiori dettagli sui risultati e gli esperimenti.

QLoRA

QLoRA (*Quantized LoRA*), vedi Dettmers et al. (2024), estende LoRA incorporando la quantizzazione, per renderlo ancora più efficiente in termini di archiviazione e calcolo. Maggiori dettagli su QLoRA possono essere trovati in Dettmers et al. (2024).

Fortunatamente per noi, la maggior parte di queste tecniche PEFT è già stata implementata dal team di Hugging Face. Trovate maggiori informazioni nel post del loro blog che annuncia il rilascio della libreria PEFT: *Parameter-Efficient Fine-Tuning of Billion-Scale Models on Low-Resource Hardware* all'indirizzo

<https://huggingface.co/blog/peft>. Per un'esplorazione più

approfondita, visitate il repository GitHub di PEFT all'indirizzo

<https://github.com/huggingface/peft>, che fornisce implementazioni di metodi PEFT all'avanguardia pienamente compatibili con la libreria Hugging Face. Inoltre, per saperne di più sull'efficienza del deep learning, seguite il corso del MIT *TinyML and Efficient Deep Learning Computing*, all'indirizzo

<https://hanlab.mit.edu/courses/2024-fall-65940>. Questa eccellente risorsa tratta ben più argomenti di quelli che possiamo affrontare in un singolo capitolo.

Come esempio, nel notebook allegato a questo capitolo, illustreremo come effettuare il fine-tuning del modello Llama 3

utilizzando LoRA. Potete estendere questo approccio per sviluppare varie applicazioni, come sistemi di domande e risposte personalizzati per uso interno o applicazioni rivolte ai clienti.

Allineare un LLM alle preferenze umane

Un altro aspetto chiave del fine-tuning è allineare le risposte degli LLM alle preferenze umane. Dopo la fase iniziale di pre-addestramento, a volte gli output del modello possono rivelarsi inutili o addirittura tossici. Tecniche come RLHF (*Reinforcement Learning from Human Feedback*), introdotto da Ouyang et al. (2022), aiutano i modelli a generare risposte più simili a quelle umane: creative, veritiere e utili. RLHF aiuta anche a ridurre problemi come i contenuti parziali o inaccurati ed è stato uno dei principali progressi utilizzati da OpenAI per trasformare GPT-3 nel notevole ChatGPT che tutti amiamo.

Un approccio più recente, DPO (*Direct Preference Optimization*) applica l'ottimizzazione diretta delle preferenze: semplifica il processo RLHF migliorando al contempo stabilità, prestazioni ed efficienza computazionale. Per maggiori dettagli su DPO vedi Rafailov et al. (2024).

Un aspetto interessante: i metodi PEFT – come quelli di cui abbiamo appena parlato – possono essere utilizzati anche con RLHF e DPO, così da rendere il fine-tuning ancora più efficiente.

Queste tecniche sono importanti per personalizzare gli LLM per casi d'uso specifici. Tuttavia, poiché sono al di là degli scopi di questo libro non entreremo nei dettagli. Detto questo, vale la pena di tenerle d'occhio, seguendo l'evoluzione di questo campo.

Per un esempio pratico di applicazione delle tecniche PEFT con RLHF, incoraggiamo i lettori a controllare il post *Fine-tuning 20B LLMs with RLHF on a 24GB consumer GPU* del blog di Hugging

Face, all'indirizzo <https://huggingface.co/blog/trl-peft>, che può essere particolarmente utile in caso di risorse di calcolo limitate – in pratica, se non si ha a disposizione un cluster di GPU per addestrare i propri modelli.

Conclusioni

In questo capitolo, abbiamo evidenziato l'importanza del ridimensionamento del modello per migliorare le prestazioni, non solo in un'ampia gamma di compiti ma anche nello sbloccare le abilità emergenti. Tuttavia, come dimostrato, il ridimensionamento dei modelli comporta compromessi, come una maggiore latenza e costi di distribuzione più elevati. Queste problematiche necessitano di soluzioni moderne per una messa in produzione efficiente, come la distillazione della conoscenza e la quantizzazione del modello.

Abbiamo dimostrato l'efficacia di queste tecniche utilizzando un caso di studio reale con FinBERT, un modello basato su BERT specializzato per l'analisi del sentiment nei dati finanziari. Attraverso la distillazione e la quantizzazione, abbiamo ottenuto un aumento della velocità di inferenza di quasi tre volte, mantenendo lo stesso livello di accuratezza.

Inoltre, abbiamo esplorato le tecniche di personalizzazione di un modello LLM, per renderlo più adatto a dati specifici. Tra queste, abbiamo approfondito LoRA e QLoRA che, al momento della stesura, si distinguono come alcuni dei metodi più efficienti per il fine-tuning degli LLM. Abbiamo menzionato brevemente anche le due principali tecniche attualmente utilizzate per allineare gli output degli LLM alle preferenze umane, che sono state cruciali nello sviluppo di soluzioni LLM ampiamente adottate, come ChatGPT di OpenAI. Queste tecniche sono versatili e possono essere applicate a vari modelli,

offrendo soluzioni potenti per ottimizzare le prestazioni e i costi di inferenza nelle applicazioni del mondo reale.

Capitolo 11

Postfazione

Nella Parte I del libro, abbiamo dimostrato come gli LLM possano assistere con un approccio no-code allo sviluppo di applicazioni di trading, aiutando i trader a generare codice o a sviluppare idee per le strategie di trading. Sebbene questi strumenti possano accelerare significativamente il processo di sviluppo, come nei tipici compiti di ingegneria del software in altri campi, abbiamo scoperto che, almeno nelle applicazioni finanziarie, l'esperienza nel dominio è ancora cruciale. Gli utenti necessitano di una solida comprensione dei propri obiettivi per generare strategie di trading significative. Gli LLM possono certamente fungere da copiloti, assistendo nei compiti di programmazione e chiarendo le idee, ma i loro output devono essere valutati criticamente da un utente esperto. Tecniche come la RAG (*Retrieval Augmented Generation*) possono potenziare gli LLM ancorando le loro risposte a contenuti specifici, ma quando si tratta di generare strategie di trading, l'intuito umano e l'esperienza nel dominio rimangono indispensabili.

Siamo poi passati dall'approccio no-code a un tour veloce delle tecniche di modellizzazione ben note ai trader quantitativi e ai gestori di hedge fund. Abbiamo esplorato i vantaggi e i limiti dell'apprendimento non supervisionato e supervisionato, dei metodi basati su alberi e su reti neurali, delle reti neurali ricorrenti e dell'apprendimento per rinforzo, evidenziando l'importanza e le sfumature dei dati e del feature engineering. Lungo il percorso,

abbiamo anche discusso i metodi di addestramento, come la discesa stocastica del gradiente, più utilizzati nel deep learning.

Nella Parte II, esploriamo la potenza dei modelli generativi profondi e le loro applicazioni nel trading e nella gestione patrimoniale. Questi modelli eccellono nel gestire dati complessi, multimodali e a elevata dimensionalità. Innanzitutto, abbiamo spiegato la differenza tra modelli discriminativi e generativi. Quindi, abbiamo esaminato diverse famiglie di modelli generativi, alcuni dei quali possiedono capacità fondamentali nel generare nuovi dati, calcolare densità di probabilità e scoprire nuove rappresentazioni. Non tutte le famiglie di modelli sono in grado di eseguire tutti questi compiti: alcune eccellono in aree specifiche come la generazione di dati, l'approssimazione della densità di probabilità o l'apprendimento delle rappresentazioni. Per ricapitolare, l'apprendimento delle rappresentazioni (*representation learning*) consiste nel trovare una nuova rappresentazione per lo spazio dei dati, denominata *spazio latente*. Idealmente, questa rappresentazione latente dovrebbe possedere proprietà desiderabili, come una dimensionalità inferiore rispetto all'effettivo spazio dei dati o assi di variazione "indipendenti". L'obiettivo è che questa nuova rappresentazione dei dati serva come forma compressa dei dati e che possa essere utilizzata per risolvere i compiti successivi, come la previsione, il rilevamento di anomalie o la generazione di dati. L'autoencoder variazionale (VAE) è un tipico esempio di un modello che offre tale rappresentazione. Queste capacità sono cruciali per sviluppare applicazioni in campi come la finanza, dove è cosa comune avere dati complessi, multimodali e a elevata dimensionalità – come le serie temporali multivariate – e dove la quantificazione dell'incertezza è critica per prendere decisioni informate.

Abbiamo esaminato i pro e i contro dei modelli generativi profondi sia espliciti sia impliciti per dati a elevata dimensionalità. I modelli espliciti propongono un modello di densità di probabilità trattabile per approssimare la distribuzione dei dati; al contrario, i modelli impliciti non definiscono una funzione di densità di probabilità. Esempi di modelli espliciti discussi in questo libro includono i modelli autoregressivi, il noto VAE e i modelli di flusso. Per i modelli impliciti, abbiamo esaminato le reti generative antagoniste (GAN). Questi modelli hanno preziose applicazioni nella previsione e nella gestione del rischio.

