2023/
2024

Report Progetto Metodi Informatici per la gestione aziendale

PROGETTO INTERMEDIO

S.SANTIN1@CAMPUS.UNIMIB.IT

SANTIN SIMONE 886116

EXECUTIVE SUMMARY

L’obiettivo di questo progetto è creare un sistema di raccomandazione attraverso l’ausilio di tecniche di collaborative filtering, come K-NN e matrix factorization, e tecniche content-based usando K-NN basato su tecniche di embedding come BoW e transformers.

Come primo passo, nel progetto è stata effettuata un'analisi esplorativa del dataset delle recensioni su prodotti Amazon riguardanti CD e vinili. Questa analisi ha incluso uno studio sulla struttura del dataset e sulle distribuzioni delle recensioni e dei rating assegnati.

Successivamente, è stata trovata la configurazione ottimale dell’algoritmo K-NN per la previsione dei rating degli utenti, con il successivo riempimento della matrice di rating e la creazione di una lista di raccomandazioni per ogni utente. Questo processo ha portato a un RMSE decisamente buono.

Poi è stato eseguito l’algoritmo di clustering K-MEANS con cosine similarity per la segmentazione degli utenti in base alle loro recensioni, e successivamente è stato calcolato il valore di silhouette per valutare la qualità dei cluster, il quale ha portato ad un ottimo valore pari a 0.97 con un numero di cluster pari a 4.

In parallelo con il K-NN, è stata trovata la configurazione ottimale dell’algoritmo matrix factorization e creata la lista di raccomandazioni per ogni utente dopo aver riempito la matrice di rating, e i risultati ottenuti sono stati confrontati in termini di MSE e RMSE con il K-NN.

Nella seconda parte del progetto, gli attributi *title* e *description* dei diversi prodotti sono stati processati con tecniche di NLP. È stato quindi poi costruito un sistema di raccomandazione content-based, dopo aver effettuato l’embedding dei campi precedentemente preprocessati attraverso tecniche basate sulla frequenza (BoW) e tecniche basate su transformers.

In conclusione, l’obiettivo di questo progetto è stato raggiunto con successo, dato che le tecniche di embedding content-based con BoW e transformers hanno ottenuto risultati eccellenti con un RMSE di 0.89 e un MSE di 0.79. D'altra parte, le tecniche di collaborative filtering con KNN e matrix factorization hanno mostrato performance leggermente inferiori per il KNN, con un RMSE di 0.90 e un MSE di 0.81, mentre la matrix factorization ha raggiunto risultati migliori con un RMSE di 0.82 e un MSE di 0.68.

INTRODUZIONE AL PROBLEMA

Descrizione dei dati

Il dataset utilizzato per questa analisi proviene dalle recensioni su prodotti Amazon riguardanti i cd e vinili.

Ogni recensione è associata a un identificativo univoco dell'utente ('*user_id*'), un identificativo univoco del prodotto ('*parent_asin*'), e a un punteggio di valutazione da 1 a 5 ('*rating*'); inoltre i dati contengono altre informazioni come il timestamp ossia il momento in cui è stata lasciata la recensione, il titolo della recensione, il testo della recensioni, le immagini, i voti utili assegnati alla review e se l'acquisto dell'ordine a cui è stata lasciata la recensione è stato realmente effettuato dallo user; inoltre per aggiungere informazioni al dataset utili per le analisi, ho aggiunto nuove colonne tra cui la lunghezza della review in lettere, il numero di immagini, l'anno della recensione e il mese, in particolare le prime due potrebbero risultare utili nel cercare una correlazione tra un rating basso e una recensione lunga o voti utili alti e recensione lunga (o anche con un gran numero di immagini)

Il dataset contiene circa 4 milioni di entries, i quali sono strutturati nel seguente modo:

	rating	timestamp	helpful_vote
count	4.827273e+06	4827273	4.827273e+06
mean	4.501898e+00	2013-07-15 16:48:38.224111872	1.908382e+00
min	1.000000e+00	1997-09-09 03:13:17	0.000000e+00
25%	4.000000e+00	2009-08-15 00:17:22	0.000000e+00
50%	5.000000e+00	2014-11-28 18:15:38	0.000000e+00
75%	5.000000e+00	2017-07-20 23:42:19.375000064	2.000000e+00
max	5.000000e+00	2023-09-09 15:33:39.315000	2.172000e+03
std	1.004442e+00	NaN	6.879886e+00

	year	month	review_length	num_images
count	4.827273e+06	4.827273e+06	4.827273e+06	4.827273e+06
mean	2.013048e+03	6.391952e+00	7.554436e+01	1.426748e-02
min	1.997000e+03	1.000000e+00	0.000000e+00	0.000000e+00
25%	2.009000e+03	3.000000e+00	9.000000e+00	0.000000e+00
50%	2.014000e+03	6.000000e+00	2.900000e+01	0.000000e+00
75%	2.017000e+03	1.000000e+01	8.500000e+01	0.000000e+00
max	2.023000e+03	1.200000e+01	6.515000e+03	1.600000e+02
std	5.810376e+00	3.550999e+00	1.316712e+02	2.292203e-01

Invece di seguito vi è una serie di informazioni sui dati, tra cui il tipo e la somma dei valori nulli, la quale è zero per ogni colonna e le prime 5 righe del dataset come esempio:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4827273 entries, 0 to 4827272
Data columns (total 14 columns):
 #   Column             Dtype
---  ------             -----
 0   rating             int64
 1   title              object
 2   text               object
 3   images             object
 4   asin               object
 5   parent_asin        object
 6   user_id            object
 7   timestamp          datetime64[ns]
 8   helpful_vote       int64
 9   verified_purchase  bool
 10  year               int32
 11  month              int32
 12  review_length      int64
 13  num_images         int64
dtypes: bool(1), datetime64[ns](1), int32(2), int64(4), object(6)
memory usage: 446.6+ MB
```

