Report Movie Recommender

15 Luglio 2023

Github Link: Movie-Recommender

Contents

1 Autori

Nome	Matricola
Federico Pomposelli	0512107799
Marco Gallo	0512109494
Nicholas De Marco	0512106465

2 Introduzione

Negli ultimi anni le piattaforme di streaming online per la visione di contenuti cinematografici sono sempre più ampiamente utilizzate, e conseguentemente vengono sviluppati diversi sistemi che mirano a consigliare ad un determinato utente un contenuto che possa piacergli.

Abbiamo quindi puntato alla creazione di un sistema di raccomandazione intelligente che possa suggerire dei contenuti cinematografici ad uno specifico utente sulla base delle sue preferenze, fornite da diversi dati raccolti precedentemente.

3 Descrizione dell'agente

3.1 Obiettivi

Lo scopo del progetto è quello di realizzare un agente intelligente che sia in grado di predirre il rating di film per utenti, in modo da suggerire un contenuto apprezzabile.

3.2 Specifica PEAS

PEAS	
Performance	La misura di performance dell'agente è la sua capacità di predire un rating ideale dei film per utenti differenti.
Ambiente	L'ambiente in cui opera l'agente è una piattaforma potenziale con accesso a dati sui film e sugli utenti. L'ambiente presenta le seguenti caratteristiche: - Completamente osservabile: l'agente ha visibilità su tutti i dati in ogni momento. - Stocastico: le informazioni sui film e sugli utenti possono essere soggette a cambiamenti e aggiornamenti. - Episodico: ogni predizione è un evento atomico che dipende dalle informazioni disponibili in quel momento. - Statico: l'ambiente rimane invariato al momento della predizione. - Discreto: viene fornito un numero limitato di percezioni. - Singolo: opera un solo agente.
Attuatori	Monitor sul quale vengono stampate le informazioni riguardo le predizioni dei film.
Sensori	Campi delle tabelle dati sul quale l'agente potrà poi operare.

3.3 Analisi del problema

Abbiamo inizialmente optato per risolvere il problema sulla selezione e valutazione dei film con un algoritmo di clustering, utilizzando un dataset di soli film. Questa opzione ci ha portato a fuorviare da una soluzione ammissibile per diversi motivi:

- Determinare il numero di cluster è risultato ostico, in quanto non riuscivamo ad offrire una completa personalizzazione dei film consigliati agli utenti

- La qualità dei cluster, in quanto a eterogeneità dei diversi gruppi sembrava pessima

- Risultava altrettanto complessa la valutazione del modello.

- Si era ridotto il tutto all'utilizzo di una Jaccard Similarity che offriva risultati statici a secondo dei film dati in input.

Abbiamo pertanto cercato un dataset che potesse darci info riguardo una serie di utenti, e valutato di cambiare tecnica spostandoci su algoritmi di classificazione che si sposassero meglio con i dataset (film e utenti) che abbiamo deciso di utilizzare. I dataset si abbinano molto bene in quanto è possibile convergere e creare nuove tabelle tra le votazioni di un utente e le rispettive pellicole votate.

L'idea è stata quindi di combinare i due dataset e lavorare su di una predizione dei valori di rating dei film votati dagli utenti.

4 Raccolta, analisi e preprocessing dei dati

4.1 Scelta del dataset

Venendo al dataset necessario per la creazione del modello di machine learning, le possibili strade da seguire erano due:

1. Creare un dataset da zero
2. Cercare dei dataset sulla rete

La prima opzione era molto dispendiosa in termini di tempo e possibile inconsistenza dei dati, dovendone generare di nuovi e/o casuali

Si è deciso pertanto di ricercare un dataset su kaggle, e abbiamo optato inizialmente per imdb, un dataset di soli 1000 film e successivamente per movielens, un dataset considerevolmente più grande, contenente anche utenti e le votazioni ai rispettivi film

4.2 Analisi del dataset

Il dataset fornito da kaggle ci ha offerto una serie di dati raccolti dall'università del Minnesota per un progetto al quale hanno lavorato.