Queste tecniche, che permettono di modellare dati multimodali a elevata dimensionalità, non sono semplici estensioni dei modelli parametrici classici presenti nella statistica classica, come i modelli di regressione sotto ipotesi di distribuzione gaussiana o *t* di Student. Inoltre, a causa del gran numero di parametri coinvolti per modellare distribuzioni di dati complesse e della quantità di dati necessaria per addestrarli, questi modelli sono progettati non solo per ottenere buone prestazioni, ma anche per essere efficienti in fase di addestramento, di inferenza e di generazione di campioni. A volte, è difficile ottenere tutti questi vantaggi simultaneamente in un'unica soluzione. Tuttavia, sono state sviluppate tecniche per migliorare alcuni aspetti specifici, come quella descritta nel Capitolo 10 per migliorare la velocità di inferenza.

Ne abbiamo illustrato il potenziale nella finanza attraverso casi d'uso come la previsione di serie temporali multivariate e la generazione di serie temporali multivariate sintetiche. Sottolineiamo che, nonostante siano chiamati “intelligenza artificiale generativa”, come accennato in precedenza, i modelli presentati non servono solo per generare serie temporali multivariate sintetiche, ma anche per prevedere queste serie o come potenti “mattoni” per risolvere vari

altri compiti, come il caso d'uso presentato nel Capitolo 9. Questi modelli possono perfino essere addestrati con la *data augmentation*, utilizzando dati sintetici per migliorare le prestazioni, o essere utilizzati per gestire il rischio simulando nuovi scenari sulla base di questi dati.

Nei capitoli finali, 9 e 10, abbiamo visto come questi modelli possano essere utilizzati per elaborare dati alternativi e generare segnali di trading, mostrandone la flessibilità e le potenzialità nella generazione di feature alternative e nello sviluppo di strategie di trading. In particolare, abbiamo mostrato come costruire un sistema che prende in input dei dati vocali e restituisce in output dei segnali di trading. Questo sistema utilizza modelli transformer per convertire i segnali audio in testo e poi applica modelli LLM per assegnare punteggi di sentiment al testo, che vengono poi utilizzati per generare segnali di trading.

Last but not least, abbiamo trattato alcuni importanti argomenti speciali. Uno di questi è l'approccio attuale del pre-addestramento più adattamento, che consente di adattare o fare il fine-tuning di potenti modelli pre-addestrati per applicarli poi alle proprie problematiche. Questa tecnica offre buone prestazioni anche quando disponiamo di dati limitati per addestrare il modello per un compito specifico. Abbiamo anche discusso tecniche che aiutano a migliorare l'efficienza del processo di fine-tuning. Inoltre, abbiamo trattato tecniche di compressione per ottimizzare la scalabilità del modello, accelerare l'inferenza, ridurre la latenza di inferenza e ridurre i costi di distribuzione, che sono critici per far funzionare questi modelli in contesti pratici. Infine, abbiamo esplorato le leggi di scala degli LLM e le abilità emergenti. Le leggi di scala degli LLM affermano che l'aumento del numero di parametri del modello, della quantità di dati di addestramento e della durata dell'addestramento porta guadagni

significativi nelle prestazioni di questi modelli. Queste leggi forniscono un modo semplice per migliorare le prestazioni negli LLM: aumentando le risorse per i dati e l'addestramento, aumenteranno le prestazioni del modello. Tuttavia, non sappiamo ancora se queste leggi raggiungeranno un punto di saturazione, oltre il quale quella "legge" inizia a non funzionare più.

Le abilità emergenti, come la capacità di questi modelli di risolvere problemi matematici, compaiono solo dopo che il modello ha raggiunto una certa scala. Queste abilità rappresentano una sorpresa per professionisti e ricercatori e, al momento della stesura, non siamo in grado di prevedere quando appariranno.

Sebbene nessuna singola famiglia di modelli generativi profondi sia universalmente superiore in tutti i compiti e set di dati, famiglie differenti eccellono in domini differenti. Per esempio, i modelli autoregressivi si sono rivelati molto efficaci per la generazione di testo, mentre i modelli di diffusione sono, al momento della stesura, l'approccio leader per la generazione di immagini. Tuttavia, nella generazione di dati per applicazioni finanziarie – specialmente per generare serie temporali sintetiche univariate o multivariate – non c'è ancora un chiaro vincitore. Il basso rapporto segnale-rumore, la non stazionarietà dei dati finanziari e i set di dati relativamente piccoli disponibili per l'addestramento, rispetto a domini come il testo e le immagini, pongono problemi significativi. Questi fattori rendono il problema più difficile, suggerendo che per questo specifico caso d'uso debbano essere create soluzioni più su misura o che occorra procedere a ulteriori lavori empirici.

Nonostante queste problematiche, i modelli generativi offrono un immenso potenziale per le applicazioni finanziarie, in particolare nella generazione di feature da fonti di dati alternative, che in passato richiedevano grandi team di talenti altamente specializzati e

significativi investimenti in termini di infrastruttura. Le feature generate da questi sistemi possono essere integrate in strategie di trading più avanzate, fornendo un vantaggio competitivo in un settore sempre più guidato dai dati.

Sebbene abbiamo trattato un'ampia gamma di argomenti, ci sono diverse aree promettenti di esplorazione nella finanza in cui i modelli di intelligenza artificiale generativa potrebbero svolgere un ruolo significativo, ma che non abbiamo avuto il tempo di trattare in questa edizione.

Modelli di diffusione

I modelli di diffusione (*diffusion model*) sono attualmente considerati all'avanguardia nelle applicazioni di generazione di immagini e sono integrati in prodotti come DALL·E di OpenAI. Questi modelli sono noti anche come DDPMs (*Denoising Diffusion Probabilistic Models*), vedi Sohl-Dickstein et al. (2015) e Ho et al. (2020).

I modelli di diffusione adottano un approccio differente alla generazione di dati rispetto ai metodi che abbiamo esplorato finora. Per esempio, le reti generative antagoniste generano campioni in un unico passaggio, alimentando il rumore nel generatore, che produce l'immagine di output in una volta sola. Al contrario, i modelli di diffusione generano dati attraverso un processo di raffinamento iterativo costituito da più passaggi. Questa procedura iterativa è considerata una delle ragioni chiave delle loro prestazioni superiori.

Si tratta di modelli espliciti, con stima della densità approssimata – ricordando in questo i VAE. In effetti, il loro processo di apprendimento può essere considerato analogo all'addestramento del solo decoder di un VAE.

Questi modelli raggiungono, nella generazione di dati, risultati pari o migliori rispetto alle reti generative antagoniste, ma offrono un processo di addestramento più stabile, con il vantaggio aggiuntivo di eseguire la stima della densità. Tuttavia, il loro campionamento è relativamente lento.

Maggiori dettagli sui modelli di diffusione possono essere trovati in Sohl-Dickstein et al. (2015), Ho et al. (2020) e Bishop and Bishop (2023).

Dato il successo dei modelli di diffusione nella generazione di immagini, i ricercatori hanno iniziato ad applicarli a molteplici applicazioni, inclusa la previsione e generazione di serie temporali multivariate. I dettagli possono essere trovati nei seguenti studi: Yang et al. (2023, 2024). Prevediamo ulteriori sviluppi in questo settore.

Combinare più varianti dei modelli generativi

In tutto il libro, abbiamo visto che alcune famiglie di modelli generativi sono molto efficaci a modellare la correlazione tra le dimensioni di un singolo punto nei nostri set di dati (per esempio la co-dipendenza cross-sectional dei rendimenti tra asset). Tuttavia, per definizione, la loro struttura ipotizza l'indipendenza tra osservazioni consecutive. In tutto il libro, abbiamo esaminato vari approcci per adattare questi modelli per catturare questa struttura, le dinamiche delle serie temporali, che possono essere cruciali nel modellare le serie a determinate scale temporali, dove le dipendenze tra le osservazioni sono significative.

Alcune di queste soluzioni richiedono di combinare più famiglie di modelli, che, come abbiamo accennato, possono servire come potenti mattoni, se usati insieme. Ecco alcuni esempi.

- Combinare modelli autoregressivi con VAE: questo è stato ottenuto aggiungendo ai VAE degli encoder e decoder sequenziali, dando origine a modelli come il VRAE (*Variational Recurrent Autoencoder*), come descritto nel Capitolo 6.
- Combinare modelli autoregressivi con i modelli di flusso: nel Capitolo 7, esploriamo la combinazione di reti neurali ricorrenti e modelli transformer con i flussi.
- Combinare modelli autoregressivi con le reti generative antagoniste: nel Capitolo 8, esaminiamo l'uso di reti neurali ricorrenti e reti generative antagoniste, per produrre modelli come RCGAN o TimeGAN. In particolare, TimeGAN impiega un generatore di sequenze e un discriminatore per modellare le dinamiche delle serie temporali nello spazio latente.

Queste sono alcune delle varianti discusse in tutto il libro. Tuttavia, ci sono varianti ben note utilizzate in altri campi che non abbiamo trattato. Alcune di esse consistono nel combinare i flussi con i VAE (vedi Kingma et al. [2016]); le reti generative antagoniste con i VAE, chiamato InfoGAN (vedi Chen et al. [2016]). Ci aspettiamo che vengano sviluppate anche altre varianti, da applicare al contesto delle serie temporali.