```
rating             0
title              0
text               0
images             0
asin               0
parent_asin        0
user_id            0
timestamp          0
helpful_vote       0
verified_purchase  0
year               0
month              0
review_length      0
num_images         0
dtype: int64
```

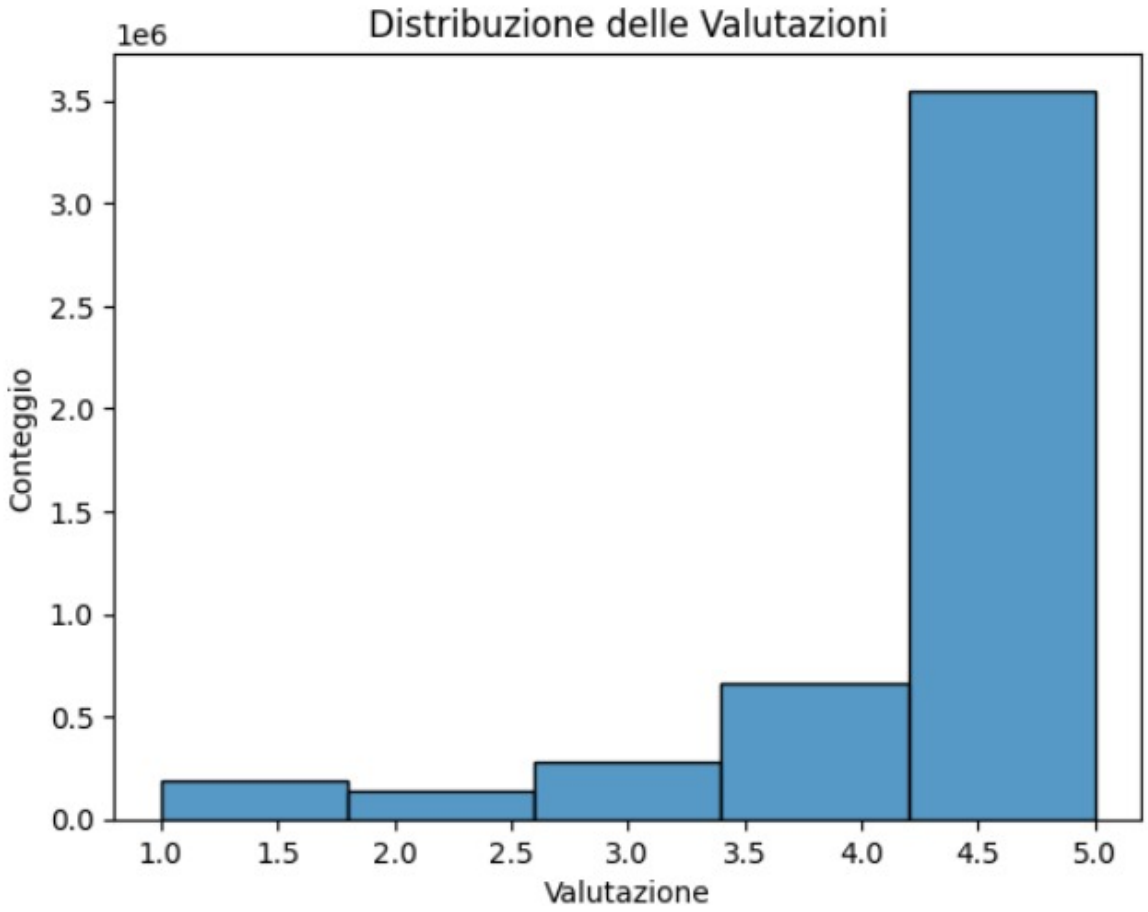
	rating	title	text	images	asin	parent_asin	user_id	timestamp	helpful_vote	verified_purchase	year	month
0	5	Five Stars	LOVE IT!	[]	B002MW50JA	B002MW50JA	AGKASBHYZPGTEPO6LWZPVJWB2BVA	2016-01-13 02:06:17.000	0	True	2016	1
1	5	Five Stars	LOVE!!	[]	B008XNPN0S	B008XNPN0S	AGKASBHYZPGTEPO6LWZPVJWB2BVA	2016-01-13 02:06:04.000	0	True	2016	1
2	3	Three Stars	Sad there is not the versions with the real/or...	[]	B00IKM5N02	B00IKM5N02	AGKASBHYZPGTEPO6LWZPVJWB2BVA	2016-01-13 01:51:25.000	0	True	2016	1
3	3	Disappointed	I have listen to The Broadway 1958 Flower Drum...	[]	B00006JKCM	B00006JKCM	AEVWAM3YWN5URJVJIZZ6XPD2MKIA	2006-11-20 15:34:24.000	3	True	2006	11
4	5	Wonderful melding	Simply great album. One of the best. Marvelous...	[]	B00013YRQY	B00013YRQY	AFWHJ6O3PV4JC7PVOJH6CPULO2KQ	2020-02-19 05:29:59.946	0	False	2020	2

Obiettivi dell’analisi

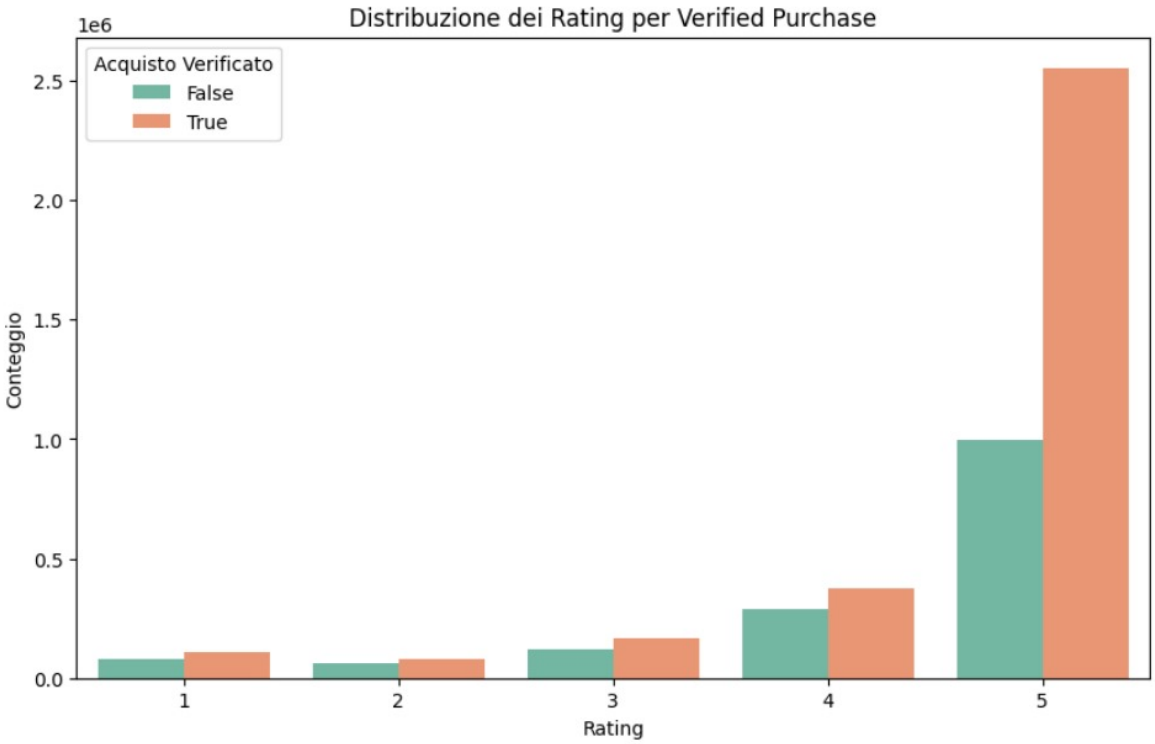
L'obiettivo principale di questa analisi è comprendere i modelli di comportamento degli utenti e la qualità dei prodotti attraverso tecniche di machine learning, in modo da personalizzare le raccomandazioni di prodotti per gli utenti, individuare utenti con comportamenti anomali (ad esempio, utenti che lasciano recensioni estremamente negative o positive in modo sproporzionato, oppure utenti che lasciano recensioni senza aver effettuato effettivamente l’acquisto), e analizzare le statistiche dei prodotti come andamento distribuzione dei rating oppure correlazione tra rating bassi/alti dati da utenti con acquisti non verificati.

Risultati dell’analisi esplorativa

Come prima analisi ho voluto effettuare un controllo sulla distribuzione delle valutazioni tramite un istogramma:

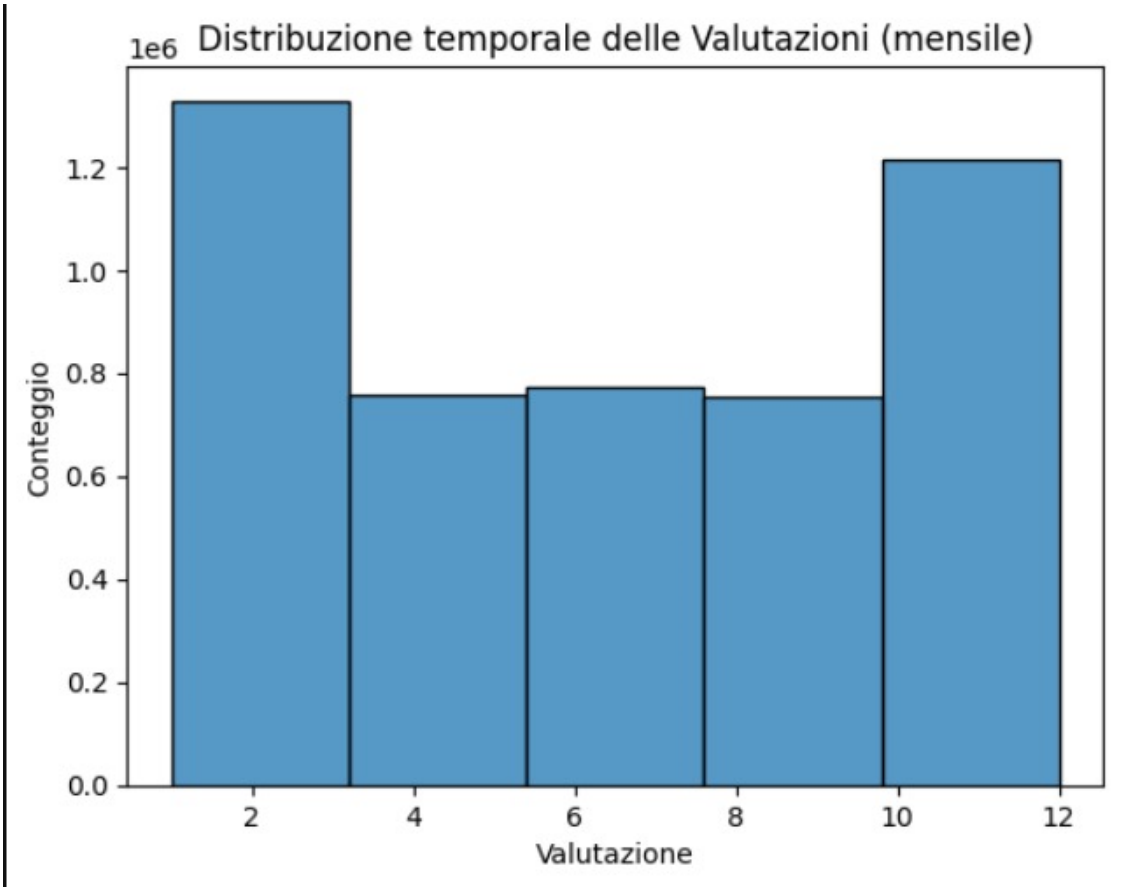


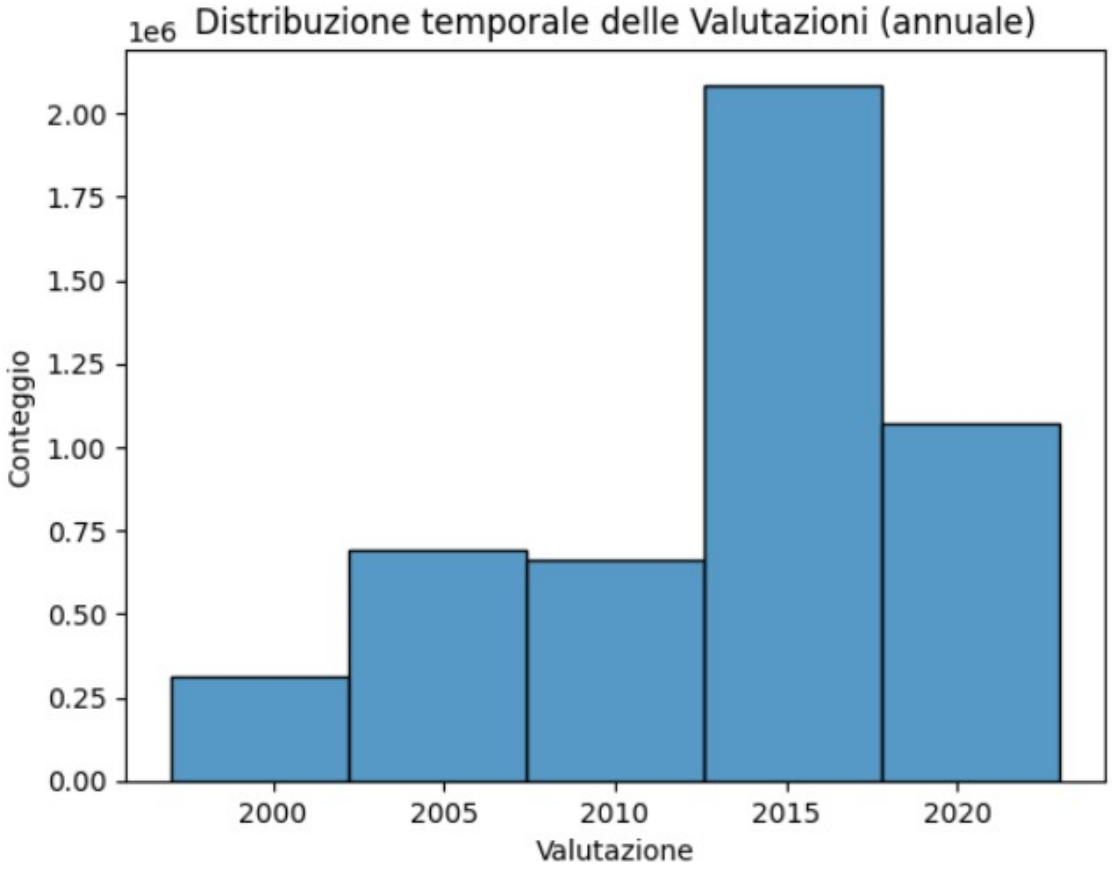
Dove si può notare che la maggior parte dei rating ha una votazione alta; tuttavia, ho voluto ricontrollare la distribuzione di tali valutazioni però sfruttando questa volta un *countplot* tramite il campo *verified_purchase* in modo da controllare la quantità di valutazioni false:



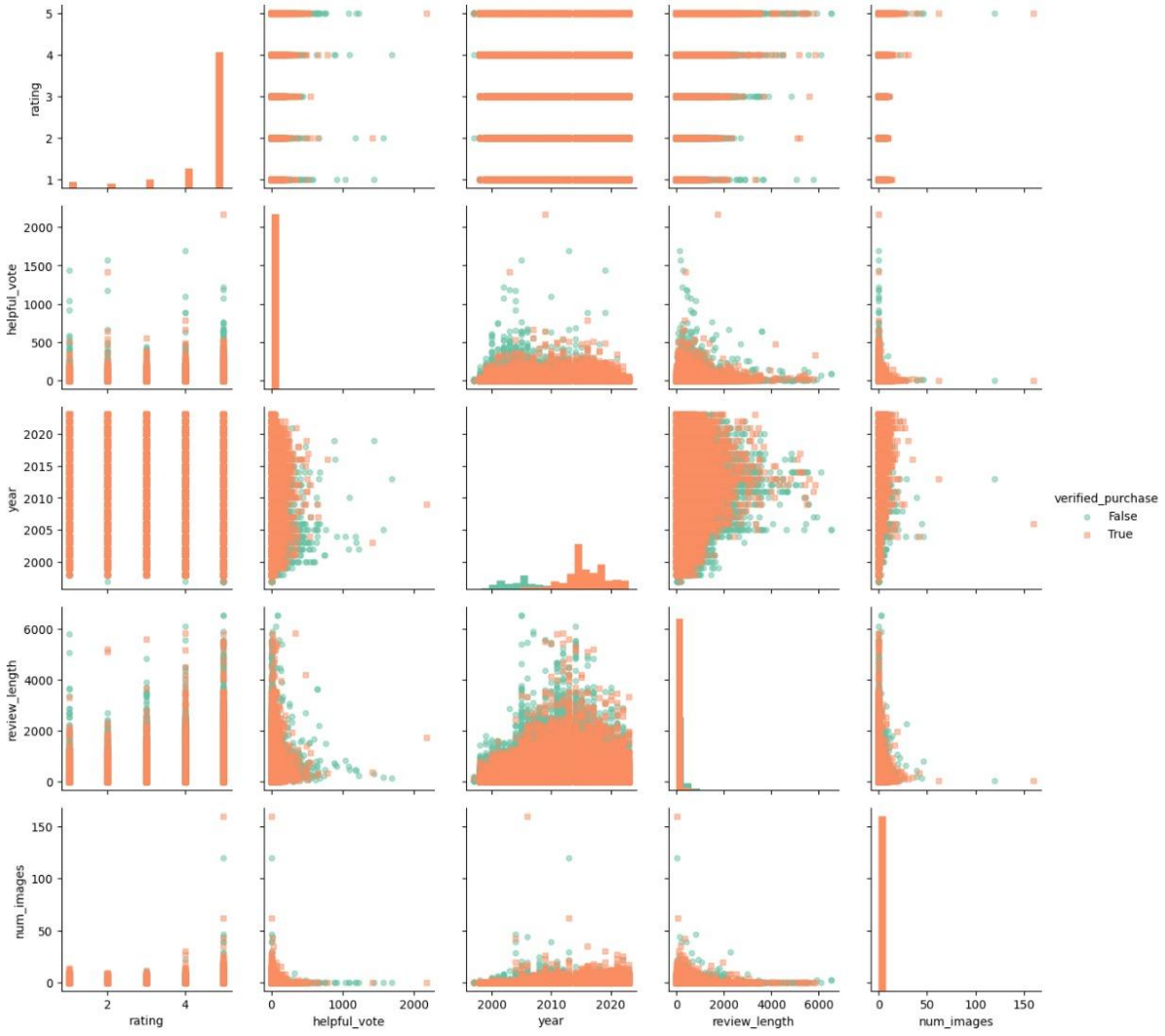
Dal quale si nota che la maggior parte delle valutazioni falsate si trovano tra i rating più alti, seppur in bassa quantità rispetto al numero complessivo di rating reali. Questo fenomeno suggerisce che le recensioni meno autentiche tendono a gonfiare le valutazioni positive, ma fortunatamente rappresentano solo una piccola parte rispetto alle numerose recensioni reali.

Ho deciso di esaminare anche la distribuzione temporale dei rating utilizzando degli istogrammi, includendo un controllo sia per il mese degli ordini che per l'anno di acquisto. Dai grafici emerge un picco degli ordini tra la fine di novembre e gennaio, probabilmente correlato al Black Friday e alle festività. Inoltre, osservando la distribuzione annuale, si nota un aumento costante degli ordini, riflettendo la crescente popolarità di Amazon, con un picco evidente nel 2015


