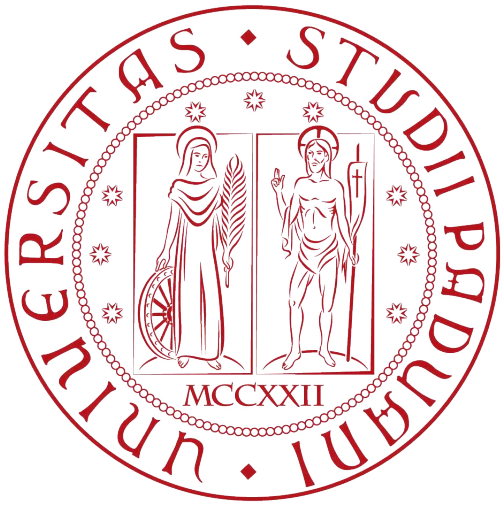


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Informatica

Sviluppo di un sistema di raccomandazione tramite algoritmi di machine learning

Relatore:
Prof. Gloria Beraldo

Laureando:
Giovanni Faedo
2013172

Anno Accademico 2023/2024

19/03/2024

A tutte le persone che mi sono state vicine, la mia famiglia, i miei amici...

Abstract

Al giorno d'oggi, a differenza del passato, è sempre più difficile reperire informazioni utili data l'enorme quantità di dati presenti in rete. E' per questo che nascono i **sistemi di raccomandazione**: strumenti software in grado di filtrare informazioni e fornire contenuti personalizzati per gli utenti. La seguente tesi ha lo scopo di fornire una panoramica sui sistemi di raccomandazione, con particolare attenzione sulle diverse tipologie di raccomandazione, i campi applicativi, vantaggi e svantaggi di ciascun modello e alcune metriche di valutazione. Successivamente nell'ambito di questa tesi, viene implementato un sistema di raccomandazione di cibi tramite le librerie Surprise e Scikit-Learn nel linguaggio di programmazione Python. Il codice viene testato con sei diverse tipologie di algoritmi di machine learning per verificare il corretto funzionamento del sistema e valutarne i risultati.

Indice

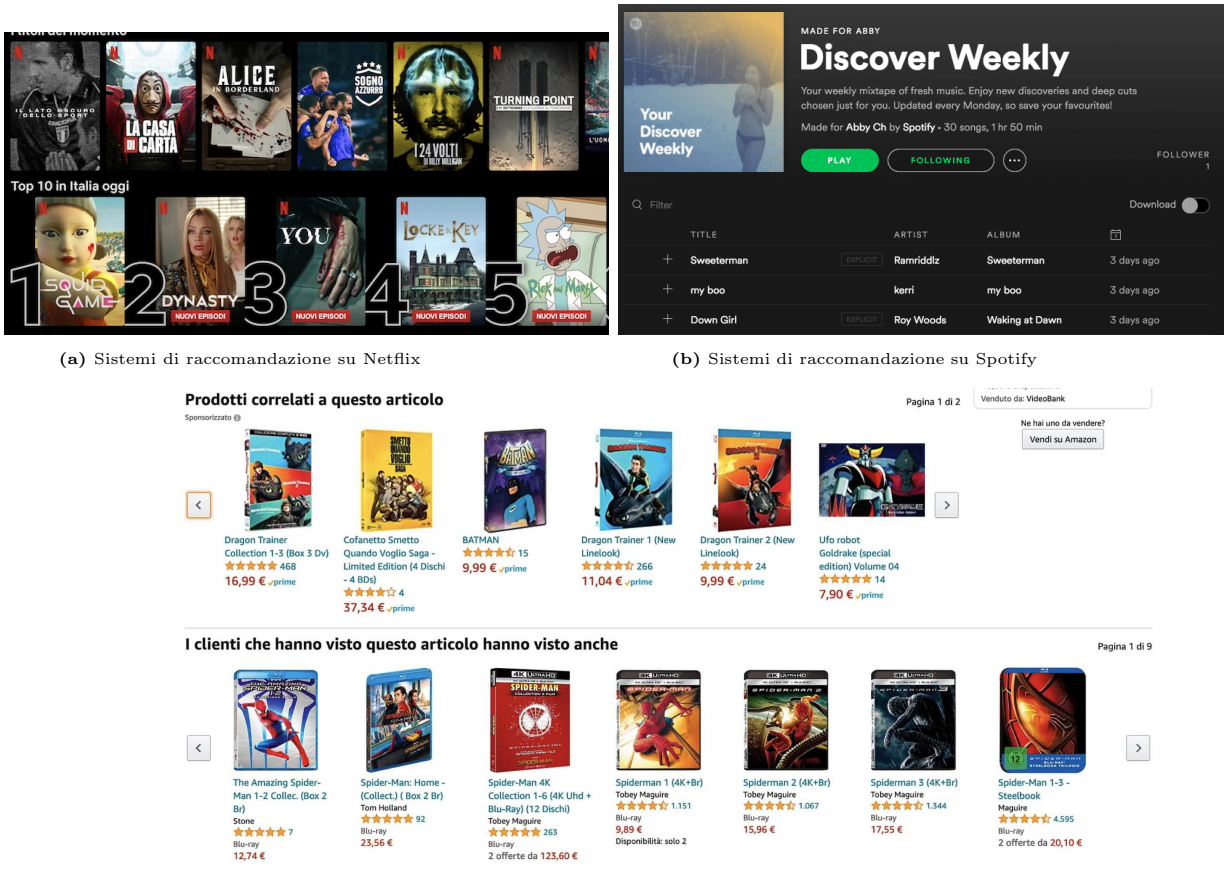
Capitolo 1

Introduzione

Prima dello sviluppo di Internet, trovare informazioni utili ad uno specifico argomento o contesto era molto complicato e richiedeva parecchio tempo data l'enorme quantità di dati. Al giorno d'oggi, fortunatamente, con lo sviluppo degli smart device e dei *Social Network Services (SNS)* questa enorme quantità di dati è immediatamente disponibile a tutti. Per quanto questo abbia diminuito di gran lunga il tempo di reperimento dei dati, ha aumentato il problema di sovraccarico dei dati (*data overload problem*): gli utenti spesso si trovano in difficoltà a cercare informazioni rilevanti e contenuti funzionali a causa del significativo aumento del traffico in rete [1]. Oltre a questo, le nuove piattaforme stanno aumentando la collezione di informazioni che possono memorizzare e identificare le preferenze di un utente. Ad esempio, la diffusione dei cosiddetti *wearable device* (tra cui gli Apple Watch) collegati agli smart device permette il salvataggio di sessioni di esercizi e parametri biometrici dell'utente. In questo modo Internet e smart device sono ambienti nei quali vengono collezionati non solo dati espliciti, forniti all'utente come likes e ratings, ma anche dati impliciti come click del mouse (*click stream data*) e informazioni implicite che controllano il comportamento dell'utente. E' proprio da quest'ultima tipologia di dati che si stanno sviluppando dei sistemi *cognitive-based* che analizzano la personalità dell'utente e il suo comportamento per identificare le sue preferenze. Per poter utilizzare dati espliciti ed impliciti per la raccomandazione di un prodotto, è necessario estrarre nuove conoscenze dai dati tramite varie tecniche di data mining. Questo procedimento, in particolare, consente di fornire raccomandazioni a un certo utente sia sulla base della storia passata di item selezionate sia sulla base dei feedback a seguito di una raccomandazione. Basandoci sui risultati di questi dati analizzati, diventa di fondamentale importanza il filtraggio della miriade di informazioni fornite per raccomandare

item personalizzate e conformi ai gusti dell'utente: questo è ciò che viene fatto dai **sistemi di raccomandazione**.

Un sistema di raccomandazione è uno strumento di filtraggio dei contenuti che permette di presentare informazioni personalizzate agli utenti sulla base delle loro scelte e delle loro preferenze. E' uno dei più comuni utilizzi degli algoritmi di machine learning nel mondo reale. Lo scopo principale di questi sistemi è quello di aiutare a ridurre lo sforzo degli utenti a ricercare informazioni rilevanti e utili in Internet, fornire suggerimenti utili in base a determinati feedback da parte degli utenti e raccomandare prodotti sulla base degli utenti simili. Al giorno d'oggi, i sistemi di raccomandazione hanno raggiunto una notevole importanza in molteplici contesti, dalla fruizione di contenuti su piattaforme di streaming (Figura 1.1a), nelle piattaforme musicali (Figura 1.1b) e negli e-commerce (Figura 1.1c), all'esplorazioni di notizie e contenuti online. L'obiettivo principale di un programmatore è quello di implementare un sistema di raccomandazione che riesca a inferire le preferenze dell'utente in maniera appropriata e coerente in base al servizio che sta utilizzando e alle sue preferenze.



(a) Sistemi di raccomandazione su Netflix

(b) Sistemi di raccomandazione su Spotify

(c) Sistemi di raccomandazione su Amazon

Figura 1.1: Applicazione dei sistemi di raccomandazione nelle principali piattaforme online

1.1 Obiettivi della tesi

Lo scopo principale della tesi è quella di approfondire i sistemi di raccomandazione e il loro funzionamento dapprima in ambito teorico e successivamente implementando un programma che, dato un insieme di cibi e dei ratings per diversi utenti riesca fornire delle raccomandazioni quanto più appropriate. In particolare verranno implementate due strategie di raccomandazione che si basano sul modello di raccomandazione *Content-Based Filtering*, che verrà illustrato nel Capitolo 2: il primo, *Key Content-Based* fornisce una serie di raccomandazioni in base a una serie di parole chiave fornite in input mentre il secondo, *User Content-Based* fornisce una serie di raccomandazioni in base agli items che un certo user ha già votato. A tale scopo, si analizzeranno le prestazioni in termine di suggerimenti, proposte e tempistiche basandosi su diversi algoritmi per il calcolo della similarità nei diversi items.

1.2 Struttura della tesi

La tesi è strutturata in sei capitoli. Nel Capitolo 2, viene spiegato più in dettaglio il funzionamento dei sistemi di raccomandazione, le varie tipologie di modelli di raccomandazione e le metriche per valutare le prestazioni dei modelli.

In seguito, nel Capitolo 3, vengono descritte nel dettaglio le librerie utilizzate per la simulazione del funzionamento dei modelli di raccomandazione (con dataset e funzionalità).

Nel Capitolo 4 viene spiegato nel dettaglio il caso studio preso in esame e vengono mostrati i dettagli implementativi tramite porzioni di codice in Python.

Successivamente nel Capitolo 5, vengono discusse le prestazioni dei 2 modelli implementati (*Key Content-Based* e *User Content-Based*) e vengono riportati i risultati ottenuti in termini di varietà nella raccomandazione e tempistiche dei diversi algoritmi testati.

Infine, nel Capitolo 6, vengono discussi i risultati ottenuti e possibili migliorie future dei sistemi presentati nella seguente tesi.

Capitolo 2

Design dei sistemi di raccomandazione

2.1 Funzionamento dei sistemi di raccomandazione

I sistemi di raccomandazione, come anticipato in precedenza, sono una tecnologia per il filtraggio di informazioni che fornisce una raccomandazione personalizzata agli utenti in un ambiente che può contenere o collezionare un numero molto elevato di dati. In particolare, i sistemi di raccomandazione trattano due entità principali: **users** e **items**, dove ogni user assegna un rating (o un valore di preferenza) a un item (o prodotto) [2]. I ratings degli utenti sono generalmente collezionati usando diversi metodi impliciti ed espliciti:

- **Ratings impliciti**: sono ratings che vengono collezionati indirettamente dall'utente attraverso le interazioni dell'utente con un certo item o con una collezione di items. Ad esempio, un sito web potrebbe collezionare ratings impliciti per diverse items basandosi sui click del mouse (*clickstream data*) oppure sulla quantità di tempo che un certo utente rimane nella stessa pagina web.

- **Ratings espliciti**: sono valori di preferenza che sono assegnati in modo diretto dall'utente scegliendo un valore da una scala finita di valori. Ad esempio, il sito web di prenotazione Booking richiede un punteggio da 1 a 5 stelle per valutare il servizio fornito all'utente.

Tutti i feedback e i ratings (sia impliciti che espliciti) forniti dall'utente vengono memorizzati in una matrice user-item che prende il nome di **utility matrix**. Un esempio di utility matrix è presentata in Figura 2.1.
La *utility matrix* spesso contiene dei valori mancanti o non chiari. Consideriamo il caso in cui, in una piattaforma di streaming di film, alcuni utenti potrebbero non aver valutato film che hanno visto, oppure potrebbero aver assegnato una valutazione ambigua (ad. esempio *"Questo film non è male ma..."*). I sistemi di raccomandazione si preoccupano di risolvere questo problema: trovare i valori mancanti o ambigui nella utility matrix.

	Item 1	Item 2	Item 3	Item 4
User 1	3	–	2	–
User 2	–	4	–	2
User 3	4	–	3	–
User 4	–	5	4	1

Figura 2.1: Utility matrix, da paper [2] a pag. 2

Questo compito, specialmente nelle prime fasi di un processo di raccomandazione, è molto difficile per il sistema sia perché è presente ancora un numero troppo basso di utenti sia perché questi utenti valutano solo un numero limitato di items.

2.2 Modelli di raccomandazione

Nei sistemi di raccomandazione, il processo di filtraggio dell'informazione dovrebbe essere adattato alle preferenze dell'utente per consigliare solo gli items che vengono ritenuti utili per l'utente. Sundaraja Sitharama Iyengar, nel documento [8], afferma che gli utenti preferiscono avere diverse opzioni di selezione ma allo stesso tempo, quando la difficoltà della selezione aumenta allora la soddisfazione della selezione diminuisce [1]. Per aumentare il grado di soddisfazione degli utenti che utilizzano sistemi di raccomandazione, è necessario ampliare il più possibile la varietà di scelta degli item, sempre rispettando i gusti e le caratteristiche dell'utente in modo da evitare il *selection overload* ovvero quando l'utente è esposto a un numero talmente elevato di opzioni che viene compromessa la sua capacità di prendere scelte che siano coerenti con i propri gusti personali.

Come si può vedere nella Figura 2.2 sono presenti tre diversi tipi di modelli di raccomandazione principali:

- **Content-Based Filtering**
- **Collaborative Filtering**
- **Hybrid Filtering**

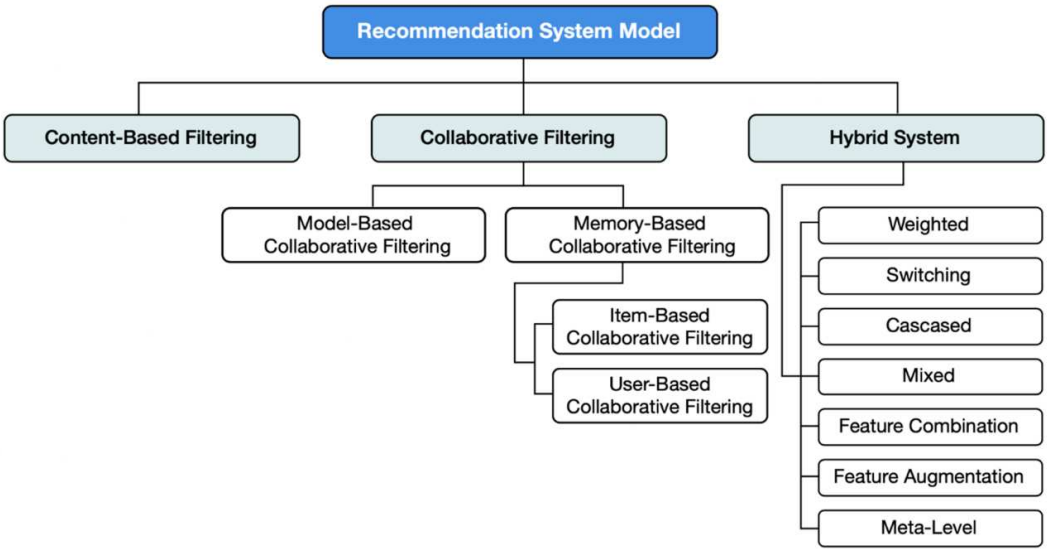


Figura 2.2: Panoramica dei diversi modelli di raccomandazione, da paper [1] pag. 9

2.2.1 Content-Based Filtering

L'approccio Content-Based Filtering nasce per la prima volta nel 1992 da uno studio di Loeb [5]. L'idea di base di questa metodologia è quella di raccomandare item con attributi simili ad altri item che hanno ottenuto like da parte dell'utente. Tutta la collezione di item viene raccolta in differenti **item profiles** basati sulla descrizione delle loro caratteristiche. Quando un utente assegna un rating positivo a un certo item, gli altri items presenti nell'item profile vengono aggregati assieme per costruire uno **user profile**, che combina tutti gli item profile che hanno ricevuto un rate positivo da parte dell'utente. Tutti gli item che sono presenti in questo *user profile* sono raccomandati all'utente, come viene mostrato in Figura 2.3.