LLM come consulenti finanziari

Analogamente a come gli LLM (*Large Language Model*) vengono messi alla prova sugli esami universitari di Medicina per valutarne le potenzialità come assistenti per i medici, Lo e il suo team stanno esplorando se gli LLM possano dimostrare la stessa conoscenza di dominio che gli esseri umani mostrano quando superano tipici esami finanziari, come il test CFA; vedi <https://mitsloan.mit.edu/ideas-made-to-matter/can-generative-ai-provide-trusted-financial-advice>.

L'obiettivo di questo approccio è quello di valutare se gli LLM sono in grado di fornire consigli finanziari pertinenti.

Secondo Lo, la ricerca mostra risultati promettenti. Ha dichiarato: “Siamo effettivamente in grado di generare conoscenza specifica del dominio sufficiente per passare gli esami tra i Large Language Model”, come riportato nell'articolo web di MIT Sloan. Ci aspettiamo di vedere una tendenza crescente nello sviluppo di LLM specifici per la finanza, per esempio, sia come consulenti sia, come dimostrato nel Capitolo 9, mettendo a punto LLM in grado di eseguire analisi del sentiment specifiche per i contesti finanziari, tra gli altri.

Questo è un campo in rapida evoluzione, e quasi quotidianamente vengono rilasciati nuovi approcci e si fanno nuovi progressi. In questo libro, il nostro obiettivo era quello di fornire una solida comprensione dei mattoni fondamentali dei modelli generativi, per consentirvi di navigare, analizzare, innovare e applicare queste innovazioni nel trading e nella gestione patrimoniale. Abbiamo disseminato esempi di codice in tutto il libro, che potete scaricare dal repository GitHub del libro: <https://github.com/genai-for-traders/genai-for-trading-and-am>. Inoltre, diamo il benvenuto alle domande e ai commenti dei lettori sul nostro account Twitter/X (eChanQT e hamletjmedina) e sul nostro blog <https://substack.com/@gatambook>. Siamo sicuri che entro la pubblicazione di questo libro, avremo già acquisito una migliore comprensione e anche strumenti migliori nell'applicare l'intelligenza artificiale alla finanza, e pubblicheremo le nostre ultime scoperte su quei media.

Riferimenti

- Abbeel, P., Chen, P., Ho, J., and Sriniva, A. (2020). Lecture slides on deep unsupervised learning, CS294-158-SP20. Course, University of California, Berkeley.
<https://sites.google.com/view/berkeley-cs294-158-sp20/home>.
- Alshahwan, N., Chheda, J., Finegenova, A., Gokkaya, B., Harman, M., Harper, I., et al. (2024). *This automated unit test improvement using large language models at Meta*.
<https://arxiv.org/abs/2402.09171>.
- Ang, A. (2014). *Asset management: A systematic approach to factor investing*. Oxford University Press.
- Ansari, A.F., Stella, L., Turkmen, C., Zhang, X., Mercado, P., et al. (2024). *Chronos: Learning the language of time series*.
<https://arxiv.org/abs/2403.07815>.
- Araci, D. (2019). *FinBERT: Financial sentiment analysis with pre-trained language models*. <https://arxiv.org/abs/1908.10063>.
- Arik, S. and Pfister, T. (2020). *TabNet: Attentive interpretable tabular learning*. <https://arxiv.org/abs/1908.07442>.
- Arjovsky, M., Chintala, S., and Bottou, L. (2017). *Wasserstein GAN*. <https://arxiv.org/abs/1701.07875>.
- Baldrige, R. (2024). *Top 10 U.S. hedge funds of December 2024*.
<https://www.forbes.com/advisor/investing/top-hedge-funds/>.
- Belov, S., Chan, E., Jetha, N., and Nautiyal, A. (2023). *Applying corrective AI to daily seasonal Forex trading*.

<https://predictnow.ai/applying-corrective-ai-to-daily-seasonal-forex-trading-paper/>.

- Bishop, C.M. (2006). *Pattern recognition and machine learning (Information science and statistics)*. Berlin, Heidelberg: Springer-Verlag.
- Bishop, C.M. and Tipping, M.E. (2001). Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61, part 3, pp. 611-622.
- Bishop, C. M. and Bishop, H. (2023). *Deep learning: Foundations and concepts*. Springer Nature.
- Bloomberg News. (2023). *JPMorgan AI analyzes Fed speeches for trading signals*. <https://www.bloomberg.com/news/articles/2023-04-26/jpmorgan-s-ai-puts-25-years-of-federal-reserve-talk-into-a-hawk-dove-score>.
- Bodie, Z., Marcus, A.J., and Kane, A. (2018). *Investments*. New York: McGraw-Hill/Irwin.
- Botte, A. and Bao, Ds. (2021). *A machine learning approach to regime modeling*. <https://www.twosigma.com/articles/a-machine-learning-approach-to-regime-modeling/>.
- Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software.
- Brubaker, M. A. and Köthe, U. (2021). *Normalizing flows and invertible neural networks in computer vision*. CVPR 2021 Tutorial.
- Brubaker, B. (2024). *How chain-of-thought reasoning helps neural networks compute*. <https://www.quantamagazine.org/how-chain-of-thought-reasoning-helps-neural-networks-compute-20240321/>.

- Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E. et al. (2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4*.
<https://arxiv.org/abs/2303.12712>.
- Catello, L., Ruggiero, L., Schiavone, L., and Valentino, M. (2023). *Hidden Markov models for stock market prediction*.
<https://arxiv.org/pdf/2310.03775.pdf>.
- Chan, E. (2013). *Algorithmic trading: Winning strategies and their rationale*. New York: Wiley.
- Chan, E. (2021). *Quantitative trading: How to build your own algorithmic trading business*, 2nd ed. New York: Wiley.
- Chan, E. (2022). *What is Corrective AI?*
<https://predictnow.ai/what-is-corrective-ai/>.
- Chan, E., Fan, H., Sawal, S., and Viville, Q. (2023). *Conditional portfolio optimization: Using machine learning to adapt capital allocations to market regimes*. <https://ssrn.com/abstract=4383184>.
- Chan, E.P. (2017). *Machine trading: Deploying computer algorithms to conquer the markets*. New York: Wiley.
- Chen, R.T.Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D.K. (2018). Neural Ordinary differential equations. *Advances in Neural Information Processing Systems*, 31.
- Chen, X., Duan, Y., Houthoofd, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in Neural Information Processing Systems*, 29.
- Claburn, T. (2024). *Simon Willison interview: AI software still needs the human touch*.
https://www.theregister.com/2024/01/24/willison_ai_software_development/?utm_source=tldrnewsletter.

- Cong, L. W., Feng, G., He, J., and He, X. (2024). Growing the Efficient Frontier on Panel Trees. *Journal of Financial Economics*, Forthcoming, <https://ssrn.com/abstract=3949463>.
- Cong, L. W., Tang, K., Wang, J., and Zhang, Y. (2021). AlphaPortfolio: *Direct construction through deep reinforcement learning and interpretable AI*. <https://ssrn.com/abstract=3554486>.
- Denev, A. and Amen, S. (2020). *The book of alternative data: A guide for investors, traders and risk managers*. New York: John Wiley & Sons.
- Deng, R., Chang, B., Brubaker M.A., Mori, G., and Lehrmann, A. (2020). Modeling continuous stochastic processes with dynamic normalizing flows. *Advances in Neural Information Processing Systems*, 33, pp. 7805-7815.
- Desai, A., Freeman, C., Wang, Z., and Beaver, I. (2021). *Timevae: A variational auto-encoder for multivariate time series generation*. arXiv Preprint arXiv:2111.08095.
- Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2024). Qlora: Efficient finetuning of quantized LLMS. *Advances in Neural Information Processing Systems*, 36.
- Devlin, J., Chang M.-W., Lee, K., and Toutanova, K. (2019). BERT: *Pre-training of deep bidirectional transformers for language understanding*. <https://arxiv.org/abs/1810.04805>.
- Dinh, L., Krueger, D., and Bengio, Y. (2015). *NICE: Non-linear independent components estimation*. <https://arxiv.org/abs/1410.8516>.
- Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). *Density estimation using real NVP*. <https://arxiv.org/abs/1605.08803>.
- Dixon, M. F., Halpern, I., and Bilokon, P. (2020). *Machine learning in finance: From theory to practice*. New York: Springer.

