App Advisor

Raccomandazione di app e predizione del successo

Gruppo di lavoro

- Caterina Miranda, 736546, c.miranda2@studenti.uniba.it
- Nicolò de Cillis,736575, n.decillis1@studenti.uniba.it
- Roberta De Tullio, 737821, r.detullio6@studenti.uniba.it

Repository:
https://github.com/nicodecillis/ICon-2324

Ingegneria della conoscenza

AA 2023-24

Sommario

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, 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: libreria per effettuare il crawling del Google Play Store.
- 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:

$$success\_rate = \frac{normalized\_rating \; + \; normalized\_downloads}{2}$$

ove:

- $normalized\_rating = \frac{rating - min\_rating}{max\_rating - min\_rating}$

  con *rating* fornito in input, *min_rating* = 0 e *max_rating* = 4.6

- $normalized\_downloads = \frac{downloads - min\_downloads}{max\_downloads - min\_downloads}$

  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.

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

Colonne del dataset prima del pre-processing

App Name	App Id	Category	Rating	Rating Count	Installs	Minimum Installs	Maximum Installs	Free	Price
Currency	Size	Minimum Android	Developer Id	Developer Website	Developer Email	Released	Last Updated		
Content Rating	Privacy Policy	Ad Supported	In App Purchases	Editors Choice	Scraped Time				

Colonne del dataset dopo il pre-processing

App Name	App Id	Category	Rating	Downloads	Price ($)	Size (MB)	Minimum Android	Developer Id
Last Updated	Content Rating	Ad Supported	In App Purchases	Editors Choice	Success Rate			

1.3.3. Operazioni di bilanciamento del dataset

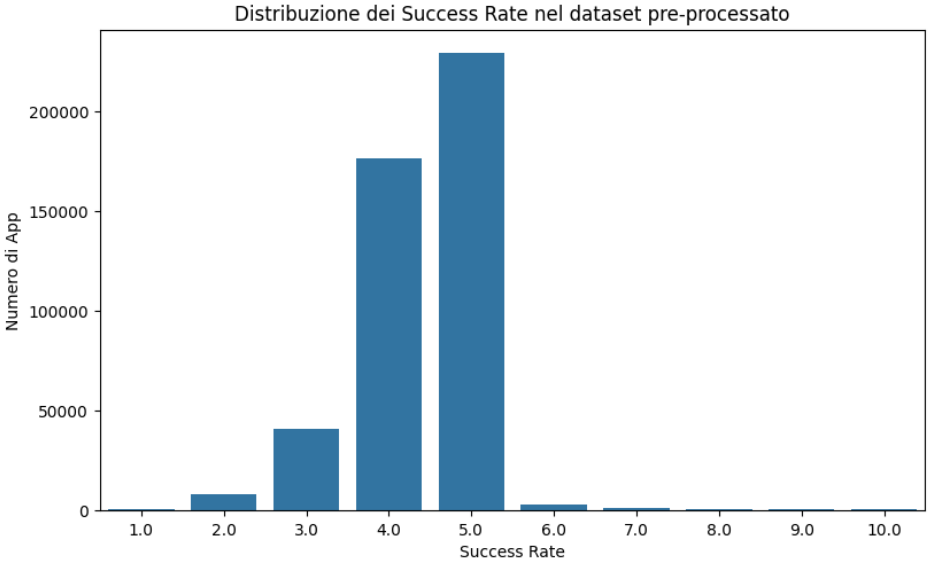
Come prima operazione abbiamo deciso di raggruppare i valori della colonna "Success Rate", in quanto sbilanciati, nel seguente modo:

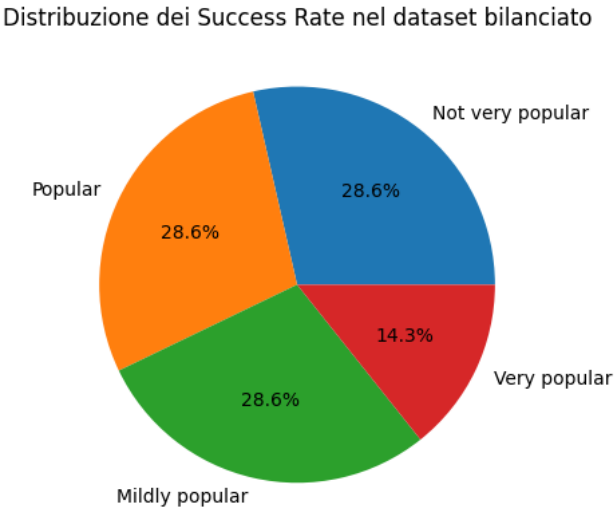
- Success Rate = 1: valori da 0 a 3
- Success Rate = 2: valore pari a 4
- Success Rate = 3: valori da 5 a 6
- Success Rate = 4: valori da 7 a 10

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"

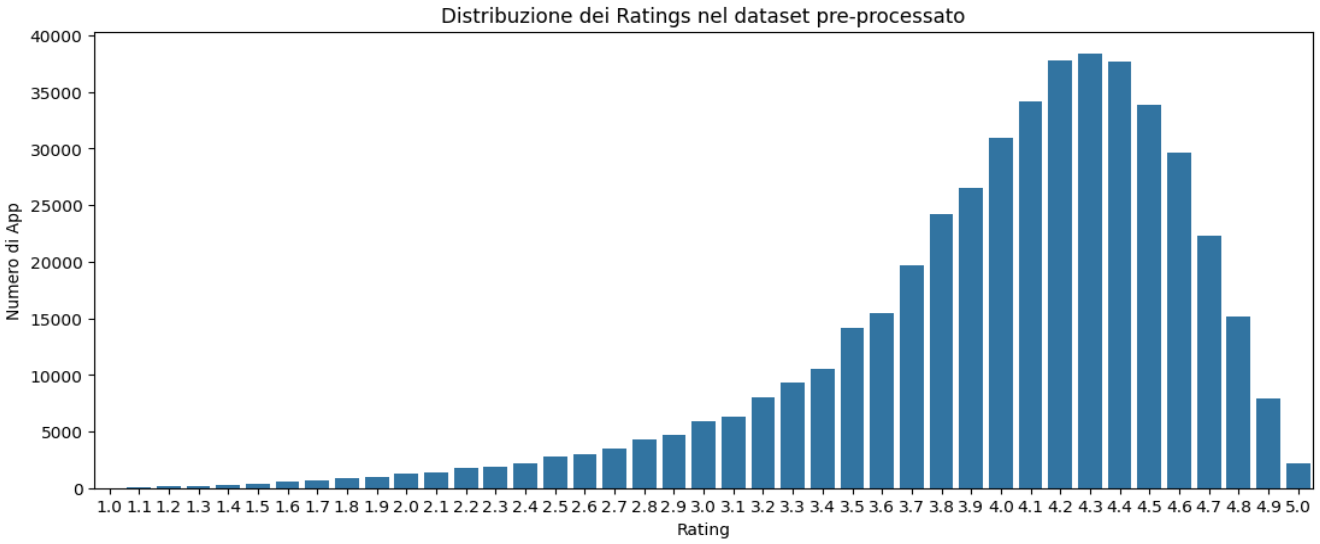
Il risultato di queste operazioni di bilanciamento è il nuovo dataset denominato "balanced-playstore-apps.csv".

