

UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE



**Analisi di Google PlayStore e delle azioni GOOG
mediante l'ecosistema Python**

Docenti:

Domenico URSINO
Luca VIRGILI

Studenti:

Lorenzo FRATINI
Federico MISCIA
Andrea PINCIAROLI

Anno Accademico 2021/2022

Indice

1	Introduzione	2
1.1	Datasets	2
2	Google Play Store	4
2.1	ETL	4
2.2	Data Visualization	8
3	Clustering	14
3.1	Preparazione del dataset ed operazioni preliminari	14
3.2	Clustering bidimensionale	15
3.2.1	K-Means	15
3.2.2	Clustering gerarchico	18
3.3	Clustering multidimensionale	19
3.3.1	PCA	19
3.3.2	Manifold t-SNE	22
3.3.3	DBSCAN	23
4	Classificazione	26
4.1	Valutazione dei classificatori	28
4.2	Classificazione con Grid Search	31
5	Serie Temporale	34
5.1	ETL	34
5.2	Analisi della serie	35
5.2.1	Augmented Dickey Fuller Test	36
5.2.2	Autocorrelazione (ACF) e Autocorrelazione parziale (PACF)	36
5.2.3	Creazione del modello ARIMA	37
5.2.4	Predizione fuori dal campione e metriche	39

Capitolo 1

Introduzione

La presente relazione riporta i risultati dell’analisi di due dataset; il primo contenente le informazioni circa le applicazioni scaricate dal Google Play Store dal 2010 al 2018, mentre il secondo contiene i dati storici delle azioni GOOG di Alphabet Inc. nel periodo 2014-2021.

I dati da noi selezionati sono pubblici e sono consultabili ai seguenti link:

- Google Play Store: <https://www.kaggle.com/lava18/google-play-store-apps>
- GOOG stock data: <https://finance.yahoo.com/quote/GOOG/>

La scelta di due dataset è dovuta al fatto che si effettuano analisi di tipo differente, infatti quello contenente le informazioni riguardo le applicazioni di Google PlayStore viene utilizzato per effettuare Clustering e Classificazione, mentre il secondo, visto che contiene i dati giornalieri delle azioni GOOG nel periodo dal 2014 al 2021, viene utilizzato per l’analisi delle serie temporali, al fine di estrarre delle previsioni future.

In questo tipo di analisi si farà uso di un linguaggio ampiamente utilizzato nell’ambito della Data Science, ovvero Python. Il motivo della sua importanza in questo campo è dovuto anche al numero di librerie che il linguaggio mette a disposizione, come Pandas, Seaborn, Matplotlib, Sklearn e Statsmodel. Tutte le librerie citate sono accessibili gratuitamente e godono di un ottimo supporto da parte della community. Nel progetto che verrà descritto di seguito si farà uso dei tool sopra elencati, al fine di effettuare delle operazioni di estrazione e visualizzazione dei dati, allenando inoltre dei regressori e classificatori, e facendo degli studi specifici su una serie temporale di interesse.

1.1 Datasets

Come anticipato, i dateset utilizzati sono due: il primo contiene le informazioni circa le applicazioni presenti nel PlayStore di Google, il secondo contiene i dati storici delle azioni GOOG di Alphabet Inc.

In questa sezione si riporta, con maggiore dettaglio, la descrizione di entrambi, nell’intento di chiarire il significato di alcuni degli attributi che, nel seguito, verranno utilizzati per le analisi.

Partendo dal dataset contenente i dati del PlayStore di Google, di seguito si riporta una

1.1. DATASETS

breve descrizione degli attributi presenti.

Google Play Store	
Attributo	Descrizione
App	Nome dell'applicazione
Category	Categoria a cui l'applicazione appartiene
Rating	Rating dell'applicazione fornito dagli utenti
Reviews	Numero di recensioni dell'applicazione fornite dagli utenti
Size	Dimensione dell'applicazione
Type	Tipo dell'applicazione: gratis o a pagamento
Price	Costo dell'applicazione
Content Rating	Intervallo d'età a cui l'applicazione è rivolta.
Genres	Genere/i a cui l'applicazione appartiene
Last Updated	Data dell'ultimo aggiornamento dell'applicazione nel PlayStore
Current Ver	Versione corrente dell'applicazione disponibile nel Play Store
Android Ver	Versione Android minima richiesta

Per quel che riguarda il dataset contenente le informazioni GOOG, analogamente, si riporta di seguito una breve descrizione degli attributi in esso presenti.

GOOG Alphabet Inc.	
Attributo	Descrizione
Date	Data di riferimento
Open	Prezzo della prima transazione nella data di riferimento
High	Prezzo massimo nella data di riferimento
Low	Prezzo minimo nella data di riferimento
Close	Prezzo dell'ultima transazione nella data di riferimento
AdjClose	Prezzo di chiusura rettificato per riflettere il valore dopo la contabilizzazione di eventuali azioni societarie
Volume	Numero di azioni scambiate in un giorno

Capitolo 2

Google Play Store

In questo capitolo verranno trattate le fasi del lavoro che, a partire da una breve analisi del dataset riguardante le App di Google Play Store, hanno portato fino all'implementazione delle tecniche di clustering e classificazione su di esso. Verranno inoltre descritti e motivati i risultati ottenuti mediante le due tecniche di apprendimento, con l'obiettivo di estrarre nuova conoscenza dal dataset in questione.



2.1 ETL

Nella fase preliminare del progetto, si è caricato il dataset su *Jupyter Notebook* e sono state eseguite delle operazioni di preprocessing e di visualizzazione dei dati mediante le librerie **pandas**, **matplotlib**, e **seaborn**.

Innanzitutto, dopo aver letto il file *.csv* contenente il dataset, per controllare che tutto fosse andato a buon fine, si sono mostrate le prime cinque righe, come si può vedere in Figura 2.1.

```
data = pd.read_csv("Dataset/googleplaystore.csv", encoding='utf-8')
data.head()
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

Figura 2.1: Dataset iniziale

Una volta caricato il dataset, è iniziata una fase di pulizia e studio, sempre per mezzo della libreria **pandas**, così da acquisire consapevolezza nei confronti del dominio di inte-

2.1. ETL

resse.

Per prima cosa, come riportato in Figura 2.2, si è osservato che il dataset conteneva dei valori nulli, da gestire, dunque, opportunamente. Infatti, ignorandoli, sarebbero potuti nascere potenziali problemi nelle fasi successive, come ad esempio la classificazione, con conseguente imprevedibilità nei risultati.

#checking NaN values	
	data.isnull().sum()
App	0
Category	0
Rating	1474
Reviews	0
Size	0
Installs	0
Type	1
Price	0
Content Rating	1
Genres	0
Last Updated	0
Current Ver	8
Android Ver	3
dtype:	int64

Figura 2.2: Valori nulli

A tal proposito, si è entrati più nel dettaglio delle colonne contenenti i valori nulli. L'approccio è stato *ad hoc* per ciascuna di esse, infatti nel caso della colonna "Rating" si è deciso di riempire i valori nulli mediante la media dei valori presenti nella stessa colonna, mentre nel caso delle altre colonne si sono riempiti i valori mancanti considerando il dato più frequente nella colonna stessa. La decisione di utilizzare tale approccio ha, alla base, una semplice motivazione. Dato che il numero di valori nulli per la colonna "Rating" era abbastanza elevato, colmare tali valori con quello più frequente avrebbe sbilanciato ulteriormente il dataset, cosa che invece non accade con le altre colonne, visto che il numero di valori mancanti, in esse, era notevolmente inferiore.

Fatto ciò, la fase successiva ha previsto la conversione di formato di alcuni campi, avendo inizialmente solamente l'attributo "Rating" come numerico. Alla luce di ciò, si è deciso di eseguire un cast dei seguenti attributi:

- **Reviews:** è stato convertito in un campo *Int*
- **Size:** è stato convertito in un campo *Float*, dopo che ogni valore fosse stato opportunamente trasformato in MB, come viene mostrato in Figura 2.3
- **Installs:** è stato convertito in un campo *Int*
- **Price:** è stato convertito in un campo *Float*
- **Last Update:** è stato convertito in un campo *DateTime*

Infine, l'ultima fase di preprocessing dei dati è stata quella in cui si è svolto un controllo dei valori duplicati. Per fare ciò, si è fatto uso della funzione *duplicates()* di Pandas, la quale ha restituito che ben 483 righe su 10357 erano duplicate. Per risolvere questo

2.1. ETL

```
def convert_size(x):
    if 'k' in x:
        x = float(x.replace('k', '')) 
        x /= 1024
    return x

#Si considera per semplicità la dimensione dell'app in MB
data["Size"] = data["Size"].apply(lambda x: x.replace("M", ""))
data["Size"] = data["Size"].apply(lambda x: convert_size(x))
data['Size'] = data['Size'].replace("Varies with device", np.nan)
data['Size'] = data['Size'].replace("1,000+", "1.000")
data['Size'] = data['Size'].astype('float')
data['Size'] = data['Size'].fillna(data['Size'].mean())
```

Figura 2.3: Conversione dell'attributo *Size*

problema, molto semplicemente si è applicata la funzione *drop_duplicates()*, la quale ha permesso di avere tutte righe tra loro differenti.

2.1. ETL

Una tabella riassuntiva riguardo i dati e i relativi tipi, a seguito di tutte le operazioni che sono state effettuate, è mostrata in Figura 2.4.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10357 entries, 0 to 10840
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   App                10357 non-null   object  
 1   Category           10357 non-null   object  
 2   Rating              10357 non-null   float64 
 3   Reviews             10357 non-null   object  
 4   Size                10357 non-null   float64 
 5   Installs            10357 non-null   object  
 6   Type                10357 non-null   object  
 7   Price               10357 non-null   float64 
 8   Content Rating     10357 non-null   object  
 9   Genres              10357 non-null   object  
 10  Last Updated        10357 non-null   object  
 11  Current Ver         10357 non-null   object  
 12  Android Ver         10357 non-null   object  
dtypes: float64(3), object(10)
memory usage: 1.1+ MB
```

Figura 2.4: Informazioni sulle colonne del dataset

2.2. DATA VISUALIZATION

2.2 Data Visualization

In questa fase si è svolta un'analisi qualitativa e descrittiva del dataset appena trattato, al fine di comprenderne meglio il contenuto. Il primo grafico che si è ricavato, riportato in Figura 2.5, riguarda la distribuzione delle applicazioni all'interno del dataset, raggruppate per categoria.

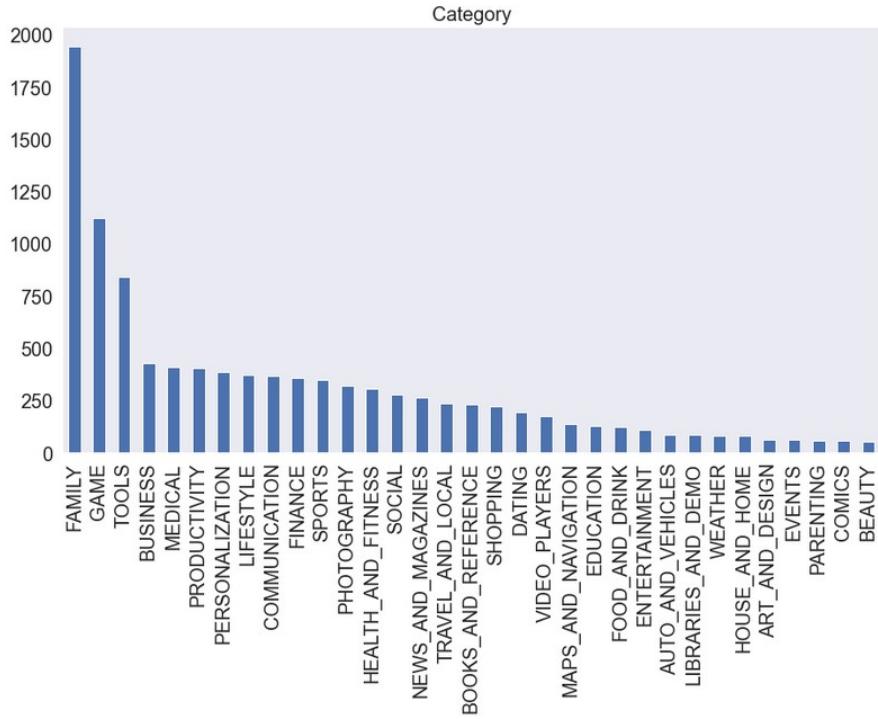


Figura 2.5: Grafico a barre di *Category*

Da tale grafico a barre si può osservare come ci sia una netta prevalenza di applicazioni di tipo FAMILY, GAMES e TOOLS. Le restanti categorie, invece, risultano essere più di nicchia rispetto alle tre appena citate.

In secondo luogo, è stato realizzato un grafico a torta che mostra la ripartizione tra le app "free" e quelle a pagamento, sulla base della colonna TYPE.

Osservando la Figura 2.6, si nota chiaramente che la maggior parte delle App presenti nel Play Store è gratuita, mentre solo meno del 10% richiedono un pagamento.

2.2. DATA VISUALIZATION

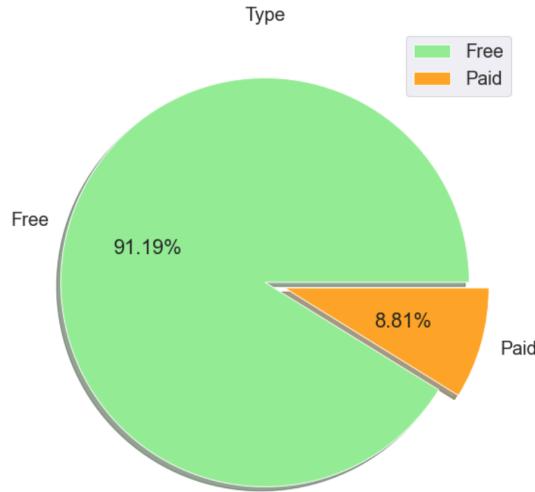


Figura 2.6: Grafico a torta di *Type*

Più interessante, probabilmente, è il grafico a torta riportato in Figura 2.7, il quale illustra la classificazione delle App sulla base del target di utenza cui esse sono destinate, descritto tramite la variabile categorica CONTENT RATING.

Il grafico mostra come ben più della metà delle applicazioni, quasi il 70%, siano classificate come adatte a tutti, dai 10 anni in su. C'è poi un 20% di App valutate genericamente come "Everyone" mentre, in minore percentuale, sono presenti le categorie "Mature 17+" e "Teen".

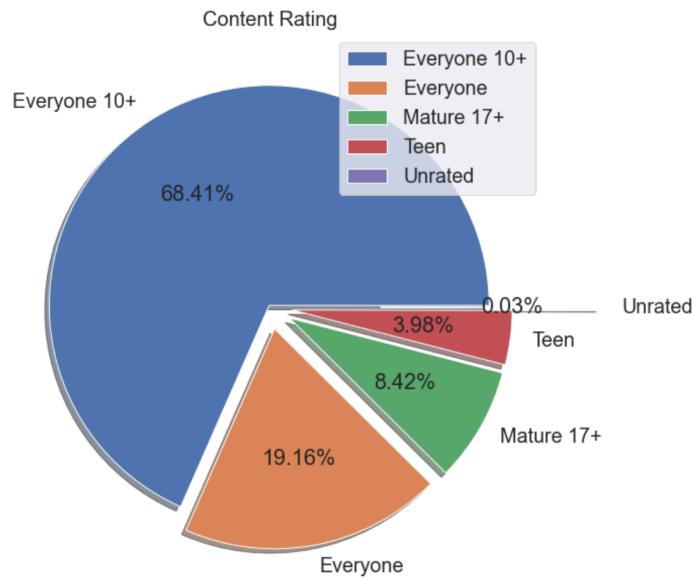


Figura 2.7: Grafico a torta di *Content Rating*

2.2. DATA VISUALIZATION

Successivamente, sono stati realizzati due histogrammi, il primo relativo alla distribuzione del rating delle applicazioni (Figura 2.8), il secondo inerente alla memoria occupata dalle stesse sullo smartphone (Figura 2.9).