- Esteban, C., Hyland, S.L., and Rätsch, G. (2017). *Real-valued (medical) time series generation with recurrent conditional GANS*. arXiv Preprint arXiv:1706.02633.
- Fabius, O. and Van Amersfoort, J.R. (2014). *Variational recurrent auto-encoders*. arXiv Preprint arXiv:1412.6581.
- Fama, E. F. and French, K. R. (1995). Size and book-to-market factors in earnings and returns. *Journal of Finance*, 50, pp. 131-155.
- Fortune. (2023). *JPMorgan created a Fed-whispering AI model to help investors stay on top of the market*.
<https://fortune.com/2023/04/28/jpmorgan-artificial-intelligence-chatgpt-llm-hawk-dove-federal-reserve-speak/>.
- Frey, B.J., Hinton, G.E., and Dayan, P. (1995). Does the wake-sleep algorithm produce good density estimators? *Advances in Neural Information Processing Systems*, 8.
- Fuhrer, B., Tessler, C., and Dala, G. (2024). *Gradient Boosting reinforcement learning*. <https://arxiv.org/abs/2407.08250>.
- Gatheral, J. (2006). *The volatility surface: A practitioner's guide*. New York: Wiley.
- Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). *MADE: Masked Autoencoder for Distribution Estimation*.
<https://arxiv.org/abs/1502.03509>.
- Grinsztajn, L., Oyallon, E., and Varoquaux, G. (2022). *Why do tree-based models still outperform deep learning on tabular data?* <https://arxiv.org/abs/2207.08815>.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.
- Goodfellow, I.J. Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Sherjil et al. (2014). *Generative adversarial networks*. <https://arxiv.org/abs/1406.2661>.

- Goodfellow, I. (2019). *Adversarial machine learning*. Invited talk at ICLR.
- Gou, J., Baosheng, Y., Maybank, S.J., and Tao, D. (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6), pp. 1789-1819.
<https://doi.org/10.1007/s11263-021-01453-z>.
- Guida, Ty. (2019). *Big data and machine learning in quantitative investment*. New York: John Wiley & Sons.
- Guijarro-Ordóñez, J., Pelger, M., and Zanotti, G. (2022). *Deep learning statistical arbitrage*. <https://arxiv.org/abs/2106.04028>.
- Guijarro-Ordóñez, J., Pelger, M., and Zanotti, G. (2021). *Deep learning statistical arbitrage*. arXiv Preprint arXiv:2106.04028.
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). *Improved training of Wasserstein GANs*.
<https://arxiv.org/abs/1704.00028>.
- Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning*. 2nd ed. New York: Springer.
- Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2018). *GANs trained by a two time-scale update rule converge to a local Nash equilibrium*.
<https://arxiv.org/abs/1706.08500>.
- Hinton, G., Vinyals, O., and Dean, J. (2015). *Distilling the knowledge in a neural network*. <https://arxiv.org/abs/1503.02531>.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33, pp. 6840-6851.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), pp. 359-366.

- Houshy, N., Giurghi, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., et al. (2019). Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pp. 2790-2799. PMLR.
- Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S. et al. (2021). *LORA: Low-rank adaptation of large language models*. arXiv Preprint arXiv:2106.09685.
- Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. et al. (2017). *Quantization and training of neural networks for efficient integer-arithmetic-only inference*.
<https://arxiv.org/abs/1712.05877>.
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R. et al. (2020). *Scaling laws for neural language models*. <https://arxiv.org/abs/2001.08361>.
- Kıcıman, E., Ness, R., Sharma, A., and Tan, C. (2023). *Causal reasoning and large language models: Opening a new frontier for causality*. [https://arxiv.org/abs/\(2305\).00050](https://arxiv.org/abs/(2305).00050).
- Kingma, D.P. (2013). *Auto-encoding variational Bayes*. arXiv Preprint arXiv:1312.6114.
- Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 29.
- Kinlaw, W., Kritzman, M.P., and Turkington, D. (2021). *Asset allocation: From theory to practice and beyond*. New York: John Wiley & Sons.
- Kobyzev, I., Prince, S.J.D., and Brubaker, M.A. (2021). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine*

Intelligence, 43 (11), pp. 3964-3979.

[https://doi.org/10.1109/tpami.\(2020\).2992934](https://doi.org/10.1109/tpami.(2020).2992934).

- Kolm, P., Turiel, J., and Westray, N. (2021). *Deep order flow imbalance: Extracting alpha at multiple horizons from the Limit Order Book*. SSRN: <https://ssrn.com/abstract=3900141>.
- Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 29-37. JMLR Workshop; Conference Proceedings.
- Lester, B., Al-Rfou, R., and Constant, N. (2021). *The power of scale for parameter-efficient prompt tuning*. arXiv Preprint arXiv:2104.08691.
- Levine, S. (2021). Lecture slides on designing, visualizing and understanding deep neural networks, CS W182/282A. Course, University of California, Berkeley. <https://cs182sp21.github.io>.
- Li, X.L. and Liang, P. (2021). *Prefix-tuning: Optimizing continuous prompts for generation*. arXiv Preprint arXiv:2101.00190.
- Liu, S., Yu, H., Liao, C., Li, J., Lin, W., Liu, A.X. et al. (2021). Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*.
- López de Prado, M. (2015). The probability of backtest overfitting. *Journal of Computational Finance (Risk Journals)*, 34, pp. 123-137.
- López de Prado, M. (2018). *Advances in financial machine learning*. New York: Wiley.
- López de Prado, M.M. (2020). *Machine learning for asset managers. Elements in quantitative finance*. Cambridge University Press.

- Malo, P., Sinha, A., Korhonen, P., Wallenius, J., and Takala, P. (2014). Good debt or bad debt: Detecting semantic orientations in economic texts. *Journal of the Association for Information Science and Technology*, 65 (4), pp. 782-796.
- Man, X. and Chan, E. P. (2021a). The best way to select features? Comparing MDA, LIME, and SHAP. *The Journal of Financial Data Science*, 3(1), pp. 127-139.
- Man, X. and Chan, E. (2021b). Cluster-based feature selection. *Market Technician*, 90, pp. 11-22.
- Mao, X., Li, Q., Xie, H., Lau, R.Y.K., Wang, Z., and Smolley, S.P. (2017). *Least squares generative adversarial networks*.
<https://arxiv.org/abs/1611.04076>.
- Medina Ruiz, H. J. (2019). Predicting Criteo's internet traffic load using Bayesian structural time series models. Presented at the Strata Data Conference.
<https://conferences.oreilly.com/strata/strata-ny-2019/public/schedule/detail/77395.html>.
- Mescheder, L., Andreas G., and Sebastian N. (2018). Which training methods for GANs do actually converge?
<https://arxiv.org/abs/1801.04406>.
- Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. (2018). *Spectral normalization for generative adversarial networks*.
<https://arxiv.org/abs/1802.05957>.
- Murphy, K. (2012). *Machine learning: A probabilistic perspective*. Boston, MA: MIT Press.
- Murphy, K.P. (2022). *Probabilistic machine learning: An introduction*. Boston, MA: MIT Press. probml.ai.
- Murphy, K.P. (2023). *Probabilistic machine learning: Advanced topics*. Boston, MA: MIT Press. <http://probml.github.io/book2>.

- Nautiyal, A. and Chan, E. (2021). *Metalabeling and the duality between cross-sectional and time-series factors*. Retrieved from <https://predictnow.ai/metalabeling-and-the-duality-between-cross-sectional-and-time-series-factors/>.
- Nalisnick, E., Matsukawa, A., Teh, Y.W., Gorur, D., and Lakshminarayanan, B. (2019). Hybrid models with deep and invertible features. In *International Conference on Machine Learning*, pp. 4723-4732. PMLR.
- Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 427-436.
- Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A. et al. (2016a). *WaveNet: A generative model for raw audio*. <https://arxiv.org/abs/1609.03499>.
- Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). *Pixel recurrent neural networks*. <https://arxiv.org/abs/1601.06759>.
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P. et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, pp. 27730-27744.
- Paleologo, G.A. (2021). *Advanced portfolio management: A quant's guide for fundamental investors*. New York: Wiley.
- Papamakarios, G., Nalisnick, E., Rezende, D.J., Mohamed, S., and Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57), pp. 1-64.
- Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems*, 30.

- Part Time Larry. (2022). 2. *OpenAI whisper—Fed speech recognition*. <https://www.youtube.com/watch?v=Wc4bQxuypo0>.
- Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. et al. (2018). *Deep contextualized word representations*. [https://arxiv.org/abs/\(1802\).05365](https://arxiv.org/abs/(1802).05365).
- Pik, J., Chan, E., Broad, J. and Sun, P. (2025) *Hands-On AI Trading with Python, QuantConnect and AWS*. New York: Wiley.
- Prado, R. and West, M. (2010). *Time series: Modeling, computation, and inference*. Chapman; Hall/CRC.
- Radford, A., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P. et al. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), p. 9.
- Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I. (2022). *Robust speech recognition via large-scale weak supervision*. [https://arxiv.org/abs/\(2212\).04356](https://arxiv.org/abs/(2212).04356).
- Radford, A. Narasimhan, K., Salimans, T., Sutskever, I. (2018). *Improving language understanding by generative pre-training*.
- Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S., and Finn, C. (2024). Direct preference optimization: your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M. et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), pp. 1-67.
- Rasul, K., Sheikh, A-S., Schuster, I., Bergmann, U.M., and Vollgraf, R. (2021). Multivariate probabilistic time series forecasting via conditioned normalizing flows. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=WiGQBFuVRv>.