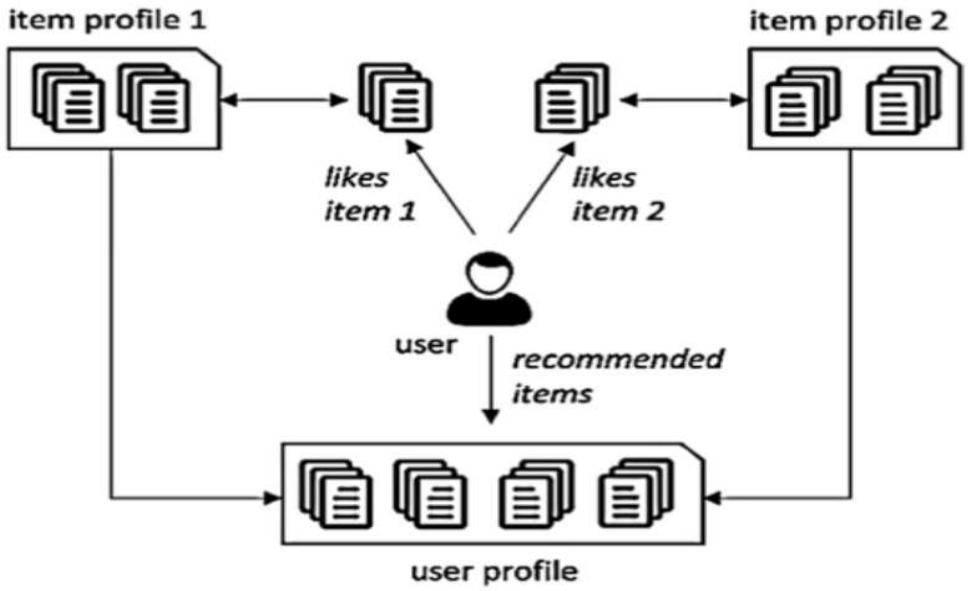


Figura 2.3: Content-Based Filtering, da paper [2] pag. 4

Pro e contro del Content-Based Filtering

Il modello sopra descritto ha diversi vantaggi e svantaggi nella raccomandazione. Uno dei principali vantaggi che questo modello offre è la capacità dinamica di adattarsi nel momento in cui le preferenze di un certo utente cambiano molto frequentemente. Uno dei principali punti di debolezza di questa tipologia di sistema sta nel fatto che il modello richiede una conoscenza approfondita delle caratteristiche degli item, che non è sempre disponibile. Inoltre, questo approccio, ha una limitata capacità di espandere le possibili scelte dell'utente.

2.2.2 Collaborative Filtering-Based

Collaborative Filtering-Based è un modello di filtraggio di informazioni apparso per la prima volta nel 1990 [6]. Questo approccio costruisce un database con le preferenze dell'utente utilizzando le sue valutazioni per predire items che siano conformi ai gusti dell'utente e in seguito li usa per la raccomandazione. Questo sistema usa il concetto di similarità tra gli utenti: il sistema inizia trovando un gruppo o una collezione di utenti X nel quale le preferenze, like o dislike sono simili all'utente da raccomandare A. In questo processo X viene chiamato **neighborhood** di A. I nuovi item che ottengono like dalla maggior parte degli utenti presenti in X, sono raccomandati all'utente A. Questo modello può essere suddiviso in due gruppi distinti: Memory-Based Collaborative Filtering e Model-Based Collaborative Filtering.

Memory-Based Collaborative Filtering

L'approccio Memory-Based raccomanda una nuova item al *target user A* tenendo in considerazione della somiglianza tra users o items. Questo approccio sfrutta le valutazioni passate degli utenti o delle interazioni degli utenti con gli elementi per predire le preferenze di un utente per gli elementi che non ha ancora valutato. Può essere a sua volta suddiviso in:

- **User-Based Collaborative Filtering**: è un modello che è basato sulla similarità tra utenti. Il rating che un utente assegnerà a un nuovo item viene calcolato considerando le valutazioni che hanno assegnato gli utenti presenti nel neighborhood per lo stesso item. Se il nuovo item ha ottenuto valutazioni positive dai neighborhood viene raccomandato all'utente, altrimenti no. Un esempio è riportato nella Figura 2.4

- **Item-Based Collaborative Filtering**: è un modello in cui viene costruito un oggetto di nome *item-neighborhood*, ovvero tutti gli items simili che l'utente ha già visto e valutato precedentemente. In questo modo il rating che l'utente assegnerà a un nuovo item viene predetto calcolando la media pesata (nella maggior parte dei casi tutti gli items hanno peso unitario) di tutti i rating presenti in un item-neighborhood simile al nuovo item (come viene mostrato in Figura 2.5)

Il modello Memory-Based viene molto utilizzato nelle piattaforme di e-commerce come ad esempio Amazon. Tuttavia, se il modello non include abbastanza dati o rating si potrebbero verificare alcuni problemi nel processo di raccomandazione.

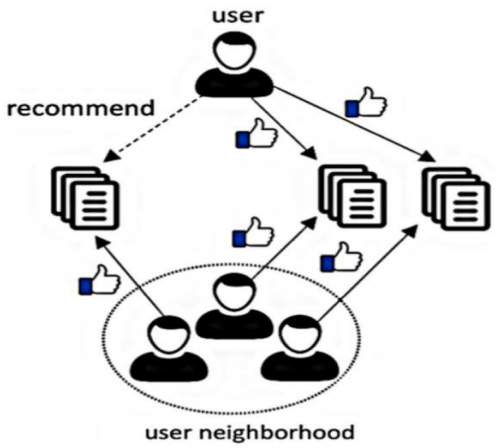


Figura 2.4: User-Based Collaborative Filtering, da paper [2] pag.5. Nella Figura viene mostrato il processo di raccomandazione basato sulla similarità tra gli users.

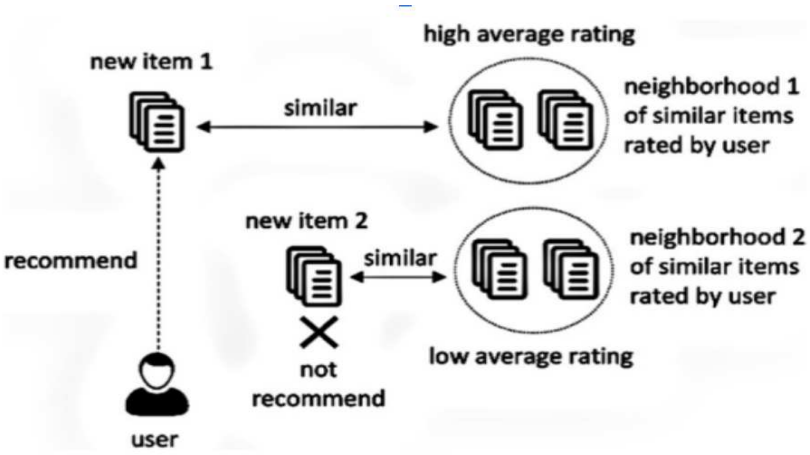


Figura 2.5: Item-Based Collaraborative Filtering, da paper [2] pag. 5. Nella Figura viene mostrato il processo di raccomandazione nel caso venga calcolata la similarità tra gli items.

Model-Based Collaborative Filtering

Il modello Model-Based sfrutta molteplici tecniche di data-mining e machine learning per sviluppare un modello per predire il rating dell'utente per una nuova item o un item senza valutazione. A differenza del precedente, che si basava sul concetto di similarità, questo modello sfrutta un processo di apprendimento automatico per estrarre caratteristiche e relazioni nei dati di training. La sostanziale differenza rispetto al modello precedente è che questo sistema permette di creare buone raccomandazioni anche per utenti non presenti nel dataset.

Pro e contro del Collaborative Filtering-Based

Rispetto al Content-Based Filtering un vantaggio del Collaborative Filtering-Based è che non richiede alcuna conoscenza delle caratteristiche degli item per

fornire una raccomandazione accurata. Inoltre, come anticipato sopra, permette di raccomandare item anche a nuovi utenti e questo permette di espandere le conoscenze di ciascun utente, in modo da ottenere suggerimenti che siano conformi ai propri gusti in base alla valutazione di utenti simili.

2.2.3 Hybrid Filtering

L'approccio hybrid è un'aggregazione di due o più tecniche descritte in precedenza e viene usato per risolvere le limitazioni di entrambi i modelli. Lo scopo principale di questo approccio è quello di compensare la mancanza di ratings nelle fasi iniziali della raccomandazione integrando le informazioni del modello Content-Based Filtering con il Collaborative-Filtering Based. In altre parole si suddivide il processo di raccomandazione in due parti:

- Content-Based: calcola le caratteristiche degli items e trova dei candidati item simili.
- Collaborative-Based: calcola la similarità tra utenti e trova un gruppo di utenti simili.

Gli output di queste due fasi vengono uniti e viene generata la raccomandazione secondo il modello Hybrid. La Figura 2.6 mostra un riassunto del processo di raccomandazione di un item da parte del modello Hybrid.

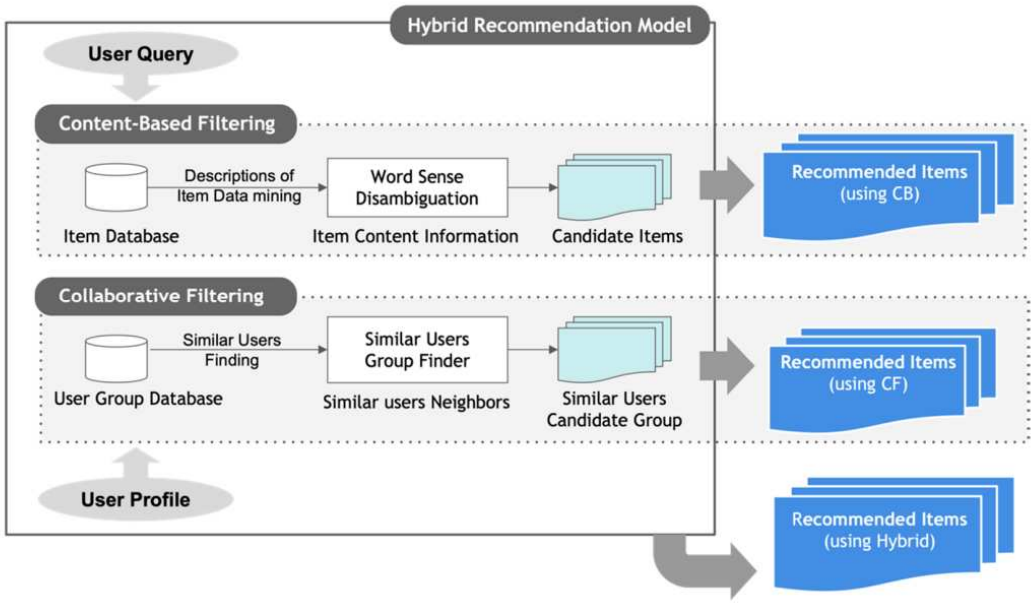


Figura 2.6: Processo di raccomandazione da parte del modello Hybrid, da paper [1] pag.9

2.3 Metriche di valutazione

Per essere in grado di valutare le performance di un modello di raccomandazione, è necessario essere in grado di quantificare e determinare i compiti che il sistema deve fare e come li deve fare. Il modo più semplice per valutare le performance di un sistema di raccomandazione è il **Root Mean Squared Error (RMSE)**. Il RMSE è un indice che viene comunemente utilizzato per valutare metriche quali precisione e accuratezza ed è ottenuto calcolando la radice quadrata dell'errore quadratico medio (MSE), calcolato dividendo la somma dei quadrati della differenza tra il rating attuale (*true rating*) e il rating predetto dal sistema per il numero totale di rating che sono stati predetti. La formula appena descritta è la seguente:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i - \hat{r_i})^2} \tag{2.1}$$

dove:

- N è il numero totale di valutazioni nei dati di test.
- $\hat{r_i}$ è la valutazione predetta per l'elemento i dal sistema di raccomandazione.
- r_i è la valutazione effettiva data dall'utente (il cosiddetto *true rating*).

Altre metriche di valutazione principali nei sistemi di raccomandazione sono presenti nella Tabella 2.1.

La **confusion matrix** viene utilizzata per valutare l'indice di qualità di un sistema di raccomandazione.
La Tabella 2.2 è un esempio di confusion matrix che mette in paragone il valore predetto con il valore attuale per misurare la performance di un sistema di raccomandazione. Ogni riga mostra se un item riflette le preferenze dell'utente o meno e ogni colonna indica se il modello di raccomandazione ha raccomandato o meno il determinato item.

Il valore *True Positive* (TP) della matrice indica il numero di items che corrispondono e sono conformi alle preferenze reali dell'utente quando il sistema raccomanda quella determinata item. In altre parole, indica le item che piacciono all'utente e che vengono raccomandate dal sistema.

Metrica di valutazione	Equazione	Definizione
Precision	$\frac{TP}{TP+FP}$	Il rapporto di items che sono conformi ai gusti dell'utente tra tutti gli item che sono raccomandate all'utente
Recall	$\frac{TP}{TP+FN}$	Il rapporto di items che sono coerenti con i gusti dell'utente tra tutti gli items
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Il rapporto di riferimenti con successo sul totale dei riferimenti
F-Measure	$2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$	Valore medio armonico dei valori di Precision e Recall
ROC Curve	$TPRate = \frac{TP}{TP+FN}$ $FPRate = \frac{FP}{FP+TN}$	E' un grafico che mostra il tasso dei veri positivi in funzione del tasso dei falsi positivi. Il tasso dei veri positivi (TPR) è anche noto come sensibilità o probabilità di rilevazione mentre il tasso dei falsi positivi (FPR) è anche noto come fall-out o probabilità di falsi allarmi [10].
AUC	Area sotto la ROC Curve	Misura la probabilità che un item rilevante casuale sia valutato in modo più alto di un altro item non rilevante

Tabella 2.1: Insieme di metriche nei sistemi di raccomandazione

Preferenza dell'utente	Raccomandato	Non raccomandato
User-preferred item	True Positive (TP)	False Negative (FN)
User-non-preferred item	False Positive (FP)	True Negative (TN)

Tabella 2.2: Confusion matrix che valuta le prestazioni di un sistema di raccomandazione

Il valore *True Negative* (TN) rappresenta il caso in cui il sistema ha evitato di fare una raccomandazione che non sarebbe piaciuta all'utente.
Il valore *False Positive* (FP) rappresenta il numero di casi nei quali il sistema di

raccomandazione suggerisce item che non sono conformi con i gusti dell'utente. Per concludere, *False Negative* (FN) si riferisce al numero di casi nei quali il sistema non fornisce una raccomandazione per items che piacciono all'utente.
Un sistema di raccomandazione efficiente dovrebbe quindi:

- raccomandare un item che piace all'utente (caso True Positive)
- non raccomandare un item che non piace all'utente (caso True Negative)

Un esempio grafico di quanto descritto nella Tabella 2.2 è possibile visualizzarlo nella Figura 2.7.

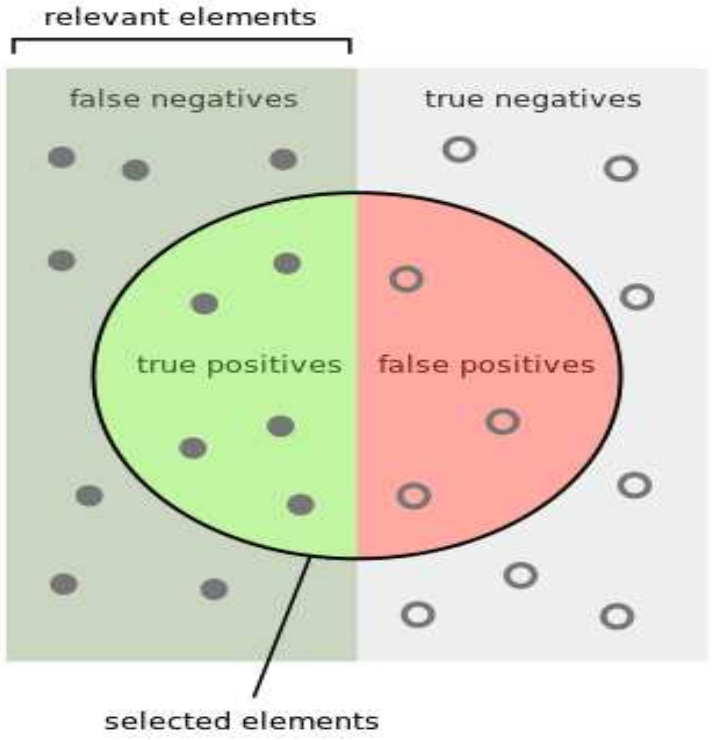


Figura 2.7: La figura mostra i parametri descritti sopra. I *relevant items* sono tutti gli item che l'utente preferisce mentre i *selected items* sono tutti gli item che il sistema seleziona per la raccomandazione.
Fonte: Wikipedia (https://en.wikipedia.org/wiki/Precision_and_recall)

Tra le metriche di valutazione specificate sopra, le più comunemente utilizzate sono le seguenti.

Accuracy

L'*accuracy* (accuratezza) indica la frazione di raccomandazioni con successo su tutti gli item raccomandati.

Precision

La *precision* viene calcolata dalla proporzione di item che sono conformi alle preferenze dell'utente sul totale di item che il modello ha raccomandato.

Recall

La *recall* è una metrica che indica quanto bene il sistema di raccomandazione riesce a suggerire item che potrebbero piacere a un determinato utente sulla base delle sue preferenze.

