|  |
| --- |
|  |
| METODI INFORMATICI PER LA GESTIONE AZIENDALE |
| ANNO SCOLASTICO 2023-2024 |
|  |
|  |

Nome e Cognome: Alessandro Rocchi

Matricola: 899810

Anno Accademico: 2023/2024

SOMMARIO

[METODI INFORMATICI PER LA GESTIONE AZIENDALE 1](#_Toc186381419)

[ANNO SCOLASTICO 2023-2024 1](#_Toc186381420)

[SOMMARIO 2](#_Toc186381421)

[executive summary 4](#_Toc186381422)

[Introduzione al problema 5](#_Toc186381423)

[Descrizione dati 5](#_Toc186381424)

[Filtraggio dataset 8](#_Toc186381425)

[Analisi esplorativa 9](#_Toc186381426)

[Risultati del progetto 14](#_Toc186381427)

[IDENTIFICAZIONE DELLA CONFIGURAZIONE OTTIMALE DELL’ALGORITMO K-NN 14](#_Toc186381428)

[IMPORTAZIONE DEI MODULI NECESSARI 14](#_Toc186381429)

[PREPARAZIONE DEI DATI E SUDDIVISIONE DEL DATASET 14](#_Toc186381430)

[FASE DI ADDESTRAMENTO, PREDIZIONE E VALUTAZIONE DEL MODELLO 15](#_Toc186381431)

[OTTIMIZZAZIONE DEL PARAMETRO K CON CROSS-VALIDATION 16](#_Toc186381432)

[RISULTATI DEI VALORI DI MSE E RMSE: CROSS-VALIDATION 17](#_Toc186381433)

[VALORE DI K OTTIMALE: ELBOW METHOD 17](#_Toc186381434)

[CONFIGURAZIONE OTTIMALE: METODO GRID SEARCH 18](#_Toc186381435)

[FILLING DELLA MATRICE DI RATING CON LA CONFIGURAZIONE OTTIMALE 19](#_Toc186381436)

[ALLENAMENTO DELL’ALGORITMO SULL’INTERO DATASET CON LA CONFIGURAZIONE OTTIMALE 19](#_Toc186381437)

[CREAZIONE DEL RATING\_MATRIX\_ORIGINALE 20](#_Toc186381438)

[PREDIZIONE DEL RATING DI TUTTI I PRODOTTI PER UN USER SPECIFICO 21](#_Toc186381439)

[TOP 10 PRODOTTI CONSIGLIATI PER L’UTENTE “AGTMZCWIWBH43TCW7UKG2YV2EKKA” 22](#_Toc186381440)

[FILLING DELLA MATRICE DEI RATING CON LA CONFIGURAZIONE OTTIMALE 23](#_Toc186381441)

[SEGMENTAZIONE DEGLI UTENTI IN BASE ALLE PREFERENZE: ALGORITMO DI CLUSTERING K-MEANS CON COSINE SIMILARITY 25](#_Toc186381442)

[CREAZIONE DELLA LISTA PERSONALIZZATA DI RACCOMANDAZIONI PER CIASCUN UTENTE 28](#_Toc186381443)

[FILLING MATRICE DEI RATING: MATRIX FACTORIZATION 29](#_Toc186381444)

[CONFIGURAZIONE OTTIMALE DELL’ALGORITMO SVD: METODO GRID SEARCH 30](#_Toc186381445)

[CONFRONTO DEGLI ERRORI MSE E RMSE: MODELLI OTTIMIZZATI K-NN E SVD 31](#_Toc186381446)

[TOP 10 PRODOTTI CONSIGLIATI PER L’UTENTE “AGTMZCWIWBH43TCW7UKG2YV2EKKA” 32](#_Toc186381447)

[FILLING DELLA MATRICE DI RATING: MATRIX FACTORIZATION 32](#_Toc186381448)

[Conclusioni 34](#_Toc186381449)

# executive summary

OBIETTIVO DEL PROGETTO

Il progetto si propone di sviluppare un sistema di raccomandazioni basato su tecniche di Collaborative Filtering, mirato a prevedere i rating mancanti e suggerire i prodotti più rilevanti per ciascun utente. L’obiettivo è confrontare approcci diversi: **K-NN** e **Matrix Factorization**, per ottimizzare la precisione delle raccomandazioni, segmentare gli utenti in base alle preferenze e generare liste personalizzate di prodotti.

METODOLOGIA

1. Analisi esplorativa dei dati: Studio preliminare delle caratteristiche del dataset, con calcolo di statistiche descrittive e analisi della correlazione tra utenti e prodotti.
2. Ottimizzazione del K-NN: Test di diverse configurazioni del modello (Similitudine, numero di vicini k, approccio user/item-based), identificando la combinazione ottimale in base alle metriche MSE e RMSE.
3. Filling della matrice dei rating: Completare la matrice dei rating inserendo i rating mancanti con i rating predetti prodotti a partire dalla configurazione ottimale dell’algoritmo K-NN.
4. Segmentazione degli utenti: Applicazione dell’algoritmo di clustering K-Means con cosine similarity per raggruppare gli utenti in base a preferenze simili.
5. Matrix Factorization (SVD): Implementazione di un modello di fattorizzazione della matrice trovando la configurazione ottimale per confrontare la performance rispetto al K-NN.
6. Generazione di raccomandazioni: Creazione di liste personalizzate di prodotti (Top N items) per ogni utente.

RISULTATI PRINCIPALI

* **Modello K-NN**: L’ottimizzazione ha prodotto un **MSE di 0.8623** e un **RMSE di 0.9286** con configurazione ottimale (**K = 24, similitudine MSD, approccio item-based**).
* **Modello SVD**: Senza ottimizzazione ha ottenuto MSE di 0.8168 e RMSE di 0.9038; Con configurazione ottimale (**n\_factors = 90, n\_epochs = 30, biased = True**), ha migliorato ulteriormente i risultati (**MSE = 0.8064**, **RMSE = 0.8980**)
* **Confronto**: SVD si è dimostrato superiore a K-NN in termini di accuratezza, grazie alla sua capacità di catturare meglio relazioni latenti nei dati.
* **Segmentazione degli utenti**: I clusters identificati con K-Means sono stati 20, i quali hanno mostrato gruppi con preferenze comuni, permettendo una personalizzazione più mirata delle raccomandazioni.
* **Liste personalizzate**: I modelli hanno generato raccomandazioni diverse per gli stessi utenti, riflettendo le differenze tra K-NN e SVD nella gestione delle similarità e dei pattern latenti.

CONCLUSIONI

Il progetto ha dimostrato che la Matrix Factorization (SVD) supera il K-NN nella previsione dei rating mancanti, grazie alla sua capacità di catturare relazioni latenti.

# Introduzione al problema

L’obiettivo principale è quello di sviluppare un sistema di raccomandazione (Recommendation System), a partire da un set di dati relativi a recensioni di prodotti Amazon raggruppate per categorie merceologiche, basato sui diversi approcci: Collaborative-Filtering e Content-Based.

Per ogni categoria vi sono due possibili dataset da scaricare: User Reviews e Item Metadata.

