Immagine che contiene testo, emblema, logo, simbolo

Descrizione generata automaticamente

**App Advisor**

**Raccomandazione di app e predizione del successo**

Gruppo di lavoro

* Caterina Miranda, 736546, [c.miranda2@studenti.uniba.it](mailto:c.miranda2@studenti.uniba.it)
* Nicolò de Cillis,736575, [n.decillis1@studenti.uniba.it](mailto:n.decillis1@studenti.uniba.it)
* Roberta De Tullio, 737821, [r.detullio6@studenti.uniba.it](mailto:r.detullio6@studenti.uniba.it)

Repository:

<https://github.com/nicodecillis/ICon-2324>

Ingegneria della conoscenza

AA 2023-24

Sommario

Introduzione ..............................................................................................................................................,........... 3

Elenco argomenti di interesse.............................................................................................................................. 3

1. Preprocessing e bilanciamento del dataset..................................................................................................... 3

1.1. Sommario................................................................................................................................................... 3

1.2. Strumenti utilizzati..................................................................................................................................... 3

1.3. Decisioni di Progetto.................................................................................................................................. 4

1.3.1. Operazioni di preprocessing del dataset……………………………………………………………..…………………..… 4

1.3.2. Feature Engineering …………………………………………………………………………………………………………………… 4

1.3.3. Operazioni di bilanciamento del dataset …………………………………………………………………………………….5

1.4. Valutazione ................................................................................................................................................ 6

2. Knowledge Base …………………………………………………………….............................................................................. 10

2.1. Sommario ................................................................................................................................................ 10

2.2. Strumenti utilizzati .................................................................................................................................. 10

2.3. Decisioni di Progetto ............................................................................................................................... 10

2.3.1. Fatti.................................................................................................................................................. 10

2.3.2. Clausole .......................................................................................................................................... 10

2.3.3. Esempi di utilizzo della KB .............................................................................................................. 13

3. Apprendimento supervisionato ...................................................................................................................... 14

3.1. Sommario ................................................................................................................................................ 14

3.2. Strumenti utilizzati .................................................................................................................................. 14

3.3. Decisioni di Progetto ............................................................................................................................... 14

3.3.1. Decision Tree .................................................................................................................................. 15

3.3.2. K-Nearest Neighbors ...................................................................................................................... 16

3.3.3. Gaussian Naive Bayes ..................................................................................................................... 17

3.3.4. Support Vector Machine ................................................................................................................ 18

3.3.5. Random Forest ............................................................................................................................... 19

3.3.6. Ada Boost ....................................................................................................................................... 21

3.3.7. Neural Network .............................................................................................................................. 22

3.4. Valutazione ............................................................................................................................................. 23

4. Apprendimento non supervisionato .............................................................................................................. 24

4.1. Sommario ……………………………………………………………………………………………………………………………………….. 24

4.2. Decisioni di Progetto ……………………………………………………………………………………………………………………….. 24

4.2.1. Metodo del gomito ………………………………………………………………………………………………………………… 24

4.2.2. Recommender System ……………………………………………………………………………………………………………. 25

4.2.3. Esempio di utilizzo ………………………………………………………………………………………………………………….. 25

5. Belief Network ………………………………………………………………………………………………………………………………………… 27

5.1. Sommario ………………………………………………………………………………………………………………………………………… 27

5.2. Strumenti utilizzati …………………………………………………………………………………………………………………………… 27

5.3. Decisioni di progetto ……………………………………………………………………………………………………………………….. 27

5.3.1. Esempio di utilizzo ………………………………………………………………………………………………………………….. 30

6. Conclusioni ...................................................................................................................................................... 31

7. Riferimenti Bibliografici ................................................................................................................................... 31

Introduzione

Google Play Store è una delle più grandi piattaforme per la distribuzione e l'accesso alle applicazioni mobili e offre una vasta gamma di applicazioni che rispondono a diverse esigenze e interessi.

Il dataset [Google Play Store Apps](https://www.kaggle.com/datasets/gauthamp10/google-playstore-apps), proveniente da Kaggle, fornisce una raccolta completa di informazioni su circa due milioni di applicazioni disponibili sul Google Play Store. Questo dataset comprende un'ampia gamma di dettagli, tra cui le categorie di app, le valutazioni degli utenti, il numero di installazioni, le valutazioni dei contenuti, i prezzi e altro ancora.

Il nostro caso di studio si pone l’obiettivo di sfruttare le informazioni presenti nel dataset per consigliare all’utente applicazioni disponibili sullo store in base alle preferenze espresse, predire il successo di una applicazione non ancora presente sul mercato e per realizzare una base di conoscenza che consenta l’inferenza di nuove informazioni.

Elenco argomenti di interesse

* **Rappresentazione e ragionamento razionale**: creazione e interrogazione di una Knowledge Base in Prolog utilizzata per il ragionamento a partire dai dati presenti nel dataset e per l’inferenza di nuove informazioni.
* **Apprendimento supervisionato**: predizione del tasso di successo effettuata tramite i seguenti modelli di classificazione:
* Decision Tree
* K-Nearest Neighbors
* Gaussian Naive Bayes
* Support Vector Machine
* Random Forest
* Ada Boost
* Neural Network
* **Apprendimento non supervisionato**: individuazione del numero di cluster tramite l'elbow method e implementazione di un recommender system basato sull’utilizzo del k-means.
* **Ragionamento e incertezza**: implementazione di una Belief Network per il calcolo della probabilità di successo di un’applicazione non ancora sul mercato.

# 1. Preprocessing e bilanciamento del dataset

## 1.1. Sommario

Partendo dal dataset originale abbiamo effettuato diverse operazioni di preprocessing al fine di ridurre il numero di esempi e mantenere soltanto le features di input ritenute rilevanti.

## 1.2. Strumenti utilizzati

Sono state utilizzate le seguenti librerie:

* Pandas: libreria per la manipolazione e l’analisi dei dati in formato sequenziale e tabellare.
* [Google Play Scraper](https://pypi.org/project/google-play-scraper/): libreria per effettuare il crawling del Google Play Store.
* [Emoji](https://pypi.org/project/emoji/): libreria utilizzata per il riconoscimento di emoji all’interno del testo.

## 1.3. Decisioni di Progetto

### 1.3.1 Operazioni di preprocessing del dataset

Partendo dal dataset “playstore-apps.csv”, abbiamo provveduto alla cancellazione delle seguenti colonne: “Installs”, “Minimum Installs”, “Free”, “Currency”, “Developer Email”, “Developer Website”, “Released”, “Privacy Policy” e “Scraped Time”. In particolare, la colonna “Released” è stata rimossa in quanto molte applicazioni sullo store (comprese alcune molto famose) non specificavano la data.

Abbiamo ridenominato le colonne “Maximum Installs”, “Price” e “Size” rispettivamente in “Downloads”, “Price ($)” e “Size (MB)”.

Successivamente, data la grande mole di dati, abbiamo deciso di lasciare solo le righe relative alle app il cui numero di recensioni è maggiore di 50 e i cui download sono maggiori di 1000 e di eliminare le app il cui nome era scritto in un alfabeto non latino.

Poiché vi erano delle app i cui campi “App Name”, “Rating” e “Rating Count” erano vuoti e considerando che l’ultimo aggiornamento del dataset risale al 2021, l’unico modo che abbiamo trovato per ottenere tali informazioni mancanti è stato quello di prelevare le informazioni attualmente presenti nello store tramite la libreria Google Play Scraper. Tuttavia, alcune di queste app non sono più disponibili sullo store e pertanto sono state eliminate dal dataset.

Dopo aver risolto le inconsistenze tra “Rating Count” e “Download” (per evitare casi in cui ci fossero più recensioni che installazioni) abbiamo eliminato anche la colonna “Rating Count”.

Lo scraper non forniva informazioni in merito alle colonne “Minimum Android” e “Size (MB)”; pertanto abbiamo deciso nel primo caso di impostare “Varies with device” in maniera generica per le applicazioni che non riportavano questo dato e nel secondo caso di aggiungere manualmente la dimensione delle applicazioni “Cuberobotics”, “Zkteco” e “Dormstudios”.

Per quanto riguarda “Content Rating” è stato prima effettuato uno scraping delle app “Unrated”, eliminando quelle non più presenti sullo store; nonostante questa operazione, tre applicazioni risultavano ancora “Unrated” e abbiamo dunque deciso di aggiungere questo dato manualmente. Si è poi deciso di raggruppare i campi relativi al “Content Rating” in questo modo:

* “Everyone” e “Unrated” in “Everyone”
* “Everyone 10+” e “Teen” in “Teen”
* “Mature 17+” e “Adults only 18+” in “Adults”

Al fine di ridurre il numero di categorie disponibili abbiamo deciso di raggrupparne alcune ottenendo un totale di ventitré categorie risultanti: "Auto & Vehicles", "Beauty", "Communication", "Creativity", "Dating", "Education", "Entertainment", "Events", "Finance", "Food & Drink", "Games", "Health & Fitness", "House & Home", "Lifestyle", "Music & Audio", "Parenting", "Personalization", "Productivity", "Reads", "Shopping", "Tools", "Travel & Navigation", "Weather".