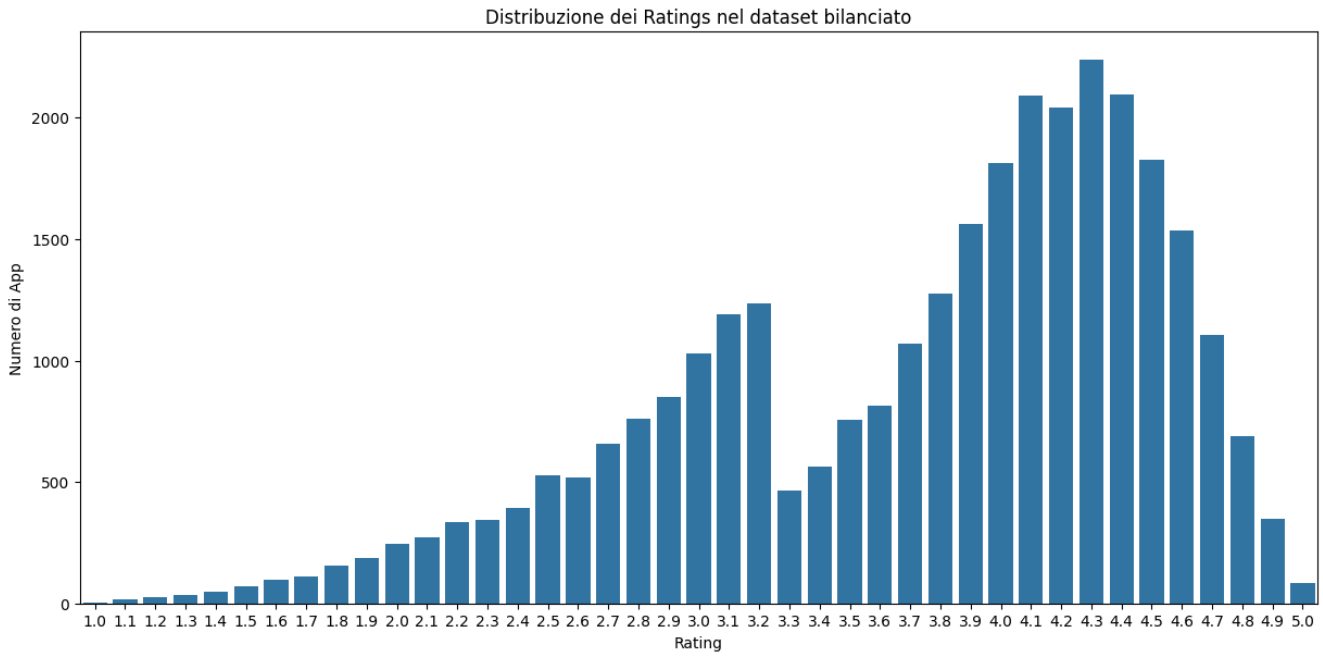




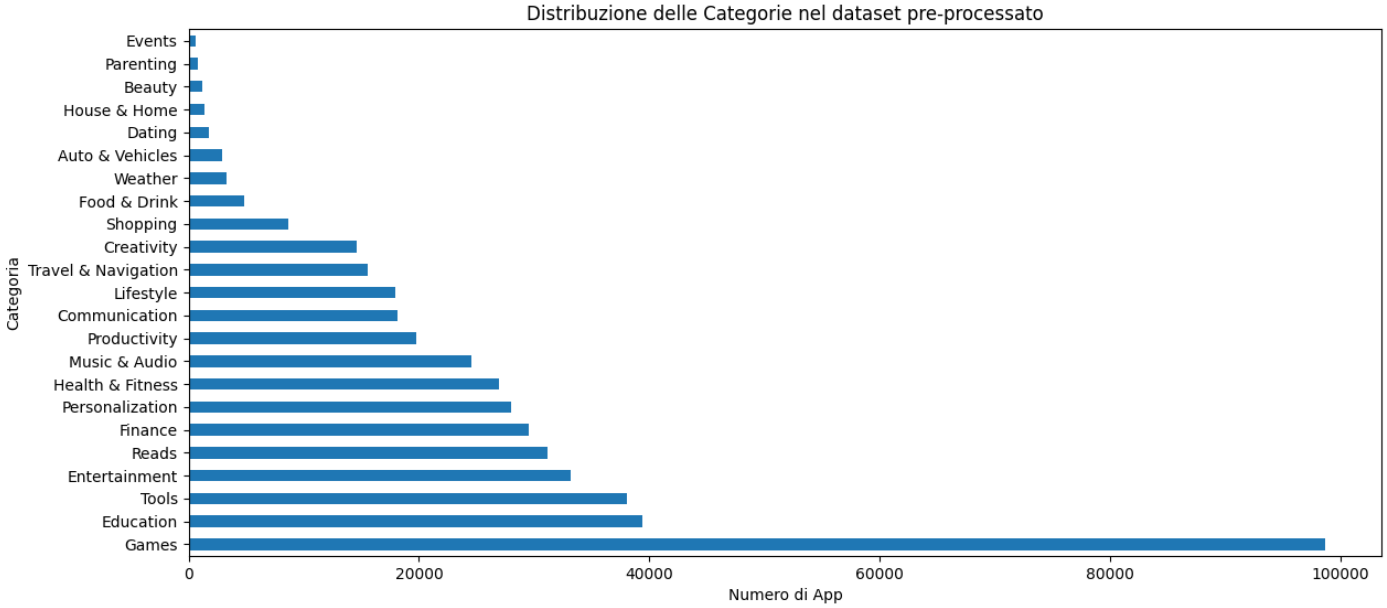
1.4. Valutazione

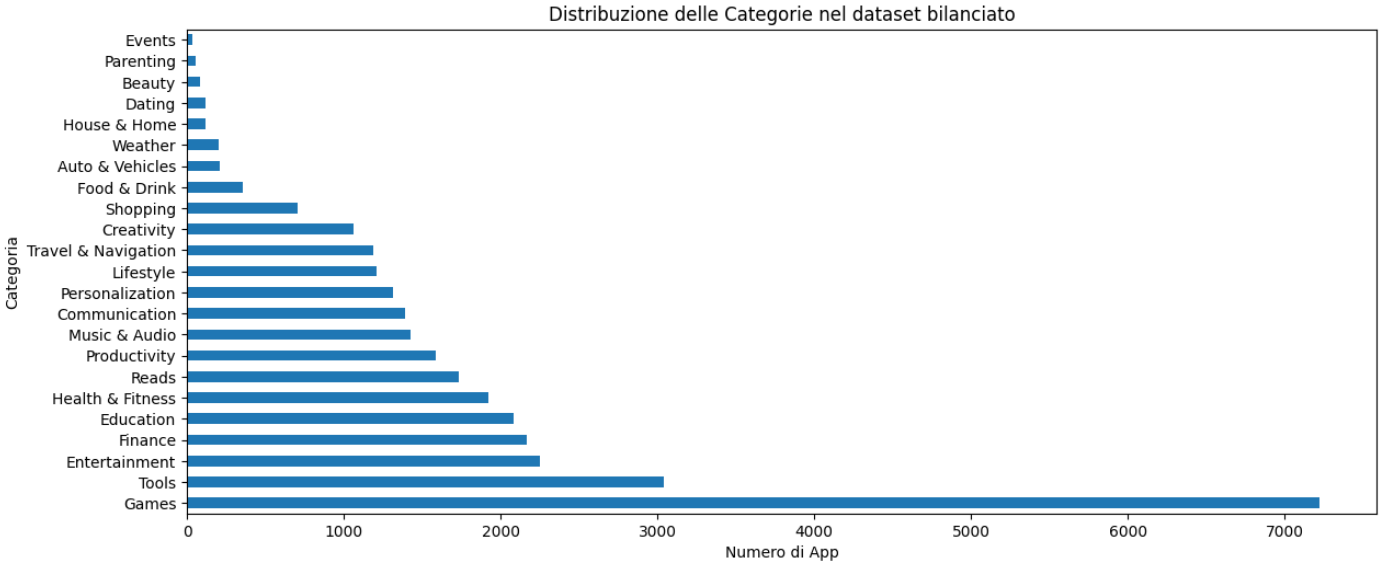
Di seguito sono riportati i grafici prima e dopo del bilanciamento:



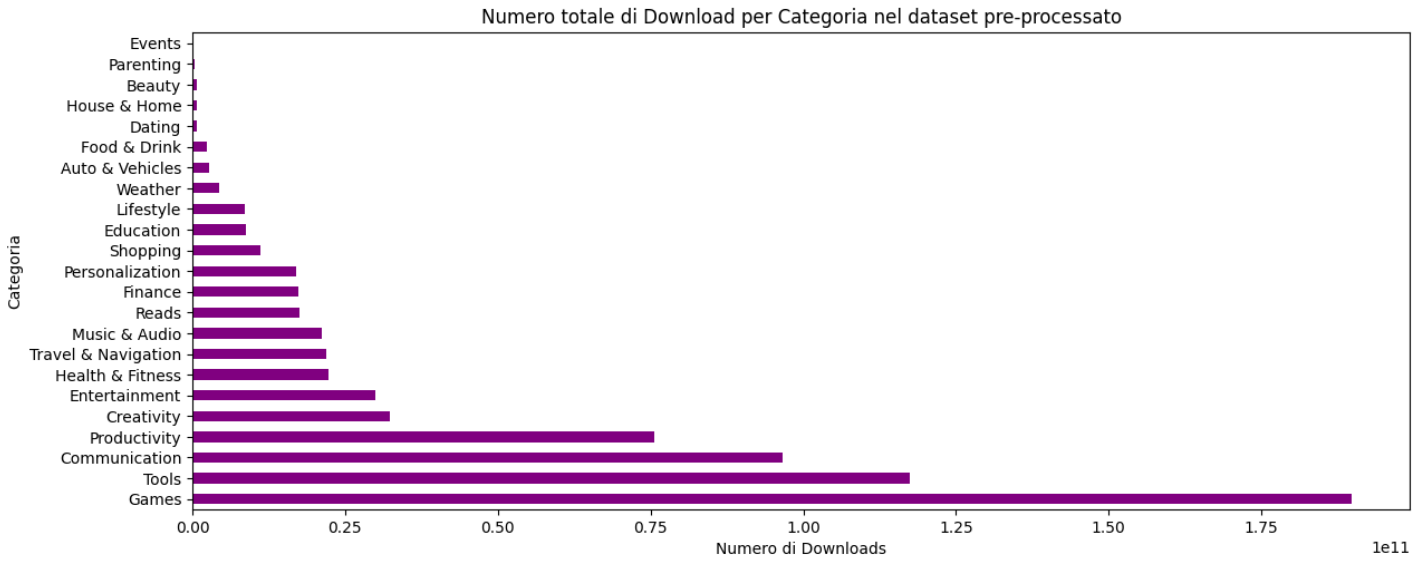


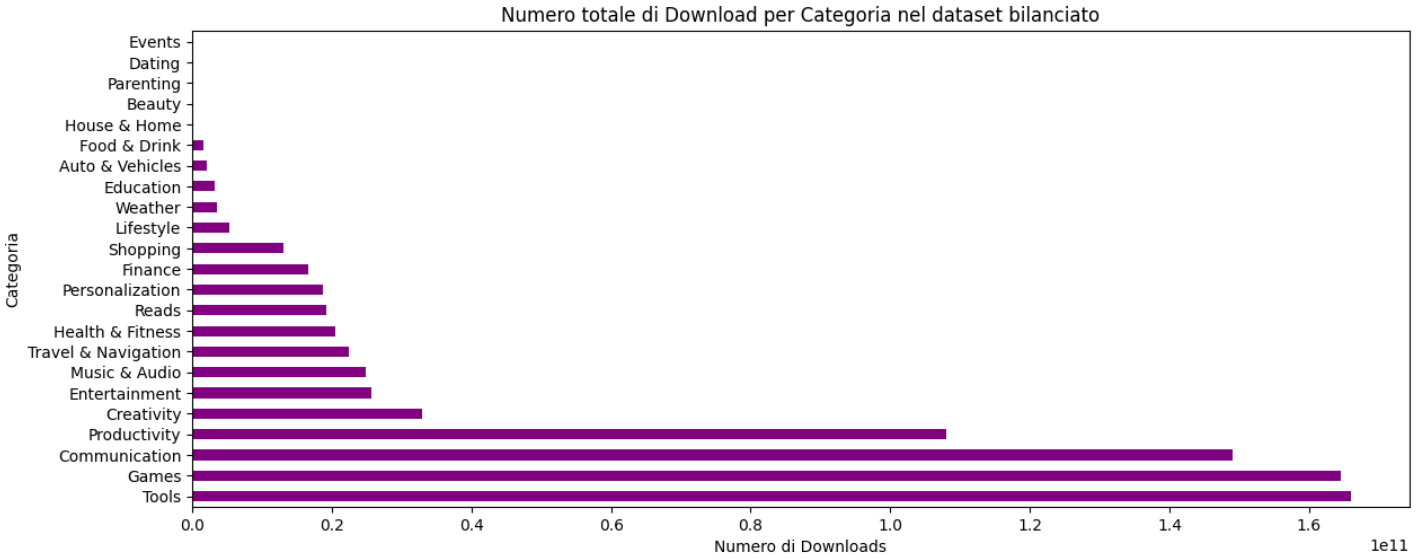
Il primo grafico mostra una distribuzione asimmetrica con un forte aumento del numero di app man mano che il numero di stelle aumenta da 1.0, con un picco intorno a 4.1 e 4.2. Dopo questo picco, c'è un calo graduale del numero di app man mano che il numero di stelle si avvicina a 5.0. Il secondo grafico 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.





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.

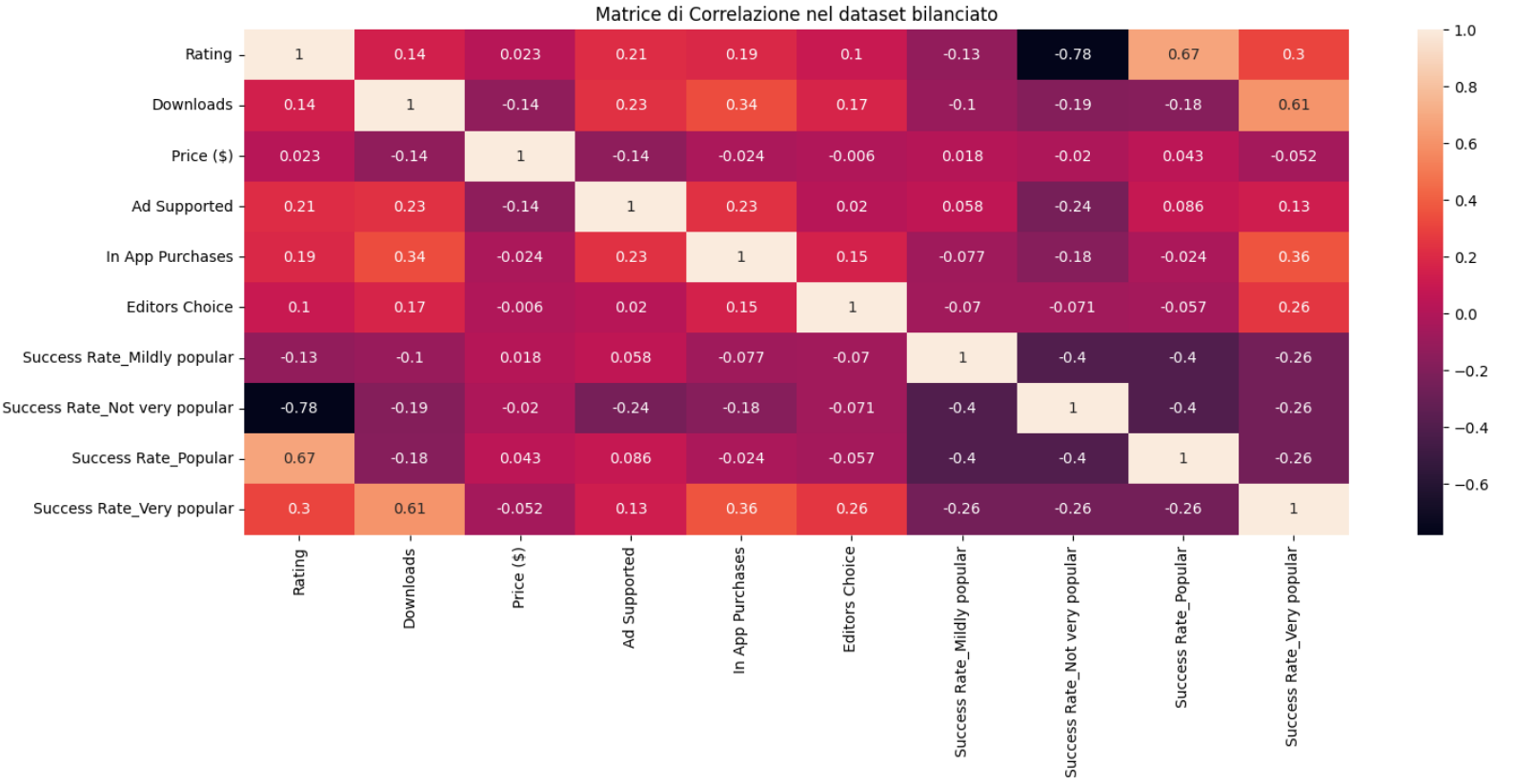




Nel primo grafico la categoria più scaricata risulta essere “Games”, mentre nel secondo grafico le categorie “Games” e “Tools” hanno approssimativamente lo stesso numero di downloads.