In questo caso ho scelto di sviluppare il progetto base che ha come obiettivo principale quello di sviluppare un sistema di raccomandazione basato su Collaborative Filtering prendendo il dataset di “User Reviews” di una sola categoria a scelta che in questo caso ho scelto la categoria di “*CDs\_and\_Vinyl*” utilizzando i campi di “user\_id”, “parent\_asin” e “rating” per l’elaborazione del sistema di raccomandazione.

* Per l’elaborazione del sistema di raccomandazione sono stati utilizzati soltanto i campi di “user\_id”, “parent\_asin” e “rating”

## Descrizione dati

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamenteLa categoria di “*CDs\_and\_Vinyl*” presentava inizialmente:

Il dataset contiene 10 colonne: “rating”, “title”, “text”, “images”, “asin”, “parent\_asin”, “user\_id”, “timestamp”, “helpful\_vote” e “verified\_purchase”

Immagine che contiene testo, schermata, Carattere, numero

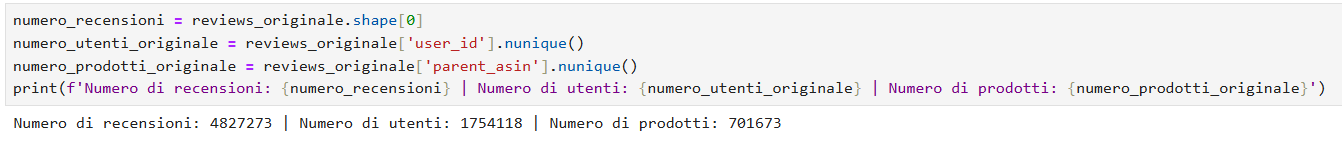
Descrizione generata automaticamentePossiamo conoscere il tipo di dato di ogni colonna facendo uso di “dtypes”, trovando:

Nonostante ciò, abbiamo bisogno di utilizzare soltanto le colonne di “user\_id”, “parent\_asin” e “rating” come quelle importanti da tenere conto al momento di sviluppare il sistema di raccomandazione.

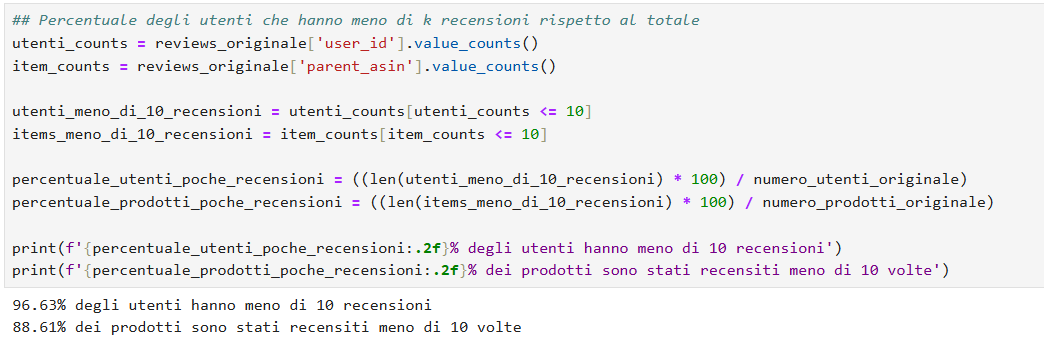
Per questo motivo abbiamo filtrato il dataset in modo tale che ci siano soltanto quelle colonne.

Dopodiché, facendo uso del metodo “shape[0]” reperiamo il numero totale di recensioni.

Cosiccome usando il metodo “nunique()” reperiamo il numero di valori unici delle colonne “user\_id” e “parent\_asin” per trovare il numero di utenti e prodotti rispettivamente



* 4,827,273 (4,8M) recensioni
* 1,754,118 (1,7M) utenti
* 701,673 prodotti

Visto che l’obiettivo principale è raccomandare dei prodotti in base alle preferenze e comportamenti in comune tra gli utenti (Stiamo prendendo in considerazione l’approccio “Collaborative Filtering”) l’idea chiave è che gli utenti simili tendono ad avere interessi simili, perciò, si vuole avere un minimo di recensioni “k” per ogni prodotto e ogni utente per fare in modo che ci sia una maggior probabilità di trovare delle somiglianze tra gli utenti in base alle loro preferenze, anche se questo comporterebbe la diminuzione della dimensione del dataset.

Usando il metodo “value\_counts()” troviamo il numero di recensioni che ha ogni utente e prodotto.

Se selezioniamo gli utenti e prodotti con meno di, ad esempio, 10 recensioni possiamo rappresentarli con rispetto al totale e abbiamo che:

* 96.63% degli utenti hanno meno di 10 recensioni
* 88.61% dei prodotti sono stati recensiti meno di 10 volte

Ciò vuol dire che la maggioranza degli utenti e prodotti non hanno uno storico precedente sufficiente per poter essere relazionati con altri utenti e prodotti, quindi, bisogna filtrarli per far si che ci siano solo gli utenti e prodotti che abbiano più di k recensioni.

## Filtraggio dataset

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Utilizziamo il metodo *value\_counts()* per reperire il numero di recensioni per utente e prodotto.

Poi filtriamo solo gli utenti e prodotti che hanno almeno 18 e 15 recensioni rispettivamente. La scelta del numero K di recensioni è stata fatta per far sì che rimangano all’incirca 100,000 recensioni in totale con un numero di utenti e prodotti né troppo alto né troppo basso. Con scelte diverse di k il numero di utenti e prodotti incrementerebbe esponenzialmente il che ci darebbe problemi di memoria e di tempi di riposta al momento sviluppare il sistema di raccomandazione.

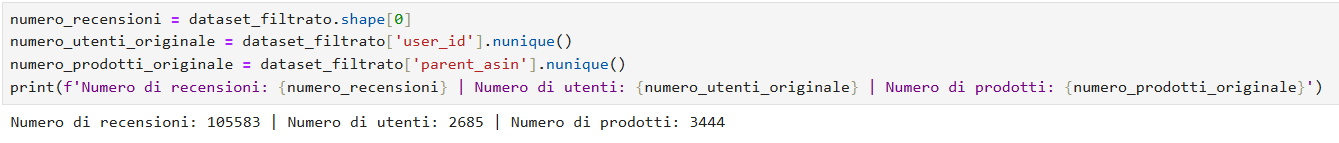
Col metodo “.index” prendiamo gli indici, ovvero, gli ID degli utenti e prodotti che soddisfano il criterio del numero k di recensioni minimi.

Alla fine utilizziamo il metodo *isin()* per verificare se gli elementi di una colonna (user\_id, parent\_asin) appartengono agli indici estratti, questo permette di mantenere nel dataset solo gli utenti e prodotti che rispettano i criteri minimi di recensioni.

È importante filtrare il dataset iterativamente dentro un ciclo in quanto c’è una dipendenza tra gli utenti e i prodotti. Ad esempio, se viene eliminato un utente con meno di k recensioni tutti i prodotti che ha recensito perderebbero 1 recensione il che potrebbe farli non soddisfare più il criterio minimo di recensioni e quindi verrebbero rimossi anch’essi dal dataset nella prossima iterazione.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente



Dopo il filtraggio il Dataset ha:

* 105,583 recensioni
* 2685 utenti
* 3444 prodotti

## Analisi esplorativa

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Facendo uso dei metodi *head(), tail()* e *sample()* possiamo verificare che le righe di intestazione all’inizio e le righe vuote alla fine non siano state incluse.

