UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Corso di Laurea Magistrale in Ingegneria Informatica

Tesina di

Big Data Analytics and Business Intelligence

MovieLens

Anno accademico 2020/2021

Autori:
Guido Guarnieri M63/0994
Giustino Esposito M63/1024
Umberto Gagliardini M63/1023

Sommario

Capitolo 1 - Introduzione

1.1 Descrizione della challenge

L'obiettivo della challenge è quello di creare un sistema di raccomandazione di film. In questo modo si potranno raccomandare dei nuovi film da vedere a partire dai gusti degli utenti, in particolare basandosi sui voti assegnati ai film precedentemente visti.

1.2 Dataset

Il gruppo di ricerca GroupLens ha collezionato e reso disponibile i datasets di voti dal sito MovieLens (https://movielens.org). Questo dataset descrive con un sistema di votazione a 5 stelle un servizio di "movie reccomendation". Contiene 20000263 ratings e 465564 tag applications di 27278 film. Questi dati sono stati raccolti a partire da 138493 utenti tra il 9 Gennaio 1995 e il 31 Marzo 2015.

Gli utenti sono stati scelti in maniera random. Ogni utente ha votato almeno 20 film ed è rappresentato da un id senza rilevare nessun'altra informazione.

I file utilizzati sono due, in particolare "*movies.csv*", "*ratings.csv*" e "*links.csv*"

1.2.1 Features

File movies.csv

- movieId: identificativo del film usato da *<https://movielens.org>*
- title: titolo del film e anno di rilascio.
- genres : genere/i a cui appartiene il film.

File ratings.csv

- userId: Identificativo assegnato all'utente
- movieId: identificativo del film usato da *<https://movielens.org>*
- rating : voto in una scala da 0.5 a 5 assegnato dall'utente al film
- timestamp

File links.csv

- movieId: identificativo del film usato da *<https://movielens.org>*
- imdbId: identificativo del film usato da *<http://www.imdb.com>*
- tmdbId: identificativo del film usato da *<https://www.themoviedb.org>*

Capitolo 2 - Sistema di Raccomandazione

I **sistemi di raccomandazione** sono strumenti di sviluppo che aiutano l'utente ad ottenere solo le risorse più pertinenti alle sue esigenze, e lo orientano nello scegliere che cosa leggere, che cosa acquistare, dove andare, con chi mettersi in contatto. A tutti è capitato, consultando Amazon o cercando un programma su Netflix, di veder apparire, accanto al libro o al film che ci interessa, suggerimenti di altri libri e altri film o serie tv.

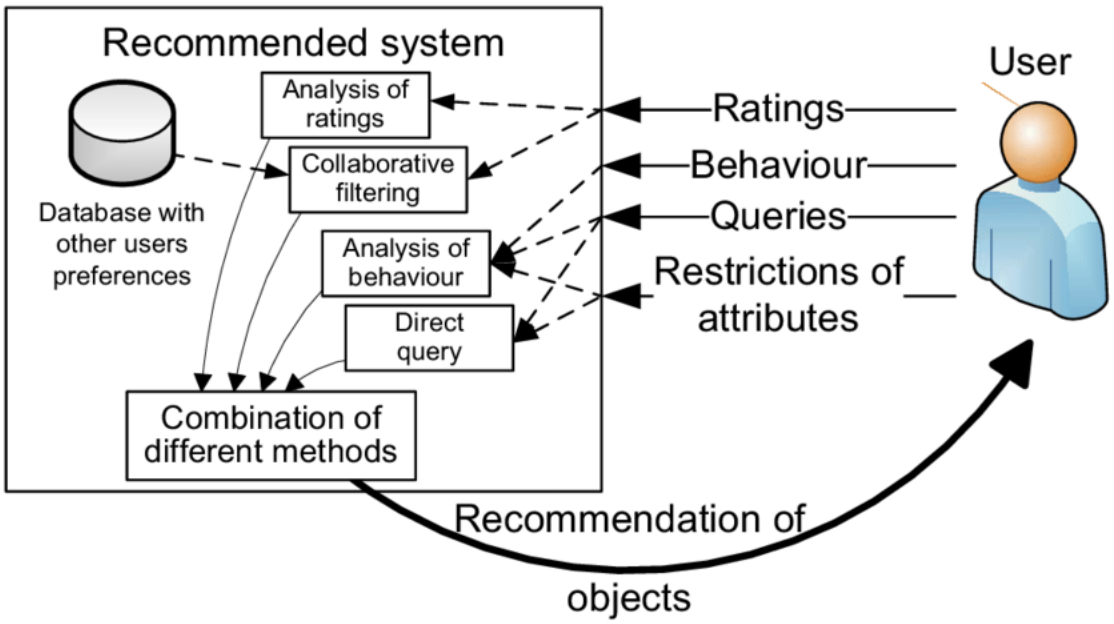
Un sistema di raccomandazione si basa su:

- items: gli oggetti da raccomandare;
- users: gli utenti di quel sistema;
- transactions: le interazioni avvenute fra utenti e sistema;
- rating: le valutazioni date dagli utenti in termini di rapporto qualità/prezzo.

Il sistema, e quindi l'algoritmo che seleziona le informazioni pertinenti, può essere:

- content based: l'item è ritenuto utile in base a item simili già considerati dall'utente;
- collaborative filtering: la valutazione di un item tiene conto di valutazioni fatte da utenti simili;
- demographic: gli utenti sono suddivisi in categorie a cui si indirizzano le raccomandazioni;
- knowledge based: la raccomandazione è fatta entro un ambito specifico di conoscenze;

Per questa challenge è stato scelto di utilizzare l'approccio del collaborative filtering.



2.1 Collaborative filtering

Il collaborative filtering si basa sull'assunzione che le persone a cui è piaciuto qualcosa in passato, piaceranno cose simili nel futuro. Il sistema genera raccomandazioni utilizzando solo informazioni sui voti di differenti utenti o di items.

Un vantaggio chiave di questo approccio è che è in grado di raccomandare items complessi in maniera molto accurata come nella fattispecie i film senza "comprendere" la natura intrinseca dell'item stesso. Inoltre per dare una misura di similarità tra items o utenti, sono stati utilizzati algoritmi basati sulla Pearson correlation oppure K-nearest-neighbor (K-NN).

Tuttavia questo approccio soffre dei seguenti problemi

- Cold start: per un nuovo utente o un item, non ci sono abbastanza dati per fare una raccomandazione accurata
- Scalabilità: In alcuni ambienti in cui questi sistemi fanno raccomandazioni, ci sono milioni di users e di items. Questo comporta la necessità di una potenza computazionale molto elevata.
- Sparsity: La maggior parte degli utenti avrà dato un voto ad un piccolo sottoinsieme di items. Per questo motivo anche gli items più richiesti godono di pochi ratings.