Distribuzione temporale delle Valutazioni (mensile)
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Conteggio
2
4
6
8
10
12
Valutazione


Distribuzione temporale delle Valutazioni (annuale)
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Conteggio
2000
2005
2010
2015
2020
Valutazione

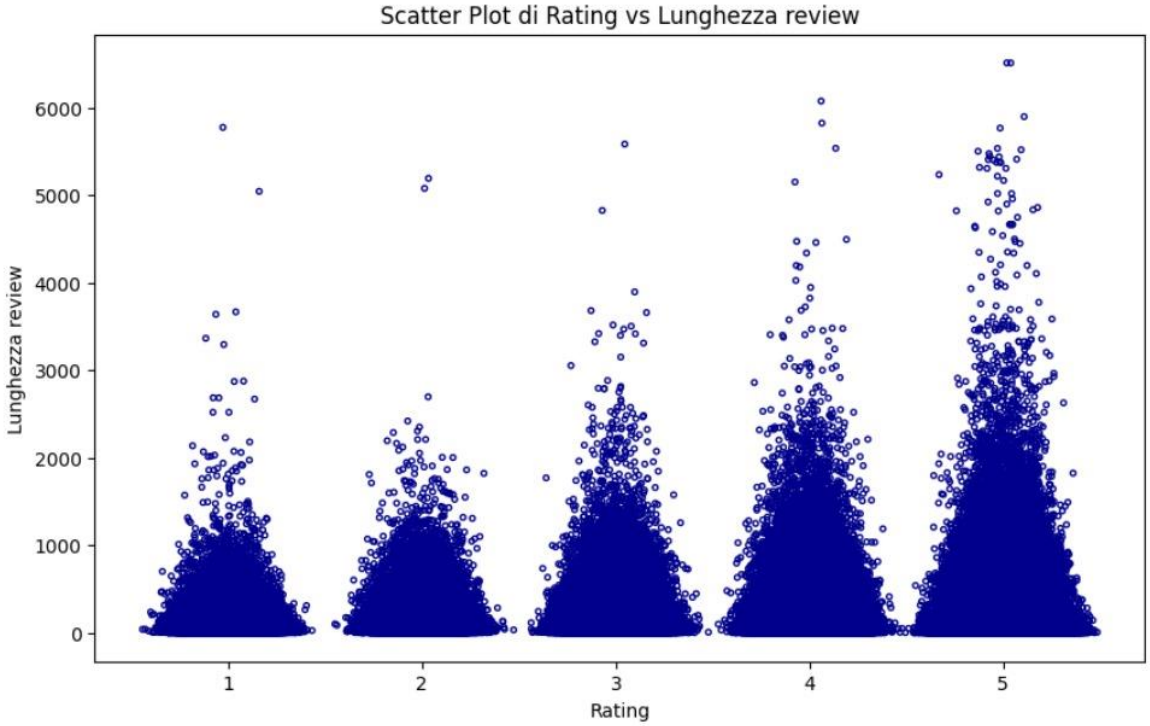
Successivamente ho voluto proseguire con una *PairGrid* sui campi [*rating, helpful_vote, year, month, review_length, num_images, verified_purchase*] in modo da avere una visione generale delle relazioni tra tali variabili e sfruttando inoltre il campo *verified_purchase* dando così una colorazioni diversa alle entries con acquisti non verificati:



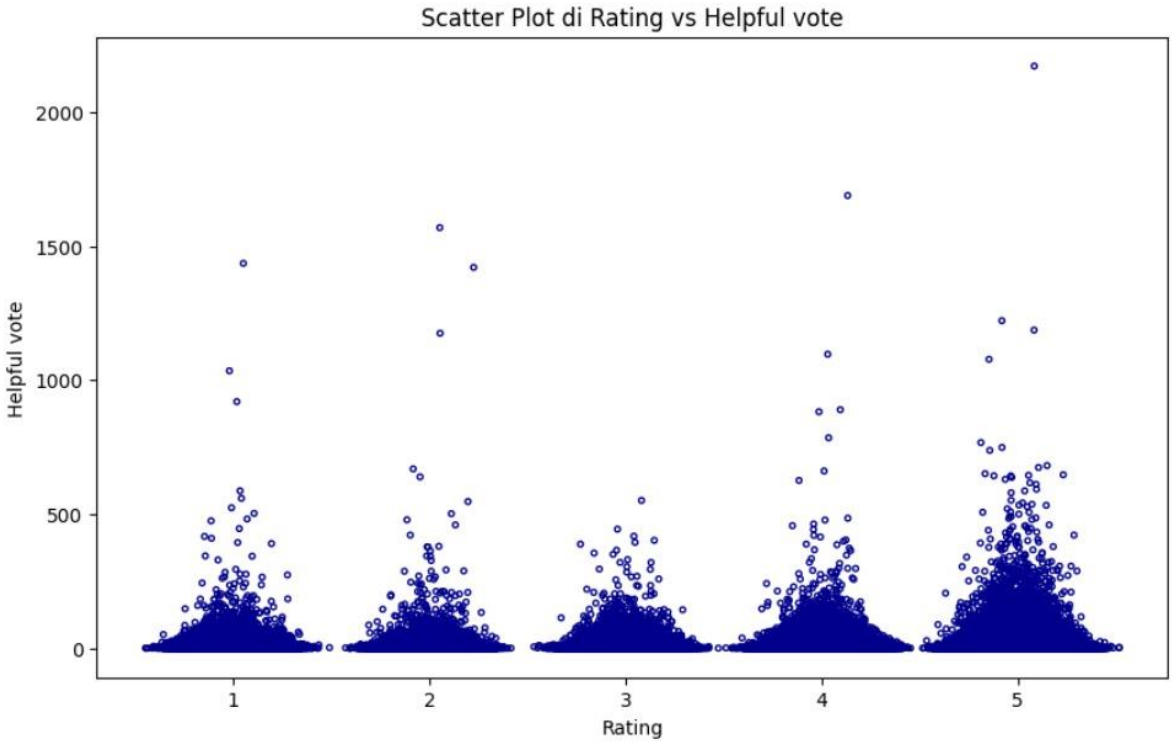
Uno dei primi dati evidenti è che la maggior parte delle recensioni, e quindi presumibilmente degli ordini, risale a partire dall'anno 2012 circa. Prima di tale periodo, la maggior parte dei rating è stata fornita da utenti con acquisti non verificati. Inoltre, si osserva che le recensioni più lunghe e con valutazioni più basse provengono principalmente da utenti con acquisti non verificati, suggerendo potenzialmente recensioni non autentiche

Tuttavia, questo grafico potrebbe risultare confusionario e poco chiaro per alcune relazioni. Pertanto, ho deciso di creare diversi *scatterplot* tra le colonne più significative per individuare eventuali correlazioni. Ho utilizzato una scala logaritmica o il Jitter per aggiungere un po' di rumore ai dati e rendere i grafici più comprensibili e leggibili.

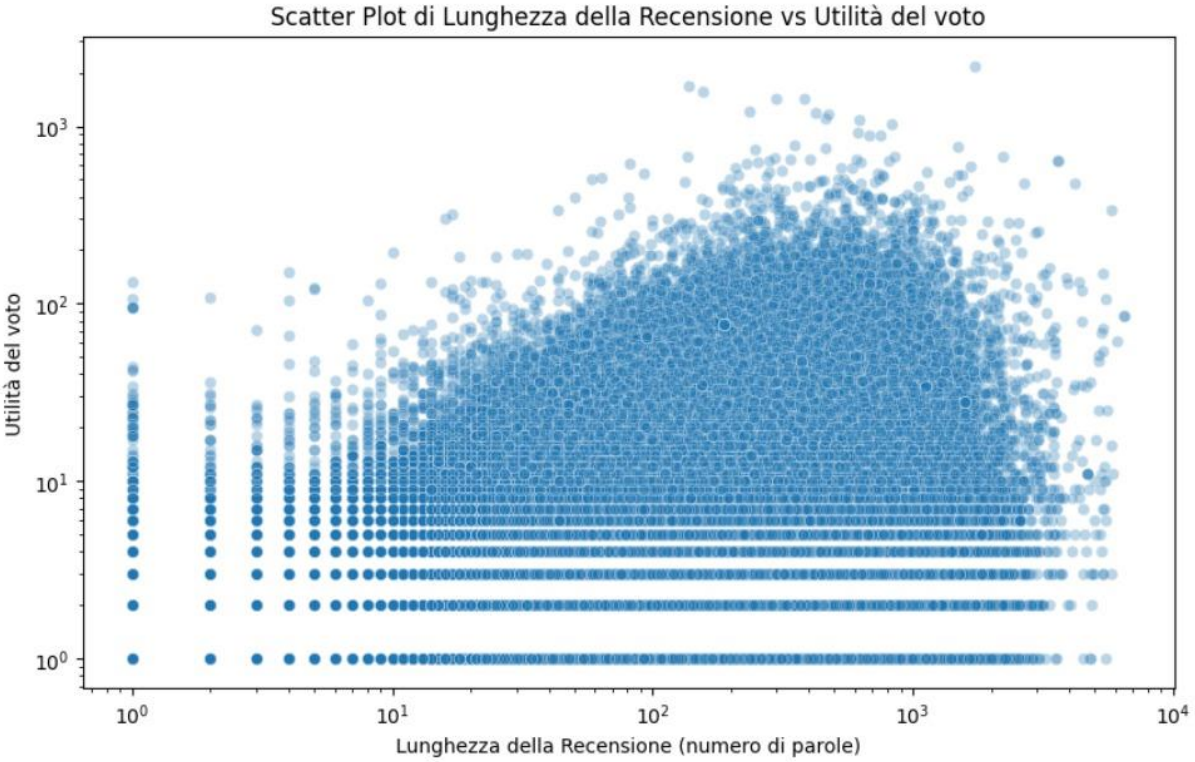
Come primo passo, ho esaminato le relazioni tra i rating e la lunghezza delle recensioni. Da questa analisi è emerso che esiste una tendenza all'aumento della lunghezza delle recensioni all'aumentare del rating, questo quindi suggerisce che gli utenti che attribuiscono valutazioni più alte tendono a scrivere recensioni più dettagliate e lunghe.



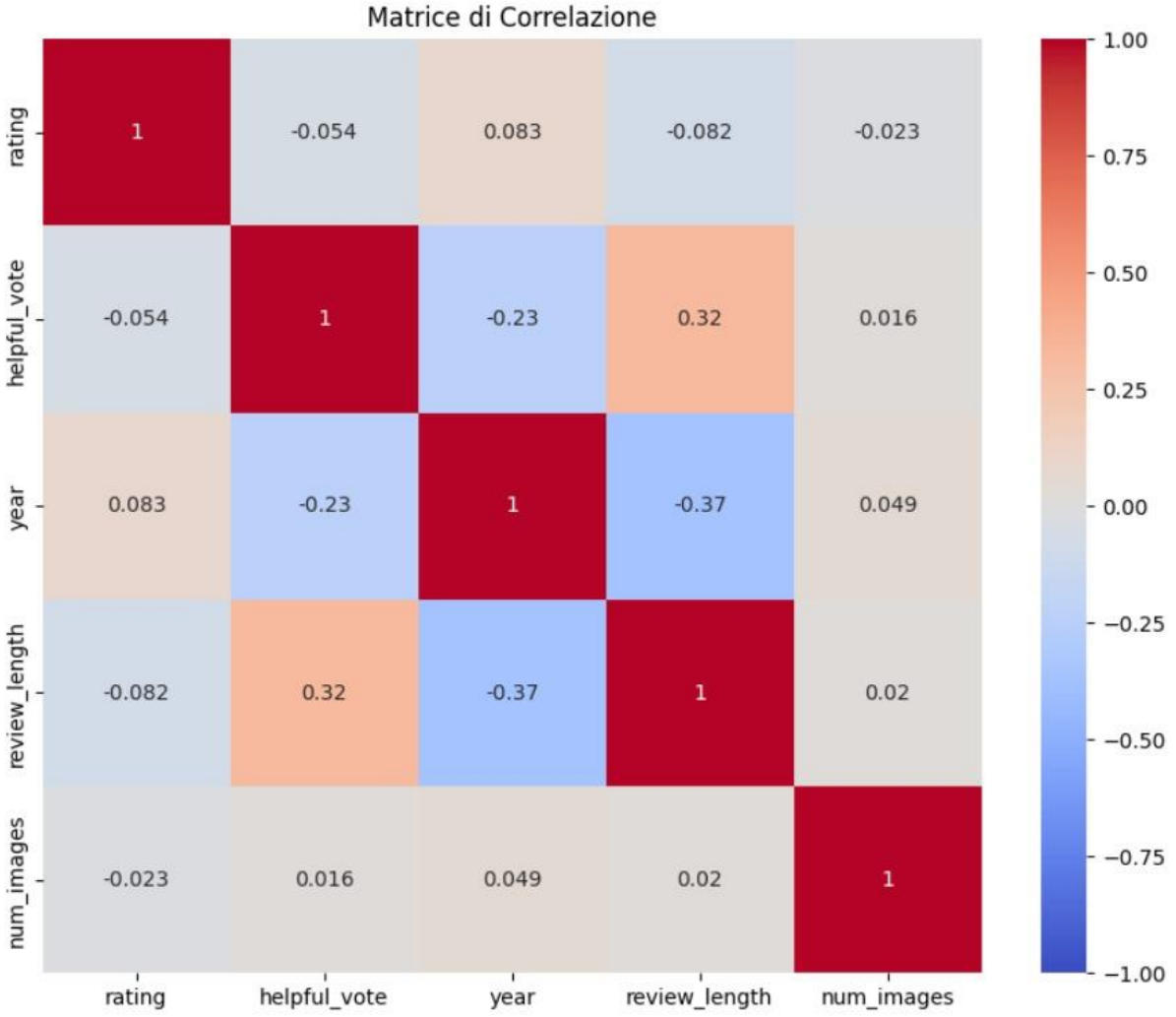
Successivamente, ho continuato con un'analisi simile alla precedente, ma questa volta ho esaminato la relazione tra i rating e l'utilità delle recensioni, misurata attraverso i voti di utilità attribuiti a ciascun rating. È emerso un leggero aumento dei voti di utilità all'aumentare del rating, cioè le recensioni che ricevono valutazioni più alte tendono ad essere considerate più utili dagli altri utenti. Tuttavia, si osserva che la quantità di voti rimane minima, con poche differenze evidenti tranne per i rating più alti.



Infine, ho voluto esaminare la relazione tra l'utilità del voto e la lunghezza della recensione, utilizzando una scala logaritmica per evidenziare i punti. È interessante notare come all'aumentare della lunghezza della recensione aumenti gradualmente anche il numero di voti utili assegnati al rating.