Questo controllo è fondamentale visto che non vogliamo che siano interpretati come dati e sporchino i dati importanti.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteDall’altro canto, tramite il metodo *isnull()* possiamo effettivamente controllare che all’interno dell’interno dataset **non** ci sono valori mancanti, valori nulli o valori inappropriati.

Immagine che contiene testo, schermata, diagramma, Carattere

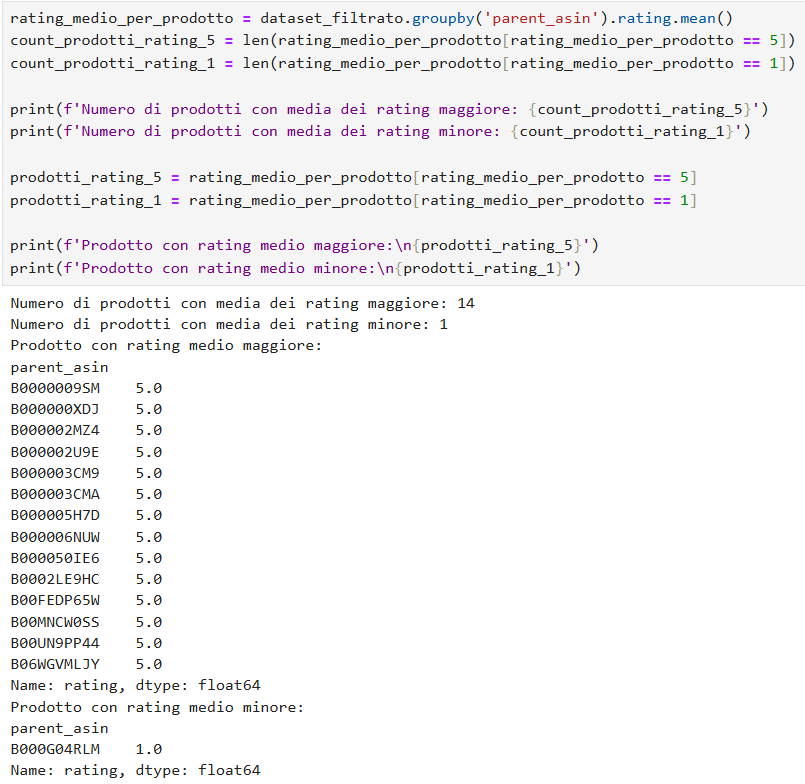
Descrizione generata automaticamente

Procediamo con il calcolo delle **proporzioni dei rating**:

* **57,6%** delle recensioni hanno avuto un rating = 5
* **23,2%** delle recensioni hanno avuto un rating = 4
* **9,5%** delle recensioni hanno avuto un rating = 3
* **5,5%** delle recensioni hanno avuto un rating = 1
* **4,2%** delle recensioni hanno avuto un rating = 2

Possiamo dire che i prodotti di CD’s e Vinili sono stati accolti in maniera positiva dagli utenti in quanto più dell’80% degli utenti hanno assegnato un voto >= 4 ai prodotti di questa categoria.

PRODOTTI PIU’ E MENO VOLUTI

* Il prodotto meno voluto in assoluto con un voto medio di 1 è stato: **B000G04RLM**
* I prodotti più voluti con un voto di 5 sono stati:
  + **B0000009SM**
  + **B000000XDJ**
  + **B000002MZ4**
  + **B000002U9E**
  + **B000003CMA**
  + **B000005H7D**
  + **B000006NUW**
  + **B000050IE6**
  + **B0002LE9HC**
  + **B00FEDP65W**
  + **B00MNCW0SS**
  + **B06WGVMLJY**

VOTO MEDIO PER UTENTE

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

* L’utente **AFGNRMK2UDSWVDQBNUNTYHNBAN3Q** ha recensito più prodotti negativamente che chiunque altro quindi possiamo inferire che la categoria di CD’s e Vinili non è la sua preferita.

* Ci sono 227 utenti che hanno assegnato un voto medio di 5 ai prodotti che hanno recensito il che vuol dire che la probabilità di recensire positivamente altri prodotti della categoria è maggiore.

DISTRIBUZIONE DEI RATING: PLOTBAR

Immagine che contiene testo, schermata, schermo, diagramma

Descrizione generata automaticamente

Con il metodo “.plot()” possiamo plottare la frequenza dei ratings.

Dal grafico possiamo notare:

* Il rating più frequente è il 5
* Il rating meno frequente è il 2

PRINCIPALI STATISTICHE DESCRITTIVE: RATING

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Utilizzando il metodo *describe()* abbiamo analizzato le principali statistiche descrittive dei rating e abbiamo ottenuto che:

* Il numero totale di valutazioni è pari a 105,583.
* La valutazione media è 4.23 indicando la tendenza generale verso recensioni positive.
* La deviazione standard è pari a 1.13, il che suggerisce che le valutazioni sono distribuite vicino alla media. La maggior parte delle recensioni sono comprese nell’intervallo [3.10, 5].

TOP 10 PRODOTTI PIU’ RECENSITI

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

I prodotti più recensiti sono stati:

* **B00008OWZG** con 220 recensioni
* **B000002H97** con 205 recensioni
* **B000002H33** con 198 recensioni
* **B000002J09** con 194 recensioni
* **B000002UB3** con 190 recensioni
* **B00004ZAV3** con 184 recensioni
* **B000089RVX** con 184 recensioni
* **B000002U82** con 179 recensioni
* **B0002OERI0** con 178 recensioni
* **B000002UAU** con 175 recensioni

RATING MEDIO ASSEGNATO DAGLI UTENTI:BOXPLOT

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

Il boxplot rappresenta i rating medi assegnati dagli utenti, calcolati raggruppando i dati per “user\_id” e prendendo la media dei rating di ciascun utente.

Possiamo vedere che l’intervallo interquartile si trova tra circa il voto 4 e 5 il che significa che il 50% dei dati si trovano proprio qui. Questo indica una distribuzione concentrata verso l’alto.

Gli outlier, ovvero, i punti fuori dai baffi rappresentano gli utenti con un comportamento anomalo rispetto alla maggior parte e ci indicano che alcuni utenti hanno assegnato rating molto bassi compressi tra l’1 e il 3.

# Risultati del progetto

In questa sezione vengono presentati i risultati ottenuti attraverso le diverse fasi di sviluppo del sistema di raccomandazioni basato su collaborative filtering. Ogni passaggio è stato progettato per ottimizzare l'accuratezza delle previsioni, garantire una segmentazione efficace degli utenti e generare raccomandazioni personalizzate.

Il lavoro è stato suddiviso in diverse fasi principali, tra cui l’identificazione della configurazione ottimale per l’algoritmo K-NN, il riempimento della matrice di rating tramite tecniche di predizione, la segmentazione degli utenti in base alle preferenze e l’applicazione della Matrix Factorization come confronto.

I risultati sono analizzati con metriche come MSE e RMSE per valutare l’efficacia degli algoritmi utilizzati, mentre l’approccio di clustering degli utenti ha permesso di evidenziare pattern di preferenze utili per migliorare ulteriormente le raccomandazioni.