- Rasul, K., Ashok, A., Williams, A.R., Ghonia, H., Bhagwatkar, R., Khorasani, A. et al. (2024). *Lag-Llama: Towards foundation models for probabilistic time series forecasting*.
[https://arxiv.org/abs/\(2310\).08278](https://arxiv.org/abs/(2310).08278).
- Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., and Bengio, Y. (2015). *FitNets: Hints for thin deep nets*.
<https://arxiv.org/abs/1412.6550>.
- Ruppert, D., and Matteson, D. S. (2015). *Statistics and data analysis for financial engineering with R examples*. 2nd ed. New York: Springer.
- Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). *Improved techniques for training GANs*.
[https://arxiv.org/abs/\(1606\).03498](https://arxiv.org/abs/(1606).03498).
- Schwartz-Ziv, R. and Armon, A. (2021). *Tabular data: Deep learning is not all you need*. <https://arxiv.org/abs/2106.03253>.
- Simon, D.P. and Campasano, J. (2014). *The VIX futures basis: evidence and trading strategies*.
https://www.efmaefm.org/0EFMA%20MEETINGS/EFMA%20ANNUAL%20MEETINGS/2013-Reading/papers/EFMA2013_0164_fullpaper.pdf.
- Sirignano, J. and Cont, R. (2018). *Universal features of price formation in financial markets: Perspectives from deep learning*.
<https://arxiv.org/pdf/1803.06917.pdf>.
- Smith, L. N. (2017, March). *Cyclical learning rates for training neural networks*. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464-472. IEEE.
<https://doi.org/10.1109/WACV.2017.58>.
- Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). *Deep unsupervised learning using nonequilibrium*

thermodynamics. In *International Conference on Machine Learning*, 2256-2265. PMLR.

- Sood, S., Papasotiriou, K., Vaiciulis, M., and Balch, T. (2023). Deep reinforcement learning for optimal portfolio allocation: A comparative study with mean-variance optimization. https://icaps23.icaps-conference.org/papers/finplan/FinPlan23_paper_4.pdf.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), pp. 1929-1958.
- Torralba, A., Isola, P., and Freeman, W.T. (2024). *Foundations of computer vision*. Boston, MA: MIT Press.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T. et al. (2023). *LLaMA: Open and efficient foundation language models*. arXiv. arXiv Preprint arXiv:2302.13971.
- Tunstall, L., Von Werra, L., and Wolf, T. (2022). *Natural language processing with transformers*. O'Reilly Media, Inc.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2023). *Attention is all you need*. <https://arxiv.org/abs/1706.03762>.
- Vig, J. (2019). A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 37-42. Florence, Italy: Association for Computational Linguistics. <https://doi.org/10.18653/v1/P19-3007>.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F. et al. (2023). *Chain-of-thought prompting elicits reasoning in large language models*. <https://arxiv.org/abs/2201.11903>.

- Wei, J. Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S. et al. (2022). *Emergent abilities of large language models*.
<https://arxiv.org/abs/2206.07682>.
- Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., and Sun, L. (2023). *Transformers in time series: A survey*.
<https://arxiv.org/abs/2202.07125>.
- WGN News. (2022). *WATCH LIVE | Fed Chair Powell Discusses Latest Fed Rate Hike*. <https://www.youtube.com/watch?v=NT2H9iyd-ms>.
- Williams, R.J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, pp. 229-256.
- WWW '18: Companion Proceedings of the Web Conference. (2018). Republic; Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee.
- Xue, C., Lute, J., Schonfeld, D., Han, G., and Dahlmann, L. (2023). *How eBay created a language model with three billion item titles*. <https://innovation.ebayinc.com/tech/engineering/how-ebay-created-a-language-model-with-three-billion-item-titles/>.
- Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y. et al. (2023). Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4), pp. 1-39.
- Yang, Y., Jin, M., Wen, H., Zhang, C., Liang, Y., Ma, L. et al. (2024). *A survey on diffusion models for time series and spatio-temporal data*. arXiv Preprint arXiv:2404.18886.
- Yoon, J., Jarrett, D., and van der Schaar, M. (2019). Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, Eds. *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

- Zaken, E.B., Ravfogel, S., and Goldberg, Y. (2021). *Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models*. arXiv Preprint arXiv:2106.10199.
- Zhang, Z., Zohren, S., and Roberts, S. (2019). deep reinforcement learning for trading.
<https://arxiv.org/pdf/1911.10107.pdf>.
- Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H. et al. (2021). *Informer: Beyond efficient transformer for long sequence time-series forecasting*.
<https://arxiv.org/abs/2012.07436>.
- Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). *Aligning books and movies: Towards story-like visual explanations by watching movies and reading books*. <https://arxiv.org/abs/1506.06724>.
- Zhuang, Y., Liu, L., Singh, C., Shang, J., and Gao, J. (2024). *Learning a decision tree algorithm with transformers*.
<https://arxiv.org/abs/2402.03774>.
- Zuckerman, G. (2019). *The man who solved the market: How Jim Simons launched the quant revolution*. Penguin Publishing Group. <https://books.google.fr/books?id=2vm0DwAAQBAJ>.

Appendice

Recupero dei prezzi di chiusura rettificati e calcolo dei rendimenti giornalieri

Spesso, per ottenere risposte corrette da ChatGPT, è meglio chiedere del codice invece di risposte numeriche vere e proprie. Ecco i risultati.

Prompt dell'utente

Mostra il codice Python che ci permetterebbe di recuperare da Yahoo Finance i prezzi di chiusura rettificati degli ultimi 100 giorni di SPY.

Supponendo che non siate programmatori esperti, almeno non in Python, potete anche chiedere le istruzioni per installare Python, che qui trovate nel paragrafo “Installazione di Python” in questa Appendice.

Come suggerito, possiamo copiare e incollare i seguenti comandi in Jupyter Notebook, per eseguirlo nel browser, o eseguirlo nell’IDE Spyder.

Risposta di ChatGPT

```
import yfinance as yf
import pandas as pd
# Definisci il simbolo del ticker
tickerSymbol = 'SPY'
# Ottieni i dati per questo ticker
tickerData = yf.Ticker(tickerSymbol)
```

```
# Ottieni i prezzi storici per questo ticker
tickerDf = tickerData.history(period='1d',
start='2023-08-25', end='2023-12-12')
# Mantieni solo la colonna Prezzo di Chiusura Rettificato
adjusted_close = tickerDf['Close']
# Mostra gli ultimi 100 giorni di prezzi di chiusura rettificati
print(adjusted_close.tail(100))
```

ChatGPT prosegue.

Risposta di ChatGPT

Questo script fa quanto segue:

Importa le librerie necessarie (yfinance e pandas).

Definisce il simbolo del ticker per l'ETF SPY.

Recupera i dati storici per SPY da Yahoo Finance.

Estrae i prezzi di chiusura rettificati.

Stampa gli ultimi 100 giorni di prezzi di chiusura rettificati.

Assicurati di modificare le date di inizio e fine nel metodo history, in modo che corrispondano alla data corrente e risalgano a 100 giorni da quella data. La libreria yfinance esegue automaticamente la rettifica per dividendi e split azionari.

È vero che “La libreria yfinance esegue automaticamente la rettifica per dividendi e split azionari”? Abbiamo verificato la prima riga dell'output,

2023-08-25 00:00:00-04:00 436.653168

ed effettivamente corrispondeva al prezzo di chiusura rettificato visualizzato su Yahoo Finance al momento della stesura di questo testo. Come possiamo scaricare in un file csv i dati stampati nel browser?

Prompt dell'utente

Come possiamo scaricare in un file csv i dati generati nel programma precedente con la prima colonna che sia la data e la seconda colonna il prezzo?

Risposta di ChatGPT

...

```
# Salva in un file CSV gli ultimi 100 giorni di prezzi di chiusura  
rettificati  
adjusted_close.tail(100).to_csv('SPY_last_100_days.csv', header=True)  
print("Data saved in 'SPY_last_100_days.csv'")
```

Potete verificare che `SPY_last_100_days.csv` sia stato creato nella cartella `book4_py/` dove avete creato l'ambiente Python. Ora vogliamo calcolare i rendimenti giornalieri.

Prompt dell'utente

Ottimo! Ora, per favore, mostra il codice che utilizza i prezzi recuperati per calcolare i rendimenti giornalieri.

Risposta di ChatGPT

...

```
# Calcola i rendimenti giornalieri  
daily_returns = adjusted_close.pct_change()  
# Mostra gli ultimi 100 giorni di rendimenti giornalieri  
print(daily_returns.tail(100))
```

Naturalmente, possiamo stamparlo in un file csv `SPY_returns_last_100_days.csv` usando lo stesso metodo impiegato sopra. Potete facilmente verificare che i rendimenti giornalieri vengono calcolati correttamente.

Installazione di Python

Come abbiamo menzionato nel paragrafo “Recupero dei prezzi di chiusura rettificati e calcolo dei rendimenti giornalieri” all’inizio di questa Appendice, volevamo usare Python per recuperare i prezzi storici. Per farlo, dobbiamo prima installare Python sul desktop Windows, e poi installare il pacchetto `yfinance` (Yahoo Finance), cosa che faremo nel Passo 4.

(Se già siete programmatori Python potete saltare questo paragrafo: questo capitolo si rivolge a un programmatore Python alle prime armi.)

Prompt dell'utente

Come installo un ambiente Python sul mio desktop Windows?

Risposta di ChatGPT

Installare un ambiente Python sul tuo desktop Windows è un processo semplice. Ecco una guida passo passo per iniziare.

