UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Corso di Laurea in Informatica

Tesi di Laurea Triennale

Applicazione di Modelli Previsionali Basati su Machine Learning Legati all'Ambito Finanziario

RELATORE
Prof. Riccardo Martoglia

LAUREANDO
Mattia Savoia

Anno Accademico 2019/2020

RINGRAZIAMENTI

Ringrazio l'Ing. Riccardo Martoglia per la sua continua disponibilità e assistenza. Inoltre un ringraziamento speciale va a tutta la mia famiglia e ai miei amici per il supporto che mi hanno dato in questi anni.

PAROLE CHIAVE

Serie Temporali

Yahoo Finance

Machine Learning

ARIMA

LSTM

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Introduzione

La previsione del mercato finanziario è un argomento di fondamentale importanza per gli investitori, ma è anche uno dei problemi più complicati nell'analisi del mercato azionario. Questo tipo di previsione può essere ad alto rischio, a causa della sua natura imprevedibile e di indicatori finanziari complicati da analizzare. Nessuno conosce con certezza il momento esatto del mercato ribassista o rialzista. Le informazioni relative ad un mercato sono generalmente incomplete, complesse e incerte, infatti questo comportamento rende molto difficile riuscire a prevedere in modo accurato l'andamento del prezzo futuro.

Prima della comparsa dei computer, gli investitori erano abituati a fare trading di azioni e materie prime in base ai loro sentimenti. Man mano che il livello di interesse nel trading cresceva, le persone cercavano metodi e strumenti che aumentassero i loro guadagni, cercando allo stesso tempo di ridurre al minimo i rischi. Al giorno d'oggi, il trading in borsa ha guadagnato un'enorme popolarità a livello globale, per molte persone è diventata anche parte della routine quotidiana.

Esistono due tipologie di studio utilizzate per riuscire a comprendere fattori che portano a variazioni di prezzo o che tentano di prevedere i prezzi futuri delle azioni:

- Analisi fondamentale: implica l'analisi della redditività futura dell'azienda sulla base delle sue attuali prestazioni finanziarie e del suo contesto economico;
- Analisi tecnica: prevede i movimenti futuri dei prezzi attraverso lo studio delle attività di trading passate (ad esempio il movimento dei prezzi di chiusura).

La previsione dei prezzi tramite tecniche di Machine Learning è un argomento importante nell'analisi tecnica al giorno d'oggi. Molte società di trading stanno studiando e utilizzando metodi basati sull'intelligenza artificiale per riuscire a potenziare il trading automatizzato.

Il presente elaborato ha l'obiettivo di progettare ed implementare modelli basati su Machine Learning per la previsione dei prezzi di mercato. Più precisamente verranno calcolate le previsioni sui tre indici più importanti d'America, ovvero: S&P 500, Dow Jones Industrial e NASDAQ.

Questa tesi è suddivisa in otto capitoli, ciascuno dei quali descrive parti essenziali dell'intero progetto. Più in particolare, il primo capitolo introduce la definizione delle serie temporali. Il secondo capitolo ha il compito di descrivere le tecnologie utilizzate per la creazione del DataFrame, quindi: l'utilizzo di Pandas per la gestione dei dati, la

libreria di Yahoo Finance per recuperare i dati e le librerie Matplotlib e Plotly per la visualizzazione dei grafici. Nel terzo capitolo viene introdotto il Machine Learning e il suo funzionamento generale. Il quarto capitolo si occupa di definire i modelli di previsione utilizzati, quindi ARIMA e LSTM. Nel quinto capitolo vengono introdotte le quattro metriche utilizzate per riuscire a valutare l'efficacia dei modelli. Il sesto capitolo si sofferma sulla progettazione dell'intero applicativo, mentre il settimo capitolo espone la fase di implementazione delle parti di codice più significative. Infine, il capitolo 8 ha il compito di illustrare tutti i risultati di previsione ottenuti e di confrontarli.

Parte I
Studio delle tecnologie utilizzate

Capitolo 1

Serie Temporali

In questo primo capitolo verrà introdotta la definizione di serie temporale. In particolare, verranno spiegati i motivi per il quale le serie temporali sono molto importanti per eseguire un'analisi iniziale dei dati. Sarà spiegato anche il concetto di stazionarietà della serie temporale e di come sia di fondamentale importanza per poter costruire un modello statistico come ARIMA. Infine verrà data una definizione di correlazione tra due variabili all'interno di una serie temporale.

1.1 Introduzione

Una serie temporale, in inglese "time serie", è semplicemente una serie di punti ordinati nel tempo.
Nel campo finanziario, una serie temporale tiene traccia del movimento dei punti dati scelti (come il prezzo di un titolo) in un periodo di tempo specificato e con punti registrati a intervalli regolari. Non esiste un periodo di tempo minimo o massimo che deve essere incluso. Questo consente quindi di raccogliere i dati in modo da fornire le informazioni ricercate dall'investitore o dall'analista che esamina l'attività.
In generale, esistono due tipi di serie temporali:

- Univariate: solo una variabile varia nel tempo. Pertanto, ad ogni passo, si avrà solamente un valore unidimensionale;
- Multivariate: molteplici variabili variano nel tempo. Quindi, ad ogni passo, si avrà un valore n-dimensionale in base al numero di variabili.

Il modo forse più ovvio per prevedere aspetti del futuro è fare riferimento al passato, cercando di individuare strutture ricorrenti in ciò che è accaduto per prevedere ciò che potrebbe accadere, assumendo che le stesse strutture si ripetano anche nel futuro. Una serie temporale è solitamente modellata attraverso un processo stocastico Y(t), cioè una

sequenza di variabili casuali. In un contesto di previsione ci troviamo al tempo t e siamo interessati a stimare Y(t+h), utilizzando solo le informazioni disponibili al tempo t.
Come per ogni altra attività di "data analysis", anche per la previsione delle serie temporali il primo passo è quello di "conoscere i propri dati". Ciò si riferisce ad una fase iniziale di analisi dei dati a disposizione, in modo da riuscire a farsi un'idea di quale sia il metodo migliore per affrontare la previsione. Più precisamente, è molto utile identificare alcune caratteristiche della serie di dati di cui si vogliono prevedere i successivi valori. La scelta del metodo appropriato è una delle decisioni più importanti che un analista deve prendere in considerazione e l'analisi delle serie temporali è uno dei modi migliori per capire come si comporta una serie storica.

1.2 Analisi delle serie temporali

Ci sono quattro componenti principali di cui è composta una serie temporale:

- Componente stagionale (seasonal): coglie la variabilità dei dati nel tempo, anche per effetto delle oscillazioni. La componente stagionale spiega gli alti e bassi periodici. Una serie temporale può contenere più periodi stagionali sovrapposti;
- Componente di tendenza (trend): la direzione in cui i dati stanno andando nel tempo. La componente di tendenza si riferisce al modello nei dati che si estende su periodi stagionali;
- Componente ciclica (cycle): la componente ciclica rappresenta i fenomeni che si verificano in periodi stagionali. Le componenti cicliche non hanno un periodo fisso come hanno le componenti stagionali. Un esempio di un modello ciclico sono i cicli di boom e crollo che hanno i mercati azionari in risposta agli eventi mondiali. La componente ciclica è difficile da isolare e spesso viene "lasciata sola" combinandola con la componente di tendenza;
- Componente irregolare (noise): questa componente cattura la variazione casuale nei dati che non possono essere previsti in anticipo. Sono generalmente causati da fattori a breve termine, imprevisti o eventi non ricorrenti che si verificano nel tempo. La componente irregolare o il rumore è ciò che rimane quando si separa la stagionalità e la tendenza dalle serie temporali. Il rumore è l'effetto di fattori che non si conosce o che non si può misurare. È il risultato delle incognite note o delle incognite sconosciute.

Questa decomposizione viene utilizzata per l'analisi delle serie temporali [1]. Il risultato può essere utilizzato per informare il modello di previsione in base al problema. Fornisce una breve conoscenza del problema di previsione in termini di complessità del modello e del modo migliore per catturare queste componenti nel modello. La

decomposizione ci aiuta ad analizzare meglio i dati e ad esplorare diversi modi per risolvere il problema.

Si possono utilizzare due tipologie di approcci per quanto riguarda l'analisi delle serie storiche:

- Approccio classico: in questo caso le componenti di tendenza, stagionalità e irregolare possono combinarsi in modo additivo o moltiplicativo;
- Approccio moderno: assume che la serie sia stata generata da un processo stocastico a componenti correlate descrivibile con appositi modelli probabilistici. Per poter applicare questo approccio alle serie temporali è quasi sempre necessario eliminare il trend e la stagionalità al fine di avere un processo stazionario.

In questo elaborato, verrà preso in considerazione l'approccio moderno, questo perché uno dei modelli che si andranno ad utilizzare è ARIMA. Esso verrà adattato ai valori delle serie temporali per comprendere meglio i dati e quindi eseguire una previsione dei punti futuri della serie.

1.3 Stazionarietà

In matematica e statistica, un processo stazionario è un processo stocastico in cui alcune sue proprietà non cambiano rispetto ad una traslazione nel tempo. Di conseguenza, parametri come la media e la varianza, non cambiano nel tempo.
Una serie storica si dice stazionaria se:

- La sua media è costante nel tempo (stazionarietà in media);
- La sua varianza è costante nel tempo (stazionarietà in varianza);
- Il rapporto tra valori distanti k periodi è influenzato solo da k, non dal punto della serie in cui viene calcolata (stazionarietà in covarianza).

Una serie stazionaria non ha nessun trend o stagionalità (stazionaria in media) e non ha neanche cicli (stazionaria in varianza).

1.3.1 Controllare la stazionarietà

Esistono diversi metodi per capire se una serie è stazionaria. Il metodo più intuitivo da utilizzare è quello della visualizzazione del grafico della serie temporale. Tramite esso, è possibile capire qual'è l'andamento della serie e controllare se è presente trend o stagionalità.

Un'altro metodo è quello di tracciare la media mobile (rolling mean) e la deviazione standard mobile (rolling standard deviation). Se, sia la media che la deviazione standard sono linee piatte, allora significa che la serie è stazionaria.
Infine il test Dickey-Fuller Aumentato (ADF) è uno dei test statistici più utilizzati. La serie temporale è considerata stazionaria se il valore di p (p-value) è basso (minore di una soglia di 0,05) e i valori critici a intervalli di confidenza dell'1%, 5%, 10% sono il più vicino possibile alle statistiche ADF.

1.3.2 Convertire la serie in stazionaria

Poiché la stazionarietà è un presupposto fondamentale in molte procedure statistiche utilizzate nell'analisi delle serie storiche (come ARIMA), i dati non stazionari sono spesso trasformati per diventare stazionari. Sopratutto per quanto riguarda l'ambito finanziario, una serie temporale relativa ad un'indice o ad un'azienda, difficilmente potrà essere già stazionaria. Esistono diverse tecniche per rendere una serie temporale stazionaria.
Un metodo molto utilizzato è quello della trasformazione logaritmica. Esso rende lineare un trend esponenziale e aiuta a rendere costante la varianza.
Un altro metodo utile a rendere la serie stazionaria è la differenziazione. Essa sottrae il valore corrente dal precedente e può essere utilizzata per trasformare una serie temporale in stazionaria.
Queste due tecniche possono essere anche utilizzate insieme.

1.4 Correlazione

La correlazione è una misura statistica che esprime la relazione lineare tra due variabili (che cambiano insieme a una velocità costante) ed è molto usata per descrivere semplici relazioni.
La correlazione viene descritta mediante un valore che non è dotato di un'unità di misura specifica, chiamato coefficiente di correlazione, compreso tra -1 e +1 e denotato da 'r':

- Più 'r' si avvicina a zero, più la correlazione è debole (le variabili non sono correlate);
- Un valore 'r' positivo implica una correlazione positiva, in cui i valori delle due variabili tendono ad aumentare in parallelo (le variabili sono correlate direttamente);

- Un valore ‘r’ negativo è indice di una correlazione negativa, in cui il valore di una variabile tende ad aumentare quando l’altra diminuisce (le variabili sono correlate inversamente).

Attraverso i coefficienti di correlazione, è possibile costruire una funzione di autocorrelazione (Auto Correlation Function o ACF), dove sull’asse delle X sono presenti i lag temporali, mentre sull’asse delle Y si trovano i coefficienti di correlazione. In questo grafico è anche rappresentata una “banda di confidenza”. Se un coefficiente di correlazione esce da questa banda, a seconda che abbia un valore positivo o negativo, indica la presenza di una correlazione diretta o inversa.
Questo grafico può essere anche utilizzato per verificare la presenza di trend o stagionalità nella serie temporale.

Capitolo 2

Creazione e visualizzazione del DataFrame

Analizzare ed interpretare i dati che provengono dai processi reali è una tematica che ormai fa parte della vita di tutti i giorni. Basta semplicemente eseguire una semplice richiesta ad un motore di ricerca per rendersi conto del fatto che enormi quantità di dati vengono generati ad ogni istante. La stessa cosa accade con i dati finanziari.
Ottenere e salvare i dati relativi al mercato finanziario è il primo passo per riuscire a creare ed impostare i modelli di previsione. Una collezione di dati strutturati che viene viene costantemente caricata, aggiornata e manipolata, prende il nome in inglese di DataFrame. Il presente capitolo ha l'obiettivo di illustrare le tecnologie utilizzate per la creazione e gestione del DataFrame. Infine verranno introdotte le tecnologie utilizzate per la rappresentazione di questi dati, quindi la visualizzazione dei grafici.

2.1 Pandas

Pandas è una libreria Python open source che fornisce strumenti di analisi e manipolazione dei dati ad alte prestazioni utilizzando le sue potenti strutture di dati [2]. Mette a disposizione varie strutture di dati e operazioni per la manipolazione di dati numerici e serie temporali. Questa libreria è costruita sulla base della libreria NumPy [3]. Pandas è veloce e offre elevate prestazioni e produttività per gli utenti. Utilizzando Pandas, è possibile eseguire cinque passaggi tipici nell'elaborazione e nell'analisi dei dati, indipendentemente dall'origine di essi. Questi passaggi sono: caricare, preparare, manipolare, modellare e analizzare.
Python con Pandas viene utilizzato in una vasta gamma di campi, compresi i domini accademici e commerciali tra cui finanza, economia, statistica, analisi, ecc.
Caratteristiche principali di Pandas:

- Veloce ed efficiente per la manipolazione e l'analisi dei dati;
- Funzioni I/O per interagire con i dati salvati in file CSV, EXCEL, database;

- Facile gestione dei dati mancanti (rappresentati come NaN) nei dati in virgola mobile e non in virgola mobile;
- Modifica delle dimensioni: le colonne possono essere inserite ed eliminate da DataFrame e oggetti di dimensioni superiori;
- Fusioni di raggruppamento, combinazione, divisione, merge, join tra insiemi di dati.
- Funzioni di indicizzazione tramite label;
- Allineamento dei dati tramite una serie di label specificate dall'utente oppure impostate automaticamente;
- Fornisce funzionalità per le serie temporali.

2.1.1 Serie e Dataframe

In Pandas, si hanno due principali strutture di dati che si possono esplorare. Il primo è una Serie e il secondo è un DataFrame.

Una Serie è una matrice unidimensionale di valori con un indice. La Serie non è altro che una colonna in un foglio Excel.
Un DataFrame è una matrice bidimensionale di valori con una riga e un indice di colonna. E' una struttura dati bidimensionale, cioè i dati sono allineati in modo tabulare in righe e colonne. Pandas DataFrame è costituito da tre componenti principali, i dati, le righe e le colonne.

Serie

	Open
0	353.3999938964844
1	359.69000244140625
2	358.760009765625
3	355.6700134277344
4	352.20001220703125

DataFrame

	Open	High	Low
0	353.3999938964844	359.69000244140625	351.9800109863281
1	359.69000244140625	360.5899963378906	357.8900146484375
2	358.760009765625	358.760009765625	352.8900146484375
3	355.6700134277344	355.6700134277344	351.3500061035156
4	352.20001220703125	354.239990234375	350.5400085449219

Figura 2.1: Serie e DataFrame

Un DataFrame più essere considerato come l'intero insieme di dati, quindi sono comprese tutte le righe e le colonne. Mentre una Serie è essenzialmente una singola colonna all'interno di quel DataFrame.
Per conoscere l'esatta posizione di una riga è possibile avvalersi degli indici. Sono come un indirizzo, e permettono di accedere a qualsiasi punto di dati di un DataFrame o di una Serie. Nella figura 2.1 l'indice è rappresentabile dalla colonna più a sinistra di tutte, senza nessuna intestazione della colonna (0, 1, 2, 3, 4 è l'indice).
La creazione di queste due strutture dati è un processo abbastanza semplice di Pandas.

Nel mondo reale, viene creato un DataFrame o una serie caricando un insieme di dati dalla memoria esistente. La memorizzazione può essere database SQL, file CSV e file Excel. Un DataFrame o una Serie può essere anche creata attraverso liste, dizionari, da un elenco di dizionari, da valori scalari, ecc.

2.1.2 Funzionalità base

In Pandas esistono molte funzionalità essenziali comuni per la gestione delle strutture dati. Solitamente i DataFrame vengono costruiti basandosi su tabelle esterne, come file CSV, file excel o altro. A seconda del file (in questo caso verrà utilizzato il file .CSV), occorre importare o esportare utilizzando la funzione appropriata. Le due funzioni principali per il caricamento ed il salvataggio del DataFrame sono:

- read_csv: che importa i dati da un file CSV e li memorizza in una struttura DataFrame;
- to_csv: che esporta i dati da una struttura DataFrame e li memorizza in un file CSV.

Una volta creato il DataFrame (chiamiamolo per convenzione 'df') è possibile utilizzare tutte le funzionalità di Pandas. Nella tabella 2.1 sono presenti i metodi più utili che verranno utilizzati in questo in questo elaborato.

Tabella 2.1: Funzionalità base Pandas

Metodo	Significato
serie = pandas.Serie([1, 2, 3])	Creazione di una Serie pandas.
df = pandas.DataFrame([[1, 2], [3, 4], columns=['col1', 'col2'])	Creazione di unDataFrame pandas.
df.info()	Mostra la informazioni del DataFrame.
df.shape	Mostra il numero di righe e colonne del DataFrame.
df.describe()	Mostra una serie di dati di riepilogo (il conteggio delle righe, la media, la deviazione standard, minimo e massimo). Si può anche limitare ad una sola colonna (df['nome_colonna'].describe()).
df.head()	Restituisce di default le prime 5 righe del Dataframe.
df.tail()	Restituisce di default le ultime 5 righe del Dataframe.
df.columns	Restituisce tutte le intestazioni di colonna. Il tipo ritornato non è una lista ma un contenitore speciale definito da Pandas. Si può comunque accedere agli elementi di questo contenitore utilizzando gli indici.
df[['nome_colonna']]	Indicizza un DataFrame a colonna singola con una sola colonna denominata 'nome_colonna'. Ritorna un DataFrame.

Metodo	Significato
df[‘nome_colonna’]	Indicizza una colonna denominata ‘nome_colonna’. Restituisce una Serie.
df.rename(columns={‘old_name’, ‘new_name’})	Viene rinominata una specifica colonna.
df.iloc[i]	Viene utilizzato per ottenere la i-esima riga. Può essere utilizzato anche il metodo dello slicing per ottenere più righe (df.iloc[5:7] ritorna le righe 5 e 6 esclusa la 7). Utilizzando la funzione df.iloc[0, 0] si accede alla prima riga della prima colonna del DataFrame.
df.loc[i]	Simile alla funzione ‘iloc’ con la differenza che possono essere utilizzate delle etichette (df.loc[’riga’, ‘nome_colonna’] accede alla riga e alla colonna selezionata).
df.drop([i])	Eliminazione della riga con indice ‘i’.
df.dropna()	Eliminazione delle righe che contengono valori nulli (NaN).
df.is_null.sum()	Sommario dei valori nulli/NaN presenti nel Dataframe.
df.fillna(n)	Inserisce il valore ’n’ dove i valori sono nulli o mancanti.
df.replace(1, ‘one’)	Sostituisce tutti i valori uguali ad 1 con ‘one’.
del df[‘nome_colonna’]	Elimina la colonna di nome ‘nome_colonna’ dal DataFrame.
df.set_index(‘nome_colonna’)	Viene impostata la colonna di nome ‘nome_colonna’ come indice del DataFrame.
df.sort_values(‘nome_colonna’)	Ordina il DataFrame in base ai valori della colonna ‘nome_colonna’. E’ possibile impostare anche il parametro ‘ascending’ a True o False per indicare che tipo di riordinamento fare.
df[df[‘nome_colonna’] > 1000]	Restituisce solamente le righe che hanno valore maggiore di 1000 nella colonna ‘nome_colonna’. E’ possibile utilizzare & (and) o | (or) per aggiungere condizioni diverse al filtro. Questo è anche chiamato filtro booleano.
df.append(df1)	Aggiunge le righe del DataFrame df1 alla fine del DataFrame df (le colonne devono essere uguali).
df.concat([df1, df2], axis)	Concatena colonne o righe di più DataFrame. Quando la variabile ‘axis’ è uguale a 0, viene impilato il secondo DataFrame sotto al primo. Se la variabile ‘axis’ è uguale ad 1, vengono impilate le colonne nel secondo DataFrame alla destra del primo DataFrame (orizzontalmente). Rileverà automaticamente se i nomi delle colonne sono gli stessi.

2.2 Yahoo Finance

Yahoo Finance [4] è un servizio gratuito messo a disposizione dal famoso motore di ricerca che, oltre alla navigazione tradizionale, propone uno strumento specifico per gli argomenti finanziari. Navigando sul sito web è possibile trovare molti servizi legati alla finanza, alle borse e al mondo degli investimenti. Questa piattaforma permette di creare portafogli virtuali, leggere le principali notizie del settore, monitorare i propri investimenti e gestire la finanza personale.
Inoltre, Yahoo Finance, mette a disposizione i dati storici relativi a criptovalute, azioni, forex, fondi comuni di investimento, futures su materie prime, ETF ed indici. Questi dati possono essere scaricati tramite il sito web di Yahoo, oppure possono essere ricavati attraverso la libreria "yfinance" [5]. Ovviamente per avere la possibilità di aggiornare i dati in modo semplice e veloce verrà utilizzata la libreria messa a disposizione.

Per poter utilizzare la libreria di Yahoo Finance e riuscire a scaricare i dati storici, bisogna prima di tutto capire su che tipo di mercato lavorare. L'obiettivo di questo elaborato è quello di eseguire una previsione sui tre indici più importanti d'America: S&P 500, Dow Jones Industrial e NASDAQ. Un indice azionario viene utilizzato per descrivere l'andamento dei mercati azionari, o di una parte di essi. Un indice, infatti, include diversi titoli (a volte anche abbastanza diversi) che rappresentano una sezione dell'intero mercato:

- S&P 500 è formato dalle 500 aziende statunitensi con maggiore capitalizzazione;
- Dow Jones Industrial, a differenza di altri indici, non tiene conto della capitalizzazione, ma rappresenta l'andamento dei primi 30 titoli del NYSE (New York Stock Exchange);
- NASDAQ rappresenta l'indice per ciò che riguarda i titoli del settore tecnologico della borsa americana. Sono presenti anche compagnie informatiche come Microsoft, Amazon, Apple, IBM, Google, Facebook.

Un altro aspetto molto importante per l'utilizzo della libreria di Yahoo Finance è l'utilizzo dei simboli azionari (in inglese Ticker). Il simbolo azionario è un'abbreviazione utilizzata per identificare in modo univoco le azioni di un determinata azienda quotata in borsa. Un Ticker può essere composto da lettere, numeri o una combinazione di entrambi. Il Ticker di S&P 500 è "^GSPC", quello del Dow Jones Industrial è "^DJI" e quello del NASDAQ è "^IXIC".

2.2.1 Metodi principali

La libreria di Yahoo Finance offre molti metodi utili per avere una panoramica generale del mercato su cui si sta lavorando. Nella tabella 2.2 vengono mostrate alcune delle funzioni principali di questa libreria. Per poter accedere ai dati relativi ad un'azienda o ad un indice, è necessario specificare il Ticker relativo.

Tabella 2.2: Metodi principali della libreria Yahoo Finance

Metodo	Significato
yfinance.Ticker("ticker")	Consente di accedere ai dati.
yfinance.Ticker("ticker").info	Ritorna le informazioni in riferimento al Ticker selezionato.
yfinance.Ticker("ticker").history(perdiod)	Mostra i dati storici di mercato in relazione al Ticker impostato. E' possibile impostare il periodo che si vuole selezionare.
yfinance.Ticker("ticker").actions	Mostra le azioni come ad esempio i dividendi e gli splits.
yfinance.Ticker("ticker").financials	Mostra i dati finanziari in riferimento al Ticker selezionato. E' possibile recuperare anche i dati finanziari trimestrali attraverso il metodo 'quarterly_financials'.
yfinance.Ticker("ticker").balance_sheet	Mostra il bilancio. E' possibile mostrare anche il bilancio trimestrale attraverso il metodo 'quarterly_balance_sheet'.
yfinance.Ticker("ticker").earnings	Mostra i guadagni. Attraverso il metodo 'quarterly_earnings' è possibile mostrare i guadagni trimestrali.
yfinance.download(ticker, start, end)	Ottiene lo storico dei dati di mercato in riferimento al Ticker utilizzato. Può essere passato alla funzione anche più di un Ticker. E' possibile impostare la data di inizio e la data di fine.

2.2.2 Download dei dati storici

Nella tabella 2.2 viene mostrata la funzione "download()" della libreria di Yahoo Finance. Questo metodo ritorna un DataFrame Pandas con al suo interno lo storico dei dati di mercato (l'intervallo tra un valore e l'altro è di un giorno).