Come ultimo grafico vi è infine una *heatmap* tra le colonne piu importanti del dataset per notare una correlazione tra di esse, quindi *rating, helpful_vote, year, review_length, num_images* :



Dalla heatmap emerge in modo evidente che c'è una buona correlazione positiva tra la lunghezza della recensione e il numero di voti utili assegnati alla stessa. Questo suggerisce che recensioni più lunghe tendono ad attrarre più attenzione e apprezzamento da parte degli utenti, indicando una possibile relazione tra la qualità o la completezza delle recensioni e l'interesse generato. Mentre per quanto riguarda le altre variabili, non si nota alcuna correlazione spiccare tra di esse.

Risultati step

Analisi esplorativa

L'analisi esplorativa del dataset rivela diversi aspetti significativi. Innanzitutto, vi è una presenza considerevole di recensioni associate a valutazioni più elevate. Tuttavia, è importante notare che una parte significativa di queste recensioni potrebbe essere falsata, poiché sono state lasciate da utenti con acquisti non verificati.

Dal punto di vista temporale invece, la distribuzione delle recensioni mostra un picco significativo tra i mesi di Novembre e Gennaio, con un volume particolarmente elevato di recensioni nel 2015. Questo potrebbe essere attribuito a una serie di fattori, come le festività natalizie o eventi di marketing come il Black Friday nel mese di Novembre, che potrebbero aver influenzato il comportamento degli utenti nel lasciare recensioni.

Per quanto riguarda la distribuzione dei voti utili lasciati alle recensioni si nota una tendenza quasi lineare, con dei lievi aumenti all'aumentare del rating. Inoltre, vi è comunque una certa correlazione tra i voti utili e la lunghezza delle recensioni. Questo suggerisce che le recensioni più dettagliate o approfondite potrebbero ricevere un maggiore coinvolgimento da parte degli utenti, le quali soprattutto mostrano un aumentare della lunghezza all'aumentare del voto di rating.

Infine, il dataset si presenta estremamente vasto, con un totale di 4 milioni di voci e con nessuna colonna contenente valori nulli. Questa vasta copertura dei dati offre un'eccellente base per l'analisi e la modellazione, consentendo una panoramica completa e dettagliata del comportamento degli utenti e delle dinamiche delle recensioni.

Configurazione ottimale K-NN

Per trovare la configurazione ottimale del K-NN, prima di caricare il dataset in un *dataset surprise*, ho effettuato una pulizia approfondita del dataset. Ho eliminato gli utenti che avevano lasciato meno di dieci recensioni e gli articoli che avevano ricevuto meno di dieci recensioni. Per garantire che dopo la pulizia non rimanessero utenti o articoli con meno di dieci recensioni, ho ripetuto questa operazione in un ciclo iterativo fino a quando ogni utente e ogni articolo avessero almeno dieci recensioni. Questo processo è stato necessario per

assicurarsi che, durante il calcolo del K-NN, ogni elemento avesse un numero sufficiente di vicini, evitando così operazioni imprecise o distorte.