Infine, sono state uniformate le dimensioni delle varie applicazioni assicurandoci che venissero tutte espresse in MB.

### 1.3.2. Feature Engineering

Dalle operazioni di preprocessing si è osservato come nessuna delle colonne descrivesse in maniera soddisfacente il successo di un’applicazione. Pertanto, abbiamo deciso di definire una nuova colonna che prendesse in considerazione le informazioni già presenti nel dataset nel seguente modo:

ove:



con *rating* fornito in input, *min\_rating* = 0 e *max\_rating* = 4.6



con *downloads* fornito in input, *min\_downloads* = 1000 e *max\_downloads* = 74000000

I *success\_rate* ottenuti sono stati poi arrotondati ad una cifra decimale, discretizzati per ottenere un valore fra 0 e 10 e infine salvati all’interno della colonna “Success Rate” che verrà utilizzata come feature target.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteDi seguito sono riportate le colonne del dataset originale e del dataset preprocessed-playstore-apps ottenuto dopo aver effettuato le operazioni di preprocessing:

### 1.3.3. Operazioni di bilanciamento del dataset

Come prima operazione abbiamo deciso di raggruppare i valori della colonna “Success Rate”, in quanto sbilanciati, nel seguente modo:

* Success Rate = 1: valori da 0 a 3
* Success Rate = 2: valore pari a 4
* Success Rate = 3: valori da 5 a 6
* Success Rate = 4: valori da 7 a 10

Successivamente abbiamo effettuato l’undersampling dei campioni con “Success Rate” pari a 1,2 o 3 al fine di ottenere 9000 campioni per ciascuno di questi valori. Tuttavia, poiché gli esempi con “Success Rate” pari a 4 erano inferiori a 9000, abbiamo deciso di effettuare l’oversampling di questi ultimi.

Infine, abbiamo ridenominato le quattro classi in questo modo:

* Success Rate = 1 in “Not very popular”
* Success Rate = 2 in “Mildly popular”
* Success Rate = 3 in “Popular”
* Success Rate = 4 in “Very popular”

Il risultato di queste operazioni di bilanciamento è il nuovo dataset denominato “balanced-playstore-apps.csv”.

Immagine che contiene testo, diagramma, schermata, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamente

## 1.4. Valutazione

Di seguito sono riportati i grafici prima e dopo del bilanciamento:

Immagine che contiene testo, schermata, Diagramma, diagramma

Descrizione generata automaticamente

Immagine che contiene diagramma, Diagramma, schermata, linea

Descrizione generata automaticamente

Il primo grafico mostra una distribuzione asimmetrica con un forte aumento del numero di app man mano che il numero di stelle aumenta da 1.0, con un picco intorno a 4.1 e 4.2. Dopo questo picco, c'è un calo graduale del numero di app man mano che il numero di stelle si avvicina a 5.0. Il secondo grafico mostra anch’esso un picco intorno a 4.1 e 4.2; tuttavia, il numero di app con il numero di stelle intorno a 3.0 è relativamente più alto rispetto al primo grafico.

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

Si osserva come la distribuzione delle categorie è rimasta la stessa in entrambi i casi, con la maggior parte delle app che rientrano nella categoria “Games” mentre “Events” rimane la categoria con meno app.

Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

Immagine che contiene testo, schermata, linea, Diagramma

Descrizione generata automaticamente

Nel primo grafico la categoria più scaricata risulta essere “Games”, mentre nel secondo grafico le categorie “Games” e “Tools” hanno approssimativamente lo stesso numero di downloads.

È interessante mettere a confronto questi grafici relativi ai download con quelli relativi al numero di app per ogni categoria, in quanto questo potrebbe fornirci informazioni sul successo di ciascuna categoria. Si osserva infatti che:

* Categorie come “Games” e “Tools” presentano un elevato numero di app e di download: questo suggerisce che si tratta di categorie molto popolari ma saturate
* Categorie come “Education” e “Reads” presentano invece molte app ma pochi download: si intuisce che si tratta di categorie poco popolari ma saturate
* Categorie come “Productivity” con poche app ma molto popolari: riteniamo che sia questa la categoria su cui conviene investire in quanto vi è più probabilità che l’app diventi di successo
* Categorie come “Events”, “Parenting” e “Beauty” con poche app e pochi download: sarebbe utile condurre uno studio per capire se il motivo della bassa popolarità è da ricondurre ad uno scarso interesse degli utenti verso la categoria oppure verso le app attualmente disponibili nella stessa

Immagine che contiene testo, schermata, quadrato, Rettangolo

Descrizione generata automaticamente

Dalla matrice di correlazione si osserva che gli elementi che più influenzano il success rate sono “Rating”, “Downloads”, “In App Purchases” e “Editors Choice”; in particolare il success rate è direttamente proporzionale ai primi due.

# 2. Knowledge Base

## 2.1. Sommario

## A partire dal dataset preprocessato, abbiamo realizzato una base di conoscenza scritta in Prolog con l’obiettivo di fornire all’utente un’interfaccia per esplorare i dati disponibili e delle statistiche relative al dataset attraverso delle query e inferire nuova conoscenza per la creazione di un nuovo dataset su cui svolgere l’apprendimento.

## 2.2. Strumenti utilizzati

È stata utilizzata la libreria Python PySWIP che permette l’integrazione della programmazione logica basata su Prolog in ambiente Python.

## 2.3. Decisioni di Progetto

Per la creazione della base di conoscenza abbiamo definito dei fatti e delle clausole riportate di seguito:

### 2.3.1 Fatti

app\_name(Id, Name)  
app\_rating\_price(Name, Rating, Price)  
app\_developer(Name, Developer)  
app\_developer\_downloads(Name, Developer, Downloads)  
app\_rating\_downloads(Name, Rating, Downloads)  
app\_category(Name, Category)  
app\_category\_price(Name, Category, Price)  
app\_category\_edchoice(app\_name, Category, Editors\_choice)  
app\_category\_edchoice\_downloads(Name, Category, Editors\_choice, Downloads)  
app\_category\_downloads(Name, Category, Downloads)  
app\_category\_rating(Name, Category, Rating)  
app\_price\_downloads(Name, Price, Downloads)  
app\_category\_developer\_success(Name, Category, Developer, Success\_rate)  
app\_success\_rating\_downloads(Name, Success\_rate, Rating, Downloads)

### 2.3.2. Clausole

Clausola che restituisce i primi N elementi di una lista:

Immagine che contiene testo, Carattere, schermata, numero

Descrizione generata automaticamente

Clausola che conta le occorrenze di un elemento in una lista:

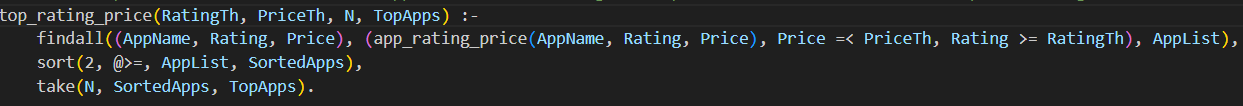
Immagine che contiene testo, schermata, Carattere

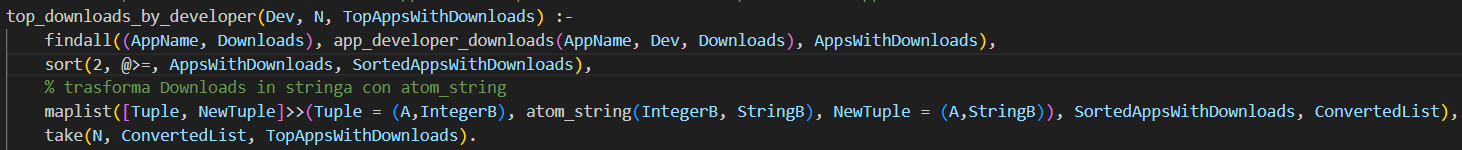
Descrizione generata automaticamente

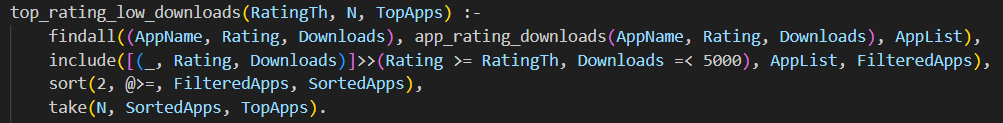
Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteClausola che conta il numero totale di app di uno sviluppatore:

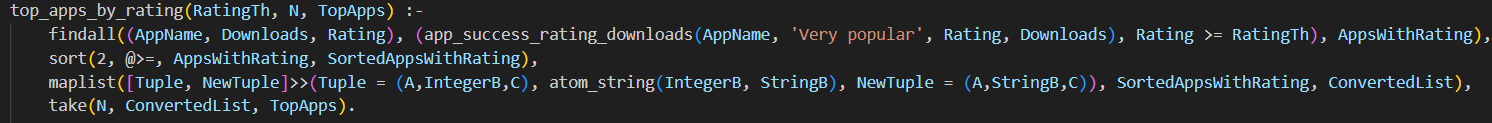
Clausola che consente di ottenere una lista di N app entro una certa soglia di prezzo e con una valutazione superiore o uguale a un certo valore:



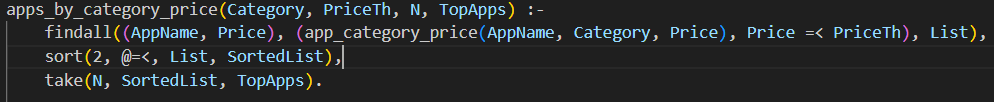
Clausola che consente di ottenere una lista di N app di uno sviluppatore dato ordinate per numero di download:

Clausola che consente di ottenere una lista di N app con rating maggiore o uguale ad un certo valore ma poco scaricate:

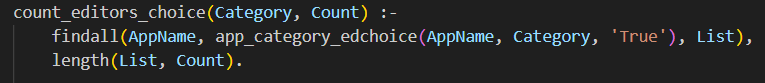
Clausola che consente di trovare N app di successo e con valutazione superiore o uguale a un certo valore ordinate per download:

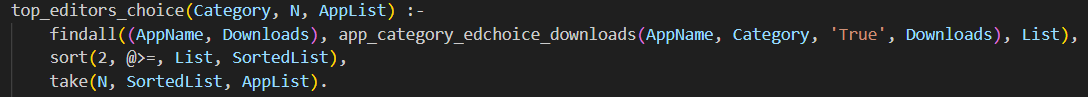


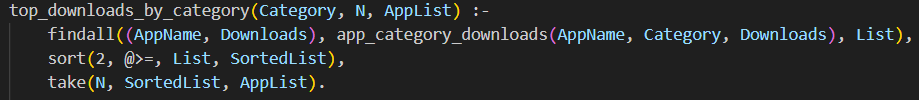
Clausola che consente di ottenere una lista di N app sotto una certa soglia di prezzo e di una certa categoria:



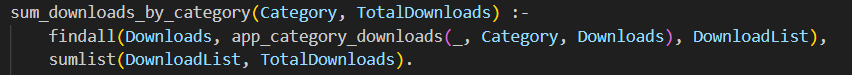
Clausola che conta il numero di app di una specifica categoria che sono Editor’s Choice:

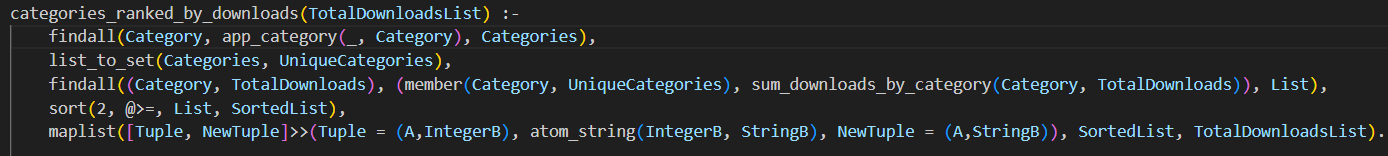


Clausola che restituisce una lista di N app di una specifica categoria che sono Editor’s Choice e ordinate per download:

Clausola che restituisce una lista di N app di una specifica categoria ordinate per numero di download:

Clausola che effettua la somma dei download delle app appartenenti ad una specifica categoria:



Clausola che ordina le categorie per numero totale di downloads:

Clausola che calcola il rating medio delle app appartenenti ad una specifica categoria:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Clausola che calcola i download medi delle app appartenenti ad una specifica categoria:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

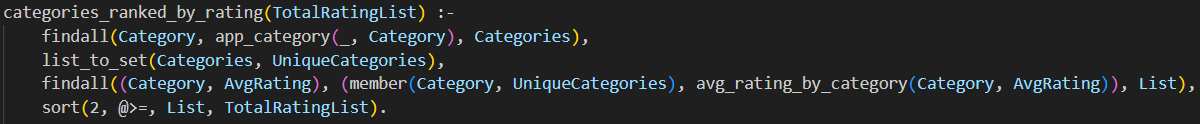
Clausola che ordina le categorie per rating medio:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteClausola che restituisce una lista di N app più costose con maggior numero di download:

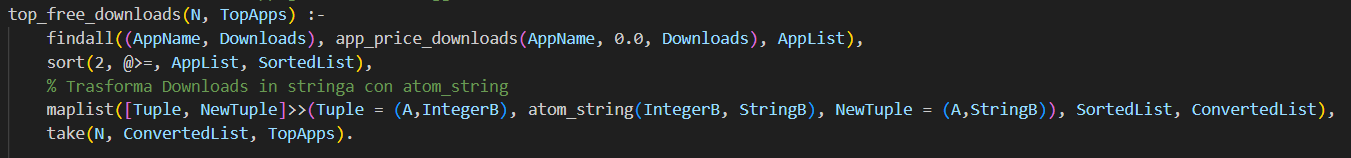
Clausola che trova le app gratuite con maggior numero di download:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteClausola che restituisce la lista degli sviluppatori con più app di successo in una specifica categoria:

Per inferire nuova conoscenza sono state poste delle query alla KB utilizzando le clausole “count\_editors\_ choice”, “avg\_downloads\_by\_category” e “avg\_rating\_by\_category”. Queste ultime ci hanno permesso di inserire le seguenti colonne nel nuovo dataset finalized-playstore-apps.csv:



2.3.3. Esempi di utilizzo della KB

### Immagine che contiene testo, schermata, Carattere Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

# 3. Apprendimento supervisionato

## 3.1. Sommario

## Partendo dal dataset “finalized-playstore-apps.csv”, sono stati addestrati diversi modelli di apprendimento supervisionato con l’obiettivo di cercare quello che riuscisse a predire la feature target “Success Rate” con maggiore accuratezza.

Per testare ogni modello è stata utilizzata una 10-fold cross validation e le valutazioni finali sono state calcolate come media delle prestazioni ottenute su ciascun fold.

I migliori iperparametri per ogni modello sono stati scelti testando varie combinazioni mediante la tecnica del grid search, utilizzando l’accuracy come strumento di confronto.

## 3.2. Strumenti utilizzati

Sono state utilizzate le seguenti librerie:

* Scikit-Learn: libreria che offre vari modelli di classificazione e lo StandardScaler per scalare i dati
* [Joblib](https://pypi.org/project/joblib/): utilizzata per memorizzare il modello migliore ottenuto dopo aver confrontato le accuracy dei vari modelli

## 3.3. Decisioni di Progetto

Di seguito mostriamo le varie combinazioni degli iperparametri e le prestazioni di ognuno dei modelli di apprendimento addestrati.

## 3.3.1. Decision tree

Si sono testate le diverse combinazioni dei seguenti iperparametri per il modello:

* criterion**:** misura la qualità delle suddivisioni (split).

Valori testati:

* gini: misura l’impurità di un nodo ossia la probabilità di classificare erroneamente un elemento scelto casualmente.
* entropy: misura l’entropia nei nodi.
* max\_depth: profondità massima dell'albero; limita la crescita dell'albero per evitare overfitting.

Valori testati: 5, 10, 20

* min\_samples\_split: numero minimo di campioni richiesti per dividere un nodo.

Valori testati: 10, 20, 30

* min\_samples\_leaf: numero minimo di campioni che deve contenere un nodo foglia.

Valori testati: 10, 20, 30

* max\_leaf\_nodes: numero massimo di nodi foglia dell'albero.

Valori testati: 10, 20, 30

* max\_features: percentuale o il numero massimo di features da considerare per suddividere un nodo.