La distribuzione della valutazione delle App, visualizzando anche il *kernel density estimate*, assomiglia ad una campana asimmetrica con la maggior parte delle applicazioni votate con un rating superiore a 4.

La distribuzione delle dimensioni delle applicazioni, invece, presenta un andamento tendenzialmente decrescente con l'eccezione di un picco in corrispondenza del bin che raccoglie le dimensioni 20-25 MB.

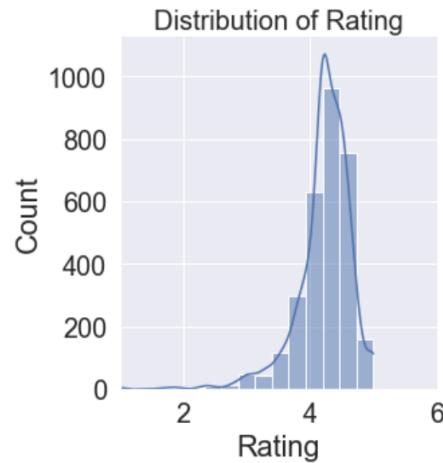


Figura 2.8: Distribuzione del *Rating*

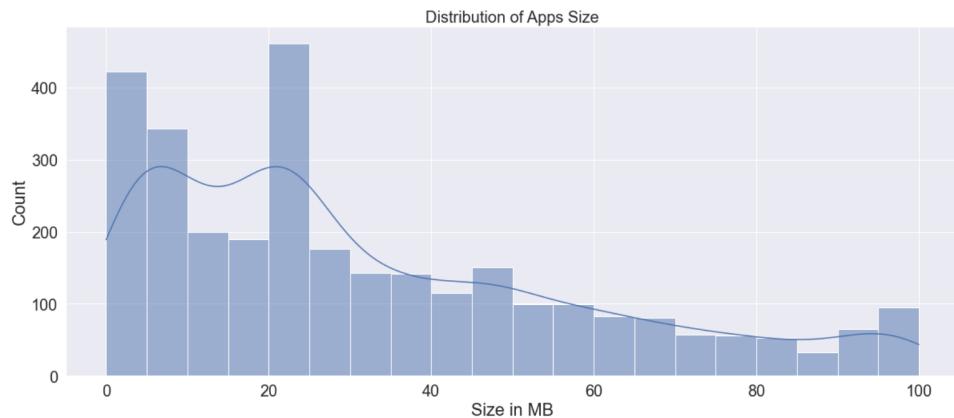


Figura 2.9: Distribuzione di *Size*

2.2. DATA VISUALIZATION

A questo punto, nell'ultima parte della visualizzazione dei dati si sono volute mostrare delle distribuzioni più specifiche. Ad esempio, in Figura 2.10 è mostrata la distribuzione delle prime 10 applicazioni per Rating e Reviews.

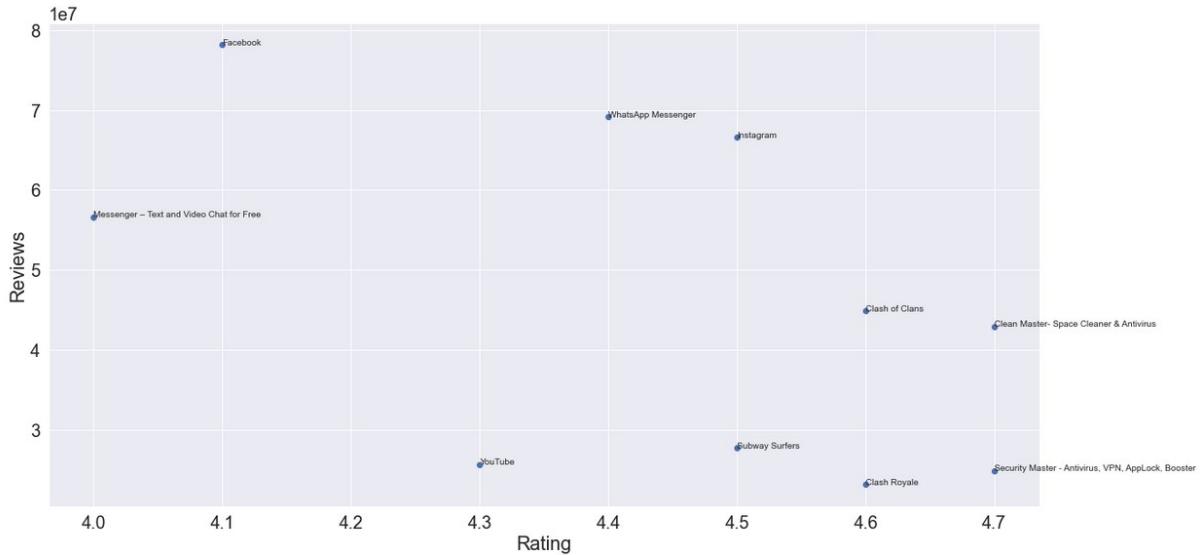


Figura 2.10: Scatter Plot app per reviews e rating

Da tale grafico si possono osservare quali sono le applicazioni più famose all'interno del dataset, con la caratteristica che tutte sono gratis, anche se ciò non si vede direttamente dallo scatterplot.

Proprio per questa ragione la successiva parte dell'analisi è stata quella di focalizzarsi sulle applicazioni a pagamento, le quali costituiscono una nicchia parte del dataset, come mostrato in Figura 2.6.

Il focus dell'analisi è stato sempre quello di comprendere l'andamento del rating, infatti, in una prima fase, esso è stato messo in relazione con la quello delle applicazioni gratis, come mostrato in Figura 2.11.

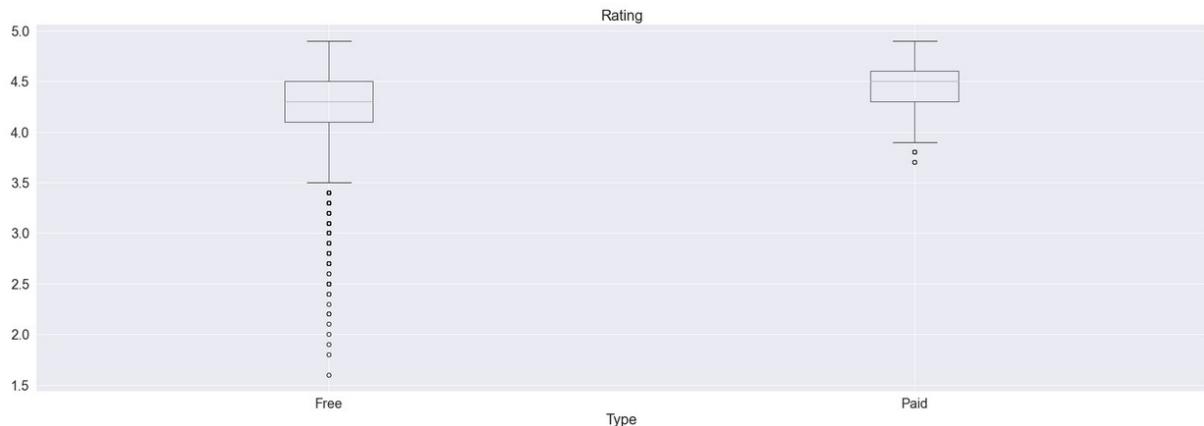


Figura 2.11: Box Plot per tipologia applicazione

Da tale boxplot si osserva come le applicazioni a pagamento abbiano mediamente un rating maggiore rispetto a quelle gratis, con una concentrazione di tali valori in un intervallo

2.2. DATA VISUALIZATION

più ristretto rispetto a quelle gratis, il che è giustificato anche da un numero di outlier (rappresentati dai cerchi in figura) inferiore.

A questo punto, per analizzare meglio la distibuzione del rating per le applicazioni a pagamento si è deciso di ripetere la stessa analisi considerando le cinque fasce di prezzo (0.99€, 1.99€, 2.99€, 3.99€, 4.99€) con un maggiore numero di app, come mostrato in Figura 2.12.

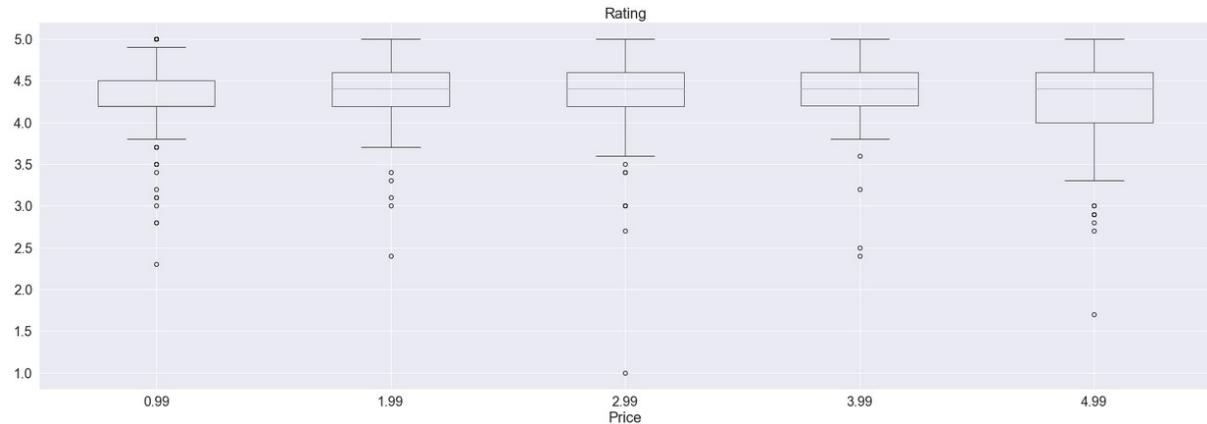


Figura 2.12: Box Plot per prezzo applicazione

Tale analisi conferma come le applicazioni che costano 0.99€ sono quelle con un intervallo di rating più ristretto delle altre, ma con un numero maggiore di outlier. Comunque, in tutte e cinque le fasce mediamente si riscontra quanto già visto in Figura 2.11, ovvero che le applicazioni a pagamento sono valutate molto positivamente dagli utenti.

L'ultima analisi effettuata ha visto la creazione di un pairplot, in modo da evidenziare eventuali correlazioni tra tutte le possibili coppie di feature del dataset, sulla base di disposizioni significative dei punti.

Il risultato è mostrato in Figura 2.13, in cui c'è una distinzione su base colore delle tre categorie principali di App.

2.2. DATA VISUALIZATION

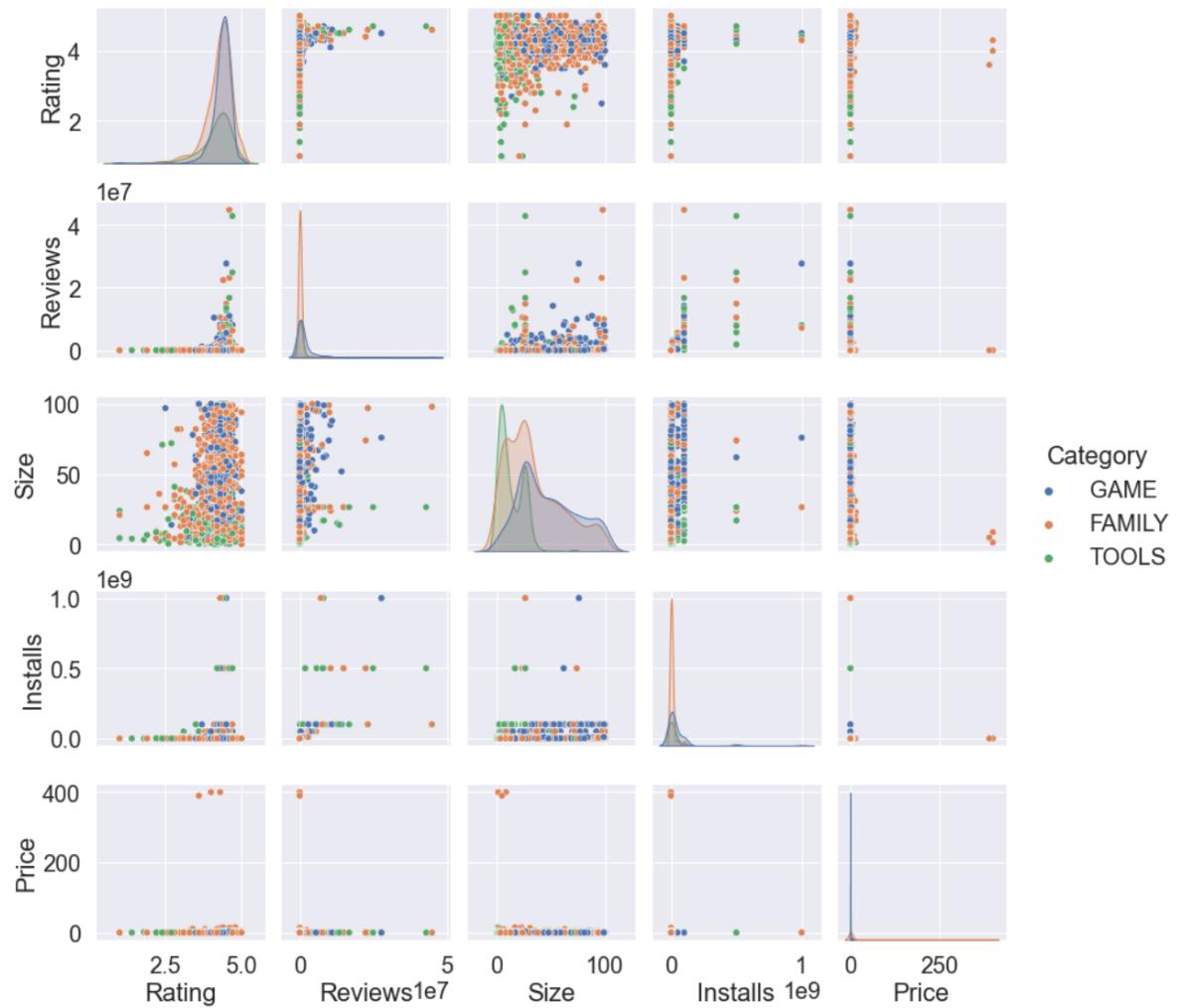


Figura 2.13: Pairplot tra le diverse coppie di variabili

Dal pairplot si evince che non ci sono, a questo livello, correlazioni evidenti tra le feature; piuttosto, si nota come i dati tendano a disporsi, in molti casi, per "fasce". Ciò è dovuto al fatto che diversi attributi del dataset, pur pensati come numerici, siano stati valorizzati sulla base di una suddivisione in bin, assomigliando, dunque, a variabili categoriche.

Il pairplot della Figura 2.13, infine, verrà ripreso nel capitolo 3, per la scelta di coppie di feature sulle quali effettuare il clustering.