Il periodo preso in considerazione parte dall'1 Gennaio 1990 fino ad arrivare 17 Febbraio 2021. In questo periodo di tempo, sono considerati solamente i giorni che vanno dal lunedì al venerdì (compresi). Questo perché nel weekend la borsa è chiusa. I dati che verranno ricavati sono:

- Date: data (rappresenta l'indice dell'intero DataFrame);
- Open: prezzo di apertura (valore di apertura dell'indice in quel giorno);
- High: prezzo raggiunto più alto in quel giorno;
- Low: prezzo raggiunto più basso in quel giorno;

- Close: prezzo di chiusura (valore di chiusura dell'indice in quel giorno);
- Adj close: prezzo di chiusura (valore di chiusura dell'indice in quel giorno). A differenza del valore "Close" che indica il prezzo di chiusura di un indice nel giorno di negoziazione, "Adj close" è un'analisi più complessa che utilizza il prezzo di chiusura come punto di partenza. Infatti tiene conto di fattori come dividendi e frazionamenti azionari. I frazionamenti azionari si verificano raramente. Questo succede quando il prezzo delle singole azioni di una società è troppo alto perché gli investitori possano acquistarne un numero tondo e quindi il suo prezzo viene frazionato, aumentando quindi il numero di azioni. Mentre per quanto riguarda i dividendi, quando un titolo acquista valore, la società può scegliere di ricompensare gli azionisti con un dividendo. In questo caso il dividendo riduce il valore del titolo perché la società si sbarazza di parte del suo valore pagando i dividendi. "Adj close" quindi mostra il valore del titolo dopo la pubblicazione di un dividendo o di un frazionamento creando, a volte, una differenza tra il prezzo tra il valore "Close" e "Adj close". Nella maggior parte dei casi il prezzo di "Close" e "Adj close" è lo stesso, ma non è sempre così, quindi per una più accurata analisi verrà preso in considerazione il campo "Adj close";
- Volume: volume in quel giorno.

S&P 500

Date	Open	High	Low	Close	Adj_close	Volume
1990-01-02	353.3999938964844	359.69000244140625	351.9800109863281	359.69000244140625	359.69000244140625	162070000
1990-01-03	359.69000244140625	360.5899963378906	357.8900146484375	358.760009765625	358.760009765625	192330000
1990-01-04	358.760009765625	358.760009765625	352.8900146484375	355.6700134277344	355.6700134277344	177000000
1990-01-05	355.6700134277344	355.6700134277344	351.3500061035156	352.20001220703125	352.20001220703125	158530000
1990-01-08	352.20001220703125	354.239990234375	350.5400085449219	353.7900085449219	353.7900085449219	140110000
1990-01-09	353.8299865722656	354.1700134277344	349.6099853515625	349.6199951171875	349.6199951171875	155210000

Figura 2.2: S&P 500 DataFrame

Dow Jones Industrial

Date	Open	High	Low	Close	Adj_close	Volume
1992-01-02	3152.10009765625	3172.6298828125	3139.31005859375	3172.39990234375	3172.39990234375	235500
1992-01-03	3172.39990234375	3210.639892578125	3165.919921875	3201.5	3201.5	236200
1992-01-06	3201.5	3213.330078125	3191.860107421875	3200.10009765625	3200.10009765625	272800
1992-01-07	3200.10009765625	3210.199951171875	3184.47998046875	3204.800048828125	3204.800048828125	255100
1992-01-08	3204.800048828125	3229.199951171875	3185.820068359375	3203.89990234375	3203.89990234375	290400
1992-01-09	3203.89990234375	3228.31005859375	3192.75	3209.5	3209.5	298200

Figura 2.3: Dow Jones Industrial DataFrame

NASDAQ

Date	Open	High	Low	Close	Adj_close	Volume
1990-01-02	452.8999938964844	459.29998779296875	452.70001220703125	459.29998779296875	459.29998779296875	110720000
1990-01-03	461.1000061035156	461.6000061035156	460.0	460.8999938964844	460.8999938964844	152660000
1990-01-04	460.3999938964844	460.79998779296875	456.8999938964844	459.3999938964844	459.3999938964844	147950000
1990-01-05	457.8999938964844	459.3999938964844	457.79998779296875	458.20001220703125	458.20001220703125	137230000
1990-01-08	457.1000061035156	458.70001220703125	456.5	458.70001220703125	458.70001220703125	115500000
1990-01-09	459.20001220703125	459.6000061035156	456.6000061035156	456.79998779296875	456.79998779296875	131130000

Figura 2.4: NASDAQ DataFrame

Un piccolo accorgimento: quando viene scaricato il DataFrame dalla libreria di Yahoo Finance, la colonna del prezzo di chiusura aggiustato è "Adj close". Essa viene rinominata in "Adj_close" per una maggiore usabilità.
Una volta ricavato il DataFrame viene salvato in un file .CSV e caricato ogni volta che si ha la necessità di leggere i dati.

2.3 Matplotlib e Plotly

Durante gli anni sono state sviluppate molte librerie per rappresentare i dati con Python. In questa ricerca verranno utilizzate due librerie principali per la creazione dei grafici: Matplotlib [6] e Plotly [7].

Matplotlib, creata da John D. Hunter e scritta principalmente in python, è una libreria open source. E' stata la prima libreria messa a disposizione degli utenti Python per poter creare i grafici. Può essere utilizzata negli script Python, nelle shell Python e IPython, nel notebook Jupyter e nei server di applicazioni web. La libreria è ampia, in grado di modificare i minimi dettagli di una figura e può rappresentare un'ampia gamma di visualizzazioni 2D. Infatti con Matplotlib si ha la possibilità di creare molte tipologie di grafici, come:

- Grafici a linee;
- Grafici a dispersione;
- Grafici dell'area;
- Grafici a barre e istogrammi;
- Grafici a torta;
- Grafici a stelo;
- Spettrogrammi.

Plotly è sempre una libreria open source per la creazione di grafici. Le interfacce sono disponibili per Python, R, MATLAB e React. Può essere utilizzata per creare e visualizzare figure, aggiornare figure, passare il mouse sul testo per i dettagli ed ha

anche una funzionalità aggiuntiva per l'invio di dati ai server cloud. Una funzionalità molto importante, la quale verrà utilizzata in questo scritto, è la possibilità di salvare i grafici su file HTML utilizzabili in locale.
Plotly ha molte sovrapposizioni con Matplotlib in termini di gamma di visualizzazioni disponibili, infatti si ha la possibilità di creare grafici come:

- Grafici di base: a linee, a torta, a dispersione, a bolle, a punti, ad albero, a raggiera, grafici ad area piena;
- Stili statistici: errore, riquadro, istogrammi, grafici ad albero, grafici a violino, linee di tendenza, grafici di faccette e tralicci;
- Carte scientifiche: Contour, Ternary, Log, Quiver, Radar, Heat map Windrose e Polar Plots;
- Grafici finanziari;
- Sottotrame;
- Mappe.

2.3.1 Rappresentazione del DataFrame

Una volta ottenuto il DataFrame è quindi possibile creare i grafici per la rappresentazione dei dati (grazie all'utilizzo delle librerie per la visualizzazione introdotte nella sezione 2.3). Per ottenere ciò, è necessario prima di tutto capire quali sono i dati realmente utili alla visualizzazione del grafico. Il DataFrame a disposizione è formato da 6 colonne (Open, High, Low, Close, Adj_close e Volume) e dal relativo indice (Date). L'obiettivo finale del presente elaborato è quello di eseguire previsioni dell'andamento dei tre indici azionari (S&P 500, Dow Jones Industrial e NASDAQ). Per andamento viene inteso il prezzo di chiusura giornaliero in relazione ad un determinato periodo di tempo. Quindi i dati utilizzati per la realizzazione del grafico sono:

- Asse delle x: vengono rappresentate le date;
- Asse delle y: vengono rappresentati i valori di chiusura (Adj_close).

I grafici illustrati nelle figure 2.5, 2.6 e 2.7 rappresentano l'andamento storico degli indici (serie temporali) e vengono salvati in un file HTML. Viene eseguito questo salvataggio perché è possibile utilizzare questi file in locale e quindi interagire col grafico: spostarsi all'interno del grafico selezionando un certo intervallo di tempo, scaricare il file .png del grafico, ricavare il prezzo posizionando il mouse sopra il grafico, ecc..

Figura 2.5: S&P 500 storico

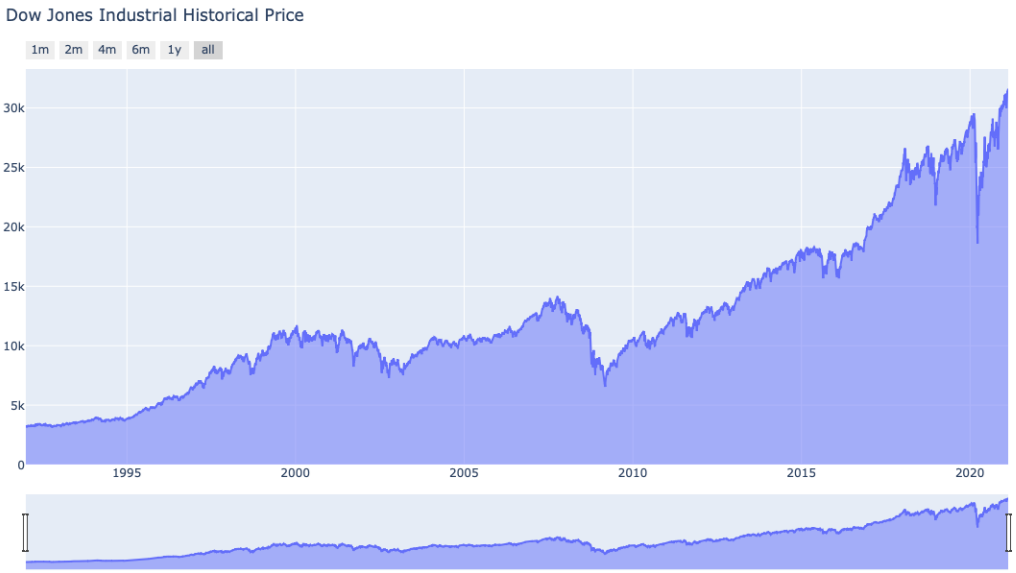

Figura 2.6: Dow Jones Industrial storico

Figura 2.7: NASDAQ storico

Capitolo 3

Machine Learning

Il desiderio di riuscire a produrre previsioni accurate di fenomeni futuri è un aspetto che ha accompagnato il genere umano sin dai suoi albori. Oggi è l'intelligenza artificiale a orientare gli sviluppi in questo ambito. Il futuro dell'intelligenza artificiale nel settore finanziario è in ascesa, molte aziende infatti stanno investendo su questa tecnologia per riuscire a guadagnare più soldi in fretta e cercare di diminuire il rischio. In questo capitolo verrà introdotto il Machine Learning e il suo funzionamento generale.

3.1 Introduzione

L'intelligenza artificiale (AI) è un termine generico e si riferisce a sistemi o macchine che imitano l'intelligenza umana. Il Machine Learning (in italiano "apprendimento automatico") è una componente (o sottoinsieme) dell'intelligenza artificiale. I due termini (Machine Learning e intelligenza artificiale) vengono spesso utilizzati insieme e in modo interscambiabile, ma non hanno lo stesso significato. Un'importante distinzione è che sebbene tutto ciò che riguarda il Machine Learning rientra nell'intelligenza artificiale, l'intelligenza artificiale non include solo il Machine Learning. Il Machine Learning nasce nel 1952 con Arthur Samuel che ne coniò il termine [8] ed è ritenuto al giorno d'oggi uno degli approcci più importanti dell'intelligenza artificiale. E' composto da due parole: "Machine" che corrisponde a un robot o un computer e "Learning" che si riferisce ad un'attività intesa ad acquisire o scoprire modelli di eventi, in cui noi umani siamo bravi.
Definire in maniera semplice le caratteristiche e le applicazioni del Machine Learning non è così semplice, dato che questo ramo è molto vasto e prevede diverse modalità, tecniche e strumenti per essere realizzato. Si può tuttavia dire che un sistema di Machine Learning utilizza differenti meccanismi che permettono di migliorare le proprie competenze e prestazioni nel tempo. Sarà quindi in grado di imparare a svolgere determinati compiti migliorando le proprie capacità, risposte e funzioni tramite

l'esperienza. Alla base dell'apprendimento automatico ci sono una serie di differenti algoritmi che, partendo da nozioni primitive, sapranno prendere una specifica decisione piuttosto che un'altra o effettuare azioni apprese nel tempo.

Il processo include:
- L'apprendimento da dati (i quali si possono manifestare in modi diversi);
- Valutazione dei dati;
- Ottimizzazione dei risultati dei modelli e formulazione di una risposta/azione.

Quindi, in breve, il compito principale del Machine Learning è quello di esplorare e costruire algoritmi in grado di apprendere dai dati storici e fare previsioni su nuovi dati di input.

3.2 Modalità di apprendimento

A seconda della natura dei dati di apprendimento, le attività di Machine Learning possono essere classificate a grandi linee nelle seguenti tre categorie:
- Apprendimento supervisionato (Supervised Learning): in questo tipo di apprendimento, l'obiettivo è quello trovare una regola generale che associ l'input all'output. Vengono forniti al modello sia dei dati in input, sia le informazioni ai relativi output, con l'obiettivo di identificare regole generali o pattern che associno le informazioni iniziali (dati di input) con le informazioni finali (dati di output). In questo modo, quando si è di fronte ad un problema, non si dovrà fare altro che attingere alle esperienze inserite nel proprio sistema, analizzarle, e decidere quale risposta dare sulla base di esperienze già codificate. Questi tipi di dati di apprendimento sono chiamati dati etichettati. La regola appresa viene quindi utilizzata per etichettare nuovi dati con output sconosciuto. L'apprendimento supervisionato viene comunemente utilizzato nelle applicazioni quotidiane, come il riconoscimento di volti e parole, consigli su prodotti o film e previsioni di vendita. E' possibile suddividere ulteriormente l'apprendimento supervisionato in: regressione e classificazione. La regressione si allena e prevede una risposta a valore continuo, ad esempio, prevedendo i prezzi del mercato azionario, mentre la classificazione tenta di trovare l'etichetta di classe appropriata, come l'analisi di un sentimento positivo/ negativo;
- Apprendimento non supervisionato (Unsupervised Learning): contrariamente all'apprendimento supervisionato, il sistema riceve solo l'insieme di dati di input senza alcuna indicazione sugli output da ottenere. I dati contengono solo segnali indicativi senza alcuna descrizione allegata, infatti questi tipi di dati di apprendimento sono chiamati dati senza etichetta. Dovrà essere la macchina stessa, quindi, a catalogare tutte le informazioni in proprio possesso, organizzarle ed imparare il loro significato, il loro utilizzo e, soprattutto, il risultato a cui esse

portano. L'apprendimento non supervisionato può essere utilizzato per rilevare anomalie (come frodi o apparecchiature difettose), per raggruppare clienti con comportamenti (online) simili per una campagna di marketing o nei motori di ricerca;

- Apprendimento per rinforzo (Reinforcement Learning): rappresenta probabilmente il sistema di apprendimento più complesso. Esso prevede l'utilizzo di sistemi e strumenti in grado di migliorare il proprio apprendimento e di comprendere le caratteristiche dell'ambiente circostante. Il sistema interagisce con un ambiente dinamico che lo guida verso il miglioramento delle proprie prestazioni attraverso dei feedback. Infatti i dati di apprendimento forniscono feedback in modo che il sistema si adatti alle condizioni dinamiche per raggiungere un determinato obiettivo. Il sistema valuta le sue prestazioni in base alle risposte di feedback e reagisce di conseguenza. A differenza dei due precedenti metodi di apprendimento, quello per rinforzo non ha la necessità di utilizzare un insieme di dati iniziali da cui partire, ma può semplicemente prende le sue prime scelte in modo assolutamente casuale.

3.3 Generalizzare con i dati

L'aspetto positivo dei dati è che ce ne sono molti nel mondo, l'aspetto negativo, invece, è che è difficile elaborarli. Essi non sono tutti uguali e sicuramente non hanno tutti gli stessi valori. Gli esseri umani di solito elaborano dati in relazione ai propri sensi (occhi, orecchie) e vengono trasformati in segnali elettrici o chimici. I computer funzionano un po' allo stesso modo, elaborano segnali elettrici e li trasformano in zero e uno. I dati sono elementi con cui il sistema di Machine Learning entra in contatto.
Nella prima fase del processo di apprendimento automatico, il sistema riceve in ingresso un insieme di dati su cui si deve addestrare, chiamato appunto "Training set" o "Training samples". Durante questa fase, il sistema deve mettere in relazione l'input con l'output, quindi fare una stima delle strutture ricorrenti e delle regole che osserva nei dati di addestramento. A questo punto il sistema, sulla base delle sue esperienze, dovrebbe essere in grado di prevedere un output dato un input mai visto prima, cioè mettere in pratica quello che il sistema ha imparato dai dati di addestramento.
La cosa molto importante è riuscire a valutare l'efficienza su informazioni future che possono essere diverse da quelle che ha già assorbito il sistema. Entra quindi in gioco un'altro insieme di dati, quello sul quale verrà testato il sistema dopo che si è addestrato. Esso è appunto chiamato "Testing set" o "Testing samples". Durante questa fase verranno utilizzate metriche apposite per riuscire a calcolare l'efficienza del sistema, cioè quanto è accurata la previsione rispetto al valore reale.

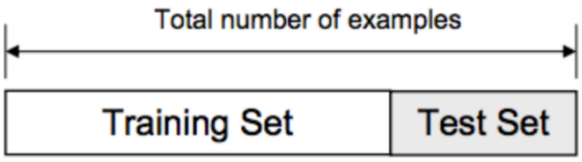

Figura 3.1: Training set e Test set

La divisione in training set e test set è bastata sull'intero DataFrame di dati. Non esiste una percentuale di divisione ottimale. Bisogna scegliere una percentuale che soddisfi gli obiettivi del progetto tenendo in considerazione:

- Costo computazionale nell'addestramento del modello;
- Costo computazionale nella valutazione del modello;
- Rappresentatività del training set;
- Rappresentatività del test set.

Le percentuali di divisione comuni sono:

- Training: 80%, Test: 20%;
- Training: 70%, Test 30%;
- Training: 67%, Test: 33%;
- Training: 50%, Test: 50%.

Spesso, tra le due categorie di insiemi di dati utilizzati, c'è di mezzo un altro insieme di dati, quello per la validazione. Esso è chiamato appunto "Validation set" o "Validation samples". Viene utilizzato per verificare le prestazioni dei sistemi in un ambiente simulato, riuscendo ad ottimizzare di conseguenza i modelli per ottenere risultati più accurati.
Probabilmente, i modelli (o sistemi) di Machine Learning hanno un unico scopo: generalizzare nel modo corretto. La generalizzazione viene intesa come la capacità di un sistema o di una macchina di portare a termine compiti nuovi, che non ha mai visto prima, in maniera accurata. Più precisamente è la capacità del modello di fornire output accurati utilizzando insiemi di dati di input sconosciuti. Questo consente di fare previsioni del futuro sui dati che il modello non ha mai incontrato.

3.4 Overfitting e Underfitting

Termini come Overfitting (sovra-adattamento) e Underfitting (sotto-adattamento) si riferiscono a carenze di cui potrebbero soffrire le prestazioni del modello, raggiungendo quindi scarse performance dopo l'addestramento. Un modello che generalizza bene è un modello che non è né sovra-adattato né sotto-adattato. Oltre però alla generalizzazione, bisogna tenere conto di due concetti importanti per capire le carenze di un modello:

- Bias (o distorsione): è la differenza tra la previsione media del modello e il valore reale che si sta cercando di prevedere. Il bias non è altro che l'errore sistematico che non dipende dalla casualità dei dati (misura quanto il sistema si adatta ai dati). Il modello con distorsione elevata (high bias) presta pochissima attenzione ai dati di addestramento e semplifica eccessivamente il modello, portando così a errori elevati. Mentre un modello con distorsione bassa (low bias) indica che i punti previsti sono vicini a quelli reali;
- Varianza: rappresenta la variabilità della previsione del modello quando viene addestrato più volte su insiemi di dati diversi. Indica, quindi, quanto il sistema è sensibile alla casualità dei dati nell'insieme di dati di addestramento. Un modello con varianza elevata (high variance) presta molta attenzione ai dati di addestramento, tuttavia non generalizza i dati che non si sono mai visti prima. Di conseguenza, tali modelli si comportano molto bene sui dati di allenamento, ma hanno una precisione molto bassa sui dati di test.

Nell'apprendimento supervisionato, l'overfitting si verifica quando il modello è troppo complesso e sensibile ai dati di training (high variance). Questo può accadere quando si estraggono troppe informazioni dai dati di training, inducendo il modello a funzionare bene con essi. Per individuare l'overfitting, un primo passo, potrebbe essere quello di verificare cosa accade ai dati di addestramento e di test. Se il modello funziona notevolmente meglio sui dati di allenamento rispetto ai dati di test, allora probabilmente il modello si sta adattando troppo (quindi ci potrebbe essere la presenza di overfitting). Questi modelli hanno spesso un basso bias e una varianza elevata.

Sempre prendendo in considerazione un tipo di apprendimento supervisionato, l'underiftting si verifica quando il modello non è in grado di acquisire pattern o regole generali durante la fase di addestramento. Più semplicemente: il processo di apprendimento è troppo semplice, quindi non si ha una grandezza adeguata dati di addestramento per fare in modo che il modello riesca ad imparare. L'underfitting spesso non viene discusso in quanto è facile da rilevare, infatti si verifica quando il modello non funziona bene sui dati ti training. Questi modelli di solito hanno un bias elevato e una bassa varianza. Per migliorare l'accuratezza di un modello che soffre di underfitting si possono provare algoritmi di apprendimento automatico alternativi oppure modificare alcuni parametri per adattarli meglio ai dati.

L'obiettivo di qualsiasi algoritmo di Machine Learning supervisionato è ottenere un bias basso e una varianza bassa. Se il modello presenta queste caratteristiche, allora l'algoritmo dovrebbe ottenere una buona accuratezza di previsione. Gli algoritmi di Machine Learning parametrici o lineari hanno spesso un bias elevato ma una varianza bassa, mentre gli algoritmi non parametrici o non lineari hanno spesso un bias basso ma una varianza elevata. Non c'è modo di evitare la relazione tra bias e varianza nel

Machine Learning, infatti diminuendo la varianza aumenterà il bias e diminuendo il bias aumenterà la varianza.
Per costruire un buon modello, si ha la necessità trovare un buon equilibrio tra bias e varianza in modo tale da ridurre al minimo l'errore totale (generalization error).

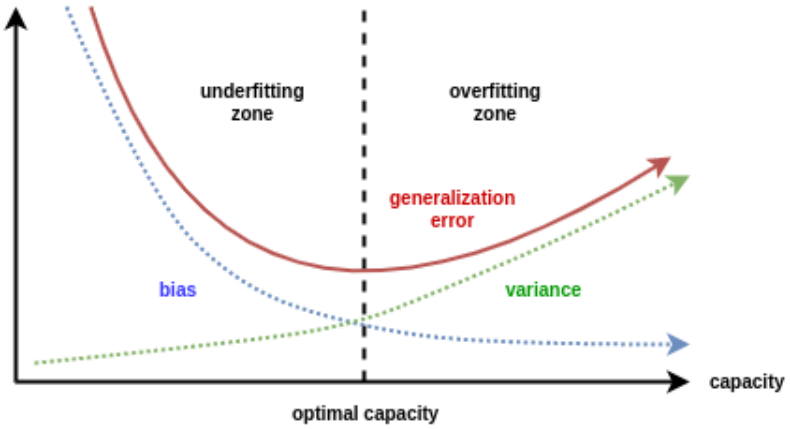

Figura 3.2: Overfitting e Underfitting

Un equilibrio ottimale di bias e varianza non implicherebbe mai un overfitting o underfitting del modello.

3.5 Evitare Overfitting

Evitare l'underfitting risulta molto più semplice rispetto a evitare l'overfitting, infatti l'underfitting si verifica quando il modello non si comporta bene sui dati di addestramento e questo comportamento si riesce a notare fin da subito. Mentre l'overfitting, può essere più complicato da individuare. A volte si può ottenere un modello che raggiunge prestazioni elevate sui dati di training e quindi è possibile pensare che il sistema sia pronto all'utilizzo. Bisogna invece fare un passo in più per comprendere se l'elevata accuratezza non sia dovuta alla presenza di overfitting e quindi controllare le prestazioni del modello anche sui dati di test. Esistono diverse tecniche per cercare di contrastare l'overfitting [9].

3.5.1 Dimensione del DataFrame

In generale, più dati di allenamento vengono forniti al modello e meno è probabile che essi si adattino troppo. Quando viene utilizzata una quantità di dati maggiore, il sistema diventa incapace di sovrautilizzare tutti i campioni ed è costretto a generalizzare per fare dei progressi.

L'utilizzo di un numero maggiore di dati è infatti uno degli aspetti principali durante un'attività di analisi dei dati. Disponendo quindi di un DataFrame (o DataSet) di grandi dimensioni, si potrà aumentare considerevolmente la precisione del modello, riducendo allo stesso tempo anche la possibilità di un eccesso di adattamento.

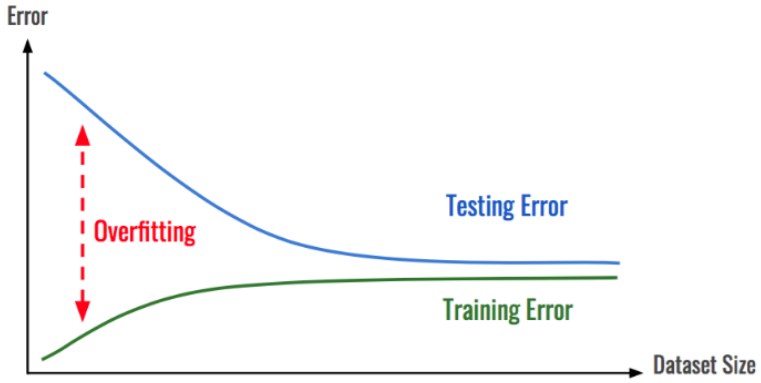

Figura 3.3: Dimensione del DataFrame

3.5.2 Convalida incrociata

La convalida incrociata (in inglese cross validation) è un metodo utilizzato per stimare l'abilità dei modelli di Machine Learning. Essa è una potente misura utilizzata per cercare di prevenire l'overfitting.

In un'impostazione di convalida tradizionale, i dati originali vengono suddivisi in tre sottoinsiemi. Di solito il 60% per quanto riguarda i dati di training, il 20% per i dati di validazione (Validation set) e il restante 20% per i dati di test. Questa impostazione è possibile solamente se si hanno abbastanza dati di addestramento dopo la suddivisione e si ha solo bisogno di una stima approssimativa delle prestazioni simulate. In caso contrario, è consigliabile utilizzare la convalida incrociata.