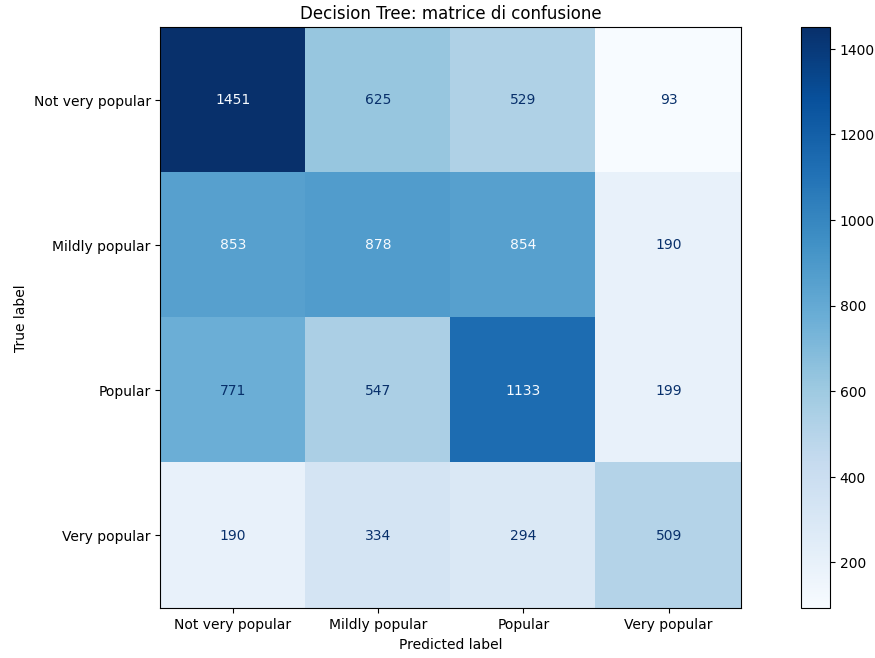
Valori testati: 0.1, 0.2

La migliore combinazione di iperparametri è stata:

* criterion: gini
* max\_depth: 20
* max\_features: 0.2
* max\_leaf\_nodes: 30
* min\_samples\_leaf: 30
* min\_samples\_split: 20

e i risultati ottenuti sono i seguenti:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.44 | 0.54 | 0.49 |
| 2 | 0.37 | 0.32 | 0.34 |
| 3 | 0.40 | 0.43 | 0.42 |
| 4 | 0.51 | 0.38 | 0.44 |
| Macro avg | 0.43 | 0.42 | 0.42 |
| Weighted avg | 0.42 | 0.42 | 0.42 |
| Accuracy | **0.42** | | |



## 3.3.2. K-nearest neighbors

Si sono testate le diverse combinazioni dei seguenti iperparametri:

* metric: metrica di distanza utilizzata per misurare la vicinanza tra i punti

Valori testati:

* minkowski: generalizzazione sia della distanza euclidea che della distanza di Manhattan
* manhattan: somma delle distanze assolute tra i punti lungo ciascuna dimensione
* euclidean: radice quadrata della somma delle distanze al quadrato tra i punti
* Chebyshev: distanza basata sulla massima differenza tra le coordinate dei punti
* n\_neighbors: numero di vicini da considerare per fare la predizione.

Valori testati: 3, 5, 7, 9, 10, 11, 13, 15

* weights: peso assegnato ai vicini; può essere uniforme o basato sulla distanza.

Valore testato:

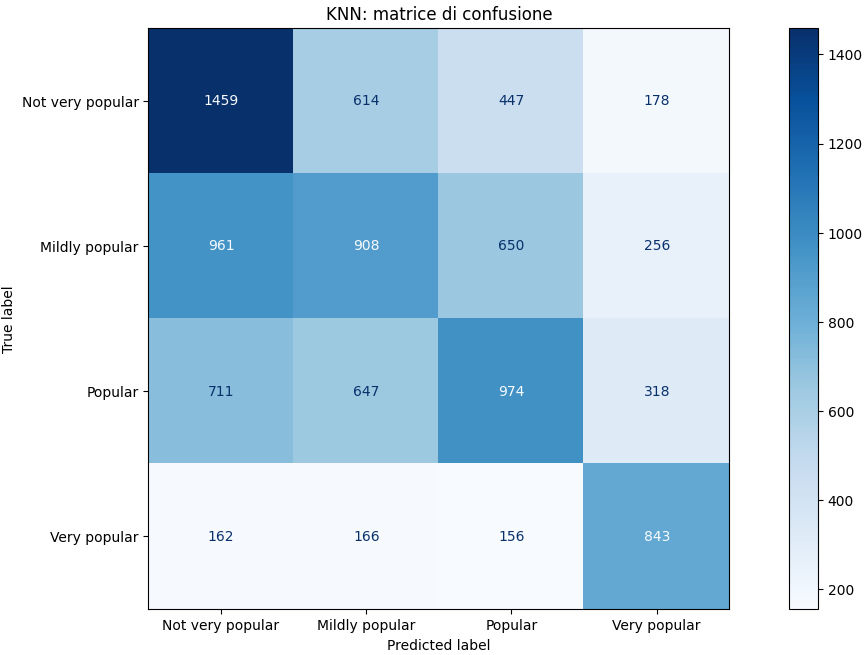
* uniform: tutti i vicini contribuiscono alla classificazione con lo stesso peso

La migliore combinazione di iperparametri è stata:

* metric: manhattan
* n\_neighbors: 15
* weights: uniform

e i risultati ottenuti sono i seguenti:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.44 | 0.54 | 0.49 |
| 2 | 0.39 | 0.33 | 0.36 |
| 3 | 0.44 | 0.37 | 0.40 |
| 4 | 0.53 | 0.64 | 0.58 |
| Macro avg | 0.45 | 0.47 | 0.45 |
| Weighted avg | 0.44 | 0.44 | 0.44 |
| Accuracy | **0.44** | | |



## 3.3.3. Gaussian Naïve Bayes

L’unico iperparametro testato per questo modello è il seguente:

* var\_smoothing: termine di stabilizzazione che aggiunge una piccola quantità di varianza a tutte le variabili per evitare divisioni per zero durante il calcolo delle probabilità

Valori testati: logspace(0, -9, num=200) ossia un array di 200 valori distribuiti su una scala logaritmica tra 10-9 e 100.

Il miglior iperparametro trovato è il seguente:

* var\_smoothing: 0.43470131581250243

e i risultati ottenuti sono i seguenti:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.49 | 0.37 | 0.42 |
| 2 | 0.38 | 0.35 | 0.36 |
| 3 | 0.40 | 0.51 | 0.44 |
| 4 | 0.45 | 0.52 | 0.49 |
| Macro avg | 0.43 | 0.44 | 0.43 |
| Weighted avg | 0.43 | 0.42 | 0.42 |
| Accuracy | **0.42** | | |

## 

## 3.3.4. Support vector machine

Sono state testate le diverse combinazioni dei seguenti iperparametri:

* C: parametro di regolarizzazione che controlla il trade-off tra classificazione corretta dei punti di addestramento e la massimizzazione del margine della funzione decisionale.

Valori testati: 0.1, 1, 10, 50

* gamma: parametro per il kernel che definisce l'influenza di un singolo punto di addestramento.

Valori testati:

* scale: calcola gamma come reciproco del prodotto tra il numero di features e la varianza dei dati
* auto: calcola gamma come reciproco del numero di features
* kernel: funzione che permette di trasformare i dati originali in uno spazio di dimensioni superiori dove diventa più facile separare i dati che non sono linearmente separabili.

Valori testati:

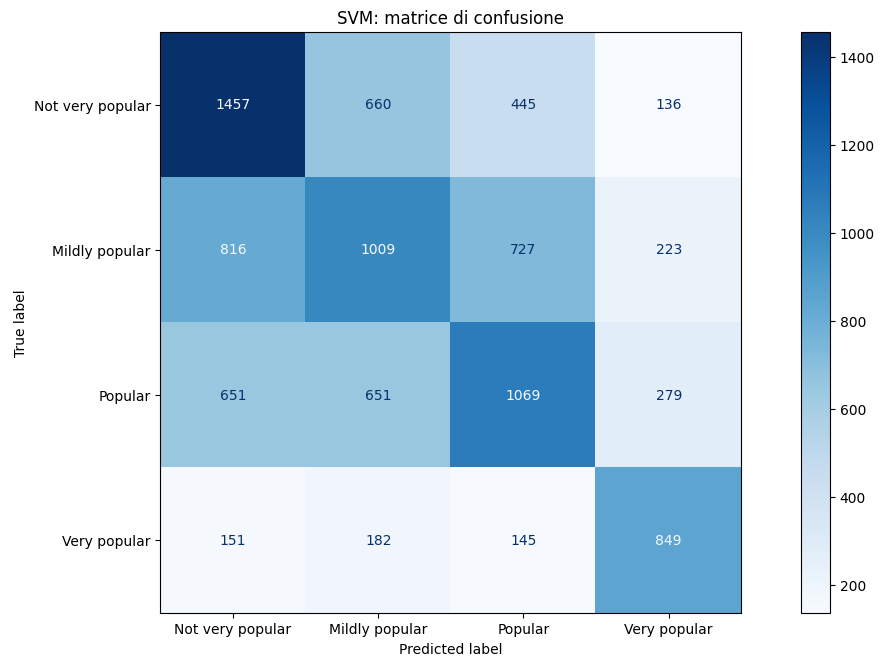
* rbf:utilizza una funzione radiale per separare i dati
* sigmoid:utilizza una funzione sigmoide per separare i dati

La migliore combinazione degli iperparametri è stata:

* C: 10
* gamma: scale
* kernel: rbf

e i risultati ottenuti sono i seguenti:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.47 | 0.54 | 0.50 |
| 2 | 0.40 | 0.36 | 0.38 |
| 3 | 0.45 | 0.40 | 0.42 |
| 4 | 0.57 | 0.64 | 0.60 |
| Macro avg | 0.47 | 0.49 | 0.48 |
| Weighted avg | 0.46 | 0.46 | 0.46 |
| Accuracy | **0.46** | | |



## 3.3.5. Random Forest

Sono state testate le diverse combinazioni dei seguenti iperparametri:

* n\_estimators: numero di alberi di decisione

Valori testati: 250, 300

* max\_depth: profondità massima di ogni albero

Valore testato: None

* min\_samples\_split: numero minimo di campioni richiesti per dividere un nodo in ogni albero.