Capitolo 3

Clustering

Una volta aver acquisito un buon livello di conoscenza e consapevolezza del dataset Google Play Store attraverso la precedente fase di analisi descrittiva, lo step successivo del lavoro si è focalizzato sulle attività di clustering dei dati. Attraverso questa tecnica esplorativa di analisi dei dati, si è cercato di organizzare le informazioni contenute nel dataset all'interno di gruppi significativi senza avere alcuna conoscenza a monte dell'appartenenza degli elementi a tali gruppi. Il clustering, infatti, rientra nella famiglia delle tecniche di apprendimento non-supervisionato ed ha a che fare con dati non etichettati o comunque dalla struttura ignota, motivo per cui si prefigge l'obiettivo di definire gruppi di oggetti sulla base della loro similarità, individuando strutture e relazioni significative nei dati. L'intento di questo capitolo è quello di mettere in luce le modalità operative, le scelte effettuate ed i risultati ottenuti attraverso l'implementazione di diverse tecniche di clustering sul dataset di riferimento.

3.1 Preparazione del dataset ed operazioni preliminari

Prima di procedere con il clustering, si è rivelato necessario effettuare delle operazioni di pre-processing aggiuntive rispetto a quanto illustrato nel Capitolo 2. L'aspetto del dataset risultante è mostrato in Figura 3.2.

Nello specifico, a causa dell'elevata popolosità del dataset in questione, è stato applicato un filtraggio delle righe. In primo luogo, facendo riferimento alla colonna LAST UPDATED, sono state selezionate solamente le App il cui ultimo aggiornamento risalisse a non prima dell'anno 2018. Dopodiché, tra queste, sono state considerate esclusivamente le App appartenenti alle tre categorie principali, cioè *Family*, *Games* e *Tools*. Così facendo, il dataset su cui operare per il clustering è arrivato a contare 2152 istanze, ritenute in numero sufficiente.

In seguito, sono state eliminate le colonne non utili per le operazioni di clustering. Le colonne rimaste sono: RATING, REVIEWS, SIZE, INSTALLS e PRICE.

Come ultimo passo, avendo constatato che i valori per tali attributi appartenevano a scale di grandezze significativamente diverse, è stata effettuata un'operazione di standardizzazione, necessaria al fine di poter dare in input, ai modelli di clustering, dati con metriche simili e, di conseguenza, agevolare il computing degli algoritmi stessi. Per quest'ultima operazione si è scelto di utilizzare lo *StandardScaler* del modulo *preprocessing* di *ScikitLearn*.

3.2. CLUSTERING BIDIMENSIONALE

La Figura 3.1 mostra i passaggi eseguiti in questa sede.

```
# Selezione sulla base dell'ultimo aggiornamento
condition = data["Last Updated"].str.contains('2018', regex=False)
data = data[condition]

# Selezione delle prime 3 categorie di app più frequenti
data = data[(data['Category'] == "FAMILY") | (data['Category'] == "GAME") | (data['Category'] == "TOOLS")]

# Rimozione colonne non significative per l'analisi
data.drop(['Category', 'App', 'Content Rating', 'Genres', 'Last Updated',
           'Current Ver', 'Android Ver', 'Type'], axis=1, inplace=True)

# Standardizzazione
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
standard_data = sc.fit_transform(data)
```

Figura 3.1: Preparazione dataset per il clustering

	Rating	Reviews	Size	Installs	Price
1653	0.521543	1.054454	1.192406	0.912582	-0.051884
1654	0.521543	8.007045	1.524310	11.552417	-0.051884
1655	0.299205	6.425166	1.450553	5.641398	-0.051884
1656	0.966218	-0.198104	-0.430234	-0.151402	-0.051884
1657	0.521543	-0.229577	0.417964	-0.151402	-0.051884

Figura 3.2: Dataset pronto per il clustering

3.2 Clustering bidimensionale

La prima analisi affrontata, nell'ambito del clustering, ha preso in considerazione due feature specifiche del dataset disponibile. Basandosi sul pairplot della Figura 2.13, sono stati scelti gli attributi RATING e SIZE, rispettivamente la valutazione e l'occupazione di memoria di un'app. La combinazione di questi attributi, infatti, è stata l'unica a dar luogo ad uno scatterplot che fosse abbastanza omogeneo nello spazio e poco influenzato dalla suddivisione in bin che caratterizzava i valori di diverse colonne del dataset di partenza.

3.2.1 K-Means

Per poter effettuare il clustering sulla base di questi due attributi, si è scelto di utilizzare in primo luogo l'algoritmo K-Means.

Una particolarità di questa tecnica è che essa richiede di conoscere in anticipo il parametro k , cioè il numero di cluster in cui si intende partizionare i dati. Per poter disporre di tale informazione, dunque, è stato seguito il cosiddetto *elbow method*, un'euristica che itera l'algoritmo “K-Means” adattandolo, di volta in volta, ad un numero crescente di cluster (da 1 a 12) così da ricavare il parametro k ottimale, in relazione ad un coefficiente

3.2. CLUSTERING BIDIMENSIONALE

di distorsione.

Per implementare e visualizzare l'*elbow method*, qui così come nei successivi clustering, è stata sfruttata la libreria *Yellowbrick*. Il grafico ottenuto è mostrato nella Figura 3.3: esso individua un "gomito" in corrispondenza di $k=3$, punto dal quale la curva registra una flessione rilevante nella propria pendenza.

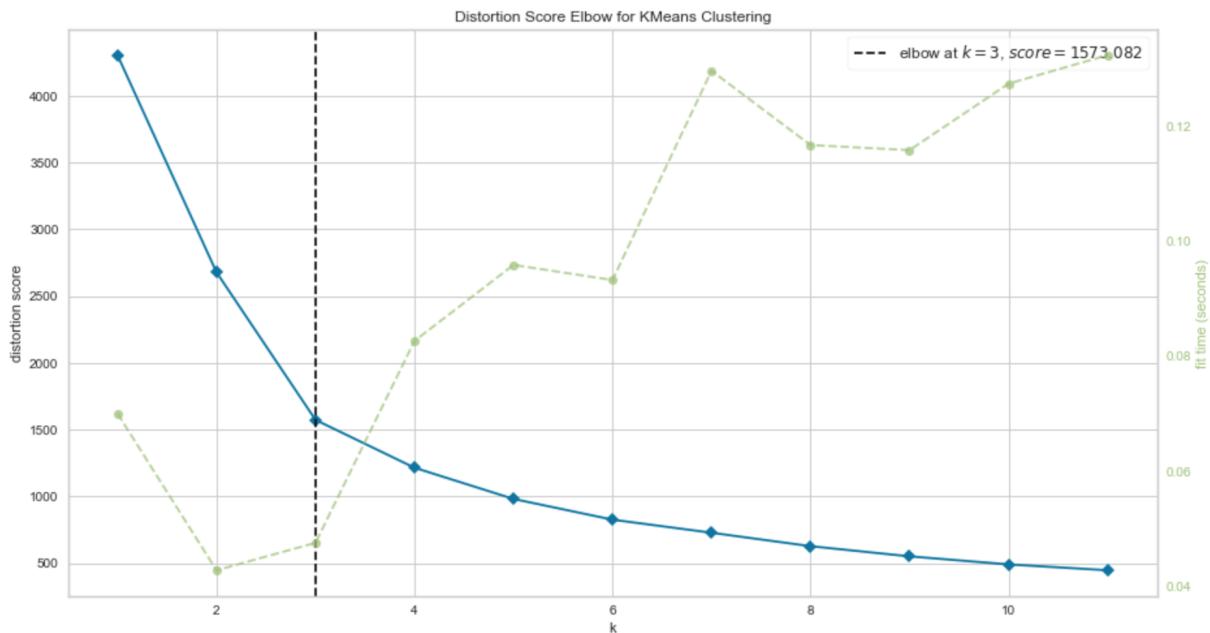


Figura 3.3: Grafico dell'elbow method

Alla luce di ciò, il numero ottimale di cluster, con cui allenare il modello del KMeans offerto da *ScikitLearn*, è stato scelto pari a 3. Il risultato del clustering è mostrato tramite lo scatterplot della Figura 3.4.

Ricordando che i valori che compaiono sugli assi in Figura 3.4 sono standardizzati, per l'interpretazione dei risultati si farà riferimento al corrispondente valore non trasformato, ricavato invertendo le operazioni dello *StandardScaler*.

Innanzitutto, è possibile notare come i 3 cluster siano stati individuati in maniera abbastanza "rigida". Ad ogni modo, un primo cluster (Cluster 0) è stato utilizzato per raggruppare le app che potremmo definire "efficienti", in quanto hanno un rating abbastanza elevato, da 3.9 a 5, e un'occupazione di memoria tutto sommato contenuta, al massimo 45 MB.

Il cluster etichettato con "1" si riferisce a quelle applicazioni con valutazione soddisfacente, come il precedente, ma con occupazione di memoria superiore, da 45 MB a 100 MB. Il terzo cluster, con label "Da migliorare", invece, raccoglie tutte quelle app "mediocri", che hanno valutazione inferiore a 3.9 e dimensioni variabili in tutto il range da pochi MB a 100 MB.

Dopodiché, al fine di ottenere una validazione della bontà del clustering, è stata graficata la metrica *silhouette*. Essa fornisce un'indicazione di quanto ciascun elemento si adatta bene al cluster cui è stato assegnato, assegnandogli uno score tra -1 e 1, sulla base del grado di similarità con i rimanenti elementi del cluster.

3.2. CLUSTERING BIDIMENSIONALE



Figura 3.4: Clustering con KMeans sugli attributi Rating e Size

Il grafico della *silhouette* è mostrato in Figura 3.5. La silhouette media risulta pari a 0.47 e, come è possibile notare, buona parte degli elementi appartenenti a ciascun cluster superano questa media, a riprova del fatto che sono ben rappresentati dal proprio cluster. Fanno eccezione alcuni elementi, seppur pochi, appartenenti ai cluster 1 e 2, i quali presentano uno score di silhouette negativo, ad indicare che potevano essere rappresentati meglio da un altro cluster.

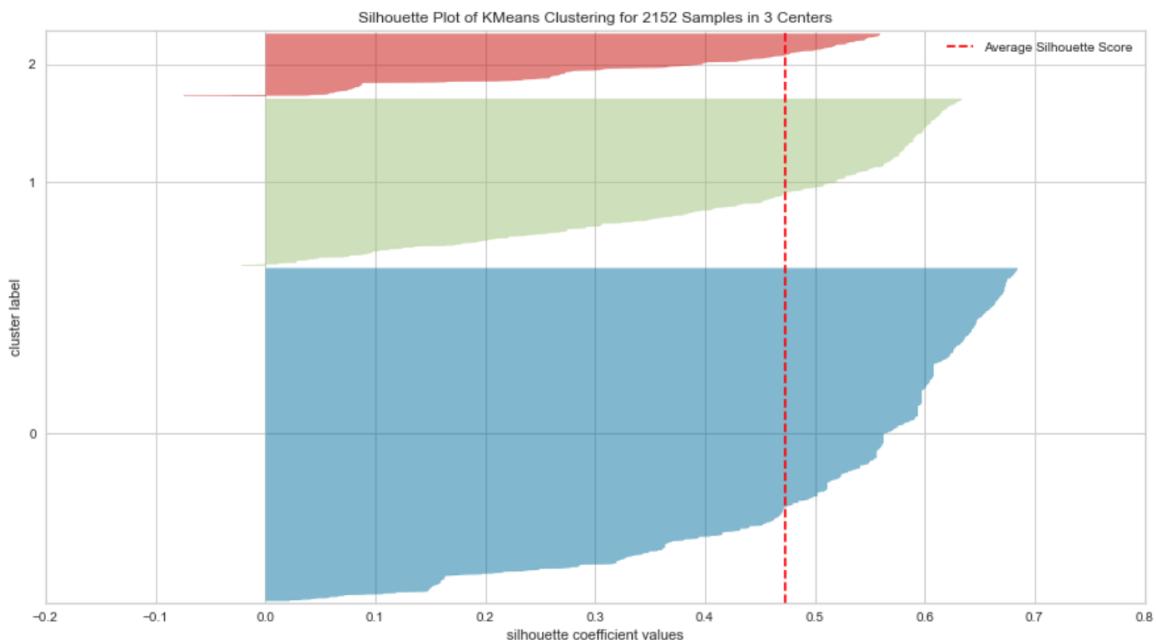


Figura 3.5: Rappresentazione grafica della metrica Silhouette

3.2. CLUSTERING BIDIMENSIONALE

3.2.2 Clustering gerarchico

In seguito, per disporre di un termine di paragone circa l'operato del KMeans, si è deciso di effettuare un clustering gerarchico, sempre sugli attributi Rating e Size.

Il clustering gerarchico si basa su un approccio di tipo bottom-up: parte da N cluster individuali ed esegue, iterativamente, il *merge* tra coppie di cluster selezionate secondo determinati requisiti di similarità.

Lo scatterplot in Figura 3.6 mostra la suddivisione in cluster ottenuta per mezzo del modello *AgglomerativeClustering* di *sklearn*, avendo settato un numero di cluster pari a 3 ed il criterio di merge "ward", il quale minimizza la varianza tra i cluster da unire.

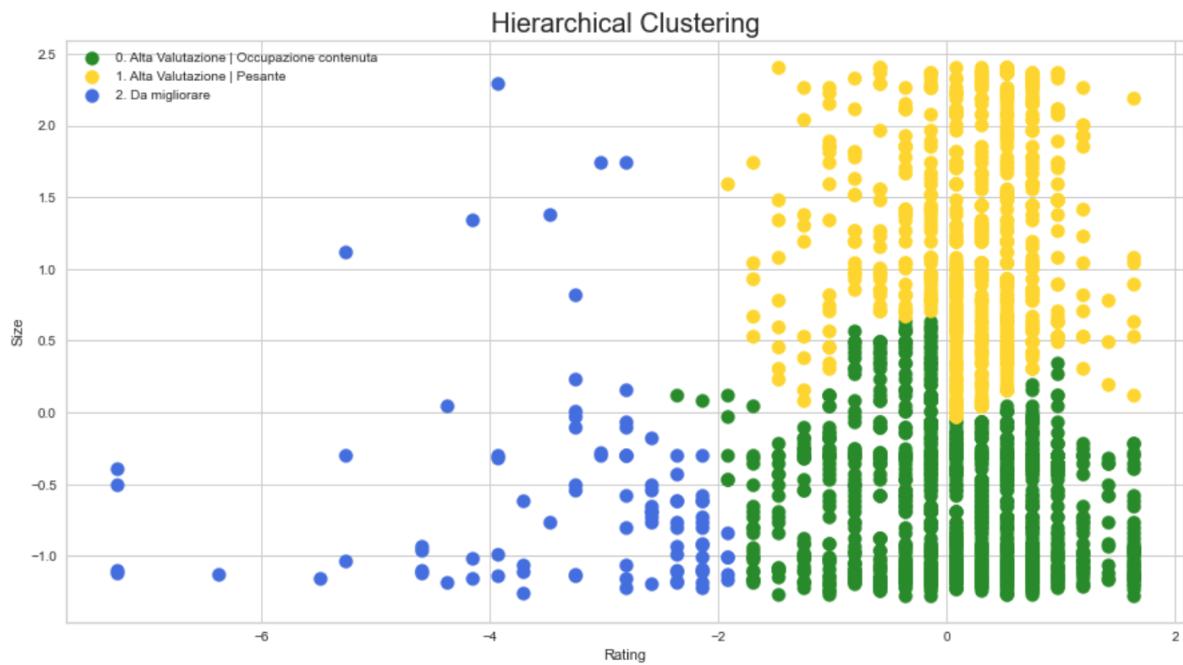


Figura 3.6: Rappresentazione grafica della metrica Silhouette

I raggruppamenti ottenuti in seguito al clustering gerarchico non sono molto dissimili da quelli individuati dal KMeans; sono rintracciabili, però, delle differenze specialmente in prossimità della zona di confine tra i vari cluster, con una suddivisione più irregolare rispetto al KMeans.