## IDENTIFICAZIONE DELLA CONFIGURAZIONE OTTIMALE DELL’ALGORITMO K-NN

### IMPORTAZIONE DEI MODULI NECESSARI

Prima di tutto dobbiamo importare dalla libreria “Surprise” i seguenti moduli:

* Immagine che contiene testo, Carattere, schermata, linea

  Descrizione generata automaticamenteDataset 🡪 Questa classe ci permette di trasformare i nostri dati in un formato compatibile con Surprise in modo di poter gestirli e manipolarli per l’addestramento dell’algoritmo K-NN.
* Reader 🡪 Ci permette di definire il range di valori dei rating.
* Accuracy 🡪 Contiene i metodi MSE e RMSE che ci permetteranno di valutare le prestazioni degli algoritmi di raccomandazione.
* KNNBasic 🡪 Algoritmo che utilizzeremo per calcolare le somiglianze tra utenti/prodotti e di effettuare raccomandazioni in base ai vicini più simili.
* Model\_selection 🡪 Ci permette di dividere i dati in trainset e testset e poi eseguire il cross-validation.

### PREPARAZIONE DEI DATI E SUDDIVISIONE DEL DATASET

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

Per preparare il dataset per l'addestramento del modello, abbiamo utilizzato la libreria Surprise. In primo luogo, abbiamo definito un oggetto Reader per specificare il range dei valori delle valutazioni, che nel nostro caso è compreso tra 1 e 5. Successivamente, il dataset è stato caricato utilizzando il metodo “load\_from\_df” per trasformarlo in un formato compatibile con la libreria.

Inoltre, abbiamo inizializzato l’algoritmo K-NN sulla variabile “algo”

Infine, il dataset trasformato è stato suddiviso in due sezioni: Il **trainset** e il **testset** utilizzando il metodo “train\_test\_split()” il cui riceve un parametro “test\_size” per specificare la porzione dell’intero dataset che vogliamo destinare al set di test, che in questo caso abbiamo scelto l’80% dei dati per l’addestramento e il 20% per il test.

### FASE DI ADDESTRAMENTO, PREDIZIONE E VALUTAZIONE DEL MODELLO

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

L’algoritmo KNN viene addestrato sul trainset con il metodo “fit()”. Questo passaggio permette all’algoritmo di apprendere le relazioni tra utenti e prodotti, successivamente calcola le similarità tra utenti/prodotti.

Dopo essere addestrato l’algoritmo viene utilizzato per effettuare predizioni sui dati destinati al test tramite il metodo “test()”. Tutte le predizioni le salviamo sulla variabile “predictions”.

Ora possiamo passare al calcolo degli errori MSE e RMSE sulle predizioni tramite il modulo “accuracy” che invoca i metodi “mse()” e “rmse()”.

In questo caso abbiamo che:

* **MSE: 0.9591**
* **RMSE: 0.9794**

Possiamo vedere che entrambi hanno un valore minore a 1, il che ci fa capire che l’errore non è così grande e quindi, **inizialmente, il modello è abbastanza accurato** anche se non abbiamo ancora trovato la configurazione ottimale.

### OTTIMIZZAZIONE DEL PARAMETRO K CON CROSS-VALIDATION

Attraverso il cross-validation possiamo provare diversi valori di K per l’algoritmo in modo tale di trovare il valore ottimo nel quale gli errori MSE e RMSE siano i minori in assoluto e, dunque, il modello sia il più accurato possibile.

Il processo del cross-validation suddivide i dati in 5 folds in cui l’algoritmo viene addestrato su 4 di essi e testato sull’ultimo, ripetendo questo processo per ogni fold.

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

Per prima cosa creiamo degli array in cui appenderemo gli eventuali valori di mse e rmse al variare di K, questi si chiameranno “mse\_over\_k” e “rmse\_over\_k”. È molto importante tener traccia dei risultati ad ogni iterazione per, poi, poter rappresentarli graficamente tramite l’elbow method.

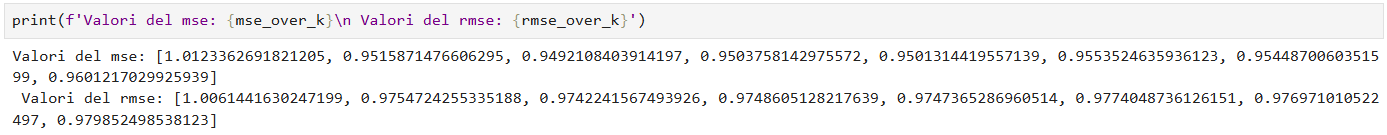
Abbiamo scelto i valori di K dal 5 al 41 con un incremento di 5 unità con il metodo “.arange()” in questa maniera l’algoritmo non diventa né troppo specifico, impostando K piccoli, né troppo generalista, impostando K molto grandi.

Un valore piccolo di K avrebbe reso il modello più focalizzato sui vicini stretti e quindi potrebbe verificarsi un overfitting, ovvero, che il modello diventerebbe troppo specifico rispetto ai dati di addestramento il che non permetterebbe al modello di apprendere pattern generalizzabili.

Dall’altro canto con una scelta grande di K renderebbe il modello più generalista il che potrebbe non catturare i pattern nei dati di addestramento, cioè, non riuscirebbe a rappresentare adeguatamente la complessità del problema.

Visto che il cross-validation deve addestrare e testare l’algoritmo su ogni fold ci saranno cinque valori di mse e rmse alla fine dell’esecuzione, procediamo a prendere il valore medio di questi valori e appenderli sugli array appositi per salvarli tramite il metodo “.append()”.

### RISULTATI DEI VALORI DI MSE E RMSE: CROSS-VALIDATION

****

Come possiamo osservare: I valori del MSE e del RMSE arrivano al minimo globale nella **seconda posizione**, ciò vuol dire che con **K = 15** si otterrebbe la migliore configurazione di K.

Procediamo a plottare i risultati tramite l’elbow method.

### VALORE DI K OTTIMALE: ELBOW METHOD

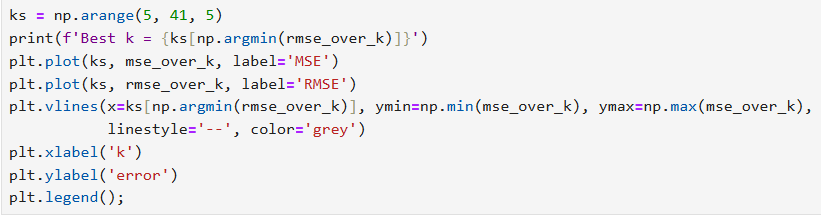


Immagine che contiene testo, diagramma, Diagramma, linea

Descrizione generata automaticamente

L’obiettivo di questo grafico è poter visualizzare l’andamento degli errori MSE e RMSE al variare del valore di K nel modello KNN.

Si è utilizzato l’Elbow method per individuare il valore di K basandosi sui valori degli errori calcolati durante la fase del cross-validation.

* L’asse x rappresenta i valori di K testati.
* L’asse y rappresenta i valori di errore MSE e RMSE.

Tra i valori di MSE e RMSE prendiamo in considerazione i valori dell’RMSE in quanto questi vengono rappresentati nella scala all’unità originale che in questo caso è [1, 5].