```python
import pandas as pd

def filter_data(ratings, user_threshold=10, item_threshold=10):
    while True:

        user_inter_count = ratings.groupby("user_id").count()[["parent_asin"]].reset_index()
        users_id = user_inter_count[user_inter_count["parent_asin"] > user_threshold]["user_id"].tolist()
        ratings = ratings[ratings["user_id"].isin(users_id)]


        item_inter_count = ratings.groupby("parent_asin").count()[["user_id"]].reset_index()
        items_id = item_inter_count[item_inter_count["user_id"] > item_threshold]["parent_asin"].tolist()
        ratings = ratings[ratings["parent_asin"].isin(items_id)]


        new_user_inter_count = ratings.groupby("user_id").count()[["parent_asin"]].reset_index()
        new_item_inter_count = ratings.groupby("parent_asin").count()[["user_id"]].reset_index()

        if all(new_user_inter_count["parent_asin"] > user_threshold) and all(new_item_inter_count["user_id"] > item_threshold):
            break

    return ratings


ratings = filter_data(amazon_sales)
print("N. Users:", len(ratings["user_id"].unique()))
print("N. Items:", len(ratings["parent_asin"].unique()))
print("N. Ratings:", len(ratings))
```

```
N. Users: 16944
N. Items: 15903
N. Ratings: 433645
```

Come risultato di questa pulizia, il numero di rating è diminuito da 4 milioni a circa 400.000. Questo ridimensionamento non solo garantisce che ogni utente e articolo abbia un numero sufficiente di recensioni per una corretta applicazione del K-NN, ma consente anche di eseguire le operazioni a una velocità significativamente maggiore. La riduzione del volume dei dati rende il processo di calcolo più efficiente e gestibile, migliorando così le prestazioni complessive del modello.

Successivamente, per trovare la configurazione ottimale dell'algoritmo K-NN, ho deciso di suddividere il problema in più fasi. Prima di tutto, ho applicato il metodo dell'elbow per identificare i tre parametri che restituiscono il miglior valore di RMSE sul dataset. Questo processo ha comportato il calcolo di ogni valore di k a partire da 5 fino a 41, incrementando di due unità a ogni iterazione. Utilizzando questa gamma di valori di k, ho potuto individuare quelli che minimizzano l'errore quadratico medio, facilitando così la scelta dei parametri più efficaci per il modello K-NN.

```python
from surprise.model_selection import cross_validate
from surprise import KNNBasic
mse_over_k, rmse_over_k = [], []
for k in np.arange(5, 41, 2):
    print(f'Trying k={k}...')
    algo = KNNBasic(k=k, verbose=False)
    k_fold_result = cross_validate(algo, dataset_surprise, cv=10,
                                   measures=['mse', 'rmse'],
                                   verbose=False)
    mse_over_k.append(k_fold_result['test_mse'].mean())
    rmse_over_k.append(k_fold_result['test_rmse'].mean())
ks = np.arange(5, 41, 2)
min_rmse_indices = np.argsort(rmse_over_k)[:3]
bestK = [ks[i] for i in min_rmse_indices]
print("I migliori valori di K corrispondenti ai minimi RMSE sono:", bestK)
```

```
Trying k=5...
Trying k=7...
Trying k=9...
Trying k=11...
Trying k=13...
Trying k=15...
Trying k=17...
Trying k=19...
Trying k=21...
Trying k=23...
Trying k=25...
Trying k=27...
Trying k=29...
Trying k=31...
Trying k=33...
Trying k=35...
Trying k=37...
Trying k=39...
I migliori valori di K corrispondenti ai minimi RMSE sono: [27, 15, 17]
```

Dopo aver individuato i valori ottimali di k utilizzando il metodo dell'elbow, ho proseguito con una GridSearch. Questa ricerca è stata applicata sui tre migliori valori k e su una vasta griglia di parametri, allo scopo di trovare la configurazione ottimale per addestrare il modello. Questo processo di ottimizzazione, il quale ha incluso la valutazione di vari parametri come il numero di vicini, le metriche di distanza e altri iperparametri, ha permesso di ottenere un RMSE di 0.8881 e un MSE di 0.78876.

```python
from surprise.model_selection import GridSearchCV
from surprise import accuracy

param_grid = {
    'k': [27, 15, 17],
    'sim_options': {
        'name': ['cosine', 'msd', 'pearson'],
        'user_based': [True, False]
    },
}

gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mse'], cv=10, n_jobs=1)
gs.fit(dataset_surprise)

best_mse = gs.best_score['mse']
best_rmse = gs.best_score['rmse']
print(f'Best MSE = {best_mse:.4f}')
print(f'Best RMSE = {best_rmse:.4f}')
print(f'Best configuration: {gs.best_params["rmse"]}')
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Best MSE = 0.7887
Best RMSE = 0.8881
Best configuration: {'k': 27, 'sim_options': {'name': 'msd', 'user_based': False}}
```

Dopo aver trovato la combinazione ottimale di parametri, l'ho applicata al dataset in modo da riuscire ad avere un modello finale. Ho suddiviso il dataset in un set di addestramento (trainset) e un set di test (testset), assegnandogli il 20% dei dati. L'applicazione del modello ottimizzato sul testset ha portato a un RMSE di 0.90 e un MSE di 0.81, i quali essendo buoni valori dimostrano che il modello è capace di fare previsioni precise.