Infine, è stato calcolato lo score medio di *silhouette* per il clustering gerarchico appena discusso; esso è pari a 0.43 dunque inferiore rispetto a quanto ottenuto con il KMeans.

3.3. CLUSTERING MULTIDIMENSIONALE

3.3 Clustering multidimensionale

3.3.1 PCA

Dopo aver condotto un'analisi circoscritta a due sole feature del dataset, il passo successivo è stato quello di prendere in considerazione tutte le feature disponibili.

Quando si parla di clustering n-dimensionale, entra in gioco la tecnica della PCA (Principal Component Analysis), la quale permette di ridurre la dimensionalità del dataset proiettando i dati in un nuovo spazio di caratteristiche e, al tempo stesso, conservando la maggior parte delle informazioni rilevanti.

Inizialmente, sono stati effettuati alcuni tentativi di ridurre la dimensionalità a 2 *componenti principali* ma i risultati non sono stati soddisfacenti e, per questo, non riportati in questa sede.

Per capirne le motivazioni e cercare di ottenere risultati migliori, dunque, è stata analizzata l'*explained variance*, come indice dell'informazione contenuta in ciascuna delle componenti principali risultanti dalla PCA. A tal proposito, sono stati calcolati autovettori ed autovalori della matrice di covarianza del dataset e, a partire da questi, la varianza spiegata. Il grafico riportato in Figura 3.7 mostra la varianza spiegata da ciascuna componente della PCA e la varianza cumulativa.

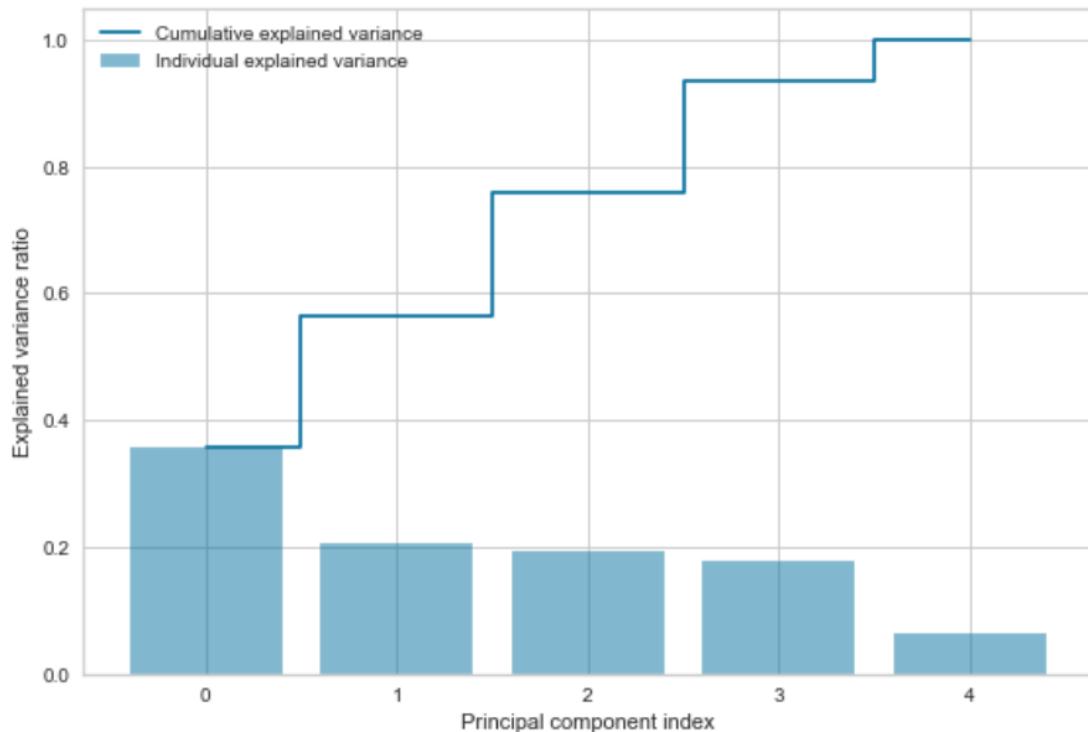


Figura 3.7: Analisi dell'explained variance in relazione alle componenti della PCA

Come è possibile notare dalla Figura 3.7, le prime due componenti, considerate nei primi test, sono quelle a più alto contenuto informativo. Nonostante questo, però, si raggiunge un livello di varianza spiegata accettabile (80% o più) considerando quattro componenti. Tale considerazione chiarifica l'iniziale fallimento.

3.3. CLUSTERING MULTIDIMENSIONALE

Per poter implementare la tecnica della PCA, è stato utilizzato l'omonimo componente del package *sklearn.decomposition*. Una volta aver eseguito la *transform* del dataset di partenza per ottenere le quattro componenti principali, dunque, si è passati alla visualizzazione del clustering, costruita sempre tramite il KMeans.

Nella Figura 3.8 è riportato l'elbow method, il quale suggerisce di considerare 5 cluster.

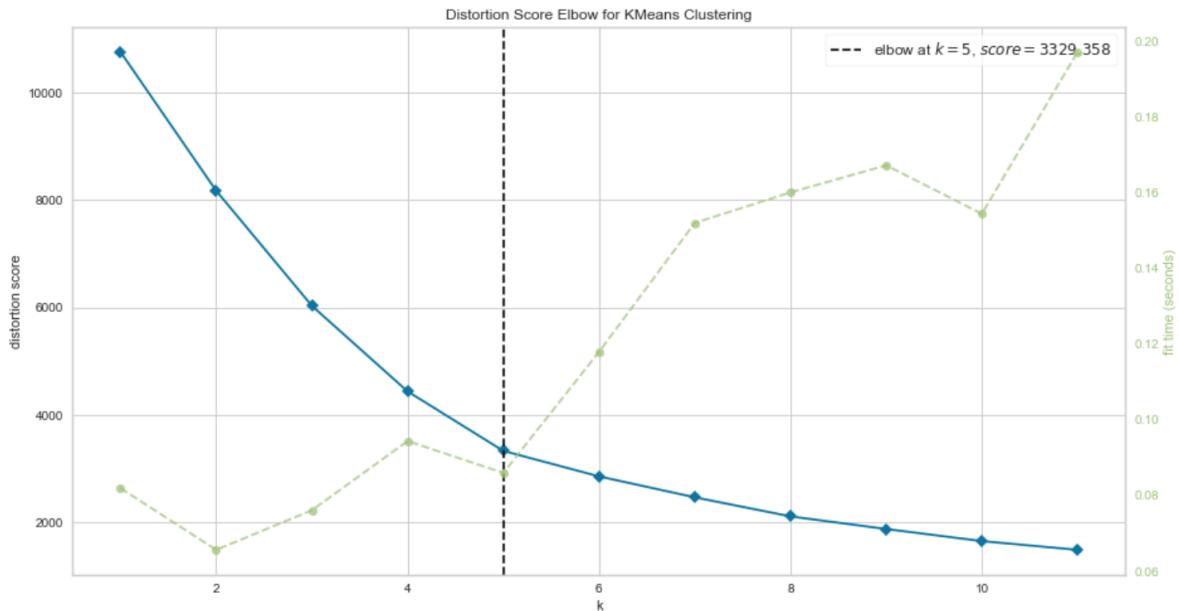


Figura 3.8: Elbow method per determinare il numero di cluster da utilizzare. Il parametro ottimale risulta $k=5$

Lo scatterplot in Figura 3.9 mostra il clustering risultante dalla proiezione tramite PCA delle feature del dataset di partenza, in combinazione con il KMeans. Avendo a che fare con più di due componenti, si è scelto di utilizzare una rappresentazione tridimensionale, così da arrivare a visualizzarne, se non altro, tre delle quattro totali.

Osservando la Figura 3.9, è possibile notare la presenza di tre cluster (Cluster 0, Cluster 1 e Cluster 4) parzialmente sovrapposti e densi. Dopodiché, è stato individuato il Cluster 3 con un numero esiguo di elementi, per altro abbastanza sparsi. Infine è presente un ultimo cluster (Cluster 2) che è nettamente isolato dagli altri e con pochissimi elementi. Per quanto sia difficoltoso capire come i diversi parametri siano stati utilizzati per definire i diversi cluster, si è provato a dare una valutazione dei cluster risultanti attraverso la *silhouette*, mostrata in Figura 3.10. Lo score medio di *silhouette* è piuttosto basso (0.4) e lo spessore dei plot che rappresentano i diversi cluster non è uniforme. Per il cluster 1, inoltre, ci sono diversi elementi con *silhouette* negativa. Sono tutti sintomi, questi, di un clustering non ottimale.

3.3. CLUSTERING MULTIDIMENSIONALE

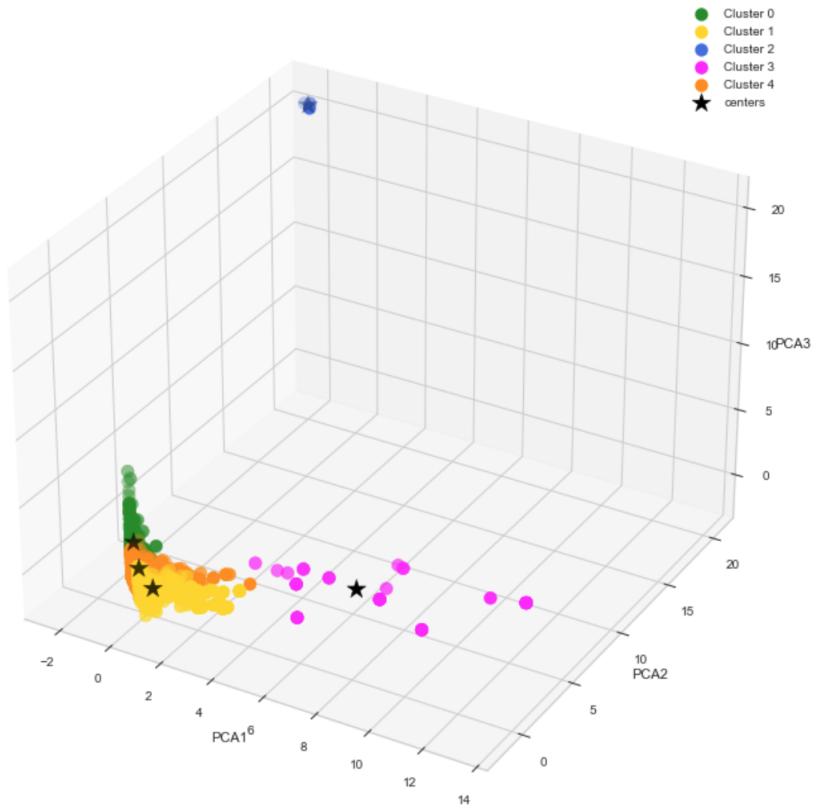


Figura 3.9: Clustering tramite KMeans di tutte le feature del dataset, ridotte per mezzo della PCA

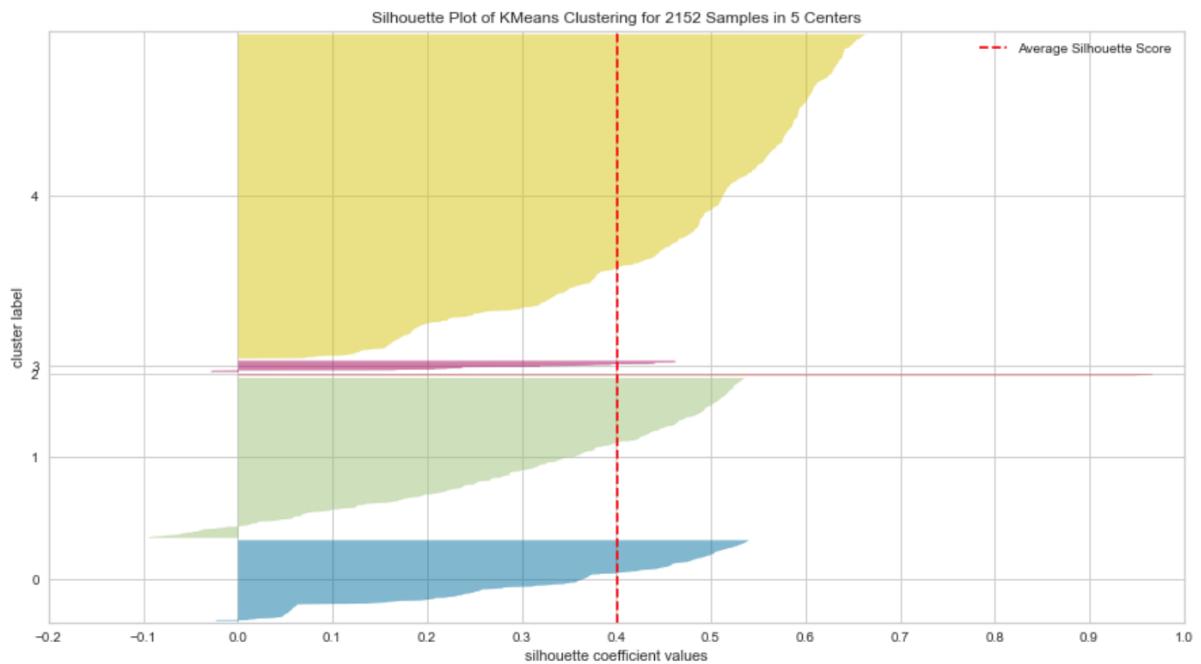


Figura 3.10: Analisi della silhouette per i cluster ottenuti dal dataset trasformato tramite PCA.

3.3. CLUSTERING MULTIDIMENSIONALE

3.3.2 Manifold t-SNE

Alla luce dei risultati ottenuti effettuando il clustering a seguito dell'applicazione della PCA, si è andati alla ricerca di una metodologia diversa per la riduzione della dimensionalità del dataset.

La scelta è ricaduta su una tecnica non lineare chiamata **t-Distributed Stochastic Neighbor Embedding (t-SNE)**, il cui obiettivo è proprio quello di agevolare la visualizzazione di dati con diverse feature. Tale algoritmo modella i punti in modo che oggetti vicini nello spazio originale risultino vicini nello spazio a dimensionalità ridotta, e oggetti lontani risultino lontani, cercando così di conservare la struttura locale.

E' stato possibile applicare il t-SNE in Python sfruttando il modulo `sklearn.manifold`, il quale mette a disposizione la classe TSNE.

Il procedimento seguito è stato analogo a quanto già fatto con la PCA: in primo luogo è stato trasformato il dataset di partenza attraverso il metodo `fit_transform` della classe TSNE così da ricavare due componenti "embedded", dopodiché è stato utilizzato l'elbow method per individuare il numero di cluster ed infine è stato applicato il KMeans ai dati non trasformati per predire le label di appartenenza ad un cluster piuttosto che a un altro. Dopo una prima fase di ottimizzazione dei parametri, sono stati ottenuti i cluster mostrati in Figura 3.11.