Il metodo *plot()*, per plottare le curve dei valori di MSE e RMSE, prende come parametri i valori dell’asse x (ks), i valori dell’asse y (mse\_over\_k/rmse\_over\_k) e un’etichetta (label = ‘MSE/RMSE’).

Dopodiché, tracciamo la linea verticale nel punto in cui si verifica il valore minimo assoluto dell’RMSE tramite il metodo *vlines()* che prende come parametri:

* Il punto dove l’RMSE è il minimo assoluto, ovvero, *np.argmin(rmse\_over\_k)*.
* I limiti **ymin** e **ymax** in cui sarà contenuta la linea verticale, lo stile e il colore della linea.

Infine, assegniamo i nomi degli assi x e y tramite i metodi *xlabel()* e *ylabel()*.

Osserviamo dal grafico che all’aumentare di K, l’errore inizialmente decresce fino a raggiungere un minimo per **K = 15**, dopodiché, inizia a crescere.

Nonostante ciò, questo è un **metodo euristico e non accurato**, serve solo per avere una prima idea di quanto potrebbe essere il valore di K, per questo motivo, esiste un altro metodo per trovare la configurazione ottimale dell’algoritmo KNN: **GridSearch**.

### CONFIGURAZIONE OTTIMALE: METODO GRID SEARCH

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteQuesto metodo ci permette di ricavare la configurazione ottimale per l’algoritmo KNN tramite una ricerca esaustiva della configurazione ottimale utilizzando diverse misure di similarità: MSD, Cosine, Pearson e diversi approcci: User-based e Item-based.

Eseguiamo i seguenti passaggi:

1. **Definiamo un dizionario** “*param\_grid*” in cui specifichiamo i valori da testare per ogni parametro.
2. K 🡪 Scegliamo i valori di K da **[15, 41].**
3. Sim\_options 🡪 Prendiamo in considerazione tutte le misure di similarità: **MSD, Cosine e Pearson** ed entrambi gli approcci: **User-Based e Item-Based**.
4. Invochiamo e inizializziamo il metodo GridSearch che prenderà come parametri:
5. KNNBasic 🡪 L’algoritmo da ottimizzare.
6. Param\_grid 🡪 La griglia di parametri da testare definita precedentemente.
7. Measures = [“rmse”, “mse”] 🡪 Le metriche di valutazione usate per il confronto dei risultati.
8. Cv = 5 🡪 Numero di folds per la cross-validation.
9. N\_jobs = -1 🡪 Numeri di processi paralleli per eseguire i calcoli. Quando impostato a -1 sfrutta tutti i core disponibili del processore.

Osserviamo che le metriche più basse ottenute durante la ricerca sono:

* **MSE = 0.8623**
* **RMSE = 0.9286**

E questi valori sono stati trovati con seguente **configurazione ottimale**:

* Best configuration = {'k': 24, 'sim\_options': {'name': 'msd', 'user\_based': False}}

In altre parole, con **K = 24, similarità MSD e con l’approccio Item-based** riusciamo a ottenere un modello KNNBasic il più accurato possibile.

## FILLING DELLA MATRICE DI RATING CON LA CONFIGURAZIONE OTTIMALE

### ALLENAMENTO DELL’ALGORITMO SULL’INTERO DATASET CON LA CONFIGURAZIONE OTTIMALE

**Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente**

Dopo aver trovato la configurazione ottimale l’algoritmo viene allenato utilizzando tutti i dati disponibili, ovvero, l’intero dataset in quanto non c’è più bisogno di riservare una parte dei dati al testset. Questo serve per sfruttare al massimo le informazioni presenti nel dataset e garantire che il modello sia il più accurato possibile prima delle predizioni.

Visto che l’obiettivo è utilizzare l’intero dataset per allenare il modello utilizzeremmo il metodo *build\_full\_trainset()* che ci permette di non splittarlo ma di unificarlo per il training.

### CREAZIONE DEL RATING\_MATRIX\_ORIGINALE

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

La creazione della matrice dei rating originale è un passaggio fondamentale in quanto ci permette di individuare i prodotti non ancora recensiti e sui quali fare predizioni, quindi, bisogna tenere traccia delle interazioni tra utente e prodotto.

L’obiettivo, quindi, è di creare una matrice in cui ogni riga rappresenti un utente e ogni colonna rappresenti un prodotto, per questo motivo abbiamo prima reperito tutti gli utenti e prodotti tramite il metodo *unique()*, dopodiché, tramite il metodo *DataFrame()* di pandas abbiamo creato la matrice:

* Impostando inizialmente tutte le celle a 0.
* Assegnando le righe tutti gli users\_id.
* Assegnando alle colonne gli items\_id.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamentePer il popolamento della matrice con i rispettivi rating utilizziamo un ciclo in cui ogni riga rappresenta un’interazione user\_id 🡪 parent\_asin con un determinato rating, quindi, per ogni interazione viene aggiornato con il rating effettivo.

La matrice dei rating originale è stata creata e presenta: **2685 righe** (Numero di utenti univoci) e **3444 colonne** (Numero di prodotti univoci).

Tramite questa matrice, quindi, possiamo fare distinzione tra i prodotti recensiti o meno.

### PREDIZIONE DEL RATING DI TUTTI I PRODOTTI PER UN USER SPECIFICO

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Prima di applicare il modello sull’intero dataset è importante testare le funzionalità del processo su un sottoinsieme dei dati su piccola scala.

Questo approccio porta molti vantaggi come:

* Riduzione dei tempi di esecuzione
* Costo computazionale minore
* Facilità di debug

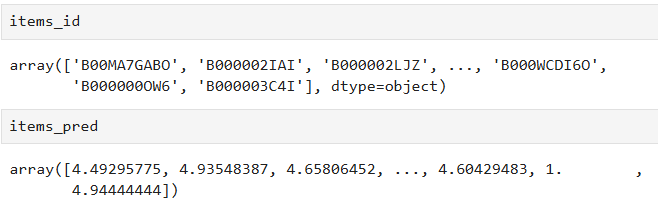
Abbiamo considerato l’utente “**AGTMZCWIWBH43TCW7UKG2YV2EKKA**” per il quale vogliamo calcolare le predizioni e la lista di tutti i prodotti univoci.

L’array “items\_pred” sarà una lista che conterrà i valori predetti per ogni prodotto.

Attraverso un ciclo controlliamo se ogni prodotto è stato recensito dall’utente in questione, si possono verificare due possibilità:

1. Se l’utente ha già assegnato un rating al prodotto allora aggiungiamo il valore di 0 alla lista delle predizioni “items\_pred” per indicare che non è necessaria una predizione.
2. Se l’utente non ha ancora assegnato un rating al prodotto allora viene utilizzato il metodo *algo.predict()* sull’utente e sul prodotto in questione e viene salvata in una variabile “res”. Possiamo reperire la predizione tramite il metodo *res.est* e l’aggiungiamo alla lista delle predizioni “items\_pred”.

Infine, la lista delle predizioni “items\_pred” viene convertita in un array Numpy per semplificare successive analisi.



A primo impatto possiamo osservare alcune delle predizioni che sono state salvate nella lista “items\_pred” per ogni prodotto.