Il dataset consiste principalmente in:

- 100.000 votazioni (0-5) da 943 utenti su 1682 film, ogni utente ha votato almeno 20 film.
- Info dei film (id, titolo, data di uscite, url imdb, generi)
- Info demografiche degli utenti (id, età, sesso, occupazione, cod. postale)

Il dataset è suddiviso in diversi file rappresentativi di diverse informazioni:

Tabella relazionale che mette in relazione l'utente con i film da esso votati:

u.data -- The full u data set, 100000 ratings by 943 users on 1682 items.
Each user has rated at least 20 movies. Users and items are
numbered consecutively from 1. The data is randomly
ordered. This is a tab separated list of
user id | item id | rating | timestamp.
The time stamps are unix seconds since 1/1/1970 UTC

Figure 1:

Tabella contenente info riguardo i film: genere, titolo, id e data di rilascio:

u.item -- Information about the items (movies); this is a tab separatec
list of
movie id | movie title | release date | video release date |
IMDb URL | unknown | Action | Adventure | Animation |
Children's | Comedy | Crime | Documentary | Drama | Fantasy |
Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi |
Thriller | War | Western |
The last 19 fields are the genres, a 1 indicates the movie
is of that genre, a 0 indicates it is not; movies can be in
several genres at once.
The movie ids are the ones used in the u.data data set.

Figure 2:

Tabella contenente info demografiche su ciascun utente:

u.user -- Demographic information about the users; this is a tab separated list of
user id | age | gender | occupation | zip code
The user ids are the ones used in the u.data data set.

Figure 3:

4.3 Preprocessing dei dati

La struttura della tabella "u.item" (rinominata poi in "movies2") era formattata in malo modo, abbiamo dunque creato un DataFrame movies con colonne specifiche, successivamente popolato con i valori delle righe del file ed eliminando caratteri superflui.

```
with open('./dataset/movies2', encoding = "ISO-8859-1") as content:
    colonne = ['movie_id', 'movie_title', 'release_date', 'video_release_date', 'imdb_url']
    generi = ['unknown', 'action', 'adventure', 'animation', 'children', 'comedy', 'crime', 'documentary', 'drama', 'fantasy',
          'film-noir',  'horror', 'musical', 'mystery', 'romance', 'sci-fi', 'thriller', 'war', 'western']
    colonneG = colonne + generi
    movies = pd.DataFrame(columns=colonneG)
    i = 0
    for x in content:
        x = x.split("|")
        x[-1] = x[-1][:-1]
        if x[1][-1] == ' ':
            x[1] = x[1][:-1]
        movies.loc[i] = [word if word!='' else "empty" for word in x]
        i = i + 1
movies['movie_id'] = movies['movie_id'].astype('int64')
movies[generi] = movies[generi].astype('category')
```

Figure 4:

La struttura della tabella "user" (rinominata poi in "users") era tale da avere una colonna contenente l'impiego dell'utente con valori in formato string. Abbiamo di conseguenza trasformato tale colonna in più colonne, una per ogni impiego esistente nel dataset, utilizzando una codifica binaria per rappresentare l'appartenenza di un utente al suo determinato impiego:

```
users['occupation'] = users['occupation'].astype('category')
users['gender'] = users['gender'].astype('category')
one_hot_occ = users.occupation.str.get_dummies()
users = users.drop('occupation',axis=1)
users = users.join(one_hot_occ,how='inner')
one_hot_g = users.gender.str.get_dummies()
users = users.drop('gender',axis=1)
users = users.join(one_hot_g,how='inner')
users.head(5)
```

Figure 5:

Abbiamo inizialmente creato una colonna contenente la variabile target "like" riempendone i valori in maniera casuale, testando l'algoritmo su questi dati, per poi repentinamente eliminarla ed utilizzare una nuova variabile target ottenuta dal join dei dataset contenente gli utenti con quello dei film, ottenendo la seguente tabella:

	user_id	movie_id	movie_title	unknown	action	adventure	animation	children	comedy	crime	...	programmer	retired	salesman	scientist	student	technician	writer	F	M	rating
0	196	242	Kolya (1996)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
1	196	257	Men in Black (1997)	0	1	1	0	0	1	0	...	0	0	0	0	0	0	1	0	1	0
2	196	111	Truth About Cats & Dogs, The (1996)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
3	196	25	Birdcage, The (1996)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
4	196	382	Adventures of Priscilla, Queen of the Desert, ...	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
5	196	202	Groundhog Day (1993)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
6	196	153	Fish Called Wanda, A (1988)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
7	196	286	English Patient, The (1996)	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	1	1
8	196	66	While You Were Sleeping (1995)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1
9	196	845	That Thing You Do! (1996)	0	0	0	0	0	1	0	...	0	0	0	0	0	0	1	0	1	1