```python
from surprise import KNNBasic, Dataset, Reader, accuracy
import numpy as np
from surprise import Dataset, Reader, accuracy
from surprise.model_selection import train_test_split

data = Dataset.load_from_df(ratings[['user_id', 'parent_asin', 'rating']], reader)
trainset, testset = train_test_split(data, test_size=0.2)


best_k = 27
best_sim_options = {
    'name': 'msd',
    'user_based': False
}

final_model = KNNBasic(k=best_k, sim_options=best_sim_options)
final_model.fit(trainset)
predictions = final_model.test(testset)
rmse = accuracy.rmse(predictions)
mse = accuracy.mse(predictions)
print(f'MSE sul set di test: {mse}')
print(f'RMSE sul set di test: {rmse}')
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9035
MSE: 0.8162
MSE sul set di test: 0.8162288722106912
RMSE sul set di test: 0.903453857267039
```

Filling rating matrix

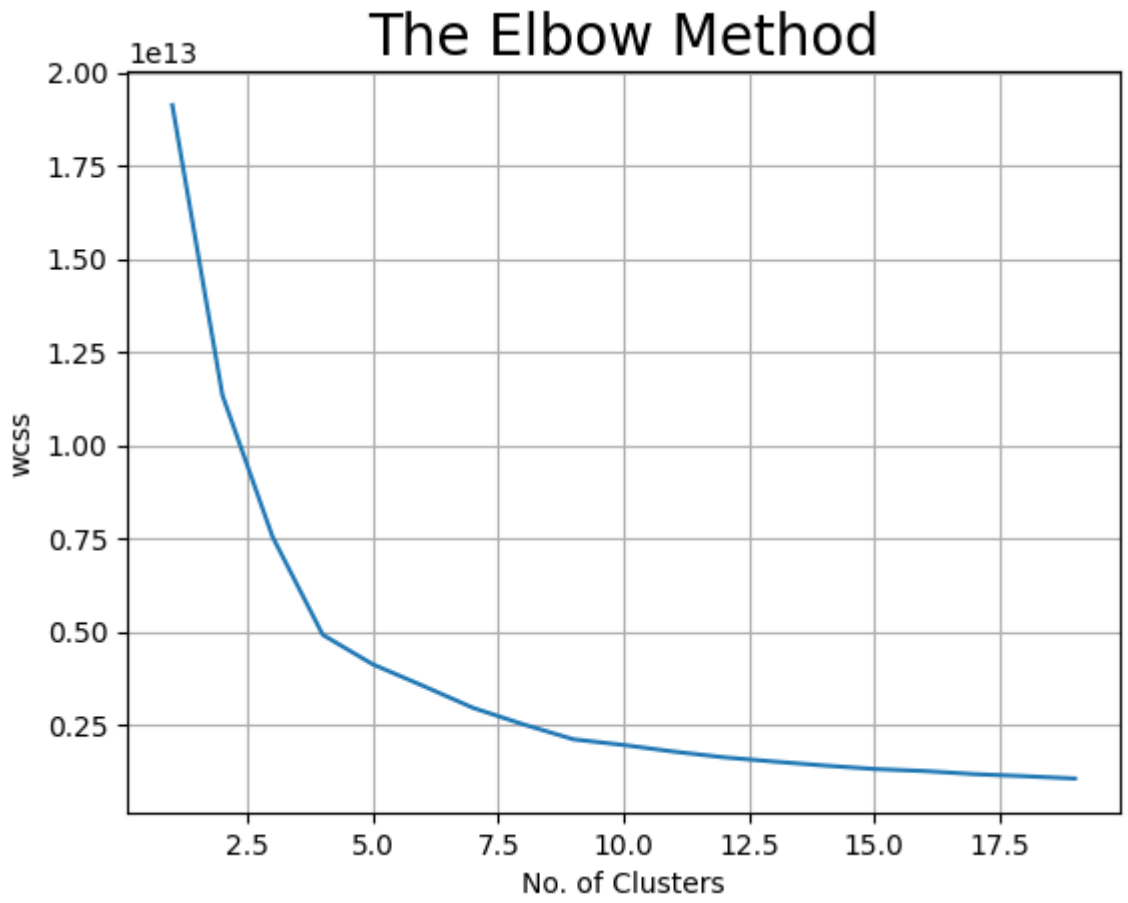
Per completare la matrice di rating, ho eseguito una doppia iterazione applicando il modello addestrato su ogni combinazione di ID utente e ID oggetto. In questo modo, ho previsto e inserito i rating mancanti nella matrice, utilizzando il modello precedentemente addestrato. Questo approccio ha permesso di popolare la matrice con valori previsti per quei rating che non erano ancora presenti.

```python
users_id = ratings["user_id"].unique()
items_id = ratings["parent_asin"].unique()
filled_rating_matrix = []
for uid in users_id:
  filled_rating_matrix.append([])
  for iid in items_id:
    res = final_model.predict(uid=uid, iid=iid)
    if res.r_ui is not None:
      filled_rating_matrix[-1].append(0)
    else:
      filled_rating_matrix[-1].append(res.est)

filled_rating_matrix = np.array(filled_rating_matrix)
```

Algoritmo di clustering

Nell'algoritmo di clustering, ho optato per utilizzare il miniBatchKMeans invece del classico algoritmo KMeans, al fine di migliorare l'efficienza computazionale. Per determinare il valore ottimale di KKK, ho applicato il metodo del gomito (elbow method) su un intervallo da 1 a 20, utilizzando un batch size di mille elementi e considerando solo le feature relative ai rating, agli ID degli utenti e degli oggetti. Dopo aver generato il grafico, ho individuato il punto di gomito intorno al valore 4 da come si può notare:



Per verificare la validità del punto di gomito individuato, ho proceduto applicando l'algoritmo per il calcolo della silhouette score sull'intero dataset con k pari a 4. Il risultato è stato un valore di silhouette particolarmente elevato, pari a 0.97, confermando così la validità e l'efficacia della scelta del valore di k.

```python
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score
numeric_data =features.select_dtypes(include='number')
optimal_k = 4
kmeans_full_data = MiniBatchKMeans(n_clusters = optimal_k, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
kmeans_full_data.fit(numeric_data)
full_data_clusters = kmeans_full_data.predict(numeric_data)
silhouette_avg = silhouette_score(numeric_data, full_data_clusters)
print("Silhouette Score:", silhouette_avg)
```

```
Silhouette Score: 0.9734388905142608
```

Creazione recommendation list

La lista delle raccomandazioni è stata generata partendo dalla matrice dei rating riempita, trasformata in un dataset in cui gli ID degli item fungono da colonne e gli ID degli utenti da righe. Per ogni utente, le colonne sono state ordinate in base ai valori dei rating in modo decrescente. Successivamente, ho costruito il DataFrame delle raccomandazioni utilizzando le colonne ordinate della matrice dei rating, e da come si può notare ogni utente possiede un ranking degli item pari al totale numero di item presenti nel dataset.

```python
res_df = pd.DataFrame(filled_rating_matrix)
res_df.columns = items_id
res_df = res_df.set_index(users_id)

def sort_columns(row):
  sorted_columns = sorted(row.items(), key=lambda x: x[1], reverse=True)
  return [col[0] for col in sorted_columns]

rec_lists = pd.DataFrame(list(res_df.apply(sort_columns, axis=1)),
                         index=res_df.index)
rec_lists
```

	0	1	2	3	4	5	6	7	8
AHZ6XMOLEWA67S3TX7IWEXXGWSOA	B000000XCH	B00006WL1Q	B0000046RN	B0007YKL2G	B00CF0ETBE	B00000IYWL	B0882NXWCX	B000002LV3	B0000793V8
AEVQ3KP55X4XECXWMHN6DHIDBYFQ	B00005R8DV	B0018LMKIK	B00006JIA4	B0000CD5FR	B000B8I8JG	B00000K1I5	B00005R8E1	B004KBSQOW	B000002JUC
AFAIJYOUO3NAWLBDIKTQSC3DASWA	B005FVFWOI	B000001ZS3	B000VFGQUC	B000003NA3	B000005HMC	B000000M0V	B000003L26	B00005KBA0	B0009SOFFY
AFW2PDT3AMT4X3PYQG7FJZH5FXFA	B000002GYZ	B002EE583E	B000F5WNTQ	B000000XCH	B00006WL1Q	B0000046RN	B000002N3H	B000002IJG	B0014DC0G4
AHB5CGLYN3Y6NIPHNQLYFJT2W2PQ	B0000065KK	B00006YXCH	B001BOMBSS	B00000F1U5	B0000248L1	B001CT05XA	B0002LI11M	B00000GBZ1	B0000005MT
...	...	...	...	...	...	...	...	...	...
AGDXS6RFX6QBYON4MRQN6ZV3V53Q	B000002GYZ	B000000XCH	B0882NXWCX	B0000793V8	B005APT8A8	B000005HBG	B000006041	B000I5X856	B000OOOJ0E
AHH3BUIIDFRBKM25VGADGXHLIM3Q	B00005MNP7	B000000XDJ	B09WNGSHJJ	B000002IT2	B000VFGQUC	B0086460KG	B00000JCFW	B0000004V2	B000002H72
AF27OXABUBMLT7Q4VZGYAY6WD35A	B000002GYZ	B000F5WNTQ	B00000348Q	B000000XCH	B00006WL1Q	B0000046RN	B0054YH8DY	B0000025UW	B000002X35
AGXZALWBZGI6ADJMLQMZTOEPFDCQ	B0714CTY2P	B004NTVMLA	B000VKJ6UY	B000002UAU	B000000XDJ	B000002KD7	B00000DQT0	B00004T6QL	B000VFGQUC
AHE4V45MNSR64IMMNE6H4XS7IW7Q	B0000046RN	B0000013G5	B000ETQRCM	B00005A1N2	B000GRTQSE	B000002TWQ	B000006041	B00005MNP7	B00MGSTND4

16944 rows × 15903 columns

Matrix Factorization

Per implementare l'algoritmo di matrix factorization, ho avviato una grid search su diversi parametri direttamente sul dataset precedentemente utilizzato per il KNN. Dopo diverse iterazioni, poiché il valore del RMSE non mostrava miglioramenti significativi, ho concluso il processo. Questo ha portato a un RMSE finale di 0.82, sui seguenti parametri:

```
from surprise.model_selection import GridSearchCV
from surprise import SVD
from surprise import accuracy

param_grid = {
    'n_factors': [80, 100, 120],
    'n_epochs': [25, 30, 35],
    'lr_all': [0.01, 0.02],
    'reg_all': [0.05, 0.1]
}

gs = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=10)
gs.fit(dataset_surprise)
best_params = gs.best_params['rmse']
print(f'Migliori parametri: {best_params}')
print(f'Miglior RMSE: {gs.best_score["rmse"]}')
```

```
Migliori parametri: {'n_factors': 120, 'n_epochs': 35, 'lr_all': 0.02, 'reg_all': 0.1}
Miglior RMSE: 0.8254433069115421
```

Successivamente, ho applicato questi parametri al dataset, dividendo nuovamente in trainset e testset con una suddivisione del 20% per il testset. Dopo aver testato le predizioni, ho ottenuto un RMSE nell'accuratezza pari a 0.837 e un MSE pari a 0.6862, quindi decisamente migliore rispetto a quello trovato con il KNN (MSE=0.8162, RMSE=0.9035).