Come ad esempio:

* Per il prodotto B00MA7GABO l’algoritmo ha calcolato un rating di 4.49295775.
* Per il prodotto B000002IAI l’algoritmo ha calcolato un rating di 4.93548387.
* Per il prodotto B000002LJZ l’algoritmo ha calcolato un rating di 4.65806452.

E così via per tutti i prodotti.

### TOP 10 PRODOTTI CONSIGLIATI PER L’UTENTE “AGTMZCWIWBH43TCW7UKG2YV2EKKA”

Dopo aver calcolato i punteggi predetti per i prodotti non recensiti da un utente specifico, abbiamo selezionato i 10 prodotti con rating predetti più alti.

Per prima cosa ordiniamo gli indici dell’array dei rating predetti in modo **crescente** tramite il metodo *np.argsort()*. Dopodiché invertiamo l’ordine in modo **decrescente** tramite l’operazione [::-1].

Usando gli indici ordinati in modo decrescente riorganizziamo sia l’array dei rating predetti che gli ID dei prodotti in modo che siano ordinati dal rating più alto al più basso.

Estraiamo una lista degli ID dei primi 10 prodotti, così come il loro punteggio.

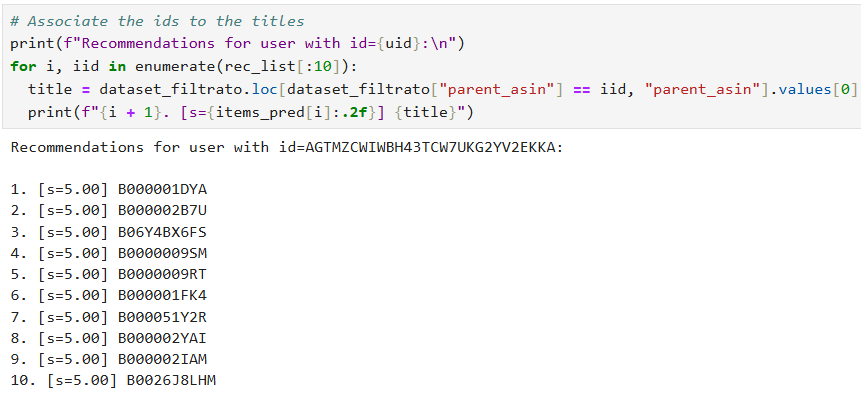
Ci rendiamo conto che i primi dieci prodotti con rating predetto più alti sono stati:

1. B000001DYA

Tutti i prodotti hanno avuto un rating predetto uguale a 5.0.

Questo accade dovuto al fatto che i dati hanno una tendenza generale verso punteggi positivi, infatti, osserviamo che quasi il 60% degli utenti hanno recensito con rating = 5. Quindi naturalmente i vicini degli utenti tenderanno a dare voti positivi.

1. B000002B7U
2. B06Y4BX6FS
3. B0000009SM
4. B0000009RT
5. B000001FK4
6. B000051Y2R
7. B000002YAI
8. B000002IAM
9. B0026J8LHM

Questo è un altro modo da rappresentare i top 10 prodotti con i loro rispettivi rating.

### FILLING DELLA MATRICE DEI RATING CON LA CONFIGURAZIONE OTTIMALE

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteDopo aver creato un sistema di raccomandazione per un solo utente procediamo a portarlo a grande scala creando la matrice dei rating predetti per ogni utente per ogni prodotto tenendo conto dei prodotti che sono stati e non sono stati recensiti per creare delle predizioni dove c’è bisogno.

Come primo passaggio estraiamo tutti gli utenti e i prodotti e li salviamo nelle variabili “user\_id” e “items\_id” rispettivamente.

Inizialmente, creiamo una lista vuota “filled\_rating\_matrix” che durante il ciclo, ad ogni iterazione, viene aggiunta una lista per ogni *uid*.

Dentro il ciclo annidato viene controllato se ogni prodotto è stato recensito o meno controllando la matrice dei rating originale in cui si verificano due casi:

1. Il rating == 0 🡪 Il prodotto NON è stato ancora recensito.
2. Il rating > 0 🡪 Il prodotto è già stato recensito.

Quindi:

* **Appendiamo uno zero** nella matrice dei ratings SE il prodotto è già stato recensito.
* **Appendiamo la predizione del rating** SE il prodotto non ha ancora un rating nella matrice dei rating originale. Come?

Invochiamo il metodo *predict(uid, iid)* e salviamo la risposta nella variabile “res”, dopodiché, appendiamo il *res.est* che rappresenta il rating predetto.

Appendiamo i valori nella posizione [-1] della matrice dei rating predetti in quanto con l’indice -1 accediamo sempre all’ultima lista aggiunta, cioè, l’ultima riga della matrice in costruzione che vogliamo popolare.

*(Se matrice\_rating\_filtrato.at[uid, iid] > 0 ALLORA filled\_rating\_matrix[-1].append(****0****))*

*(Se matrice\_rating\_filtrato.at[uid, iid] == 0 ALLORA filled\_rating\_matrix[-1].append(****res.est****))*

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Alla fine del processo la matrice viene convertita in un array Numpy per una facile manipolazione successiva.

Come passaggio finale creiamo il DataFrame corrispondente alla matrice dei rating per ogni utente e prodotto in cui abbiamo fillato i rating mancanti con i rating predetti.

## SEGMENTAZIONE DEGLI UTENTI IN BASE ALLE PREFERENZE: ALGORITMO DI CLUSTERING K-MEANS CON COSINE SIMILARITY

Come prima considerazione, dobbiamo tener presente che l’algoritmo di clustering K-MEANS utilizza la distanza Euclidea (che è diversa alla similarità del coseno) per determinare la somiglianza degli utenti attraverso la distanza tra i punti e i centroidi dei clusters.

Immagine che contiene Carattere, testo, bianco, diagramma

Descrizione generata automaticamente

* Distanza Euclidea

Immagine che contiene testo, Carattere, bianco, schermata

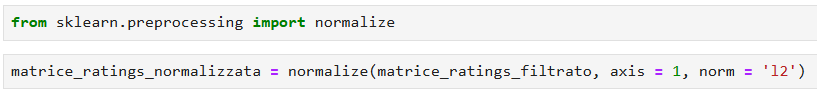
Descrizione generata automaticamente

* Cosine Similarity

Per utilizzare la similarità del coseno, al posto della distanza Euclidea, bisogna prima normalizzare vettori così abbiamo che se |A| = 1 e |B| = 1 ALLORA:

* Cosine\_similarity (A, B) = A . B
* Distanza Euclidea (A, B) = =

Quindi abbiamo che la distanza Euclidea dipende dall’angolo tra i vettori e non più dalle loro lunghezze. Possiamo dire, dunque, che la distanza Euclidea diventa equivalente alla similarità del coseno.



Importiamo il modulo ‘normalize’ di Sklearn.preprocessing che avrà lo scopo di normalizzare ogni riga della matrice originale dei rating utilizzando la norma l2.

*Abbiamo scelto la matrice dei ratings originale e non la matrice dei ratings predetti in quanto potrebbe portare a distorsioni nel clustering.*

La normalizzazione con norma l2 trasforma ogni riga della matrice (corrispondente ad ogni utente) in un vettore di lunghezza unitaria per poter finalmente utilizzare la similarità del coseno.