In questa challenge, per costruire il sistema di raccomandazione, è stato utilizzato come algoritmo di machine learning ALS (Alternating Least Squares)

2.2 Alternating Least Squares (ALS)

Gli utenti votano gli items e queste informazioni vengono utilizzate per predire ratings per altri oggetti. I dati di rating possono essere rappresentati come una matrice R di dimensione $m \times n$, dove n sono gli utenti e m gli items, nella fattispecie i film.

L'elemento (u,i)$^{\text{-simo}}$ della matrice R rappresenta il rating dell'utente u-simo all'items i-simo.

La matrice R è una matrice sparsa, in quanto gli oggetti non ricevono voti da molti utenti, per questo motivo sarà caratterizzato da un elevato numero di missing values. Per risolvere questo problema con matrice sparsa viene risolto attraverso la fattorizzazione di matrici.

Definendo due vettori k-dimensionali che prenderanno il nome di "fattori"

- X_u vettore di dimensione k che rappresenta ogni utente u
- Y_i vettore di dimensione k che rappresenta ogni item i

Siano

$$r_{ui} \approx x_u^T Y_i$$

$$x_u = x_1, x_2, \dots, x_k \in R^k$$

$$y_i = y_1, y_2, \dots, y_k \in R^k$$

Posso riscrivere la prima equazione come il seguente problema di ottimizzazione

$$argmin \sum_{r_u i} (r_{ui} - x_u^T y_i)^2 + \lambda (\sum_u \| x_u \|^2 + \sum_i \| y_i \|^2)$$

Dove λ è un fattore di regolarizzazione che viene utilizzato per risolvere problemi di overfitting e di default assume valore pari a 1.

Algorithm : *Alternating Least Squares (ALS)*

Procedure ALS (x_u ,y_i)
Initialization $x_u \leftarrow 0$
Initialization matrix y_i *with random values*
Repeat
    Fix y_i , *solve* x_u *by minimizing's the objective function (the sum of squared errors)*
    Fix y_i *solve* y_i *by minimizing the objective function similarly*
Until reaching the maximum iteration
Return x_u ,y_i
End procedure

Capitolo 3 – Implementazione ed architettura

In questo capitolo è riportata una descrizione dettagliata della soluzione proposta per la creazione del sistema di raccomandazione

3.1 Strumenti utilizzati

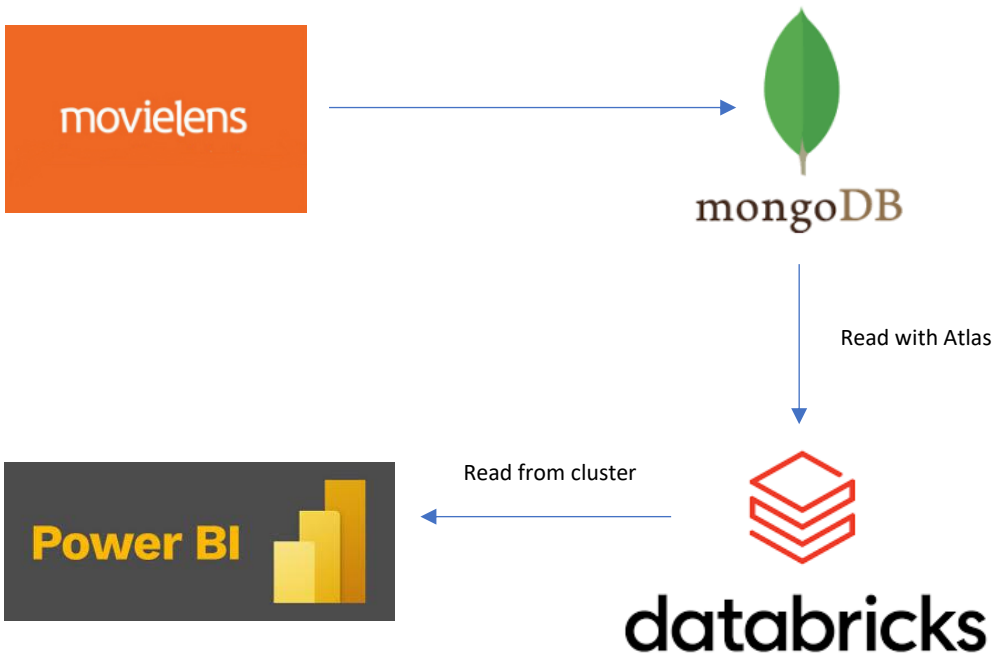
Per effettuare le Analisi e le Elaborazioni sono stati utilizzati i seguenti strumenti:

- MongoDB
- Databricks con Spark
- PowerBI

3.2 Architettura

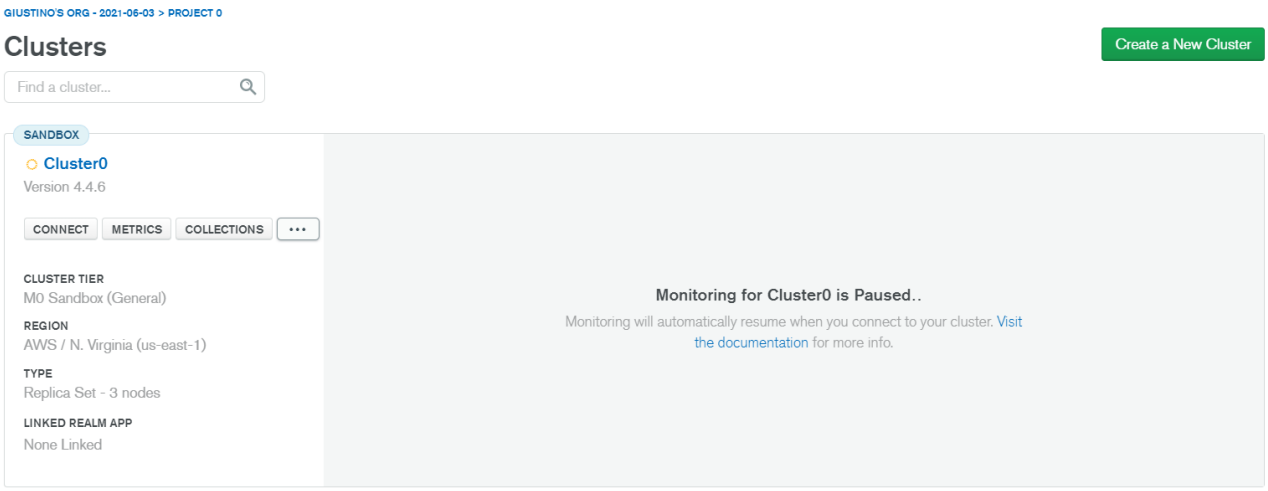
Da un punto di vista architetturale, si è scelto di importare i dataset rilevanti sul database mongoDB; tale database è stato, poi, collegato alla piattaforma databricks per consentire manipolazione dei dati e applicazione di algoritmi di raccomandazione; infine, i risultati sono stati importanti sul software di business intelligence Power Bi.