Valori testati: 2, 4, 6

* min\_samples\_leaf: numero minimo di campioni che un nodo foglia deve contenere.

Valori testati: 1, 2, 4

* max\_features: percentuale o il numero massimo di features da considerare per dividere un nodo in ogni albero.

Valori testati: 0.1, 0.2, 0.3

La migliore combinazione degli iperparametri è stata:

* n\_estimators: 300
* max\_depth: None
* min\_samples\_leaf: 1
* min\_samples\_split: 4
* max\_features: 0.3

e i risultati ottenuti sono i seguenti:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.49 | 0.56 | 0.52 |
| 2 | 0.41 | 0.36 | 0.38 |
| 3 | 0.47 | 0.44 | 0.45 |
| 4 | 0.70 | 0.80 | 0.75 |
| Macro avg | 0.52 | 0.54 | 0.53 |
| Weighted avg | 0.49 | 0.50 | 0.49 |
| Accuracy | **0.50** | | |

## 

## 3.3.6. Ada Boost

Sono state testate le diverse combinazioni dei seguenti iperparametri:

* n\_estimators: numero di classificatori da combinare

Valori testati: 50, 100, 200, 300

* learning\_rate: fattore di riduzione del peso degli errori

Valori testati: 0.01, 0.1, 0.5, 1, 1.5

* algorithm: algoritmo utilizzato per l'aggiornamento del peso dei classificatori deboli

Valore testato:

* SAMME: algoritmo multi-classe, che utilizza la log-loss come funzione di errore
* estimator: tipo di classificatore utilizzato nel boosting

Valori testati: GaussianNB, RidgeClassifierCV, DecisionTreeClassifier,RandomForestClassifier

La migliore combinazione degli iperparametri è stata:

* n\_estimators: 100
* learning\_rate: 0.1
* algorithm: SAMME
* estimator: RandomForestClassifier

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.48 | 0.55 | 0.51 |
| 2 | 0.40 | 0.36 | 0.38 |
| 3 | 0.47 | 0.43 | 0.45 |
| 4 | 0.70 | 0.79 | 0.74 |
| Macro avg | 0.51 | 0.53 | 0.52 |
| Weighted avg | 0.49 | 0.49 | 0.49 |
| Accuracy | **0.49** | | |

e i risultati ottenuti sono i seguenti:

## 

## 3.3.7. Neural Network

Sono state testate le diverse combinazioni dei seguenti iperparametri:

* hidden\_layer\_sizes: dimensione e il numero di strati nascosti nella rete neurale

Valori testati: (10,), (50,), (120, 80, 40), (150, 100, 50)

* activation: funzione di attivazione utilizzata nei neuroni

Valori testati:

* tanh:funzione tangente iperbolica che mappa l'input nell'intervallo [-1, 1].
* relu:funzione che restituisce l'input se positivo, altrimenti restituisce zero
* solver: algoritmo utilizzato per ottimizzare i pesi della rete neurale

Valori testati:

* sgd:ottimizzatore che utilizza lo stochastic gradient descent
* adam:metodo di ottimizzazione basato su adaptive momentum
* alpha: parametro di regolarizzazione L2, che riduce l'overfitting

Valori testati: 0.001, 0.05

* learning\_rate: tasso di apprendimento, che determina quanto vengono aggiornati i pesi durante il training

Valori testati:

* constant:il tasso di apprendimento rimane fisso durante l'addestramento
* adaptive:il tasso di apprendimento diminuisce se non ci sono miglioramenti nelle epoche successive
* max\_iter: numero massimo di epoche consentite per l'ottimizzazione della rete

Valore testato:1500

La migliore combinazione degli iperparametri è stata:

* hidden\_layer\_sizes: (50,)
* activation: relu,
* solver: adam
* alpha: 0.05
* learning\_rate: constant
* max\_iter: 1500

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F1-Score |
| 1 | 0.48 | 0.52 | 0.50 |
| 2 | 0.40 | 0.33 | 0.36 |
| 3 | 0.43 | 0.43 | 0.43 |
| 4 | 0.53 | 0.63 | 0.58 |
| Macro avg | 0.46 | 0.48 | 0.47 |
| Weighted avg | 0.45 | 0.45 | 0.45 |
| Accuracy | **0.45** | | |

e i risultati ottenuti sono i seguenti:

## 

## 3.4.Valutazione

Di seguito sono riportate le accuracy dei modelli addestrati:

|  |  |
| --- | --- |
| Modello | Accuracy |
| Decision Tree | 0.42 |
| KNN | 0.44 |
| Gaussian Naive Bayes | 0.42 |
| SVM | 0.46 |
| Random Forest | 0.50 |
| Ada Boost | 0.49 |
| Neural Network | 0.45 |

Tra questi si può osservare che il modello migliore è il Random Forest, seguito da Ada Boost.

Confrontando le matrici di confusione dei vari modelli, si osserva come per tutti i modelli (tranne per il Gaussian Naive Bayes) la prima classe è classificata quasi sempre correttamente; questo potrebbe suggerire che la classe “Not very popular” è particolarmente ben rappresentata nel dataset o che i suoi campioni sono molto distintivi rispetto alle altre classi.

Confrontando invece le tabelle relative alle prestazioni, emerge come tutti i modelli mostrino precision e recall più bassi per la classe "Mildly popular" e più alti per la classe "Very popular".  
Le basse precision e recall per la classe “Mildly popular” suggeriscono che potrebbe esserci una sovrapposizione significativa tra i valori delle features di questa classe e quelli delle altre: è possibile che abbia caratteristiche simili a quelle delle altre classi, rendendo difficile per i modelli distinguerla correttamente.

Nonostante il numero ridotto di esempi, la classe “Very popular” viene invece classificata con alta precision e recall. Questo può essere dovuto a due ragioni: questa classe potrebbe essere caratterizzata da features molto distintive che la rendono facilmente distinguibile dalle altre classi; oppure, più verosimilmente, potrebbe essersi verificato overfitting sui pochi campioni disponibili di questa classe. Questo significa che i modelli potrebbero essersi legati troppo agli esempi appartenenti a questa classe piuttosto che generalizzare bene su nuovi dati.

# 4. Apprendimento non supervisionato

## 4.1. Sommario

Un metodo generale per l’apprendimento non supervisionato è il clustering che partiziona gli esempi in cluster (o classi). Ogni cluster predice i valori delle features per gli esempi contenuti nel cluster.

Nel nostro progetto la tecnica del clustering è stata impiegata per l'implementazione di un recommender system: l'obiettivo è quello di raggruppare tra loro applicazioni in cluster sulla base delle loro somiglianze e consigliare all'utente applicazioni appartenenti allo stesso cluster dell'app da lui cercata, utilizzando l'algoritmo di hard clustering K-Means.