10 rows × 47 columns

Figure 6:

4.4 Data-preparation

La data-preparation è avvenuta in più fasi, adattando i dati all'algoritmo, a seconda dei risultati ottenuti da quest'ultimo.

4.4.1 Data cleaning

Il data cleaning, o pulizia dei dati, è un processo che consiste nell'individuare, correggere o rimuovere errori, rumori o incongruenze presenti in un set di dati al fine di migliorare la qualità e l'affidabilità delle informazioni contenute.
Durante il data cleaning, vengono eseguite diverse operazioni, tra cui:

- Rimozione dati mancanti
- Rimozione dei dati duplicati
- Correzione degli errori di formattazione
- Gestione dei valori anomali
- Normalizzazione dei dati

Abbiamo effettuato la pulizia dei dati, rimuovendo le colonne inutili ai fini del nostro algoritmo:

-Data di rilascio
-Data di rilascio del trailer
-Url imdb
-Zip code (dal dataset utente)

```
#@title Data cleaning: Tabelle con feature inutili
movies = movies.drop('video_release_date', axis=1)
movies = movies.drop('release_date', axis=1)
movies = movies.drop('imdb_url', axis=1)
```

Figure 7:

4.4.2 Feature scaling

Il feature scaling, o ridimensionamento delle caratteristiche, è una tecnica utilizzata per portare le diverse caratteristiche (variabili) di un set di dati su una scala comune. L'obiettivo principale del feature scaling è quello di assicurarsi che le caratteristiche abbiano lo stesso ordine di grandezza, eliminando eventuali disparità di scala o unità di misura tra di loro.

Abbiamo scalato i valori della colonna rating cambiando il range di valori da 0 a 5 in 0 a 1.

```
final_dataset['rating'] = final_dataset['rating'].replace(0,0)
final_dataset['rating'] = final_dataset['rating'].replace(1,0)
final_dataset['rating'] = final_dataset['rating'].replace(2,0)
final_dataset['rating'] = final_dataset['rating'].replace(3,1)
final_dataset['rating'] = final_dataset['rating'].replace(4,1)
final_dataset['rating'] = final_dataset['rating'].replace(5,1)
final_dataset.head(10)
```

Figure 8:

4.4.3 Feature selection

La selezione delle caratteristiche, o selezione delle caratteristiche, è il processo di identificazione delle variabili indipendenti più rilevanti o informativo per la previsione della variabile dipendente in un modello di apprendimento automatico. L'obiettivo è ridurre la dimensionalità dei dati, mantenendo solo le caratteristiche più significative, al fine di migliorare l'efficienza, l'interpretabilità e le prestazioni del modello.

Nel nostro caso, abbiamo creato una tabella che mette in relazione la variabile dipendente "rating" con le variabili indipendenti come le categorie dei film e le caratteristiche degli utenti, come impiego, sesso ed età attraverso una matrice di correlazione. Abbiamo dunque preso in considerazione le migliori 24 feature, eliminando le altre colonne. Eliminando successivamente anche le istanze che non presentavano più o alcuna occupazione o alcun genere di film.

drama	0.083915
healthcare	0.080840
age	0.057271
comedy	0.052966
war	0.046359
horror	0.045483
executive	0.039378
writer	0.037685
educator	0.034889
film-noir	0.032470
romance	0.029448
children	0.028512
administrator	0.022354
fantasy	0.021812
action	0.019794
scientist	0.017199
mystery	0.016042
crime	0.015191
western	0.014460
engineer	0.014318
librarian	0.013515
none	0.013513
entertainment	0.012831
doctor	0.012356

Figure 9:

4.4.4 Data balancing

Il data balancing, o bilanciamento dei dati, è una tecnica utilizzata per affrontare il problema delle classi sbilanciate in un set di dati di addestramento. Si verifica quando una o più classi sono rappresentate da un numero si discosta di istanze inferiori rispetto alle altre classi.