Di seguito è mostrata l'architettura proposta per la soluzione:



3.3 Database - MongoDB

Per quanto riguarda il database, come accennato in precedenza, è stato utilizzato MongoDB in combinazione con MongoDB Atlas. Il collegamento con quest'ultimo è stato necessario per ottenere un link con la piattaforma Databricks che verrà illustrata in seguito. Atlas MongoDB è un database NoSql e cloud-based che mette a disposizione le funzionalità di MongoDB su cloud.



In particolare, è stato utilizzato un cluster di tipo M0 che mette a disposizione una quantità di storage limitata ma in maniera gratuita.

Di seguito vengono presentati i campioni delle collezioni ottenute a partire dai 3 dataset accennati in precedenza:


```
_id: ObjectId("60b8cf6cb3f21c18049184e0")
movieId: 1
title: "Toy Story (1995)"
genres: "Adventure|Animation|Children|Comedy|Fantasy"
```

```
_id: ObjectId("60b8cf6cb3f21c18049184e1")
movieId: 2
title: "Jumanji (1995)"
genres: "Adventure|Children|Fantasy"
```

```
_id: ObjectId("60b8cf6cb3f21c18049184e2")
movieId: 3
title: "Grumpier Old Men (1995)"
genres: "Comedy|Romance"
```

```
_id: ObjectId("60b8cf6cb3f21c18049184e3")
movieId: 4
title: "Waiting to Exhale (1995)"
genres: "Comedy|Drama|Romance"
```

```
_id: ObjectId("60b8cf6cb3f21c18049184e4")
movieId: 5
title: "Father of the Bride Part II (1995)"
genres: "Comedy"
```


Collection movies

```
_id: ObjectId("60b8ca74b3f21c18046056d8")
userId: 1
movieId: 2
rating: 3
timestamp: 1112486027
```

```
_id: ObjectId("60b8ca74b3f21c18046056d9")
userId: 1
movieId: 29
rating: 3
timestamp: 1112484676
```

```
_id: ObjectId("60b8ca74b3f21c18046056da")
userId: 1
movieId: 32
rating: 3
timestamp: 1112484819
```

```
_id: ObjectId("60b8ca74b3f21c18046056db")
userId: 1
movieId: 47
rating: 3
timestamp: 1112484727
```


Collection ratings


```
_id: ObjectId("60c8bf2464413e1698e79129")
movieId: 1
imdbId: 114709
tmdbId: 862
```

```
_id: ObjectId("60c8bf2464413e1698e7912a")
movieId: 2
imdbId: 113497
tmdbId: 8844
```

```
_id: ObjectId("60c8bf2464413e1698e7912b")
movieId: 3
imdbId: 113228
tmdbId: 15602
```

```
_id: ObjectId("60c8bf2464413e1698e7912c")
movieId: 4
imdbId: 114885
tmdbId: 31357
```


Collection links

3.3.1 Operazioni database

Di seguito viene mostrata una prima operazione effettuata su MongoDB, in particolare viene mostrato come, a partire, dal dataset movies, sfruttando il campo title, è stato estratto l'anno di ciascun film. Nell'immagine seguente è illustrato il risultato ottenuto.

Collection movieId_anno

Essendo l'anno contenuto nel titolo tra parentesi e data la presenza di anche altre parentesi, la collection mostrata è stata ottenuta creando due collection temporanee: una relativa agli anni antecedenti al 2000 ed un'altra relativa agli anni 2000. Infine, le due view sono state riunite nella singola collection.

Di seguito è riportato il codice utilizzato solo per creare la collection relativa agli anni prima del 2000:

```
result = client['final_project']['movies'].aggregate([
    {
        '$set': {
            'title': {
                '$split': [
                    '$title', '(19'
                ]
            }
        }
    }, {
        '$set': {
            'title': {
                '$arrayElemAt': [
                    '$title', 0
                ]
            },
            'anno': {
                '$arrayElemAt': [
                    '$title', 1
                ]
            }
        }
    }, {
        '$set': {
            'anno': {
                '$split': [
                    '$anno', ')'
                ]
            }
        }
    }, {
        '$set': {
            'anno': {
                '$arrayElemAt': [
                    '$anno', 0
                ]
            }
        }
    }, {
        '$project': {'movieId': 1, 'anno': 1
        }
    }, {
        '$merge': {
            'into': 'movieID_anno',
            'on': '_id'
        }
    }
])
```

Codice movieId_anno

La collection degli anni 2000 è stata ottenuta usando lo stesso codice sulla collection appena creata aggiungendo all'inizio della pipeline il seguente codice:

```
{
    '$match': {
        'anno': {
            '$ne': None
        }
    }
}
```

Match

In questo modo è stata creata anche la collection relativa agli anni 2000 e da quest'ultima è bastato effettuare il seguente merge:

```
result = client['final_project']['anno_temp2000'].aggregate([
    {
        '$merge': {
            'into': 'movieID_anno',
            'on': '_id'
        }
    }
])
```

Merge

Una seconda operazione effettuata è stata la creazione di una nuova collection che mettesse in relazione l'id di ciascun film con il rispettivo link al sito https://www.themoviedb.org/movie/ . Per fare ciò, è stato sfruttato il dataset links, il quale contiene gli indici di ciascun film su vari siti, tra i quali è stato selezionato *themoviedb.org*.

```
_id: ObjectId("60c8bf2464413e1698e79129")
movieId: 1
link: "https://www.themoviedb.org/movie/862"

_id: ObjectId("60c8bf2464413e1698e7912a")
movieId: 2
link: "https://www.themoviedb.org/movie/8844"

_id: ObjectId("60c8bf2464413e1698e7912b")
movieId: 3
link: "https://www.themoviedb.org/movie/15602"

_id: ObjectId("60c8bf2464413e1698e7912c")
movieId: 4
link: "https://www.themoviedb.org/movie/31357"

_id: ObjectId("60c8bf2464413e1698e7912d")
movieId: 5
link: "https://www.themoviedb.org/movie/11862"
```

Collection movieId_links

Creata col seguente codice:

```
result = client['final_project']['links'].aggregate([
    {
        '$addFields': {
            'link': {
                '$concat': [
                    'https://www.themoviedb.org/movie/', {
                        '$convert': {
                            'input': '$tmdbId',
                            'to': 'string'
                        }
                    }
                ]
            }
        }
    }, {
        '$project': {
            'movieId': 1,
            'link': 1
        }
    }, {
        '$merge': {
            'into': 'movieId_links',
            'on': '_id'
        }
    }
])
```

Codice movieId_links

3.4 Manipolazione dati - Databricks

Databricks è stato utilizzato per il suo supporto a Spark, il quale permette di manipolare facilmente e velocemente i dati.