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



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:

```
take(0, _, []).
take(N, [Head|Tail], [Head|Taken]) :-
    N > 0,
    N1 is N - 1,
    take(N1, Tail, Taken).
```

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

```
count_occurrences(_, [], 0).
count_occurrences(Elem, [Elem|Tail], Count) :-
    count_occurrences(Elem, Tail, SubCount),
    Count is SubCount + 1.
count_occurrences(Elem, [Head|Tail], Count) :-
    Elem \= Head,
    count_occurrences(Elem, Tail, Count).
```

Clausola che conta il numero totale di app di uno sviluppatore:

```
count_apps_by_developer(Dev, Count) :-
    findall(AppName, app_developer(AppName, Dev), AppList),
    length(AppList, Count).
```

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:

```
top_rating_price(RatingTh, PriceTh, N, TopApps) :-
    findall((AppName, Rating, Price), (app_rating_price(AppName, Rating, Price), Price =< PriceTh, Rating >= RatingTh), AppList),
    sort(2, @>=, AppList, SortedApps),
    take(N, SortedApps, TopApps).
```

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

```
top_downloads_by_developer(Dev, N, TopAppsWithDownloads) :-
    findall((AppName, Downloads), app_developer_downloads(AppName, Dev, Downloads), AppsWithDownloads),
    sort(2, @>=, AppsWithDownloads, SortedAppsWithDownloads),
    % trasforma Downloads in stringa con atom_string
    maplist([Tuple, NewTuple]>>(Tuple = (A,IntegerB), atom_string(IntegerB, StringB), NewTuple = (A,StringB)), SortedAppsWithDownloads, ConvertedList),
    take(N, ConvertedList, TopAppsWithDownloads).
```

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

```
top_rating_low_downloads(RatingTh, N, TopApps) :-
    findall((AppName, Rating, Downloads), app_rating_downloads(AppName, Rating, Downloads), AppList),
    include([(_, Rating, Downloads)]>>(Rating >= RatingTh, Downloads =< 5000), AppList, FilteredApps),
    sort(2, @>=, FilteredApps, SortedApps),
    take(N, SortedApps, TopApps).
```

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

```
top_apps_by_rating(RatingTh, N, TopApps) :-
    findall((AppName, Downloads, Rating), (app_success_rating_downloads(AppName, 'Very popular', Rating, Downloads), Rating >= RatingTh), AppsWithRating),
    sort(2, @>=, AppsWithRating, SortedAppsWithRating),
    maplist([Tuple, NewTuple]>>(Tuple = (A,IntegerB,C), atom_string(IntegerB, StringB), NewTuple = (A,StringB,C)), SortedAppsWithRating, ConvertedList),
    take(N, ConvertedList, TopApps).
```

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

```
apps_by_category_price(Category, PriceTh, N, TopApps) :-
    findall((AppName, Price), (app_category_price(AppName, Category, Price), Price =< PriceTh), List),
    sort(2, @=<, List, SortedList),
    take(N, SortedList, TopApps).
```

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

```
count_editors_choice(Category, Count) :-
    findall(AppName, app_category_edchoice(AppName, Category, 'True'), List),
    length(List, Count).
```

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

```
top_editors_choice(Category, N, AppList) :-
    findall((AppName, Downloads), app_category_edchoice_downloads(AppName, Category, 'True', Downloads), List),
    sort(2, @>=, List, SortedList),
    take(N, SortedList, AppList).
```

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

```
top_downloads_by_category(Category, N, AppList) :-
    findall((AppName, Downloads), app_category_downloads(AppName, Category, Downloads), List),
    sort(2, @>=, List, SortedList),
    take(N, SortedList, AppList).
```

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

```
sum_downloads_by_category(Category, TotalDownloads) :-
    findall(Downloads, app_category_downloads(_, Category, Downloads), DownloadList),
    sumlist(DownloadList, TotalDownloads).
```

Clausola che ordina le categorie per numero totale di downloads:

```
categories_ranked_by_downloads(TotalDownloadsList) :-
    findall(Category, app_category(_, Category), Categories),
    list_to_set(Categories, UniqueCategories),
    findall((Category, TotalDownloads), (member(Category, UniqueCategories), sum_downloads_by_category(Category, TotalDownloads)), List),
    sort(2, @>=, List, SortedList),
    maplist([Tuple, NewTuple]>>(Tuple = (A,IntegerB), atom_string(IntegerB, StringB), NewTuple = (A,StringB)), SortedList, TotalDownloadsList).
```

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

```
avg_rating_by_category(Category, AvgRating) :-
    findall(Rating, app_category_rating(_, Category, Rating), RatingList),
    sumlist(RatingList, TotalRating),
    length(RatingList, N),
    AvgRatingRaw is TotalRating / N,
    format(atom(AvgRatingAtom), '~2f', [AvgRatingRaw]),
    atom_number(AvgRatingAtom, AvgRating).
```

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

```
avg_downloads_by_category(Category, AvgDownloads) :-
    findall(Downloads, app_category_downloads(_, Category, Downloads), DownloadList),
    sumlist(DownloadList, TotalDownloads),
    length(DownloadList, N),
    AvgDownloadsRaw is TotalDownloads / N,
    format(atom(AvgDownloadsAtom), '~0f', [AvgDownloadsRaw]),
    atom_number(AvgDownloadsAtom, AvgDownloads).
```

Clausola che ordina le categorie per rating medio:

```
categories_ranked_by_rating(TotalRatingList) :-
    findall(Category, app_category(_, Category), Categories),
    list_to_set(Categories, UniqueCategories),
    findall((Category, AvgRating), (member(Category, UniqueCategories), avg_rating_by_category(Category, AvgRating)), List),
    sort(2, @>=, List, TotalRatingList).
```

Clausola che restituisce una lista di N app più costose con maggior numero di download:

```
top_expensive_downloads(N, SortedByDownloads) :-
    findall((AppName, Price, Downloads), app_price_downloads(AppName, Price, Downloads), List),
    sort(2, @>=, List, SortedByPrice),
    take(N, SortedByPrice, TopApps),
    maplist([Tuple, ReversedTuple]>>(Tuple = (A,B,C), ReversedTuple = (A,C,B)), TopApps, ReversedList),
    sort(2, @>=, ReversedList, SortedByDownloads).
```

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

```
top_free_downloads(N, TopApps) :-
    findall((AppName, Downloads), app_price_downloads(AppName, 0.0, Downloads), AppList),
    sort(2, @>=, AppList, SortedList),
    % Trasforma Downloads in stringa con atom_string
    maplist([Tuple, NewTuple]>>(Tuple = (A,IntegerB), atom_string(IntegerB, StringB), NewTuple = (A,StringB)), SortedList, ConvertedList),
    take(N, ConvertedList, TopApps).
```