Immagine che contiene testo, Carattere, linea, numero

Descrizione generata automaticamente

Dopo aver normalizzato tutta la matrice dei ratings l’obiettivo è trovare il numero ottimale k di clusters per l’algoritmo K-Means per segmentare gli utenti in gruppi con comportamenti simili.

Creiamo l’array “wcss” (Within-Cluster Sum of Squares) in cui appenderemo ad ogni iterazione il valore della somma delle distanze al quadrato tra ogni punto all’intero del cluster e il centroide del cluster stesso.

Attraverso un ciclo fitiamo un valore diverso di K clusters che va da 1 al 121 con step di 5 e appendiamo il valore del wcss tramite l’attributo “km.inertia\_”, cosi via per tutti i valori di K clusters applicati nel modello.

Immagine che contiene testo, linea, Diagramma, diagramma

Descrizione generata automaticamente

Infine, plottiamo tuti i valori della lista “wcss” al variare del numero K di clusters e applichiamo l’Elbow method per trovare il miglior K.

Come possiamo osservare, il gomito si evince con **K = 20** clusters in cui si verifica un cambiamento significativo nella pendenza della curva.

* Prima del gomito la WCSS diminuisce rapidamente.
* Dopo il gomito la WCSS diminuisce più graduale.

Ora, applichiamo l’algoritmo K-Means alla matrice dei ratings normalizzata per assegnare ciascun utente a uno dei K = 20 clusters.

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Per primo passaggio dobbiamo impostare i parametri del modello K-Means i quali sono i seguenti:

* N\_clusters = 20
* Init = ‘k-means++’ 🡪 Inizializza i centroidi in modo intelligente
* N\_init = 10 🡪 Ripete l’algoritmo 10 volte con diverse inizializzazioni e sceglie quella con WCSS più basso.
* Max\_iter = 300 🡪 Assegniamo un numero massimo di iterazioni per una singola esecuzione dell’algoritmo.
* Random\_state = 0 🡪 L’algoritmo prende lo stesso seed ad ogni iterazione cosi’ garantiamo clusters costanti.

Dopo aver impostato l’algoritmo K-Means con i parametri ottimali utilizziamo il metodo “fit\_predict()” per addestrare il modello sulla matrice dei ratings normalizzata con l’obiettivo di calcolare i centroidi dei clusters per poi assegnare, a ciascun utente, il cluster più vicino a lui.

La variabile *y\_means* fornirà le etichette dei cluster, indicando per ciascun utente a quale gruppo appartiene. Avrà la forma 🡪 [0, 17, 5, …, 9] con lunghezza = numero di utenti.

Infine, avendo segmentato gli utenti con successo creiamo un pandas DataFrame per visualizzare la relazione tra utente e cluster in cui avremmo due colonne ‘utente\_id’ e ‘cluster’.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, numero

Descrizione generata automaticamenteTramite il metodo ‘.value\_counts()’ possiamo vedere la frequenza degli utenti assegnati a ciascun cluster dopo l’applicazione dell’algoritmo K-Means con k = 20 clusters.

* Il **cluster 9** contiene il numero maggiore di utenti con 502 utenti. Ciò suggerisce che una porzione significativa di utenti ha preferenze molto simili rappresentate dal centroide di questo cluster.
* Il cluster 12 contiene il numero minore di utenti con solo 46 utenti.
* Ci sono diversi cluster (12, 19, 17, 7, 2, 1, 15, …) che hanno pochi utenti e ci sono diversi cluster (9, 18, 0, 14, 8, 6, 2, 5, …) che hanno molti utenti.

Possiamo osservare una **distribuzione non uniforme tra i cluster**: I Cluster hanno dimensioni variabili. Questa variazione significa che ci sono gruppi di utenti che hanno preferenze più specifiche/uniche rappresentati in cluster piccoli, così come, ci sono gruppi di utenti che hanno preferenze più comuni rappresentati in cluster grandi.

## CREAZIONE DELLA LISTA PERSONALIZZATA DI RACCOMANDAZIONI PER CIASCUN UTENTE

**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**

Dopo avere la matrice dei rating predetti per ogni utente e prodotto possiamo creare delle liste personalizzate di raccomandazioni in base alle predizioni dei rating.

In primo luogo, introduciamo la funzione “sort\_columns(row)” che ha il compito di ordinare le colonne (prodotti) di ogni riga (utente) in base al valore del rating predetto.

All’intero della funzione troviamo il metodo *sorted(a, b, c)* che ordina in ordine decrescente i prodotti in base al punteggio predetto e prende come parametri:

* Row.items() 🡪 restituisce una coppia (prodotto, rating)
* Key = lambda x : x[1] 🡪 Definisce la chiave di ordinamento, in questo caso vogliamo ordinare in base al **rating** quindi dovremmo prendere il secondo campo della coppia (x[indice = 1])
* Reverse = True 🡪 Indica che vogliamo un ordinamento decrescente

Infine, la funzione ritorna i nomi dei prodotti già ordinati in ordine decrescente in base al rating predetto.

Vogliamo applicare la suddetta funzione su tutte le righe (utenti) della nostra matrice dei rating predetti *res\_df.* Utilizziamo il metodo. apply() 🡪 res\_df.apply(sort\_columns, axis = 1) e creiamo un altro pandas DataFrame *rec\_lists.*

Come possiamo vedere dall’output di *rec\_lists* per ogni utente abbiamo una lista di prodotti consigliati, ad esempio, i top dei primi cinque prodotti consigliati per i primi tre utenti sono:

* Per l’utente “**AE7BV6IMNPZ3F266H7PXMH3BZQNQ**” i prodotti consigliati sono:

1. B000G1ALRU
2. B00006EXLQ
3. B00005OWEJ
4. B000002NHN
5. B00006J3UT

* Per l’utente “**AGTMZCWIWBH43TCW7UKG2YV2EKKA**” i prodotti consigliati sono:

1. B000002GKT
2. B000002UVZ
3. B000002O4I
4. B00005LNBH
5. B00000ICO0

* Per l’utente “**AGWDYYVVWM3DC3CASUZKXK67G6IA**” i prodotti consigliati sono:

1. B000002J0N
2. B01BGDXOFC
3. B0034CNSFU
4. B000NY17J6
5. B015EYVWPI

## FILLING MATRICE DEI RATING: MATRIX FACTORIZATION

Per applicare la tecnica del Matrix Factorization per riempire la matrice dei rating bisogna utilizzare l’algoritmo Singular Value Decomposition (SVD).

Quindi, come primo passaggio, bisogna importare SVD from surprise.

Immagine che contiene testo, Carattere, software, Pagina Web

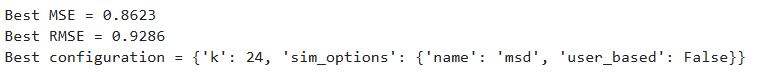
Descrizione generata automaticamente

Inizializziamo il modello SVD e dividiamo il dataset in un set di addestramento e un set di test utilizzando un rapporto di 80/20 come abbiamo fatto prima per il modello di K-NN.

Abbiamo ottenuto i seguenti valori:

* **MSE: 0.8168**
* **RMSE: 0.9038**

Bisogna tener conto che questi valori non sono ancora stati ottimizzati in quanto non abbiamo ancora trovato la configurazione ottimale dell’algoritmo SVD.