In particolare, in questo paragrafo è spiegato come è stato creato il sistema di raccomandazione e come sono stati trattati i dati prima di inoltrarli a PowerBI per creare la dashboard.

3.4.1 Caricamento dati

La prima operazione effettuata con Databricks, ovviamente, è il caricamento dei dati sfruttando un collegamento con MongoDB Atlas.

In particolare, è stato collegato mongoDB Compass ad Atlas in modo da avere un cluster con un certo URL ed in seguito è stato eseguito il seguente codice per caricare i dati da Atlas su Databricks:

```
from pyspark.sql import SparkSession
database = "final_project"
collection_movies = "movies" #your collection name
connectionString= "mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass&ssl=false"
spark = SparkSession\
    .builder\
    .config('spark.mongodb.input.uri',connectionString)\
    .config('spark.mongodb.output.uri', connectionString)\
    .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-connector_2.12:3.0.1')\
    .getOrCreate()
# Reading from MongoDB
df_movies = spark.read\
    .format("mongo")\
    .option("uri", connectionString)\
    .option("database", database)\
    .option("collection", collection_movies)\
    .load()
```

Caricamento collection da mongodb

Come si può notare l'unica collection caricata è *movies* poiché il codice mostrato è solo un esempio di come caricare i dati direttamente da MongoDB. In realtà i dati sono stati caricati su Databricks da locale poiché Atlas mette a disposizione gratuitamente uno storage limitato che non ha permesso di caricare tutti i file a disposizione.

Un esempio di codice usato per caricare il resto delle collection è il seguente:

```
#caricamento dataframe
file_location = "/FileStore/tables/ratings.csv"
file_type = "csv"
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","
df_rating = spark.read.format(file_type) \
  .option("inferSchema", "true") \
  .option("header", "true") \
  .option("sep", delimiter) \
  .load(file_location)
df_rating=df_rating.select("userID","movieID","rating")
df_rating.printSchema()
```

Caricamento collection da locale

In questo modo le *collection* di MongoDB sono state caricate all'interno dei seguenti *dataframe* Spark:

- Df_rating
- Df_movies
- Df_links
- Df_anno

A questo punto non è necessaria alcuna fase di *pre-processing* ed è possibile passare direttamente alla creazione del modello per il sistema di raccomandazioni.

3.4.2 Sistema di raccomandazione

Per la creazione del sistema di raccomandazione tramite ALS si ricorda che servono unicamente:

- movieId
- userId
- rating

e tutti questi valori sono contenuti nel dataframe df_rating, quindi sarà l'unico ad essere utilizzato.

La prima operazione da eseguire è lo split del dataframe in training e test set, in questo modo sarà possibile in seguito effettuare una stima della bontà del modello:

```
#split in train e test set
(train, test) = df_rating.randomSplit([0.8, 0.2], seed = 2020)
```

1 Split dataset

A questo punto si richiama la funzione ALS di python per la definizione del modello ALS:

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

#creazione modello ALS
als = ALS(
         userCol="userID",
         itemCol="movieID",
         ratingCol="rating",
         nonnegative = True,
         implicitPrefs = False,
         coldStartStrategy="drop"
)
```

Creazione ALS model

Da notare che all'interno della funzione sono stati settati i seguenti parametri:

- nonnegative=True, in modo da avere i ratings predetti non negativi;
- implicitPrefs=False, per indicare che le votazioni sono esplicite;
- coldStartStrategy=drop, in questo modo l'algoritmo scarta utenti che non hanno votato nessun film e film che non hanno nessuna votazione, altrimenti potrebbe restituire delle votazioni predette pari a NaN.

Per utilizzare il modello ALS è necessario settare alcuni iperparametri: rango, parametro di regolarizzazione e numero di iterazioni. Quindi, è stato creato un modello in cui sono stati inseriti vari valori per questi iperparametri, il quale in seguito verrà utilizzato in una cross-validation:

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator

#creazione di una griglia di iperparametri con i valori da testare
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [100,150]) \
            .addGrid(als.regParam, [.1, .15]) \
            .build()

#creazione evaluator di tipo rmse per la crossvalidation
evaluator = RegressionEvaluator(
           metricName="rmse",
           labelCol="rating",
           predictionCol="prediction")
#modello per la crossvalidation
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=2)
```

Creazione evaluator model

Di seguito è riportato il codice che effettua la cross-validation ed estrae il modello migliore:

```
import pandas as pd
import matplotlib.pyplot as plt

from numpy import savetxt

from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

#esecuzione crossvalidation
model = cv.fit(train)
#estrazione best model dalla validazione
best_model = model.bestModel
```

Cross-validation

```
#print results
print("**Best Model**")
# Print "Rank"
print("  Rank:", best_model._java_obj.parent().getRank())
# Print "MaxIter"
print("  MaxIter:", best_model._java_obj.parent().getMaxIter())
# Print "RegParam"
print("  RegParam:", best_model._java_obj.parent().getRegParam())
```

```
**Best Model**
  Rank: 50
  MaxIter: 10
  RegParam: 0.05
```

Migliori iperparametri

A questo punto è possibile effettuare anche un testing per avere una stima della bontà del modello:

```
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print(RMSE)
```

```
0.785846517612491
```

Testing e RMSE

Il valore ottenuto di RMSE è abbastanza buono considerando che i valori di rating predetti superano anche il 5, quindi al più il modello può sbagliare di 0,78 punti.

Quindi ora si ha a disposizione un modello che permette di effettuare raccomandazioni agli utenti già presenti nel database.

Il seguente codice mostra come sono state ottenute 10 raccomandazioni per ogni utente ed inoltre, ad ogni movieId associa anche il titolo ed il genere ottenuti dal dataframe df_movies tramite un join:

```
#Generazioni 10 raccomandazioni per tutti gli utenti
recommendations = best_model.recommendForAllUsers(10)
from pyspark.sql.functions import explode,col
nrecommendations = recommendations\
    .withColumn("rec_exp", explode("recommendations"))\
    .select('userID', col("rec_exp.movieID"), col("rec_exp.rating"))
df_recc_tot=nrecommendations.join(df_movies, on='movieId').sort("rating", ascending = False)
```

Raccomandazioni

Infine, tramite la funzione *saveAsTable,* il dataframe appena ottenuto è stato salvato in una tabella di Databricks per poterla esportare in seguito su PowerBI:

Schema:

	col_name	data_type	comment
1	movieID	int	
2	userID	int	
3	rating	float	
4	title	string	
5	genres	string	
6			
7	# Partitioning		

Showing all 8 rows.