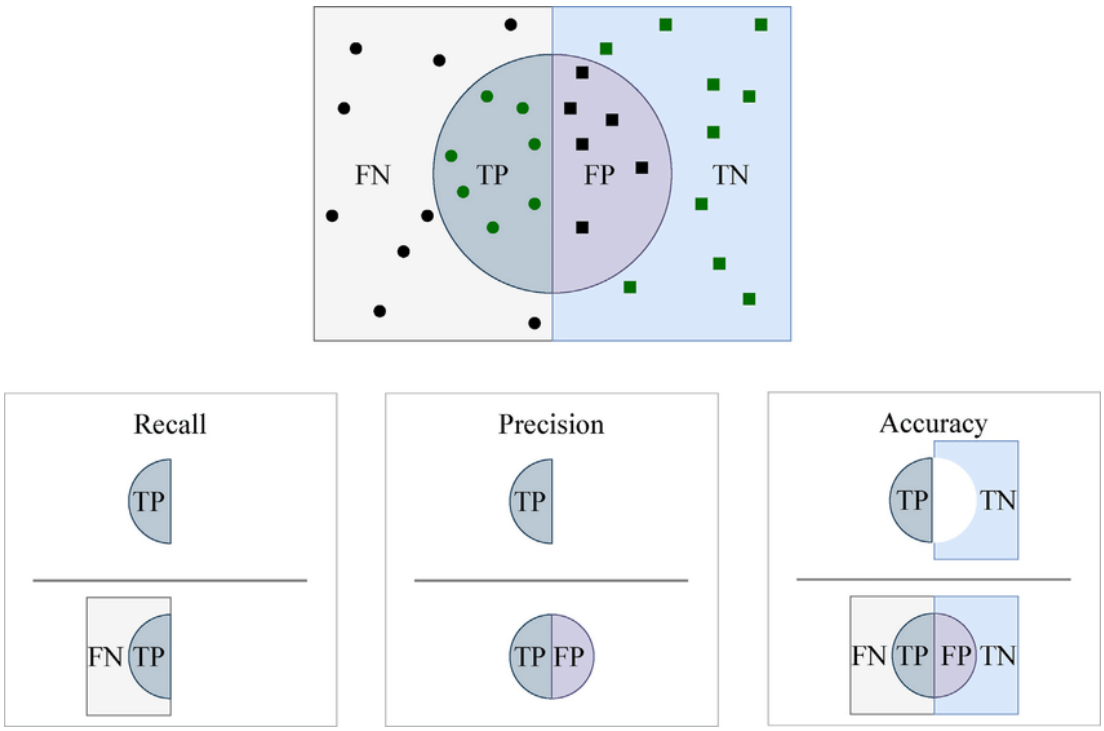


Figura 2.8: Esempio delle metriche di valutazione principali
Fonte: https://www.researchgate.net/figure/Visualizing-accuracy-recall-aka-sensitivity-and-precision-which-are-the-common$_fig3_346129022$

2.4 Tecniche di raccomandazione e algoritmi

Il **data mining** è una particolare tecnica che permette di ottenere informazioni utili tramite la ricerca di correlazioni e pattern tra i dati nel processo di analisi di una grande quantità di dati. Nei sistemi di raccomandazione vengono utilizzate

varie tecniche di data mining. Tra le principali vi sono il Text Mining, in cui vengono estratte informazioni tramite il significato delle parole, il K-Nearest Neighbors, in cui viene usato il concetto di similarità per fornire raccomandazioni, il Clustering che suddivide i dati in gruppi di user e item simili chiamati cluster e le Reti Neurali (Neural Network) che vengono usate per modellare pattern complessi nelle raccomandazioni e migliorare le prestazioni dei sistemi stessi. Nelle sezioni sottostanti vediamo questi 4 approcci più nel dettaglio.

2.4.1 Text Mining

Il *text mining* è una tecnica che permette di scoprire l'utilità di informazioni testuali cercando di trovare una relazione tra i dati. In questo ambito acquisisce notevole importanza la semantica: cresce la tendenza di analizzare un testo basandosi sulla frequenza delle parole o su come queste sono accostate e collegate tra loro. Il text mining viene utilizzato per raccomandare items simili tramite un processo di analisi della semantica degli items nei modelli di raccomandazione che utilizzando un approccio Content-Based Filtering. Nel Collaborative Filtering, invece, si valuta la conoscenza semantica delle informazioni da parte degli utenti per poter raccomandare items con una certa similarità.

2.4.2 K-Nearest Neighbors (KNN)

Il *K-Nearest Neighbors* (KNN) è un algoritmo che si basa sulla ricerca dei K vicini più prossimi di una specifica istanza di test all'interno del training set. KNN classifica il dataset basandosi sul concetto di distanza, calcolata in termini di **similarità**: minore è il valore di distanza maggiore sarà il grado di similarità. Alcune delle metodologie per comparare la similarità sono ad esempio:

- **Euclidean distance**: l'insieme di user e item vengono rappresentati come vettori in uno spazio di caratteristiche dove ogni vettore corrisponde a una particolare tipo o attributo. Ad esempio se abbiamo due utenti $A = (a_1, a_2, a_3)$ e $B = (b_1, b_2, b_3)$ la loro distanza euclidea viene calcolata con la seguente formula:

$$\textbf{Euclidean Distance (A,B)} = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2} \tag{2.2}$$

  Più il valore di distanza è piccolo più i due utenti considerati hanno caratteristiche e gusti simili.

- **Cosine similarity**: in questo caso l'idea è simile alla distanza euclidea ma viene calcolato il coseno dell'angolo tra i due vettori nello spazio delle caratteristiche. La cosine similarity tra due vettori A e B viene calcolata nel seguente modo:

$$\textbf{Cosine similarity (A,B)} = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{2.3}$$

  in cui:

  - il numeratore indica il prodotto scalare tra A e B
  - il denominatore indica il prodotto scalare tra le norme di A e B

  La Formula 2.3 produce un valore compreso tra -1 e 1 dove:

  - 1 indica che i due vettori sono perfettamente simili
  - 0 indica che i due vettori sono perfettamente diversi (non hanno nulla in comune)
  - -1 indica che i due vettori sono completamente diversi (massimo grado di dissimilarità)

- **Pearson correlation**: viene utilizzato per misurare la correlazione lineare tra due vettori di dati. La Pearson Correlation tra due vettori X e Y viene misurata nel seguente modo:

$$\textbf{Pearson Correlation (X, Y)} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \cdot (Y_i - \bar{Y})^2}} \tag{2.4}$$

  dove:

  - n è il numero di elementi nel vettore
  - X_i e Y_i sono le preferenze dell'utente i nei vettori $\mathbf{X}$ e $\mathbf{Y}$ rispettivamente
  - $\bar{X}$ e $\bar{Y}$ sono le medie dei valori nei vettori $\mathbf{X}$ e $\mathbf{Y}$ rispettivamente

2.4.3 Clustering

Clustering è un algoritmo che identifica gruppi finiti di categorie o cluster per descrivere i dati e viene largamente usato nei sistemi di raccomandazione perché permette di ottenere bassa ridondanza e ambiguità nelle informazioni. Tra gli

algoritmi più utilizzati vi è il *K-means clustering*: un algoritmo che suddivide un insieme di dati in gruppi omogenei, in modo che tutti gli elementi all'interno dello stesso *cluster* siano simili tra loro. Dopo aver selezionato il numero K di cluster da formare, vengono selezionati K punti di centro per ogni cluster. In questo modo ogni punto dati viene assegnato al cluster il cui punto di centro è più vicino al punto dati fra tutti quanti i centri, secondo la seguente formula:

$$\text{Cluster assegnato} = argmin_i Distanza(\text{punto dati}, \text{centro}_i) \tag{2.5}$$

Dopo aver assegnato tutti i punti dati ai cluster, viene effettuato l'aggiornamento dei punti di centro, calcolati come la media delle coordinate di tutti i punti assegnati a quel cluster.

$$\text{Nuovo punto centro} = \frac{1}{N_{cluster}} \sum_{\text{punti all'interno del cluster}} punto \tag{2.6}$$

Questa tecnica viene usata principalmente nel modello Collaborative Filtering.

2.4.4 Neural Network

Negli ultimi anni l'uso delle reti neurali si è diffuso in svariati campi come, ad esempio, il riconoscimento di voce e immagini, la ricerca di foto, la traduzione automatica. Le reti neurali sono largamente utilizzate nei sistemi di raccomandazione perché permettono di ottenere dati aggiuntivi in situazioni nelle quali è difficile capire le preferenze degli utenti basate sulle scelte passate. Nei moderni sistemi di raccomandazione vengono usate le reti neurali profonde (*Deep Neural Network*) per modellare i dati ottenuti tramite i feedback impliciti (es. click del mouse). Queste tipologie di tecniche vengono utilizzate principalmente per risolvere il problema dello sparsity, in cui il sistema si trova a dover analizzare un numero troppo elevato di dati e il cold start, ovvero quando il sistema non è in grado di fornire delle raccomandazioni in modo efficiente a causa di un numero basso di rating.

2.5 Criticità e problemi

In questa sezione vengono descritti i principali problemi e criticità di cui soffrono i sistemi di raccomandazione, con una possibile soluzione a ciascuno di essi.

2.5.1 Cold start problem

Il *cold start problem* si riferisce a una condizione che si verifica quando il sistema non è in grado di produrre raccomandazioni efficienti e di qualità a causa dell'ingresso nel sistema di nuovi utenti che non hanno valutato nessun item o un numero molto basso di item. Generalmente sorge nel momento in cui un nuovo utente o un nuovo item viene inserito all'interno del database. Una possibile soluzione a questo problema è quella di chiedere esplicitamente a un nuovo user di fornire le loro preferenze per gli item già presenti oppure di assegnare i rating all'inizio, ad esempio nella fase di registrazione a una piattaforma online oppure nel corso di un tutorial introduttivo.

2.5.2 Shilling attack problem

Questo problema si verifica nel momento in cui un *malicious user* falsifica la sua identità e corrompe il sistema assegnando rating falsi agli item. Questa tipologia di problema condiziona negativamente l'affidabilità del sistema. Una delle possibili soluzioni al problema è quella di identificare i possibili utenti fake e rimuoverli dal sistema.

2.5.3 Synonymy problem

Questo problema accade quando item simili per contenuto e caratteristiche hanno nomi diversi o quando addirittura lo stesso item viene salvato con due o più nomi nel sistema. Questo mette molta confusione al sistema che ha notevole difficoltà a distinguere le differenze tra due items simili e riduce drasticamente l'accuratezza del modello. Per risolvere questo problema sono utilizzati alcuni metodi come ad esempio il *demographic filtering*, *automatic term expansion* e il *Singular Value Decomposition* (SVD).

- Demographic filtering: questa soluzione si basa sulle differenze demografiche degli users memorizzando nome, anno di nascita, sesso, posizione geografica. In questo modo il sistema di raccomandazione può risolvere il problema della sinonima basandosi sulla similitudine demografica degli utenti, ovvero verranno raccomandati, ad esempio, items simili a utenti dello stesso sesso, o dello stesso anno di nascita.

- Automatic term expansion: questa soluzione usa una serie di algoritmi per il calcolo della similarità tra i termini inseriti dagli utenti valutando la semantica tra le parole.

- Singular Value Decomposition (SVD): questa soluzione riesce a catturare in modo automatico relazioni semantiche nascoste tra termini inseriti e gestire la raccomandazione in modo più efficiente.

2.5.4 Latency problem

Questo problema è specifico all'approccio *Collaborative Filtering* e sorge nel momento in cui sono continuamente inserite nuove item nel database che non permettono al modello di raccomandare nuove item. Per risolvere questo problema il calcolo potrebbe essere fatto in un ambiente offline usando alcune tecniche di *clustering-based*.

2.5.5 Sparsity problem

Il problema della sparsità dei dati (*data sparsity*) sorge nei sistemi che analizzano una grande quantità di dati. Nel caso dei sistemi di raccomandazione questo accade nel momento in cui gli utenti attivi assegnano un rating solo a pochissime items. In questo modo il sistema di raccomandazione è poco accurato e potrebbe essere risolto tramite Singular Value Decomposition e usare tecniche Model-Based. Un esempio comune di tecnica Model-Based sono i Modelli Bayesiani che possono integrare conoscenze a priori sui dati e sulle distribuzioni dei parametri del modello per generare raccomandazioni più robuste.

2.5.6 Grey sheep problem

Il *grey sheep problem* (problema della pecora nera) è un problema molto comune nei modelli di raccomandazione che utilizzano l'approccio collaborative filtering. Questa tipologia di problema sorge nel momento in cui un feedback dato da un utente o da un gruppo di utenti non corrisponde a nessuna preferenza nei suoi neighborhood e causa parecchi problemi al sistema che si trova in difficoltà a raccomandare accuratamente una nuova item. L'approccio content-based permette di risolvere parzialmente questa tipologia di problema dato che la predizione viene fatta basandosi principalmente sulle caratteristiche dell'item.

2.5.7 Scalability problem

I sistemi di raccomandazione, generalmente, richiedono una grande quantità di dati da addestrare per fornire delle raccomandazioni accurate ed efficienti. Questo causa lo *scalability problem*, ovvero un problema che sorge nel momento in cui la quantità di dati da dover gestire cresce in modo molto velocemente (pensiamo ad esempio ad un utente che cambia dinamicamente e continuamente i propri ratings e le proprie preferenze rispetto a un determinato item). Due modalità possibili per risolvere questo problema è di ridurre la dimensionalità del sistema oppure utilizzare tecniche clusering-based per suddividere gli utenti in gruppi e considerare un gruppo per volta piuttosto che considerare ad ogni step l'intero database.

Capitolo 3

Metodi

3.1 Librerie

Sono presenti diverse librerie per l'implementazione e la valutazione di sistemi di raccomandazione. Di seguito riporto le principali:

3.1.1 Surprise

La libreria *Surprise*[1], detta anche *scikit-surprise*, è una libreria Python per lo sviluppo di modelli di raccomandazione. Viene utilizzata principalmente nei modelli di Collaborative Filtering e supporta:

1. **Singular Value Decomposition(SVD)**
2. **k-Nearest Neighbors(k-NN)**

Vi sono diversi tipi di algoritmi di predizione suddivisi in due categorie principali:

1. Baselines estimates (stimatori di base)

E' un metodo per migliorare le previsioni di un sistema di raccomandazione considerando le preferenze medie degli users e gli items. In un sistema di raccomandazione alcuni users tendono a valutare determinati items assegnando scores più alti o più bassi in base alle loro preferenze. Questa tecnica cerca di valutare queste tendenze medie tramite la formula:

$$b_{ui} = \mu + b_u + b_i \tag{3.1}$$

[1] https://surpriselib.com/

dove

- b_{ui} è la stima di base per l'user u e l'item i
- μ è la media globale delle valutazioni di tutti gli utenti e gli elementi nell'intero insieme di dati
- b_u è la deviazione media dell'user u rispetto alla media globale, in altre parole quanto uno user è incline a dare valutazioni più alte o più basse rispetto alla media delle valutazioni date dagli altri utenti. In sintesi, se b_u è positivo, l'utente tende a dare ratings che sono più alti rispetto alla media globale delle valutazioni date dagli altri utenti, mentre se b_u è negativo, l'utente tende a dare ratings che sono più bassi rispetto alla media globale delle altre valutazioni date dagli altri utenti.
- b_i è la deviazione media dell'item i rispetto alla media globale, in altre parole quanto un item è valutato più o meno rispetto alla media degli altri items presenti nel sistema di raccomandazione. Se b_i è positivo, allora l'item tende a ricevere voti che sono più altri rispetto alla media delle valutazioni per tutti gli elementi, mentre se b_i è negativo, allora l'item tende a ricevere voti che sono più bassi rispetto alla media di tutte le altre valutazioni degli items.

Gli stimatori di base sono utili per risolvere il problema del bias nei dati: minimizzano l'errore tra le valutazioni reali e le previsioni del modello. La stima complessiva $\hat{r}_{ui}$ è data quindi dalla seguente formula:

$$\hat{r}_{ui} = \mu + b_u + b_i + model_{ui} \tag{3.2}$$

dove $model_{ui}$ è la previsione di rating data dal modello. Gli stimatori di base possono stimare in due modi differenti:

- **Stochastic Gradient Descent (SGD)**: minimizzano l'errore quadratico medio (RMSE) tramite l'ottimizzazione dei parametri b_i e b_u. La funzione SGD aggiorna iterativamente i parametri in modo proporzionale al gradiente della funzione di costo rispetto ai parametri.
- **Alternating Least Squares (ALS)**: in questo caso i parametri di b_u e b_i vengono ottimizzati in modo alternato fino a che non si ottiene una convergenza. Iterativamente viene fissato il valore di b_u e viene ottimizzato

b_i e viceversa fino a che non si ottiene un valore dei due parametri che minimizza l'errore tra le previsioni del modello e le valutazioni effettive.