Il concetto fondamentale su cui si basa il funzionamento della convalida incrociata, è quello di non utilizzare l'intero training set durante l'addestramento del modello, una parte dei dati infatti viene rimossa dai dati di training prima dell'inizio dell'allenamento. Al termine della formazione del modello, esso viene testato sull'insieme di dati che sono stati precedentemente rimossi per poter verificare la precisione. In definitiva, il modello viene valutato solamente dopo che è stato addestrato. Insieme alla formazione del modello, la convalida incrociata punta a trovare un modello ottimale con le migliori prestazioni.

Esistono diverse tipologie di cross validation, come:

- Leave-One-Out-Cross-Validation (LOOCV);
- K-Fold standard;
- Holdout Method;

- Stratified K-Fold Cross-Validation;
- Leave-P-Out Cross-Validation.

3.5.3 Regolarizzazione

La regolarizzazione è un altro modo per prevenire l'overfitting. L'inutile complessità del modello è fonte di overfitting. La regolarizzazione aggiunge parametri in più alla funzione di errore (loss function) che si sta cercando di minimizzare, così da penalizzare i modelli complessi.
La regolarizzazione si riferisce a una vasta gamma di tecniche utilizzate per forzare artificialmente il modello in modo che sia più semplice, e il metodo utilizzato dipenderà dal tipo di algoritmo che viene allenato.
Bisogna tenere in considerazione che la regolarizzazione deve essere mantenuta a un livello moderato o, per essere più precisi, messa a punto a un livello ottimale:
- Una regolarizzazione troppo piccola non ha alcun impatto;
- Una regolarizzazione troppo grande si tradurrà in un underfitting, poiché allontana il modello dalla realtà.

3.5.4 Rimozione delle funzionalità/caratteristiche

La selezione delle caratteristiche (features) è il processo di selezione di un sottoinsieme di caratteristiche significative da utilizzare per costruire il modello nei migliore dei modi.
Il numero di features corrisponde alla dimensionalità dei dati. L'approccio al Machine Learning dipende dal numero di dimensioni rispetto al numero di esempi. Ad esempio, i dati di testo e immagini hanno dimensioni molto grandi, mentre i dati relativi al mercato finanziario hanno dimensioni relativamente inferiori.
Quindi, invece di utilizzare tutte le funzionalità, è meglio eliminare quelle ridondanti o irrilevanti e utilizzare solo quelle più importanti. Questo, da un lato, renderà il processo di addestramento notevolmente più veloce, dall'altro, può aiutare a prevenire un eccesso di adattamento (overfitting) perché il modello non ha bisogno di apprendere funzionalità inutili.

Capitolo 4

Modelli

Esistono diversi modelli di Machine Learning applicabili alle serie temporali per eseguire previsioni. In questo capitolo verranno presi in considerazione due modelli: ARIMA e LSTM. Verrà introdotto il loro funzionamento descrivendo nel dettaglio le loro caratteristiche e i loro particolari meccanismi.

4.1 Introduzione

La previsione dei dati delle serie storiche è un argomento molto importante nella finanza, economia e affari. Esistono diverse tecniche per prevedere in modo efficace i dati delle serie temporali. In particolare, il modello ARIMA ha dimostrato la sua efficacia in termini di precisione e accuratezza nella previsione dei dati delle serie storiche [10]. Con il recente progresso della potenza di calcolo dei computer e, cosa più importante, grazie allo sviluppo di algoritmi e approcci di Machine Learning più avanzati come il Deep Learning, è stato possibile sviluppare nuovi algoritmi per la previsione di dati delle serie storiche. Le reti neurali ne sono un esempio, infatti algoritmi come "Long Short-Term Memory" (LSTM) [11], possono avere molte più potenzialità rispetto agli algoritmi tradizionali, anche se non è sempre così.
Inizialmente è stata la comunità statistica a concentrarsi maggiormente sullo sviluppo di metodi per la previsione di serie temporali. Più recentemente, anche la comunità informatica ha contribuito.
Il modello ARIMA, è definito come un "metodo classico", proveniente quindi dalla comunità statistica. LSTM, invece, è definito come un "metodo di Machine Learning", riferendosi quindi a tecniche di ispirazione più informatiche. È importante tenere a mente che in alcuni casi questa distinzione potrebbe non essere totalmente accettata. Infatti la discussione su dove giaccia il fragile confine tra la statistica e il Machine Learning è ancora molto incerto.

4.2 ARIMA

L'acronimo ARIMA sta per Auto-Regressive Integrated Moving Average ed è uno degli strumenti più diffusi per effettuare previsioni di serie temporali [12]. Questo modello viene ampiamente utilizzato nel campo dell'economia e della finanza in quanto è noto per essere efficiente, robusto e ha un forte potenziale per la previsione del prezzo di mercato a breve termine. Nella previsione delle serie temporali, ARIMA è una generalizzazione di un modello a media mobile auto-regressiva (ARMA). Entrambi questi modelli vengono adattati ai dati delle serie temporali per comprendere meglio i dati o per prevedere i punti futuri della serie (previsione). Il modello ARIMA viene utilizzato in alcuni casi in cui i dati mostrano evidenza di non stazionarietà nel senso di media, dove può essere applicata una fase di differenziazione iniziale (che corrisponde alla parte integrata del modello) più volte per eliminare la non stazionarietà della funzione (il trend e la stagionalità). Il modello ARIMA converte quindi dati non stazionari in dati stazionari prima di occuparsi dei passi successivi.
ARIMA può essere diviso in tre categorie:

- Non stagionale (ARIMA);
- Stagionale (SARIMA);
- Multivariata (ARIMAX).

I modelli ARIMA regolari (non stagionali), sono modelli di serie temporali univariate perché funzionano su dati che consistono in singole osservazioni di una variabile dipendente su intervalli di tempo regolari e senza variabili predittive esterne.

4.2.1 Composizione ARIMA

I modelli ARMA sono composti da due componenti fondamentali: AR (Auto-Regressive) e MA (Moving Average). ARIMA è un'espansione del modello ARMA, composto quindi dai modelli AR e MA, con l'aggiunta di una fase di integrazione I (Integrated).

Auto-Regressive (AR):
I modelli auto-regressivi operano sul presupposto che i valori passati hanno un effetto sui valori attuali. I modelli AR vengono solitamente utilizzati per analizzare dati finanziari, economici e altri processi che variano nel tempo. Quindi, è possibile costruire un modello di regressione lineare che cerca di prevedere il valore di una variabile dipendente (valore attuale), dati i valori che aveva nei giorni precedenti (valori passati).

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1 \quad (4.1)$$

L'ordine del modello AR corrisponde al numero di giorni inglobati nella formula. Il valore di "p" è chiamato ordine del modello AR. Si riferisce al numero di ritardi (lags) di Y da utilizzare come predittori.

Moving Average (MA):
I modelli MA presuppongono che il valore della variabile dipendente del giorno attuale dipenda dai termini di errore dei giorni precedenti. Viene matematicamente rappresentato nella formula 4.2.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q} \quad (4.2)$$

Il valore di "q" è chiamato ordine del modello MA. Si riferisce quindi al numero di errori di previsione ritardati (lags) che dovrebbero essere impostati nel modello ARIMA.

Auto-Regressive Moving Average (ARMA):
Il modello ARMA è semplicemente la combinazione dei modelli AR e MA:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q} \quad (4.3)$$

Auto-Regressive Integrated Moving Average (ARIMA):
Il modello ARIMA aggiunge una fase di differenziazione a un modello ARMA, infatti la serie temporale deve essere differenziata almeno una volta per fare in modo che diventi stazionaria (se essa non lo è già). Il valore di "d", quindi, è il numero minimo di differenziazioni necessarie per rendere stazionaria la serie. Se la serie temporale è già stazionaria, allora "d" è uguale a zero.
Si hanno quindi tre numeri interi (p, d, q) che vengono generalmente utilizzati per parametrizzare i modelli ARIMA:

- p: numero di termini auto-regressivi (ordine AR);
- d: numero di differenze non stagionali (ordine di differenziazione). Di solito "d" è maggiore o uguale di uno, ma se "d" è uguale a zero, allora ARIMA può essere considerato uguale ad un modello ARMA;
- q: numero di termini di media mobile (ordine MA).

4.2.2 Stazionarietà

Il primo passo per costruire un modello ARIMA è appunto quello di rendere la serie temporale stazionaria. Se una serie temporale è stazionaria e ha un comportamento particolare in un certo intervallo di tempo, allora è lecito ritenere che potrebbe avere lo stesso comportamento in un momento successivo. La maggior parte dei metodi di modellazione statistica richiedono che le serie temporali siano stazionarie e ARIMA è uno di questi. Esistono diversi criteri per determinare se una serie temporale è stazionaria o meno:

- Visualizzazione del grafico: questo è il metodo più semplice ed intuitivo per ottenere informazioni sulla stazionarietà della serie temporale. Dal grafico è possibile capire a vista d'occhio se la serie storica ha comportamenti stazionari o non stazionari. Questo metodo, ovviamente, non si può ritenere troppo affidabile, ma riesce comunque a dare un'idea dell'andamento;
- Rolling Statistics: viene calcolata la Rolling Mean (media mobile) e la Rolling Standard Deviation (deviazione standard mobile). La serie è considerata stazionaria se entrambe le linee create sono costanti nel tempo. Per farlo basta solamente visualizzare i valori della Rolling Mean e della Rolling Standard Deviation su grafico e, se entrambe le linee sono rette e parallele all'asse delle x (quindi la media e la varianza sono costanti), allora la serie temporale si può definire stazionaria. Nella figura 4.1 è possibile notare come, sia la Rolling Mean che la Rolling Standard Deviation non siano esattamente parallele all'asse delle x, quindi la serie in quel caso non è stazionaria;

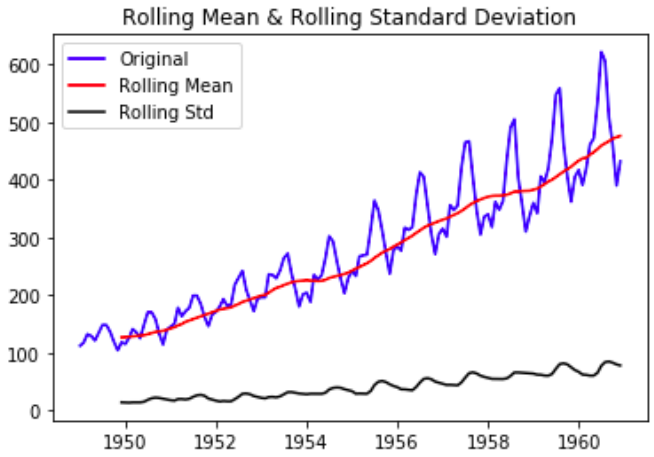

Figura 4.1: Rolling Mean e Rolling Standard Deviation

- Augmented Dickey-Fuller (ADF) Test: in questo test, la serie temporale viene considerata stazionaria se il valore risultante di p (p-value) è basso (minore di una soglia di 0,05) e i valori critici negli intervalli di confidenza dell'1%, 5% e 10% sono il più vicino possibile alle statistiche dell'ADF (ADF Statistics). Nell'esempio in figura 4.2 è possibile notare come la statistica ADF è lontana dai valori critici e il

valore di p è maggiore della soglia minima di 0,05. Quindi la serie temporale presa in considerazione non è possibile definirla stazionaria.

```
ADF Statistic: 0.8153688792060423
p-value: 0.9918802434376409
Critical Values:
        1%: -3.4816817173418295
        5%: -2.8840418343195267
        10%: -2.578770059171598
```

Figura 4.2: Augmented Dickey-Fuller Test

Possono essere utilizzate diverse tecniche per la trasformazione di una serie storica non stazionaria in una serie stazionaria:

- Differenziazione: viene utilizzata per rendere costante la media nel tempo, eliminando quindi la tendenza e la stagionalità. Questo metodo sottrae il valore attuale dal valore precedente, eseguendo quindi differenze tra le osservazioni della serie storica. Possono essere utilizzati diversi ordini di differenziazione. Ad esempio, la differenziazione del primo ordine affronta tendenze lineari e utilizza la trasformazione zi = yi - yi-1. La differenziazione del secondo ordine affronta le tendenze quadratiche eseguendo un'ulteriore differenziazione partendo dalla serie già differenziata, vale a dire zi = (yi - yi-1) - (y-1 -yi-2). La differenziazione è appunto utilizzata nei modelli ARIMA;
- Trasformazione logaritmica: questa trasformazione rende lineare un trend esponenziale e aiuta a rendere costante la varianza.

La differenziazione e la trasformazione logaritmica sono due tecniche che spesso possono anche essere utilizzate insieme.

4.2.3 ACF e PACF

Una volta verificata la stazionarietà della serie temporale, non resta che determinare i parametri "p" e "q" per costruire il modello ARIMA. Per trovare questi valori in genere si utilizzano le funzioni di autocorrelazione (Auto Correlation Function o ACF) e funzioni di autocorrelazione parziale (Partial Auto Correlation Function o PACF). A seconda del valore di questi due grafici si riesce a comprendere se utilizzare il modello AR o MA, o entrambi.

Funzione di autocorrelazione (ACF):
Un grafico di autocorrelazione, noto anche come correlogramma, mostra la correlazione della serie con se stessa ritardata di n unità di tempo. L'autocorrelazione si riferisce a

quanto è correlata una serie temporale con i suoi valori passati. Sull'asse delle Y si trovano i coefficienti di correlazione, mentre sull'asse delle X sono presenti i ritardi (lag) temporali. Nel grafico dell'ACF è anche rappresentata una banda di confidenza che, a seconda dei valori risultanti, indica se è presente correlazione all'interno della serie temporale.

Funzione di autocorrelazione parziale (PACF):
Come suggerisce il nome, PACF è un sottoinsieme di ACF. La funzione di autocorrelazione pariziale calcola la correlazione tra il valore attuale e il valore precedente, escludendo però correlazioni intermedie (se presenti). L'autocorrelazione parziale al ritardo (lag) k è la correlazione che risulta dopo aver rimosso l'effetto di eventuali correlazioni dovute ai termini di ritardi più brevi.

Ci possono essere molte regole e pratiche su come selezionare i valori appropriati di "p" e "q" utilizzando i grafici di ACF e PACF. Questi grafici possono essere utilizzati anche per verificare la stazionarietà di una serie temporale.

4.2.4 Auto ARIMA

Per un utilizzo corretto del modello ARIMA, è necessario quindi fare in modo che la serie sia stazionaria e inoltre determinare i parametri adeguati di "p", "d" e "q". Esistono diverse tecniche per raggiungere questo obiettivo (come ad esempio l'utilizzo dei grafici ACF e PACF). Tuttavia la corretta impostazione dei parametri del modello ARIMA può essere un processo lungo e che richiede competenze statistiche.
Entra quindi in gioco l'utilizzo della funzione Auto ARIMA della libreria Pmdarima [13]. Infatti essa viene utilizzata per determinare automaticamente l'ordine ottimale dei parametri del modello senza nemmeno tracciare i grafici di ACF e PACF. Viene eseguito un test di differenziazione iniziale per determinare l'ordine di differenziazione (d), adattando il modello entro certi parametri definiti. Per riuscire a trovare il modello migliore, la funzione ottimizza per un determinato criterio e restituisce il modello che riduce al minimo il valore.
Durante l'utilizzo di questa funzione è importante impostare nel modo corretto tutti i parametri necessari alla costruzione del modello. Nella tabella 4.1 sono descritti alcuni dei parametri più importanti per la costruzione e definizione di ARIMA.

Tabella 4.1: Parametri Auto ARIMA

Parametro	Significato
y	Dati di training utilizzati per addestrare il modello. Può essere una serie Pandas o un array Numpy.

Parametro	Significato
start_p	Valore (intero) di partenza del parametro “p” per l’ordine del modello auto regressivo (AR).
start_q	Valore (intero) di partenza per il parametro “q” per l’ordine del modello a media mobile (MA).
test	Tipologia di test utilizzata per rilevare la stazionarietà. Può essere utilizzato: Kwiatkowski Philips Schimdt Shin, Augmented Dickey-Fuller o Philips Perron. Di default è ‘kpss’ (Kwiatkowski Philips Schimdt Shin).
max_p	Valore massimo che può raggiungere il parametro “p”. Deve essere un valore maggiore o uguale al parametro “start_p”. Di default è uguale a 5.
max_q	Valore massimo che può raggiungere il parametro “q”. Deve essere un valore maggiore o uguale al parametro “start_q”. Di default è uguale a 5.
m	Periodo di differenziazione stagionale (se presente stagionalità). Se non è presente deve essere impostato ad 1. Di default è uguale a 1.
d	Valore dell’ordine di differenziazione. Se uguale a “None” (di default) il valore verrà ricavato automaticamente in base al tipo di test utilizzato.
seasonal	Valore booleano per impostare l’utilizzo di ARIMA stagionale. Di default è True.
trace	Valore booleano per indicare se stampare lo stato di adattamento del modello. Se impostato a False non verrà stampato alcuna informazione, se True ne verranno visualizzate alcune. Di default è False.
error_action	Viene controllato il comportamento di gestione degli errori durante la fase di adattamento. E’ possibile impostarlo a: warn, ignore, trace o raise. Di default è warn.
suppress_warning	Valore booleano per l’impostazione degli errori. Se uguale a True, tutti gli errori provenienti da ARIMA, verranno soppressi.
stepwise	Valore booleano il quale, se impostato a True, utilizza l’algoritmo graduale di Hyndman e Khandakar per ottenere i parametri ottimali del modello. L’utilizzo di questo algoritmo può incrementare ragionevolmente la velocità di adattamento di tutte le combinazione di iperparametri ed è meno probabile il verificarsi di overfitting.

4.2.5 Sommario

Passaggi importanti:

1. Verifica della stazionarietà: se una serie temporale non è stazionaria, è necessario renderla tale prima della creazione del modello ARIMA;
2. Differenziazione: se la serie temporale non è stazionaria, è necessario applicare il metodo di differenziazione. Questo passaggio sarà integrato direttamente all'utilizzo della funzione Auto ARIMA, la quale scoprirà in modo automatico l'ordine di differenziazione necessario a rendere la serie stazionaria;
3. Selezione dei termini "p" e "q": utilizzo dei grafici ACF e PACF per determinare il valore dei termini "p" e "q". Anche questo passaggio sarà integrato alla funzione Auto ARIMA, la quale scoprirà i due valori in base ad un determinato criterio;
4. Creazione del modello e previsione: una volta creato il modello è possibile effettuare previsioni impostando semplicemente il numero di periodi da prevedere;
5. Validazione: confrontare i valori reali con i valori previsti per calcolare l'accuratezza del modello creato.

Riepilogo dello sviluppo ARIMA:

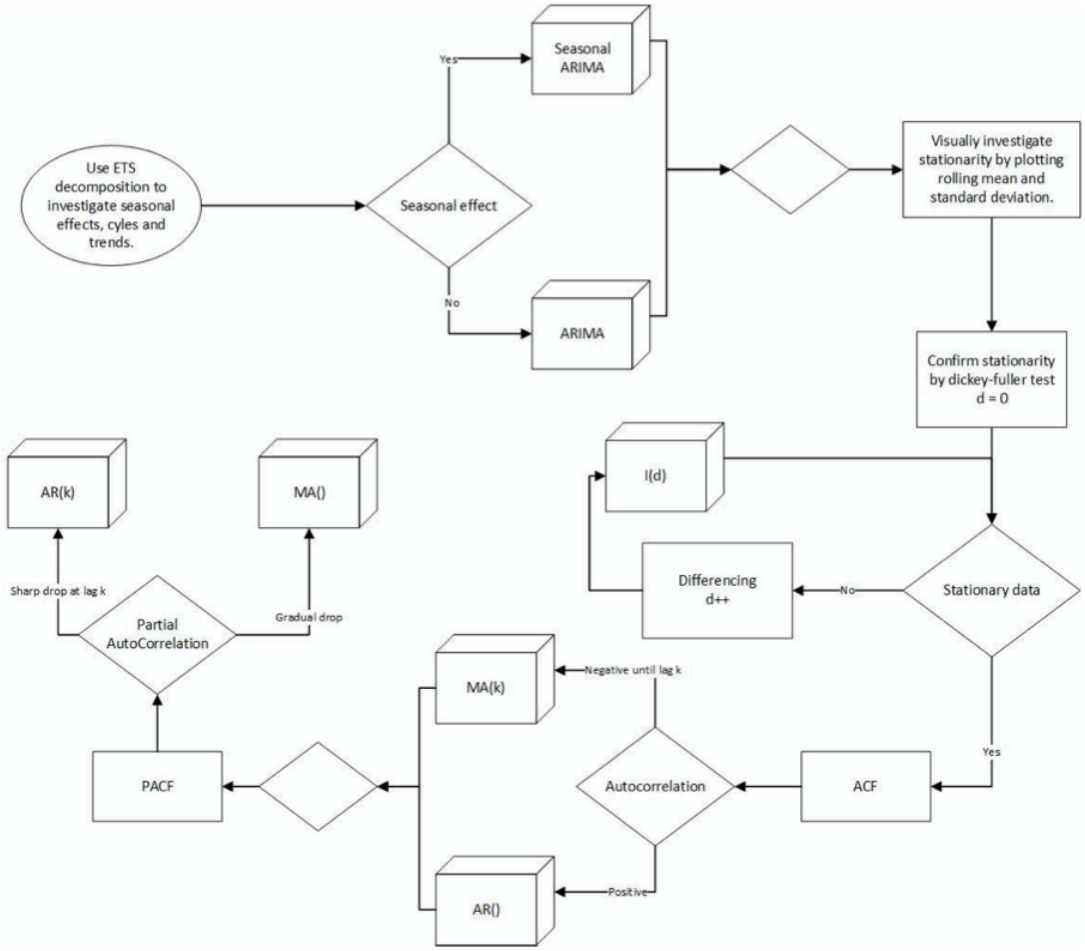

Figura 4.3: Riepilogo sviluppo ARIMA

Un'altra libreria utilizzata per la costruzione del modello ARIMA è statsmodels [14]. Essa fornisce funzioni e classi per la valutazione di diversi modelli statistici. Quindi per condurre test statistici e per l'esplorazione statistica dei dati (test dell'Augmented Dickey-Fuller, decomposizione della serie temporali, visualizzazione del grafico ACF e PACF, ecc..).

4.3 Reti neurali

Con il veloce sviluppo dell'intelligenza artificiale, l'utilizzo di reti neurali nella previsione dei prezzi di mercato è diventata una questione molto importante della ricerca [15]. A causa delle proprietà non stazionarie, non lineari e ad alto rumore delle serie temporali finanziarie, i modelli statistici tradizionali (appunto come ARIMA) potrebbero avere complicazioni nel prevedere i prezzi con accuratezza elevata. Ci sono, però, ancora alcuni problemi e difficoltà nelle previsioni finanziarie che fanno uso di reti neurali. Le persone sperano sempre di creare un modello di previsione del mercato azionario affidabile, anche se questo obiettivo non è assolutamente facile da raggiungere.

Le reti neurali hanno un comportamento che rispecchia molto quello del cervello umano, infatti sono alla base di sistemi in grado di imparare sfruttando meccanismi simili a quelli dell'intelligenza umana. Consentono ai programmi di risolvere problemi comuni e riconoscere schemi nei campi dell'intelligenza artificiale, del Machine Learning e più in particolare del Deep Learning. Le reti neurali funzionano grazie a dati di addestramento, i quali vengono utilizzati per apprendere e migliorare la loro accuratezza nel tempo. Infatti, una volta che questi algoritmi di apprendimento vengono ottimizzati, possono diventare potenti strumenti di informatica e intelligenza artificiale.

Le reti neurali possono essere utilizzate per risolvere essenzialmente tre tipi di problemi:

- Classificazione: la variabile da prevedere è di tipo categorico, cioè una classe (si hanno un insieme di immagini di cani che vengono utilizzate per poter riconoscere i cani in altre foto);
- Regressione: la variabile da prevedere è di tipo continuo, quindi un numero (si ha lo storico dei valori di un indice azionario e si vuole prevedere il valore futuro);
- Clustering: simile alla classificazione, ma non si hanno a disposizione le classi (si hanno un insieme di immagini di cani e si vuole riconoscere la razza).

4.3.1 Neurone classico

Un neurone classico esegue delle attività tutto sommato abbastanza semplici:

- Riceve uno o più ingressi;
- Moltiplica ogni ingresso per il rispettivo coefficiente (o peso);
- Somma il risultato di questi prodotti;
- Aggiunge al risultato un eventuale valore denominato "bias";
- Applica al risultato totale una funzione di attivazione;
- Utilizza il risultato di questa funzione come uscita (output).

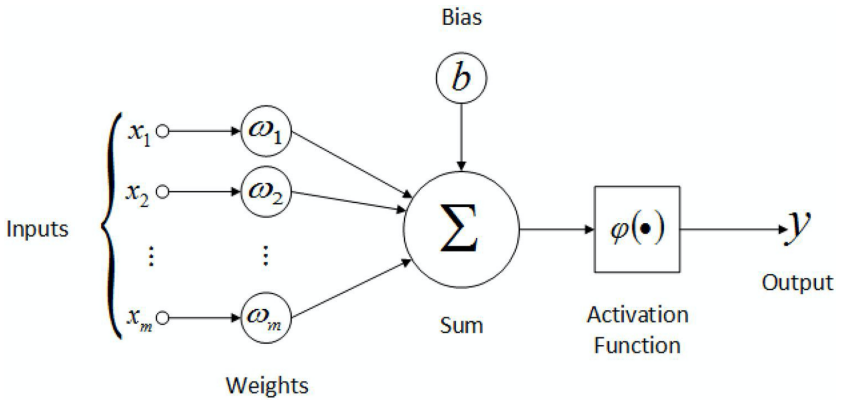

Figura 4.4: Neurone classico

La procedura di addestramento di una rete neurale serve a trovare e impostare i valori dei singoli pesi e bias relativi ad ogni neurone.