```python
from surprise import SVD, Dataset, Reader, accuracy
import numpy as np
from surprise import Dataset, Reader, accuracy
from surprise.model_selection import train_test_split

data = Dataset.load_from_df(ratings[['user_id', 'parent_asin', 'rating']], reader)
trainset, testset = train_test_split(data, test_size=0.2)

n_factors = 100
n_epochs = 35
lr_all = 0.02
reg_all = 0.1

final_modelSVD = SVD(n_factors=n_factors, n_epochs=n_epochs, lr_all=lr_all, reg_all=reg_all)
final_modelSVD.fit(trainset)
predictions = final_modelSVD.test(testset)
rmse = accuracy.rmse(predictions)
mse = accuracy.mse(predictions)
print(f'MSE sul set di test: {mse}')
print(f'RMSE sul set di test: {rmse}')
```

```
RMSE: 0.8284
MSE: 0.6862
MSE sul set di test: 0.686241798498792
RMSE sul set di test: 0.8283971260807174
```

Infine, ho completato il riempimento della matrice di rating utilizzando lo stesso algoritmo applicato per il KNN. Ho poi generato la lista di raccomandazioni per ciascun utente, ordinata in modo decrescente in base al valore del rating:

	0	1	2	3	4	5	6	7	8
AHZ6XMOLEWA67S3TX7IWEXXGWSOA	B000000XCH	B00006WL1Q	B0000046RN	B0007YKL2G	B00CF0ETBE	B00000IYWL	B0882NXWCX	B000002LV3	B0000793V8
AEVQ3KP55X4XECXWMHN6DHIDBYFQ	B00005R8DV	B0018LMKIK	B00006JIA4	B0000CD5FR	B000B8I8JG	B00000K1I5	B00005R8E1	B004KBSQOW	B000002JUC
AFAIJYOUO3NAWLBDIKTQSC3DASWA	B005FVFWOI	B000001ZS3	B000VFGQUC	B000003NA3	B000005HMC	B000000M0V	B000003L26	B00005KBA0	B0009SOFFY
AFW2PDT3AMT4X3PYQG7FJZH5FXFA	B000002GYZ	B002EE583E	B000F5WNTQ	B000000XCH	B00006WL1Q	B0000046RN	B000002N3H	B000002IJG	B0014DC0G4
AHB5CGLYN3Y6NIPHNQLYFJT2W2PQ	B0000065KK	B00006YXCH	B001BOMBSS	B00000F1U5	B0000248L1	B001CT05XA	B0002LI11M	B00000GBZ1	B0000005MT
...	...	...	...	...	...	...	...	...	...
AGDXS6RFX6QBYON4MRQN6ZV3V53Q	B000002GYZ	B000000XCH	B0882NXWCX	B0000793V8	B005APT8A8	B000005HBG	B000006041	B000I5X856	B000OOOJ0E
AHH3BUIIDFRBKM25VGADGXHLIM3Q	B00005MNP7	B000000XDJ	B09WNGSHJJ	B000002IT2	B000VFGQUC	B0086460KG	B00000JCFW	B0000004V2	B000002H72
AF27OXABUBMLT7Q4VZGYAY6WD35A	B000002GYZ	B000F5WNTQ	B00000348Q	B000000XCH	B00006WL1Q	B0000046RN	B0054YH8DY	B0000025UW	B000002X35
AGXZALWBZGI6ADJMLQMZTOEPFDCQ	B0714CTY2P	B004NTVMLA	B000VKJ6UY	B000002UAU	B000000XDJ	B000002KD7	B00000DQT0	B00004T6QL	B000VFGQUC
AHE4V45MNSR64IMMNE6H4XS7IW7Q	B0000046RN	B0000013G5	B000ETQRCM	B00005A1N2	B000GRTQSE	B000002TWQ	B000006041	B00005MNP7	B00MGSTND4

16944 rows × 15903 columns

Come si può notare, gli item consigliati per questi dieci utenti (head e tail del dataset) sono identici a quelli consigliati nella lista di raccomandazioni costruita tramite K-NN, nonostante l'RMSE e l'MSE sulla matrix factorization sia più basso.

Processamento attributi testuali

Per processare gli attributi testuali, ho inizializzato il Lemmatizer e le stopwords in lingua inglese. Successivamente, ho processato i campi *title* e *description*, assicurandomi prima che

le descrizioni non fossero vuote. Ho aggiunto al dataset i campi *processed_title* e *processed_description*, contenenti i titoli e le descrizioni processate attraverso una funzione specifica. Questa funzione esegue prima la tokenizzazione del testo per parola, poi applica la lemmatizzazione a ogni parola in minuscolo, filtrando le stopwords e i caratteri non alfabetici, per poi ricomporre i token filtrati in una singola stringa. Alla fine, vi è una stampa dei titoli e descrizioni

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word.lower()) for word in tokens if word.lower() not in stop_words and word.isalpha()]
    return ' '.join(tokens)

amazon_items['description'] = amazon_items['description'].fillna('')
amazon_items['description'] = amazon_items['description'].astype(str)

amazon_items['processed_title'] = amazon_items['title'].apply(preprocess_text)
amazon_items['processed_description'] = amazon_items['description'].apply(preprocess_text)
print(amazon_items[['processed_title', 'processed_description']].head())
```

```
   processed_title                          processed_description
0  release tension                                 release tension
1       rio angie  shrimp city slim aka gary erwin b chicago long...
2       lost love
3   somewhere time  soundtrack classic motion picture starring lat...
4  kimmon waldruff                      acoustic fingerstyle guitar
```

Embedding dei campi

Prima di effettuare l'embedding dei campi del dataset, ho deciso di ripulirlo eliminando gli item non presenti nel dataset delle review. Questo passaggio ha permesso di allineare i due dataset, evitando discrepanze e riducendo la dimensione complessiva del dataset, rendendo i processi più veloci ed efficienti.

```
filtered_amazon_items = amazon_items[amazon_items["parent_asin"].isin(ratings["parent_asin"])].reset_index(drop=True)
print("Total items: " + str(len(amazon_items)) + ", Filtered items: " + str(len(filtered_amazon_items)))
```

```
Total items: 701959, Filtered items: 15903
```

Successivamente, ho proseguito con gli embedding, iniziando con l'approccio Bag of Words (BoW). Utilizzando il *CountVectorizer*, ho creato un dataset che contiene il conteggio delle parole, trasformando così i campi testuali in una rappresentazione numerica utile per le analisi successive.

Ho deciso di basare il vocabolario sul campo *processed_description* del dataset processato e filtrato in precedenza, per ottenere un vocabolario pulito. Alla fine, ho incorporato il vocabolario nel dataset, mantenendo l'ID dell'articolo nel DataFrame per preservare

l'associazione tra la rappresentazione numerica e l'articolo corrispondente. Ecco un esempio del dataset risultante:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
bow_model = vectorizer.fit_transform(filtered_amazon_items['processed_description'])
bow_dataset = pd.DataFrame(bow_model.toarray(), columns=vectorizer.get_feature_names_out())
bow_dataset["parent_asin"] = filtered_amazon_items["parent_asin"]
bow_dataset
```

	aaa	aaaah	aachen	aaf	aaliyah	aames	aaron	ab	abacab	aback	...	émigrés	évaluer	éxitos	íllgresi	ör	öyster	últimos	üaut	über	parent_asin
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B000002X4C
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B0002ADY0Q
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B00KE3B7SC
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B000000H57
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B000R7I3FA
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15898	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B0009JK0XY
15899	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B0009NR7YK
15900	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B00026WU82
15901	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B003097B64
15902	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	B0002MPQIW

15903 rows × 42688 columns

Per quanto riguarda l'embedding con i trasformatori, ho deciso di utilizzare il modello preaddestrato *average_word_embeddings_komninos*. Ho generato gli embedding basati sul campo *processed_description* del dataset filtrato, ottenendo così rappresentazioni in vettori numerici delle descrizioni tutti della stessa grandezza di 300 colonne. Successivamente, ho creato un DataFrame associando ogni rappresentazione numerica all'articolo corrispondente, mantenendo l'associazione tra gli embedding e i rispettivi ID degli articoli.

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('sentence-transformers/average_word_embeddings_komninos')
embeddings = model.encode(filtered_amazon_items['processed_description'])
embeddings_dataset = pd.DataFrame(embeddings)
embeddings_dataset["parent_asin"] = filtered_amazon_items["parent_asin"]
embeddings_dataset
```

```
C:\Users\santi\AppData\Local\Programs\Python\Python312\Lib\site-packages\sentence_transformers\cross_encoder\CrossEncoder.py:11: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from tqdm.autonotebook import tqdm, trange
```

	0	1	2	3	4	5	6	7	8	9	...	291	292	293	294	295
0	0.376667	-0.240261	0.168186	0.071327	-0.193679	0.244111	-0.195401	-0.278250	-0.333683	0.188953	...	-0.164099	-0.068406	0.012150	-0.297209	0.445731
1	0.218384	0.044372	0.110889	-0.100399	-0.151894	0.086375	-0.124995	-0.058304	-0.192454	0.121374	...	0.004997	-0.187514	0.079925	-0.162875	0.298790
2	0.213495	0.056054	0.012911	-0.033832	0.033396	-0.023286	-0.198576	-0.073960	-0.119037	0.137012	...	-0.031984	-0.064959	-0.026999	-0.060447	0.249051
3	0.079970	0.024426	0.079674	-0.029951	-0.159861	0.062422	-0.179397	-0.120607	-0.153526	0.186836	...	-0.014667	-0.136893	0.005965	-0.092427	0.387112
4	0.062232	0.076767	0.041308	-0.012437	-0.159351	0.044738	-0.184855	-0.069320	-0.168199	0.142204	...	-0.007671	-0.187883	0.065462	-0.106061	0.304100
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15898	0.096333	0.072506	0.046591	-0.009555	-0.152288	0.035958	-0.140665	-0.034602	-0.199579	0.166849	...	0.007903	-0.143102	-0.005368	-0.111682	0.307279
15899	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000
15900	0.104837	0.082602	0.101372	-0.034208	-0.216469	0.098428	-0.231500	-0.042215	-0.129367	0.087681	...	-0.022346	-0.181646	0.053668	-0.141349	0.232995
15901	0.225839	0.127714	0.027265	-0.017645	-0.071721	0.119368	-0.185136	-0.105418	-0.253481	0.141724	...	-0.110875	-0.060810	0.070991	-0.156214	0.213669
15902	0.146378	0.110221	0.006608	-0.004598	-0.118063	0.056833	-0.195176	-0.035556	-0.160796	0.090115	...	-0.039338	-0.154626	-0.005841	-0.099212	0.255293

15903 rows × 301 columns

Predizione dei rating

Per effettuare la predizione dei rating attraverso il BoW utilizzando il K-NN, ho iniziato ottenendo gli item a cui l'utente ha dato un rating, prelevandoli dal dataset creato in precedenza con il BoW. Successivamente, ho unito le recensioni date agli item con gli item stessi in un unico dataset.