Passo 1: Scarica Python

- 1. Visita il sito web ufficiale di Python: vai su python.org.*
- 2. Cerca Download: il sito dovrebbe rilevare il tuo sistema operativo e mostrarti l'ultima versione di Python per Windows.*
- 3. Scarica Python: fai clic sul link del download per Windows. È meglio scaricare l'ultima versione stabile.*

Passo 2: installa Python

- 1. Esegui il programma di installazione: una volta completato il download, apri il programma di installazione.*
- 2. Seleziona le opzioni di installazione.*
 - Attiva l'opzione "Aggiungi Python al PATH": questo è cruciale, in quanto ti permette di eseguire Python dal prompt dei comandi.*
 - Scegli Installazione personalizzata (opzionale): puoi personalizzare le funzionalità, se necessario, ma le impostazioni di default di solito sono adatte alla maggior parte degli utenti.*
- 3. Procedi con l'Installazione: fai clic su "Installa ora" e attendi il completamento dell'installazione.*
- 4. Verifica l'installazione.*
 - Apri il prompt dei comandi.*
 - Digita `python --version` e premi Invio. Dovrebbe essere visualizzata la versione di Python, che conferma che Python è installato.*

Passo 3: imposta un ambiente virtuale (opzionale ma consigliato)

Gli ambienti virtuali di Python ti permettono di gestire installazioni separate di pacchetti per i tuoi vari progetti. Ciò previene l'insorgere di conflitti dovuti alle dipendenze.

- 1. Apri il prompt dei comandi: raggiungi la cartella nella quale vuoi creare il tuo progetto.*
- 2. Crea un ambiente virtuale:*
 - Esegui `python -m venv myenv` (sostituisci a `myenv` il nome che desideri dare al tuo ambiente).*

Questo crea nella tua directory dei progetti una cartella chiamata myenv contenente l'ambiente virtuale.

3. Attiva l'ambiente virtuale.

Per Windows, esegui myenv\Scripts\activate.

Saprai che l'ambiente virtuale è attivato perché il prefisso della riga di comando mostra il nome del tuo ambiente.

Passo 4: installa i Pacchetti con pip

Ora che Python è installato, puoi usare pip per installare i pacchetti.

Per installare un pacchetto, usa pip install nome-pacchetto.

Ricorda di attivare il tuo ambiente virtuale se ne stai usando uno.

[Nota dell'Autore: questo dovrebbe essere pip install yfinance.]

Passo 5: considera l'uso di un IDE (ambiente di sviluppo Integrato)

Sebbene non necessario, un IDE può rendere la programmazione in Python molto più facile. Le scelte più note sono:

PyCharm;

Visual Studio Code;

Jupyter Notebook (ottimo per progetti di data science).

[Nota dell'Autore: non capisco perché non consigli anche Spyder, che viene fornito automaticamente con la libreria Anaconda. Per un inizio rapido e facile, consiglio di usare Jupyter Notebook, che può essere eseguito in un browser web. Come fare? Vedi il prossimo prompt.]

Consigli aggiuntivi

Aggiorna Python regolarmente: mantieni aggiornata la tua versione di Python per ottenere gli ultimi miglioramenti in termini di sicurezza e funzionalità.

Esplora le librerie Python: Python ha un vasto ecosistema di librerie per diversi scopi - sviluppo web, analisi dei dati, machine learning e così via.

Impara le basi di Python: se sei alle prime armi, segui qualche tutorial per comprendere le basi.

Seguendo questi passi, dovresti avere un ambiente Python funzionante configurato sul tuo desktop Windows.

Prompt dell'utente

Supponendo di aver installato un ambiente virtuale Python chiamato book4_py sul desktop Windows, come posso installare Jupyter Notebook come IDE?

Risposta di ChatGPT

Installazione di Jupyter Notebook.

Una volta che il tuo ambiente virtuale è attivo, puoi installare Jupyter Notebook usando pip. Esegui il seguente comando:

```
pip install notebook
```

Questo comando scaricherà e installerà Jupyter Notebook e le sue dipendenze nel tuo ambiente virtuale book4_py.

Avvio di Jupyter Notebook.

Dopo che l'installazione sarà completata, puoi avviare Jupyter Notebook eseguendo:

```
jupyter notebook
```

Si aprirà una nuova finestra del browser con Jupyter Notebook. Ora basta raggiungere la cartella in cui volete scaricare i dati dei prezzi di chiusura rettificati e copiare e incollare il codice nell'Appendice A.1.

Tracciamento del tasso privo di rischio nel corso degli anni

Molte applicazioni finanziarie, come il calcolo dell'indice di Sharpe, richiedono l'uso del tasso privo di rischio (rf , risk-free). Tuttavia, ChatGPT ha faticato a localizzare un sito web per scaricarli gratuitamente e in un formato comodo. Quindi usiamo come proxy i rendimenti dell'ETF dei buoni del tesoro a breve termine: BIL.

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
# Carica il tuo file CSV
file_path = 'BIL.csv' # Sostituisci con il percorso del tuo file
data_plot = pd.read_csv(file_path)
# Converti la colonna 'Date' in formato datetime
data_plot['Date'] = pd.to_datetime(data_plot['Date'])
# Calcola i rendimenti giornalieri e annualizzandoli
data_plot['Daily Returns'] = data_plot['Close'].pct_change()
data_plot['Annualized Returns'] = data_plot['Daily Returns'] * 252 #
Annualizza i rendimenti
# Traccia i rendimenti annualizzati
plt.figure(figsize=(12, 6))
plt.plot(data_plot['Date'], data_plot['Annualized Returns'],
color='green', marker='o', linestyle='-', linewidth=2)
plt.title('Annualized Returns Over Time')
plt.xlabel('Year')
```

```
plt.ylabel('Annualized Returns')
plt.grid(True)
plt.xticks(rotation=45)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y')) # Solo
l'anno per le etichette dell'asse x
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.tight_layout()
# Mostra il grafico
plt.show()
```

Per scaricare il grafico nella Figura A.1 invece di visualizzarlo in Jupyter Notebook, chiedete a ChatGPT le istruzioni. Risponderà così.

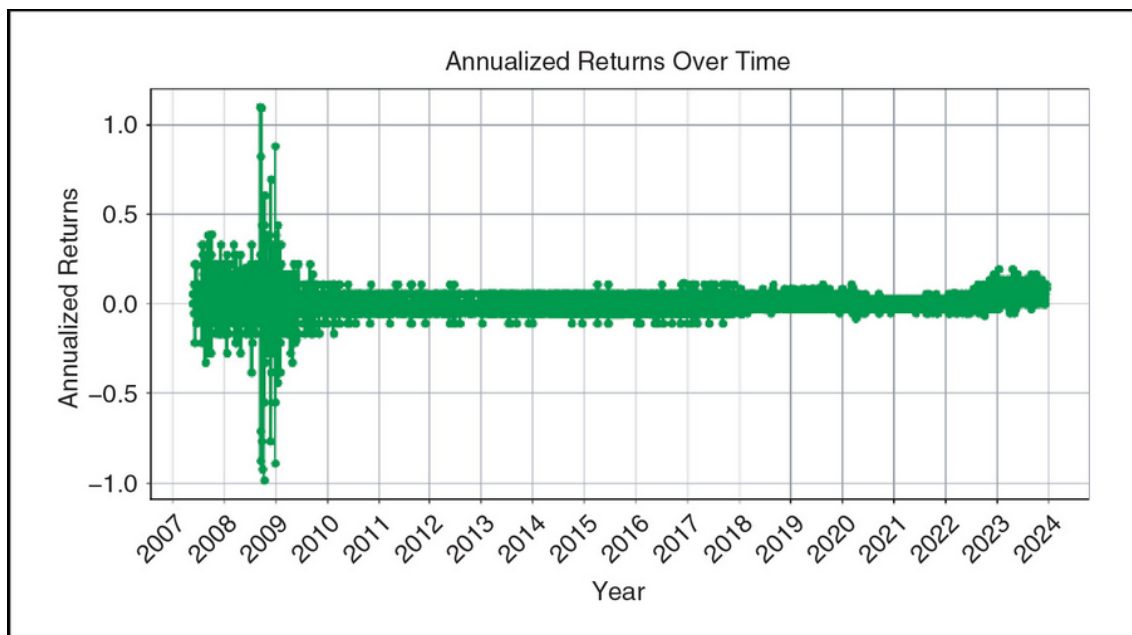


Figura A.1 Grafico dei rendimenti annualizzati di BIL.

Risposta di ChatGPT

```
# Salva il grafico come file PNG
plot_file_path = 'your_plot.png'
...
# Salva la figura
plt.savefig(plot_file_path)
```

Sebbene il grafico sia stato tracciato correttamente, i rendimenti giornalieri mostrano troppa volatilità per permetterci di trarre informazioni e, cosa più importante, per verificare se i rendimenti sono sensati. Dopotutto, questo è un ETF che viene scambiato ogni

microsecondo, non il Federal Funds Rate stabilito dal Federal Open Market Committee. Una media mobile di circa un mese dovrebbe avere un aspetto migliore.

Prompt dell'utente

Ridisegna il grafico sopra con una media mobile a 20 giorni

Sfortunatamente, non è stato possibile per alcune ragioni tecniche, ma ChatGPT ha generato del codice che possiamo usare. La riga chiave è