Sample Data:

	movieID	userID	rating	title	genres
1	104390	79274	8.545578	When Nietzsche Wept (2007)	Drama
2	123571	108065	8.5664835	Jim Jefferies: Alcoholocaust (2010)	Comedy
3	104390	116448	8.585828	When Nietzsche Wept (2007)	Drama
4	74159	12168	8.612513	Ethan Mao (2004)	Drama|Thriller
5	97300	67836	8.630106	Björk: Volumen (1999)	Animation|Musical
6	116821	86451	8.631997	Dirty Bomb (2011)	Comedy
7	74159	106593	8.634046	Ethan Mao (2004)	Drama|Thriller

Campione tabella raccomandazioni

3.4.3 Join dataframe

Per facilitare la creazione della dashboard è stato effettuato anche un join tra i dataframe df_rating e df_movies:

```
1  #join dei due dataframe
2  movie_ratings = df_rating.join(df_movies, ['movieId'], 'left')
3  movie_ratings.show()
```

▸ (1) Spark Jobs

▸ movie_ratings: pyspark.sql.dataframe.DataFrame = [movieID: integer, userID: integer ... 3 more fields]

```
+-------+------+------+--------------------+--------------------+
|movieID|userID|rating|               title|              genres|
+-------+------+------+--------------------+--------------------+
|      2|     1|   3.5|      Jumanji (1995)|Adventure|Childre...|
|     29|     1|   3.5|City of Lost Chil...|Adventure|Drama|F...|
|     32|     1|   3.5|Twelve Monkeys (a...|Mystery|Sci-Fi|Th...|
|     47|     1|   3.5|Seven (a.k.a. Se7...|    Mystery|Thriller|
|     50|     1|   3.5|Usual Suspects, T...|Crime|Mystery|Thr...|
|    112|     1|   3.5|Rumble in the Bro...|Action|Adventure|...|
|    151|     1|   4.0|      Rob Roy (1995)|Action|Drama|Roma...|
|    223|     1|   4.0|       Clerks (1994)|              Comedy|
|    253|     1|   4.0|Interview with th...|        Drama|Horror|
```

Join df_rating-df_movies

In questo modo per ogni utente sarà possibile sapere che votazione ha dato a determinati titoli.

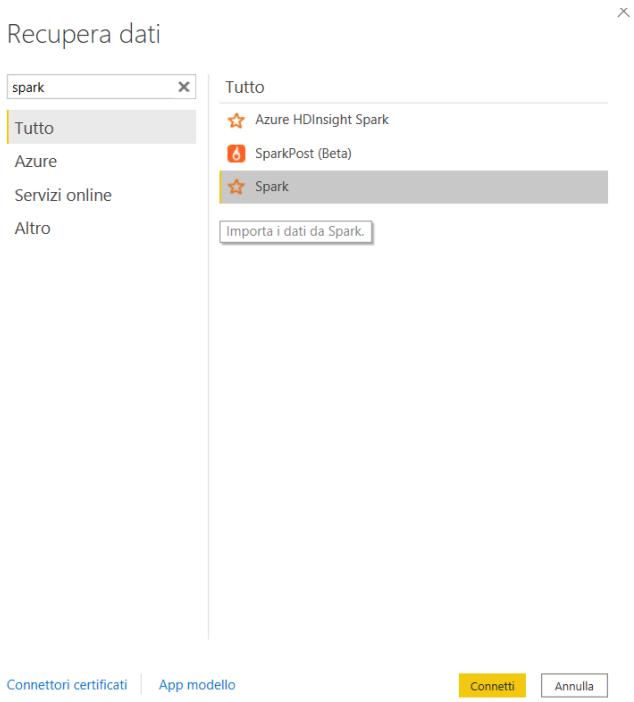
3.5 Business Intelligence -PowerBI

In questo paragrafo verrà mostrata la realizzazione di una dashboard sul software Power Bi in grado di presentare i dati più rilevanti estratti dai dataset e ottenuti mediante manipolazione su Databricks.

3.5.1 Collegamento con Databricks

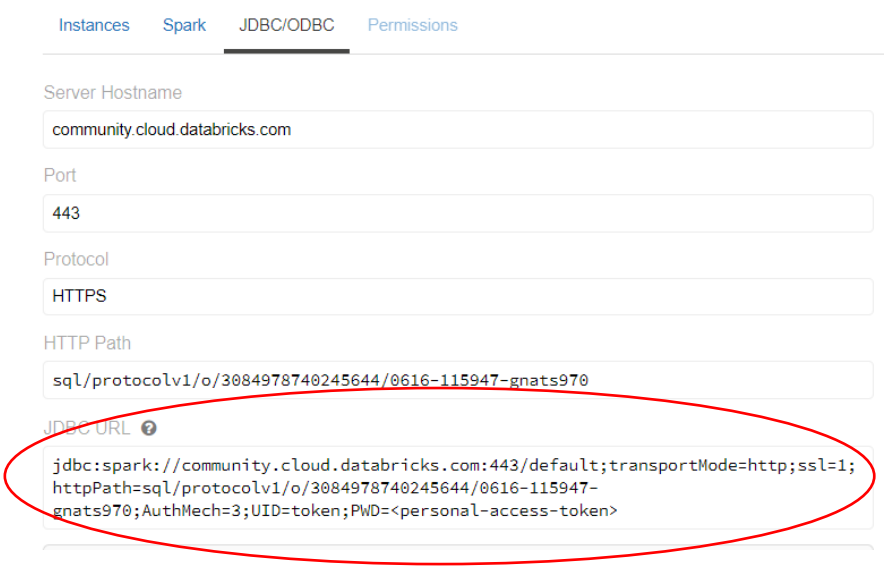
Per prelevare i dati direttamente da Databricks è stata effettuata la seguente procedura direttamente da PowerBI:

1) recupera dati da spark



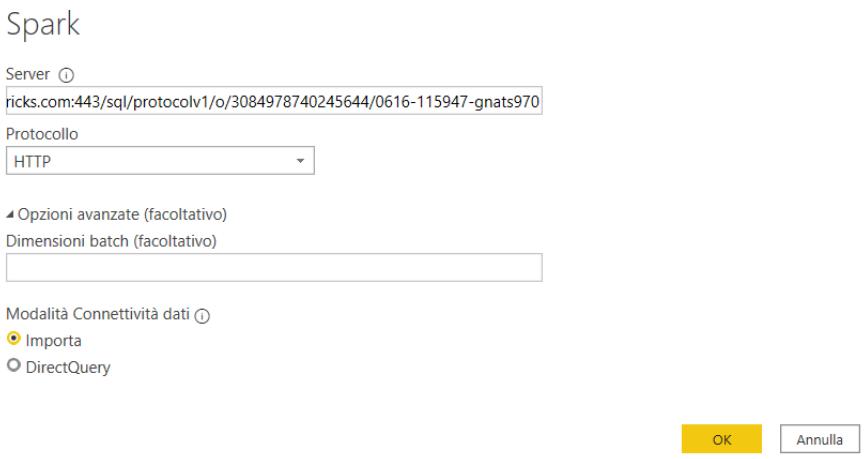
Recupera dati: spark

2) ottenere l'URL del cluster di databricks:



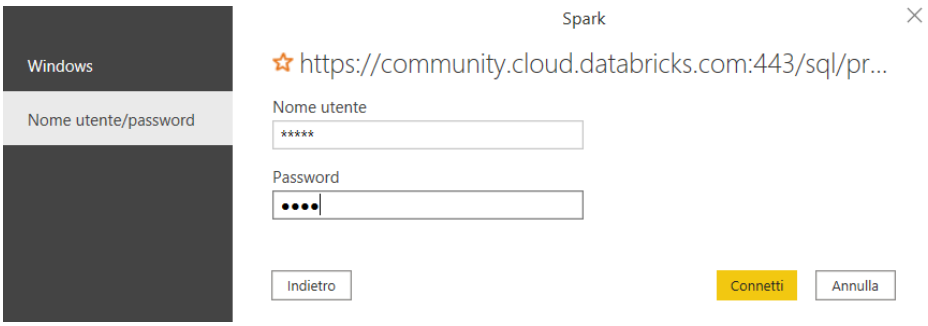
URL cluster

3) modificare opportunamente l'URL ed inserirlo in PowerBI:



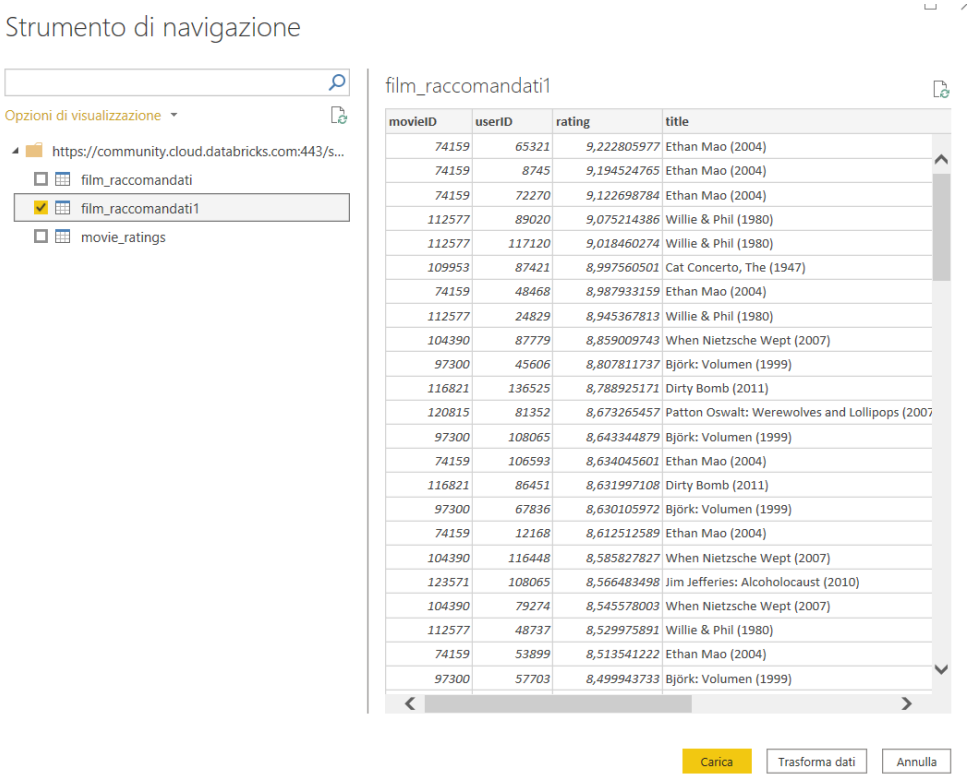
Collegamento URL

4) Accedere con le proprie credenziali:



Accesso

A questo punto PowerBI mostrerà tutte le table salvate in databricks e sarà possibile caricarle:



Load table

3.5.2 Dashboard

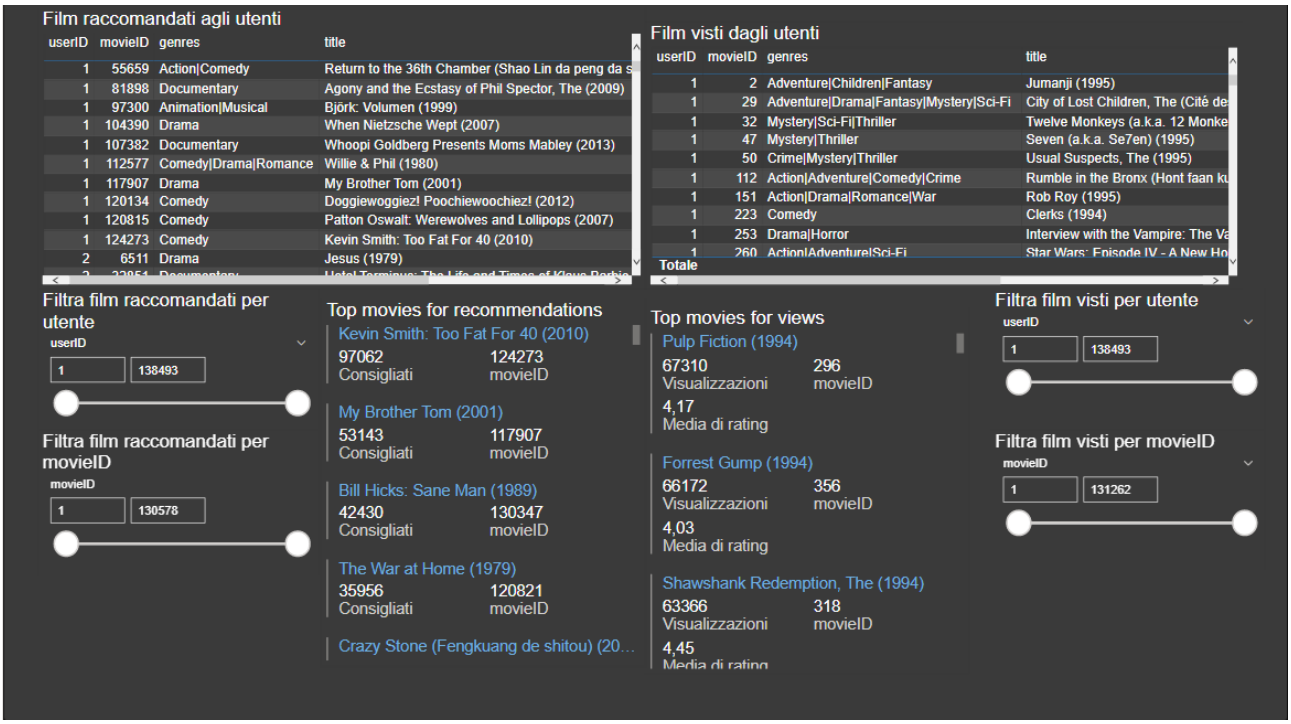
Si è deciso di utilizzare tre pagine per la realizzazione della dashboard su Power Bi:

1. Contenente grafici relativi sia ai film già visti dagli utenti sia quelli consigliati dal sistema di raccomandazione.
2. Contenente grafici relativi alla suddivisione dei film, visti e consigliati, in base al genere.
3. In grado di mostrare a un utente quali sono i film più consigliati in base al genere da lui selezionato.