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

```
top_developers_by_success(Category, N, TopDevList) :-
    findall(Dev, app_category_developer_success(_, Category, Dev, 'Very popular'), Devs),
    msort(Devs, SortedDevs),
    bagof((Dev, Count), (member(Dev, SortedDevs), count_occurrences(Dev, SortedDevs, Count)), DevCountList),
    list_to_set(DevCountList, DevCountUniqueList),
    sort(2, @>=, DevCountUniqueList, SortedDevCountList),
    take(N, SortedDevCountList, TopDevList).
```

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:

Num Editors Choice in Category	Average Downloads in Category	Average Rating of Category

2.3.3. Esempi di utilizzo della KB

```
---- Esplorazione della Knowledge Base ----
1. Cerca app di uno specifico sviluppatore ordinate per download
2. Cerca app sotto un certo prezzo e con rating maggiore o uguale ad un certo valore
3. Cerca app poco scaricate ma con rating maggiore o uguale ad un certo valore
4. Cerca app di successo e con valutazione maggiore o uguale a un certo valore e ordinate per download
5. Cerca app sotto un certo prezzo e appartenenti ad una specifica categoria
6. Cerca app Editor's Choice appartenenti ad una specifica categoria e ordinate per download
7. Cerca app gratuite e con maggior numero di download
8. Cerca app più costose ordinate per download

---- Statistiche della Knowledge Base ----
9. Ottieni il numero di app Editor's Choice appartenenti ad una specifica categoria
10. Ottieni la valutazione media per una specifica categoria
11. Ordina tutte le categorie per valutazione media
12. Ordina tutte le categorie per numero totale di download
13. Cerca gli sviluppatori con più app di successo in una specifica categoria
X. Torna al menu principale

Scegli un'opzione:
```

```
Scegli un'opzione: 7
Inserisci il numero di app da visualizzare: 10
-------------------------------------------
Nome App                        | Downloads
-------------------------------------------
Google Play services            | 12057627016
YouTube                         | 9766230924
Google                          | 9154248491
Google Maps - Navigate & Explore | 9141671889
Google Text-to-Speech           | 9034404884
Google Chrome: Fast & Secure    | 8925640788
Gmail                           | 8756574289
Android Accessibility Suite     | 7408134567
Google Drive                    | 7028265259
Facebook                        | 6782619635
-------------------------------------------
```

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: 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:

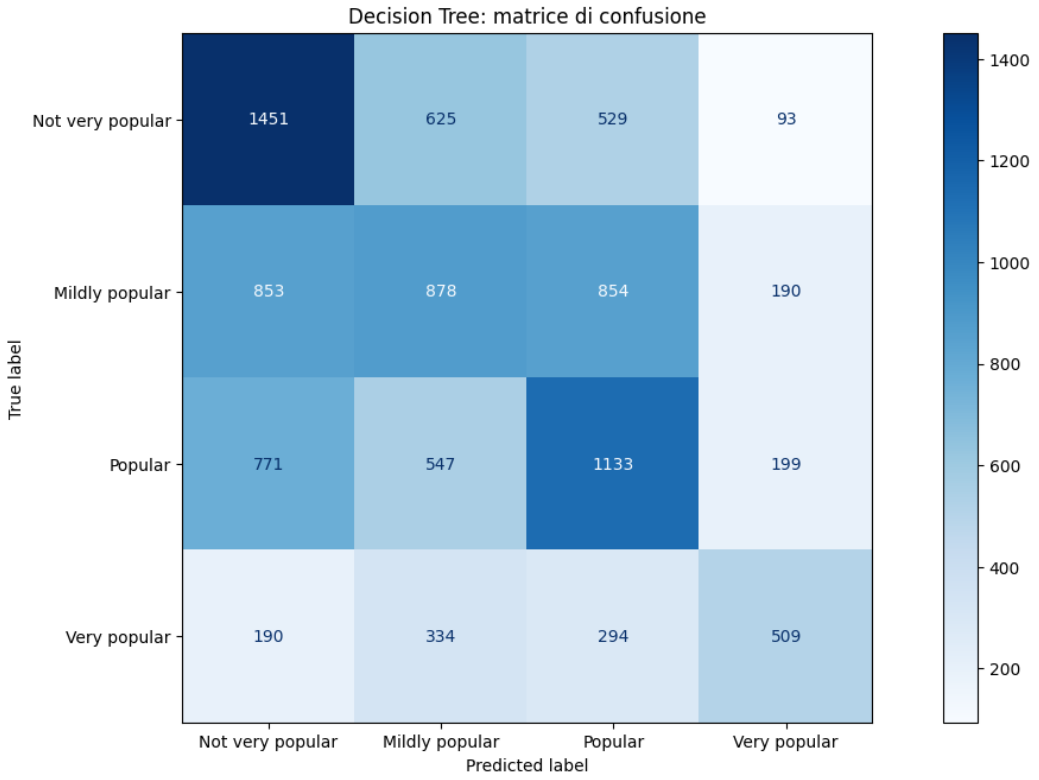
- **`criterion`:** misura la qualità delle suddivisioni (split).
  Valori testati:
  - **`gini`**: misura l'impurità di un nodo ossia la probabilità di classificare erroneamente un elemento scelto casualmente.
  - **`entropy`:** misura l'entropia nei nodi.
- **`max_depth`:** profondità massima dell'albero; 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:

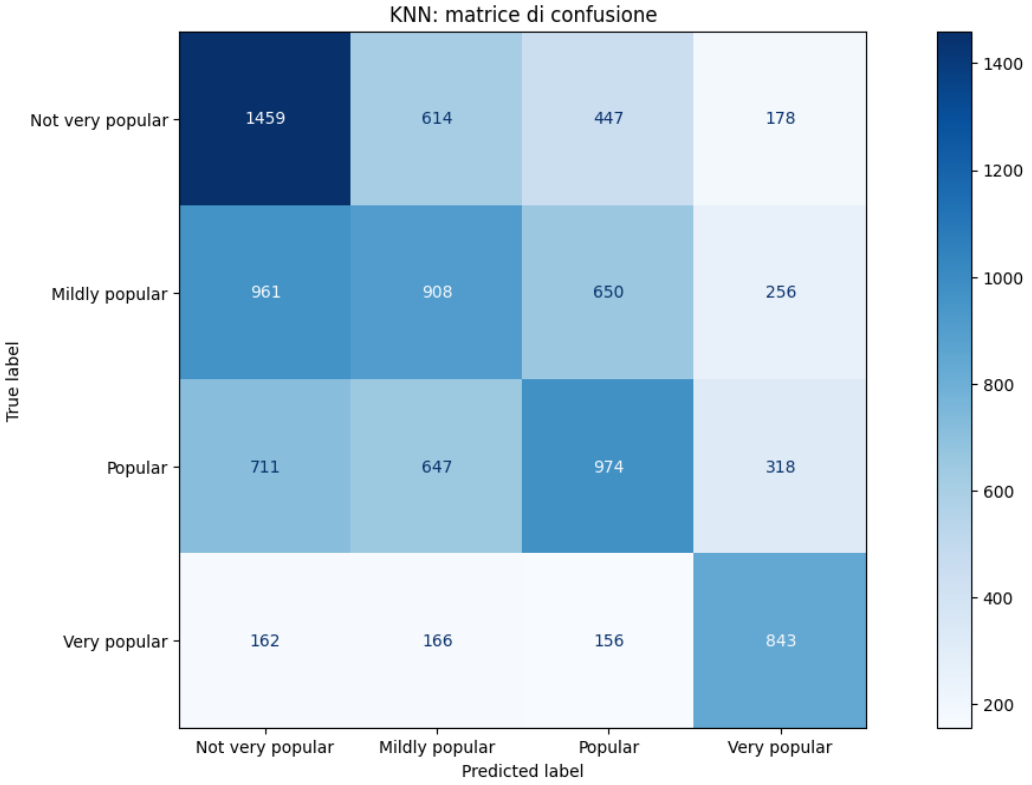
- **`metric`**: metrica di distanza utilizzata per misurare la vicinanza tra i punti
  Valori testati:
  - **`minkowski`**: generalizzazione sia della distanza euclidea che della distanza di Manhattan
  - **`manhattan`**: somma delle distanze assolute tra i punti lungo ciascuna dimensione
  - **`euclidean`**: 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:

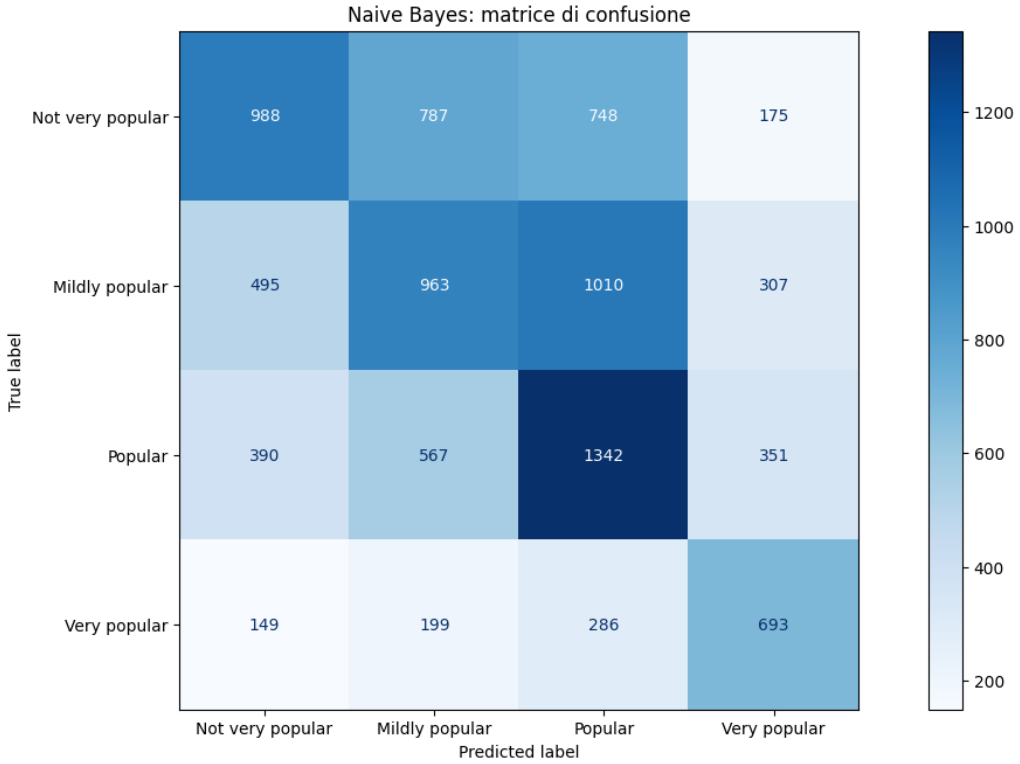
- **`var_smoothing`**: termine di stabilizzazione che aggiunge una piccola quantità di varianza a tutte le variabili per evitare divisioni per zero durante il calcolo delle probabilità
  Valori testati: **`logspace(0, -9, num=200)`** ossia un array di 200 valori distribuiti su una scala logaritmica tra 10^{-9} e 10^{0}.

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:

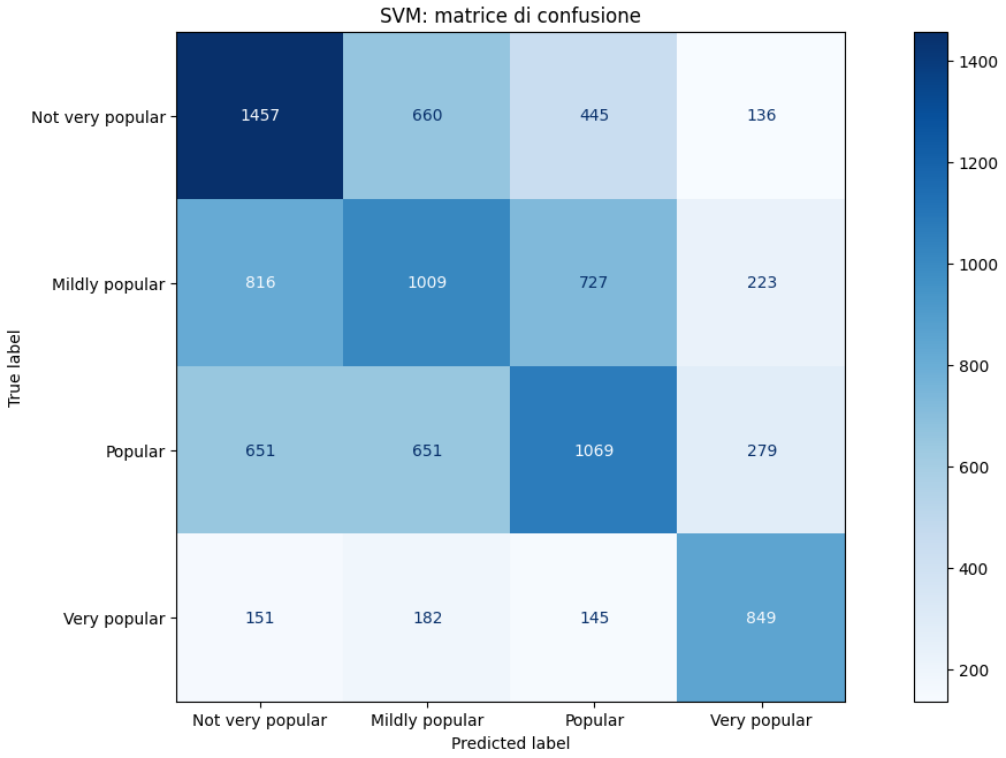
- `C`: parametro di regolarizzazione che controlla il trade-off tra classificazione corretta dei punti di addestramento e la massimizzazione del margine della funzione decisionale.
  Valori testati: 0.1, 1, 10, 50
- `gamma`: parametro per il kernel che definisce l'influenza di un singolo punto di addestramento.
  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

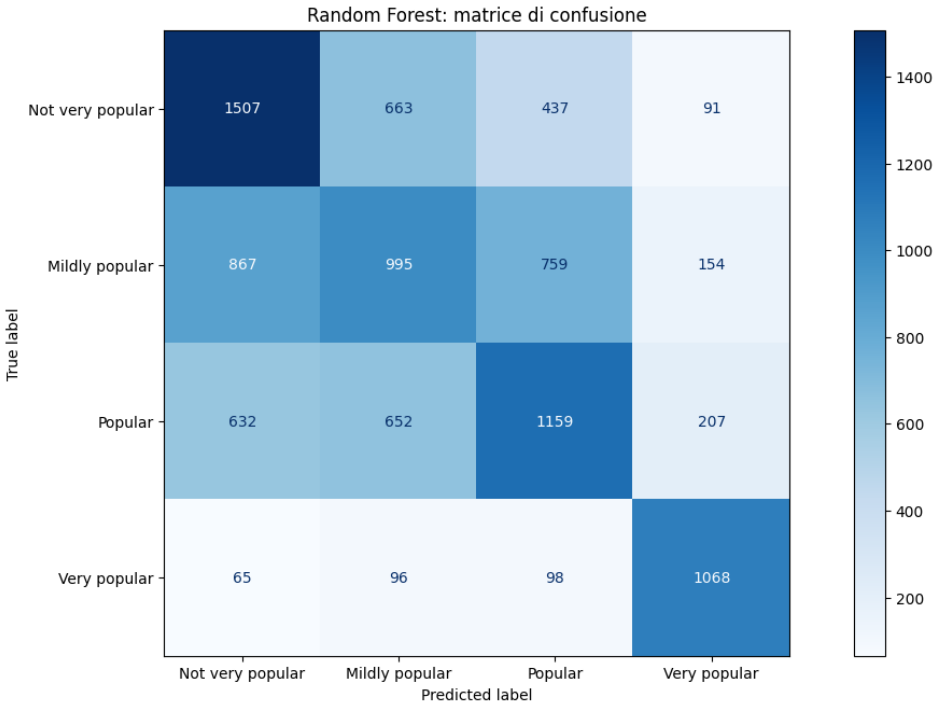
- `max_features`: percentuale o il numero massimo di features da considerare per dividere un nodo in ogni albero.
  Valori testati: 0.1, 0.2, 0.3