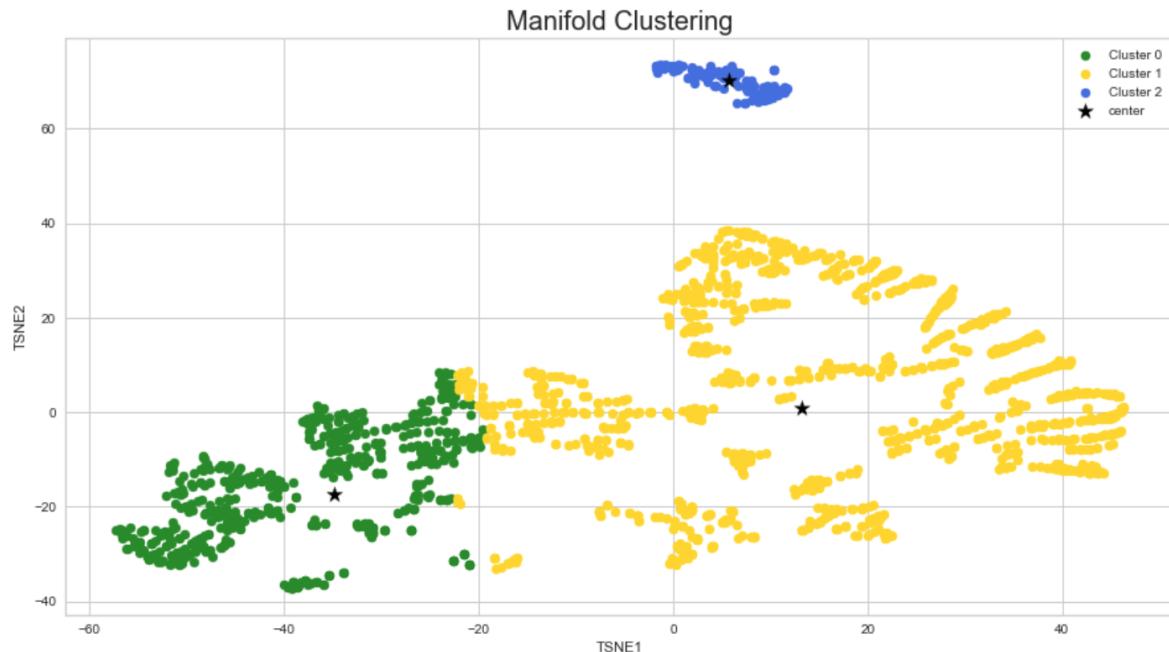


Figura 3.11: Clustering ottenuto decomponendo il dataset attraverso la tecnica t-SNE.

Al fine di valutare la bontà del clustering, è stata studiata la *silhouette*, ottenendo così il plot della Figura 3.12. Da quest'ultimo, si nota una *silhouette* media di 0.54 ed uno score superiore al valor medio per gran parte degli elementi di ciascun cluster, infatti si osservano solo lievi fluttuazioni nello spessore dei singoli plot.

Per le motivazioni appena elencate, il clustering in esame è stato ritenuto accettabile.

3.3. CLUSTERING MULTIDIMENSIONALE

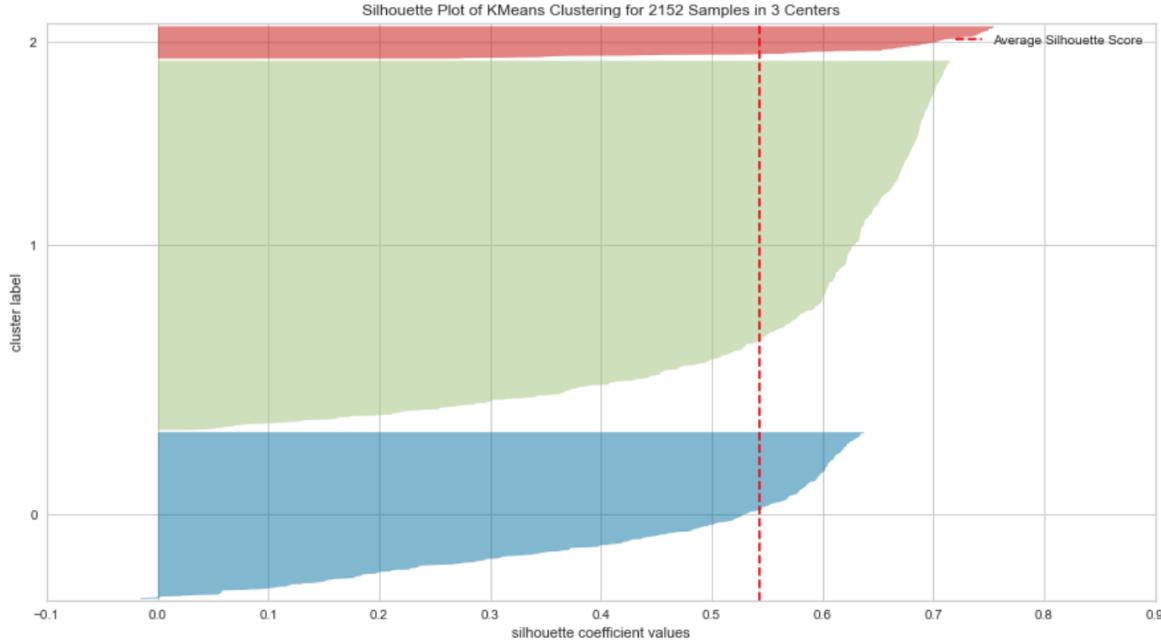


Figura 3.12: Silhouette per il clustering tramite t-SNE e KMeans.

3.3.3 DBSCAN

Avendo ottenuto dei risultati accettabili a seguito della riduzione della dimensionalità per mezzo della tecnica t-SNE, si è deciso di applicare la stessa, in combinazione con l'algoritmo DBSCAN, per la visualizzazione dei risultati.

Trattandosi di un clustering basato sulla densità, il DBSCAN non richiede la conoscenza a priori del numero di cluster come il KMeans, bensì di altri due parametri: *epsilon*, la distanza tra i punti affinché questi siano considerati parte dello stesso cluster, e *MinPts*, il numero minimo di punti per formare una "regione densa".

Per stimare la distanza *epsilon*, si è fatto uso della classe *NearestNeighbors* ed è stato realizzato un grafico che rappresenta l' *epsilon* al variare della distanza tra i punti, in relazione all'indice di questi ultimi. L'*epsilon* ideale è quello che corrisponde al punto della curva in cui la pendenza aumenta drasticamente, in maniera simile ad un andamento esponenziale. Il grafico in questione è riportato nella Figura 3.13; da esso si è ricavato *epsilon*=0.3.

3.3. CLUSTERING MULTIDIMENSIONALE

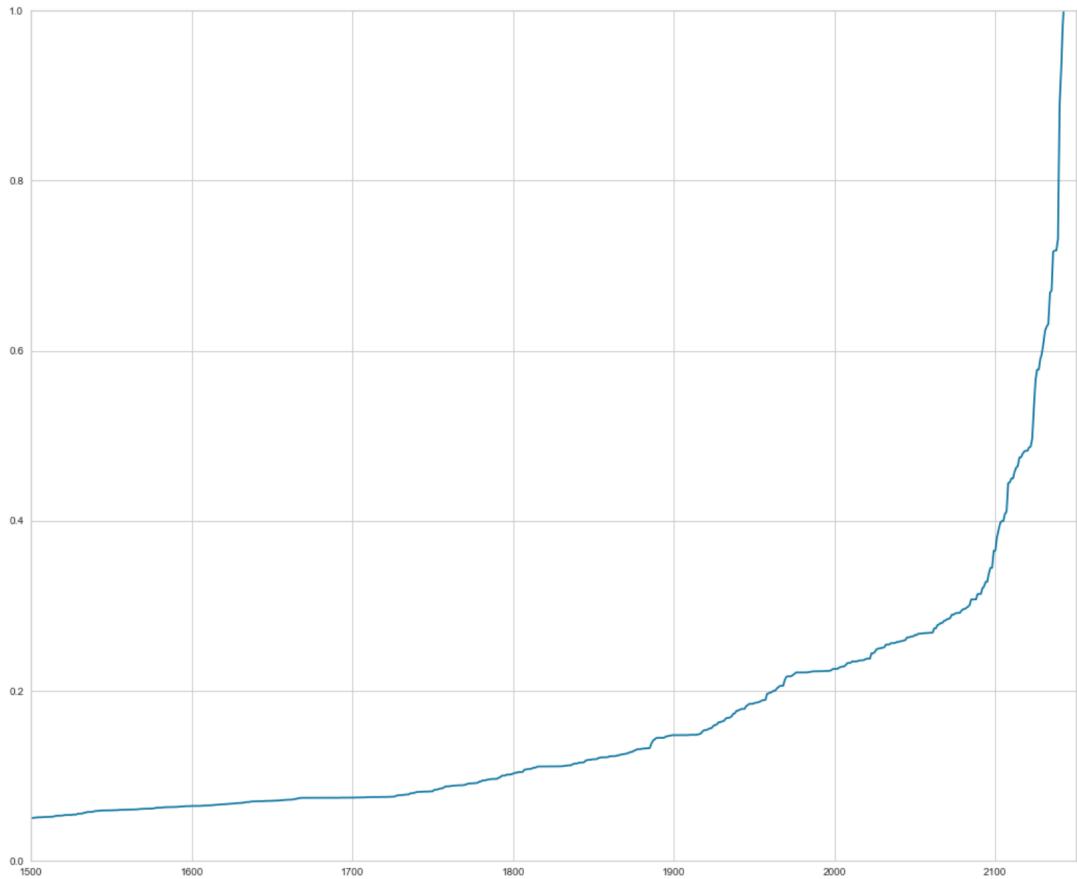


Figura 3.13: Grafico utilizzato per la stima del parametro ϵ . In ascissa sono riportati gli indici degli elementi, in ordinata le distanze.

Per quanto riguarda il parametro $MinPts$, facendo alcune ricerche, è emerso che solitamente si sceglie un valore maggiore o uguale al doppio del numero di feature del dataset; a tal proposito, si è posto $MinPts=10$.

Il clustering ottenuto per mezzo dell'algoritmo DBSCAN è visualizzato nella Figura 3.14.

3.3. CLUSTERING MULTIDIMENSIONALE

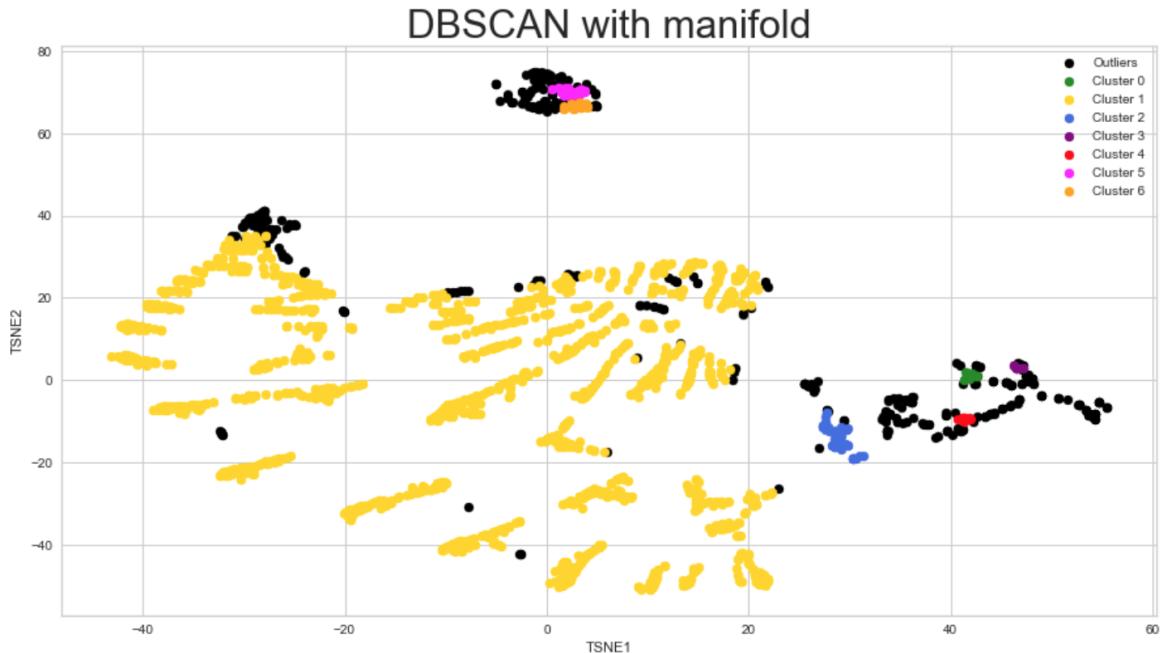


Figura 3.14: Clustering ottenuto applicando il DBSCAN combinato con il t-SNE.

Sono risultati ben sette cluster, tutti di dimensione ristretta, ad eccezione del Cluster 1 che occupa quasi totalmente lo scatterplot. Oltre a ciò, come è possibile notare, sono molti i punti che non è stato possibile inserire all'interno di un cluster, e perciò etichettati come outliers . Sembra che l'algoritmo DBSCAN non sia riuscito a lavorare bene con il dataset in considerazione; una spiegazione può essere la disomogeneità della densità degli elementi al suo interno.

Capitolo 4

Classificazione

In questa sezione si illustra la realizzazione di modelli predittivi, attraverso tecniche di classificazione, che si occupano di individuare attraverso l'utilizzo dei dati la categoria di un'applicazione.

Dato l'elevato numero di categorie, si è scelto di realizzare una classificazione binaria tenendo conto solo delle prime due categorie più presenti (FAMILY e GAMES), dove dal grafico in Figura 2.5 si può osservare anche che il numero di applicazioni è notevolmente superiore rispetto alle restanti.

Anche in questo caso sono state effettuate delle piccole operazioni di preprocessing, come già fatto relativamente al clustering. Innanzitutto, è stato filtrato il dataset mantenendo solamente le informazioni di interesse e associando alla categoria "Family" il valore 0, mentre a "Games" il valore 1. Il codice che ha realizzato tale operazione è mostrato in Figura 4.1.

```
#Si mantengono solo le due categorie più frequenti
data = data[(data['Category'] == "GAME") | (data['Category'] == "FAMILY")]
data["Category"] = data["Category"].map(lambda x: 0 if x == "FAMILY" else 1)
category = data["Category"]
```

Figura 4.1: Filtraggio dataset

Successivamente si è deciso di eliminare delle colonne non significative per la classificazione. Dopo tale operazione, le colonne rimaste sono: RATING, REVIEWS, SIZE, INSTALLS, PRICE, TYPE e CONTENT RATING.

Osservando i dati contenuti in tali colonne si è visto che le ultime due, cioè le colonne TYPE e CONTENT RATING presentavano dati categorici, i quali non sono utilizzabili negli algoritmi di classificazione. Per risolvere questo problema si è deciso di creare dei *dummies*, ovvero mediante la funzione di Pandas *getDummies* sono state specificate le colonne di interesse e automaticamente sono state create un numero di ulteriori colonne pari ai valori categorici valorizzati con 1 o 0 in funzione se l'attributo categorico fosse presente o meno nella riga della tabella. Per comprendere meglio tale output in Figura 4.2 viene mostrato il risultato ottenuto.