3.5.3 Pagina 1

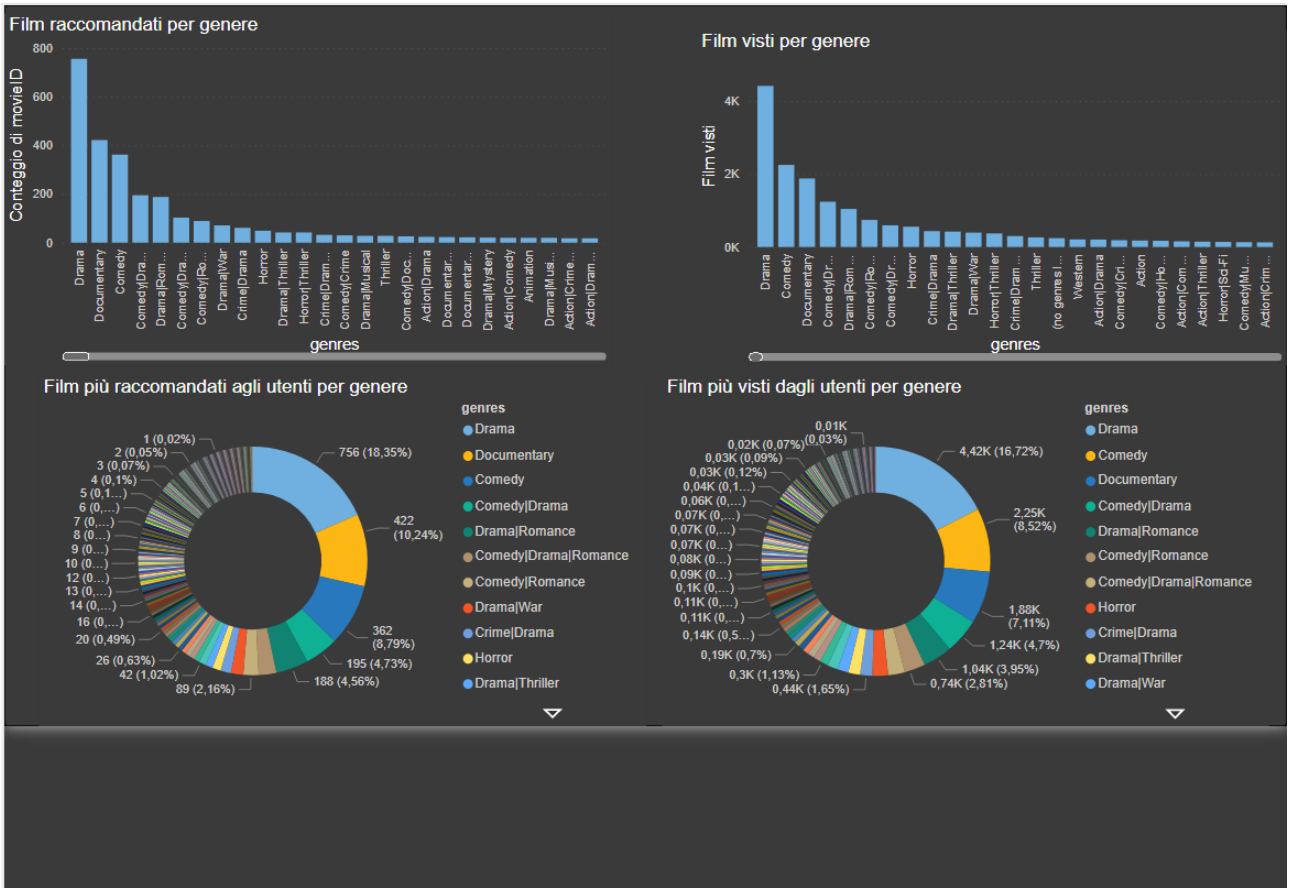
Nella prima pagina l'obiettivo è stato quello di dare una panoramica generale relativa ai due dataset utilizzati, dando la possibilità all'utente di filtrare in base all'id dell'utente e del film. Inoltre, sono presenti due schede contenenti, in maniera ordinata, i film più visti e i film più raccomandati mostrando rispettivamente il numero di visualizzazioni (con relativo rating medio) e il numero di utenti a cui è stato raccomandato quel film.

Vale la pena fare un'osservazione sulle due schede mostrate nella pagina: i film con maggiori visualizzazioni (si noti *Pulp Fiction* o *Forrest Gump)* non risultano anche tra i film più raccomandati in quanto, banalmente, se un film è stato già visto da migliaia di persone saranno poche le rimanenti a cui si può raccomandare.