In presenza di classi sbilanciate, un modello di apprendimento automatico potrebbe avere difficoltà a catturare correttamente i pattern oa fornire previsioni accurate per le classi di minoranza.

Dopo aver notato che i dati relativi alla variabile target fossero sbilanciati, abbiamo deciso di bilanciare il dataset attraverso la tecnica di undersampling.

In particolare erano molto piu presenti nel dataset voci di film con votazione 1 rispetto a 0, il bilanciamento è stato effettuato quindi eliminando 6000 righe con valore rating pari ad 1.

(Abbiamo optato successivamente anche per l'oversampling, una volta compreso che il problema principale era dovuto da un bias eccessivamente alto).

```
# Determina il numero di righe da eliminare casualmente
num_rows_to_drop = 60000  # Specifica il numero di righe da eliminare

# Seleziona solo le righe con rating = 1 e determina l'indice di queste righe
rows_to_drop = final_dataset.loc[final_dataset['rating'] == 1].index

# Genera un array casuale di indici di riga da eliminare
random_rows = np.random.choice(rows_to_drop, size=num_rows_to_drop, replace=False)

# Elimina le righe casuali dal DataFrame
final_dataset.drop(random_rows, inplace=True)
```

Figure 10:

5 Algoritmo di clustering

Il clustering è una tecnica di apprendimento non supervisionato che mira a suddividere un insieme di dati in gruppi omogenei, chiamati cluster. L'obiettivo è creare raggruppamenti basati sulla somiglianza tra gli elementi, in modo che gli oggetti all'interno di uno stesso cluster siano più simili tra loro rispetto a quelli in cluster diversi.

Il clustering si basa sulla struttura intrinseca dei dati, cercando di identificare pattern o strutture nascoste. Non richiede l'utilizzo di etichette di classe preesistenti, a differenza della classificazione supervisionata.

L'idea iniziale di un algoritmo di clustering consisteva nel creare dei cluster costituiti da una lista di film consigliabili per diversi tipi di utenti. Dati i problemi già citati in 3.3, l'algoritmo risultante si limitava a consigliare una lista di film attraverso una jaccard similariy effettuata su vari film dati in input. In particolare, utilizzando il dataset imdb, abbiamo creato una lista di parole ottenute dai film dati in input; tale lista comprendeva il nome del regista, dei due attori principali, del genere del film e della sinossi del film (opportunamente filtrata tramite la rimozione di stopwords). Confrontando la lista ottenuta dai film dati input con tutti gli altri film del dataset, ottenevamo un output di film soddisfacente, ma l'algoritmo risultante non poteva certamente apprendere da ciò.

```
#prova jaccard similarity

def jaccard_similarity(list1, list2):
    s1 = set(list1)
    s2 = set(list2)
    return float(len(s1.intersection(s2)) / len(s1.union(s2)))

list1 = imdb.loc[0, 'target'].split()

list2 = imdb.loc[1, 'target'].split()
print(list1)
print(list2)
jaccard_similarity(list1, list2)
```

Figure 11:

Abbiamo di conseguenza scartato l'idea, concentrandoci su un algoritmo di classificazione.

6 Algoritmo di classificazione

6.1 Decision Tree

Il decision tree è un algoritmo di classificazione che utilizza una struttura ad albero per prendere decisioni o effettuare previsioni. Organizza i dati in un albero composto da nodi decisionali e nodi foglia, ed il suo scopo è quello di assegnare un'istanza ad una classe specifica.

Nell'albero di decisione, i nodi decisionali rappresentano i criteri di suddivisione dei dati in base alle caratteristiche o attributi dei dati stessi. Ogni nodo decisionale corrisponde a una domanda o ad un test sul valore di una caratteristica specifica. Le risposte alle domande portano a un percorso lungo l'albero fino ai nodi foglia, che rappresentano le classi di appartenenza o le previsioni finali.

