Title: Movie Recommendation Engine

Abstract

Online Recommender System are ubiquitous in present world. Due to their wide applicability, recommendation systems have become an area of active research in the field of Machine Learning. Several machine learning related algorithms – baseline predictor,Stochastic Gradient Descent, KNN, SVD, SVD++, asymmetric SVD, Bayesian Factorisation etc are used to predict the rating from particular users for unrated movies.These different models are compared using RMSE (Root-Mean-Square-Error) as the main criteria to evaluate their performance. The simulation result shows distinct performance due to selected algorithms as well as the corresponding learning rates.

Here, we have implemented various feature based filtering algorithms as different classes (models)
We use python and Machine learning to build a popularity based recommendation engine, which recommends movies to the user based on his/her previous choices and ratings.
Thus, we produce personalized suggestions to the user based on his previous choice and rating history.
Here we have used the IMDB movie dataset to train our model for 45000 movies.

This recommendation engine can also be used for recommending something else like products from an e-commerce site. So according to the dataset this project finds use in recommending various services to the user, extremely useful for service providers like flipkart, amazon, netflix, spotify etc.

Made by-

RA1611003010987
Shivang Kaul

RA1611003010907
Yash Aggarwal

Introduction

In the digital world, recommendation systems play a significant role - both for the users and for the company/platform/sellers.For the users, a new world of options are thrown up - that were hitherto tough to find.

For companies, it helps drive up user engagement and satisfaction, directly impacting their bottom line.If you’ve shopped on e-commerce platforms like Amazon or Flipkart, you would’ve seen options like:

“People who viewed this product also viewed…” “Products similar to this one…”

These are the results from recommendation systems. Netflix threw up a major data science challenge last decade: a million dollars to anyone who can improve their recommendation system by 10%. A recent estimate pegs the value of Netflix’s recommendation system to be worth $ 1 Billion.In this full-day workshop, we will walk you through the various types of recommendation system. By the end of the workshop, you will have enough knowledge to build one for your problem.

So here we use the IMDB movie dataset with an array of 45000 movies to demonstrate recommending movies based on the user’s choice. We take a movie input from the user and then using feature based filtering algorithm print the top 10 most similar movies to the movie chosen by user.

This system can be extended to recommending anything based on the dataset fed into the system.

Problem Description

The problem that we address in this project can be formulated as follows. Let us have a table with 45000 rows and 24 columns with information on keywords in description movie id, cast, adult rating, vote count etc…

We rate the movies using the following formula:

$$\text{Rating} = (v/(v+m) * R) + (m/(m+v) * C)$$

Where, v= vote count of each movie

R=vote average of this movie

m=data in dataset

C=mean of all the votes

Using this formula all movies are ranked and sorted. Then we extract the keywords from the description and using the user input form a similarity matrix and recommend 10 movies similar to the user’s choice.

Algorithm

1. Read the dataset values into a list
2. Compute the mean scores by using the IMDB formula and store it for each movie.
3. Sort the movies based on the score
4. Take a movie as input from user
5. Call the recommender function:

   Recommender func:

6. Create a similarity matrix using the scikit modules feature extraction.
7. Compute similarity scores of each movie and store in a list.
8. Sort the list and print the top 10 elements of the list.