	Rating	Reviews	Size	Installs	Price	Type_Free	Type_Paid	Content Rating_Everyone	Content Rating_Everyone 10+	Content Rating_Mature 17+	Content Rating_Teen	Content Rating_Unrated
1653	4.5	4447388	67.0	100000000	0.0	1	0	0	1	0	0	0
1654	4.5	27722264	76.0	1000000000	0.0	1	0	0	1	0	0	0
1655	4.4	22426677	74.0	500000000	0.0	1	0	1	0	0	0	0
1656	4.7	254258	23.0	10000000	0.0	1	0	1	0	0	0	0
1657	4.5	148897	46.0	10000000	0.0	1	0	1	0	0	0	0

Figura 4.2: Creazione dummies

Sempre da tale figura si può osservare come i valori numerici presenti appartengano ad insiemi numerici molto diversi, per questo motivo si è deciso di procedere con un'operazione di **normalizzazione** dei dati affinchè tutti fossero compresi nell'intervallo [0,1]. In questo caso, si è fatto uso della funzione *MinMaxScaler* della libreria *Sklearn*, ottenendo i risultati mostrati in Figura 4.3.

	Rating	Reviews	Size	Installs	Price	Type_Free	Type_Paid	Content Rating_Everyone	Content Rating_Everyone 10+	Content Rating_Mature 17+	Content Rating_Teen	Content Rating_Unrated
1653	0.875	0.099064	0.669955	0.10	0.0	1	0	0	1	0	0	0
1654	0.875	0.617506	0.759967	1.00	0.0	1	0	0	1	0	0	0
1655	0.850	0.499549	0.739964	0.50	0.0	1	0	1	0	0	0	0
1656	0.925	0.005664	0.229895	0.01	0.0	1	0	1	0	0	0	0
1657	0.875	0.003317	0.459926	0.01	0.0	1	0	1	0	0	0	0

Figura 4.3: Normalizzazione dataset

Il passo successivo è stato quello di controllare se il dataset fosse bilanciato o meno. Facendo questo si è visto che attualmente nel dataset erano presenti 1943 applicazioni con categoria "FAMILY" e 1121 applicazioni con categoria "GAMES". Alla luce di ciò si è deciso di bilanciare il dataset considerando tutte le 1121 applicazioni di "GAMES", ma scegliendo un sample della stessa misura di tipo "FAMILY".

Una volta eseguite queste operazioni preliminari si è potuto procedere con l'**analisi di correlazione** del dataset, con l'obiettivo di trovare delle variabili che fossero correlate tra di loro.

Il risultato che si è ottenuto è visibile in Figura 4.4.

4.1. VALUTAZIONE DEI CLASSIFICATORI

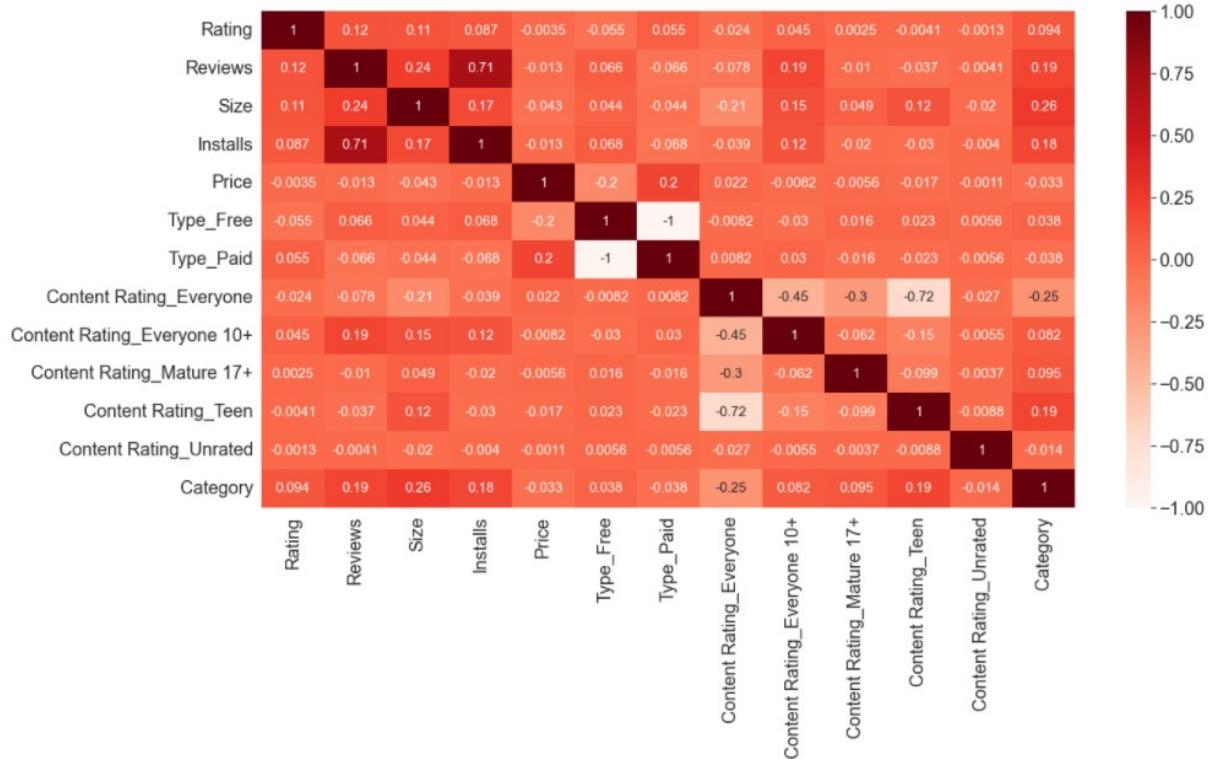


Figura 4.4: Analisi di correlazione

Da tale figura si osserva che le variabili "Type_Free" e "Type_Paid" sono fortemente correlate, infatti hanno una correlazione pari a 1, cioè il massimo, che è superiore alla soglia scelta di 0.8; per tale motivo si è scelto di eliminare la variabile "Type_Free". In realtà, data la scarsa presenza dei valori relativi alla colonna "Content Rating_Unrated" si è deciso di eliminare anche tale colonna dal dataset.

Una volta eseguiti tutti i passaggi sopra citati si è ottenuto un dataset bilanciato che verrà poi utilizzato per addestrare i classificatori.

4.1 Valutazione dei classificatori

Gli algoritmi di classificazione scelti sono stati i seguenti:

- LogisticRegression
- DecisionTreeClassifier
- SVC
- RandomForestClassifier
- AdaBoostClassifier
- GradientBoostingClassifier
- LinearDiscriminantAnalysis

4.1. VALUTAZIONE DEI CLASSIFICATORI

Per effettuare l’addestramento ed il test degli algoritmi, è necessario procedere alla suddivisione del dataset bilanciato in due parti. A tal proposito si è deciso di utilizzare il metodo dell’holdout, dove il train-set rappresenta l’80% dei dati mentre il restante 20% costituisce il test-set.

I risultati di accuratezza ottenuti dai diversi modelli sono stati i seguenti:

- LogisticRegression: 0.62
- DecisionTreeClassifier: 0.62
- SVC: 0.64
- RandomForestClassifier: 0.67
- AdaBoostClassifier: 0.72
- GradientBoostClassifier: 0.71
- LinearDiscriminantAnalysis: 0.63

Per visualizzare meglio i risultati di accuratezza ottenuti si riporta in Figura 4.5 la rappresentazione grafica indicando anche per ogni risultato la media della varianza e la relativa deviazione standard per comprendere il range in cui il valore può variare o meno.

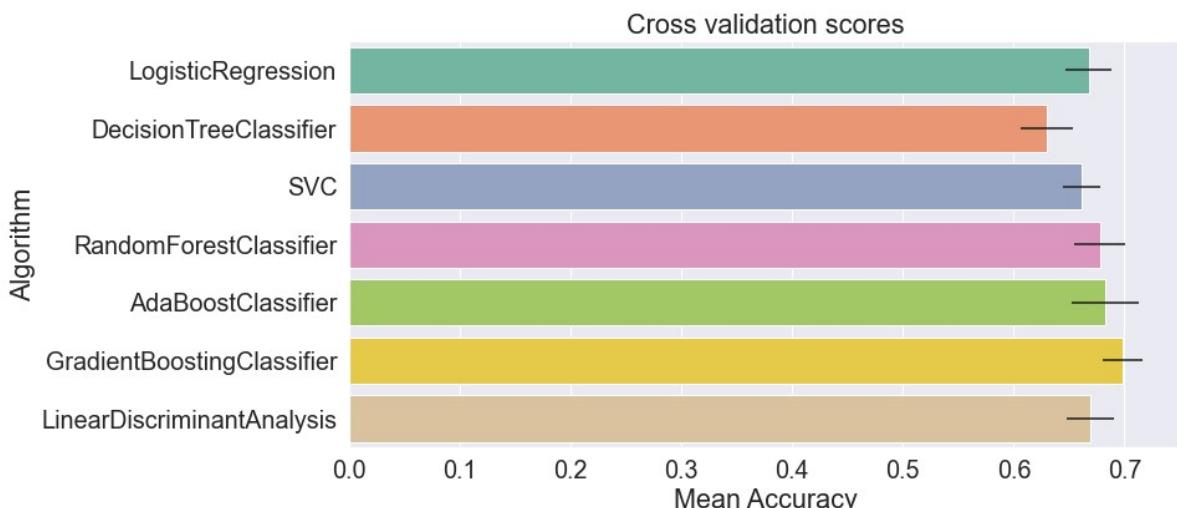


Figura 4.5: Accuratezza media

Sempre sulla base dei risultati ottenuti, in Figura 4.6 sono riportate le diverse matrici di confusione associate ad ogni classificatore.

4.1. VALUTAZIONE DEI CLASSIFICATORI

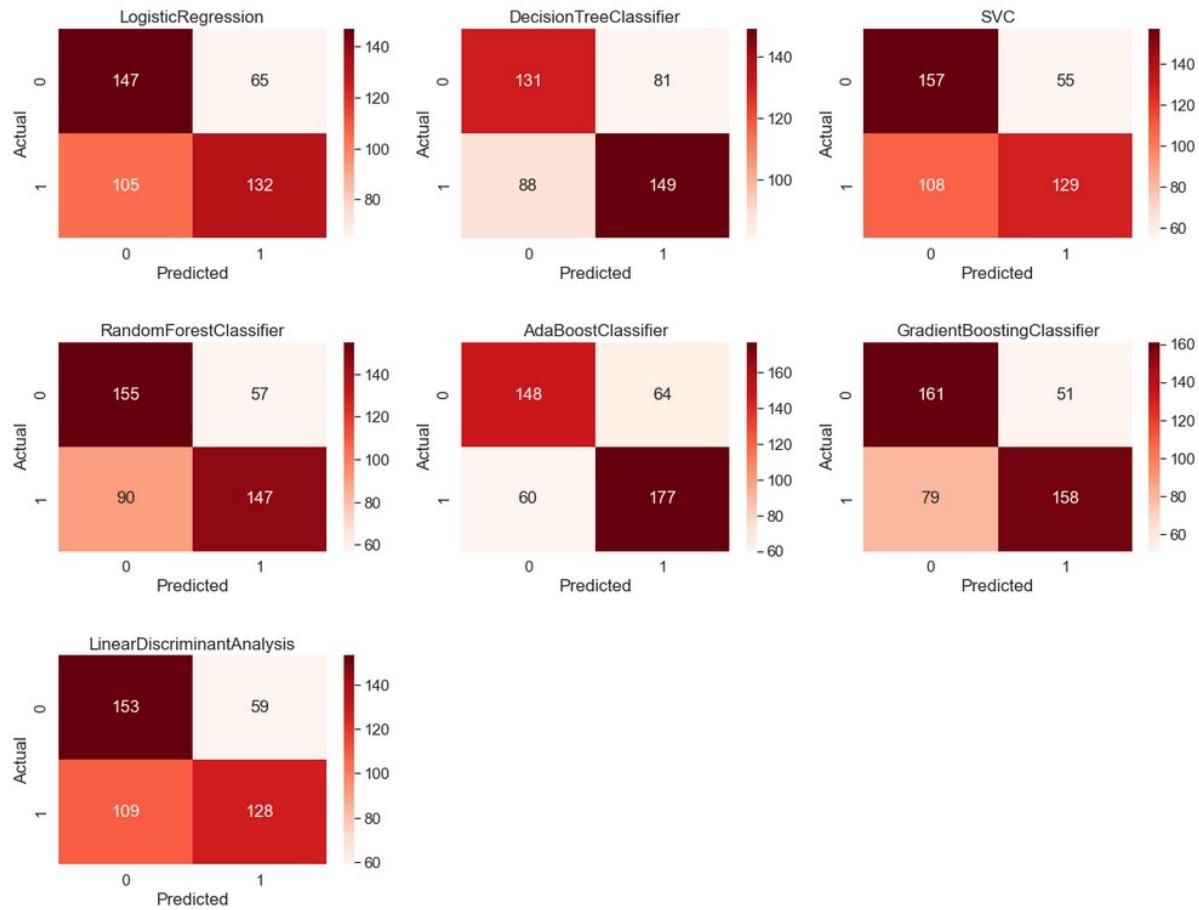


Figura 4.6: Matrici di confusione

Si osserva che il risultato migliore di accuratezza è stato ottenuto con il classificatore AdaBoost.

	Precision	Recall	F1-Score	Support
Family	0.71	0.70	0.70	212
Games	0.73	0.75	0.74	237
Accuracy			0.72	449
Macro Avg	0.72	0.72	0.72	449
Micro Avg	0.72	0.72	0.72	449

Si osserva che il classificatore ha un'affidabilità praticamente identica nel riconoscere gli elementi appartenenti alla categoria "FAMILY" e "GAMES", infatti misura un valore di F1Score nel primo caso pari a 0.70, mentre nel secondo caso pari a 0.74.

Un altro metodo che è stato utilizzato per valutare il modello ottenuto è mediante la **curva ROC**. Essa rappresenta un grafico che mostra le performance di un modello di classificazione considerando tutte le possibili soglie di classificazione. Tale curva rappresenta due parametri: *True Positive Rate* e *False Positive Rate*.

Nel caso del modello di classificazione ottenuto la ROC curve che si ottiene è rappresen-

4.2. CLASSIFICAZIONE CON GRID SEARCH

tata in Figura 4.7.