Ho quindi eliminato diverse colonne superflue che non risultavano utili per l'addestramento, come gli ID, il titolo, il testo (la lunghezza della recensione risulta molto più utile), le immagini (anche qui il numero di immagini inserite precedentemente può essere considerato più utile nell'addestramento) e il timestamp (sostituibile facilmente con il mese e l'anno della recensione, già inseriti nel dataset).

Infine, ho calcolato il MSE per ogni utente dopo aver suddiviso il dataset in set di addestramento e test, creando un array per salvare i valori di MSE calcolando poi la media di essi; e da come si può notare l'utilizzo del KNN con metrica coseno su un dataset creato attraverso il BoW solo sul campo *processed_description* restituisce una media di MSE di 0.79 e una media di RMSE di 0.89.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
mse_users = []
for user_id in ratings['user_id'].unique():
    user_ratings = ratings[ratings['user_id'] == user_id]
    rated_items = bow_dataset[bow_dataset['parent_asin'].isin(user_ratings['parent_asin'])]
    dataset = pd.merge(rated_items, user_ratings, on="parent_asin")
    dataset = dataset.drop(columns=["parent_asin", "user_id", "text_y", "title_y", "images", "asin_y", "timestamp"])

    try:
        X_train, X_test, y_train, y_test = train_test_split(
            dataset.drop(columns="rating_y"),
            dataset['rating_y'],
            test_size=0.20,
            random_state=0
        )


        neigh_reg = KNeighborsRegressor(n_neighbors=min(40, len(X_train)),
                                        metric="cosine")
        y_pred = neigh_reg.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        mse_users.append(mse)

    except Exception as e:
        print(f"Error processing user {user_id}: {str(e)}")
        continue

print(f"Average MSE over users: {np.mean(mse_users):.2f}")
print(f"Average RMSE over users: {np.sqrt(np.mean(mse_users)):.2f}")
```

```
Average MSE over users: 0.79
Average RMSE over users: 0.89
```

Successivamente, ho proseguito con la predizione dei rating utilizzando il dataset creato con i transformers. I passaggi effettuati sono stati i medesimi di quelli utilizzati per il BoW: ho unito

gli oggetti recensiti con il dataset degli oggetti in un unico dataset e, dopo aver rimosso le colonne superflue, ho calcolato l'MSE per ogni utente dopo aver suddiviso il dataset.

Come si può notare, questo approccio ha portato agli stessi valori di MSE e RMSE delle predizioni effettuate con il BoW, che sono rispettivamente 0.79 e 0.89. Questi valori indicano una buona accuratezza del modello.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
mse_users = []
for user_id in ratings['user_id'].unique():
    user_ratings = ratings[ratings['user_id'] == user_id]
    rated_items = embeddings_dataset[embeddings_dataset['parent_asin'].isin(user_ratings['parent_asin'])]
    dataset_rec = pd.merge(rated_items, user_ratings, on="parent_asin")
    dataset_rec = dataset_rec.drop(columns=["parent_asin", "user_id", "text", "title", "images", "asin", "timestamp"])

    try:
        X_train, X_test, y_train, y_test = train_test_split(dataset_rec.drop(columns="rating"),
                                                            dataset_rec['rating'],
                                                            test_size=0.20,
                                                            random_state=0)

        X_train.columns = X_train.columns.astype(str)
        X_test.columns = X_test.columns.astype(str)
        neigh_reg = KNeighborsRegressor(n_neighbors=min(40, len(X_train)),
                                        metric="cosine")
        neigh_reg.fit(X_train, y_train)
        y_pred = neigh_reg.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        mse_users.append(mse)

    except Exception as e:
       print(f"Error processing user {user_id}: {str(e)}")
    continue
print(f"Average MSE over users: {np.mean(mse_users):.2f}")
print(f"Average RMSE over users: {np.sqrt(np.mean(mse_users)):.2f}")
```

```
Average MSE over users: 0.79
Average RMSE over users: 0.89
```

Valutazione risultati con tecniche di embedding

Le due tecniche di embedding portano risultati identici nonostante la loro rappresentazione delle descrizioni sia differente, questo porta a dire che il modello KNN è robusto in base alla scelta di embedding. I risultati di RMSE pari a 0.89 e di MSE pari a 0.79 indicano quindi una buona capacità del modello di predire i rating dato che entrambe le tecniche di embedding catturano efficacemente le informazioni rilevanti dalle descrizioni.

Questo porta quindi a pensare che le descrizioni degli articoli sono sufficientemente informative in modo da permettere a entrambe le rappresentazioni di fornire prestazioni equivalenti. La tecnica BoW, sebbene più semplice, riesce a catturare le parole chiave indicative delle recensioni, mentre i transformers offrono una maggiore comprensione semantica che però, in questo caso, non aggiunge alcun vantaggio significativo.

Valutazione risultati dei sistemi di raccomandazione

Le tecniche di embedding content-based con BoW e transformers hanno mostrato un RMSE di 0.89 e un MSE di 0.79, risultati ottimi per un sistema di raccomandazione con valutazioni da 1 a 5. D'altra parte, le tecniche di collaborative filtering con KNN e matrix factorization hanno riportato prestazioni quasi identiche, sebbene leggermente inferiori per il KNN con un RMSE di 0.90 e un MSE di 0.81. La matrix factorization invece, ha ottenuto i risultati migliori con un RMSE di 0.82 e un MSE di 0.68, il più basso tra tutti i valori ottenuti. Ciò indica che, nonostante l'efficacia delle tecniche content-based, la matrix factorization è quella che ha portato a una maggior accuratezza nelle previsioni.

Considerando la complessità computazionale invece, le tecniche di collaborative filtering come il KNN e la matrix factorization tendono ad essere più lente rispetto a quelle content-based, soprattutto su dataset di grandi dimensioni. Questo è dovuto alla necessità di calcolare le similarità tra utenti o item, richiedendo più risorse computazionali e tempo di elaborazione dovuto soprattutto alla grande quantità di iterazioni e di parameter tuning.

Le tecniche content-based, al contrario, analizzano direttamente le caratteristiche degli item come descrizioni o titoli per generare raccomandazioni, il che le rende generalmente più veloci, specialmente quando il testo è già stato preprocessato, senza la necessità di tuning dei parametri o di iterazioni multiple per ottenere risultati accettabili.

In sintesi, quindi, le tecniche content-based offrono una buona velocità e accuratezza, ma la matrix factorization si è dimostrata la tecnica più precisa in questo dataset nonostante la maggiore complessità computazionale. D'altra parte, il KNN nel collaborative filtering ha mostrato risultati quasi identici a quelli delle tecniche content-based, ma ha richiesto decisamente più tempo e iterazioni per raggiungere il miglior valore di performance.

Conclusioni e interpretazione sintetica dei risultati

In conclusione, l'obiettivo principale di questo progetto era sviluppare un sistema di raccomandazione utilizzando sia tecniche di collaborative filtering (K-NN e matrix factorization) sia tecniche content-based K-NN utilizzando embedding (BoW e transformers), ed è riuscito con ottimi valori per quanto riguarda RMSE e MSE:

Il K-NN nel collaborative filtering ha ottenuto un RMSE di 0.90 e un MSE di 0.81. Questi valori indicano una buona accuratezza nella predizione, anche se leggermente inferiori rispetto ad altre tecniche. La matrix factorization invece ha fornito risultati migliori con un RMSE di 0.82 e un MSE di 0.68, dimostrando che questa tecnica è particolarmente efficace nel catturare le preferenze degli utenti in questo dataset.

Per quanto riguarda il Content-Based Filtering, l'uso del BoW (Bag of Words) per l'embedding delle descrizioni dei prodotti ha portato a un RMSE di 0.89 e un MSE di 0.79, mentre l'uso dei transformers ha raggiunto risultati comparabili a quelli del BoW, con un RMSE di 0.89 e un MSE di 0.79.

Per quanto riguarda la segmentazione degli utenti attraverso l'algoritmo K-MEANS, essa ha portato a un ottimo valore di silhouette pari a 0.97 con un numero di cluster pari a 4. Questo indica quindi che i cluster identificati sono ben definiti e distinti, migliorando la qualità della segmentazione degli utenti.

In sintesi, la scelta tra collaborative filtering e content-based filtering dipenderà dalle risorse computazionali disponibili e dall'accuratezza richiesta. In questo progetto, con il dataset delle recensioni di prodotti Amazon, la matrix factorization si è rivelata il miglior sistema di raccomandazione, grazie ai risultati superiori ottenuti in termini di RMSE e MSE. Tuttavia, le tecniche content-based hanno mostrato comunque notevoli vantaggi, richiedendo una quantità di tempo e risorse decisamente inferiore per raggiungere risultati molto buoni.

D'altra parte, con questo dataset, non considererei l'uso del K-NN nel collaborative filtering, non tanto per i risultati ottenuti, ma per la quantità di tempo e iterazioni che si sono rese necessarie per ottenere quei risultati. Le tecniche content-based, come BoW e transformers, hanno dimostrato di essere più efficienti e scalabili, offrendo una buona alternativa con costi computazionali ridotti.