La funzione di attivazione viene utilizzata per determinare se un neurone deve essere attivato o meno. Esistono diversi tipi di funzione di attivazione, tra cui:

- Funzione di soglia (Threshold): è una funzione di attivazione basata su un valore di soglia. Se il valore di ingresso è al di sopra o al di sotto di una certa soglia, il neurone viene attivato e invia esattamente lo stesso segnale al livello successivo.
- Funzione sigmoide (Sigmoid): è una funzione più fluida rispetto alla funzione di soglia. È anche molto utile al livello di output della rete neurale. Potrebbe essere costosa dal punto di vista computazionale.
- Funzione Rectifier (ReLU): è una delle funzioni più popolari per le reti neurali. Può aiutare a combattere i problemi di Exploding e Vanishing gradient. ReLU è una funzione non lineare e molto efficiente dal punto di vista computazionale.
- Funzione tangente iperbolica (Tanh): è simile alla funzione sigmoide ma con la differenza che i valori possono avere un valore sotto lo zero. Viene utilizzato principalmente quando l'input ha valori fortemente negativi, neutri o fortemente positivi.

Alla fine del processo, quindi, applicando alla rete un ingresso conosciuto, si dovrebbe ottenere un'uscita che si avvicina il più possibile al valore reale.

4.3.2 Reti neurali artificiali

Le reti neurali artificiali (Artificial Neural Network o ANN) [16] sono costituite da strati (layers) di nodi. Esse contengono un livello di ingresso (input), uno o più livelli nascosti (hidden) e un livello di uscita (output). Ogni nodo, o neurone artificiale, si collega a un altro e ha un peso e una soglia associati. Se l'uscita di un singolo nodo è al di sopra del valore di soglia specificato (funzione attivazione), quel nodo viene attivato, inviando i dati al livello successivo della rete. In caso contrario, nessun dato viene passato al livello successivo della rete.

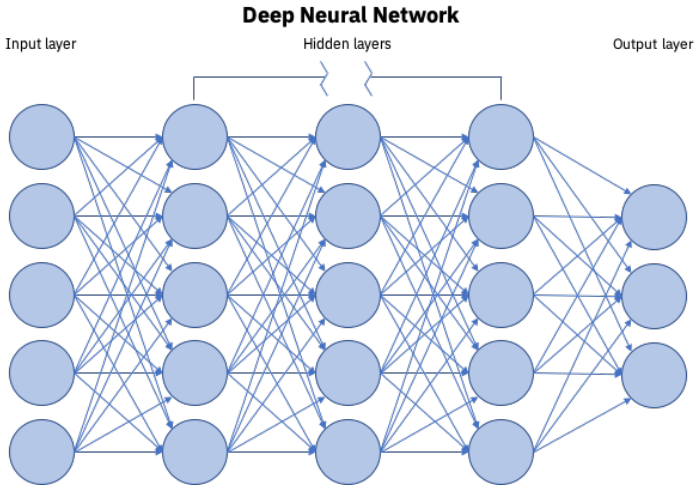

Figura 4.5: Rete neurale artificiale

Si può pensare ad ogni singolo nodo (neurone) composto da dati di input, pesi, un bias e un output. Rappresentato nelle formule 4.4 e 4.5.

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \tag{4.4}$$

$$\text{output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases} \tag{4.5}$$

Una volta impostato un livello di input, vengono assegnati i pesi, i quali aiutano a stabilire l'importanza di una variabile (i pesi più grandi contribuiscono in modo più significativo al valore di output rispetto a pesi più piccoli). Tutti gli input vengono poi moltiplicati con i relativi pesi e sommati tra loro. Infine, il risultato viene passato attraverso una funzione di attivazione che ne determina l'uscita. Se tale output supera una determinata soglia, allora viene attivato il nodo, passando quindi il risultato al livello successivo della rete. Questo significa che l'output di un nodo diventa l'input del

nodo successivo. Il processo di passaggio dei dati da un livello al livello successivo definisce la rete neurale come una rete di tipo feedforward.
La fase di apprendimento avviene quando i dati passano dal livello di input al livello di output per poi tornare indietro. Questo processo è chiamato epoca (Epoch).
Nella prima epoca, i dati vengono inseriti al livello di input e trasmessi al livello di output (forward propagation) calcolando così un valore di output. La differenza tra l'output reale e l'output previsto è chiamata funzione di costo (cost function). L'obiettivo della rete neurale è quello ridurre il più possibile questa funzione. Quindi, la rete neurale si propagherà dal livello di output fino al livello di input (backward propagation) e aggiornerà di conseguenza i pesi dei neuroni nel tentativo di ridurre al minimo questa funzione di costo. Per seguire questa operazione, si possono utilizzare svariati algoritmi di ottimizzazione, tra cui:

- Gradient Descent;
- Stochastic Gradient Descent (SGD);
- RMSprop;
- Adam.

La fase di backpropagation, viene quindi utilizzata in combinazione con un algoritmo di ottimizzazione, con il fine di regolare tutti i pesi della rete dopo aver calcolato la funziona di costo. Tutti i pesi della rete vengono aggiornati simultaneamente.
Inizialmente i pesi e i valori di bias vengono impostati in modo casuale, quindi nella prima epoca la risposta della rete sarà casuale, e quasi sicuramente sbagliata. Poi, nelle epoche successive, la rete neurale aggiornerà i pesi dei neuroni. Dopo svariate epoche e aggiornamenti dei pesi, la funzione di perdita (loss function) (la differenza tra l'output della rete neurale e l'output reale) dovrebbe raggiungere il minimo.

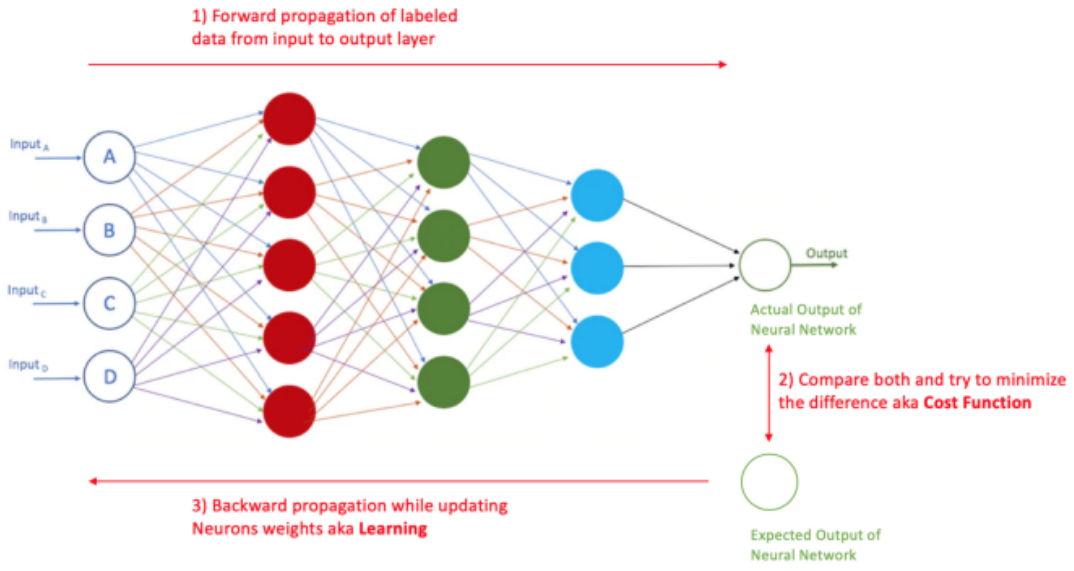

Figura 4.6: Forward e Backward propagation

Alla fine dell'addestramento, i neuroni avranno tutti pesi differenti tra loro e questi serviranno per determinare i risultati futuri, quindi sarà possibile eseguire delle previsioni.

Le reti neurali artificiali possono essere suddivise in diversi tipi, in base al proprio caso d'uso:

- Il perceptron è la più antica rete neurale, creata da Frank Rosenblatt nel 1958. E' composta da un singolo neurone ed è la configurazione più semplice di una rete neurale;
- Le reti neurali feedforward, o perceptrons multistrato (MLP), sono costituiti da un livello di input, uno o più livelli nascosti (hidden) e un livello di output. Questa tipologia di rete neurale è la base per il computer vision, l'elaborazione del linguaggio naturale e altre reti neurali;
- Le reti neurali convoluzionali (Convolutional Neural Network o CNN) sono simili alle reti feedforward, ma solitamente vengono utilizzate per il riconoscimento di immagini e il riconoscimento di pattern. Queste reti approfittano dei principi di algebra lineare, in particolare la moltiplicazione di matrici, per riconoscere i pattern all'interno di un'immagine;
- Le reti neurali ricorrenti (Recurrent Neural Network o RNN) sono identificate dai loro sistemi di feedback. Questi algoritmi di apprendimento vengono sfruttati sopratutto quando vengono utilizzati dati di serie temporali per fare previsioni sui valori futuri, come previsioni di vendita o previsioni del mercato azionario.

4.3.3 Reti neurali ricorrenti (RNN)

Le reti neurali ricorrenti [17] sono un tipo di rete potente e robusto, ritenuto uno degli algoritmi più promettenti in uso perché è l'unico con una memoria interna. Come molti altri algoritmi di Deep Learning, gli RNN sono relativamente vecchi. Infatti questa tipologia di rete è stata creata negli anni 80, ma solo recentemente si è potuto scoprire il suo vero potenziale. L'aumento della potenza di calcolo insieme alla grande quantità di dati a disposizione e l'invenzione della memoria a lungo termine (Long Short-Term Memory o LSTM) hanno portato in primo piano gli RNN.

Una rete neurale ricorrente è un tipo di rete neurale artificiale che utilizza dati sequenziali o dati di serie temporali, come ad esempio dati finanziari. Ugualmente alle reti neurali feedforward e convoluzionali (CNN), le reti neurali ricorrenti utilizzano i dati di addestramento per imparare, con la differenza della loro "memoria". Infatti, grazie alla loro memoria interna, gli RNN possono ricordare cose importanti sull'input che hanno ricevuto, consentendo di avere una maggiore precisione nel prevedere cosa succederà in futuro. Mentre le reti neurali tradizionali suppongono che gli input e gli output siano indipendenti l'uno dall'altro, l'output delle reti neurali ricorrenti dipende

dai valori precedenti all'interno della sequenza. Le reti neurali ricorrenti possono sviluppare una comprensione molto più profonda di una sequenza di dati e del relativo contesto rispetto ad altri algoritmi.

4.3.4 Funzionamento delle reti neurali ricorrenti

In una rete neurale feedforward, le informazioni si muovono in una sola direzione: dal livello di input, attraverso i livelli nascosti (hidden), al livello di output. Le informazioni che attraversano la rete non passano mai due volte in un nodo (neurone). Esse considerano solamente l'input corrente (non hanno la nozione di ordine nel tempo) e quindi non riescono a ricordare nulla di quello che è successo in passato, a parte ovviamente il loro addestramento.
In una rete neurale ricorrente, le informazioni scorrono attraverso un ciclo. Quando viene presa una decisione, viene considerato l'input corrente e anche tutto ciò che ha appreso dagli input ricevuti in precedenza.

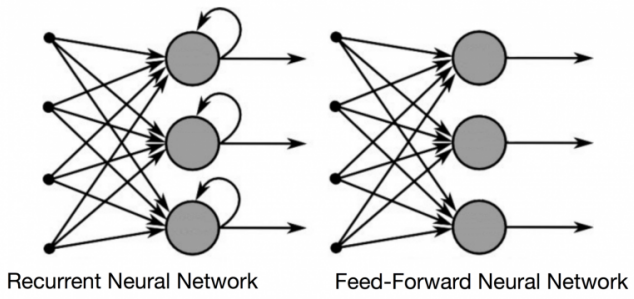

Figura 4.7: RNN vs Feedforward

E' possibile spiegare il funzionamento di una rete neurale ricorrente attraverso un semplice esempio: in una rete neurale feedforward viene data la parola "ciao" come input e si ha la necessità di elaborare questa parola carattere per carattere. Quando raggiunge il carattere "o", il sistema si è già dimenticato dei caratteri "c", "i" e "a". Questo rende impossibile per questo tipo di rete stabilire quello che verrà successivamente. Gli RNN, invece, sono in grado di ricordare quei caratteri grazie alla loro memoria interna. Esse producono un output, copiano questo output e lo reinseriscono nella rete. In sintesi, le reti neurali ricorrenti aggiungono il passato immediato al presente.
Gli RNN, quindi, hanno due input: il presente e il recente passato. Questo è un fattore molto importante perché la sequenza dei dati racchiude informazioni fondamentali su quello che verrà dopo. E' per questo motivo che le reti neurali ricorrenti hanno la possibilità di fare cose che per altri algoritmi sembrano impossibili.

Una volta appreso che una rete neurale ricorrente può essere rappresentata da una cella con un loop a stati finiti, bisogna capire come sia possibile adattarla (addestrarla). Per fare questo è necessario introdurre il concetto di "unfolding" della rete che consiste in una trasformazione di una rete di tipo RNN in una rete di tipo feedforward.

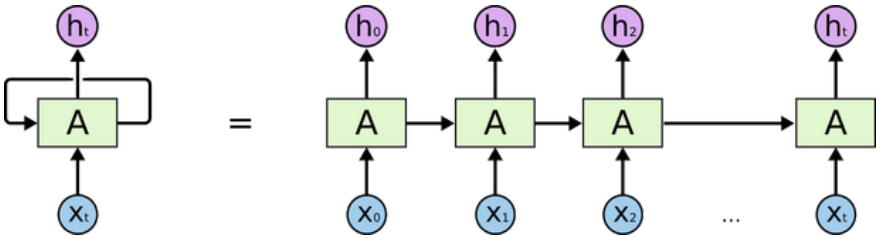

Figura 4.8: Unfolding della rete

Semplicemente, procedere con l'unfolding della rete, significa fissare a priori il numero di passi temporali (timesteps) su cui eseguire l'analisi. Per cui una rete neurale RNN "unfolded" ad esempio su 5 step, equivale ad una rete feedforward con 5 livelli.
Le reti neurali ricorrenti si servono dell'algoritmo di backpropagation through time (BPTT) per determinare i gradienti, che è leggermente diverso dalla backpropagation delle reti feedforward in quanto è specifico per i dati sequenziali (serie temporali). Il concetto del BPTT è molto simile alla backpropagation tradizionale, in cui il modello si allena calcolando gli errori partendo dal livello di output fino al livello di input. Questi calcoli permettono di adattare e regolare i parametri del modello in modo corretto. BPTT si distingue rispetto all'approccio feedforward in quanto esso somma gli errori in ogni fase temporale, mentre le reti tradizionali non hanno bisogno di sommare gli errori dato che non condividono parametri su ogni livello. All'interno del BPTT, l'errore viene trasmesso all'indietro dall'ultimo al primo timestep. Ciò consente di calcolare l'errore per ogni passo temporale, aggiornando di conseguenza i pesi. Bisogna tenere in considerazione che BPTT può essere computazionalmente costoso quando si utilizza un numero elevato di timesteps.
Infine gli RNN, a differenza delle reti tradizionali che mappano un input a un output, possono mappare uno a molti, molti a molti e molti a uno.

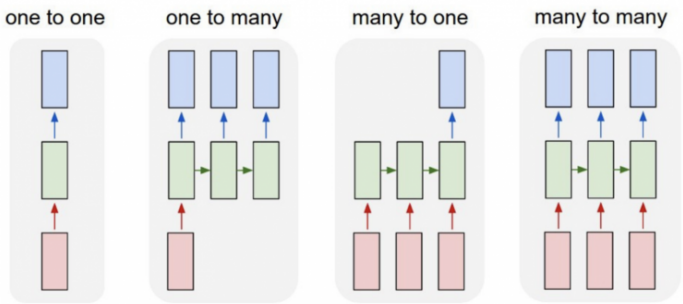

Figura 4.9: Mappatura degli RNN

4.3.5 Exploding e Vanishing gradient

Le reti neurali ricorrenti sono solite ad affrontare due principali problemi, noti come Exploding gradient e Vanishing gradient. Questi problemi sono definiti dalla dimensione del gradiente (valore utilizzato per aggiornare i pesi della rete).
Quando il gradiente è troppo piccolo si verifica il Vanishing gradient. Il suo valore si riduce sempre di più, andando ad aggiornare i valori dei pesi finché non diventano insignificanti e quando questo accade, l'algoritmo non riesce più ad imparare. L'exploding gradient, al contrario, si verifica quando il gradiente è troppo grande, creando un modello molto instabile. In questo caso, i pesi del modello diventano troppo grandi e alla fine verranno rappresentati come un valore nullo (NaN).
Per diminuire l'effetto di Exploding e Vanishing gradient, si possono utilizzare diversi metodi. Uno tra questi è l'utilizzo di neuroni ricorrenti più complessi di quelli utilizzati nelle semplici reti neurali ricorrenti. Due tipologie molto comuni sono i neuroni LSTM (Long Short-Term Memory) ed i GRU (Gatered Recurrent Unit).

4.3.6 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) [18] è una particolare architettura di rete neurale ricorrente (RNN). Funziona bene su molte tipologie di problemi ed è vastamente utilizzato al giorno d'oggi. Gli LSTM sono stati progettati principalmente per limitare i problemi di Exploding e Vanishing gradient e sopratutto per evitare il problema della dipendenza a lungo termine. Riuscire a ricordare le informazioni per lunghi periodi di tempo è praticamente il loro comportamento principale.
Gli LSTM hanno una struttura molto simile a quella di un neurone di una semplice rete neurale ricorrente, con la differenza che sono presenti quattro livelli al suo interno che interagiscono tra loro.

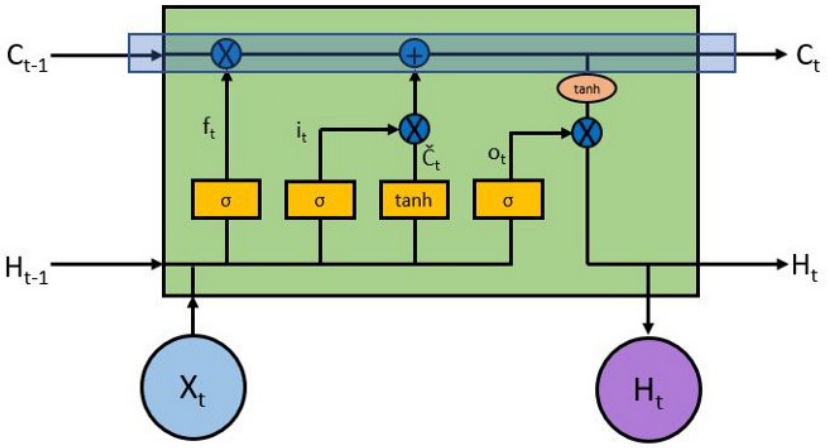

Figura 4.10: Neurone LSTM

Per riuscire a comprendere a pieno il suo funzionamento è necessario scomporlo e capire qual'è lo scopo di ogni livello e blocco. La chiave del funzionamento di LSTM è la linea orizzontale superiore (cell state) che va da sinistra a destra racchiusa nell'evidenziazione (colorata di blu) nella figura 4.10. Insieme allo stato della cella (cell state), LSTM è formato anche da tre porte (gate).
Lo stato della cella C consente alle informazioni di passare attraverso l'intero neurone LSTM, può essere interpretato come la "memoria" della rete. All'interno dello stato della cella, possono fluire informazioni rilevanti durante l'elaborazione della sequenza. Quindi, anche le informazioni provenienti dai fasi temporali precedenti possono passare a fasi temporali successive, limitando così gli effetti della memoria a breve termine. Man mano che lo stato della cella procede nel suo viaggio all'interno del neurone, incontra diversi ingressi e uscite che consentono di rimuovere o aggiungere informazioni sul suo stato. La rimozione o l'aggiunta di informazioni è controllata dalle porte (gate). Queste porte sono considerate come diverse reti neurali che decidono quali informazioni sono permesse sullo stato della cella durante la fase di addestramento.
Gli strati sigmoide (caselle gialle all'interno della cella nella figura 4.10) producono numeri compresi tra 0 e 1, impostando quindi la quantità di ciascun componente che deve essere lasciata passare. Un valore uguale a 0 significa che non passa nessuna informazione, mentre un valore uguale ad 1 significa che vengono lasciate passare tutte le informazioni. Sono presenti anche strati tanh i quali producono numeri compresi tra -1 e 1 utilizzati per regolare le rete.
Un neurone LSTM è composto da tre porte (gate) per la gestione dello stato della cella.

Forget gate:
Il primo gate preso in considerazione è il Forget gate (f(t)). Esso decide quali informazioni devono essere conservate o eliminate. Le informazioni dell'output precedente (H(t-1)) e le informazioni dell'input corrente (X(t)) vengono passate attraverso la funzione sigmoide. Se il valore è più vicino a 0 significa che le informazioni vengono dimenticate, mentre se il valore è più vicino a 1 significa che le informazione vengono conservate.

Input gate:
Per riuscire ad aggiornare lo stato della cella, viene utilizzato l'Input gate. Per prima cosa, le informazioni dell'output precedente e quelle dell'input corrente vengono passate in una funzione sigmoide. Ciò decide quali informazioni vengono aggiornate trasformando i valori in modo che siano compresi tra 0 e 1. In questo caso 0 significa "non importante" e 1 significa "importante" (i(t)). L'output precedente e l'input corrente vengono passati anche alla funzione tanh per schiacciare i valori tra -1 e 1, in modo da aiutare la regolazione della rete (Č(t)). A questo punto viene moltiplicato il

valore dell'uscita tanh con il valore dell'uscita sigmoide per riuscire ad ottenere un singolo valore.

Dopo il calcolo dei valori del Forget gate e dell'Input gate, è necessario aggiornare il vecchio stato della cella al nuovo stato della cella.

Cell state:
In questa fase si hanno abbastanza informazioni per riuscire a calcolare il nuovo stato della cella. Prima di tutto, il cell state viene moltiplicato per il valore del Forget gate. Questo passaggio ha la possibilità di far diminuire i valori nello stato della cella se esso viene moltiplicato per valori vicini allo zero. Dopodiché il valore risultante della moltiplicazione, viene aggiunto al valore del gate di input. Il risultato è il nuovo stato della cella (C(t)).

Output gate:
L'ultima porta è quella dell'Output gate. Essa ha il compito di decidere quale sarà il successivo stato di output (bisogna tenere presente che lo stato di output contiene informazioni sugli input precedenti). Per prima cosa, vengono passate le informazioni dell'output precedente e quelle dell'input corrente in una funzione sigmoide (O(t)). Allo stesso tempo viene passato lo stato della cella aggiornato alla funzione tanh. Viene quindi moltiplicato il valore dell'uscita tanh con il valore dell'uscita sigmoide per decidere quali informazioni deve contenere lo stato di output (H(t)). Il valore di output appena calcolato viene quindi trasmesso al passaggio temporale successivo o utilizzato come valore di previsione.

4.3.7 TensorFlow e Keras

Keras [19] è una libreria di Deep Learning open source scritta in Python. Questo progetto è stato avviato nel 2015 ed è diventato velocemente una delle librerie di Deep Learning più popolari tra gli sviluppatori. Keras è diventata una delle API più utilizzate grazie alla sua semplicità. Infatti consente la definizione, l'adattamento e la valutazione di modelli di Deep Learning standard in poche righe di codice. Un altro motivo per cui Keras è decollato è stato perché ha permesso l'utilizzo di alcune tra le più popolari libreria matematiche come back-end, tra cui: TensorFlow, Theano e CNTK. Questo ha permesso di sfruttare le potenzialità di queste librerie (com ad esempio la GPU) con un'interfaccia molto semplice e pulita.
Google ha recentemente pubblicato una nuova versione della sua libreria di Deep Learning (TensorFlow [20]), la quale integra direttamente l'API di Keras promuovendo questa interfaccia come predefinita per lo sviluppo di algoritmi di Deep Learning sulla piattaforma.

Dato che TensorFlow è il back-end standard per il progetto open source Keras, l'integrazione significa che è possibile utilizzare una singola libreria invece di due librerie separate. Inoltre, il progetto autonomo di Keras consiglia a tutti gli sviluppi futuri di Keras l'utilizzo dell'API basata su TensorFlow.
In questo scritto verrà appunto utilizzata la libreria di TensorFlow con Keras per riuscire a costruire il modello LSTM in un modo relativamente semplice.
Un modello è definito come la struttura principale dei dati di Keras. I due modelli principali disponibili in Keras sono:

- il modello "Sequential";
- la classe "Model" utilizzata con l'API funzionale.

Verrà preso in considerazione solamente il modello Sequential che, sostanzialmente, è una pila lineare di livelli (layers) che possono essere creati in modo molto semplice. Basta solamente inizializzare il modello Sequential e tramite la funzione "add" sarà possibile inserire al suo interno tutti i livelli desiderati.
In Keras si può trovare una vasta selezione di tipi di livelli predefiniti, è inoltre supportata anche la scrittura dei propri livelli. Quindi, a seconda di come dovrà essere costruito il modello, verranno utilizzati diversi tipi di layers. I livelli principali includono: Dense, Activation, Dropout, ecc.

Capitolo 5

Valutazione dei modelli

La valutazione di un algoritmo di Machine Learning è una parte fondamentale di qualsiasi progetto [21], sopratutto quando si ha la necessità (come in questo scritto) di fare un confronto tra più modelli. Esistono diverse metriche per valutare l'accuratezza e la precisione di un algoritmo. Esse vengono applicate all'insieme di dati di test per riuscire a calcolare gli errori tra i valori reali e i valori previsti dal modello. In questo capitolo verrà fatta un'introduzione alle quattro metriche prese in considerazione in questo progetto e alla relativa libreria utilizzata per la loro implementazione.

5.1 Mean Absolute Error (MAE)

Il Mean Absolute Error (errore medio assoluto) è la media della differenza tra i valori reali e quelli previsti. Questa metrica calcola una misura di quanto sono lontane le previsioni rispetto all'output originale. Matematicamente, è rappresentato dalla formula 5.1.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{5.1}$$

Dove, y(i) è il valore reale e ŷ(i) è il valore previsto dal modello. Il risultato di questa misura non fornisce alcuna idea della direzione dell'errore (essendo presente il valore assoluto nella formula, non esistono valori negativi). Rappresenta quindi, la media dell'errore sia nella direzione positiva, che negativa.
Il vantaggio dell'utilizzo di questa metrica è la sua facile interpretazione. Più il suo valore è basso, più il modello è performante.