```
# Calcola la media mobile a 20 giorni
data_plot['20-Day MA'] = data_plot['Annualized
Returns'].rolling(window=20).mean()
```

Ora possiamo vedere chiaramente nella Figura A.2 il tasso di interesse quasi zero dovuto al Quantitative Easing dalla crisi finanziaria del 2008, fino a quando è gradualmente salito a partire dal 2017 con il recupero dell'economia, e poi un altro crollo a zero durante l'era COVID del 2020, prima di risalire stabilmente nel 2022 in risposta alla pressione inflazionistica dovuta all'enorme stimolo fiscale, assestandosi sopra il 5% a fine 2023.

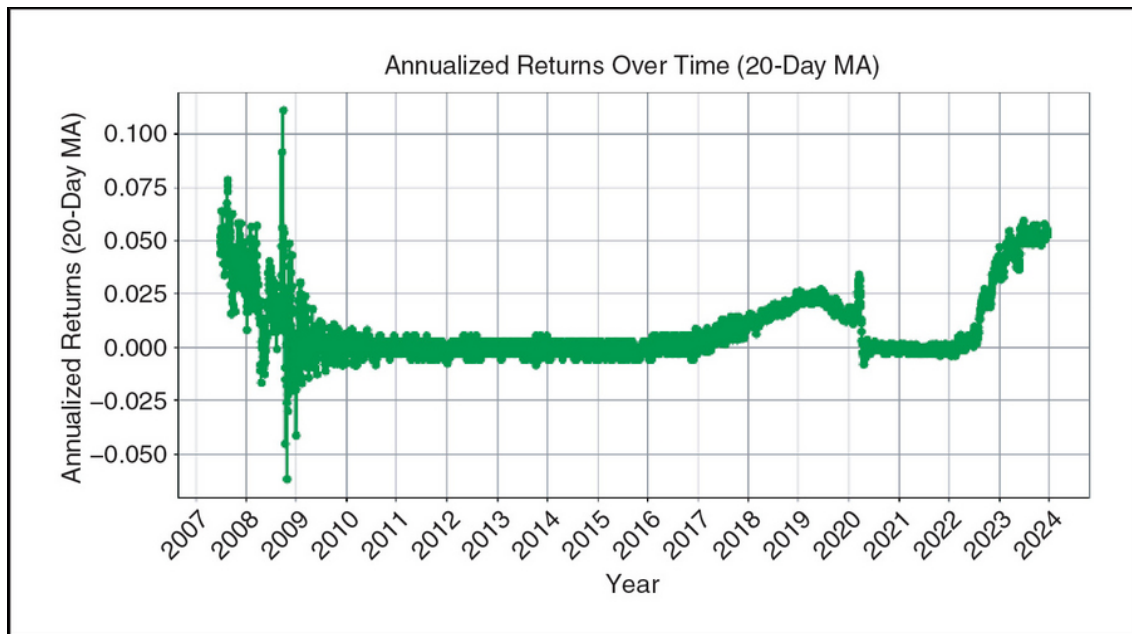


Figura A.2 Grafico della media mobile a venti giorni dei rendimenti annualizzati di BIL.

Quindi, dopo molto lavoro, siamo soddisfatti che i dati di BIL siano stati recuperati correttamente e i rendimenti giornalieri calcolati correttamente. Useremo anche questi rendimenti della media mobile a 20 giorni come *risk-free* per altre applicazioni.

Prompt dell'utente

Come posso scaricare le date e i numeri della media mobile in un file csv con due colonne?

Risposta di ChatGPT

Seleziona solo le colonne 'Data' e 'Media Mobile 20 Giorni' per l'esportazione

```
export_data = data_plot[['Date', '20-Day MA']]
```

Salva i dati in un file CSV

```
export_file_path = 'BIL_annualized_returns_20_day_MA.csv'  
export_data.to_csv(export_file_path, index=False)
```

Calcolo dell'indice di Sharpe di SPY

Nel paragrafo “Recupero dei prezzi di chiusura rettificati e calcolo dei rendimenti giornalieri” di questa Appendice abbiamo calcolato i rendimenti giornalieri di SPY e nel precedente paragrafo “Tracciamento del tasso privo di rischio nel corso degli anni” il tasso privo di rischio. Ora usiamoli per calcolare l'indice di Sharpe di SPY.

Prompt dell'utente

Ipotizziamo che in A.csv sia memorizzata una serie di rendimenti giornalieri A, e che in B.csv sia memorizzato il tasso giornaliero privo di rischio, e che le date di A e B non siano identiche, sebbene si sovrappongano, come calcoliamo l'indice di Sharpe di A? Per favore mostrami il codice Python.

Risposta di ChatGPT

Per calcolare l'indice di Sharpe per una serie di rendimenti giornalieri (Serie A) con una corrispondente serie di tassi giornalieri privi di rischio (Serie B), dovrai allineare le due serie sulle loro date comuni. Tipicamente, l'indice di Sharpe viene calcolato come il rendimento medio dell'investimento meno il tasso privo di rischio, diviso per la deviazione standard dei rendimenti dell'investimento. Ecco una guida passo passo per eseguire questo calcolo in Python.

Carica entrambi i dataset. Allinea i dataset in base alle loro date. Calcola i rendimenti in eccesso sottraendo il tasso privo di rischio dai rendimenti della Serie A. Calcola l'indice di Sharpe usando la formula: (Rendimenti in Eccesso Medi) / (Deviazione standard dei Rendimenti in Eccesso).

Il codice Python per ottenere ciò sarebbe simile a questo:

```
import pandas as pd
# Carica Serie A (rendimenti investimento) e Serie B (tasso privo di rischio)
series_a = pd.read_csv('percorso_del_tuo_csv/A.csv')
series_b = pd.read_csv('percorso_del_tuo_csv/B.csv')
# Converti le colonne 'Date' in datetime e assicurati che siano l'indice
series_a['Date'] = pd.to_datetime(series_a['Date'])
series_b['Date'] = pd.to_datetime(series_b['Date'])
series_a.set_index('Date', inplace=True)
series_b.set_index('Date', inplace=True)
# Allinea le due serie in base alle date
aligned_a, aligned_b = series_a.align(series_b, join='inner')
```

```

# Calcola i rendimenti in eccesso
"""
[Nota dell'Autore: ricordando che il TassoPrivoRischio è annualizzato,
inseriamo manualmente /252 nella riga sottostante]
[Nota dell'Autore: attenzione, l'intestazione per le colonne dei
rendimenti potrebbe
non chiamarsi Returns o RiskFreeRate nei tuoi file. Adatta di
conseguenza.]
"""
excess_returns = aligned_a['Returns'] - aligned_b['RiskFreeRate']/252
# Calcola l'indice di Sharpe
sharpe_ratio = excess_returns.mean() / excess_returns.std()
# Aggiusta per l'annualizzazione se necessario

# Se i rendimenti sono giornalieri, moltiplica l'indice di Sharpe
# per sqrt(numero giorni di trading)
from math import sqrt
trading_days = 252
annualized_sharpe_ratio = sharpe_ratio * sqrt(trading_days)
print("Sharpe Ratio:", sharpe_ratio)
print("Sharpe Ratio Annualizzato:", annualized_sharpe_ratio)

```

Il codice non è del tutto corretto: aligned_a e aligned_b contengono solo le date, non i rendimenti.

Prompt dell'utente

Qualcosa non va nel codice sopra: aligned_a e aligned_b non contengono i rendimenti, solo le date.

ChatGPT sostituisce il codice con

Risposta di ChatGPT

```

...
# Unisci i dataset sulla colonna 'Data'
merged_data = pd.merge(series_a, series_b, on='Date', how='inner')
# Supponendo che la colonna dei rendimenti in series_a si chiami
'Returns' e
# la colonna del tasso privo di rischio in series_b si chiami
'RiskFreeRate'
# Calcola i rendimenti in eccesso
merged_data['ExcessReturns'] = merged_data['Returns'] -
merged_data['RiskFreeRate']
# Calcola l'indice di Sharpe
sharpe_ratio = merged_data['ExcessReturns'].mean() /
merged_data['ExcessReturns'].std()
...

```

... che finalmente ha prodotto:

Sharpe Ratio: 0.06826677308094933
Annualized Sharpe Ratio: 1.0837014262864237

Codice Matlab per calcolare la frontiera efficiente e trovare il portafoglio di tangenza

Questo è il codice da *Machine Trading* (Chan, 2017). Per semplicità, abbiamo ipotizzato che il tasso privo di rischio sia 0. Lo trovate disponibile al download dal nostro repos GitHub.