Abbiamo dunque utilizzato il decision tree per assegnare ad ogni film il suo valore di rating, e dopo ogni test di valutazione dell'algoritmo, abbiamo apportato delle modifiche ai dati che potessero permetterci di avere risultati migliori.

Nel nostro caso i nodi decisionali rappresentano l'appartenenza del film in questione ad un genere cinematografico o meno, l'età dell'utente e l'occupazione di quest'ultimo. I nodi foglia rappresentano invece il valore di rating predetto del film.

6.1.1 Risultati decision tree

Abbiamo effettuato vari test, in modo da capire come potessimo migliorare il modello. Abbiamo iniziato eseguendo un test con un database di soli film (Figure 2) associati ad un singolo utente. Abbiamo effettuato datacleaning rimuovendo le colonne inutili(Figure 7) e abbiamo inserito manualmente una colonna target ("like"), riempendone i valori (da 0 a 5) in maniera casuale.

Il **primo test** ha prodotto i seguenti risultati:

```
Cross validation:
 [0.38518519 0.31851852 0.38518519 0.37777778 0.36296296 0.37313433
 0.41044776 0.41044776 0.38059701 0.37313433]
Testing score: 0.40059347181008903
```

Figure 12:

```
Average expected loss: 0.622
Average bias: 202.098
Average variance: 0.165
Sklearn 0-1 loss: 0.599
/n
Average expected loss--After pruning: 0.567
Average bias--After pruning: 187.000
Average variance--After pruning: 0.039
```

Figure 13:

Utilizzando come misura di prestazione il cross-validation, abbiamo osservato degli score piuttosto bassi. Prodotti anche risultati di bias alti, abbiamo utilizzato la tecnica di pruning sull'albero decisionale, riducendone la complessità e migliorando leggermente la condizione di eccessivo bias.

Abbiamo dunque effettuato la prima modifica ai dati: il join della tabella contenente i dati degli utenti (Figure 3) con la tabella contenente i dati dei film che stavamo utilizzando, in modo da avere un dataset di dimensioni decisamente maggiori e dati più consistenti. Dopo il join abbiamo eliminato la colonna "like", e abbiamo utilizzato la nuova colonna "rating" (valori da 0 a 5) ottenuta dal join stesso, evitando di avere così valori casuali della variabile target e ottenendo dati consistenti.

Il **secondo test** ha prodotto i seguenti risultati:

```
cross validation scores:
 [0.331125 0.337125 0.323125 0.33825  0.33275  0.345    0.342375 0.332
 0.32925  0.326   ]
testing score: 0.3373
```

Figure 14:

```
Average expected loss: 0.739
Average bias: 14664.338
Average variance: 0.335
Sklearn 0-1 loss: 0.662
/n
Average expected loss--After pruning: 0.662
Average bias--After pruning: 13243.892
Average variance--After pruning: 0.007
```

Figure 15:

I risultati erano ancora scarsi, se non peggiori. Abbiamo di conseguenza preparato il dataset per il prossimo test. In particolare questa è la fase in cui abbiamo effettuato feature scaling sulla variabile target(Figure 8), questo affinchè potesse risultare più semplice all'algoritmo di decision tree classificare l'istanza in sole due etichette.

Il **terzo test** ha prodotto i seguenti risultati:

```
Cross validation:
 [0.781125 0.780625 0.786375 0.783375 0.7855   0.78025  0.784    0.78625
 0.783    0.782875]
Testing score: 0.7857


Accuracy modello: 0.7857
Confusion matrix

 [[  796  2608]
 [ 1678 14918]]
```

Figure 16:

```
Average expected loss: 0.258
Average bias: 4678.347
Average variance: 0.091
Sklearn 0-1 loss: 0.214
/n
Average expected loss--After pruning: 0.175
Average bias--After pruning: 3507.557
Average variance--After pruning: 0.000
Sklearn 0-1 loss--After pruning: 0.167
```

Figure 17:

Abbiamo utilizzato un'ulteriore misura di valutazione del modello: la matrice di confusione.

Possiamo osservare come il punteggio del modello sia salito ad un valore accettabile, 78, sia per la cross-validation che per l'accuracy. Il bias rimane comunque molto alto, nonostante il pruning ne migliori leggermente il valore.