In Surprise è possibile calcolare gli stimatori di base tramite la classe `BaselineOnly` come viene mostrato nel seguente esempio:

```
from surprise import BaselineOnly

# Definizione del modello BaselineOnly utilizzando il metodo ALS (Alternating Least Squares) oppure SGD (Stochastic Gradient Descent)
baseline_model = BaselineOnly(bsl_options={'method': 'als', 'n_epochs': 5, 'reg_u': 12, 'reg_i': 5})
```

Listing 3.1: Utilizzo degli stimatori di base nella libreria Surprise

2. Similarity measure (misura di similarità)

Viene utilizzata per valutare quanto due users o items siano simili o vicini in base al loro comportamento o alle loro caratteristiche. Le due misure più utilizzate sono le seguenti:

- **Cosine similarity**
- **Pearson Correlation Coefficient**

In Surprise è possibile calcolare la misura di similarità nel seguente modo:

```
from surprise import similarities

# data e' un oggetto di tipo Dataset

# Calcolo della cosine similarity tra utenti
user_sim_cosine = similarities.cosine(data.build_full_trainset())

# Calcolo del pearson correlation coefficient tra utenti
user_sim_pearson = similarities.pearson(data.build_full_trainset())
```

Listing 3.2: Utilizzo della misura di similarità nella libreria Surprise

3.1.2 Scikit-learn

Scikit-learn[2] è una libreria di machine learning in Python che offre una vasta gamma di strumenti per la costruzione, l'addestramento e la valutazione di mo-

[2]https://scikit-learn.org/

delli di machine learning tra cui anche modelli di raccomandazione. In particolare la libreria include alcuni moduli specializzati nella manipolazione e nella trasformazione dei dati (`sklearn.preprocessing` e `sklearn.feature_extraction`), molto utili nei modelli di raccomandazione Content-Based e offre una serie di moduli per la valutazione del modello (`sklearn.model_selection`). Nello specifico, nei programmi seguenti, si sfrutta la classe `TfidfVectorizer` che converte una raccolta di testo o documenti di testo in una rappresentazione vettoriale TF-IDF (Term Frequency-Inverse Document Frequency).

3.1.3 Lenskit

Lenskit è una libreria open-source scritta in linguaggio Java che fornisce una piattaforma per lo sviluppo e la sperimentazione di algoritmi di raccomandazione. Il framework è stato rilasciato per la prima volta nel 2010 [9] e in seguito è nato Lenskit for Python (LKPY), creato per fornire funzionalità simili a quelle di Lenskit ma in un ambiente di programmazione Python. Alcune delle caratteristiche principali di LKPY sono le seguenti:

- **Supporto di diversi algoritmi**: Lenskit for Python supporta diversi algoritmi per modelli Content-Based e Collaborative-Filtering.
- **Gestione di feedback espliciti ed impliciti**: permette di lavorare con diversi tipi di sistema di raccomandazione tramite feedback impliciti o espliciti.
- **Funzionalità di valutazione**: LKPY offre una serie di strumenti per valutare le prestazioni di un sistema di raccomandazione.

3.1.4 Spotlight

Spotlight è una libreria che utilizza PyTorch per costruire modelli profondi e superficiali di raccomandazione. Viene utilizzato con due modelli principali:

- **Modelli di fattorizzazione**: sono una categoria di modelli che si occupano di estrarre informazioni utili dai dati. Uno degli obiettivi principali di questi modelli è quello di catturare informazioni nascoste o non chiare nelle relazioni user-item. Un esempio di modello di fattorizzazione è il Singular Value Decomposition (SVD).

- **Modelli sequenziali**: in questa tipologia di modelli la raccomandazione può essere vista come una sequenza di task di predizione. L'obiettivo è determinare il prossimo item con cui l'utente interagirà conoscendo tutte le items con cui ha interagito in passato.

3.2 Datasets

Ognuna delle seguenti librerie presentate sopra supporta un insieme di dataset predefiniti (*build-in*). Per comprendere al meglio le funzionalità di Surprise, in particolare, è stato utilizzato il dataset *MovieLens-100K*[3]. Questo dataset comprende:

- 100,000 ratings (da 1 a 5) dati da 943 utenti su 1683 films differenti.
- Ogni user ha votato almeno 20 film.
- Ogni utente ha associate delle informazioni personalizzate quali nome, età, genere, occupazione.

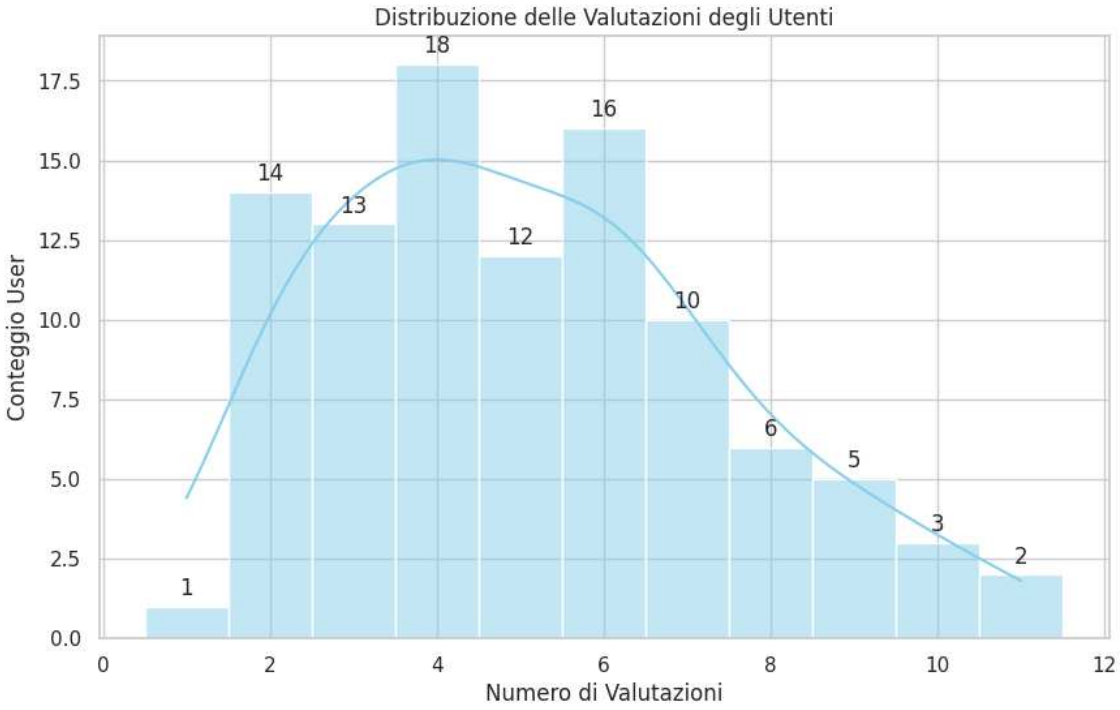
Il dataset utilizzato nell'implementazione dei codici presenti nel Capitolo 4 è stato scaricato dal sito kaggle[4]. Il **food dataset** è composto dai seguenti due file csv:

- **1662574418893344.csv** contenente la descrizione di cibi o piatti tipici con diversi parametri di caratterizzazione quali: identificativo del piatto (Food_ID), nome del piatto (Name), tipo del piatto (C_Type), vegano o non vegano (Veg_Non) e una descrizione del piatto (Describe).
- **ratings.csv** contenente un insieme di ratings dati da alcuni utenti a un determinato piatto. Il file è scritto nel formato User_ID — Food_ID — Rating.

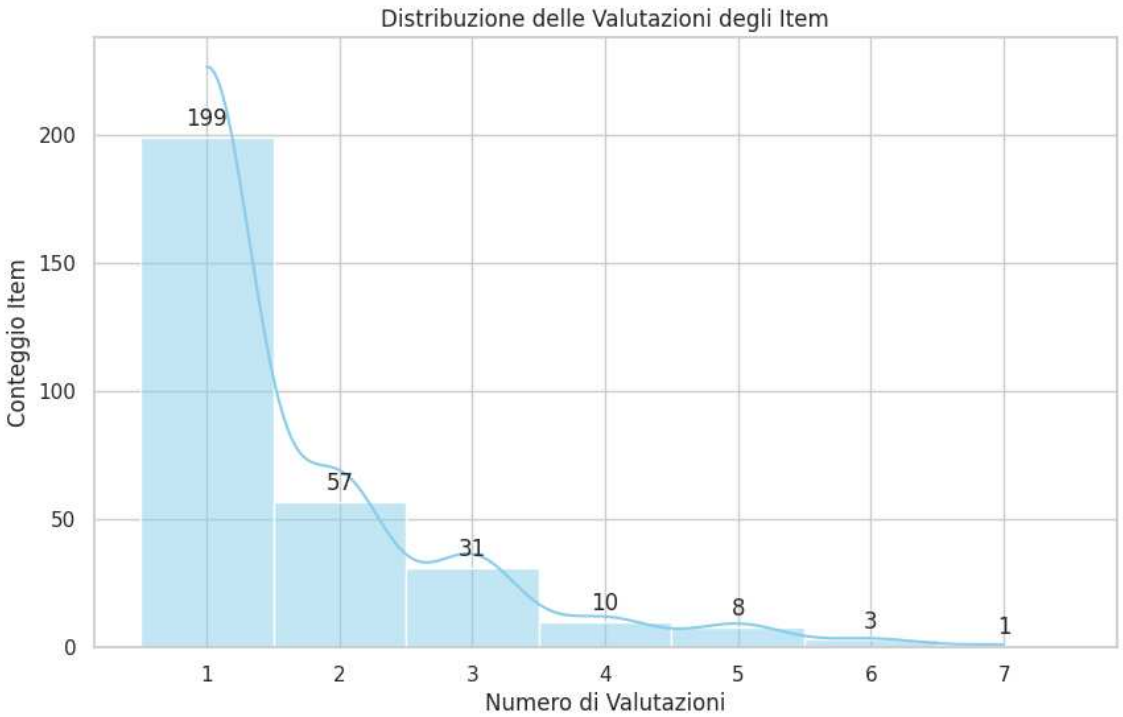
Nella Figura 3.1 viene descritta la distribuzione dei dati nel Food Dataset, in particolare:

- la Figura 3.1a descrive il numero di valutazioni date dagli utenti presenti nel dataset
- la Figura 3.1b descrive il numero di valutazioni assegnate agli items presenti nel dataset

[3]https://grouplens.org/datasets/movielens/100k/
[4]https://www.kaggle.com/datasets/schemersays/food-recommendation-system



(a) Istogramma che visualizza la distribuzione del numero di valutazioni e del numero di utenti che ha dato un certo numero di valutazioni



(b) Istogramma che visualizza la distribuzione del numero di valutazioni date ai diversi items

Figura 3.1: Distribuzione del numero di valutazioni del food dataset

Capitolo 4

Implementazione del caso studio

Il caso d'uso scelto per la seguente tesi è un sistema di raccomandazione per fornire suggerimenti alimentari. Il sistema proposto si basa sul modello **Content-Based Filtering** ed è stato implementato tramite le librerie `scikit-learn` e `surprise` in linguaggio di programmazione Python. Il codice dei programmi implementati è interamente disponibile in GitHub[1].

4.1 Content-Based Filtering con scikit-learn

Il seguente codice ha l'obiettivo di testare il modello Content-Based tramite l'utilizzo della libreria Scikit-Learn. In particolare il modello *Content-Based Filtering* viene testato in due modi differenti:

- **Content-Based con keywords (Key Content-Based)**: il software prende in input alcune parole chiave ed effettua una raccomandazione sulla base di queste
- **Content-Based con UserID (User Content-Based)**: il software prende in input il codice di un determinato user ed effettua una raccomandazione sulla base degli items a cui l'utente ha già dato un voto

[1] https://github.com/giova211001/Food-Recommender-System

4.2 Pseudocodice

Entrambi i programmi implementati svolgono le seguenti funzionalità principali:

- **Richiedere parametri in input**: nel caso di *Key Content-Based* vengono richiesti una serie di termini da includere ed eventualmente escludere mentre nel caso di *User Content-Based* viene richiesto il codice dell'utente da raccomandare ed eventualmente una tipologia di cibo preferita
- **Calcolare similarità tra gli items**: viene calcolata in entrambi i casi la similarità tra gli items per suggerire gli items più simili:
  - alle parole chiave inserite nel caso di *Key Content-Based*
  - agli items che l'utente ha già valutato nel caso di *User Content-Based*
- **Testare le raccomandazioni con diversi algoritmi di similarità**: per ogni algoritmo tra quelli descritti sopra viene calcolato il coefficiente di similarità e viene salvato in un dizionario contenente coppie item-valore di similarità che è poi ordinato in ordine crescente o decrescente (a seconda dell'algoritmo) per memorizzare le items più simili.
- **Fornire l'output**: vengono restituiti gli N items più simili per ogni algoritmo testato

Algorithm 1 Key Content-Based

Input: Insieme di parole chiave da considerare *keywords*
Input: Insieme di parole chiave da escludere *not present*(opzionale)
Input: Numero N di elementi da raccomandare
Output: Gli N elementi più simili sulla base agli input inseriti

description_list ← lista contenente l'insieme di descrizioni di tutti i cibi
keywords ← lista contenente le parole chiave inserite in input
not_present ← lista di parole chiave da non tenere in considerazione
algorithms ← lista contenente gli algoritmi da testare
dictionary ← dizionario contenente coppia item-similarità
sorted_dictionary ← dizionario ordinato in base al valore di similarità
intollerance ← intero che mi indica se ci sono cibi da escludere o meno
removed_food ← lista di cibi che vengono eventualmente rimossi
if intollerance == 1 **then**
    for *words* in *not_present* **do**
        aggiunto *food* in *removed_food*
        eliminato dal dataset *food*
    end for
end if
for *algo* in *algorithms* **do**
    similarity ← *algo*(description_list, keywords)
    sim_rows, sim_cols ← insieme di righe e colonne con valore diverso da zero
    sim_not_null ← insieme di valori di similarità diversi da zero
    for *rows*, *cols*, *value* in *sim_rows, sim_cols, sim_not_null* **do**
      Aggiungi Al Dizionario (dictionary, nome_item, valore di similarità)
    end for
    sorted_dictionary ← *sorted*(dictionary)
    sorted_dictionary ← *islice*(dictionary.items(), N)
    for *key*, *value* in dictionary.items() **do**
      stampa a video la coppia di *chiave* (nome del cibo) - *valore* (coefficiente di similarità)
    end for
end for

Algorithm 2 User Content-Based

Input: Codice identificativo di un utente *user to recommand*[0-99]
Input: Tipologia di cibo *type of food* (opzionale)
Input: Numero N di elementi da raccomandare
Output: Gli N elementi più simili sulla base dei rating dati dall'utente e dagli input

description_list ← lista contenente l'insieme di descrizioni di tutti i cibi
all_index_to_recommend ← lista contenente tutti gli indici degli item a cui l'utente ha dato un voto
algorithms ← lista contenente gli algoritmi da testare
dictionary ← dizionario contenente coppie item-similarità
sorted_dictionary ← dizionario contenente coppie item-similarità ordinato in base al valore di similarità
all_similarities ← lista di valori di similarità per tutti gli items valutati dall'utente
descr_vector ← vettore di caratteristiche delle descrizioni
for *algo* in *algorithms* **do**
    similarity ← *algo*(descr_vector, descr_vector)
    for *index* in *all_index_to_recommend* **do**
        sim_scores ← insieme di valori di similarità per un determinato *index*
        sim_scores_not_zero ← lista contenente tutti i valori di similarità diversi da 0.0 e 1.0
        top_scores ← lista ordinata dei valori di similarità per un determinato item_id
    end for
     all_similarities ← contiene tutte le similarità per ogni items che l'utente ha votato
    if type_of_food == 0 **then**
        dictionary ← dizionario contenente coppie item-valore di similarità
        sorted_dictionary ← *sorted*(dictionary)
        sorted_dictionary ← *islice*(dictionary.items(), N)
        for *key*, *value* in dictionary.items() **do**
     stampa a video la coppia di *chiave* (nome del cibo) - *valore* (coefficiente di similarità)
        end for
    else
        dictionary ← dizionario contenente coppie item-valore di similarità
        filtraggio del dizionario secondo la tipologia inserita
        sorted_dictionary ← *sorted*(dictionary)
        sorted_dictionary ← *islice*(dictionary.items(), N)
        for *key*, *value* in dictionary.items() **do**
     stampa a video la coppia di *chiave* (nome del cibo) - *valore* (coefficiente di similarità)
        end for
    end if
end for

4.3 Codice e funzionalità principali

Key Content-Based

Il programma *KeyContentBased.py*[2] contiene alcune funzionalità principali che verranno descritte in questa sezione:

- **Inserimento keywords da considerare ed escludere**: viene richiesto in input una serie di parole legate alle preferenze dell'utente (elementi da favorire ed escludere).

```
print("Inserisci un insieme di parole chiave, 0 per
    terminare l'inserimento:")
# inserimento funzionalit  in pi   -> parole che non devono
    contenere i cibi!!!
# esempio persona celiaca -> non deve esserci pane, farina
keywords = []
while True:
    input_usr = input()
    if input_usr == '0':
        break #esci dal while
    else:
        keywords.append(input_usr)

print("Le parole chiave DA TENERE IN CONSIDERAZIONE sono le
    seguenti:")
print(keywords)


print("Se sei intollerante a qualche cibo inseriscilo qui, 0
    per terminare")
not_present = []
while True:
  input_usr = input()
  if input_usr == '0':
    break
  else:
    not_present.append(input_usr)
```

Listing 4.1: Codice per l'inserimento in input delle keywords

[2]https://github.com/giova211001/Food-Recommender-System

- **Processamento dei dati**: viene fatto tramite la classe `TfidfVectorizer` del modulo `sklearn.feature_extraction.text`. Vengono creati due vettori TF-IDF di dati nel seguente modo:

```
vectorizer = TfidfVectorizer(stop_words='english')
#vettore di descrizioni
description_vector = vectorizer.fit_transform(
    description_list)
#vettore di parole chiave
keywords_vector = vectorizer.transform(keywords_en)
```

Listing 4.2: Codice per il processamento dei dati

La porzione di codice 4.2 utilizza la funzione `fit_transform()` per elaborare e processare una lista di descrizioni salvata in `description_list`. In particolare, effettua due compiti fondamentali:

  - **Fit (Addestramento del vettorizzatore)**: qui viene addestrato il vettorizzatore di nome `vectorizer` sui dati di addestramento, ovvero le parole presenti in `description_list`. In questa fase il vettorizzatore apprende il vocabolario delle parole presenti nelle descrizioni ed eventuali trasformazioni da effettuare in seguito.
  - **Transform (Trasformazione dei dati)**: in questa fase vengono trasformati i dati in input in una rappresentazione vettoriale. Nel nostro caso, la rappresentazione è basata sul conteggio delle parole ovvero la *Term Frequency-Inverse Document Frequency (TF-IDF)*.