## 4.2. Strumenti utilizzati

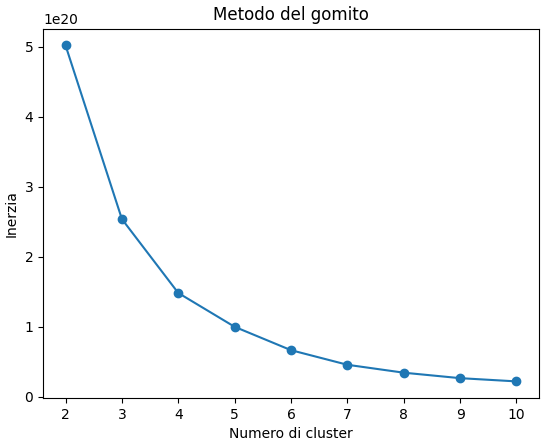
Sono state utilizzate le seguenti librerie:

* Scikit-Learn: utilizzata per l’algoritmo K-Means e per la similarità del coseno
* [Natural Language Toolkit](https://www.nltk.org/): libreria per l’elaborazione del linguaggio naturale utilizzata per la tokenizzazione
* [Gensim](https://pypi.org/project/gensim/): libreria utilizzata per il word embedding tramite Word2Vec

## 4.3. Decisioni di Progetto

### 4.3.1 Metodo del gomito

Come prima operazione, abbiamo individuato il numero ottimale di cluster in cui suddividere il dataset utilizzando il metodo del gomito, secondo cui il numero migliore di cluster è quello rappresentato dal "gomito" del grafico avente il numero di possibili cluster sull'asse delle ascisse e il valore dell'inerzia sull'asse delle ordinate, ossia la somma dei quadrati delle distanze tra ciascun punto dati e il centroide del suo cluster di appartenenza. Il grafico è il seguente:



Dal grafico si osserva come i valori migliori di k su cui applicare il K-Means siano 3, 4 e 5: abbiamo pertanto scelto di dividere il dataset in 4 cluster.

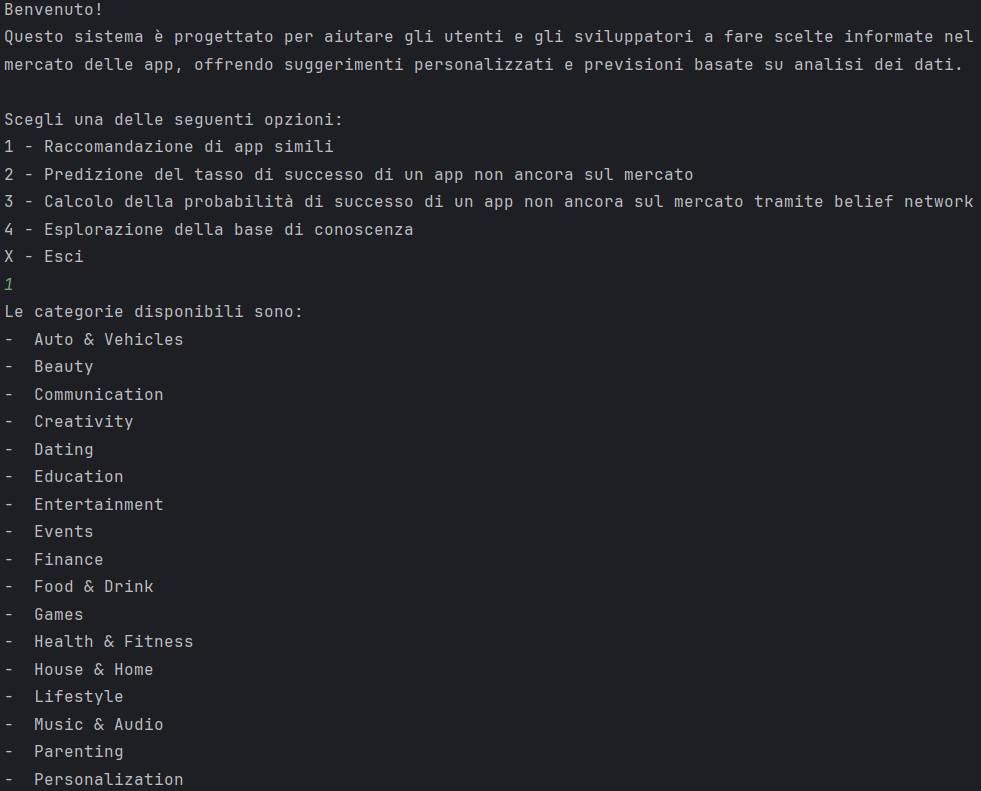
### 4.3.2. Recommender System

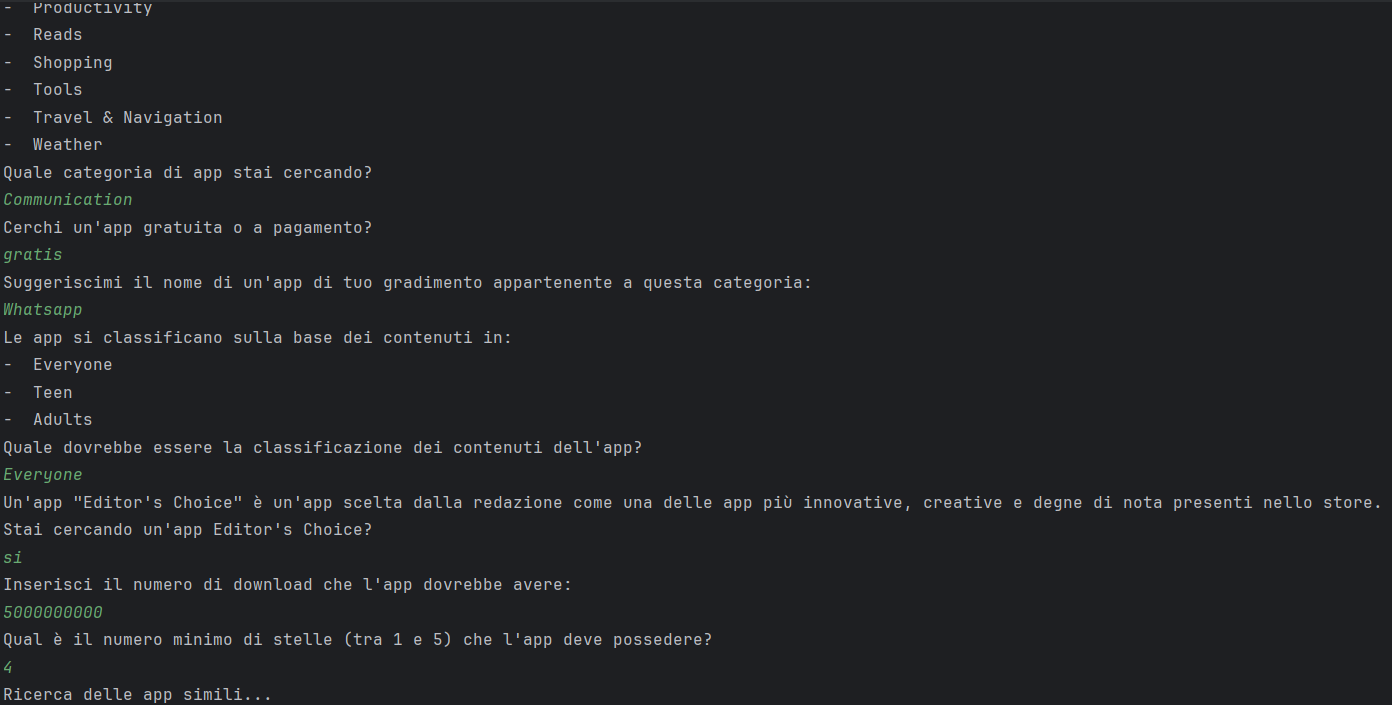
Successivamente, ci siamo occupati della realizzazione del recommender system. Per effettuare il calcolo della similarità del coseno, abbiamo per prima cosa convertito il dataset in formato numerico; in particolare, ritenendo ”App Name” una feature rilevante per il calcolo della similarità, abbiamo pensato di convertire anch'essa in una sequenza di numeri effettuando prima un'operazione di tokenizzazione e poi di embedding utilizzando la libreria Word2Vec di Gensim. In seguito, abbiamo utilizzato l’algoritmo di hard clustering K-Means per suddividere il dataset in 4 cluster ed infine abbiamo calcolato la similarità del coseno tra l’app inserita dall’utente, anch’essa convertita in formato numerico, e le app nel dataset appartenenti allo stesso cluster dell’app fornita in input per ottenere una misura della somiglianza tra queste.