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:

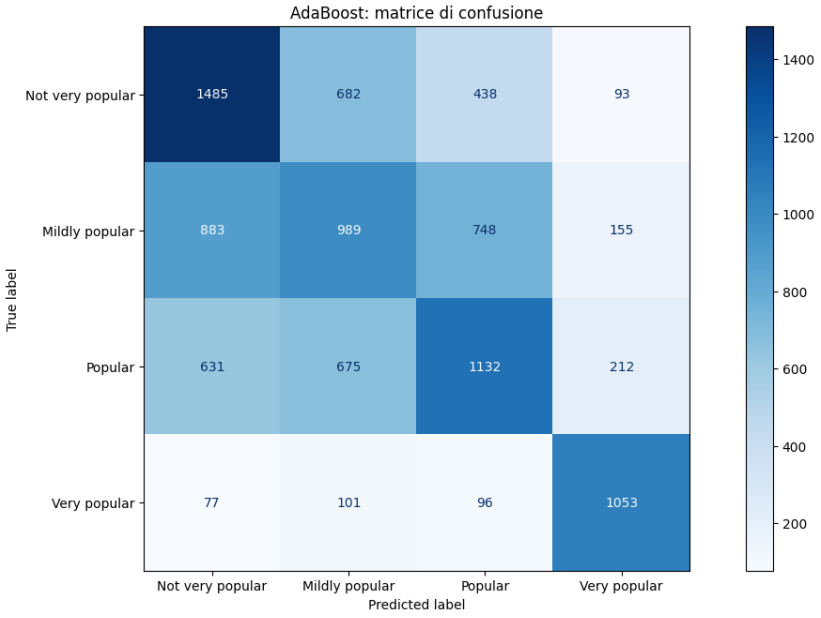
- `n_estimators:` numero di classificatori da combinare
  Valori testati: 50, 100, 200, 300
- `learning_rate:` fattore di riduzione del peso degli errori
  Valori testati: 0.01, 0.1, 0.5, 1, 1.5
- `algorithm:` algoritmo utilizzato per l'aggiornamento del peso dei classificatori deboli
  Valore testato:
  - `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`

e i risultati ottenuti sono i seguenti:

	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**		



3.3.7. Neural Network

Sono state testate le diverse combinazioni dei seguenti iperparametri:

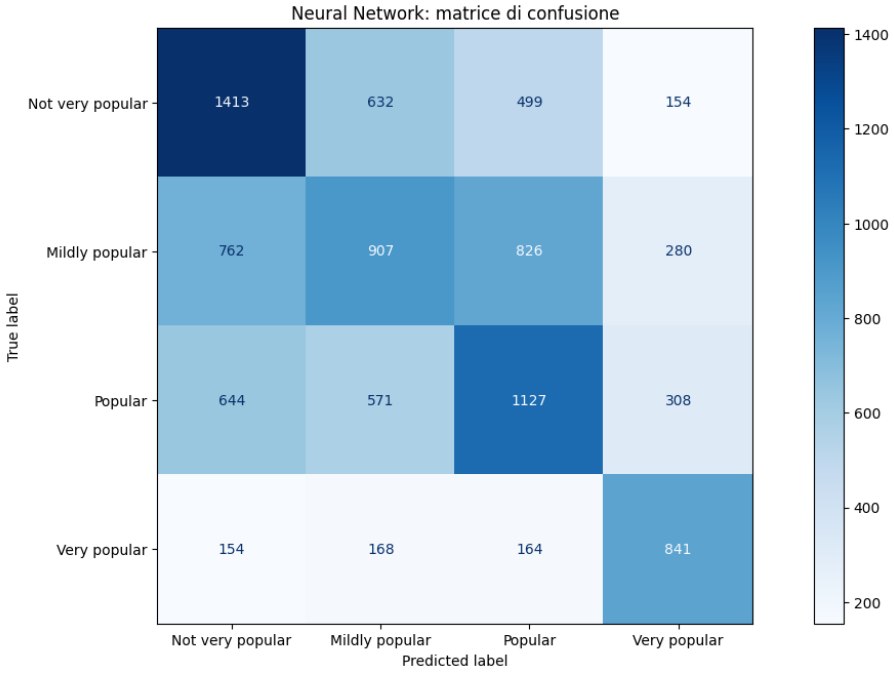
- **`hidden_layer_sizes`**: dimensione e il numero di strati nascosti nella rete neurale
  Valori testati: (10,), (50,), (120, 80, 40), (150, 100, 50)
- **`activation`**: funzione di attivazione utilizzata nei neuroni
  Valori testati:
  - **`tanh`**: funzione tangente iperbolica che mappa l'input nell'intervallo [-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`

e i risultati ottenuti sono i seguenti:

	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**		



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

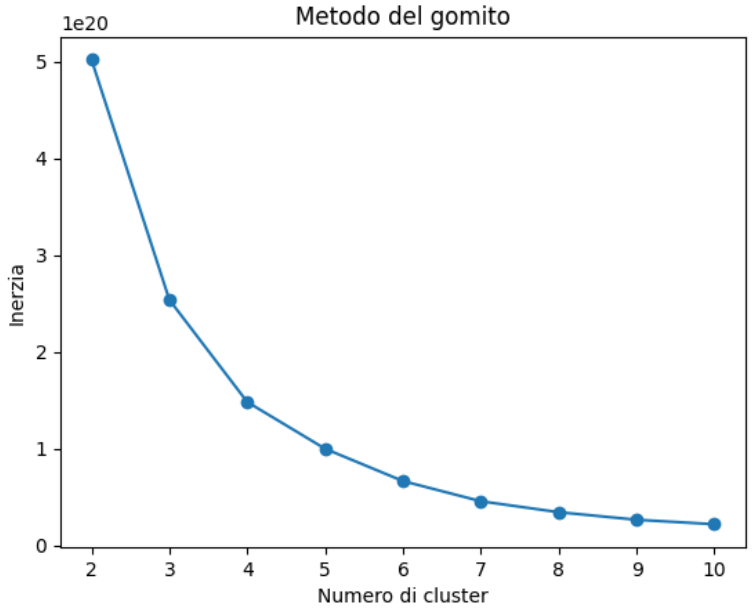
Sono state utilizzate le seguenti librerie:

- Scikit-Learn: utilizzata per l'algoritmo K-Means e per la similarità del coseno
- Natural Language Toolkit: libreria per l'elaborazione del linguaggio naturale utilizzata per la tokenizzazione
- 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:

```
Benvenuto!
Questo sistema è progettato per aiutare gli utenti e gli sviluppatori a fare scelte informate nel
mercato delle app, offrendo suggerimenti personalizzati e previsioni basate su analisi dei dati.

Scegli una delle seguenti opzioni:
1 - Raccomandazione di app simili
2 - Predizione del tasso di successo di un app non ancora sul mercato
3 - Calcolo della probabilità di successo di un app non ancora sul mercato tramite belief network
4 - Esplorazione della base di conoscenza
X - Esci
1
Le categorie disponibili sono:
-  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
Quale categoria di app stai cercando?
Communication
Cerchi un'app gratuita o a pagamento?
gratis
Suggeriscimi il nome di un'app di tuo gradimento appartenente a questa categoria:
Whatsapp
Le app si classificano sulla base dei contenuti in:
-  Everyone
-  Teen
-  Adults
Quale dovrebbe essere la classificazione dei contenuti dell'app?
Everyone
Un'app "Editor's Choice" è un'app scelta dalla redazione come una delle app più innovative, creative e degne di nota presenti nello store.
Stai cercando un'app Editor's Choice?
si
Inserisci il numero di download che l'app dovrebbe avere:
5000000000
Qual è il numero minimo di stelle (tra 1 e 5) che l'app deve possedere?
4
Ricerca delle app simili...
```

```
Ecco le applicazioni suggerite in base alle tue richieste:
-------------------------------------------------------------------------------------------------------------------
App Name                                          | Rating | Downloads  | Price ($) | Editors Choice | Success Rate
-------------------------------------------------------------------------------------------------------------------
Contacts                                          | 4.3    | 885927111  | 0.0       | False          | Very popular
Opera Mini - fast web browser                     | 4.3    | 551153058  | 0.0       | False          | Very popular
ANT Radio Service                                 | 4.0    | 1494252350 | 0.0       | False          | Very popular
Hangouts                                          | 4.0    | 5019518222 | 0.0       | False          | Very popular
Facebook                                          | 2.3    | 6782619635 | 0.0       | False          | Very popular
Messenger Lite: Free Calls & Messages             | 3.9    | 810116851  | 0.0       | True           | Very popular
TikTok                                            | 4.4    | 1645811582 | 0.0       | False          | Very popular
Gmail                                             | 4.2    | 8756574289 | 0.0       | False          | Very popular
Skype - free IM & video calls                     | 4.3    | 1769991234 | 0.0       | True           | Very popular
WhatsApp Messenger                                | 4.0    | 6265637751 | 0.0       | True           | Very popular
Samsung Push Service                              | 4.2    | 4186667750 | 0.0       | False          | Very popular
Google Duo - High Quality Video Calls             | 4.5    | 4022259636 | 0.0       | False          | Very popular
LINE: Free Calls & Messages                       | 4.1    | 861495092  | 0.0       | False          | Very popular
Carrier Services                                  | 4.3    | 1793502218 | 0.0       | False          | Very popular
Google Chrome: Fast & Secure                      | 4.1    | 8925640788 | 0.0       | False          | Very popular
Messages                                          | 4.4    | 1931517750 | 0.0       | False          | Very popular
Snapchat                                          | 4.3    | 1621265491 | 0.0       | True           | Very popular
Instagram                                         | 3.8    | 3559871277 | 0.0       | True           | Very popular
Facebook Lite                                     | 3.1    | 2072296494 | 0.0       | False          | Very popular
imo free video calls and chat                     | 4.1    | 926726001  | 0.0       | False          | Very popular
-------------------------------------------------------------------------------------------------------------------

Vuoi vedere altri risultati? (si/no):
```

5. Belief Network

5.1. Sommario

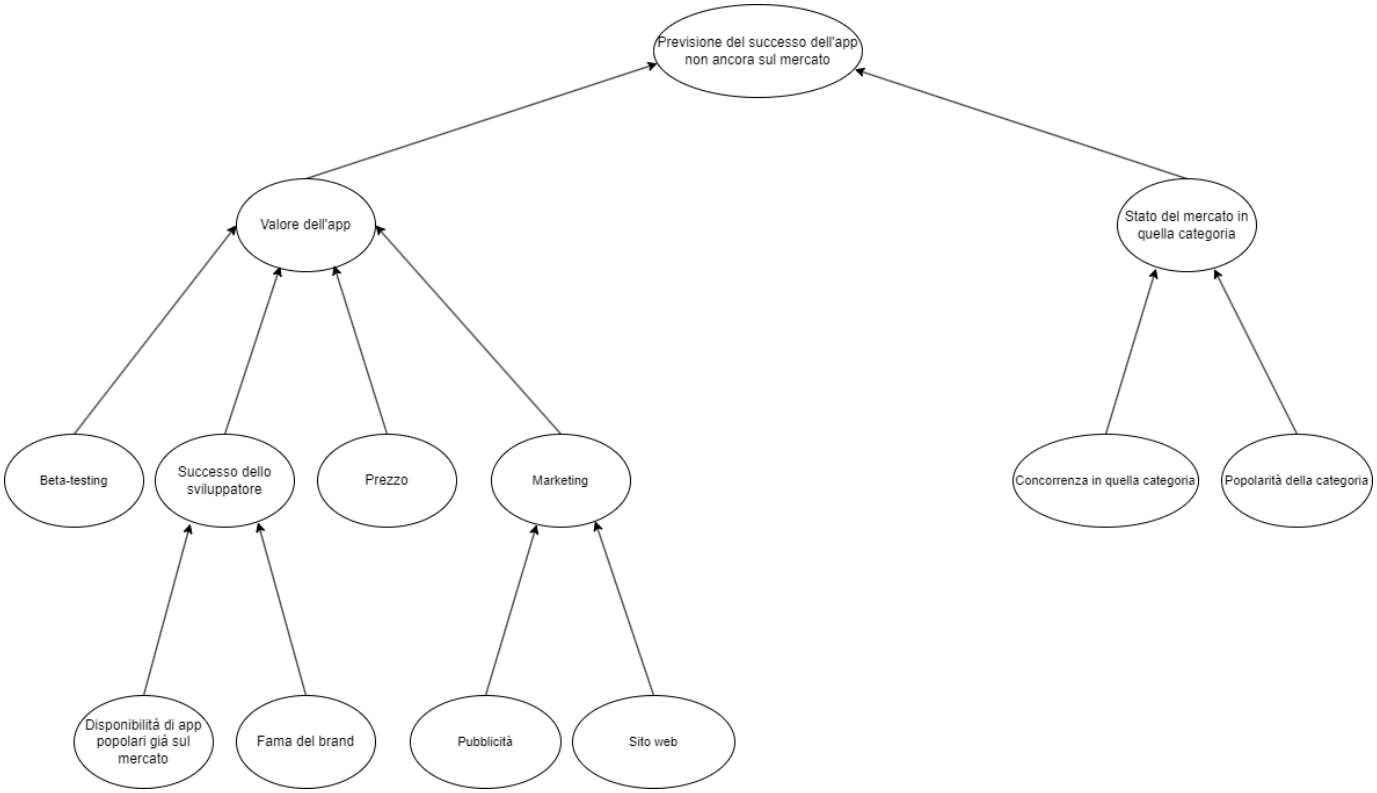
Il ragionamento probabilistico è una forma di ragionamento in presenza di incertezza che sfrutta la teoria della probabilità. Un'applicazione del ragionamento probabilistico è la belief network, un modello che rappresenta la dipendenza condizionata tra un insieme di variabili casuali. Nel nostro progetto, 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 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

```
Benvenuto!
Questo sistema è progettato per aiutare gli utenti e gli sviluppatori a fare scelte informate nel
mercato delle app, offrendo suggerimenti personalizzati e previsioni basate su analisi dei dati.

Scegli una delle seguenti opzioni:
1 - Raccomandazione di app simili
2 - Predizione del tasso di successo di un app non ancora sul mercato
3 - Calcolo della probabilità di successo di un app non ancora sul mercato tramite belief network
4 - Esplorazione della base di conoscenza
X - Esci
3
Per stimare la probabilità di successo della tua app, rispondi alle seguenti domande:
Hai già app di successo sul mercato?
Risposte possibili: si / no
no
Sei affiliato ad un brand famoso?
Risposte possibili: si / no
no
La tua app ha previsto una fase di beta testing?
Risposte possibili: si / no
si
La tua app è gratis o a pagamento?
Risposte possibili: gratis / a pagamento
gratis
```

```
La tua app ha un sito web associato?
Risposte possibili: si / no
si
Hai investito in pubblicità per la tua app?
Risposte possibili: si / no
si
La concorrenza nella categoria associata alla tua app è alta o bassa?
Risposte possibili: alta / bassa
bassa
Quanto è popolare la categoria associata alla tua app?
Risposte possibili: molto / poco
molto
La probabilità di successo dell'app è pari a 80.0%
Probabilità di successo alta.
Ottimo lavoro! Le probabilità di successo della tua app sono eccellenti.
Mantieni questo ritmo e continua a innovare; il successo è dietro l'angolo e il tuo impegno sarà sicuramente ricompensato.
```

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)
- Analyzing Key Factors and Predicting App Success on the Google Play Store | by Sahithi Arnika | Medium
- A Recommender System for Mobile Applications of Google Play Store
- Scikit-learn API Reference