Per migliorare ulteriormente l'algoritmo abbiamo effettuato una feature selection (Figure 9). Il nostro scopo principale restava quello di abbassare il valore di bias, rendendo con la feature selection l'albero meno "intricato" e più semplice da scorrere per l'algoritmo.

Il **quarto test** ha prodotto i seguenti risultati:

```
Cross validation:
 [0.80717256 0.81081081 0.81094075 0.81068087 0.81026641 0.80883691
 0.81052632 0.8100065  0.80818713 0.81221572]
Testing score: 0.8067467124070897


Accuracy modello: 0.8067
Confusion matrix

 [[  482  2915]
 [  803 15039]]
```

Figure 18:

```
Average expected loss: 0.225
Average bias: 4123.427
Average variance: 0.047
Sklearn 0-1 loss: 0.193
/n
Average expected loss--After pruning: 0.182
Average bias--After pruning: 3492.089
Average variance--After pruning: 0.000
Sklearn 0-1 loss--After pruning: 0.172
```

Figure 19:

I risultati restano pressochè gli stessi, migliorando leggermente.

Notando che le occorrenze dei valori di rating fossero molto sbilanciate, abbiamo pensato di effettuare un data balancing (Figure 10).

Per ovviare allo sbilanciamento abbiamo utilizzato una tecnica di oversampling. Questo per fare in modo di far operare il modello su un dataset più grande, per via del bias fin troppo elevato.

Il **quinto test** ha prodotto i seguenti risultati:

```
Cross validation:
 [0.63665645 0.64506246 0.69534135 0.69227748 0.69264613 0.69099623
 0.68942489 0.692489   0.69217473 0.69453174]
Testing score: 0.5899475024689433


Accuracy modello: 0.5899
Confusion matrix

 [[2034 1363]
 [6526 9316]]
```

Figure 20:

```
Average expected loss: 0.462
Average bias: 8861.700
Average variance: 0.091
Sklearn 0-1 loss: 0.410
Average expected loss--After pruning: 0.521
Average bias--After pruning: 10029.288
Average variance--After pruning: 0.003
Sklearn 0-1 loss--After pruning: 0.480
```

Figure 21:

Si può notare come sia il punteggio che il bias siano sensibilmente peggiorati, addirittura peggiorano con la tecnica del pruning.

Abbiamo così cambiato tecnica di data balancing, effettuando così, un undersampling.

L'**ultimo test** ha prodotto i seguenti risultati:

```
Cross validation:
 [0.61118785 0.58321823 0.59461326 0.58563536 0.59620035 0.60103627
 0.58307427 0.58860104 0.59620035 0.58583765]
Testing score: 0.5857162591518166


Accuracy modello: 0.5857
Confusion matrix

 [[1940 1394]
 [1605 2300]]
```

Figure 22:

```
Average expected loss: 0.498
Average bias: 3603.592
Average variance: 0.187
Sklearn 0-1 loss: 0.402
Average expected loss--After pruning: 0.507
Average bias--After pruning: 3685.442
Average variance--After pruning: 0.036
Sklearn 0-1 loss--After pruning: 0.443
```

Figure 23:

I risultati sono ancora peggiori. Si è deciso di utilizzare come versione finale l'algoritmo con oversampling, questo perchè il database, prima del lavoro di data balacing, era sbilanciato e i risultati migliori non erano sintomo di modello migliore, ma erano soltanto dovuti ad un'occorrenza eccessiva di valori di rating pari ad 1.

6.2 Naive-Bayes

Abbiamo testato anche un altro algoritmo di classificazione, il Naive-Bayes. L'algoritmo Naive-Bayes è un metodo di classificazione basato sul teorema di Bayes, che sfrutta l'assunzione di indipendenza condizionata tra le caratteristiche dei dati. Questo algoritmo è noto per la sua semplicità e velocità di esecuzione, ed è particolarmente efficace quando si lavora con un numero elevato di caratteristiche.