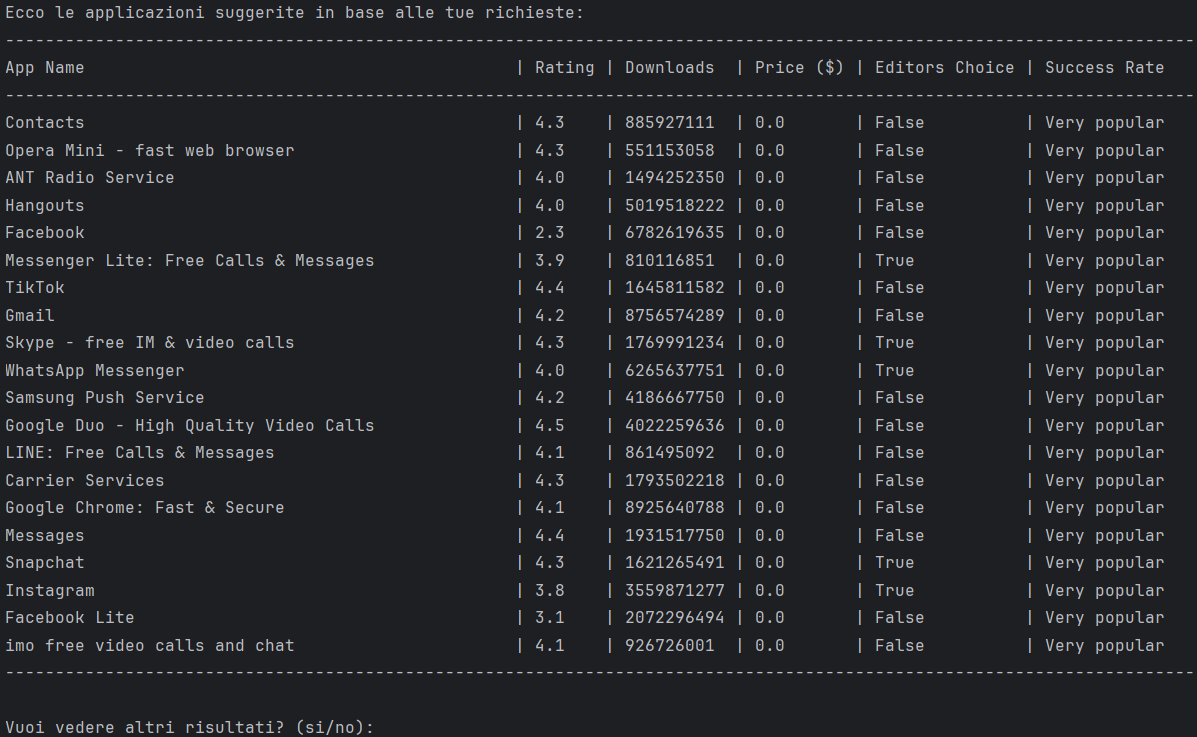
### 4.3.3. Esempio di utilizzo

Il recommender system chiede in input delle informazioni all'utente in merito al tipo di app che sta cercando. Una volta ottenute tali informazioni, l’algoritmo K-Means calcola il cluster di appartenenza dell'app descritta dall'utente, seleziona le app nel dataset appartenenti al medesimo cluster e le ordina in maniera decrescente sulla base della similarità del coseno tra l'app fornita in input e quelle selezionate dal sistema.

Di seguito è mostrato un esempio di esecuzione del recommender:







## 5. Belief Network

## 5.1. Sommario

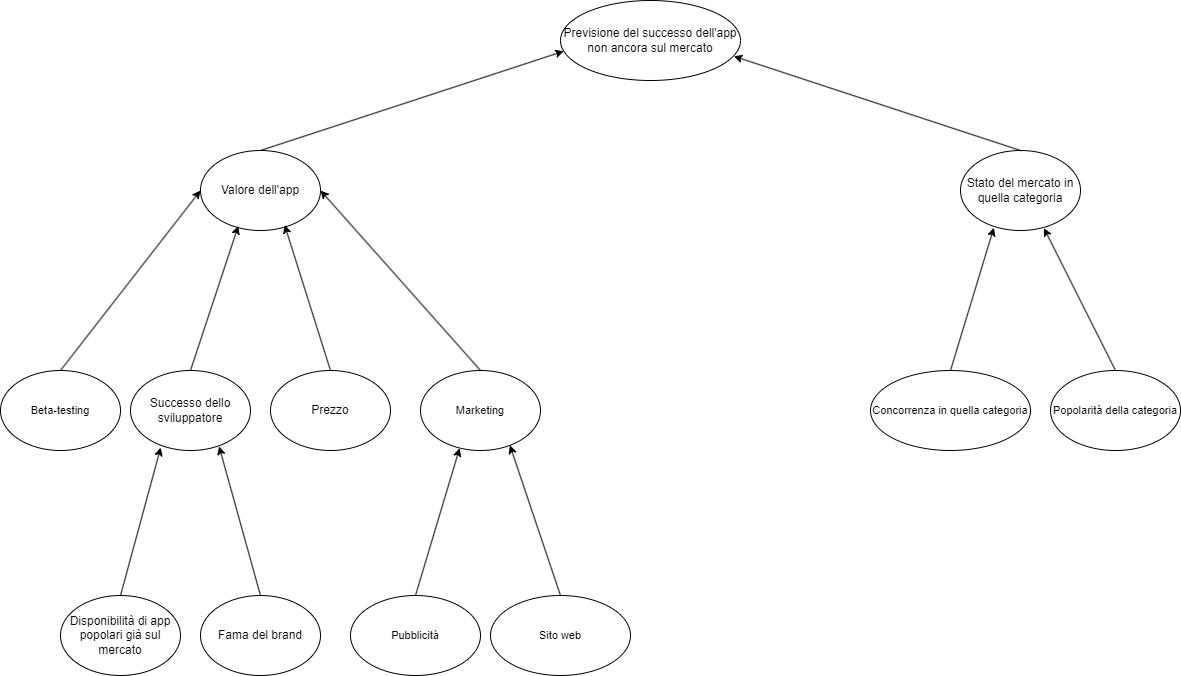
Il ragionamento probabilistico è una forma di ragionamento in presenza di incertezza che sfrutta la teoria della probabilità. Un'applicazione del ragionamento probabilistico è la belief network, un modello che rappresenta la dipendenza condizionata tra un insieme di variabili casuali. Nel nostro progetto, la rete bayesiana è stata utilizzata al fine di predire la probabilità con cui una nuova app non ancora sul mercato possa riscuotere successo una volta rilasciata.

## 5.2. Strumenti utilizzati

È stata utilizzata la libreria [pybbn](https://pypi.org/project/pybbn/) per la creazione e gestione della rete bayesiana.

## 5.3. Decisioni di Progetto

L’indipendenza condizionata in una belief network è rappresentata attraverso un grafo aciclico orientato, in cui le variabili sono organizzate secondo un ordinamento totale, che riflette le relazioni di dipendenza tra di esse. Come primo step abbiamo quindi costruito il grafo mostrando le variabili da cui riteniamo possa dipendere il successo di una nuova applicazione e le loro relazioni. Il grafo risultante è il seguente:



Successivamente, abbiamo stimato sia le probabilità a priori che quelle condizionate, riportate qui di seguito:  
P(sviluppatoreSulMercato = si) = 0,3

P(sviluppatoreSulMercato = no) = 0,7

P(affiliazioneBrand = si) = 0,19

P(affiliazioneBrand = no) = 0,81

P(successoSviluppatore = alto | sviluppatoreSulMercato = si ∧ affiliazioneBrand = si) = 0,95

P(successoSviluppatore = basso | sviluppatoreSulMercato = si ∧ affiliazioneBrand = si) = 0,05

P(successoSviluppatore = alto | sviluppatoreSulMercato = no ∧ affiliazioneBrand = si) = 0,8

P(successoSviluppatore = basso | sviluppatoreSulMercato = no ∧ affiliazioneBrand = si) = 0,2

P(successoSviluppatore = alto | sviluppatoreSulMercato = si ∧ affiliazioneBrand = no) = 0,35

P(successoSviluppatore = basso | sviluppatoreSulMercato = si ∧ affiliazioneBrand = no) = 0,65

P(successoSviluppatore = alto | sviluppatoreSulMercato = no ∧ affiliazioneBrand = no) = 0,05

P(successoSviluppatore = basso | sviluppatoreSulMercato = no ∧ affiliazioneBrand = no) = 0,95

P(pubblicità = si) = 0,2

P(pubblicità = no) = 0,8

P(sitoWeb = si) = 0,1

P(sitoWeb = no) = 0,9

P(marketing = ottimo | pubblicità = si ∧ sitoWeb = si) = 0,97

P(marketing = scarso | pubblicità = si ∧ sitoWeb = si) = 0,03

P(marketing = ottimo | pubblicità = si ∧ sitoWeb = no) = 0,8

P(marketing = scarso | pubblicità = si ∧ sitoWeb = no) = 0,2

P(marketing = ottimo | pubblicità = no ∧ sitoWeb = si) = 0,22

P(marketing = scarso | pubblicità = no ∧ sitoWeb = si) = 0,78

P(marketing = ottimo | pubblicità = no ∧ sitoWeb = no) = 0,02

P(marketing = scarso | pubblicità = no ∧ sitoWeb = no) = 0,98

P(betaTesting = si) = 0,26

P(betaTesting = no) = 0,74

P(prezzo = gratis) = 0,88

P(prezzo = aPagamento) = 0,12

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = ottimo) = 0,98

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = ottimo) = 0,02

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = scarso) = 0,76

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = scarso) = 0,24

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,82

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,18

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,64

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,36

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = ottimo) = 0,73

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = ottimo) = 0,27

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = scarso) = 0,29

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = scarso) = 0,71

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,47

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,53

P(valoreApp = alto | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,14

P(valoreApp = basso | betaTesting = si ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,86