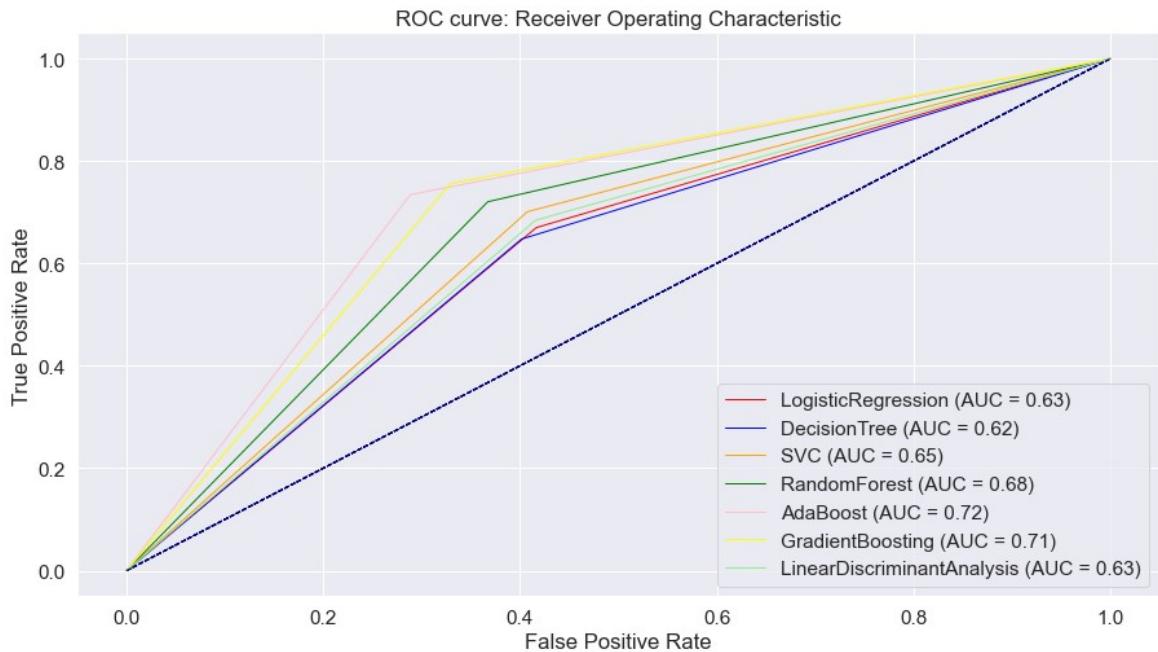


Figura 4.7: Curva ROC

4.2 Classificazione con Grid Search

La valutazione dei vari classificatori permette di definire una heatmap, nella quale si possono mostrare le correlazioni fra i modelli utilizzati. Quindi, tramite la tabella mostrata in Figura 4.8, si riesce a vedere quali classificatori restituiscono predizioni simili tra di loro.

Come si può notare i classificatori *LogisticRegression* e *LinearDiscriminantAnalysis* hanno correlazioni molto alte, quindi utilizzare uno piuttosto che l'altro non cambia troppo le predizioni che vengono fatte.

Mediante tale grafico è anche possibile individuare i modelli che verranno utilizzati nell'analisi successiva. Infatti, un' ulteriore modalità di classificazione è mediante la tecnica **Grid Search**. Essa è una tecnica che tenta di calcolare i valori ottimali degli iperparametri, mediante una ricerca esaustiva che viene eseguita sui valori dei parametri specifici di un modello. Alla luce della heatmap mostrata in Figura 4.8 si è deciso di utilizzare come classificatori il *DecisionTree* e il *GradientBoosting*, poichè fra loro c'è una bassa correlazione (0.42).

Gli iperparametri utilizzati sono mostrati in Figura 4.9

4.2. CLASSIFICAZIONE CON GRID SEARCH

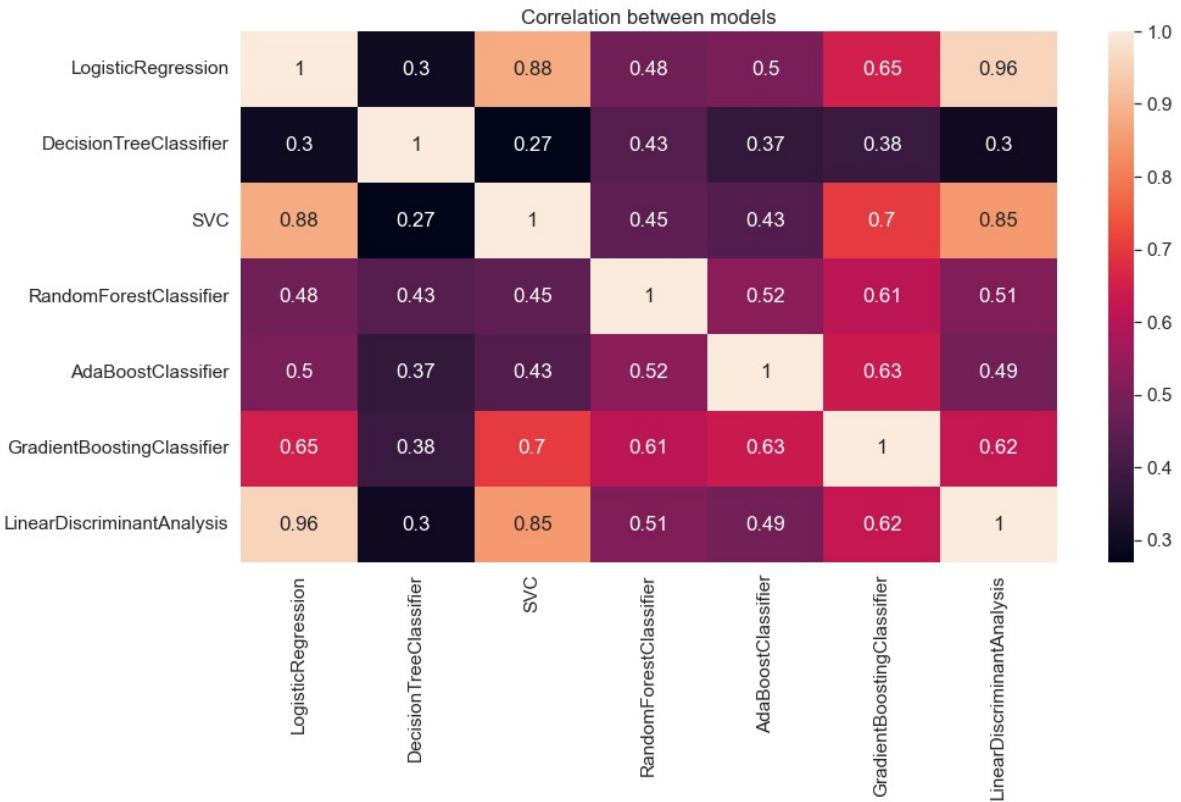


Figura 4.8: Correlazione fra i classificatori

```
DT_param = {"max_depth": [2,3,8,10],
            "max_features": [0.3, 0.7, 1],
            "min_samples_split": [2, 3, 10],
            "min_samples_leaf": [1, 3, 10],
            "criterion": ["gini"]}

GB_param = {'loss' : ["deviance"],
            'n_estimators': [100,200,300],
            'learning_rate': [0.1, 0.05, 0.01],
            'max_depth': [4, 8],
            'min_samples_leaf': [100,150],
            'max_features': [0.3, 0.1]}
```

Figura 4.9: Iperparametri GridSearch con DecisionTree e GradientBoosting

In questo caso si è deciso di effettuare una suddivisione del dataset bilanciato utilizzando la “k-cross fold validation”, operazione che consiste nella suddivisione dell’insieme di dati totale in un numero k di parti di uguale numerosità e, ad ogni passo, la k-esima parte dell’insieme di dati viene utilizzata come set di test, mentre la restante costituisce sempre l’insieme di addestramento. Successivamente tutti i risultati vengono mediati. Questa operazione permette di diminuire drasticamente le possibilità di incorrere in problemi di overfitting.

Eseguito l’addestramento sono stati valutati i modelli con e senza grid-search in termini di accuracy ed il risultato è stato il seguente:

4.2. CLASSIFICAZIONE CON GRID SEARCH

	DecisionTreeClassifier	GradientBoostingClassifier
Accuratezza senza GridSearchCV	0.63	0.678
Accuratezza con GridSearchCV	0.697	0.71

Si osserva che i modelli ottenuti sono migliori rispetto a quelli ottenuti effettuando un normale training, infatti in entrambi i casi l'accuratezza è prossima al 70%, che rappresenta il risultato migliore che si era ottenuto al primo addestramento.

Infine, come ultima analisi, si è deciso di effettuare un **ensemble** dei modelli. Esso consiste nell'andare a utilizzare più classificatori contemporaneamente per ottenere delle performance migliori rispetto ad utilizzare i singoli classificatori da soli. In questo caso, sempre alla luce di quanto fatto con la modalità di classificazione GridSearch, si è deciso di effettuare un ensemble dei classificatori DecisionTree e GradientBoosting. Il questo caso, il livello di accuratezza che si ottiene è 0.708, quindi prossimo a quello ottenuto con GridSearch, ma più affidabile siccome è stato ottenuto valutando più classificatori contemporaneamente.

Capitolo 5

Serie Temporale

In questo capitolo viene analizzato il tema delle serie temporali, i trend e le previsioni che possono essere realizzate con specifici algoritmi. Questo tipo di analisi sono state effettuate prevalentemente grazie alla libreria Statsmodels di Python. Statsmodel consente di esplorare dati, eseguire test statistici e stimare modelli statistici.

Il dataset utilizzato per questa sezione è quello relativo all'andamento temporale del prezzo delle azioni Google nel corso dell'ultimo ventennio circa. Il riferimento al dataset è esplicitato nell'introduzione al capitolo uno. Le colonne che sono state utilizzate per le analisi sono ovviamente la colonna 'Date' contenente il trading day di riferimento e la colonna 'Close' la quale si riferisce al prezzo di chiusura dell'ultima transazione alla data.

5.1 ETL

Prima di iniziare con le analisi vere e proprie è stata eseguita una breve fase di ETL. In particolare si è controllato (Fig. 5.1) se il dataset contenesse dei valori nulli che, nel caso, non dovrebbero essere ignorati.

```
#Check for null values
data.isnull().sum()

```

Date	0
Open	0
High	0
Low	0
Close	0
Adj Close	0
Volume	0
dtype:	int64

Figura 5.1: Valori nulli

Successivamente è stata manipolata la colonna 'Date' utilizzando la funzione `to_datetime` per convertire i valori nel tipo `datetime` e in seguito è stata applicata la funzione `set_index` così da trasformare il riferimento temporale nell'indice del dataset. Analizzando più nello specifico la colonna Date del dataset si può osservare che i trading day, cioè i giorni in

5.2. ANALISI DELLA SERIE

cui il mercato finanziario è aperto, sono cinque (sabato e domenica vengono esclusi per la chiusura del mercato). Questo si è visto essere problematico per le successive analisi e per la visualizzazione dei risultati perché crea dei salti temporali nella serie. Per risolvere questo problema è stato fatto quindi un **resample** della colonna con frequenza giornaliera ed è stata poi utilizzata la funzione **interpolate** (Fig. 5.2) con metodo *linear* in modo che venissero assegnati dei valori più o meno reali in corrispondenza dei salti temporali in base all'ultimo prezzo di chiusura. Facendo queste operazioni si ottiene la continuità temporale e la frequenza del dataset diventa di sette giorni.

Date		Date	
2004-08-20	53.952770	2004-08-20	53.952770
2004-08-23	54.495735	2004-08-21	54.133758
2004-08-24	52.239197	2004-08-22	54.314747
2004-08-25	52.802086	2004-08-23	54.495735
2004-08-26	53.753517	2004-08-24	52.239197

Figura 5.2: Dataset prima e dopo il resample e l'interpolate

5.2 Analisi della serie

Giunti a questo punto si è ottenuto il dataset che ora può essere analizzato a partire dal mostrare l'andamento generale dei prezzi di chiusura delle azioni nel corso degli anni attraverso un opportuno grafico (Fig. 5.3).

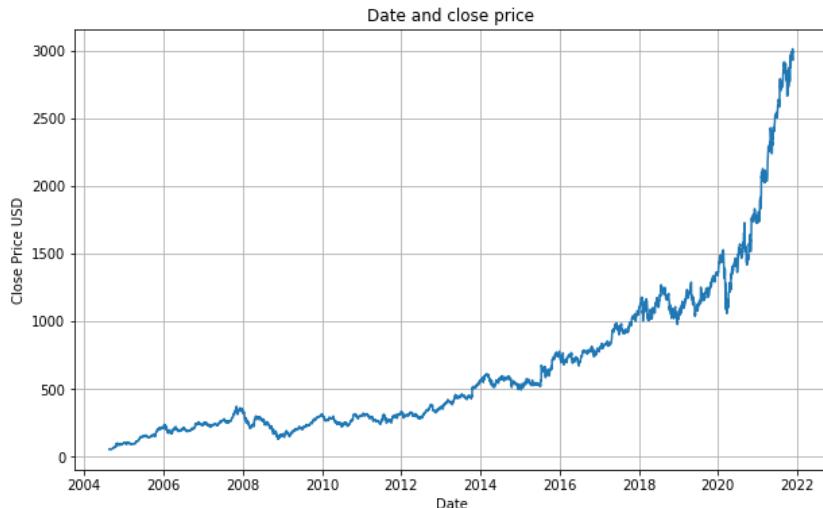


Figura 5.3: Grafico andamento dei prezzi di chiusura delle azioni Google

Fatto cioè si è scelto di analizzare circa gli ultimi due anni del dataset, più precisamente dal 2020-01-01 al 2021-11-23. Questo è stato fatto al solo scopo di ottenere una rappresentazione dei dati più efficace, i risultati ottenuti sono gli stessi anche prendendo una porzione più ampia del dataset.

5.2. ANALISI DELLA SERIE

5.2.1 Augmented Dickey Fuller Test

Ciò che è stato fatto ora è andare a testare se la serie è stazionaria o meno. La stazionarietà è una proprietà delle serie temporali che indica se i valori non dipendono dal tempo, in una serie stazionaria la media e la varianza devono essere costanti nel tempo. L'augmented dickey fuller test viene eseguito proprio per mostrare se la serie è stazionaria o meno così da poter poi differenziarla nella maniera più opportuna per ottenere il parametro di differenziazione necessario per la stima del modello di previsione. L'ipotesi nulla dell'ADF test è che la serie sia non stazionaria quindi se il valore **p-value < 0.05** allora si rifiuta l'ipotesi nulla e si può affermare che la serie è stazionaria. Nel caso della serie in analisi i risultati del test sono i seguenti:

- ADF statistics: 0.244793
- p-value: 0.974665

5.2.2 Autocorrelazione (ACF) e Autocorrelazione parziale (PACF)

Dai risultati ottenuti dal test si può affermare che la serie sia non stazionaria; si procede quindi con la differenziazione della serie partendo da un singolo ordine di differenziazione poichè una serie troppo differenziata potrebbe perdere di significato. Una volta che la serie è stata differenziata sono stati calcolati i grafici (Fig. 5.4) di autocorrelazione e autocorrelazione parziale.

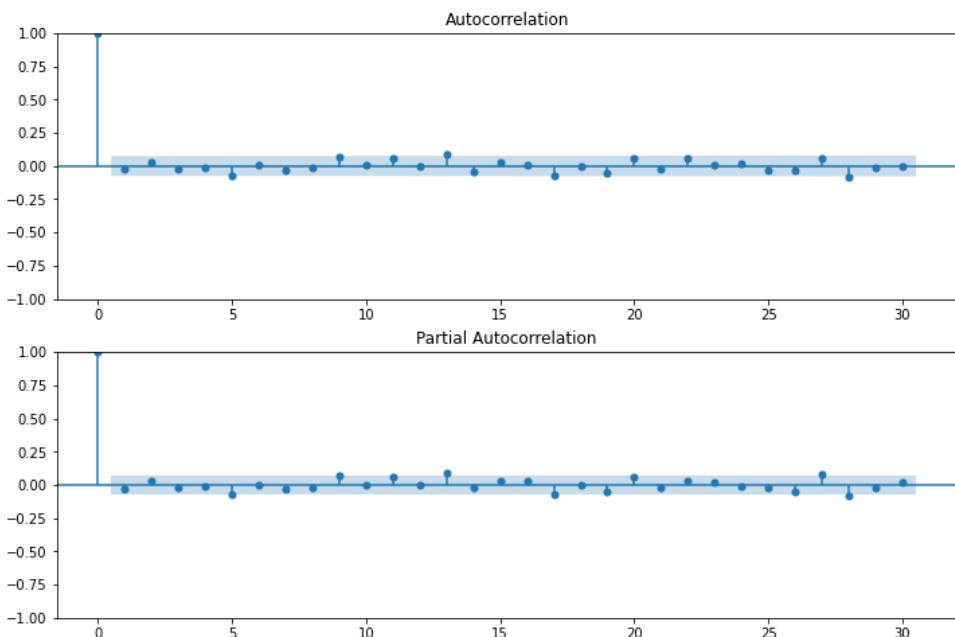


Figura 5.4: Grafici di autocorrelazione e autocorrelazione parziale

Questi grafici vengono utilizzati per la stima dei parametri del modello **ARIMA** con il quale è possibile prevedere una serie temporale attraverso i valori passati.