5.2 Mean Squared Error (MSE)

Il Mean Squared Error (errore quadratico medio) è abbastanza simile al MAE, con l'unica differenza che MSE calcola la differenza tra il valore reale e quello previsto elevandola al quadrato, come mostrato nella formula 5.2.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{5.2}$$

Dato che viene preso in considerazione il quadrato dell'errore, la conseguenza di errori grandi diventa più evidente rispetto ad un errore più piccolo. Il risultato quindi si concentra maggiormente sugli errori più grandi e questo fa si che essi siano fortemente penalizzati. Anche in questo caso, più i valori sono bassi, più il modello è considerato preciso.
L'utilizzo di questa metrica, di solito, viene utilizzata dove gli errori grandi sono particolarmente costosi.

5.3 Root Mean Squared Error (RMSE)

Il Root Mean Squared Error (radice dell'errore quadratico medio) è definito come la radice quadrata della differenza elevata al quadrato tra il valore reale e il valore previsto. Esso viene rappresentato matematicamente nella la formula 5.3.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{5.3}$$

RMSE è considerata una buona metrica di errore per quanto riguarda le previsioni numeriche. Come succede con la metrica MSE, anche RMSE si concentra maggiormente sugli errori di grandi dimensioni. Questo implica che essa è utilizzata quando gli errori di grandi dimensioni non sono desiderati. Come anche negli altri casi, più il valore è basso, più il modello è considerato performante.

5.4 Mean Absolute Percentage Error (MAPE)

Il Mean Absolute Percentage Error (errore percentuale media assoluta) è una misura dell'accuratezza della previsione di un modello. Il MAPE viene calcolato utilizzando la formula 5.4.

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{A_i - F_i}{A_i} \right| \tag{5.4}$$

Dove, A(i) è il valore attuale, mentre F(i) è il valore di previsione del modello. Il valore risultante dovrebbe essere compreso in un intervallo che va da 0 a 1. Quindi più il valore è vicino ad 1, più la previsione è sbagliata. Mentre se il risultato è vicino allo 0, allora l'accuratezza del modello è alta. Per utilizzare una misura in percentuale, basta semplicemente moltiplicare per 100 il valore risultante dalla formula (in questo progetto il MAPE sarà rappresentato con un valore in percentuale).

5.5 Libreria per il calcolo delle metriche: Scikit-learn

Scikit-learn (o Sklearn) [22] è una libreria del linguaggio Python che comprende funzioni utili per il Machine Learning. E' probabilmente una delle librerie più utili per l'apprendimento automatico, contiene infatti molti strumenti fondamentali per eseguire operazioni come quelle di convalida incrociata, preprocessing e calcolo delle metriche. Bisogna però tenere presente che il suo scopo principale è quello di costruire modelli di Machine Learning.
In questo scritto, la libreria di scikit-learn, viene utilizzata sopratutto per il calcolo degli errori. Infatti, attraverso il suo modulo "sklearn.metrics" è possibile andare a calcolare metriche come MAE, MSE e RMSE in modo semplice ed efficace. Per quanto riguarda invece la metrica MAPE, essa non è presente all'interno della libreria. E' stato quindi necessario implementare in Python la formula 5.4 per riuscire ad utilizzare anche questa metrica (il valore del MAPE verrà rappresentato in percentuale).

Parte II
Progetto e sviluppo dei modelli

Capitolo 6

Progettazione

Il presente capitolo ha l'obiettivo di descrivere la progettazione dell'applicativo reale, andando ad analizzare e motivare le scelte progettuali. In particolare, verrà descritta inizialmente la parte di pre-elaborazione che hanno in comune i due modelli. In seguito saranno spiegati tutti i vari passaggi effettuati per la preparazione e la costruzione di ARIMA e LSTM, descrivendo nel dettaglio i motivi di alcune le scelte fatte.

6.1 Preprocessing dei dati

Il preprocessing (pre-elaborazione) dei dati nel Machine Learning è un passaggio di fondamentale importanza che aiuta a migliorare la qualità dei dati. La pre-elaborazione si riferisce alla tecnica di preparazione, organizzazione e pulizia dei dati grezzi per renderli adatti alla costruzione e all'adattamento dei modelli. Più in particolare, questo procedimento, viene definito come una tecnica di "data mining" che trasforma i dati originali in un formato leggibile e comprensibile dal modello.
La pre-elaborazione dei dati, infatti, è il primo passaggio che segna l'inizio del processo di costruzione di un modello di Machine Learning. Solitamente, i dati del mondo reale sono imprecisi, incompleti, incoerenti (contengono valori anomali o errori) e spesso presentano valori nulli in riferimento ad attributi specifici. È qui che il preprocessing dei dati entra in gioco: pulisce, formatta e organizza i dati grezzi, rendendoli così pronti all'uso per i modelli di Machine Learning.

6.1.1 Acquisizione e salvataggio del DataFrame

Il primo passo fondamentale per la creazione e lo sviluppo di modelli di Machine Learning, è l'acquisizione dell'insieme di dati pertinente (dati che dovranno essere utilizzati per il funzionamento del modello).
Per riuscire a ricavare i dati finanziari relativi ad azioni o indici, verrà utilizzata la libreria di Yahoo Finance. Essa, a seconda dell'intervallo di tempo in cui si vogliono analizzare i dati e del Ticker preso in considerazione, recupera lo storico dei valori (quindi i dati necessari alla costruzione del modello).
Come già anticipato nella sezione 2.2, gli indici che verranno presi in considerazione sono: S&P 500, Dow Jones Industrial e Nasdaq. Mentre si è deciso che l'intervallo di tempo in cui si vogliono analizzare i dati parte dall'1 Gennaio 1990 fino ad arrivare al 17 Febbraio 2021 (compreso). Per tutti e tre gli indici, verrà quindi selezionato lo stesso intervallo di tempo, così da poter utilizzare tre insiemi di dati di grandezza uguale. I tre DataFrame, conterranno quindi gli ultimi 31 anni (circa) di dati giornalieri (tenendo però presente che il sabato, la domenica e giorni festivi non sono considerati, dato che il mercato azionario in quei giorni è chiuso).
Una volta acquisiti i dati, essi verranno salvati in un file CSV per poterli caricare ed utilizzare in modo semplice e veloce.

6.1.2 Identificazione e gestione dei valori nulli (NaN)

A volte può capitare che all'interno dell'insieme di dati ricavato siano presenti valori nulli (NULL o NaN). E' quindi necessario trovare una soluzione, dato che nessun modello è in grado di gestire questi valori. Fondamentalmente si utilizzano due tecniche principali per controllare questo tipo di problema:

- Eliminazione delle righe o colonne: vengono rimosse delle righe specifiche che hanno un valore nullo, oppure viene rimossa una colonna particolare in cui manca più del 75% dei valori. Tuttavia, questa soluzione non è efficiente al 100% ed è preferibile utilizzarla solamente quando l'insieme di dati ha un numero di campioni adeguatamente elevato;
- Calcolo della media: questo metodo è utile quando vengono presi in considerazioni insiemi che contengono dati numerici, come in questo caso. E' possibile calcolare la media o la mediana di una particolare colonna o riga che contiene un valore mancante e sostituire il risultato con esso. Questa tecnica produce risultati migliori rispetto all'eliminazione di righe o colonne.

In questo elaborato, per ovviare al problema dei valori nulli o mancanti, verrà utilizzato il KNNImputer ("Nearest Neighbor Imputation" o "KNN imputation"). E' una classe della libreria "sklearn" utilizzata per calcolare o compilare i valori mancanti in un insieme di dati. E' un metodo utile che funziona sull'approccio di base dell'algoritmo

KNN piuttosto che sul semplice approccio di riempire tutti i valori con la media o la mediana. Infatti, in questo metodo, come prima cosa verrà specificata una distanza dai valori mancanti (definizione del parametro K). Infine, il valore mancante sarà semplicemente calcolato in riferimento alla media dei vicini.

6.1.3 Conclusione

Per quanto riguarda le due parti di preprocessing dei dati descritte finora, quindi l'acquisizione e salvataggio del DataFrame e l'identificazione e la gestione dei valori nulli, esse vengono utilizzate sia per quanto riguarda la costruzione del modello ARIMA, sia per la costruzione del modello LSTM. Bisogna però tenere in considerazione che la parte di pre-elaborazione non è finita, infatti verranno eseguiti altri passaggi che, a seconda del modello utilizzato, saranno leggermente differenti tra loro (come il logaritmo dei valori, normalizzazione, divisione in training set e test set, ecc..).

6.2 Modello ARIMA

I modelli ARIMA sono una classe di modelli statistici utilizzati per analizzare e prevedere i dati delle serie temporali. Come già anticipato nella sezione 4.2, ARIMA è l'acronimo di Autoregressive Integrated Moving Average ed è un'estensione di un modello ARMA più semplice. Poiché il modello ARIMA richiede che le serie temporali siano stazionarie, lo scopo della componente di integrazione (I) aggiuntiva è appunto quello di garantire la stazionarietà della serie.

6.2.1 Stazionarietà

Dopo aver eseguito i passaggi iniziali di preprocessing, quindi dell'acquisizione e salvataggio del DataFrame e dell'identificazione e gestione del valori nulli descritti nella sezione 6.1, è necessario controllare la stazionarietà della serie. ARIMA si basa sulla condizione che le serie temporali devono essere stazionarie, ovvero non devono essere presenti tendenze o stagionalità. Una serie temporale è detta stazionaria quando è presente una media costante e una varianza relativamente costante, ovvero questi parametri non variano nel tempo. Le serie temporali stazionarie sono più facili da prevedere.
Andando a selezionare i valori di "Adj_close" all'interno del DataFrame costruito precedentemente, è possibile calcolare la Rolling Mean e Standard Deviation.

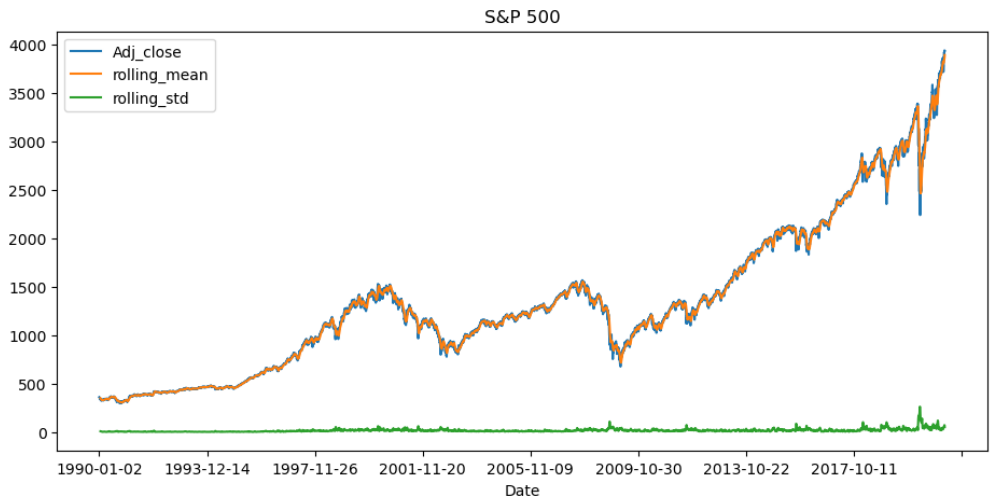

Figura 6.1: Rolling Mean e Standard Deviation S&P 500

Nella figura 6.1 è rappresentato il risultato dell'indice S&P 500, ma il comportamento degli altri due indici (Dow Jones Indusrtial e Nasdaq) è praticamente lo stesso. E' facile notare come il risultato della Rolling Mean e Standard Deviation non siano costanti nel tempo e si può già intuire che la serie non è stazionaria.

Per avere una conferma ancora più precisa della non stazionarietà delle serie temporali prese in considerazione, verrà eseguito anche il test dell'ADF: la serie temporale è considerata stazionaria se il valore di p (p-value) è basso (minore di 0,05) e i valori critici negli intervalli di confidenza dell'1%, 5% e 10% sono il più vicino possibile alle statistiche dell'ADF.

```
Results of dickey fuller test S&P 500
ADF Statistic: 2.070797261061899
p-value: 0.998758171268207
Critical Values:
        1%: -3.4311880030362194
        5%: -2.861910336031308
        10%: -2.5669671252692026
```

Figura 6.2: ADF S&P 500

Anche in questo caso, nella figura 6.2 è rappresentato il risultato dell'indice S&P 500, ma il comportamento degli altri due indici (Dow Jones Indusrtial e Nasdaq) è circa lo stesso. Si può notare prima di tutto che il valore di p (p-value) è praticamente uguale ad uno, quindi molto più grande di 0,05. Inoltre il valore della statistica dell'ADF (ADF Statistic) è molto maggiore rispetto ai valori critici (1%, 5%, 10%).
In relazione ai risultati ottenuti, si può quindi concludere che tutte e tre le serie temporali (S&P 500, Dow Jones Industrial e NASDAQ) non sono stazionarie.

6.2.2 Analisi

Per eseguire l'analisi delle serie storiche, è necessario separare la stagionalità, la tendenza e residui dalle serie temporali prese in considerazione. Come già anticipato alla sezione 1.2, la scomposizione delle serie temporali è un compito statistico che separa una serie temporale in diverse componenti.

Esistono due tipi di modelli utilizzati per scomporre le serie temporali:

- Additivo: modello lineare, tendenza lineare e stagionalità lineare con la stessa frequenza e ampiezza dei cicli nel tempo.
- Moltiplicativo: modelli non lineari, tendenza non lineare e stagionalità non lineare.

Non è sempre consigliabile lavorare con un modello moltiplicativo. Una possibile soluzione è applicare alcune trasformazioni per rendere lineare il trend e la stagionalità. Un esempio di trasformazione potrebbe essere il logaritmo di una serie in cui si osserva una crescita esponenziale.

Dalla figura 6.1, è possibile notare che esiste un modello di crescita non lineare nella Rolling Mean e che la Standard Deviation aumenta nel tempo. Si è quindi deciso di utilizzare il modello moltiplicativo.

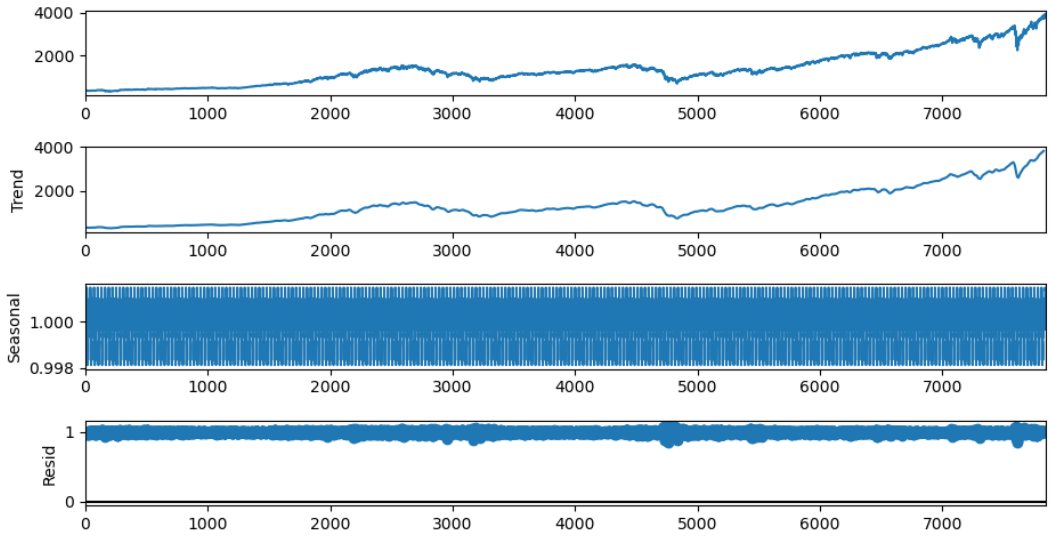

Figura 6.3: Scomposizione S&P 500

Nella figura 6.3 è rappresentato il risultato di scomposizione dell'indice S&P 500, ma il comportamento degli altri due indici (Dow Jones Indusrtial e Nasdaq) è praticamente lo stesso. Da questa decomposizione si può capire che la serie temporale non ha stagionalità, ma ha un trend che sembra rialzista (quasi esponenziale). E' stato quindi deciso di applicare il logaritmo sull'intera serie temporale prima della costruzione del modello.

6.2.3 Costruzione di Auto ARIMA

Prima della costruzione del modello e dell'impostazione dei suoi parametri, è necessario eseguire un passaggio iniziale di divisione dell'intero insieme di dati in training set e test set. Si è deciso di dividere il DataFrame in 80% per quanto riguarda il training set e 20% per il test set.

Viene utilizzato Auto ARIMA per ottenere i parametri migliori del modello senza nemmeno tracciare i grafici ACF e PACF. Prima di tutto viene passato alla funzione l'insieme di dati di addestramento (training set), sul quale si basa. L'algoritmo poi effettua inizialmente un test di differenziazione (si è deciso di utilizzate l'Augmented Dickey-Fuller "adf") riuscendo così a determinare l'ordine di differenziazione (d). Per trovare il modello migliore, Auto ARIMA ottimizza per un certo criterio (verrà utilizzato Akaike Information Criterion o "aic") e restituisce ARIMA che riduce al minimo il valore. Il criterio informativo "aic" confronta la qualità di un insieme di modelli statistici tra loro. L'aic considera ogni modello e lo classifica dal migliore al peggiore. Il modello che verrà selezionato è quello che non è né underfitting né overfitting. I parametri impostati sono praticamente quelli di default.

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-38295.696, Time=1.28 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-38314.562, Time=0.89 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-38316.474, Time=2.05 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-38293.995, Time=0.48 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-38324.577, Time=2.02 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-38323.149, Time=1.37 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-38323.305, Time=2.08 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-38325.314, Time=1.74 sec
 ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=-38323.272, Time=2.58 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=-38321.188, Time=1.86 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=-38322.703, Time=0.56 sec

Best model:  ARIMA(0,1,2)(0,0,0)[0] intercept
Total fit time: 16.926 seconds
```

Figura 6.4: Output Auto ARIMA

Nella figura 6.4 è rappresentato l'output dell'esecuzione di Auto ARIMA. Si può notare come l'algoritmo crea alcune combinazioni possibili dei parametri e, a seconda del valore "aic" di ogni combinazione, riesce a trovare il modello migliore. A questo punto, viene costruito il modello selezionando i valori ottimali di "p", "d" e "q". Durante questa fase il modello viene anche addestrato.

Una volta adattato il modello, è possibile visualizzare il grafico delle diagnostiche dei residui. I residui in un modello di serie temporale sono quello che rimane dopo l'addestramento, infatti sono uguali alla differenza tra le osservazioni e i corrispondenti valori previsti. Essi sono utili per verificare se un modello ha catturato nel modo corretto le informazioni nei dati. Un buon algoritmo di previsione genera residui che rispettano alcune caratteristiche:

- Non devono essere correlati.
- Devono avere media zero. Se i residui hanno una media diversa da zero allora le previsioni saranno distorte.
- Devono avere una varianza costante.
- Devono essere normalmente distribuiti.

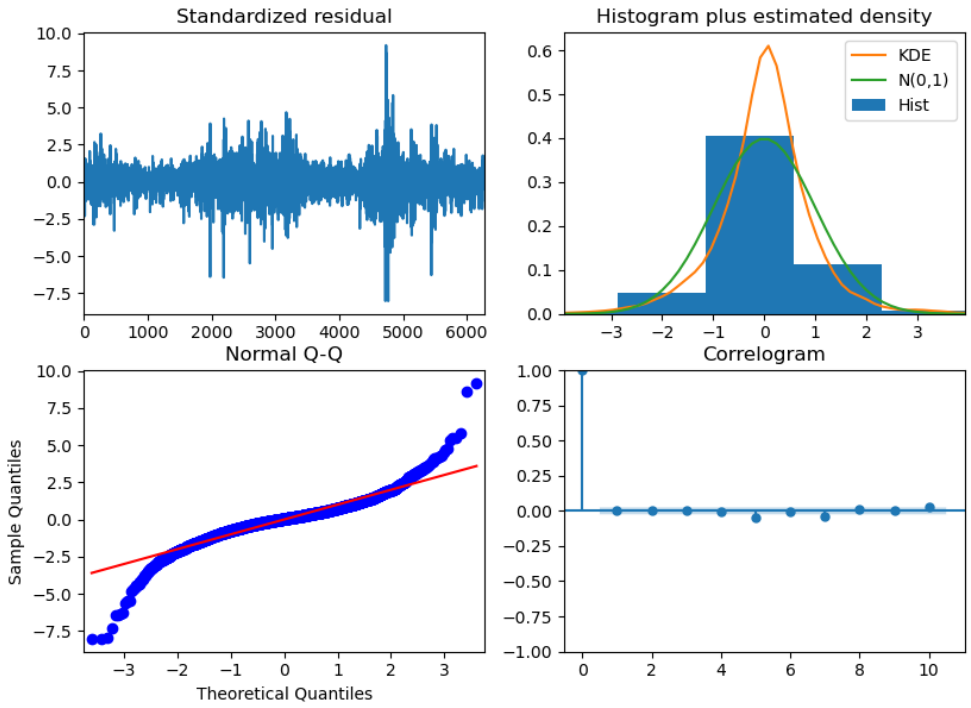

Figura 6.5: Diagnostica residui ARIMA

La figura 6.5 mostra i risultati della diagnostica dei residui riferito all'indice S&P 500 dopo aver allenato il modello ARIMA (gli altri due indici si comportano in modo uguale). E' possibile notare come:

- In alto a sinistra: i residui sembrano fluttuare intorno ad una media pari a zero.
- In alto a destra: il grafico della densità suggerisce una distribuzione normale con media zero.
- In basso a sinistra: i punti sono quasi allineati alla linea rossa, anche se alle estremità si ha una piccola deviazione (probabilmente trascurabile).
- In basso a destra: il grafico ACF mostra come i residui non sono correlati tra loro.

Si può quindi concludere dicendo che il modello sembra soddisfare le proprietà necessarie per effettuare una buona previsione.

6.3 Modello LSTM

Il modello Long Short-Term Memory (LSTM) è un tipo di rete neurale ricorrente (RNN) che ha la capacità di apprendere e ricordare su lunghe sequenze di dati di input attraverso l'utilizzo di "porte" (gate), le quali gestiscono il flusso di informazioni che passano all'interno della rete.

6.3.1 Feature engineering

Il feature engineering è definito come il processo di estrazione delle funzionalità dai dati grezzi tramite tecniche di "data mining". Questa funzionalità è utilizzata per migliorare le prestazioni degli algoritmi di Machine Learning e in questo caso viene suddivisa in quattro punti fondamentali:

1. Preprocessing dei dati: questo passaggio è in comune con il modello ARIMA, infatti si riferisce all'acquisizione e salvataggio del DataFrame e all'identificazione e gestione del valori nulli descritti nella sezione 6.1;
2. Normalizzare i dati: la normalizzazione è un'attività essenziale durante la creazione dell'architettura di una rete neurale. Quando una rete neurale viene addestrata a dati non normalizzati che hanno un intervallo di valori grandi (ad esempio di quantità comprese tra 10 e 1000), è possibile che si verifichi un rallentamento dell'apprendimento, in alcuni casi impedisce alla rete neurale di imparare. La normalizzazione viene appunto utilizzata per risolvere questo tipo di problema, andando a ridimensionare i dati dall'intervallo originale in un intervallo dove i valori sono compresi tra 0 e 1.
3. Divisione training set e test set: il passaggio successivo alla normalizzazione è quello di divisione dell'intero insieme di dati in training set e test set. Si è deciso di dividere il DataFrame in 80% per l'insieme di dati di training e 20% per l'insieme di dati di test.
4. Conversione dei dati nella forma corretta: l'input della rete neurale deve essere tridimensionale, composto da campioni (samples: dimensione del training set), fasi temporali (time steps: numero di volte in cui viene eseguita la rete neurale) e le caratteristiche (features: quantità di funzionalità per ogni time steps) inserite in quest'ordine. Prima di eseguire la conversione è necessario prima di tutto impostare la grandezza delle fasi temporali (time steps) che, in questo caso, è inizialmente impostato ad 1.

6.3.2 Ciclo di vita del modello

Prima della costruzione e implementazione del modello, è importante capire il suo ciclo di vita. Esso è diviso in cinque fasi:

1. Definizione del modello: questo passaggio richiede la selezione del tipo di modello di cui si ha bisogno, quindi scegliere l'architettura o la topologia di rete. Questo comporta la definizione dei livelli (layers) del modello, la configurazione di ogni livello con un numero di nodi, una funzione di attivazione e il collegamento dei layers. I modelli possono essere definiti con un API sequenziale o un API funzionale. In questo progetto verrà utilizzata un API sequenziale per la sua semplicità;
2. Compilazione del modello: la compilazione richiede prima di tutto la selezione di una funzione di perdita (loss function) che si desidera ottimizzare, come "mean squared error". Richiede inoltre la selezione di un algoritmo per eseguire la procedura di ottimizzazione, tipicamente la discesa del gradiente stocastico (SGD) o una variazione moderna, come Adam. A volte, può essere utile anche selezionare una qualsiasi metrica delle prestazioni (ad esempio "accuracy") per tenere traccia durante il processo di addestramento del modello;
3. Adattamento del modello: l'adattamento richiede innanzitutto la selezione della configurazione di addestramento, come il numero di epoche (numero di volte in cui l'algoritmo passerà attraverso l'intero set di dati di addestramento) e la dimensione del batch (numero di campioni di dati da utilizzare in ogni iterazione di addestramento). In questa fase viene applicato l'algoritmo di ottimizzazione scelto durante la compilazione per ridurre al minimo la funzione di perdita (loss function) scelta e aggiorna il modello utilizzando l'algoritmo di backpropagation. Il processo di adattamento (fit) del modello è considerato la parte lenta dell'intero processo e può richiedere da pochi secondi a ore o giorni (a seconda della complessità del modello). Il tempo dipende molto anche dall'hardware che si sta utilizzando e dalle dimensioni dell'insieme di dati di addestramento;
4. Valutazione del modello: la valutazione richiede di scegliere prima un insieme di dati di controllo (validation set) utilizzato per valutare il modello. Questi dati non saranno utilizzati nel processo di addestramento, così da poter ottenere una stima imparziale delle prestazioni del modello quando si effettuano previsioni su nuovi dati. La velocità della valutazione del modello è proporzionale alla quantità di dati che vengono utilizzati per la valutazione. E' molto più veloce dell'addestramento in quanto il modello non viene modificato;
5. Previsione: calcolare una previsione è il passaggio finale. È necessario avere a disposizione nuovi dati per poter riuscire a fare una previsione senza avere dei valori target (test set).

6.3.3 Costruzione e addestramento del modello

Dopo aver definito la classe Sequential per la creazione del contenitore dei layers, è possibile iniziare a definire tutti i vari livelli del modello. Viene prima di tutto creato un primo layer LSTM con 100 unità (neuroni) e una funzione di attivazione di tipo "relu". In questo primo livello, è necessario definire il valore di input da aspettarsi (input_shape). L'input deve essere tridimensionale, composto quindi da: campioni (samples), fasi temporali (time_steps) e caratteristiche (features).
Viene poi aggiunto un layer per quanto riguarda il Dropout. Il Dropout è una tecnica di regolarizzazione nella quale vengono eliminati casualmente alcuni neuroni (insieme alle loro connessioni) dalla rete neurale durante l'addestramento. Questo impedisce alle unità di adattarsi troppo e quindi di generare overfitting. Viene inizialmente impostato ad un valore di 0.0, quindi non viene utilizzato.
Infine, per rendere il modello più robusto, è stato aggiunto uno strato denso (Dense layer) alla fine del modello. Il numero di neuroni nel Dense layer è impostato a 1 poiché si vuole prevedere un singolo valore nell'output.
Dopo aver impostato il modello, esso viene compilato utilizzando un ottimizzatore di tipo "Adam" e un MSE (Mean Squared Error) per poter calcolare la funzione di perdita (loss function).