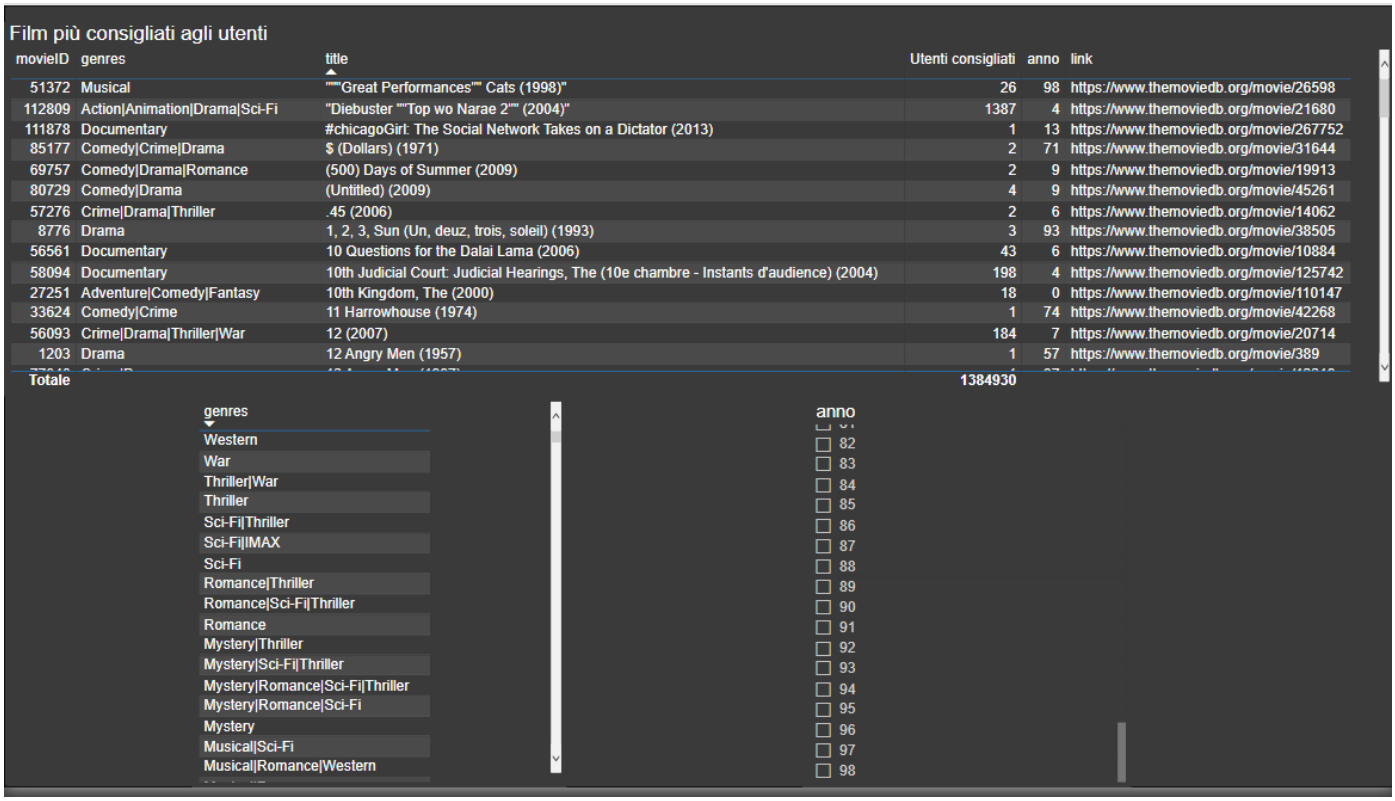
3.5.4 Pagina 2

Per quanto riguarda la pagina 2, sono stati utilizzati due istogrammi e due grafici ad anello: l'obiettivo, in entrambi i casi, è lo stesso, cioè mostrare la suddivisione dei film per genere nella raccolta dei film visti dagli utenti e verificare se, in percentuale, tale suddivisione sussiste anche nei film raccomandati agli utenti. Si può notare dall'istogramma *Film visti per genere* e dal grafico ad anello *Film più visti dagli utenti per genere* come i tre generi più visti dagli utenti siano drama, comedy e documentary; a questo punto, sfruttando i grafici *Film raccomandati per genere* e *Film più raccomandati agli utenti per genere,* si può notare come il sistema di raccomandazione consigli agli utenti effettivamente i generi maggiormente visti.

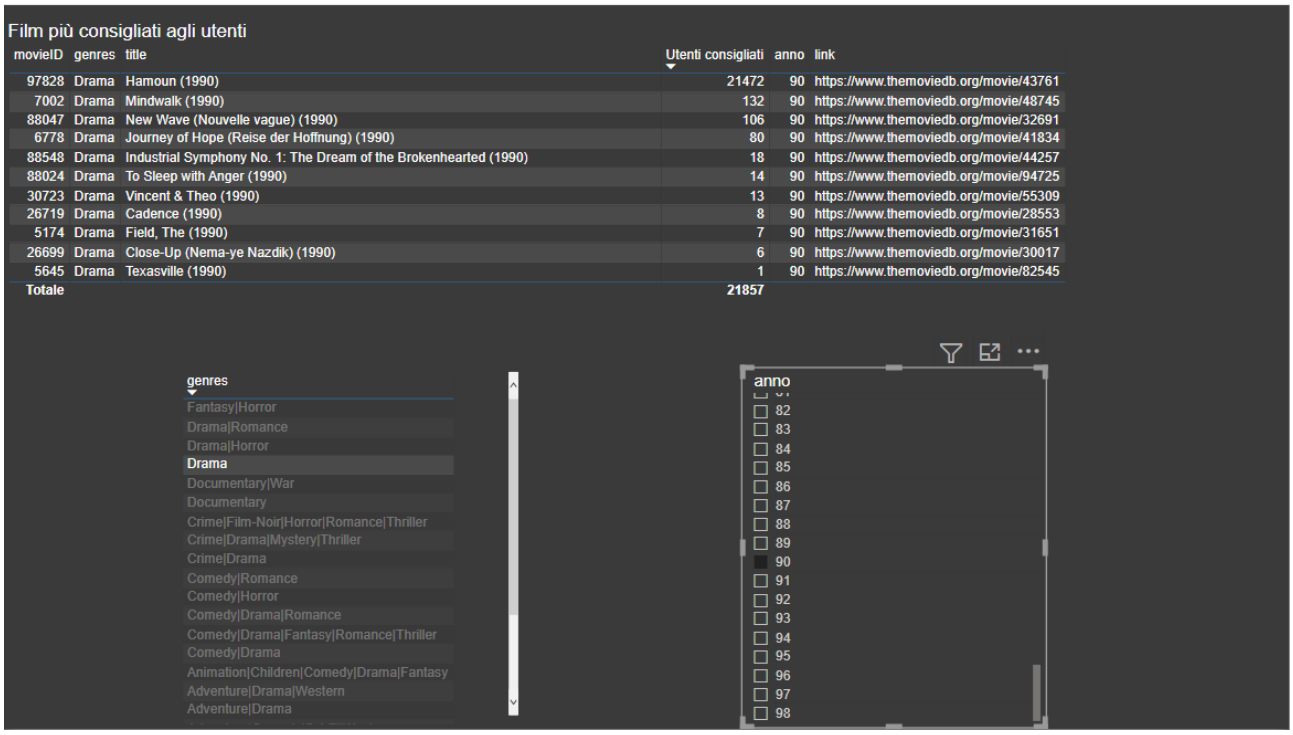


3.5.5 Pagina 3

Per quanto riguarda la pagina 3, l'obiettivo è stato quello di simulare uno scenario in cui un nuovo utente vuole visualizzare, in base ai suoi generi preferiti, quali sono i film maggiormente raccomandati dal sistema. Per fare ciò, viene messa a disposizione una tabella contenente i film raccomandati con il numero di utenti a cui è stato raccomandato quel film e con un link che rimanda a www.themoviedb.org dove è possibile visualizzare la scheda del film. L'utente, pertanto, deve soltanto selezionare, nella tabella relativa ai generi, i suoi preferiti, ed eventualmente l'anno del film che vorrebbe vedere. Dopodiché, la tabella delle raccomandazioni sarà filtrata e ordinata in base al numero di utenti a cui quello specifico film è stato raccomandato.



Di seguito viene mostrato un esempio in cui si vuole vedere quali sono i film drammatici più raccomandati dell'anno 1990.



Come si può notare, la tabella restituisce in output una lista di film drama consigliati, ordinata per numero di utenti a cui il sistema di raccomandazione ha consigliato quello specifico film.