- **Calcolo dei coefficienti di similarità**: per ogni algoritmo presente in una lista vengono calcolati i coefficienti di similarità tra vettore descrizione e vettore di keywords

```
#insieme di algoritmi da testare
algo = [linear_kernel, cosine_similarity, cosine_distances,
    rbf_kernel, sigmoid_kernel, pairwise_distances]

for algorithm in algo:

  print(f"STO UTILIZZANDO L'ALGORITMO {str(algorithm).split
    (' ')[1].upper()}")
  name_of_algorithm = str(algorithm).split(' ')[1]

```

```
9   #viene calcolato il coefficiente di similarita' tra
     vettore di descrizioni di tutti i cibi e vettore di
     parole inserite dall'utente
10  cosine_sim = algorithm(description_vector,keywords_vector)
```

Listing 4.3: Codice per la misura di similarità

- **Aggiunta coppie item-similarità in un dizionario**: nel calcolo della similarità molti valori sono nulli. Questi valori vengono eliminati e tutti i termini non nulli vengono inseriti in un dizionario.

```
1 #ottengo le righe e le colonne con valori diversi da zero
2   nonzero_rows, nonzero_cols = np.nonzero(cosine_sim)
3
4   #ottengo tutti i valori diversi da zero
5   cosine_sim_not_zero = cosine_sim[nonzero_rows,
     nonzero_cols]
6   #ogni elemento nel dizionario sara' memorizzato nella
     forma
7   # {nome_cibo: coefficiente di similarita'}
8   dictionary = {}
9   for riga, colonna, valore in zip(nonzero_rows,
     nonzero_cols, cosine_sim_not_zero):
10    #inserisco coppia chiave-valore
11    if name_of_algorithm == 'cosine_distances' or
     name_of_algorithm == 'pairwise_distances':
12       #devo salvare il minimo
13       aggiungi_al_dizionario_cosine(dictionary, data.loc[
     data['Describe'] == description_list[riga], 'Name'].
     values[0], valore)
14    else:
15       #salvo il massimo
16       aggiungi_al_dizionario(dictionary, data.loc[data['
     Describe'] == description_list[riga], 'Name'].values[0],
     valore)
```

Listing 4.4: Codice per l'aggiunta delle coppie item-similarità ad dizionario in base all'algoritmo

Vorrei porre particolare attenzione ai 2 casi di inserimento delle coppie nel dizionario:

CASO 1: viene utilizzata la funzione `aggiungi_al_dizionario()` che nel momento in cui trova una chiave duplicata (già presente nel dizionario) mantiene la chiave con il valore **massimo**. Questo perché, nella maggior

parte degli algoritmi testati, valore massimo ci indica maggiore similarità.

CASO 2: negli algoritmi `cosine_distances` e `pairwise_distances` viene utilizzata una differente funzione di nome `aggiungi_al_dizionario_cosine` che, a differenza del precedente, nel momento in cui trova una chiave duplicata mantiene quella con valore **minimo**. Questo perché, nei seguenti due algoritmi i coefficienti indicano la *dissimilarità* tra i due vettori e quindi un valore minimo indica maggiore similarità.

- **Ordinamento del dizionario in base al valore**: per ottenere i cibi maggiormente simili alle parole chiave inserite è necessario ordinarlo:
  - in ordine decrescente (`reverse=True`): nel caso di `linear_kernel`, `cosine_similarity`, `rfb_kernel`, `sigmoid_kernel`
  - in ordine crescente (`reverse=False`): nel caso di `cosine_distances` e `pairwise_distances`

```
print("ORDINAMENTO DEL DIZIONARIO")
  if name_of_algorithm == 'cosine_distances' or
   name_of_algorithm == 'pairwise_distances':
    # ordinamento in ordine crescente
    # valore piccolo == similarita' alta
    dict_sorted = dict(sorted(dictionary.items(), key=lambda
    item: item[1], reverse=False))
  else:
    # ordinamento in ordine descrescente
    # valore grande == similarita' alta
    dict_sorted = dict(sorted(dictionary.items(), key=lambda
    item: item[1], reverse=True))
```

Listing 4.5: Codice per l'ordinamento del dizionario in base all'algoritmo

- **Slicing del dizionario**: per ottenere i primi N valori di similarità effettuo un' estrazione del dizionario tramite la funzione `islice()` del modulo `itertools`.

```
# Prendi i primi n elementi del dizionario
  dict_sorted = dict(islice(chiavi_processate.items(), n))
  print(dict_sorted)
```

Listing 4.6: Codice per ottenere i primi n elementi nel dizionario

User Content-Based

Il programma *UserContentBased.py*[3] è molto simile al precedente ma gestisce l'inserimento della tipologia del cibo, in modo da poter dare la possibilità di escludere a priori un certo numero di cibi dalla raccomandazione:

- **Inserimento User ID e tipologia di cibo**: viene richiesto in input il codice dell'utente da raccomandare ed eventualmente la tipologia del cibo.

```
# inserimento user_id da raccomandare
user_to_recommend = int(input("Inserisci numero utente da
    raccomandare: "))

# inserimento tipologia di cibo, oppure 0
type_of_food = input("Inserisci una categoria tra quelle
    specificate sopra, oppure 0 per avere una raccomandazione
     di default: ")
```

Listing 4.7: Codice per inserimento dell'input iniziale

- **Controllo validità dell'User ID e recupero dei ratings**: viene controllato che l'id inserito in input sia compreso tra 0 e 99 e vengono salvati in una lista (`usr_rat`) tutti i rating dell'utente.

```
#controllo se presente nel trainset
if trainset.knows_user(user_to_recommend):
    #devo prelevare i ratings dell'utente da raccomandare
    usr_rat = user_ratings.get(user_to_recommend)
    num_ratings = len(usr_rat)
    print(f"I rating che l'utente {user_to_recommend} ha
    dato sono in totale {num_ratings} e sono i seguenti:")
    for item_id, ratings in usr_rat:
        print(f"{data.iloc[item_id]['Name']}  --> valutato {
    ratings}")
else:
    #stampa messaggio nel caso venga inserito un valore non
    valido
    print(f"Utente {user_to_recommend} non presente nel
    dataset")
```

Listing 4.8: Codice per controllo e reperimento dei voti dell'utente inserito

- **Salvataggio indici degli items**: vengono salvati in una lista (`all_index_to_recommend`) tutti gli indici degli items valutati dall'utente.

[3]https://github.com/giova211001/Food-Recommender-System

```
#recupero tutti gli indici dei cibi valutati dall'utente
# all index to recommend contiene tutti gli item_id degli
    item che ha valutato l'utente selezionato
all_index_to_recommend = []
for item_id, _ in usr_rat:
    all_index_to_recommend.append(item_id)

print(all_index_to_recommend)
```

Listing 4.9: Codice per salvataggio degli item id

- **Calcolo dei coefficienti di similarità**: per ogni algoritmo presente in una lista di algoritmi da testare viene calcolato il coefficiente di similarità.

```
# insieme di algoritmi da testare
algo = [linear_kernel, cosine_similarity, cosine_distances,
    rbf_kernel, sigmoid_kernel, pairwise_distances]

for algorithm in algo:

  print(f"STO UTILIZZANDO L'ALGORITMO {str(algorithm).split
    (' ')[1].upper()}")
  name_of_algorithm = str(algorithm).split(' ')[1]

  #applico la funzione di similarita' (come nel caso del
    codice precedente)
  matrice_compatta = algorithm(descr_vector, descr_vector)
```

Listing 4.10: Codice per calcolo dei coefficienti di similarità

Nota che, a differenza del caso precedente, in questo caso la similarità viene calcolata tra lo stesso vettore di descrizione. In questo modo viene calcolata la similarità incrociata per ottenere un coefficiente di similarità per ogni coppia di descrizioni di cibi presenti nel `descr_vector`.

- **Salvataggio di tutti i valori di similarità in una lista**: per ogni item che l'utente ha valutato vengono salvati i valori di similarità tra quell'item e tutti gli altri items presenti nel food dataset. Ad esempio, se abbiamo bisogno di ottenere tutti i valori di similarità dell'item 67 lo ottengo con `matrice_compatta[67]`. Questi valori di similarità vengono in seguito ordinati in modo decrescente (per scoprire gli items più simili) e viene eliminato il primo elemento, in modo da escludere l'item di cui sto calcolando la similarità (che sarà uguale a 1).

```
# lista che mi salva tutti i coefficienti di similarita'
  all_similarities = []

# itero per tutti gli item_id
  for i in  all_index_to_recommend:

      # matrice_compatta[i] contiene tutti i valori di
   similarita' per un determinato item_id
      # salvo i valori in un vettore
      sim_scores = list(enumerate(matrice_compatta[i]))

      # Filtra gli elementi con similarit  diversa da 0.0 (
   nel caso di algoritmi in cui i valori bassi indicano
   assenza di similarita') e 1.0 (nel caso in cui valori
   alti indicano assenza di similarita')
      sim_scores_not_zero = [(indice, similarita) for indice
   , similarita in sim_scores if similarita != 0.0 and
   similarita != 1.0]
      # li ordino per quelli pi  simili (stampa numero item
   , e similarit )
      top_scores = sorted(sim_scores_not_zero, key=lambda x:
    x[1], reverse = True)
      #elimino il primo elemento in modo tale da escludere l
   'item uguale a quella di cui sto calcolando la
   similarit
      # nella lista ordinata il primo elemento sar  quello
   con coefficiente = 1
      top_scores = top_scores[1:]
      all_similarities.append(top_scores)
```

Listing 4.11: Codice per salvataggio della similarità

- **Controllo tipologia di cibo**: viene controllato se è stato inserita una tipologia in input oppure il valore 0 (nessuna tipologia). Se la tipologia è presente viene effettuato un filtraggio del dataframe e vengono inserite le coppie item-similarità in un dizionario (`dict`).

```
# CASO 1 -> non ho inserito nessun tipo di tipologia
  if type_of_food == '0':

    #creo un dizionario
    dic = {}
    # non effettuo alcun tipo di filtraggio ma inserisco
   elementi nel dizionario
```

```
    for i in range (0, len(all_similarities)):
      for j in range (0, len(all_similarities[i])):
        # dic[chave] = valore
        dic[data.loc[all_similarities[i][j][0], 'Name']] =
   all_similarities[i][j][1]


# CASO 2 -> ho inserito una tipologia
  else:
    dic = {}
    # se ho inserito la tipologia allora qui    il momento
   di filtrare
    for i in range (0, len(all_similarities)):
      for j in range (0, len(all_similarities[i])):
        # estrazione dal dataset
        if(data.loc[all_similarities[i][j][0], 'C_Type'] ==
   type_of_food):
          # elemento aggiunto al dizionario solo se ha la
   tipologia specificata
          dic[data.loc[all_similarities[i][j][0], 'Name']] =
    all_similarities[i][j][1]
```

Listing 4.12: Codice per controllo tipologia di cibo e inserimento nel dizionario

- **Ordinamento del dizionario in base al valore**: per ottenere i cibi maggiormente simili alle parole chiave inserite è necessario ordinarlo:
  - in ordine decrescente (`reverse=True`): nel caso di `linear_kernel`, `cosine_similarity`, `rfb_kernel`, `sigmoid_kernel`
  - in ordine crescente (`reverse=False`): nel caso di `cosine_distances` e `pairwise_distances`

```
#ordinamento del dizionario in base all'algoritmo
    if name_of_algorithm == 'cosine_distances' or
   name_of_algorithm == 'pairwise_distances':
      # ordinamento crescente
      dic_sorted = dict(sorted(dic.items(), key=lambda item:
    item[1], reverse=False))
    else:
      # ordinamento descrescente
      dic_sorted = dict(sorted(dic.items(), key=lambda item:
    item[1], reverse=True))
```

Listing 4.13: Codice per ordinamento del dizionario

- **Slicing dizionario e output**: vengono presi i primi n elementi del dizionario e vengono stampati a video.

```
    # Prendi i primi n elementi del dizionario
    dic_sorted = dict(islice(dic_sorted.items(), n))

    #stampo gli elementi
    for key, value in dic_sorted.items():
      print(f"{key} -----> {value * 100}")
```

Listing 4.14: Codice ottenere i primi n elementi e stamparli a video

Capitolo 5

Valutazioni delle prestazioni

Il codice è stato testato con diverse tipologie di algoritmi per poter verificare diverse funzionalità principali. In particolare sono stati testati i seguenti algoritmi per la scelta della similarità:

- **Kernel lineare**: il kernel lineare calcola il prodotto scalare tra due vettori e il risultato è un coefficiente di similarità. Dati x e y due vettori di dati:

$$K(x, y) = x^T \cdot y \tag{5.1}$$

- **Similarità del coseno**: calcola la somiglianza tra due vettori di dati calcolando il coseno tra i due vettori nello spazio delle caratteristiche. Dati x e y due vettori di dati:

$$cosine\_similarity(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|} \tag{5.2}$$

- **Distanza del coseno**: è la misura complementare alla cosine_similarity. Essa misura la dissimilarità tra due vettori di dati. Dati x e y due vettori:

$$cosine\_distances(x, y) = 1 - cosine\_similarity(x, y) \tag{5.3}$$

  In questo caso valori di *cosine_distances* prossimi allo 0 indica che i due vettori sono simili

- **Kernel RBF**: funzione kernel comunemente utilizzata in machine learning. Dati x e y vettori di input

$$K(x, y) = e^{-\gamma\|x-y\|^2} \tag{5.4}$$

con γ che rappresenta un parametro positivo che regola la lunghezza della funzione gaussiana. In altre parole regola quanto rapidamente la similarità tra punti diminuisce con l'aumentare della distanza tra di essi.

- **Kernel sigmoide**: è un'altra tipologia di kernel usato nel machine learning. Dato x e y la formula matematica è la seguente:

$$K(x, y) = tanh(\alpha x^T y + c) \tag{5.5}$$

dove α è un parametro che controlla la pendenza della funzione di tangente iperbolica e c è una costante.

- **Pearson Correlation**: funzione che calcola la distanza tra coppie di vettori. La distanza presa in considerazione di solito è la distanza euclidea che viene calcolata con la seguente formula:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{5.6}$$

5.1 Dati raccolti

Per ognuno dei due codici sono state provate diverse combinazioni di input per verificare la correttezza degli algoritmi e la coerenza negli output forniti. Vediamo tutti gli input forniti nelle Tabelle 5.1 e 5.2.

Key Content-Based

Nome input	Keywords da considerare	Keywords da escludere
Input 1	pizza, pesce	pane
Input 2	carota, limone	pepe, sale
Input 3	pollo	0
Input 4	cipolla, pomodoro	pollo, olio
Input 5	insalata, uovo	0
Input 6	cioccolato, latte	burro

Tabella 5.1: Insieme di tutte le combinazioni di input testate per il programma *Key Content-Based*

User Content-Based

Nome input	Codice utente	Tipologia cibo
Input 1	99	Snack
Input 2	15	Dessert
Input 3	55	Healthy Food
Input 4	87	0
Input 5	27	Japanese
Input 6	4	0

Tabella 5.2: Insieme di tutte le combinazioni di input testate per il programma *User Content-Based*

5.2 Valutazione dei dati raccolti

Il programma è stato eseguito nel software Google Colab[1] con i seguenti parametri:

- Processore: Intel(R) Xeon(R) CPU @ 2.20GHz
- RAM: 13,29 GB
- Sistema operativo: Ubuntu 22.04.3 LTS

Sono state effettuate in totale 12 simulazioni dei due programmi:

- 6 simulazioni per *Key Content-Based* con le combinazioni di input specificate nella Tabella 5.1
- 6 simulazioni per *User Content-Based* con le combinazioni di input specificate nella Tabella 5.2

[1]https://colab.research.google.com/

Key Content-Based

- **Input 1** (considero pizza e pesce ed escludo pane)

Nome algoritmo	**Sequenza ordinata di cibi raccomandati**	
	Nome cibo	**Valore di similarità**
Linear Kernel	tricolour pizza	63.237743119142664
	amritsari fish	61.89153009495434
	Sea Food Soup	43.679899212633316
	fish with jamun sauce	40.49873782232446
	fish with white sauce	37.801294439730555
Cosine similarity	tricolour pizza	63.237743119142664
	amritsari fish	61.89153009495434
	Sea Food Soup	43.679899212633316
	fish with jamun sauce	40.49873782232446
	fish with white sauce	37.801294439730555
Cosine distances	tricolour pizza	36.762256880857336
	amritsari fish	38.10846990504566
	Sea Food Soup	56.32010078736669
	fish with jamun sauce	59.50126217767554
	fish with white sauce	62.19870556026945
RBF Kernel	tricolour pizza	99.93348411012879
	amritsari fish	99.93104917585033
	Sea Food Soup	99.89811510393689
	fish with jamun sauce	99.89236337763715
	fish with white sauce	99.88748650131204
Sigmoid Kernel	tricolour pizza	76.18343971888545
	amritsari fish	76.18292850850497
	Sea Food Soup	76.1760118987196
	fish with jamun sauce	76.17480354471132
	fish with white sauce	76.17377888792214
Pairwise distances	tricolour pizza	85.74643652170899
	amritsari fish	87.30231372082376
	Sea Food Soup	106.13208825550045
	fish with jamun sauce	109.08827817659929
	fish with white sauce	111.53358737193872

Tabella 5.3: Risultati ottenuti con la combinazione di input 1

Come si vede nella Tabella 5.14 ciascuno dei 6 algoritmi mi fornisce la stessa raccomandazione per le parole chiave inserite ma con diverso valore di similarità. Nel caso di `cosine_distances` e `pairwise_distances` si può notare come il primo cibo consigliato abbia il valore di similarità più basso,

mentre nel caso degli altri 4 algoritmi il primo cibo sia quello con il valore più alto. Questo è dovuto alle diverse formule matematiche per il calcolo del valore di similarità tra il vettore descrizione e il vettore di parole chiave.