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = ottimo) = 0,82

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = ottimo) = 0,18

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = scarso) = 0,46

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = gratis ∧ marketing = scarso) = 0,54

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,67

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,33

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,35

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = alto ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,65

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = ottimo) = 0,62

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = ottimo) = 0,38

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = scarso) = 0,12

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = gratis ∧ marketing = scarso) = 0,88

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,41

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = ottimo) = 0,59

P(valoreApp = alto | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,04

P(valoreApp = basso | betaTesting = no ∧ successoSviluppatore = basso ∧ prezzo = aPagamento ∧ marketing = scarso) = 0,96

P(concorrenza = alta) = 0,78

P(concorrenza = bassa) = 0,22

P(popolarità = molto) = 0,45

P(popolarità = poco) = 0,55

P(statoMercato = emergente | concorrenza = alta ∧ popolarità = molto) = 0,52

P(statoMercato = saturato | concorrenza = alta ∧ popolarità = molto) = 0,48

P(statoMercato = emergente | concorrenza = alta ∧ popolarità = poco) = 0,04

P(statoMercato = saturato | concorrenza = alta ∧ popolarità = poco) = 0,96

P(statoMercato = emergente | concorrenza = bassa ∧ popolarità = molto) = 0,97

P(statoMercato = saturato | concorrenza = bassa ∧ popolarità = molto) = 0,03

P(statoMercato = emergente | concorrenza = bassa ∧ popolarità = poco) = 0,53

P(statoMercato = saturato | concorrenza = bassa ∧ popolarità = poco) = 0,47

P(successoApp = alto | valoreApp = alto ∧ statoMercato = emergente) = 0,98

P(successoApp = basso | valoreApp = alto ∧ statoMercato = emergente) = 0,02

P(successoApp = alto | valoreApp = alto ∧ statoMercato = saturato) = 0,77

P(successoApp = basso | valoreApp = alto ∧ statoMercato = saturato) = 0,23

P(successoApp = alto| valoreApp = basso ∧ statoMercato = emergente) = 0,34

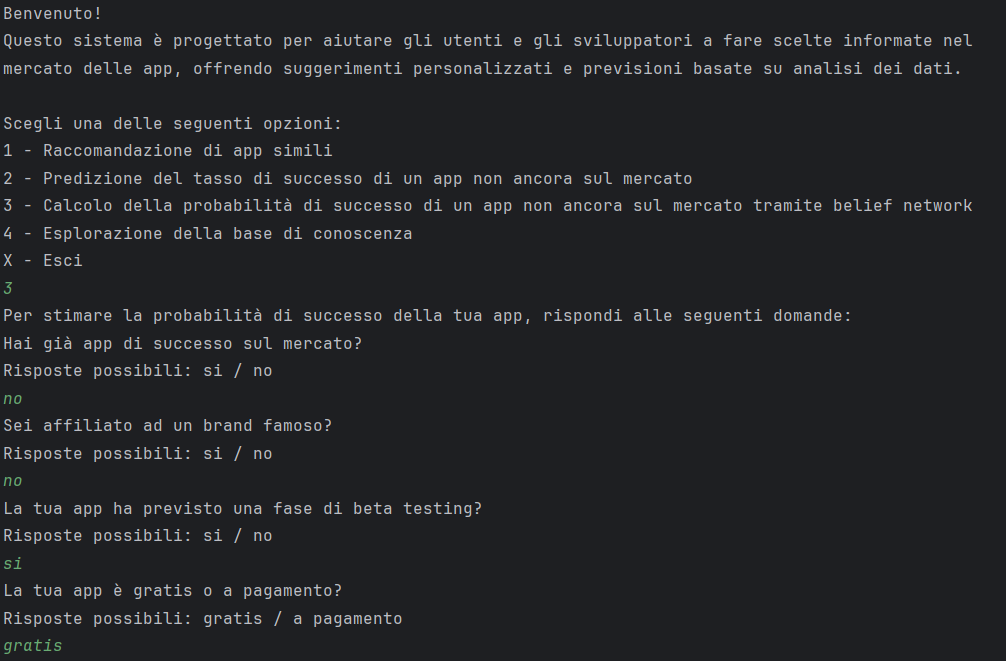
P(successoApp = basso | valoreApp = basso ∧ statoMercato = emergente) = 0,66

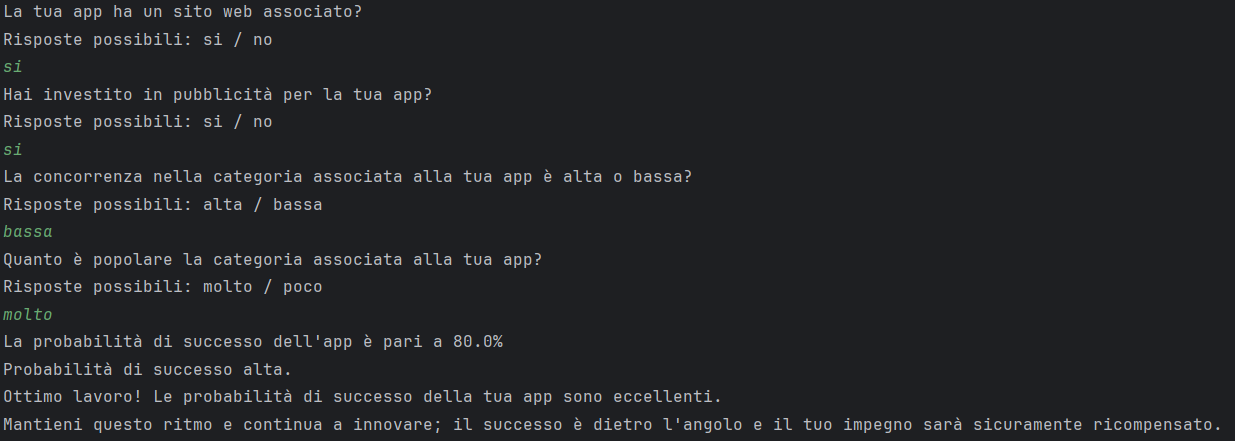
P(successoApp = alto | valoreApp = basso ∧ statoMercato = saturato) = 0,03

P(successoApp = basso | valoreApp = basso ∧ statoMercato = saturato) = 0,97

Abbiamo poi codificato la struttura della rete bayesiana utilizzando la libreria pybbn. Le probabilità condizionate vengono aggiornate dinamicamente quando l'utente fornisce nuove informazioni come evidenze, permettendo una previsione più accurata del successo dell'applicazione.

### 5.3.1 Esempio di utilizzo





# 6. Conclusioni

Il sistema realizzato soddisfa i requisiti funzionali inizialmente definiti; tuttavia, sono certamente possibili ulteriori miglioramenti a partire dalla struttura del dataset per rendere la predizione ancora più accurata.

In particolare, si è osservato come tutti i modelli mostrino un accuracy non superiore al 50%; è possibile ovviare al problema cercando features più distintive, come ad esempio delle recensioni testuali per un'analisi più accurata del sentiment verso le varie app.

Inoltre, si potrebbe ampliare il dataset con i dati del Google Play Store aggiornati al 2024 e migliorare il calcolo del "Success Rate" affinché tenga conto di altre features (es. "Rating Count" e "Category"), pesate in base alla loro influenza sul tasso di successo.

Un ulteriore miglioramento del recommender system potrebbe consistere nell'introduzione di una funzionalità che permetta all'utente di creare e personalizzare un profilo di interessi. Questo profilo verrebbe automaticamente aggiornato e affinato sulla base delle preferenze fornite in precedenza. In questo modo, il sistema potrebbe memorizzare e analizzare le preferenze individuali dell'utente, consentendogli di ricevere raccomandazioni sempre più accurate e rilevanti nel tempo, migliorando così l'esperienza complessiva di ricerca e scoperta di nuove app.

Infine, riteniamo che il sistema si presti bene alla realizzazione di un'interfaccia grafica che possa rendere la navigazione più semplice e intuitiva.

# 7. Riferimenti Bibliografici

* [Artificial Intelligence: Foundations of Computational Agents, 3rd Edition (artint.info)](https://artint.info/3e/html/ArtInt3e.html)
* [Analyzing Key Factors and Predicting App Success on the Google Play Store | by Sahithi Arnika | Medium](https://medium.com/@sahithi.arnika/analyzing-key-factors-and-predicting-app-success-on-the-google-play-store-9ad6da5b5420)
* [A Recommender System for Mobile Applications of Google Play Store](https://thesai.org/Downloads/Volume11No9/Paper_6-A_Recommender_System_for_Mobile_Applications.pdf)
* [Scikit-learn API Reference](https://scikit-learn.org/stable/api/index.html)