Possiamo, però, confrontare i valori di MSE e RMSE tra l’algoritmo KNN ottimizzato e l’algoritmo SVD non ancora ottimizzato:

* Il modello SVD ha un MSE leggermente più basso rispetto al KNN.
* Anche in termini di RMSE, il modello SVD mostra una performance superiore rispetto al KNN ottimizzato.

Questo suggerisce che una migliore capacità predittiva di SVD anche senza ottimizzazioni; quindi, proseguiamo e cerchiamo di trovare la configurazione ottimale dell’algoritmo SVD e fare il filling della matrice dei rating.

### CONFIGURAZIONE OTTIMALE DELL’ALGORITMO SVD: METODO GRID SEARCH

Per trovare la configurazione ottimale dell’algoritmo SVD non possiamo farlo tramite l’elbow method in quanto l’obiettivo non è trovare il numero K di vicini bensì altri parametri che non possono essere analizzati de rappresentati tramite una curva al variare degli errori di MSE e RMSE.

Il miglior metodo per l’algoritmo SVD è il Grid Search.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

I parametri ottimali dell’algoritmo SVD che vogliamo ottimizzare sono:

* N\_factors 🡪 Numero di fattori latenti per modellare le interazioni tra utente e prodotti: Varia tra 80 e 160 con step 10.
* N\_epochs 🡪 Numero di interazioni per l’apprendimento. Varia tra 10 e 50 con step di 10.
* Biased🡪 I bias rappresentano tendenze osservati nei dati.

Tramite il grid Search abbiamo trovato la configurazione ottimale esaustiva:

* **N\_factors = 90**
* **N\_epochs = 30**
* **Biased = True**

E otteniamo i risultati ottimizzati di MSE e RMSE:

* **MSE: 0.8064**
* **RMSE: 0.8980**

### CONFRONTO DEGLI ERRORI MSE E RMSE: MODELLI OTTIMIZZATI K-NN E SVD

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Ora che abbiamo ottimizzato entrambi i modelli possiamo confrontare i loro valori degli errori MSE e RMSE:

* Il modello ottimizzato SVD ha un MSE e RMSE **leggermente minore** rispetto al modello KNN.
* Il modello SVD ottimizzato mostra una **performance più accurata** rispetto al modello KNN.

Sebbene KNN si basa sulla similarità diretta tra utente e prodotto questo modello potrebbe non essere adatto a dataset con molte celle mancanti come in questo caso, per questo motivo l’algoritmo SVD funziona meglio: permette di catturare pattern latenti complessi.

### TOP 10 PRODOTTI CONSIGLIATI PER L’UTENTE “AGTMZCWIWBH43TCW7UKG2YV2EKKA”

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Utilizzando lo stesso procedimento come con il modello KNN troviamo il top 10 prodotti consigliati per lo stesso utente: AGTMZCWIWBH43TCW7UKG2YV2EKKA.

Ci rendiamo conto che ogni prodotto di questo top è completamente diverso al top precedente calcolato con il modello KNN.

### FILLING DELLA MATRICE DI RATING: MATRIX FACTORIZATION

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Applichiamo la stessa logica per fare il filling della matrice dei rating così come abbiamo fatto con il modello KNN.

* Aggiungiamo uno zero se il prodotto è già stato valutato.
* Altrimenti, appendiamo la predizione del prodotto non ancora recensito tramite il metodo .*predict().*

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteOtteniamo la matrice fillata con i rating predetti per i prodotti non ancora valutati. Di conseguenza, procediamo a generare la lista degli n prodotti consigliati per ogni utente:

Possiamo osservare che tutte le liste sono completamente differenti alle liste generate tramite il modello KNN.

Il motivo per il quale possiamo evincere risultati diversi può essere al fatto della natura degli algoritmi in questione.

Da una parte, l’algoritmo KNN è un **modello locale** che si concentra sulle relazioni immediate tra utenti o prodotti, il che può essere più adatto in contesti in cui la similarità tra utenti/prodotti ha una grande importanza; Dall’altro lato, l’algoritmo SVD è un **modello globale** che cerca pattern generali nella matrice dei rating, il che può essere più efficace in scenari con dati sparsi (come nel nostro caso).

In sostanza, il modello SVD si è mostrato più adatto nel nostro caso in quanto la nostra matrice dei rating presentava dati molto sparsi, ecco perché abbiamo riscontrato valori di MSE e RMSE più bassi rispetto all’algoritmo KNN.

# Conclusioni

Abbiamo sviluppato un sistema di raccomandazione basato su **Collaborative Filtering** utilizzando due principali approcci: **K-Nearest Neighbors (KNN)** e **Matrix Factorization (SVD)**. Attraverso i sei step previsti, abbiamo analizzato, implementato e confrontato entrambi gli algoritmi, ottenendo i seguenti risultati principali.

1. Analisi Esplorativa dei Dati

L’analisi ha evidenziato una forte prevalenza di rating uguale a 5 (Quasi il 60%) e una minoranza di rating uguale a 2 (Il 4.2%). Inoltre, la matrice dei rating presentava una forte sparsita’: Più del 95% degli utenti e dei prodotti aveva meno di 10 recensioni, quindi, ha dovuto essere filtrato. Questo ha sottolineato l’importanza di scegliere algoritmi capaci di gestire dati sparsi.

1. Ottimizzazione dell’Algoritmo KNN

Tramite l’analisi delle combinazioni di similarità, del numero di vicini K e dalla modalità user-based e item-based, abbiamo individuato la configurazione ottimale: **Similarità: MSD, K = 24 e Item-based.** Questa configurazione ha ottenuto un **MSE: 0.6823** e un **RMSE: 0.9286.**

1. Filling della matrice dei rating e Raccomandazioni con KNN

Utilizzando il modello KNN, abbiamo riempito la matrice dei rating predicendo i valori mancanti e generato liste di raccomandazioni personalizzate per ogni utente. Questo approccio ha prodotto risultati adeguati anche se fortemente influenzati dalla similarità locale.

1. Segmentazione degli utenti tramite K-Means

Abbiamo segmentato gli utenti in base alle loro preferenze utilizzando l’algoritmo di clustering K-Means con cosine similarity utilizzando la matrice dei rating iniziale. Ci ha permesso di individuare **20 clusters** tramite l’elbow method in cui potevamo evincere la distribuzione non uniforme degli utenti tra i diversi clusters: Esistenza di gruppi di utenti con preferenze più specifiche e di gruppi di utenti con preferenze più comuni.

1. Matrix Factorization

Abbiamo applicato la Matrix Factorization tramite SVD, ottimizzando i parametri principali. La configurazione migliore ha ottenuto un **MSE di 0.8064** e un **RMSE di 0.8980**, superando il modello KNN in termini di accuratezza.

1. Confronto tra KNN e SVD

Confrontando i due approcci, il modello SVD si è dimostrato più preciso nella predizione dei rating mancanti (minori MSE e RMSE).

In sostanza, il sistema di raccomandazione sviluppato si è dimostrato più accurato utilizzando l’algoritmo SVD anziché l’algoritmo K-NN in quanto è più adatto a sistemi con dati scarsamente correlati (Come in questo caso in cui più del 95% dei prodotti/utenti aveva meno di 10 recensioni), grazie alla sua capacità di sfruttare informazioni latenti.