- **Input 2** (considero carota e limone ed escludo pepe e sale)

Nome algoritmo	**Sequenza ordinata di cibi raccomandati**	
	Nome cibo	**Valore di similarità**
Linear Kernel	grilled lemon margarita	61.52681553518215
	Grilled Chicken with Almond and Garlic Sauce	44.92648613572568
	Jeera Alu	33.980484926538765
	Pho Tai rare beef	27.112903152968528
	tricolour pizza	26.0376119381517
Cosine similarity	grilled lemon margarita	61.52681553518215
	Grilled Chicken with Almond and Garlic Sauce	44.92648613572568
	Jeera Alu	33.980484926538765
	Pho Tai rare beef	27.112903152968528
	tricolour pizza	26.0376119381517
Cosine distances	grilled lemon margarita	38.47318446481785
	Grilled Chicken with Almond and Garlic Sauce	55.07351386427432
	Jeera Alu	66.01951507346124
	Pho Tai rare beef	72.88709684703147
	tricolour pizza	73.96238806184829
RBF Kernel	grilled lemon margarita	99.87676450870586
	Grilled Chicken with Almond and Garlic Sauce	99.82363802817524
	Jeera Alu	99.78862270726229
	Pho Tai rare beef	99.76666017171766
	tricolour pizza	99.76322182669865
Sigmoid Kernel	grilled lemon margarita	76.20079425915436
	Grilled Chicken with Almond and Garlic Sauce	76.18963615324837
	Jeera Alu	76.18227619526023
	Pho Tai rare beef	76.17765751249306
	tricolour pizza	76.17693427238187
Pairwise distances	grilled lemon margarita	87.71907941242641
	Grilled Chicken with Almond and Garlic Sauce	104.95095413027393
	Jeera Alu	114.9082373665711
	Pho Tai rare beef	120.73698426499769
	tricolour pizza	121.62432985373303

Tabella 5.4: Risultati ottenuti con la combinazione di input 2

Si può notare come, anche in questo caso, i cibi raccomandati sono tutti gli stessi

(e nello stesso ordine) per tutti e 6 gli algoritmi testati, come si vede nella Tabella 5.4.

- **Input 3** (considero pollo e non escludo nulla)

Nome algoritmo	**Sequenza ordinata di cibi raccomandati**	
	Nome cibo	**Valore di similarità**
Linear Kernel	chettinadu chicken	52.17206695240569
	Chicken Sweet Corn Soup	50.10295650506137
	chicken 65	49.95484427782311
	Pho Ga Chicken	39.055994889039475
	Chicken and Dumplings	38.37115994502664
Cosine similarity	chettinadu chicken	52.17206695240569
	Chicken Sweet Corn Soup	50.10295650506138
	chicken 65	49.95484427782311
	Pho Ga Chicken	39.055994889039475
	Chicken and Dumplings	38.37115994502664
Cosine distances	chettinadu chicken	47.82793304759431
	Chicken Sweet Corn Soup	49.89704349493862
	chicken 65	50.045155722176894
	Pho Ga Chicken	60.94400511096052
	Chicken and Dumplings	61.62884005497335
RBF Kernel	chettinadu chicken	99.91799602064819
	Chicken Sweet Corn Soup	99.91444991943405
	chicken 65	99.91419608525028
	Pho Ga Chicken	99.89551944726004
	Chicken and Dumplings	99.89434600745454
Sigmoid Kernel	chettinadu chicken	76.17820072759527
	Chicken Sweet Corn Soup	76.17745596522553
	chicken 65	76.17740265245463
	Pho Ga Chicken	76.1734793450531
	Chicken and Dumplings	76.17323280330889
Pairwise distances	chettinadu chicken	97.80381694759599
	Chicken Sweet Corn Soup	99.89699044009146
	chicken 65	100.0451455315818
	Pho Ga Chicken	110.40290314204653
	Chicken and Dumplings	111.02147544954836

Tabella 5.5: Risultati ottenuti con la combinazione di input 3

Qui notiamo come tutti i cibi suggeriti siano a base di pollo (parola chiave inserita in input). In questo caso non l'utente non ha alcun tipo di intolleranza quindi la raccomandazione viene effettuata tra tutti i possibili cibi

presenti nel dataset.

- **Input 4** (considero cipolla e pomodoro ed escludo pollo e olio)

Nome algoritmo	**Sequenza ordinata di cibi raccomandati**	
	Nome cibo	**Valore di similarità**
Linear Kernel	Shirazi Salad	45.69819607813348
	bhurji- egg	37.77336510114871
	Sukuti Chatpate	34.30679455640933
	Crispy Pakora	32.93367670301269
	prawn potato soup	28.72787662901763
Cosine similarity	Shirazi Salad	45.69819607813348
	bhurji- egg	37.77336510114872
	Sukuti Chatpate	34.30679455640933
	Crispy Pakora	32.933676703012694
	prawn potato soup	28.727876629017636
Cosine distances	Shirazi Salad	54.30180392186652
	bhurji- egg	62.22663489885127
	Sukuti Chatpate	65.69320544359067
	Crispy Pakora	67.0663232969873
	prawn potato soup	71.27212337098237
RBF Kernel	Shirazi Salad	99.85154176144927
	bhurji- egg	99.82989413323439
	Sukuti Chatpate	99.82042625472022
	Crispy Pakora	99.81667625103186
	prawn potato soup	99.80519102992031
Sigmoid Kernel	Shirazi Salad	76.18565763958145
	bhurji- egg	76.18110862168231
	Sukuti Chatpate	76.17911850180448
	Crispy Pakora	76.17833017024552
	prawn potato soup	76.17591540468987
Pairwise distances	Shirazi Salad	104.21305476941602
	bhurji- egg	111.55862575242783
	Sukuti Chatpate	114.62391150505262
	Crispy Pakora	115.81564945808256
	prawn potato soup	119.39189534552366

Tabella 5.6: Risultati ottenuti con la combinazione di input 4

- **Input 5** (considero insalata e uovo e non escludo nulla)

Nome algoritmo	**Sequenza ordinata di cibi raccomandati**	
	Nome cibo	**Valore di similarità**
Linear Kernel	chocolate doughnut	59.182574721111436
	Stir-Fried Lettuces With Crispy Shallots	43.07309204483353
	Summer Rolls	37.70828500620742
	Black-Bean Burgers	35.483508823323874
	spiced orange valencia cake	35.312860894470724
Cosine similarity	chocolate doughnut	59.182574721111436
	Stir-Fried Lettuces With Crispy Shallots	43.07309204483353
	Summer Rolls	37.70828500620742
	Black-Bean Burgers	35.483508823323874
	spiced orange valencia cake	35.312860894470724
Cosine distances	chocolate doughnut	40.817425278888564
	Stir-Fried Lettuces With Crispy Shallots	56.92690795516646
	Summer Rolls	62.29171499379258
	Black-Bean Burgers	64.51649117667613
	spiced orange valencia cake	64.68713910552928
RBF Kernel	chocolate doughnut	99.93001176687002
	Stir-Fried Lettuces With Crispy Shallots	99.90240287492021
	Summer Rolls	99.89321020844852
	Black-Bean Burgers	99.88939827372252
	spiced orange valencia cake	99.88910589132145
Sigmoid Kernel	chocolate doughnut	76.18072396296715
	Stir-Fried Lettuces With Crispy Shallots	76.17492546243763
	Summer Rolls	76.17299416510284
	Black-Bean Burgers	76.1721932197334
	spiced orange valencia cake	76.17213178352765
Pairwise distances	chocolate doughnut	90.35200637383608
	Stir-Fried Lettuces With Crispy Shallots	106.70230358822292
	Summer Rolls	111.61694763233099
	Black-Bean Burgers	113.59268565948788
	spiced orange valencia cake	113.74281437130811

Tabella 5.7: Risultati ottenuti con la combinazione di input 5

E' possibile notare come, anche nel caso della seguente Tabella 5.7, i cibi suggeriti siano coerenti con la combinazione di input inserita (insalata e uovo).

- **Input 6** (considero cioccolato e latte ed escludo burro)

Nome algoritmo	**Sequenza ordinata di cibi raccomandati**	
	Nome cibo	**Valore di similarità**
Linear Kernel	duo of chocolate and strawberry	74.31761099518084
	chocolate kaju katli	68.55579665996764
	chocolate and almond rum ball	41.61197873643401
	filter coffee	41.505574662863665
	homemade gulab jamun	41.285801250975865
Cosine similarity	duo of chocolate and strawberry	74.31761099518084
	chocolate kaju katli	68.55579665996764
	chocolate and almond rum ball	41.61197873643401
	filter coffee	41.505574662863665
	homemade gulab jamun	41.285801250975865
Cosine distances	duo of chocolate and strawberry	25.68238900481914
	chocolate kaju katli	31.444203340032363
	chocolate and almond rum ball	58.38802126356599
	filter coffee	58.494425337136335
	homemade gulab jamun	58.714198749024135
RBF Kernel	duo of chocolate and strawberry	99.94895460515193
	chocolate kaju katli	99.93750620878936
	chocolate and almond rum ball	99.88398778112996
	filter coffee	99.88377648785077
	homemade gulab jamun	99.88334007134193
Sigmoid Kernel	duo of chocolate and strawberry	76.19042348191218
	chocolate kaju katli	76.18802070084627
	chocolate and almond rum ball	76.17678185676783
	filter coffee	76.17673746425548
	homemade gulab jamun	76.17664577304791
Pairwise distances	duo of chocolate and strawberry	71.6692249223042
	chocolate kaju katli	79.30221099065568
	chocolate and almond rum ball	108.0629642972707
	filter coffee	108.16138436349299
	homemade gulab jamun	108.36438413890805

Tabella 5.8: Risultati ottenuti con la combinazione di input 6

I risultati presenti nella Tabella 5.8 sono consistenti con le aspettative in quanto vengono suggeriti gran parte di cibi a base di cioccolato o con il cioccolato tra gli ingredienti.

User Content-Based

Nei seguenti output viene inclusa un'ulteriore tabella per visualizzare la tipologia

e il numero di cibi che l'utente da raccomandare ha votato.

- **Input 1** (User ID = 99 e tipologia di cibo: Snack)

User ID	Nome cibo votato	Voto
99	sweet chilli almonds	10.0
99	puffed rice	10.0
99	hot chocolate	7.0

Nome algoritmo	Sequenza ordinata di cibi raccomandati	
	Nome cibo	Valore di similarità
Linear Kernel	chocolate appe	48.845463508146516
	banana phirni tartlets with fresh strawberries	14.688662031497609
	chocolate samosa	14.628583877063232
	sweet chilli almonds	8.241545404104187
	southern fried chicken tenders	6.852121999009933
Cosine similarity	chocolate appe	48.845463508146516
	banana phirni tartlets with fresh strawberries	14.688662031497612
	chocolate samosa	14.628583877063237
	sweet chilli almonds	8.241545404104187
	southern fried chicken tenders	6.852121999009936
Cosine distances	chocolate appe	51.154536491853484
	banana phirni tartlets with fresh strawberries	85.31133796850239
	chocolate samosa	85.37141612293676
	sweet chilli almonds	91.75845459589581
	southern fried chicken tenders	93.14787800099008
RBF Kernel	banana phirni tartlets with fresh strawberries	99.85377538650177
	chocolate samosa	99.85367248723166
	sweet chilli almonds	99.84273364822974
	almond pearls	99.83967689120671
	baked shankarpali	99.83679617420604
Sigmoid Kernel	chocolate appe	76.17700332925345
	banana phirni tartlets with fresh strawberries	76.164705706502
	chocolate samosa	76.1646840714659
	sweet chilli almonds	76.16238390714322
	almond pearls	76.16174106821069
Pairwise distances	chocolate appe	101.14794757369374
	banana phirni tartlets with fresh strawberries	130.622615169428
	chocolate samosa	130.66860076004238
	sweet chilli almonds	135.4684129942444
	almond pearls	136.7796609918707

Tabella 5.9: Risultati ottenuti con la combinazione di input 1

- **Input 2** (User ID = 15 e tipologia di cibo: Dessert)

User ID	Nome cibo votato	Voto
15	spiced coffee	1.0
15	beetroot and green apple soup	3.0
15	baba budan no. 7	4.0
15	fruit infused tea	3.0
15	soya milk	10.0

Nome algoritmo	Sequenza ordinata di cibi raccomandati	
	Nome cibo	Valore di similarità
Linear Kernel	chocolate marquise	39.664375981317534
	chocolate and almond rum ball	34.5648173641072
	chocolate kaju katli	33.05507593148292
	pista chocolate and mandarin	31.88863342205281
	homemade gulab jamun	31.31901372810874
Cosine similarity	chocolate marquise	39.664375981317534
	chocolate and almond rum ball	34.5648173641072
	chocolate kaju katli	33.05507593148292
	pista chocolate and mandarin	31.88863342205281
	homemade gulab jamun	31.31901372810874
Cosine distances	chocolate marquise	60.33562401868245
	chocolate and almond rum ball	65.4351826358928
	chocolate kaju katli	66.94492406851708
	pista chocolate and mandarin	68.11136657794718
	homemade gulab jamun	68.68098627189126
RBF Kernel	chocolate marquise	99.89656189772896
	chocolate and almond rum ball	99.88782422888501
	chocolate kaju katli	99.88523755934395
	pista chocolate and mandarin	99.88323911643458
	homemade gulab jamun	99.88226321279569
Sigmoid Kernel	chocolate marquise	76.17369836143033
	chocolate and almond rum ball	76.171862473303
	chocolate kaju katli	76.17131892899722
	pista chocolate and mandarin	76.17089897346398
	homemade gulab jamun	76.17069389035584
Pairwise distances	chocolate marquise	109.85046565097704
	chocolate and almond rum ball	114.39858621145002
	chocolate kaju katli	115.71078088796834
	pista chocolate and mandarin	116.71449488212437
	homemade gulab jamun	117.20152411286404

Tabella 5.10: Risultati ottenuti con la combinazione di input 2

- **Input 3** (User ID = 55 e tipologia di cibo: Healthy Food)

User ID	Nome cibo votato	Voto
55	hassel back sweet potatoes	9.0
55	mother christmas cake	4.0
55	matcha tea macarons	9.0

Nome algoritmo	Sequenza ordinata di cibi raccomandati	
	Nome cibo	Valore di similarità
Linear Kernel	gluten free almond cake	23.36180614927581
	whole wheat cake	16.166360577442617
	cream of almond soup	12.268967490148926
	coconut lime quinoa salad	11.566707474818774
	spinach banana pancakes	11.512055792978071
Cosine similarity	gluten free almond cake	23.36180614927581
	whole wheat cake	16.166360577442617
	cream of almond soup	12.268967490148926
	coconut lime quinoa salad	11.566707474818774
	spinach banana pancakes	11.512055792978071
Cosine distances	gluten free almond cake	76.63819385072419
	whole wheat cake	83.83363942255738
	cream of almond soup	87.73103250985108
	coconut lime quinoa salad	88.43329252518123
	spinach banana pancakes	88.48794420702193
RBF Kernel	gluten free almond cake	99.86863148611721
	whole wheat cake	99.85630635817748
	cream of almond soup	99.84963112198592
	spinach banana pancakes	99.84833477784964
	strawberry quinoa pancakes	99.84669898975412
Sigmoid Kernel	gluten free almond cake	76.1678288563996
	whole wheat cake	76.16523784235977
	cream of almond soup	76.16383432511338
	spinach banana pancakes	76.16356174010714
	strawberry quinoa pancakes	76.16321777060917
Pairwise distances	gluten free almond cake	123.80484146488311
	whole wheat cake	129.48640038440902
	cream of almond soup	132.46209458547082
	spinach banana pancakes	133.03228495896923
	strawberry quinoa pancakes	133.74831966405137

Tabella 5.11: Risultati ottenuti con la combinazione di input 3

- **Input 4** (User ID = 87 e nessuna tipologia di cibo)

User ID	Nome cibo votato	Voto
87	cheese chicken kebabs	3.0
87	baba budan no. 7	6.0
87	amritsari fish	2.0
87	active charcoal modak	9.0