```
Model: "RNN_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               40800
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

Figura 6.6: Sommario modello LSTM

Nella figura 6.6 si può visualizzare il sommario del modello appena definito. Una volta che la rete è stata compilata, è possibile eseguire l'operazione di "fit" cioè l'operazione di adattamento del modello, il che significa adattare i pesi su un set di dati di training. La fase di addestramento della rete richiede la presenza di dati di input (x) e dati di output corrispondenti (y). La rete viene addestrata utilizzando l'algoritmo di backpropagation through time (BPTT) e ottimizzata secondo l'algoritmo di ottimizzazione e la funzione di perdita (loss function) impostati durante la fase di compilazione del modello. L'algoritmo di backpropagation richiede che la rete venga addestrata per un numero specifico di epoche. Ogni epoca può essere suddivisa in gruppi di coppie di pattern input-output denominati batch. Il batch_size è il numero di

campioni (samples) di dati da utilizzare in ogni iterazione di addestramento dopo il quale i pesi della rete vengono aggiornati. In questo caso il numero di epoche viene impostato a 100, mentre il valore di batch_size è impostato a 32.
Inoltre, in questa fase, è possibile definire anche un insieme di validazione (validation test) sul quale valutare la perdita (loss) alla fine di ogni epoca. Il valore del validation_split è impostato a 0.2, ciò significa che questa frazione di dati di addestramento non verrà utilizzata per allenare il modello. Questa metrica può essere analizzata e tracciata per capire se è presente overfitting o underfitting sui dati di addestramento.

Figura 6.7: Model train vs. validation loss

Dall'analisi della figura 6.7, pare che non ci sia la presenza di overfitting o underfitting sui dati di addestramento. Questo si può notare dalla perdita del train e dalla convalida (validation) che diminuiscono e si stabilizzano intorno allo stesso punto alla fine delle 100 epoche [23].

6.3.4 Grid Search

L'ottimizzazione degli iperparametri è una parte di fondamentale importanza nel Machine Learning. Il motivo è che le reti neurali sono molto difficili da configurare ed esistono molti parametri che devono essere impostati nel modo corretto.
Il Grid Search è una tecnica di ottimizzazione degli iperparametri di un modello. E' stato appunto utilizzato questo metodo per riuscire a configurare la rete neurale e trovare i parametri migliori da utilizzare per costruire il modello. Per riuscire ad

utilizzare il Grid Search, è necessario impostare una serie di valori che possono essere utilizzati dagli iperparametri. Viene impostato:

- batch_size: può avere un valore di 16 o 32;
- epochs: può avere un valore di 100 o 200;
- neurons: può avere valore 50 o 100. E' il numero di neuroni per il layer LSTM;
- dropout_rate: può avere valore 0.0 o 0.2.

A questo punto i valori riferiti agli iperparametri che vogliamo ottimizzare vengono passati alla funzione di Grid Search che, a seconda del miglior punteggio che trova andando a testare tutte le varie combinazioni, restituisce i parametri migliori.
Ovviamente c'è anche la possibilità di aggiungere altri possibili valori in riferimento ad altri iperparametri (come ad esempio la funzione di ottimizzazione o la funzione di attivazione), oppure semplicemente aggiungere altri valori agli iperparametri già presenti. Ma sorge un problema: aggiungendo valori e iperparametri, il costo computazionale del Grid Search diventerebbe molto elevato e il tempo impiegato per calcolare il risultato diventerebbe molto lungo (anche alcuni giorni).
Si è quindi deciso di utilizzare un numero di valori ed iperparametri non elevati durante l'utilizzo del Grid Search, anche se questo potrebbe ridurre l'efficienza del modello.

Capitolo 7

Implementazione

In questo capitolo verranno commentante alcune delle parti di codice più importanti. In particolare verrà esaminata inizialmente la parte di preprocessing in comune tra i due modelli. In seguito verranno illustrate le parti fondamentali del codice per la preparazione e la costruzione del modello ARIMA e del modello LSTM.

7.1 Operazioni di preprocessing

L'implementazione iniziale di preprocessing viene suddivisa in due parti: una prima parte per quanto riguarda la creazione dei DataFrame contenenti le informazioni dei tre indici azionari, e una seconda parte per la gestione dei valori nulli all'interno dei DataFrame.

7.1.1 Creazione del DataFrame

La libreria di Yahoo Finance viene utilizzata per riuscire a recuperare le informazioni necessarie alla costruzione dei modelli. Per farlo è necessario impostare un intervallo di tempo e il Ticker in riferimento all'indice.

```
def update_csv_ticker(name, ticker):

    start = '1990-01-01'
    today = '2021-02-18'

    df_yahoo = yf.download(ticker, start=start, end=today, progress=False)

    df_yahoo.rename(columns={'Adj Close': 'Adj_close'}, inplace=True)

    df_yahoo.to_csv("dataframes/"+name+".csv")

    plot_ticker(name, df_yahoo)
```

Listato 7.1: Creazione del DataFrame

Il listato 7.1 mostra la funzione “update_csv_ticker()” utilizzata per la creazione del DataFrame. Essa prende in ingresso il nome completo dell’indice e il relativo Ticker. Vengono poi impostate due date, le quali indicano l’intervallo di tempo in cui scaricare i dati: la data di partenza e la data di fine (bisogna tenere in considerazione che la data di fine è impostata al 18 Febbraio 2021, ma questa data non è compresa e quindi vengono scaricati di dati fino al 17 Febbraio 2021). La funzione “download()” della libreria di Yahoo Finance è molto intuitiva: è necessario fornire solamente il Ticker dell’indice e scaricherà tutti i dati in base alla date di partenza e di fine precedentemente impostate. Questa funzione ritorna un DataFrame Pandas che conterrà le seguenti colonne e i relativi dati:

- Date: già impostato come indice del DataFrame il quale rappresenta la data;
- Open: prezo di apertura;
- High: prezzo più alto raggiunto in una giornata;
- Low: prezzo più basso raggiunto in una giornata;
- Close: prezzo di chiusura giornaliero;
- Adj Close: prezzo di chiusura aggiustato. Esso viene rinominato in “Adj_close” per un migliore utilizzo;
- Volume: volume giornaliero.

Il DataFrame creato viene poi salvato in un file CSV all’interno dell’apposita cartella (dataframes/). Infine viene chiamata la funzione “plot_ticker()” che, come è possibile notare nel listato 7.2 crea il grafico utilizzando il prezzo di chiusura aggiustato (Adj_close) e lo salva in un formato HTML.

```
def plot_ticker(name, df_yahoo):

    fig = go.Figure(data=[go.Scatter(x=df_yahoo.index, y=df_yahoo['Adj_close'], fill='tozeroy')],
                    layout_title_text=name+" Historical Price",
                    )

    fig.update_xaxes(
        rangeslider_visible=True,
        rangeselector=dict(
            buttons=list([
                dict(count=1, label="1m", step="month", stepmode="backward"),
                dict(count=2, label="2m", step="month", stepmode="backward"),
                dict(count=4, label="4m", step="month", stepmode="backward"),
                dict(count=6, label="6m", step="month", stepmode="backward"),
                dict(count=1, label="1y", step="year", stepmode="backward"),
                dict(step="all")
            ])
        )
    )

    plotly.offline.plot({"data": fig}, auto_open=False, filename="plots/"+name+".html")
```

Listato 7.2: Creazione del grafico

7.1.2 Gestione dei valori mancanti

Per la gestione dei valori mancanti o nulli (NaN) viene utilizzato il KNNImputer, il quale calcola i valori nulli in relazione alla media dei i vicini.
Il metodo "KNNimputer()" contiene i seguenti argomenti:

- n_neighbors: numero di punti dati da includere più vicini al valore mancante (vicini);
- metric: è la metrica della distanza da utilizzare per la ricerca. Viene utilizzato il parametro di default che è "nan_euclidean";
- weights: utilizzata per determinare su quale base devono essere gestiti i valori vicini. Viene utilizzato il parametro di default che è "uniform";

```
def handling_null_values(df):

    if df.isnull().sum()['Adj_close'] != 0:
        imputer = KNNImputer(n_neighbors=2)
        df['Adj_close'] = imputer.fit_transform(df['Adj_close'].values.reshape(-1, 1))

    if df.isnull().sum()['Close'] != 0:
        imputer = KNNImputer(n_neighbors=2)
        df['Close'] = imputer.fit_transform(df['Close'].values.reshape(-1, 1))

    if df.isnull().sum()['Open'] != 0:
        imputer = KNNImputer(n_neighbors=2)
        df['Open'] = imputer.fit_transform(df['Open'].values.reshape(-1, 1))

    if df.isnull().sum()['High'] != 0:
        imputer = KNNImputer(n_neighbors=2)
        df['High'] = imputer.fit_transform(df['High'].values.reshape(-1, 1))

    if df.isnull().sum()['Low'] != 0:
        imputer = KNNImputer(n_neighbors=2)
        df['Low'] = imputer.fit_transform(df['Low'].values.reshape(-1, 1))

    if df.isnull().sum()['Volume'] != 0:
        imputer = KNNImputer(n_neighbors=2)
        df['Volume'] = imputer.fit_transform(df['Volume'].values.reshape(-1, 1))

    return df
```

Listato 7.3: Controllo dei valori nulli

Nel listato 7.3 si può notare come la funzione "handling_null_values()" prende in ingresso il DataFrame salvato precedentemente. Questa funzione ha il compito di controllare per ogni colonna del DataFrame (Adj_close, Close, Open, High, Low e Volume) la presenza di valori NaN. In caso affermativo viene utilizzato il KNNImputer per compilare i valori mancanti.
Dai risultati ottenuti si può confermare che non sono presenti valori nulli o mancanti all'interno del DataFrame. Questa funzione è comunque indispensabile nel caso in cui il DataFrame venga modificato o aggiornato.

7.2 Implementazione ARIMA

7.2.1 Preparazione dei dati

In questo primo passo, verranno eseguiti alcuni controlli e saranno preparati i dati per riuscire a costruire il modello ARIMA.
Come prima cosa viene controllata la stazionarietà della serie temporale. Nel listato 7.4 la funzione "check_stationarity()" prende in ingresso il DataFrame totale e il nome dell'indice. Viene estratta solamente la colonna "Adj_close" e calcolata la Rolling Mean e la Standard Deviation riportando poi i risultati su grafico. Per controllare più accuratamente se la serie è stazionaria o meno, viene eseguito anche il test dell'Augmented Dickey-Fuller (ADF) e stampato il risultato calcolato.

```
def check_stationarity(df, name):
    df_mean_std = df[['Adj_close']]

    window_size = 12
    df_mean_std['rolling_mean'] = df_mean_std.Adj_close.rolling(window=window_size).mean()
    df_mean_std['rolling_std'] = df_mean_std.Adj_close.rolling(window=window_size).std()
    df_mean_std.plot(title=name)
    plt.show()

    print("Results of dickey fuller test " + name)
    result = adfuller(df['Adj_close'])
    print('ADF Statistic: {}'.format(result[0]))
    print('p-value: {}'.format(result[1]))
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t{}: {}'.format(key, value))
```

Listato 7.4: Rolling Mean e Standard Deviation

Viene poi eseguita un'analisi della serie storica. La funzione "make_decomposition()" presente nel listato 7.5 prende in ingresso il DataFrame totale e, facendo riferimento sempre alla colonna "Adj_close", stampa su grafico la decomposizione della serie utilizzando un modello moltiplicativo.

```
def make_decomposition(df):
    df_close = df[['Adj_close']].reset_index(drop=True)
    result = seasonal_decompose(df_close, model='multiplicative', freq=30)
    result.plot()
    plt.show()
```

Listato 7.5: Decomposizione

Durante la fase di progettazione, nell'analisi della serie temporale, si è deciso di applicare il logaritmo sull'intera serie (valori di Adj_close) per cercare di rendere lineare il trend (questo passaggio è di fondamentale importanza).

```
adj_close_price_df = np.log(adj_close_price_df)
```

Listato 7.6: Logaritmo della serie temporale

Infine, è necessario effettuare la divisione del DataFrame in training set e test set. Nel listato 7.7 la funzione "split_training_testing()" prende in ingresso il DataFrame totale e restituisce due DataFrame (uno di training e l'altro di test) divisi in: 80% per l'insieme di training e 20% per l'insieme di test.

```
def split_training_testing(df):
    return df[:int(len(df) * 0.8)], df[int(len(df) * 0.8):]
```

Listato 7.7: Divisione training e testing set ARIMA

7.2.2 Creazione del modello ARIMA

Nel listato 7.8 la funzione "create_model_auto_arima()" si occupa della creazione del modello ARIMA. Essa prende in ingresso solamente il training set, il quale viene poi utilizzato dalla funzione "auto_arima()" per trovare i parametri ottimali (p, d, q) del modello.

```
def create_model_auto_arima(train_set):
    return auto_arima(train_set,
                      start_p=0,
                      start_q=0,
                      test='adf',
                      max_p=5,
                      max_q=5,
                      m=1,
                      d=None,
                      seasonal=False,
                      information_criterion='aic',
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)
```

Listato 7.8: Creazione modello ARIMA

Come spiegato nella sezione 6.2.3, la maggior parte dei parametri impostati per la costruzione del modello sono quelli di default. La cosa importante da tenere in considerazione è il parametro "seasonal" che, per impostazione predefinita, è settato a True. E' necessario quindi impostare questo parametro a False dato che non viene utilizzato un ARIMA stagionale. Un altro parametro che è stato modificato è quello del "test". E' stato impostato l'utilizzo dell'ADF per determinare il parametro di differenziazione (d), al posto del "kpss" di default. Anche altri parametri sono stati modificati (come ad esempio "error_action"), ma questi non hanno influenza sul risultato finale.
Durante questa fase, il modello viene già allenato sull'insieme di dati di training passati alla funzione.

7.2.3 Previsione a lungo termine

Una volta creato e addestrato il modello viene eseguita una prima previsione a lungo termine. Nel listato 7.9 viene inizialmente creato e addestrato il modello attraverso l'utilizzo della funzione "create_model_auto_arima()" rappresentata nel listato 7.8. Una volta ottenuto il modello, basta semplicemente chiamare il metodo "predict()" passando i parametri:

- n_periods: numero di passi che si vuole prevedere. In questo caso è uguale alla lunghezza del set di test dato che viene eseguita una previsione a lungo termine.
- return_conf_int: è una variabile booleana che consente di ottenere gli intervalli di confidenza delle previsioni. L'intervallo di confidenza è rappresentato come la probabilità in percentuale che il valore previsto sia compreso in un intervallo di valori (upper_bound e lower_bound).
- alpha: gli intervalli di confidenza per le previsioni sono (1 - alfa)%. In questo caso è impostato a 0.3 che significa quindi il 70%.

Questa funzione ritorna poi un array contenete le previsioni e gli intervalli di confidenza. Il risultato viene concatenato con con i dati reali creando così un DataFrame contenete i valori reali, quelli previsti e gli intervalli di confidenza. L'ultimo passaggio importante è quello di eseguire l'esponenziale sul DataFrame appena creato (perché inizialmente è stato applicato il logaritmo sull'intera serie temporale). Alla fine di tutti questi passaggi, il DataFrame finale viene rappresentato e salvato su grafico tramite la funzione "plot_predictions()".

```
model = create_model_auto_arima(train_set=train_data)

future_forecast = model.predict(n_periods=len(test_data), return_conf_int=True, alpha=0.3)
auto_arima_pred = [pd.DataFrame(future_forecast[0], columns=['Prediction']),
                   pd.DataFrame(future_forecast[1], columns=['lower_bound', 'upper_bound'])]
auto_arima_pred = pd.concat(auto_arima_pred, axis=1).set_index(test_data.index)
df_concat = pd.concat([adj_close_price_df, auto_arima_pred], axis=1)
df_concat = np.exp(df_concat)

plot_predictions(df_concat, 'Long_Term_' + name)
```

Listato 7.9: Previsione lungo termine ARIMA

7.2.4 Previsione one-step

Nella previsione di tipo one-step, come è possibile notare nel listato 7.10, viene innanzitutto passata la lista contenente i valori di training alla funzione "create_model_auto_arima()" la quale ritorna il modello ARIMA. Per riuscire ad effettuare questo tipo di previsione (su tutti i giorni del test set), è necessario ad ogni iterazione aggiornare i dati di training (rappresentati dalla variabile "history") e allenare nuovamente il modello. Per questo motivo viene implementato un ciclo il quale, per ogni valore presente nel test set:

- calcola la previsione del giorno successivo sul modello appena allenato;
- una volta fatta la previsione, viene aggiornata la lista "model_predictions" (lista contenete i valori previsti);
- viene ricavato il prezzo di chiusura reale del giorno sul quale è appena stata fatta la previsione;
- viene aggiunto questo valore alla lista "history", la quale viene poi utilizzata per allenare il modello nella prossima iterazione.

Alla fine del ciclo, la lista "model_predictions" conterrà tutti i valori previsti. Successivamente viene creato il DataFrame finale con i valori reali e quelli previsti, eseguito l'esponenziale sull'intero Dataset (anche in questo caso è stato eseguito inizialmente la funzione logaritmo sull'intera serie) e infine viene creato il grafico.

```
model_one_step = create_model_auto_arima(train_set=train_data)

history = [x for x in train_data['Adj_close'].values]

model_predictions = []

for time_point in list(test_data.index):
    model_one_step.fit(history)
    output = model_one_step.predict(n_periods=1)
    yhat = output[0]
    model_predictions.append(yhat)
    true_test_value = test_data.loc[time_point, 'Adj_close']
    print(time_point)
    history.append(true_test_value)

df_prediction = pd.DataFrame(model_predictions, columns=['Prediction']).set_index(test_data.index)
df_concat = pd.concat([adj_close_price_df, df_prediction], axis=1)
df_concat = np.exp(df_concat)

plot_predictions(df_concat, 'Short_Term_' + name, bounds=False)
```

Listato 7.10: Previsione one-step ARIMA

Questa tipologia di previsione viene fatta per riuscire a valutare il modello in base alla previsione di un giorno eseguita sull'intero insieme di dati di test. Considerando invece un tipo di previsione adatto all'utilizzo reale del modello, è necessario considerare l'intero insieme di dati (quindi senza dividere in training set e test set) effettuando la previsione solamente sull'ultimo giorno. Nel listato 7.11 è possibile notare come la funzione "predict_one_day_ahead()" viene divisa in due casi:

1. Variabile "test" impostata a True: in questo caso viene prima di tutto salvato in una variabile l'ultimo valore del DataFrame (questo valore sarà poi utilizzato per confrontarlo con il valore previsto). Successivamente viene creato un nuovo DataFrame senza considerare l'ultima riga (la riga eliminata contiene appunto il valore appena salvato). Viene infine creato il modello ARIMA ed eseguita la previsione del giorno successivo.
2. Variabile "test" impostata a False: in questo caso, invece, per la creazione e l'allenamento del modello viene considerato anche l'ultimo valore dell'intero DataFrame, effettuando così una previsione del prezzo di chiusura su un valore ancora sconosciuto.

```python
def predict_one_day_ahead(df, real_df, test=True):
    if test:
        log_last_close_price = df.tail(1)['Adj_close'].values
        train_data = df[:-1]
        model_1 = create_model_auto_arima(train_set=train_data)
        model_1.fit(train_data)
        output = model_1.predict(n_periods=1)[0]
        yhat = np.exp(output)
        last_close_price = np.exp(log_last_close_price)

        print(f'Il prezzo reale al giorno {real_df[-1:].index[0]} è di {last_close_price}$')
        print(f'Il prezzo previsto nello stesso giorno è: {yhat}$')

    else:
        model_2 = create_model_auto_arima(train_set=df)
        model_2.fit(df)
        output = model_2.predict(n_periods=1)[0]
        yhat = np.exp(output)

        print(f'Il prezzo al giorno {real_df[-1:].index[0]} è di {real_df[-1:].Adj_close.values}$')
        print(f'Il prezzo di previsione del giorno successivo è: {yhat}$')
```

Listato 7.11: Previsione giorno successivo ARIMA

7.3 Implementazione LSTM

7.3.1 Preparazione dei dati

In questo caso, per preparazione dei dati, si intende la tecnica feature engineering utilizzata per migliorare le prestazioni degli algoritmi di Machine Learning.

```python
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(adj_close_price_df)

train, test = adj_close_price_df[:int(len(df) * 0.8)], adj_close_price_df[int(len(df) * 0.8):]

train_data = data_scaled[0: len(train), :]
test_data = data_scaled[len(train): len(data_scaled), :]

time_steps = 1

x_train, y_train = dataset_train_creation(train_data=train_data, time_steps=time_steps)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

x_test = dataset_test_creation(total_dataset=data_scaled, time_steps=time_steps, train_data=train_data)
y_test = np.array(test_data)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

Listato 7.12: Feature Engineering LSTM

Nel listato 7.12, come primo passaggio, viene eseguita la normalizzazione dell'intero insieme di dati. Utilizzando la funzione "MinMaxScaler()" della libreria sklearn e passandogli in ingresso l'intervallo desiderato viene creata la variabile "scaler". Chiamando la funzione "fit_transform()" sulla variabile "scaler" e passando in ingresso l'insieme totale di valori di chiusura (Adj_close), essi vengono normalizzati e salvati nell'array "data_scaled". La variabile "scaler" è molto importante, poiché verrà utilizzata alla fine di tutto il processo di previsione per riuscire a riconvertire i dati normalizzati utilizzati dalla rete, nei valori originali.

Il passaggio successivo alla normalizzazione è quello di divisione dell'intero insieme di dati appena normalizzato, in training set e test set. Anche in questo caso (come in ARIMA), si è deciso di dividere il DataFrame in 80% per la parte di training e 20% per la parte di test. Le variabili create sono quindi: "train_data" e "test_data".
L'input della rete neurale deve essere tridimensionale, composto da campioni (samples), fasi temporali (time steps) e dalle caratteristiche (features). E' necessario quindi impostare il valore della variabile "time_steps", in questo caso settata ad 1.
Dopodiché un punto molto importante, è la creazione dei dati di training ("x_train" e "y_train"). E' necessario creare un set di dati di addestramento (x_train) che contenga i valori dei prezzi di chiusura (la grandezza dipende dal time steps impostato) e un altro set di dati (y_train) che contiene i valori di previsione in relazione a quelli di addestramento. Più semplicemente: la prima colonna di "x_train" contiene il valore del set di training con indice 0 (perchè time steps è 1, se fosse stato 60 allora avrebbe considerato i primi 60 valori), la seconda colonna contiene il valore del set di addestramento con indice 1, ecc. Mentre "y_train" contiene il valore del set di training con indice 1 nella prima colonna, il valore con indice 2 nella seconda colonna, ecc.
Per eseguire questa operazione, viene chiamata la funzione "dataset_train_creation()" illustrata nel listato 7.13, la quale ritorna due array NumPy.

```
def dataset_train_creation(train_data, time_steps):
    x_data = []
    y_data = []
    for i in range(time_steps, len(train_data)):
        x_data.append(train_data[i - time_steps: i, 0])
        y_data.append(train_data[i, 0])
    return np.array(x_data), np.array(y_data)
```

Listato 7.13: Creazione train set

Una volta ottenuti i due array, è necessario trasformare la variabile "x_train" in una forma tridimensionale attraverso la funzione "reshape()" per poterla utilizzare come input della rete neurale.
Gli stessi passaggi valgono anche per quanto riguarda la creazione dei dati di test, ma in questo caso è stato indispensabile creare un funzione leggermente diversa per avere un test set esattamente del 20%. Come si nota nel listato 7.14, la funzione "dataset_test_creation()" prende in considerazione gli ultimi "time_steps" valori (in questo caso un solo valore) del training set così da poter eseguire la prima previsione sul primo valore del test set.

```
def dataset_test_creation(total_dataset, time_steps, train_data):
    test_data_adjusted = total_dataset[len(train_data) - time_steps:, :]
    x_data = []
    for i in range(time_steps, len(test_data_adjusted)):
        x_data.append(test_data_adjusted[i - time_steps: i, 0])
    return np.array(x_data)
```

Listato 7.14: Creazione test set

7.3.2 Creazione e addestramento del modello LSTM

Dopo aver eseguito tutti i passaggi necessari di preparazione dei dati, è possibile costruire il modello. Le reti neurali sono definite in Keras come una sequenza di livelli (layers) e il contenitore di questi strati è definito dalla classe “Sequential()”.