5.2. ANALISI DELLA SERIE

5.2.3 Creazione del modello ARIMA

Un modello ARIMA è caratterizzato da tre parametri:

- **p**: ordine del termine AR (Auto Regressive)
- **q**: ordine del termine MA (Moving Average)
- **d**: numero di differenziazioni necessarie per rendere la serie stazionaria

Il parametro **d** relativo al numero di differenziazioni per rendere la serie stazionaria è stato stimato pari a 1. Una volta effettuata la differenziazione è stato infatti ripetuto l'ADF test che ha evidenziato come la serie differenziata di un ordine sia stazionaria, di seguito i risultati ottenuti:

- ADF statistics: -26.937023
- p-value: 0.000000

Il parametro **p** si stima osservando il grafico di autocorrelazione parziale in base a quanti lag si trovano al di sopra del limite di significatività. In maniera analoga si stima il parametro **q** che indica quanti termini sono necessari per rimuovere qualsiasi autocorrelazione nella serie stazionaria; il grafico che si analizza, infatti, è quello dell'autocorrelazione. Essendo, purtroppo, la serie fortemente non stazionaria dai grafici di ACF e PACF non si riesce ad ottenere una stima corretta dei parametri p e q. Grazie, inoltre, ad una funzione messa a disposizione da statsmodel che è la **seasonal _decompose** si evidenziano (Fig. 5.5) le componenti trend e stagionalità che caratterizzano la serie.

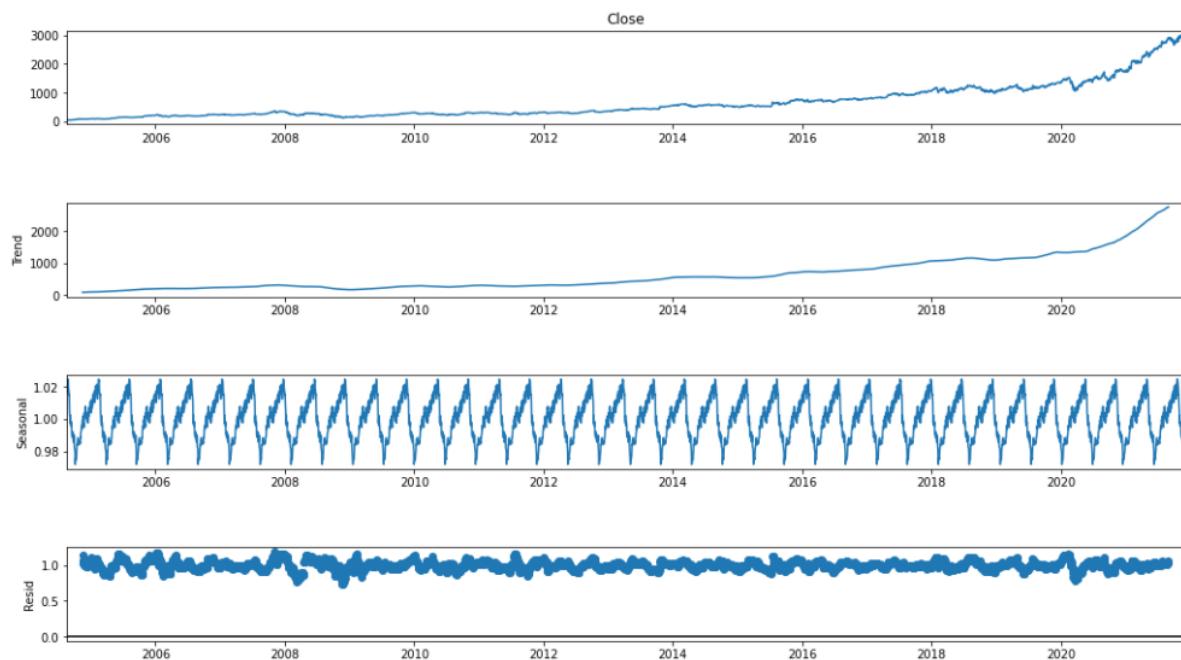


Figura 5.5: Applicazione della seasonal decompose alla serie

Il grafico mostra come la serie presenti una componente stagionale di cui bisogna tener

5.2. ANALISI DELLA SERIE

conto nella costruzione del modello ARIMA. Si vanno ad aggiungere quindi i parametri $(P, D, Q)m$ relativi alla parte stagionale dove m indica la periodicità della serie. Tale considerazione viene implementata in Python aggiungendo al modello il parametro `seasonal_order`. Riassumendo il modello ARIMA che si vuole ottenere è così formato:

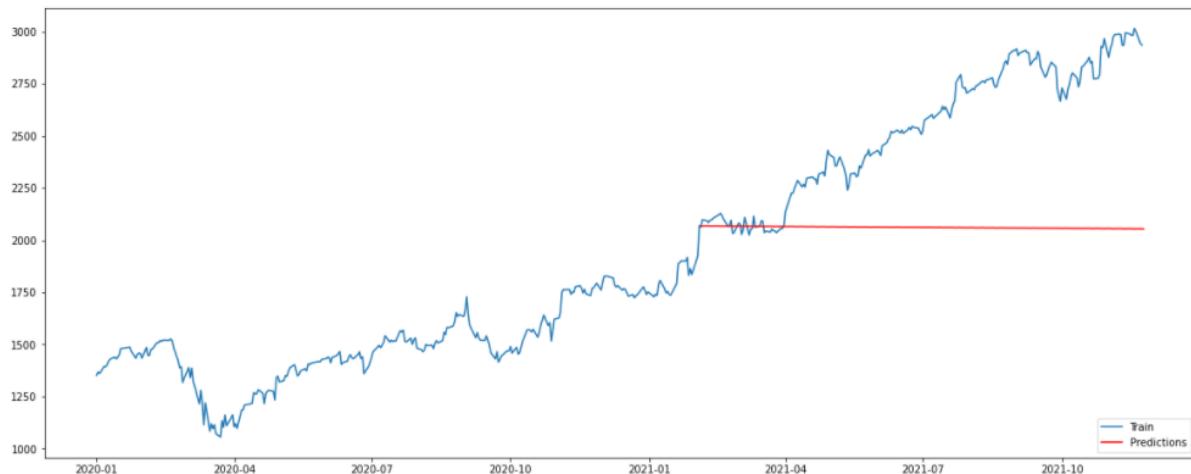
- ARIMA: $(p,d,q) \times (P,D,Q)m$

Vista la poca utilità dei grafici di autocorrelazione e autocorrelazione parziale è stata utilizzata un ulteriore libreria messa a disposizione dall'ambiente Python che è **pmdarima** con il metodo `auto_arima`. Tale funzione, tramite un procedimento euristico, permette di definire i parametri migliori per il modello in base al training set che gli viene dato in input. Il risultato ottenuto è stato il seguente:

- ARIMA: $(1,0,1) \times (1,0,1,7)$

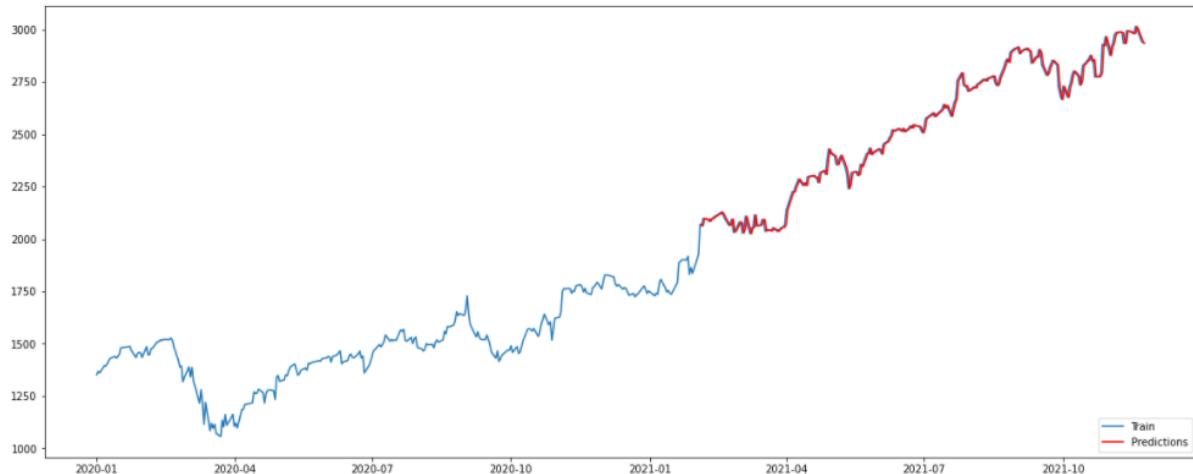
Si è, quindi, creato un modello utilizzando questi valori. Modello con il quale poi è stata calcolata la predizione all'interno del campione con il metodo `predict()`. Tale metodo di predizione ha un attributo chiamato **dynamic** che può essere settato a True o False. Nel primo caso il modello predice un passo avanti ($t+1$) e poi per la previsione del secondo passo ($t+2$), aggiunge il valore predetto ($t+1$) ai dati, riadatta il modello e fa la previsione dei nuovi dati. Nel secondo caso, invece, il modello predice sequenzialmente un passo avanti usando il valore precedente vero e non il valore predetto. In entrambe le casistiche i risultati non sono stati dei migliori:

- `dynamic = True` (previsione costante)



- `dynamic = False` (previsione che overfitta)

5.2. ANALISI DELLA SERIE



Facendo qualche tentativo nella modifica dei parametri si sono ottenuti dei risultati migliori con il seguente modello:

- ARIMA: $(1,1,1) \times (1,1,2,7)$

Si è ottenuto un notevole miglioramento nella predizione con l'attributo `dynamic = True` (Fig. 5.6) passando da una predizione piatta ad una che azzecca il trend in rialzo dei prezzi

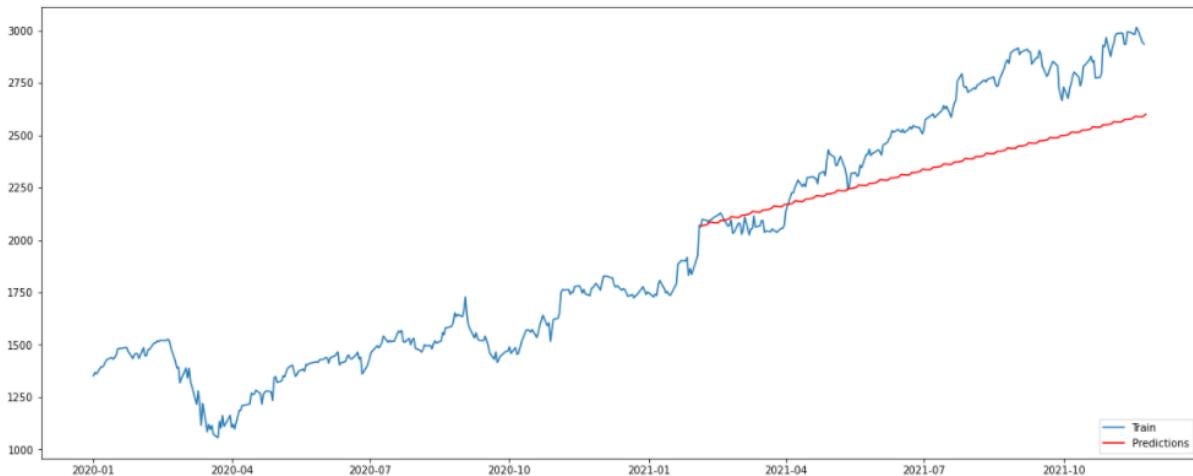


Figura 5.6: Predizione con parametri $(1,1,1) \times (1,1,2,7)$

5.2.4 Predizione fuori dal campione e metriche

L'analisi della serie temporale si conclude con la previsione dei valori fuori dal campione utilizzando la funzione `get_forecast` della libreria statsmodels. La linea tratteggiata nel grafico (Fig. 5.7) indica la previsione per i successivi trenta giorni a partire dalla data di fine del dataset. Per facilitare la visione dell'andamento della previsione è stato fatto uno zoom in modo da mostrare solo l'andamento degli ultimi due mesi circa. Infine vengono riportate alcune metriche statistiche per rendersi conto della bontà del modello e della previsione.

5.2. ANALISI DELLA SERIE

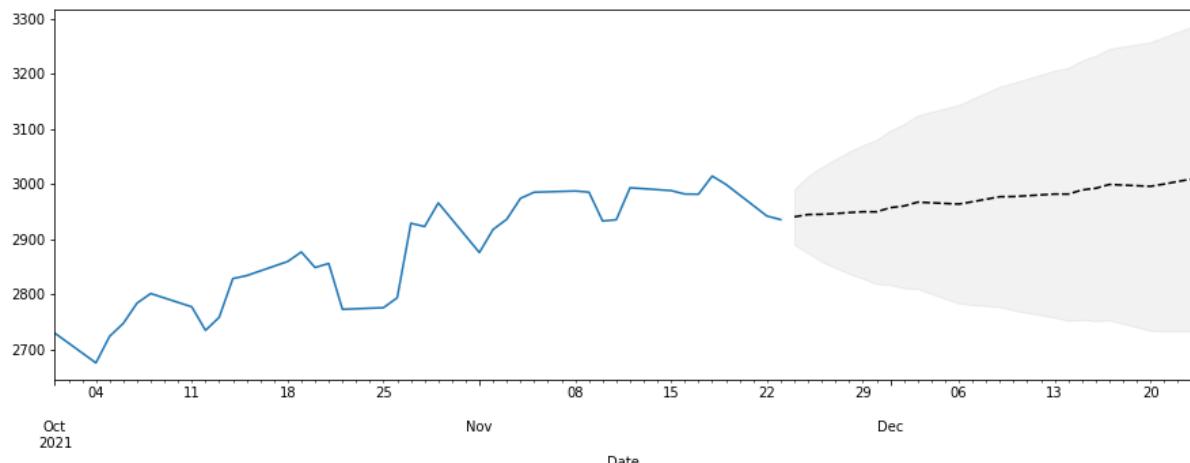


Figura 5.7: Predizione fuori dal campione

- MAPE: 8.21931715617697
- MAE: 220.8118313846531

Con un valore del MAPE pari all' 8,21% si ottiene che il modello è al 91,8% accurato nella predizione. Il MAE (Mean Absolute Error), invece, indica la media delle differenze assolute tra i valori predetti e i valori reali.