Nome algoritmo	Sequenza ordinata di cibi raccomandati	
	Nome cibo	Valore di similarità
Linear Kernel	sunga pork	59.08106345479116
	beetroot modak	56.86727972261273
	instant rava dosa	31.6432123157208
	Sea Food Soup	27.60987674001281
	Noodle Curry	18.672816434111613
Cosine similarity	sunga pork	59.08106345479116
	beetroot modak	56.86727972261273
	instant rava dosa	31.6432123157208
	Sea Food Soup	27.60987674001281
	Noodle Curry	18.672816434111613
Cosine distances	sunga pork	40.91893654520884
	beetroot modak	43.13272027738725
	instant rava dosa	68.35678768427918
	Sea Food Soup	72.39012325998719
	Noodle Curry	81.32718356588839
RBF Kernel	instant rava dosa	99.88281864639254
	jalebi with fennel yogurt pudding	99.85952984332211
	Crispy Pakora	99.85787284599802
	filter coffee	99.8572942106381
	methi chicken masala	99.85642619177514
Sigmoid Kernel	beetroot modak	76.17989066312335
	instant rava dosa	76.17081061342793
	jalebi with fennel yogurt pudding	76.16591554451725
	Crispy Pakora	76.16556718394027
	filter coffee	76.16544553152718
Pairwise distances	beetroot modak	92.87919064826873
	instant rava dosa	116.92458055026682
	jalebi with fennel yogurt pudding	128.02474264794824
	Crispy Pakora	128.77815809241852
	filter coffee	129.04022283180947

Tabella 5.12: Risultati ottenuti con la combinazione di input 4

- **Input 5** (User ID = 27 e tipologia di cibo: Japanese)

User ID	Nome cibo votato	Voto
27	methi malai cranberry chicken	7.0
27	thai style chicken tikka	9.0
27	chicken gilafi kebab	10.0
27	cheese chicken kebabs	10.0

Nome algoritmo	Sequenza ordinata di cibi raccomandati	
	Nome cibo	Valore di similarità
Linear Kernel	flax seed and beetroot modak	22.09552285634141
	sunga pork	17.18177460811914
	almond white chocolate gujiya	15.493576249111912
	beetroot modak	15.434613183070256
	active charcoal modak	14.045994262433634
Cosine similarity	flax seed and beetroot modak	22.09552285634141
	sunga pork	17.18177460811914
	almond white chocolate gujiya	15.493576249111912
	beetroot modak	15.434613183070256
	active charcoal modak	14.045994262433634
Cosine distances	flax seed and beetroot modak	77.90447714365858
	sunga pork	82.81822539188086
	almond white chocolate gujiya	84.50642375088808
	beetroot modak	84.56538681692975
	active charcoal modak	85.95400573756636
RBF Kernel	flax seed and beetroot modak	99.86646235050549
	almond white chocolate gujiya	99.85515401906318
	beetroot modak	99.85505302827306
	active charcoal modak	99.85267465838241
	japanese fish stew	99.84661715751754
Sigmoid Kernel	flax seed and beetroot modak	76.16737289704535
	almond white chocolate gujiya	76.16499556643397
	beetroot modak	76.1649743331812
	active charcoal modak	76.16447427140842
	japanese fish stew	76.16320056286695
Pairwise distances	flax seed and beetroot modak	124.8234570452674
	almond white chocolate gujiya	130.00494125292937
	beetroot modak	130.0502878250792
	active charcoal modak	131.11369549941483
	japanese fish stew	133.7840398012615

Tabella 5.13: Risultati ottenuti con la combinazione di input 5

- **Input 6** (User ID = 4 e nessuna tipologia di cibo)

User ID	Nome cibo votato	Voto
4	dates and nuts ladoo	9.0
4	green lentil dessert fudge	2.0
4	cashew nut cookies	10.0
4	almond pearls	9.0
4	hawaiin papaya salad	6.0
4	vegetable som tam salad	6.0

Nome algoritmo	Sequenza ordinata di cibi raccomandati	
	Nome cibo	**Valore di similarità**
Linear Kernel	Quinoa Bowl and Berries	28.365621301779615
	Jeera Alu	28.162491948735894
	thai pineapple rice	28.05812365677954
	tricolour salad	25.879288757002673
	Hyakula	25.744187836157906
Cosine similarity	Quinoa Bowl and Berries	28.365621301779615
	Jeera Alu	28.162491948735894
	thai pineapple rice	28.05812365677954
	tricolour salad	25.879288757002673
	Hyakula	25.744187836157906
Cosine distances	watermelon and strawberry smoothie	62.45286087129789
	Quinoa Bowl and Berries	71.63437869822037
	Jeera Alu	71.8375080512641
	thai pineapple rice	71.94187634322047
	tricolour salad	74.12071124299733
RBF Kernel	veg summer rolls	99.86676159344078
	Vietnamese Chicken Salad	99.86443514592038
	avial with red rice	99.86124636752048
	mixed vegetable soup	99.85836853684764
	badam papite ke kebab with pineapple salsa	99.85726033260369
Sigmoid Kernel	tricolour salad	76.16873532131059
	veg summer rolls	76.16743579992017
	Vietnamese Chicken Salad	76.16694675628196
	avial with red rice	76.16627640905098
	mixed vegetable soup	76.16567139709542
Pairwise distances	tricolour salad	121.75443420508128
	peach, raspberry and nuts smoothie	123.95468363900773
	veg summer rolls	124.68342752317088
	Vietnamese Chicken Salad	125.76798461224448
	avial with red rice	127.23957006076223

Tabella 5.14: Risultati ottenuti con la combinazione di input 6

5.3 Analisi dei risultati ottenuti

Alla luce delle tabelle inserite sopra, si può notare come nella maggior parte dei casi diverse tipologie di algoritmi per il calcolo della similarità si ottengono le stesse raccomandazioni. Nel primo programma, *Key Content-Based*, vengono fornite delle raccomandazioni più accurate perché la similarità viene calcolata tra un vettore di parole chiave. Vediamo infatti, in questo caso, che quasi in tutti i test effettuati ogni algoritmo raccomanda gli stessi cibi (con stesso ordine ma con diversi valori di similarità). Nel secondo programma, invece, la raccomandazione è un po' più complicata perché, al posto che avere un vettore di parole chiave (come nel caso precedente), ho un vettore di descrizioni di cibi e quindi viene calcolata la similitudine tra vettori che possiedono moltissimi termini. E' proprio per questo motivo che in questo caso, a differenza del precedente, molto spesso non tutti gli algoritmi mi raccomandano gli stessi items.

5.3.1 Occorrenza dei cibi raccomandati

Nel caso di *Key Content-Based* ogni test effettuato raccomanda gli stessi cibi per ogni algoritmo, quindi in questo caso le occorrenze di ogni cibo sono pari al numero di algoritmi, ovvero 6. Nel programma *User Content-Based* non vengono raccomandati sempre gli stessi 5 cibi ma variano in base all'algoritmo. Vediamo quindi il numero diverso di cibi raccomandati per ogni input.

User Content-Based

- **Input 1**

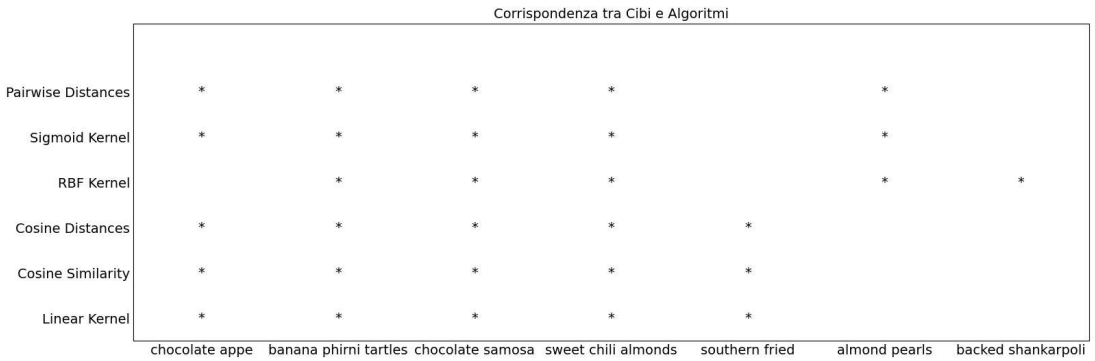


Figura 5.1: Istogramma che mostra il numero di cibi diversi raccomandati per l'input 1 nel programma *User Content-Based*

- **Input 2**

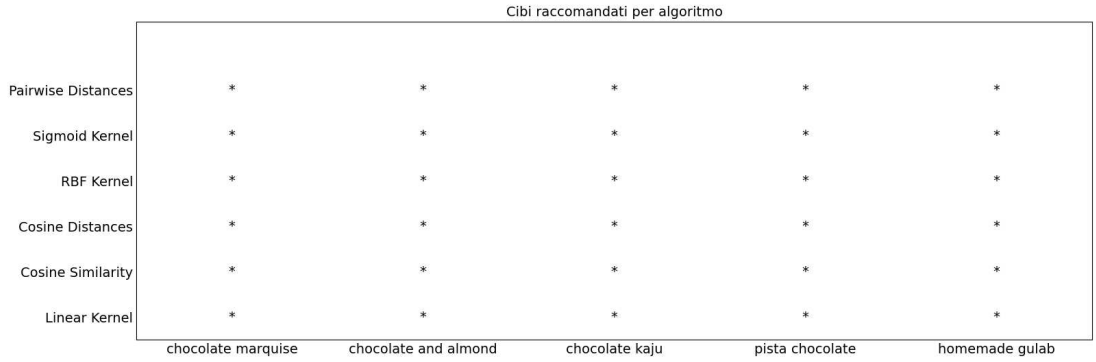


Figura 5.2: Istogramma che mostra il numero di cibi diversi raccomandati per l'input 2 nel programma *User Content-Based*

- **Input 3**

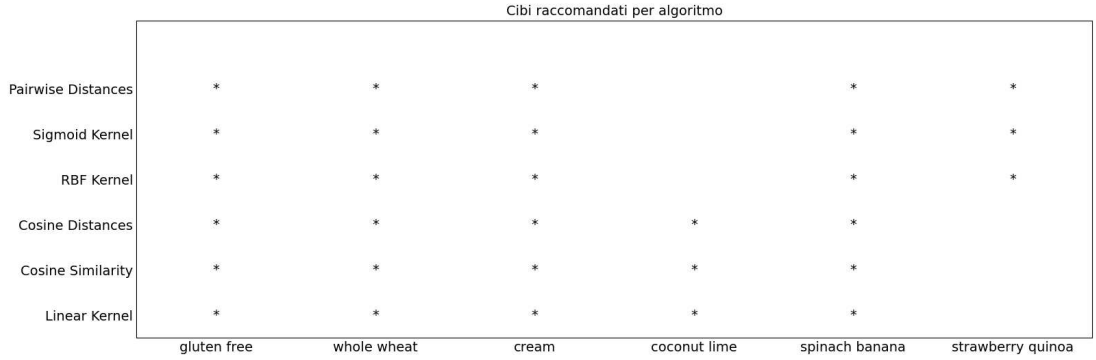


Figura 5.3: Istogramma che mostra il numero di cibi diversi raccomandati per l'input 3 nel programma *User Content-Based*

- **Input 4**

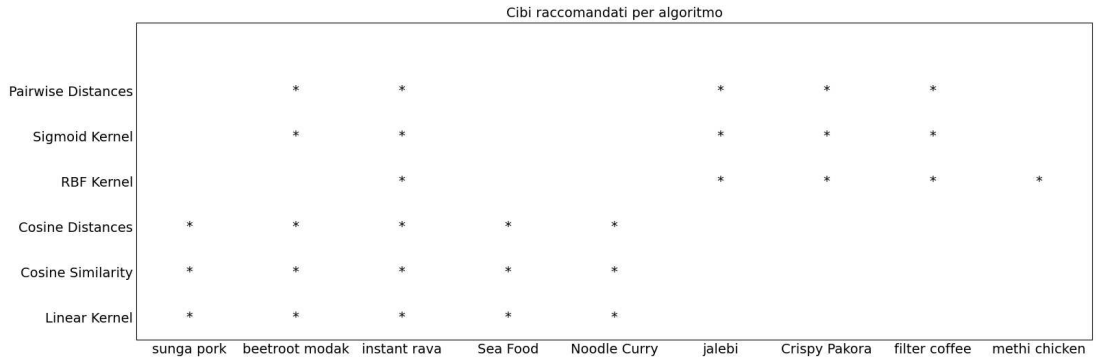


Figura 5.4: Istogramma che mostra il numero di cibi diversi raccomandati per l'input 4 nel programma *User Content-Based*

- **Input 5**

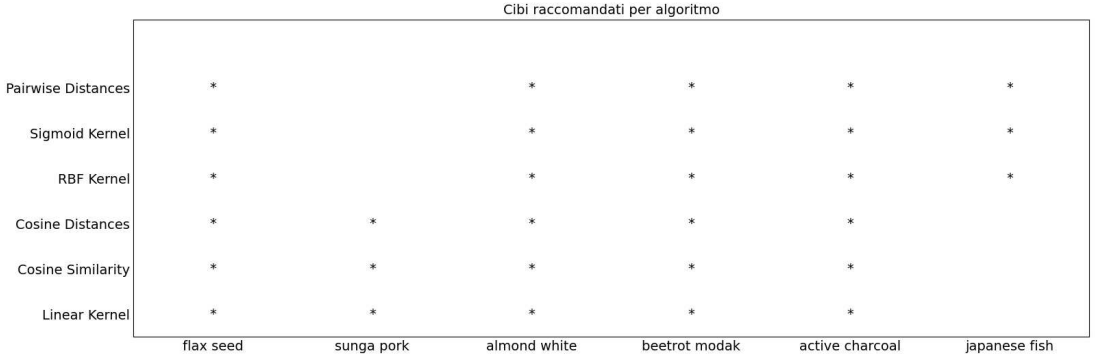


Figura 5.5: Istogramma che mostra il numero di cibi diversi raccomandati per l'input 5 nel programma *User Content-Based*

- **Input 6**

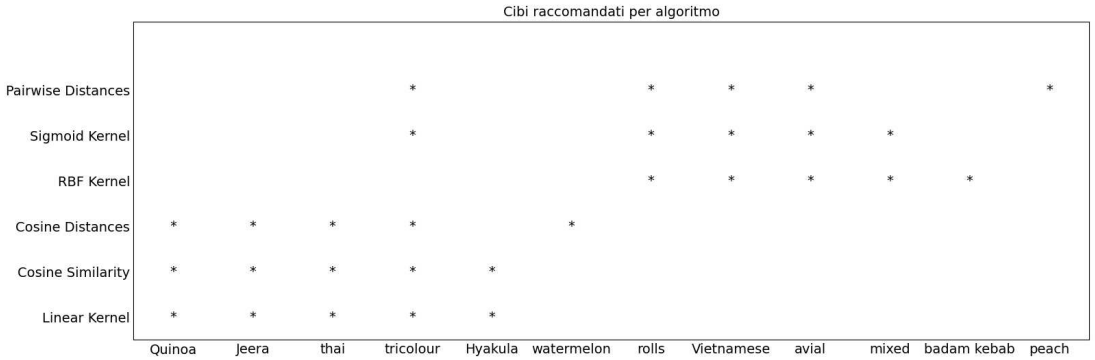


Figura 5.6: Istogramma che mostra il numero di cibi diversi raccomandati per l'input 6 nel programma *User Content-Based*

Come si può notare dai seguenti istogrammi alcune combinazioni di input (in particolare Input 4 e Input 6) ritornano raccomandazioni molto diverse tra loro in base all'algoritmo testato. Questo è sicuramente dovuto al fatto che entrambi gli input non prendono in considerazione alcun tipo di tipologia e quindi l'insieme di dati da considerare e processare è più elevato.

5.3.2 Tempi di esecuzione

Per il confronto dei diversi algoritmi sono state inserite anche alcune tabelle che confrontano i diversi tempi di esecuzione sia per il programma *Key Content-Based* che per il programma *User Content-Based*.

Key Content-Based

In seguito vengono mostrate le tabelle con i tempi di esecuzione (running time) con le diverse combinazioni di input per il programma *Key Content-Based*.