```
def create_model(time_steps=1, neurons=100, activation='relu', optimizer='adam', dropout_rate=0.0):
    model = Sequential(name="RNN_model")
    model.add(LSTM(units=neurons, activation=activation, return_sequences=False, input_shape=(time_steps, 1)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1))

    model.compile(optimizer=optimizer, loss='mse')

    return model
```

Listato 7.15: Modello LSTM

Nel listato 7.15 la funzione “create_model()” si occupa della creazione del modello LSTM. Essa definisce inizialmente la classe “Sequential()” per la creazione del contenitore. I livelli creati sono:

- un layer LSTM con 100 neuroni e una funzione di attivazione di tipo “relu”. Questo layer della rete prende in ingresso anche l’input tridimensionale “input_shape” (non è necessario specificare il numero di campioni);
- un layer per per il Dropout, anche se non viene utilizzato poiché i risultati del Grid Search (sezione 6.3.4) indicano che un valore di 0.0 aumenta le performance del modello;
- infine viene aggiunto uno strato denso (Dense layer) per rendere il modello più robusto. Il numero di neuroni impostato è uguale ad 1 poiché si vuole prevedere un singolo valore di output.

Una volta definita la rete, è essenziale compilarla. La compilazione trasforma la semplice sequenza di layer definiti, in un formato destinato ad essere eseguito su GPU o CPU a seconda delle configurazioni di Keras. Durante il processo di compilazione viene indicato l’algoritmo di ottimizzazione da utilizzare per addestrare la rete (Adam) e la funzione di perdita (loss function) per la sua valutazione (MSE).

Alla fine della definizione e compilazione del modello, la rete viene essere addestrata, quindi avviene la fase di adattamento dei pesi su dati di addestramento.

```
model = create_model(time_steps=time_steps)

model.summary()

model.fit(x_train, y_train,
          epochs=100,
          batch_size=32,
          shuffle=False,
          validation_split=0.2,
          verbose=1)
```

Listato 7.16: Addestramento modello LSTM

Come si può notare nel listato 7.16, per adattare il modello è necessario definire:

- x_train: dati di input (di addestramento);
- y_train: dati di output corrispondenti ai dati di input (valori target);
- epochs: numero di volte in cui l'algoritmo passerà attraverso l'intera rete andando ad aggiornare i pesi. Vengono quindi passati tutti i dati di addestramento 100 volte all'interno della rete;
- batch_size: numero di campioni (samples) di dati utilizzati in ogni epoca dopo il quale i pesi della rete vengono aggiornati (impostato a 32);
- shuffle: valore booleano per decidere se mescolare i dati di addestramento prima di ogni epoca;
- validation_split: frazione di dati di addestramento utilizzata per valutare l'adattamento alla fine di ogni epoca;
- verbose: verbosità (0 = silenzioso, 1 = barra di avanzamento, 2 = una riga per epoca).

7.3.3 Previsione one-step

Dopo aver allenato il modello, esso può essere utilizzato per effettuare le previsioni. Per calcolare la previsione basta semplicemente chiamare la funzione "predict()" sul modello, come si può vedere nel listato 7.17. Ovviamente, la funzione ritorna i valori previsti in un formato ancora normalizzato. Risulta essenziale quindi riconvertire i dati nel loro formato originale e, per farlo, viene chiamata la funzione "inverse_tranform()" sulla variabile "scaler" definita inizialmente (nella sezione 7.3.1). A questo punto, la variabile "test_preds" contiene i valori di previsione dell'interno test set. Viene infine definito un DataFrame il quale racchiude i valori reali e i valori previsti, riuscendo così a creare e salvare il grafico attraverso la funzione "plot_predictions()".

```
scaled_preds = model.predict(x_test)
test_preds = scaler.inverse_transform(scaled_preds)

prediction = pd.DataFrame(test_preds, columns=['Prediction']).set_index(test.index)
df_concat = pd.concat([adj_close_price_df, prediction], axis=1)

plot_predictions(df_concat, 'RNN_Short_Term_' + name)
```

Listato 7.17: Previsione one-step LSTM

La tipologia di previsione rappresentata nel listato 7.17, viene utilizzata per valutare il modello sull'intero insieme di dati di test. Andando a considerare invece, un tipo di previsione adatto all'utilizzo reale del modello, esso verrà adattato (addestrato) sull'intero DataFrame ed eseguite due tipologie di previsione:

1. Previsione del giorno successivo eliminando dal DataFrame solamente l'ultimo dato, il quale verrà poi comparato con il valore previsto dal modello;
2. Previsione del giorno successivo considerando tutto il DataFrame, prevedendo quindi un valore ancora sconosciuto.

```python
def predict_one_day_ahead(total_df, scaler, time_steps=1, test=True):
    last_close_price = 0
    if test:
        last_close_price = total_df[-1:]['Adj_close'].values
        df = total_df[:-1]
    else:
        df = total_df
    train_data = scaler.fit_transform(df[['Adj_close']])
    x_train, y_train = dataset_train_creation(train_data=train_data, time_steps=time_steps)
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

    last_time_step_close_price = df[-time_steps:][['Adj_close']]
    last_time_steps_days_scaled = scaler.fit_transform(last_time_step_close_price)
    x_test = [last_time_steps_days_scaled]
    x_test = np.array(x_test)
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    model = create_model(time_steps=time_steps)
    model.fit(x_train, y_train,
              epochs=100,
              batch_size=32,
              shuffle=False,
              verbose=1)
    output = model.predict(x_test)
    yhat = scaler.inverse_transform(output)