```
% ef.m
% Trova la frontiera efficiente di un insieme di ETF
% Nota in questo programma media e sd dei rendimenti sono vettori riga,
clear;

load('inputDataOHLCDaily ETF_20150417', 'stocks', 'tday', 'cl');
% Rimuovi EWZ e FXI
stocks=stocks(~strcmp('EWZ', stocks) & ~strcmp('FXI', stocks));
cl=cl(:, ~strcmp('EWZ', stocks) & ~strcmp('FXI', stocks));
R=calculateReturns(cl, 1); % rendimenti "netti" di 1 periodo
R(1, :)=[]; % Salta la prima riga poiché i rendimenti lì sono NaN
mi=mean(R, 1); % rendimento medio di ogni stock i.
C=cov(R); % covarianza dei rendimenti
m=[min(mi):(max(mi)-min(mi))/20:max(mi)]; % prepara diversi rendimenti
medi target del portafoglio per la frontiera efficiente
v=NaN(size(m));
% Varianza del portafoglio da minimizzare
H=2*C;
% Vincolo di vendita allo scoperto
A=-eye(length(mi));
b=zeros(length(mi), 1);
% Nessun termine lineare nella minimizzazione
f=zeros(1, length(mi));
% Fissa il rendimento medio del portafoglio e il vincolo di
normalizzazione
Aeq=[mi; ones(1, length(mi))];
for i=1:length(m)
    beq=[m(i); 1];
    [F, v(i)]=quadprog(H, f, A, b, Aeq, beq);
end
sd=sqrt(v);
scatter(sd, m);
hold on;
% Trova il portafoglio di tangenza
sharpeRatio=m./sd;
```

```

[~, idx]=max(sharpeRatio);
scatter(sd(idx), m(idx), 'red');
beq=[m(idx); 1];
[F]=quadprog(H, f, A, b, Aeq, beq)
% F =
%
% 0.451338068065785
% 0.263444604411169
% 0.000019328104256
% 0.000004913963978
% 0.000007497342085
% 0.285185588112728
% Trova il portafoglio a varianza minima
[~, idxMin]=min(sd);

scatter(sd(idxMin), m(idxMin), 'green', 'filled');
beq=[m(idxMin); 1];
[F]=quadprog(H, f, A, b, Aeq, beq)
% F =
%
% 0.381518036613685
% 0.000033166607068
% 0.604011066007258
% 0.000133505575004
% 0.014299849485899
% 0.000004375711085

```

Otterrete il grafico rappresentato nella Figura A.3, dove il punto pieno (Verde) indica il portafoglio a varianza minima e il cerchio chiaro (Rosso) indica il portafoglio di tangenza (con indice di Sharpe massimo).

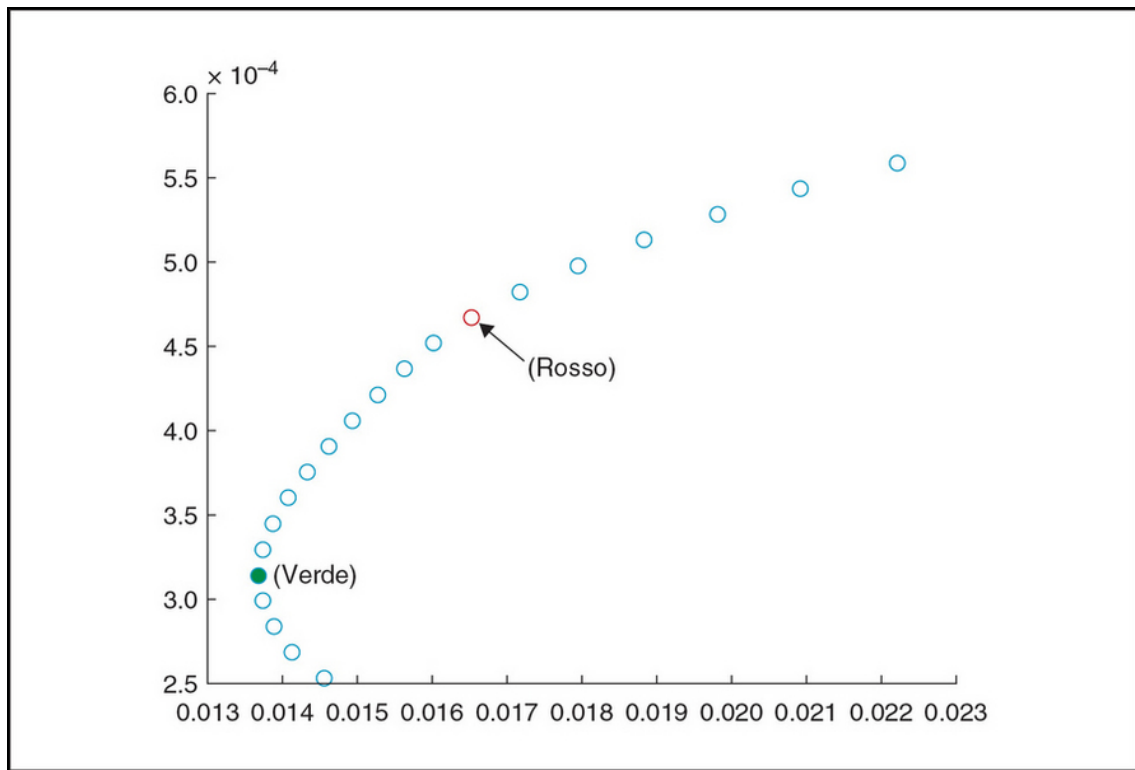


Figura A.3 Grafico della frontiera efficiente prodotta dai listati Matlab.

Indice

Introduzione

Ringraziamenti

Hamlet Jesse Medina Ruiz

Ernest Chan

Gli autori

Parte I - L'AI generativa per il trading e la gestione degli investimenti: un'introduzione no-code

Capitolo 1 - Elementi di AI generativa no-code per la finanza quantitativa

Recupero dei dati storici di mercato

Calcolo dell'indice di Sharpe

Formattazione e analisi dei dati

Traduzione di listati Matlab in codice Python

Conclusioni

Capitolo 2 - AI generativa no-code per lo sviluppo di strategie di trading

Creazione di codice da una specifica di strategia

Riassumere un articolo su una strategia di trading e creare codici di backtest

Ricerca di un algoritmo di ottimizzazione del portafoglio basato
sul machine learning

Esplorare le strategie di arbitraggio sulla struttura a termine delle
opzioni

Conclusioni

Esercizi

Calcolo del rendimento del giorno successivo

Caricamento dei fattori Fama-French

Combinazione dei fattori Fama-French con i rendimenti del giorno
successivo

Capitolo 3 - Il machine learning nella gestione degli investimenti

Apprendimento non supervisionato

Apprendimento supervisionato

Apprendimento per rinforzo profondo

Ingegneria dei dati

Ingegneria delle feature

Conclusioni

Parte II - Modelli generativi profondi per il trading e la gestione degli asset

Capitolo 4 - L'intelligenza artificiale generativa

Perché usare i modelli generativi

Differenze rispetto ai modelli discriminativi

Come possiamo utilizzarli?

I modelli generativi e ChatGPT

Modellazione ibrida: combinare modelli generativi e
discriminativi

Tassonomia dei modelli generativi

Conclusioni

Capitolo 5 - Modelli autoregressivi profondi per la modellazione di sequenze

Complessità della rappresentazione

Rappresentazione e riduzione della complessità

Le principali famiglie di modelli

Adattamento del modello (Model Fitting)

Conclusioni

Capitolo 6 - Modelli profondi a variabili latenti

Introduzione

Modelli a variabili latenti

Esempi di modelli tradizionali a variabili latenti

Apprendimento

Autoencoder variazionali

VAE per dati sequenziali e serie temporali

Conclusioni

Capitolo 7 - Modelli di flusso

Introduzione

Addestramento del modello

Flussi lineari

Progettazione di flussi non lineari

Flussi di accoppiamento

Flussi autoregressivi

Flussi normalizzanti continui

Modellazione di serie temporali finanziarie con i modelli di flusso

Conclusioni

Capitolo 8 - Reti generative antagoniste

Introduzione

Addestramento

Alcuni approfondimenti teorici sulle reti generative antagoniste

Perché l'addestramento delle reti generative antagoniste è così difficile? Tecniche per migliorarlo

Wasserstein GAN

Estendere le reti generative antagoniste per le serie temporali

Conclusioni

Capitolo 9 - Sfruttare gli LLM per l'analisi del sentiment nel trading

Analisi del sentiment nei discorsi alle conferenze stampa della Fed utilizzando Large Language Model

Dati: video e prezzi di mercato

Conversione da parlato a testo

Analisi del sentiment

Risultati sperimentali

Conclusioni

Capitolo 10 - Efficienza dell'inferenza

Introduzione

Regolare la scala degli LLM: prestazioni, costo computazionale e abilità emergenti

Rendere più veloce FinBERT

Quantizzazione del modello

Personalizzare un LLM: adattare i modelli a esigenze specifiche

Conclusioni

Capitolo 11 - Postfazione

Modelli di diffusione

Combinare più varianti dei modelli generativi
LLM come consulenti finanziari

Riferimenti

Appendice

Recupero dei prezzi di chiusura rettificati e calcolo dei rendimenti
giornalieri

Installazione di Python

Tracciamento del tasso privo di rischio nel corso degli anni

Calcolo dell'indice di Sharpe di SPY

Codice Matlab per calcolare la frontiera efficiente e trovare il
portafoglio di tangenza