Input 1

Algoritmo	Running Time [s]
Linear Kernel	0.013196945190
Cosine Similarity	0.013458967209
Cosine Distances	0.206068992615
RBF Kernel	0.199371576309
Sigmoid Kernel	0.222962141037
Pairwise Distances	0.224032402039

(a) Tabella del tempo di esecuzione con l'input 1

Input 2

Algoritmo	Running Time [s]
Linear Kernel	0.014405250549
Cosine Similarity	0.012283086777
Cosine Distances	0.091499090195
RBF Kernel	0.102983713150
Sigmoid Kernel	0.103869438171
Pairwise Distances	0.110168457030

(b) Tabella del tempo di esecuzione con l'input 2

Input 3

Algoritmo	Running Time [s]
Linear Kernel	0.049060821533
Cosine Similarity	0.046884059906
Cosine Distances	0.155583143234
RBF Kernel	0.106647729874
Sigmoid Kernel	0.129055976868
Pairwise Distances	0.118241310120

(a) Tabella del tempo di esecuzione con l'input 3

Input 4

Algoritmo	Running Time [s]
Linear Kernel	0.018626928329
Cosine Similarity	0.015365362167
Cosine Distances	0.111115455627
RBF Kernel	0.098597049713
Sigmoid Kernel	0.102463960648
Pairwise Distances	0.100836753845

(b) Tabella del tempo di esecuzione con l'input 4

Input 5

Algoritmo	Running Time [s]
Linear Kernel	0.046395063400
Cosine Similarity	0.032213449478
Cosine Distances	0.370556116104
RBF Kernel	0.291006326675
Sigmoid Kernel	0.228868007660
Pairwise Distances	0.245876550674

(a) Tabella del tempo di esecuzione con l'input 5

Input 6

Algoritmo	Running Time [s]
Linear Kernel	0.026693582535
Cosine Similarity	0.020376205444
Cosine Distances	0.229528665543
RBF Kernel	0.234026908875
Sigmoid Kernel	0.215573549271
Pairwise Distances	0.175908565521

(b) Tabella del tempo di esecuzione con l'input 6

Figura 5.9: Tabelle con i tempi di esecuzione di tutti gli input testati per il programma *Key Content-Based*

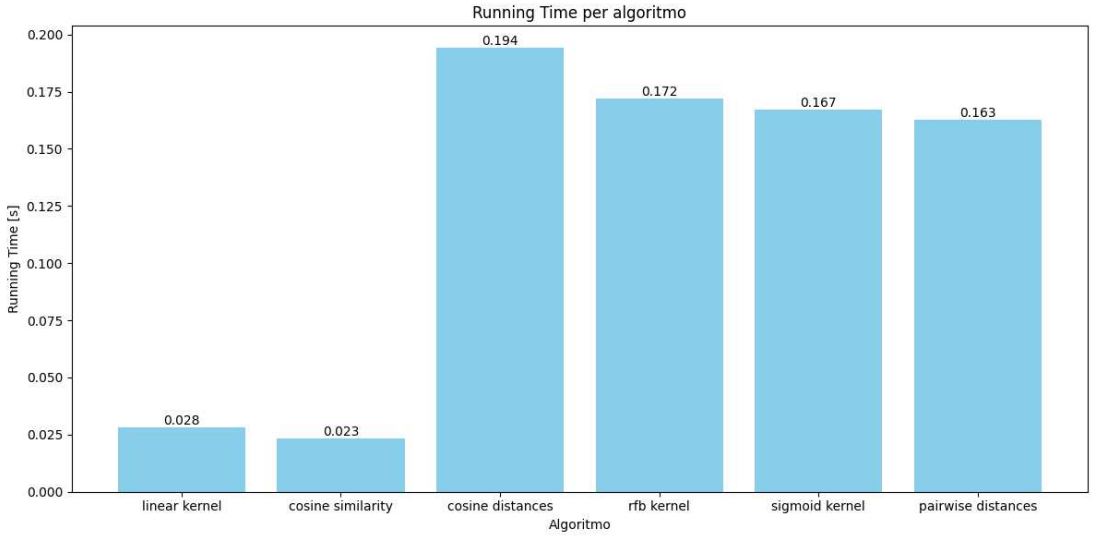


Figura 5.10: Istogramma che mostra il tempo di esecuzione medio per ogni algoritmo nel programma *Key Content-Based*

Come posso notare, nel caso del *Key Content-Based*, il miglior algoritmo per velocità di esecuzione risulta essere **cosine similarity**.

User Content-Based

In seguito sono mostrate le tabelle con i tempi di esecuzione (running time) con le diverse combinazioni di input per il programma *User Content-Based*.

Input 1

Algoritmo	Running Time [s]
Linear Kernel	0.018192291260
Cosine Similarity	0.016182899475
Cosine Distances	0.017318487167
RBF Kernel	0.028910398483
Sigmoid Kernel	0.032181262970
Pairwise Distances	0.023186445236

(a) Tabella del tempo di esecuzione con l'input 1

Input 2

Algoritmo	Running Time [s]
Linear Kernel	0.020625352859
Cosine Similarity	0.023348331451
Cosine Distances	0.020941019058
RBF Kernel	0.034244060516
Sigmoid Kernel	0.039921045303
Pairwise Distances	0.033549070358

(b) Tabella del tempo di esecuzione con l'input 2

Input 3

Algoritmo	Running Time [s]
Linear Kernel	0.019712209702
Cosine Similarity	0.018227815628
Cosine Distances	0.018185138702
RBF Kernel	0.025459766388
Sigmoid Kernel	0.036210298538
Pairwise Distances	0.025979757309

(a) Tabella del tempo di esecuzione con l'input 3

Input 4

Algoritmo	Running Time [s]
Linear Kernel	0.035217285156
Cosine Similarity	0.031158685684
Cosine Distances	0.030447483063
RBF Kernel	0.040132045746
Sigmoid Kernel	0.048246860504
Pairwise Distances	0.041155338287

(b) Tabella del tempo di esecuzione con l'input 4

Input 5

Algoritmo	Running Time [s]
Linear Kernel	0.029464721680
Cosine Similarity	0.029235363007
Cosine Distances	0.025700569153
RBF Kernel	0.029099464417
Sigmoid Kernel	0.034436702728
Pairwise Distances	0.028735399246

(a) Tabella del tempo di esecuzione con l'input 5

Input 6

Algoritmo	Running Time [s]
Linear Kernel	0.020894289017
Cosine Similarity	0.020346164703
Cosine Distances	0.022489309311
RBF Kernel	0.030979633331
Sigmoid Kernel	0.042219638824
Pairwise Distances	0.032397270203

(b) Tabella del tempo di esecuzione con l'input 6

Figura 5.13: Tabelle con i tempi di esecuzione di tutti gli input testati per il programma *User Content-Based*

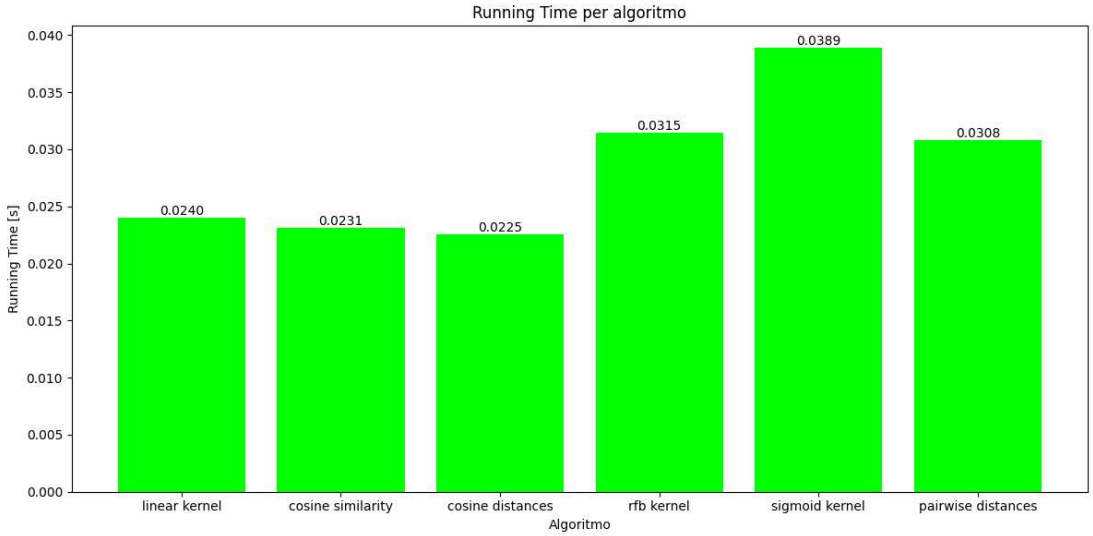


Figura 5.14: Istogramma che mostra il tempo di esecuzione medio per ogni algoritmo nel programma *User Content-Based*

In questo caso, a differenza del precedente, i tempi di esecuzione sono molto ridotti in tutti e sei gli algoritmi. L'algoritmo di similarità più veloce in questo caso, in termini di esecuzione è il **cosine distances**, come si può vedere in Figura 5.14.

5.3.3 Test su nuovi utenti

Per valutare le prestazioni del sistema di raccomandazione basato sul modello *Content-Based Filtering* sono state calcolate le tre metriche principali: accuracy, precision e recall. Per fare ciò sono state reclutate tre persone fisiche per comprendere se le raccomandazioni fornite dal sistema fossero di loro gradimento o meno. In particolare sono stati creati due forms con Google Forms[2] dove veniva richiesto a ogni persona di esprimere una valutazione tra *Mi piace* e *Non mi piace* in merito ai cibi delle categorie **Beverage** e **Snack**. In base ai valori dal sistema e dalle risposte ai sondaggi sono state calcolati:

- True Positive (TP): cibo che è stato votato come *Mi piace* e che è stato raccomandato dal software

- False Positive (FP): cibo che è stato votato come *Non mi piace* e che è stato raccomandato dal software

[2]https://www.google.it/intl/it/forms/about/

- True Negative (TN): cibo che è stato votato come *Non mi piace* e che non è stato raccomandato dal software
- False Negative (FN): cibo che è stato votato come *Mi piace* e che non è stato raccomandato dal software

Di seguito sono riportati i valori ottenuti di accuracy, precision e recall per entrambi i test eseguiti.

- **Test 1 (tipologia Snack)**

Metrica di valutazione	Valore
Precision	0.33
Accuracy	0.47
Recall	0.12

Figura 5.15: Risultati ottenuti nel test 1

- **Test 2 (tipologia Beverage)**

Metrica di valutazione	Valore
Precision	0.60
Accuracy	0.55
Recall	0.50

Figura 5.16: Risultati ottenuti nel test 2

La numerosità delle persone coinvolte è troppo bassa per riuscire a fare una considerazione significativa sulle performance del sistema. Saranno necessari test futuri effettuati a un maggior numero di persone per poter comprendere al meglio il valore delle seguenti metriche di valutazione.

Capitolo 6

Conclusioni

Nella seguente tesi sono stati implementati due differenti programmi per il calcolo di raccomandazioni basate sul modello *Content-Based Filtering*. I risultati ottenuti risultano essere in linea con le aspettative, infatti in entrambi i programmi vengono forniti suggerimenti che sono precisi e coerenti con i diversi input inseriti. Nelle sezioni seguenti verranno descritti alcuni punti di forza e di debolezza dei sistemi di raccomandazione e della libreria Scikit-learn, utilizzata nella scrittura del codice.

6.1 Punti di forza dei sistemi di raccomandazione

Durante lo studio dei sistemi di raccomandazione e delle loro caratteristiche sono emersi diversi punti di forza e vantaggi nel loro uso. I principali sono i seguenti:

- Personalizzazione dei contenuti: i sistemi di raccomandazione sono in grado di fornire contenuti in base alle preferenze e alle esperienze passate degli utenti. Questo aiuta a migliorare l'esperienza complessiva e aumentare il grado di soddisfazione dell'utente.

- Risparmio di tempo: gli utenti o clienti di piattaforme riescono a trovare informazioni rilevanti in modo rapido e veloce e non si perdono in una miriade di opzioni.

- Scoperta di contenuti: nelle piattaforme di streaming sono molto utili a scoprire nuovi contenuti, in base alle caratteristiche e alle preferenze, che magari non si conoscevano prima.

- Aumento delle vendite: in molte piattaforme di e-commerce i sistemi di raccomandazione possono aumentare le vendite suggerendo ai clienti prodotti correlati (che spesso sono acquistati insieme) oppure prodotti simili a quello acquistato.

6.2 Punti di debolezza dei sistemi di raccomandazione

Nonostante i numerosissimi vantaggi, i sistemi di raccomandazione posso avere alcuni punti di debolezza, oltre a quelli già trattati in precedenza:

- Filter Bubble: i sistemi di raccomandazione tendono a creare una "bolla di filtraggio" attorno agli utenti, ovvero suggeriscono solo contenuti simili a quelli che l'utente ha già consumato. Questo può limitare la scoperta di nuovi contenuti da parte dell'utente.

- Problema della privacy: affinché il sistema fornisca delle raccomandazioni accurate molto spesso vengono richiesi dati personali dell'utente e questo potrebbe portare ad un uso non autorizzato di dati sensibili.

- Problema del mainstream: i sistemi di raccomandazione molto spesso tendono a raccomandare contenuti che siano conformi ai gusti e alle preferenze della maggior parte degli utenti. Questo porta a una mancanza di diversità nelle raccomandazioni e a una concentrazione solo su item popolari o mainstream.

- Problema dell'overfitting: molti sistemi di raccomandazione si adattano troppo alle scelte e comportamenti passati degli utenti e perdono la capacità di fornire raccomandazioni diversificate e innovative.

6.3 Punti di forza di scikit-learn

Nella fase di implementazione dei due modelli di raccomandazione presentati è stata utile la libreria Scikit-learn. Vediamo in seguito alcuni punti di forza di questa libreria:

- Semplicità d'uso: Scikit-learn è progettato con lo scopo di essere *user-friendly* con una sintassi chiara e comprensibile per tutti.

- Vasta gamma di algoritmi per il calcolo della similarità: come abbiamo potuto testare nelle simulazioni, Scikit-learn offre una vasta gamma di algoritmi di machine learning per il calcolo della similarità tra items. In particolare grazie alla classe `TfidfVectorizer` del modulo `sklearn.feature_extraction.text` è stato possibile estrarre informazioni utili da un insieme di descrizioni e ricavare i coefficienti di similarità desiderati.

- Scalabilità: scikit-learn è progettato per gestire facilmente grandi quantità di dati. Nel nostro caso, ha gestito un dataset di cibi (food dataset) di 400 entries in modo efficace e preciso.

- Qualità della documentazione: scikit-learn ha una documentazione[1] chiara e ben strutturata. Ogni funzionalità è spiegata nel dettaglio per comprenderne al meglio ogni singola parte.

6.4 Punti di debolezza di scikit-learn

Fortunatamente, gli svantaggi sono in numero molto minore rispetto ai vantaggi nell'uso di Scikit-learn per programmare modelli di raccomandazione. Vediamone alcuni:

- Limiti nel modello di filtraggio collaborativo: Scikit-learn è una libreria generica per il machine learning e non è specializzata in algoritmi di filtraggio collaborativo nel caso dei sistemi di raccomandazione.

- Problema nella gestione di una grande quantità di dati: la libreria scikit-learn riscontra molte difficoltà nella gestione di dataset di grandi dimensioni. Questo perché scikit-learn nasce per la progettazione su una singola macchina mentre, per evitare tempi di computazione e complessità troppo elevati, potrebbe essere più appropriato utilizzare framework distribuiti.

- Non include deep learning: scikit-learn offre una vasta gamma di algoritmi tradizionali di machine learning ma non è ancora specializzato negli algoritmi di deep learning. Per progetto che utilizzano reti neurali profonde si potrebbe ricorrere a librerie come TensorFlow o PyTorch.

[1] https://scikit-learn.org/0.21/documentation.html

6.5 Utilizzo di scikit-learn per la creazione di modelli di raccomandazione

Nella seguente tesi, in cui sono stati studiati, analizzati e simulati i sistemi di raccomandazione, è possibile affermare che le raccomandazioni ottenute sono in linea con le aspettative. Nel caso del *Key Content-Based* sono risultate coerenti con le parole chiave inserite. Questo, in qualche modo, simula il comportamento di molte piattaforme e-commerce come ad esempio Amazon: i clienti della piattaforma inseriscono sulla barra di ricerca alcune parole e ottengono risultati in base alle parole chiave scritte. Nel caso del *User Content-Based* le raccomandazioni sono state accurate in base ai voti che l'utente aveva dato in precedenza. Nonostante il successo di questi test rimangono alcune migliorie o challenge da apportare al codice affinché rappresenti una situazione del mondo reale. Uno di questi potrebbe essere l'inserimento dello status sportivo e dei propri parametri biometrici come input del programma in modo da avere delle raccomandazioni più accurate e personalizzate. In un secondo momento, magari, questi parametri potrebbero essere ottenuti in tempo reale tramite l'uso di smart watch o dispositivi fisiologici. Un altro possibile miglioramento del sistema potrebbe essere dato integrando le funzionalità di entrambi i programmi in un software unico in modo da poter ricevere delle raccomandazioni sia in base alla tipologia sia in base ad alcune parole chiave inserite.

Bibliografia

[1] Ko, H., Lee, S., Park, Y., Choi, A. (2022). *A survey of recommendation systems: recommendation models, techniques, and application fields.* Electronics, 11(1), 141.

[2] Roy, Deepjyoti, and Mala Dutta. *A systematic review and research perspective on recommender systems.* Journal of Big Data 9.1 (2022): 59.

[3] HUG, Nicolas. *Surprise: A Python library for recommender systems.* Journal of Open Source Software, 2020, 5.52: 2174.

[4] BUITINCK, Lars, et al. *API design for machine learning software: experiences from the scikit-learn project.* arXiv preprint arXiv:1309.0238, 2013.

[5] Loeb, S.; Terry, D. Information Filtering. *Commun. ACM* 1992, 35, 26-28

[6] Goldberg, D.; Nichols, D.; Oki, B.M; Terry, D. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM* 1992, 35, 61-70

[7] EKSTRAND, Michael D. Lenskit for python: Next-generation software for recommender systems experiments. In: Proceedings of the 29th ACM international conference on information knowledge management. 2020. p. 2999-3006.

[8] IYENGAR, Sheena S.; LEPPER, Mark R. *When choice is demotivating: Can one desire too much of a good thing?.* Journal of personality and social psychology, 2000, 79.6: 995.

[9] EKSTRAND, Michael D. *Lenskit for python: Next-generation software for recommender systems experiments.* In: Proceedings of the 29th ACM international conference on information knowledge management. 2020. p. 2999-3006.

[10] Receiver operating characteristic (ROC)
Fonte: https://it.wikipedia.org/wiki/Receiver_operating_characteristic