    if test:
        print(f'Il prezzo reale al giorno {total_df[-1:].index[0]} è di {last_close_price}$')
        print(f'Il prezzo previsto nello stesso giorno è: {yhat[0, 0]}$')
    else:
        print(f'Il prezzo al giorno {total_df[-1:].index[0]} è di {total_df[-1:].values[0, 0]}$')
        print(f'Il prezzo di previsione del giorno successivo è: {yhat[0, 0]}$')
```

Listato 7.18: Previsione giorno successivo LSTM

La funzione “predict_one_day_ahead()” rappresentata nel listato 7.18 esegue le due tipologie di previsione. Viene preso in ingresso l’intero DataFrame, la variabile “scaler” (per la normalizzazione), il “time_steps” e una variabile booleana “test" per decidere che tipo di previsione effettuare.

Nel primo caso (variabile “test” uguale a True) viene salvato inizialmente l’ultimo valore del DataFrame (valore che bisogna prevedere) e creato un nuovo DataFrame eliminando il valore appena salvato. Dopodiché vengono eseguiti tutti i passaggi per la preparazione dei dati di addestramento, viene creato e allenato il modello e infine eseguita la previsione.

Per quanto riguarda invece il secondo tipo di previsione (variabile “test” uguale a False), viene considerato l’intero DataFrame per l’allenamento del modello. Il risultato di questa previsione non potrà essere confrontando poiché il valore reale è ancora sconosciuto.

Capitolo 8

Risultati raggiunti

Nel presente capitolo verranno interpretati i risultati di previsione raggiunti per ognuno dei tre indici considerati. Per prima cosa verrà fatta un'introduzione alle tipologie di previsione effettuate. Di seguito verranno analizzate le previsioni utilizzando il modello ARIMA e il modello LSTM. Infine verranno messi a confronto i risultati ottenuti dei due modelli.

8.1 Introduzione

Gli algoritmi di Machine Learning sono noti per essere molto efficaci nei problemi di previsione. Vengono utilizzati due modelli (ARIMA e LSTM) per prevedere il prezzo delle azioni di S&P 500 (ticker: ^GSPC), Dow Jones Industrial (ticker: ^DJI) e NASDAQ (ticker: ^IXIC). In questo caso, viene utilizzato solamente il prezzo di chiusura aggiustato e un intervallo di tempo che parte dall'1 Gennaio 1990 e arriva al 17 Febbraio 2021 (diviso poi in 80% training e 20% test), per eseguire le previsioni.
Nel mercato azionario sono presenti due tipi di investimenti principali: investimenti a lungo termine e investimenti a breve termine. Negli investimenti a lungo termine, gli investitori, acquistano un certo numero di azioni e dopo un lungo periodo di tempo (può essere mesi o anni), potrebbero avere dei rendimenti. Questo significa che il valore delle azioni acquistate è aumentato. Mentre per quanto riguarda gli investimenti a breve termine, gli investitori, anche in questo caso acquistano un certo numero di azioni, ma il periodo di trading è molto corto, di solito un giorno (day trading). In ARIMA sono state eseguite due tipologie di previsione: a lungo termine e a breve termine (previsione one-step e previsione del giorno successivo). Mentre per quanto riguarda LSTM è stata eseguita solamente una previsione a breve termine (previsione on-step e previsione del giorno successivo), questo per un motivo di progettazione della rete neurale. Lo scopo

principale del progetto, infatti, è quello di comparare le previsioni a breve termine dei due modelli.
Per riuscire a valutare l'accuratezza di previsione, verranno utilizzate quattro metriche (introdotte al capitolo 5): Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) e Mean Absolute Percentage Error (MAPE).

8.2 Risultati ARIMA

8.2.1 Previsione a lungo termine

Questo tipo di previsione è stata fatta solamente per avere una visione in termini di investimenti a lungo termine, infatti essa non verrà confrontata con LSTM. E' una previsione molto difficile poiché, considerando un tempo lungo, esistono moltissimi fattori che potrebbero influenzare il prezzo di un'azione.

S&P 500:

Figura 8.1: Previsione S&P 500 lungo termine ARIMA

Tabella 8.1: Metriche S&P 500 lungo termine ARIMA

Indice	MAE	MSE	RMSE	MAPE (%)
S&P 500	158,568	46164,881	214,86	5,896%

Dow Jones Industrial:

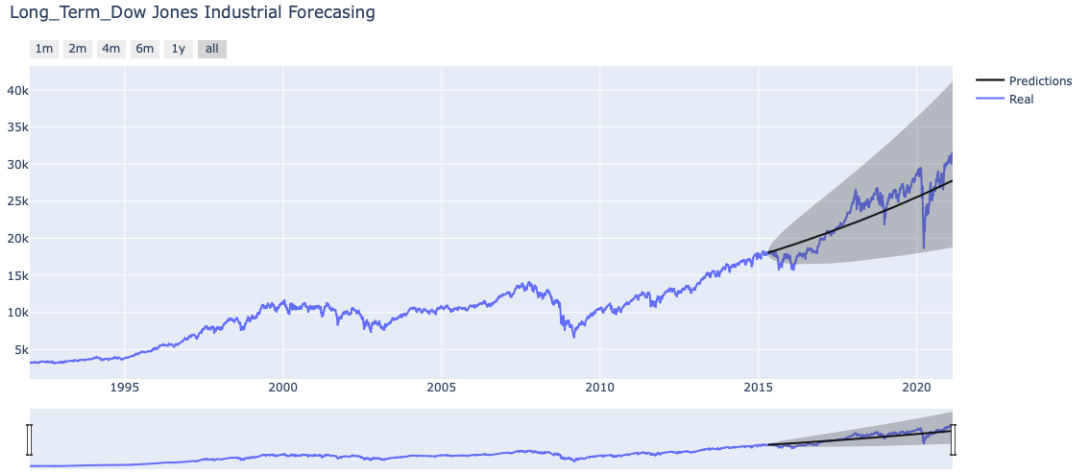

Figura 8.2: Previsione Dow Jones Industrial lungo termine ARIMA

Tabella 8.2: Metriche Dow Jones Industrial lungo termine ARIMA

Indice	MAE	MSE	RMSE	MAPE (%)
Dow Jones Industrial	1639,263	3756827,17	1938,254	7,16%

NASDAQ:

Figura 8.3: Previsione NASDAQ lungo termine ARIMA

Tabella 8.3: Metriche NASDAQ lungo termine ARIMA

Indice	MAE	MSE	RMSE	MAPE (%)
NASDAQ	871,415	1902680,04	1379,377	10,202%

Nelle figure 8.1, 8.2 e 8.3 si può da subito notare come la previsione rispetto ai loro andamenti reali, non sia molto precisa. Riesce comunque a dare un'idea di quale potrebbe essere il prezzo dopo alcuni anni. Per quanto riguarda invece le metriche rappresentate nelle tabelle 8.1, 8.2 e 8.3, considerando che, a seconda dell'indice, si hanno degli intervalli di valori molto grandi, gli errori sono relativamente bassi (considerando che viene eseguita una previsione di circa 5 anni).

8.2.2 Previsione one-step

La previsione di tipo one-step è una previsione utilizzata per riuscire a valutare l'accuratezza del modello. In ARIMA viene eseguita questa previsione allenando il modello sul training set iniziale, viene calcolata la previsione di un giorno e aggiunto il valore reale del giorno appena previsto al training set. Questa operazione si ripete per ogni valore del test set, calcolando così la previsione di un giorno sull'intero insieme di test.

S&P 500:

Figura 8.4: Previsione S&P 500 one-step ARIMA

Tabella 8.4: Metriche S&P 500 one-step ARIMA

Indice	MAE	MSE	RMSE	MAPE (%)
S&P 500	18,401	953,987	30,887	0,705%

Dow Jones Industrial:

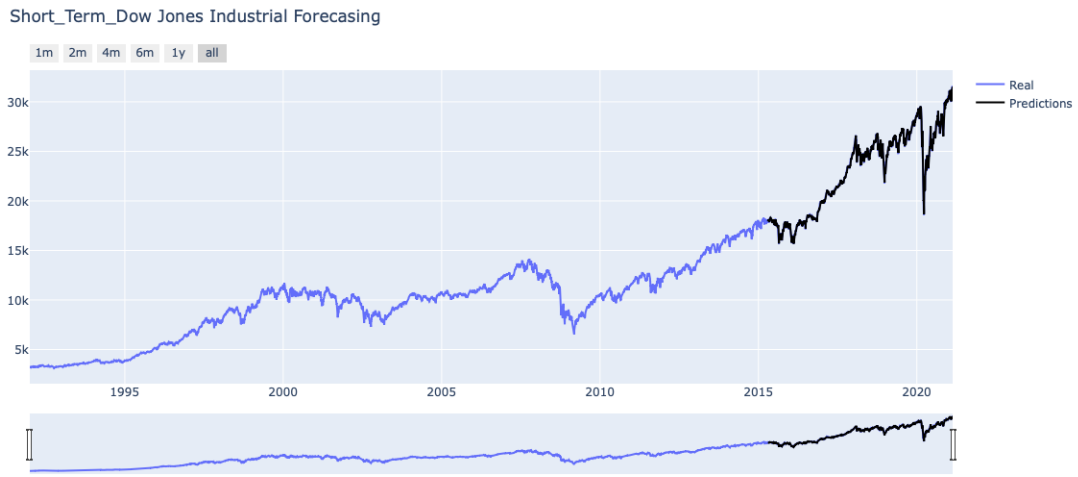

Figura 8.5: Previsione Dow Jones Industrial one-step ARIMA

Tabella 8.5: Metriche Dow Jones Industrial one-step ARIMA

Indice	MAE	MSE	RMSE	MAPE (%)
Dow Jones Industrial	166,854	80721,956	284,116	0,725%

NASDAQ:

Figura 8.6: Previsione NASDAQ one-step ARIMA

Tabella 8.6: Metriche NASDAQ one-step ARIMA

Indice	MAE	MSE	RMSE	MAPE (%)
NASDAQ	61,555	10172,749	100,86	0,845%

Dalla visualizzazione delle figure 8.4, 8.5 e 8.6, sembra quasi che le previsioni siano esattamente uguali ai valori reali, ma non è assolutamente così. Infatti ingrandendo i grafici, si possono notare tutti gli errori di previsione. Ovviamente, rispetto alla previsione a lungo termine, gli errori sono molto più piccoli dato che la previsione è di un giorno (effettuata sull'intero insieme di test). I risultati delle metriche nelle tabelle 8.4, 8.4 e 8.6, sembrano fornire una buona accuratezza di previsione considerando i grandi valori degli intervalli dei tre indici. Il modello potrebbe calcolare previsioni molto precise in determinati giorni, come potrebbe calcolare previsioni poco precise in altri giorni.

8.2.3 Previsione giorno successivo

La previsione del giorno successivo è sostanzialmente uguale alla previsione one-step con la differenza che viene considerato l'intero DataFrame come training set ed eseguita solamente una previsione (quella del giorno seguente all'ultimo valore nel DataFrame). Sono state prese in considerazioni due tipi di previsione del giorno successivo:

1. Nel primo caso viene eliminato l'ultimo giorno dal DataFrame totale (17 Febbraio 2021), eseguita la previsione su quel giorno e confrontata col valore eliminato. Questo per avere un'idea dell'accuratezza della previsione. E' solo un confronto tra il valore di chiusura previsto e quello reale;
2. Nel secondo caso viene considerato l'intero DataFrame (17 Febbraio compreso) ed eseguita la previsione sul valore successivo all'ultimo dato dell'intero set, quindi una previsione su un valore ancora sconosciuto (18 Febbraio 2021).

Queste due tipologie di previsione non vengono effettuate per valutare l'accuratezza, è solo un modo per capire quale sarà il reale utilizzo dei modelli che sono stati costruiti.

S&P 500:

Tabella 8.7: Previsione giorno successivo (1) S&P 500 ARIMA

	Prezzo reale	Previsione
17 Febbraio 2021	3931,33$	3933,93$

Tabella 8.8: Previsione giorno successivo (2) S&P 500 ARIMA

	Previsione
18 Febbraio 2021	3932,73$

Dow Jones Industrial:

Tabella 8.9: Previsione giorno successivo (1) Dow Jones Industrial ARIMA

	Prezzo reale	Previsione
17 Febbraio 2021	31613,01$	31527,62$

Tabella 8.10: Previsione giorno successivo (2) Dow Jones Industrial ARIMA

	Previsione
18 Febbraio 2021	31615,58$

NASDAQ:

Tabella 8.11: Previsione giorno successivo (1) NASDAQ ARIMA

	Prezzo reale	Previsione
17 Febbraio 2021	13965,49$	14053,93$

Tabella 8.12: Previsione giorno successivo (2) NASDAQ ARIMA

	Previsione
18 Febbraio 2021	13975,25$

Nelle tabelle 8.7, 8.9 e 8.11 i valori previsti dal modello, sono molto vicini ai valori reali, a volte la differenza può essere anche minima. Ovviamente non è sempre così, ci possono essere giorni in cui la previsione risulta essere molto distante dal valore reale e questo dipende sopratutto da fattori esterni. Per quanto riguarda invece i valori dichiarati nelle tabelle 8.8, 8.10 e 8.12, le previsioni non possono essere messe a confronto con nessun dato poiché sono valori ancora sconosciuti.

8.3 Risultati LSTM

8.3.1 Previsione one-step

La previsione di tipo one-step viene chiamata in questo modo per riuscire a confrontarla con il modello ARIMA ma, nel caso di LSTM, questa è la previsione vera e propria. E' possibile paragonarla con ARIMA, poiché anche in questo caso è come se venisse eseguita una previsione one-step sull'intero test set (vengono presi in considerazione gli stessi insiemi di training e test utilizzati anche precedentemente).

S&P 500:

Figura 8.7: Previsione S&P 500 one-step LSTM

Tabella 8.13: Metriche S&P 500 one-step LSTM

Indice	MAE	MSE	RMSE	MAPE (%)
S&P 500	19,13	982,374	31,343	0,73%

Dow Jones Industrial:

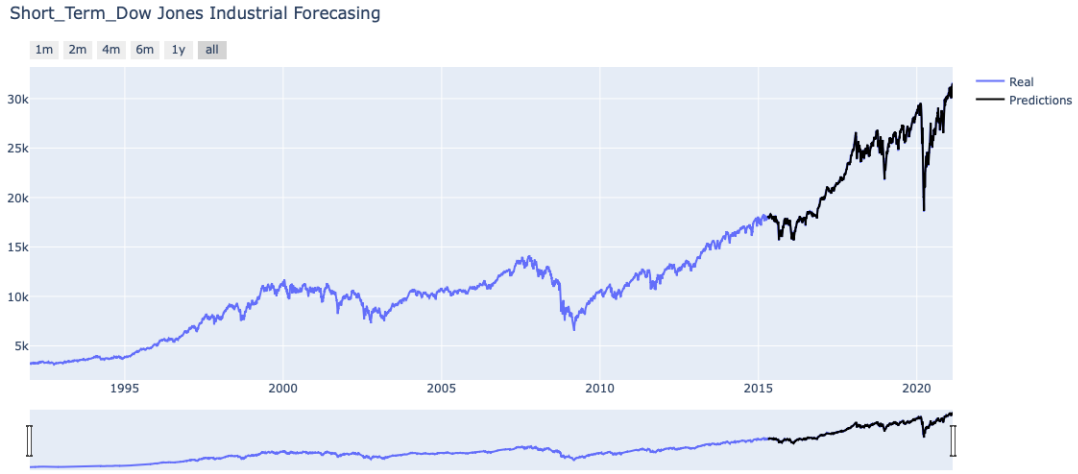

Figura 8.8: Previsione Dow Jones Industrial one-step LSTM

Tabella 8.14: Metriche Dow Jones Industrial one-step LSTM

Indice	MAE	MSE	RMSE	MAPE (%)
Dow Jones Industrial	169,499	83435,713	288,852	0,735%

NASDAQ:

Figura 8.9: Previsione NASDAQ one-step LSTM

Tabella 8.15: Metriche NASDAQ one-step LSTM

Indice	MAE	MSE	RMSE	MAPE (%)
NASDAQ	78,977	15078,832	122,796	1,064%

Dalle visualizzazione dei grafici 8.7, 8.8 e 8.9, può sembrare che la previsione sia esattamente uguale al valore reale, ma non è assolutamente così. Infatti, andando ad ingrandire i grafici, si può constatare che le previsioni non sono esattamente uguali agli andamenti reali, ma sono presenti errori. Per quanto riguarda invece i risultati delle tabelle 8.13, 8.14 e 8.15, gli errori di previsione non sembrano eccessivamente alti, considerando sempre la presenza di grandi valori degli intervalli dei tre indici. Anche in questo caso il modello potrebbe fare buone previsioni in alcuni giorni, come potrebbe fare previsioni errate in altri giorni.

8.3.2 Previsione giorno successivo

La previsione del giorno successivo viene eseguita in modo esattamente uguale a quella mostrata nella sezione 8.2.3 con ARIMA. Quindi, anche in questo caso, vengono calcolate due tipologie di previsione del giorno successivo.

S&P 500:

Tabella 8.16: Previsione giorno successivo (1) S&P 500 LSTM

	Prezzo reale	Previsione
17 Febbraio 2021	3931,33$	3925,70$

Tabella 8.17: Previsione giorno successivo (2) S&P 500 LSTM

	Previsione
18 Febbraio 2021	3930,00$

Dow Jones Industrial:

Tabella 8.18: Previsione giorno successivo (1) Dow Jones Industrial LSTM

	Prezzo reale	Previsione
17 Febbraio 2021	31613,01$	31530,01$

Tabella 8.19: Previsione giorno successivo (2) Dow Jones Industrial LSTM

	Previsione
18 Febbraio 2021	31616,11$

NASDAQ:

Tabella 8.20: Previsione giorno successivo (1) NASDAQ LSTM

	Prezzo reale	Previsione
17 Febbraio 2021	13965,49$	14049,99$

Tabella 8.21: Previsione giorno successivo (2) NASDAQ LSTM

	Previsione
18 Febbraio 2021	13965,44$

I valori previsti nelle tabelle 8.16, 8.18, 8.20, sembrano andare abbastanza vicino ai loro valori reali. Ovviamente può non essere sempre così, a volte l'accuratezza di previsione risulta essere migliore e in altri casi peggiore. Mentre per quanto riguarda i valori di previsione nelle tabelle 8.17, 8.19 e 8.21 non possono essere confrontati con nessun dato, visto che i valori reali sono ancora sconosciuti.

8.4 Confronto

Per riuscire ad effettuare un confronto tra i modelli ARIMA e LSTM, bisogna tenere in considerazione alcuni fattori:

- Utilizzare gli stessi dati: infatti, per tutti e tre gli indici, sono stati utilizzati i dati che vanno dall'1 Gennaio 1990 al 17 Febbraio 2021 e divisi nella stessa frazione per l'insieme di training (80%) e di test (20%), sia in ARIMA che in LSTM;
- Eseguire lo stesso tipo di previsione: verrà confrontata la previsione di tipo one-step effettuata sui due modelli. Inoltre verrà fatta qualche considerazione sulla previsione del giorno successivo, anche se questa non viene utilizzata per la loro valutazione;
- Calcolare le stesse metriche di errore per ogni tipo di previsione: vengono utilizzate le metriche MAE, MSE, RMSE e MAPE.

8.4.1 Confronto previsione one-step

S&P 500:

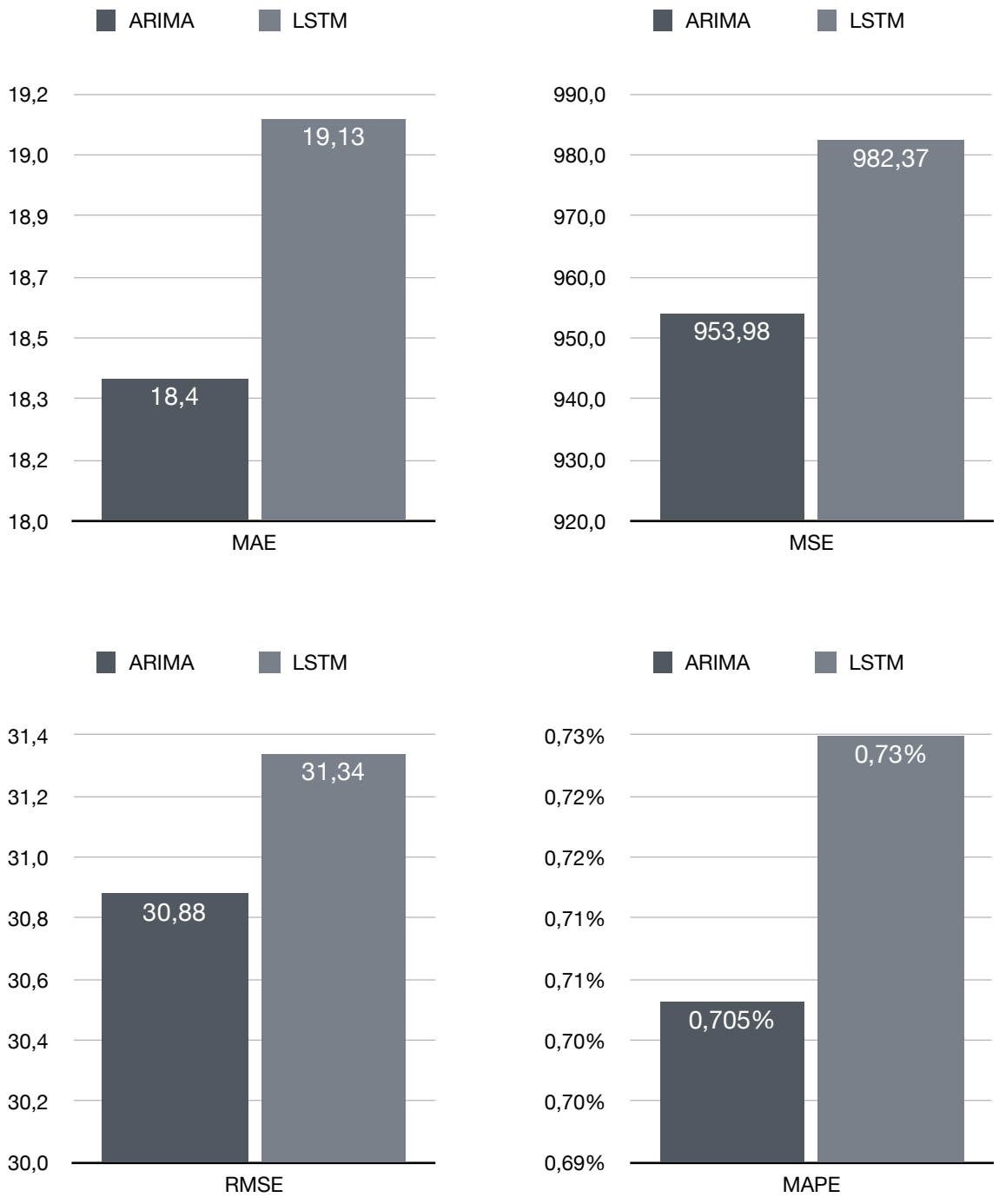

Figura 8.10: Confronto previsione one-step S&P 500

Dow Jones Industrial:

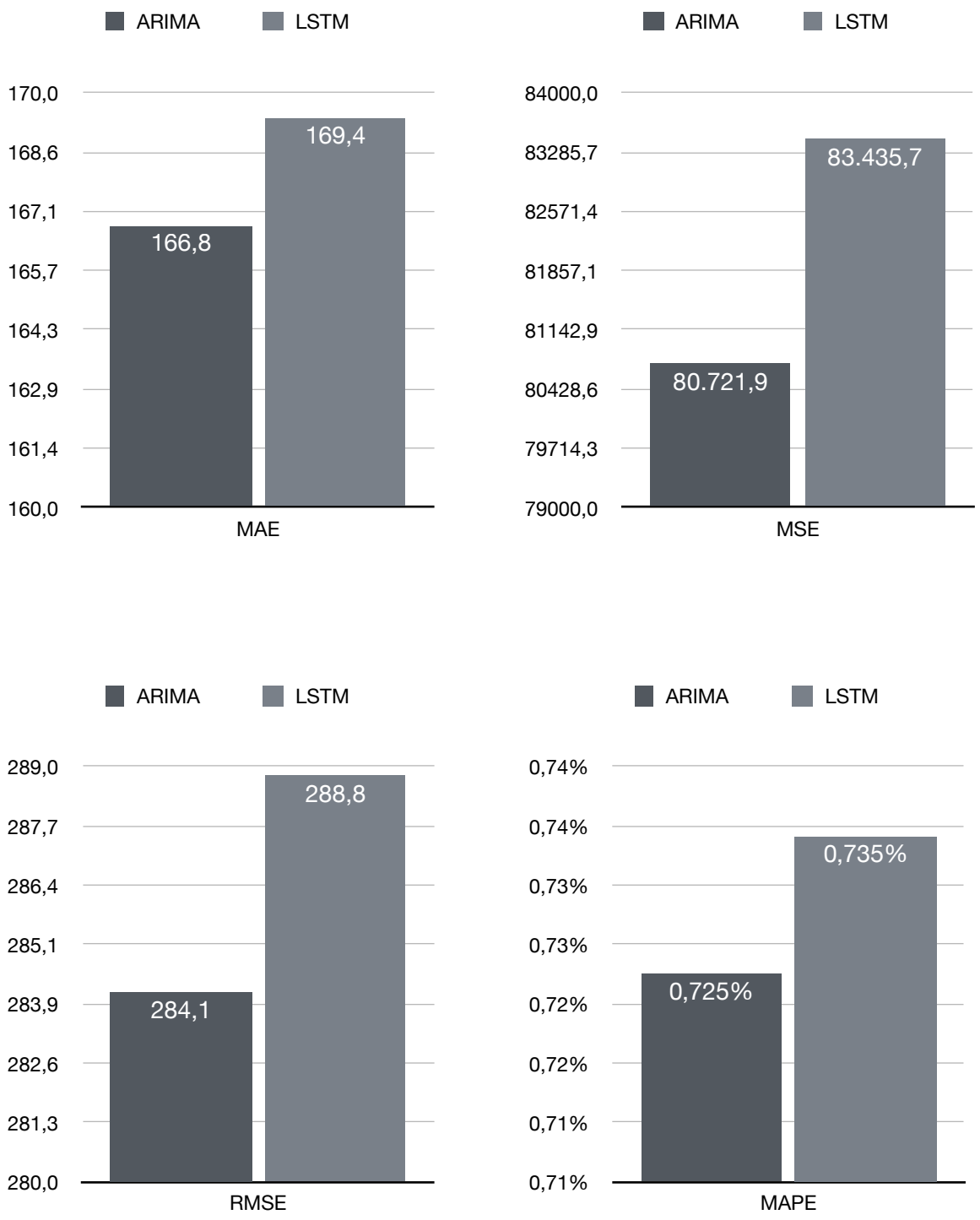

Figura 8.11: Confronto previsione one-step Dow Jones Industrial

NASDAQ:

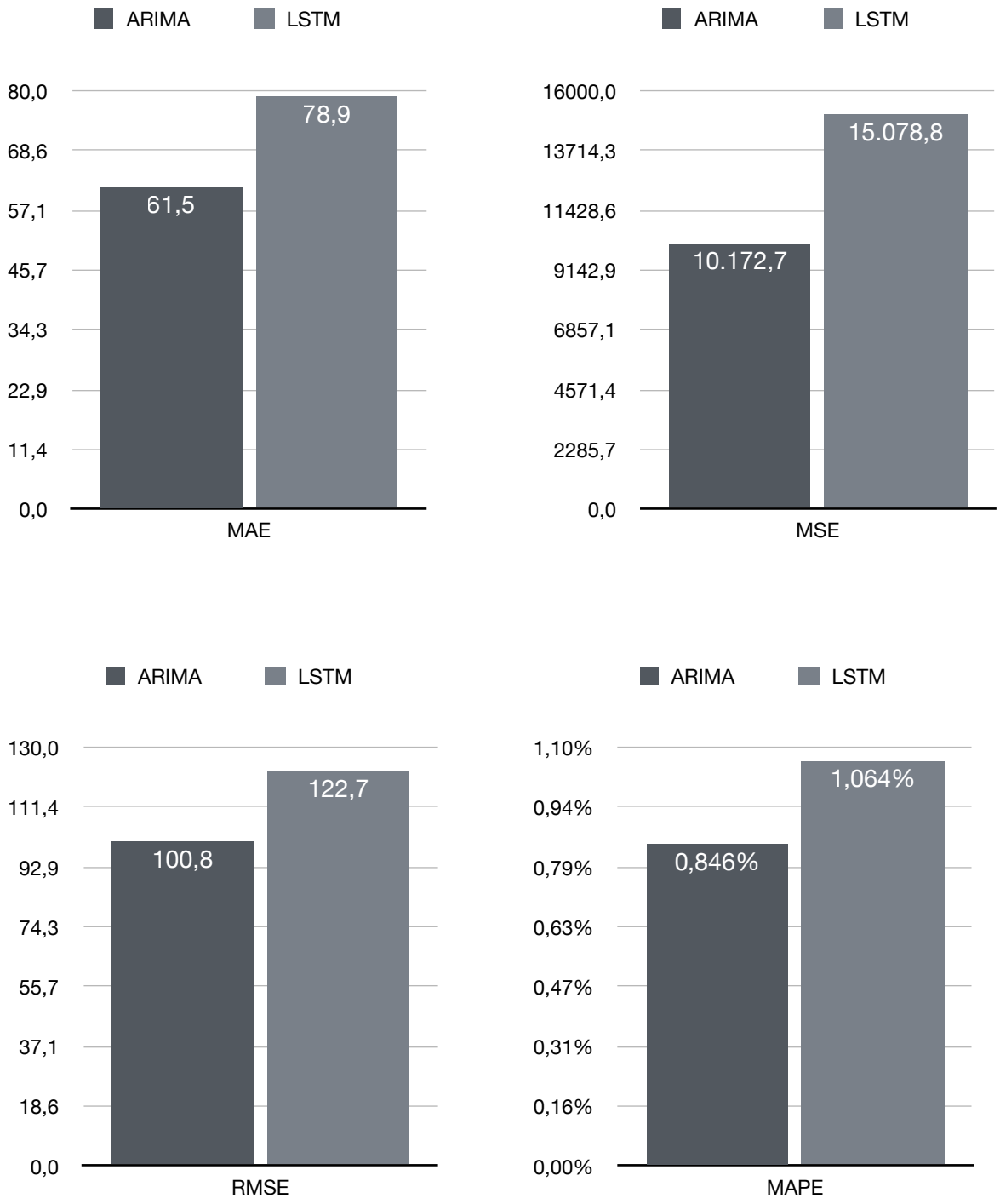

Figura 8.12: Confronto previsione one-step NASDAQ

8.4.2 Confronto previsione giorno successivo

Per il confronto del giorno successivo, viene preso in considerazione solamente il primo tipo di previsione (confrontando il valore reale con quello previsto). Ovviamente questo tipo di confronto non è possibile utilizzarlo per determinare quale tra i due modelli sia il migliore in termini di performance. Infatti, questa tipologia di confronto viene fatta solamente per riuscire a capire il vero utilizzo dei modelli nel mondo reale. I prezzi reali e i prezzi previsti nelle tabelle 8.22, 8.23 e 8.24 si riferiscono al giorno 17 Febbraio 2021.

S&P 500:

Tabella 8.22: Confronto previsione giorno successivo S&P 500

	Prezzo reale	Previsione	Errore (MAE)
ARIMA	3931,33$	3933,93$	2,6
LSTM	3931,33$	3925,70$	5,63

Dow Jones Industrial:

Tabella 8.23: Confronto previsione giorno successivo Dow Jones Industrial

	Prezzo reale	Previsione	Errore (MAE)
ARIMA	31613,01$	31527,62$	85,39
LSTM	31613,01$	31530,01$	83

NASDAQ:

Tabella 8.24: Confronto previsione giorno successivo NASDAQ

	Prezzo reale	Previsione	Errore (MAE)
ARIMA	13965,49$	14053,93$	88,44
LSTM	13965,49$	14049,99$	84,5

8.4.3 Analisi delle prestazioni

I risultati dei test sono basati su tre indici azionari (S&P 500, Dow Jones Industrial e NASDAQ), quindi i modelli lavorano su tre tipi di serie temporali diverse (anche se il periodo considerato è uguale). Gli andamenti dei tre indici sono abbastanza simili tra loro, la principale differenza è il valore che assume ogni singolo indice azionario (prezzo di chiusura aggiustato o Adj_close), infatti tutti e tre hanno valori molto diversi tra loro.
Dalla visualizzazione delle figure 8.10, 8.11 e 8.12, si riesce a capire che la differenza di errore di previsione non è molto distante tra i due modelli. Infatti nella maggior parte delle situazioni, le metriche risultanti sono molto simili tra loro. Sia per quanto riguarda ARIMA, sia per LSTM, i risultati di previsione raggiunti possono essere considerati abbastanza buoni, gli errori calcolati non sono molto alti e questo significa che entrambi i modelli hanno una buona precisione in termini di accuratezza. Questo lo si può notare sopratutto dal risultato percentuale del MAPE, il quale risulta sempre un valore basso (in media poco meno dell'1% di errore). Considerando la previsione di tipo one-step, il modello ARIMA risulta avere una performance un pochino migliore rispetto al modello LSTM, anche se i risultati non sono molto lontani.

Per quanto riguarda invece il confronto della previsione del giorno successivo rappresentato nelle tabelle 8.22, 8.23 e 8.24, come già anticipato, non viene utilizzato per valutare i modelli. Esso è stato implementato solamente per riuscire a capire il reale utilizzo dei modelli costruiti. Il risultato di questa previsione può cambiare radicalmente di giorno in giorno. A volte può essere più preciso il modello ARIMA, mentre altre volte può essere più preciso il modello LSTM, questo perché la differenza di errore tra i due modelli, in generale, non è molto alta.

8.5 Valutazioni generali

In un'analisi tecnica, parametri come il Moving Average (MA), Exponential Moving Average (EMA), Moving Average Convergence/Divergence (MACD) e Relative Strength Index (RSI) vengono utilizzati comunemente per riuscire a comprendere meglio gli andamenti di mercato. Nessuna di queste metodologie ha dimostrato però di essere uno strumento di previsione costantemente corretto (a volte possono funzionare, altre volte invece no). Esistono comunque diverse tecniche all'avanguardia per la previsione del prezzo nel mercato azionario [24] [25] [26] come: Auto-Regressive Integrated Moving Average (ARIMA), Generalized Auto-Regressive Conditional Heteroscedasticity (GARCH), Artificial Neural Networks (ANNs), Linear e Multi-Linear Regression (LR, MLR), Randomwalk (RW), k-Nearest Neighbors (KNN),

Prophet, Long Short-Term Memory (LSTM) e altri. Ognuna di queste tecniche può avere dei pro e dei contro.
In uno studio condotto da Kenneth Page [27] in cui vengono comparati alcuni algoritmi di Machine Learning (ARIMA, GARCH, Prophet, KNN e reti neurali) per effettuare una previsione del prezzo di mercato di Amazon, viene dimostrato che ARIMA e le reti neurali sono i due modelli più performanti.
ARIMA è una delle tecniche più popolari e ampiamente utilizzate per fare previsioni nell'ambito finanziario. Infatti i risultati sperimentali ottenuti mediamente con questo modello hanno dimostrato un grande potenziale di previsione dei prezzi delle azioni a breve termine [28]. Invece, per quanto riguarda i modelli di reti neurali artificiali (ANN), secondo Secondo Khashei e Bijari [29], essi possono essere molto efficaci per la modellazione predittiva a causa della loro natura auto adattativa basata sui dati. Le reti neurali ricorrenti (RNN), sono un sottotipo di reti neurali artificiali che utilizzano connessioni di feedback. Diversi tipi di modelli RNN vengono utilizzati nella previsione delle serie temporali finanziarie. I modelli Long Short-Term Memory (LSTM) ne sono un esempio. Essi evitano problemi di dipendenza a lungo termine grazie alla loro struttura unica di unità di archiviazione, aiutando così a prevedere le serie storiche finanziarie [30].
Il motivo per cui si è scelto di confrontare ARIMA e LSTM in questo scritto, è proprio perché sono due tecniche molto utilizzate nel campo economico e finanziario. Tutti e due i modelli riescono a raggiungere una buona accuratezza di previsione e a volte può non essere così scontato riuscire a capire quale dei due sia il migliore. In alcuni casi è il modello ARIMA ad avere la meglio su LSTM [31], mentre in altri casi è LSTM ad avere prestazioni migliori rispetto ad ARIMA [32].
Nella ricerca svolta da Aloysius Edward e Jyothi Manoj [33] viene sviluppato un modello ARIMA per l'analisi e la previsione dei prezzi delle azioni di quattro aziende automobilistiche. I risultati sono buoni e la media di errore del MAPE delle quattro compagnie si aggira intorno all'1,8%. Mentre nel trattato scritto da Israt Jahan [34] viene creata una rete neurale ricorrente con LSTM per la previsione del prezzo di chiusura di cinque aziende tecnologiche (Amazon, Facebook, Google, Microsoft e Netflix). Anche in questo caso, i risultati sono soddisfacenti e la media di errore del MAPE delle cinque aziende è minore del 2%.
I risultati ottenuti nella presente ricerca sono buoni, essi sono calcolati sulla base della previsione giornaliera (previsione one-step) effettuata sull'interno insieme di test. In generale, sia in ARIMA che in LSTM, la media di errore del MAPE dei tre indici considerati (S&P 500, Dow Jones Industrial e NASDAQ) è leggermente inferiore dell'1%. Solitamente, però, non è facile riuscire a confrontare i risultati ottenuti con risultati derivanti da altre prove sperimentali, questo sopratutto a causa dell'utilizzo di diversi tipi di dati. I risultati di previsione dei modelli dipendono molto dal tipo di mercato che si sta analizzando. Il mercato di S&P 500, Dow Jones Industrial e

NASDAQ è un mercato che può essere definito come non molto volatile, ossia le variazioni giornaliere del prezzo non sono molto alte (in media intorno all'1% o anche meno). Mentre se viene analizzato un mercato molto più volatile, come ad esempio quello delle criptovalute in cui la variazione del prezzo giornaliero può essere molto alta (a volte può raggiungere anche il 30% del suo valore), è facile capire che le due cose sono difficili da paragonare. Utilizzando lo stesso modello su serie temporali diverse, i risultati possono essere molto diversi.

Molte società di investimento e trading stanno cercando di sviluppare algoritmi e modelli per riuscire a calcolare previsioni in relazione ai mercati finanziari. Quasi sempre, questi modelli non vengono esposti pubblicamente, ma vengono utilizzati da queste aziende per ricavare profitti in base agli esiti di previsione ottenuti (capire quando comprare o vendere). Spesso quindi è anche difficile riuscire a capire quali sono i risultati che sono stati raggiunti finora in questo ambito e se esistono algoritmi in grado di funzionare alla perfezione.

Conclusioni

In questo scritto sono stati implementati due algoritmi di Intelligenza Artificiale utilizzati per la previsione di serie temporali finanziarie.

Sono stati presi in considerazione tre indici (S&P 500, Dow Jones Industrial e NASDAQ) in un intervallo di tempo che parte dall'1 Gennaio 1990 fino ad arrivare al 17 Febbraio 2021. I dati relativi ad essi, sono stati analizzati ed utilizzati dai modelli ARIMA e LSTM per calcolare una previsione del prezzo.

Dai risultati ottenuti, è possibile dedurre che le prestazioni del modello ARIMA sono migliori rispetto ad LSTM, anche se la loro differenza di errore non è molto lontana.

Probabilmente, non è del tutto giusto concludere dicendo che, in generale, ARIMA è più preformante rispetto ad LSTM. Accumulando esperienza in questo campo, si potrebbero costruire modelli migliori, riuscendo così anche a ribaltare la situazione. Quello che è possibile dire, è che per il lavoro svolto in questo elaborato, l'accuratezza di previsione con il modello ARIMA è leggermente migliore rispetto alla precisione del modello LSTM.

Lo scopo iniziale di questo progetto é stato raggiunto, ma esso lascia ampio spazio all'aggiunta di nuove funzionalità. Esistono varie tecnologie che possono essere unite a quanto già sviluppato per riuscire ad effettuare previsioni ancora più accurate. La "Sentiment Analysis" ne è un esempio, essa è una tecnica di Machine Learning la quale riesce ad estrarre un "sentimento" (positivo, negativo o neutrale) eseguendo un'analisi su documenti testuali. Questo potrebbe aiutare molto a capire quali sono le intenzioni degli investitori e, di conseguenza, capire quale potrebbe essere l'andamento in relazione ad un determinato mercato azionario.

Bibliografia

[1] Sachin Date, “What is time series decomposition and how does it work?”, 20 Giugno 2020. URL: https://towardsdatascience.com/what-is-time-series-decomposition-and-how-does-it-work-9b67e007ae90

[2] Pandas. URL: https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html

[3] NumPy. URL: https://numpy.org

[4] Yahoo Finance. URL: https://it.finance.yahoo.com

[5] Libreria “yfinance”. URL: https://pypi.org/project/yfinance/

[6] Matplotlib. URL: https://matplotlib.org

[7] Plotly. URL: https://plotly.com

[8] Keith D. Foote, “A Brief History of Machine Learning”, 26 Marzo 2019. URL: https://www.dataversity.net/a-brief-history-of-machine-learning/#

[9] Haider Khalaf Allamy, “METHODS TO AVOID OVER-FITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING”, Dicembre 2014, URL: https://www.researchgate.net/publication/295198699_METHODS_TO_AVOID_OVER-FITTING_AND_UNDER-FITTING_IN_SUPERVISED_MACHINE_LEARNING_COMPARATIVE_STUDY

[10] Shakir Khan, Hela Alghulaiakh, “ARIMA Model for Accurate Time Series Stocks Forecasting”. URL: https://thesai.org/Downloads/Volume11No7/Paper_65-ARIMA_Model_for_Accurate_Time_Series.pdf

[11] War Ahmed, Mehrdad Bahador, “The accuracy of the LSTM model for predicting the S&P 500 index and the difference between prediction and backtesting”, https://www.diva-portal.org/smash/get/diva2:1213449/FULLTEXT01.pdf

[12] PRABHAT9, “How to Create an ARIMA Model for Time Series Forecasting in Python”, 29 Ottobre 2020. URL: https://www.analyticsvidhya.com/blog/2020/10/how-to-create-an-arima-model-for-time-series-forecasting-in-python/

[13] Pmdarima. URL: https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

[14] Statsmodel. URL: https://www.statsmodels.org/stable/index.html

[15] Doğuş Üniversitesi Dergisi, “FORECASTING DAILY AND SESSIONAL RETURNS OF THE ISE-100 INDEX WITH NEURAL NETWORK MODELS”, 2007, http://journal.dogus.edu.tr/index.php/duj/article/viewFile/86/pdf_eavci

[16] Enzo Grossi, Massimo Buscema, “Introduction to artificial neural networks”, Gennaio 2008. URL: https://www.researchgate.net/publication/5847739_Introduction_to_artificial_neural_networks

[17] Robin M. Schmidt, “Recurrent Neural Networks (RNNs): A gentle Introduction and Overview”, 23 Novembre 2019. URL: https://arxiv.org/pdf/1912.05911.pdf

[18] Pranjal Srivastava, “Essentials of Deep Learning : Introduction to Long Short Term Memory”, 10 Dicembre 2017. URL: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

[19] Keras. URL: https://keras.io

[20] TensorFlow. URL: https://www.tensorflow.org

[21] Alexei Botchkarev, “Evaluating performance of regression machine learning models using multiple error metrics in Azure Machine Learning Studio”. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3177507

[22] Scikit-learn. URL: https://scikit-learn.org/stable/

[23] Jason Brownlee, “How to Diagnose Overfitting and Underfitting of LSTM Models”, 8 Gennaio 2020. URL: https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/

[24] George S. Atsalakis, Kimon P. Valavanis, “SURVEYING STOCK MARKET FORECASTING TECHNIQUES - PART I: CONVENTIONAL METHODS”. URL: https://www.researchgate.net/profile/George-Atsalakis/publication/236620807_Surveying_stock_market_forecasting_techniques_-_Part_I_Conventional_methods/links/540e0dce0cf2df04e756c884/Surveying-stock-market-forecasting-techniques-Part-I-Conventional-methods.pdf

[25] George S. Atsalakis, Kimon P. Valavanis, “Surveying stock market forecasting techniques – Part II: Soft computing methods”. URL: https://reader.elsevier.com/reader/sd/pii/S0957417408004417?token=93666AAAAACF118DEE45A11A66E503794DB45C065479AC2CBE3CD87B3BB493EEDA727D79B55678678C203C834801C909

[26] Aishwarya Singh, “Stock Prices Prediction Using Machine Learning and Deep Learning Techniques”, 25 Ottobre 2018. URL: https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/

[27] Kenneth Page, “Stock Price Forecasting Using Time Series Analysis, Machine Learning and single layer neural network Models”, 25 Agosto 2019. URL: https://rpubs.com/kapage/523169

[28] Ayodele A. Adebiyi., Aderemi O. Adewumi, Charles K. Ayo, “Stock Price Prediction Using the ARIMA Model”, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7046047

[29] Mehdi Khashei e Mehdi Bijari, “An artificial neural network (p,d,q) model for timeseries forecasting”. URL: https://reader.elsevier.com/reader/sd/pii/S0957417409004850?token=FDB2DE2234FA8212302689864652F07CD03515459ADB369CF9028884572D0449F642014902A096B1FD2F7CEF4F1A271D

[30] Adil MOGHAR ,Mhamed HAMICHE, “Stock Market Prediction Using LSTM Recurrent Neural Network”. URL: https://reader.elsevier.com/reader/sd/pii/S1877050920304865?token=E0F350C8FED68FA48AFCE3BCBCC82E1A057653643E113E25BEDAFDB095CAC987A1716F6A233BB4E96B50EE461FE7482B

[31] Jae Hyuk Han, “Comparing Models for Time Series Analysis”. URL: https://repository.upenn.edu/cgi/viewcontent.cgi?article=1166&context=wharton_research_scholars

[32] Sima Siami Namin, Akbar Siami Namin, “FORECASTING ECONOMIC AND FINANCIAL TIME SERIES: ARIMA VS. LSTM”. URL: https://arxiv.org/pdf/1803.06386.pdf

[33] Aloysius Edward e Jyothi Manoj, “FORECAST MODEL USING ARIMA FOR STOCK PRICES OF AUTOMOBILE SECTOR”, 4 Aprile 2016. URL: http://euroasiapub.org/wp-content/uploads/2016/10/1FMApril-3374-1.pdf

[34] Israt Jahan, “STOCK PRICE PREDICTION USING RECURRENT NEURAL NETWORKS”, Giugno 2018. URL: https://library.ndsu.edu/ir/bitstream/handle/10365/28797/Stock%20Price%20Prediction%20Using%20Recurrent%20Neural%20Networks.pdf?sequence=1&isAllowed=y