Nell'algoritmo Naive-Bayes, si calcolano le probabilità condizionali per ogni classe di appartenenza, dati e valori delle caratteristiche. Successivamente, si utilizza il teorema di Bayes per calcolare la probabilità a posteriori delle classi date le caratteristiche osservate, e si assegna al campione la classe con la probabilità più alta.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Figure 24:

6.2.1 Risultati Naive Bayes

I risultati ottenuti dall'algoritmo Naive-Bayes possono essere simili a quelli ottenuti con un albero di decisione, in quanto entrambi gli approcci possono essere utilizzati per la classificazione. Tuttavia, ci sono alcune differenze importanti tra i due. Mentre Naive-Bayes utilizza il calcolo delle probabilità condizionali e il teorema di Bayes, gli alberi di decisione prendono decisioni basate su una serie di regole di test condizionali.

Inoltre, i modi di valutazione utilizzati per valutare la performance dei modelli sono gli stessi sia per Naive Bayes che per gli alberi di decisione.

Il primo test con Naive-Bayes è stato effettuato dopo la feature selection (Figure 9).

Risultati **primo test**:

```
Cross validation:
 [0.71387734 0.72570166 0.71738565 0.72284304 0.70929175 0.71163093
 0.71578947 0.71994802 0.71786875 0.72501624]
Testing score: 0.7121991787514944


Accuracy: 0.7121991787514944
Confusion matrix

 [[ 1057  2340]
 [ 3197 12645]]
```

Figure 25:

I risultati prodotti, a parità di dataset, risultano leggermente peggiori rispetto al decision tree. Il punteggio si aggira intorno al 70

Abbiamo così deciso di scartare l'idea di utilizzare Naive-Bayes

7 Conclusioni

L'algoritmo offre risultanti poco soddisfacenti, determinati da un bias eccessivamente alto. Nonostante l'utilizzo di tecniche quali feature selection, oversampling e pruning, il risultato è scarsamente migliorato.

8 Glossario

- Dataset: insieme strutturato di dati.
- Dataframe: struttura dati tabellare, utile alla rappresentazione e manipolazione di dati tramite linguaggio Python
- Figure: immagine.
- Rating: valutazione o punteggio.
- Jaccard similarity: misura utilizzata per calcolare la distanza tra due insiemi.
- Pruning: processo che consiste nella rimozione di parti non significative o ridondanti da un albero di decisione al fine di semplificarlo e migliorare le prestazioni.
- Cross Validation: Tecnica utilizzata per valutare le prestazioni di un modello di apprendimento automatico utilizzando un set di dati limitato. Consiste nel dividere il set di dati in diverse parti, chiamate "fold", e utilizzarle in modo alternativo come set di addestramento e di test.
- Matrice di confusione: rappresentazione tabellare delle prestazioni di un modello di classificazione. Si utilizza per valutare quantitativamente l'accuratezza del modello nella previsione delle classi di appartenenza.
- Accuracy: è una misura comune utilizzata per valutare le prestazioni di un modello di classificazione. Rappresenta la percentuale di casi correttamente classificati rispetto al numero totale di casi valutati.
- Bias/Underfitting: si verifica quando un modello non riesce a catturare correttamente la complessità dei dati di addestramento. Ciò si traduce in prestazioni scadenti sia sui dati di addestramento che su quelli di test.
- Variance/Overfitting: si verifica quando un modello si adatta troppo bene ai dati di addestramento, includendo anche il rumore o l'eccessiva variazione dei dati. Questo porta a prestazioni eccellenti sui dati di addestramento, ma prestazioni scadenti su nuovi dati di test.

- Specifica P.E.A.S.: Acronimo di Performance Environment Actuators Sensors, aspetti da prendere in considerazione nello studio dell'intelligenza Artificiale.
- 0-1 loss: è una misura di errore utilizzata per valutare la precisione di un modello di classificazione. Questa misura assegna un valore di 0 quando la previsione del modello coincide con la classe reale e un valore di 1 in caso contrario.
- Expected loss: è una misura che valuta l'errore previsto o il costo associato alle decisioni prese da un modello di classificazione o di previsione. Rappresenta la media ponderata delle perdite associate a ciascuna previsione, considerando la probabilità delle diverse previsioni e delle relative perdite.

Federico Pomposelli, Marco Gallo, Nicholas De Marco