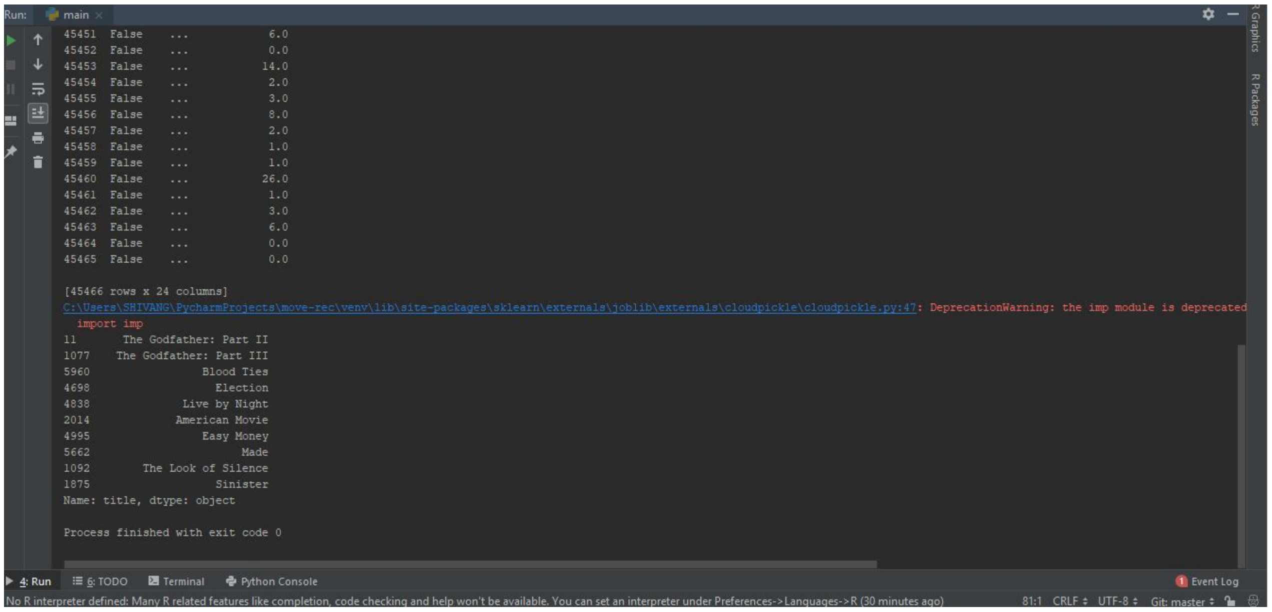
Python Code

```
import pandas as pd

metadata = pd.read_csv('the-movies-dataset/movies_metadata.csv',
low_memory=False)

print(metadata)

# Calculate C
C = metadata['vote_average'].mean()

#print(C)

m = metadata['vote_count'].quantile(0.80)
#print(m)

# Filter out all qualified movies into a new DataFrame
q_movies = metadata.copy().loc[metadata['vote_count'] >= m]
#print(q_movies.shape)

# Function that computes the weighted rating of each movie
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)

# Define a new feature 'score' and calculate its value with
`weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

```
#Sort movies based on score calculated above
metadata = q_movies.sort_values('score', ascending=False)
metadata = metadata.reset_index()
#Print the top 1500 movies
#print(q_movies[['title', 'vote_count', 'vote_average', 'score']].head(1500))

#print(metadata['overview'].head())


#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such
as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
metadata['overview'] = metadata['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(metadata['overview'])
#Output the shape of tfidf_matrix
#print(tfidf_matrix.shape)

# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
#Construct a reverse map of indices and movie titles
indices = pd.Series(metadata.index,
index=metadata['title']).drop_duplicates()


# Function that takes in movie title as input and outputs most similar
movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]
print(get_recommendations('The Godfather'))
```

<u>Output Screen</u>



```
45451  False    ...           6.0
45452  False    ...           0.0
45453  False    ...          14.0
45454  False    ...           2.0
45455  False    ...           3.0
45456  False    ...           8.0
45457  False    ...           2.0
45458  False    ...           1.0
45459  False    ...           1.0
45460  False    ...          26.0
45461  False    ...           1.0
45462  False    ...           3.0
45463  False    ...           6.0
45464  False    ...           0.0
45465  False    ...           0.0

[45466 rows x 24 columns]
C:\Users\SHIVANG\PycharmProjects\move-rec\venv\lib\site-packages\sklearn\externals\joblib\externals\cloudpickle\cloudpickle.py:47: DeprecationWarning: the imp module is deprecated
  import imp
11       The Godfather: Part II
1077    The Godfather: Part III
5960                Blood Ties
4698                  Election
4838             Live by Night
2014            American Movie
4995                Easy Money
5662                      Made
1092        The Look of Silence
1875                  Sinister
Name: title, dtype: object

Process finished with exit code 0
```

No R interpreter defined: Many R related features like completion, code checking and help won't be available. You can set an interpreter under Preferences->Languages->R (30 minutes ago)   81:1 CRLF ÷ UTF-8 ÷ Git: master ÷

The above screen shows the 10 most similar movies to <u>GODFATHER</u>

Conclusion

This project successfully recommends similar movies based on user’s choice of favourite movies. This can be further extended to items such as commodity items, a specific brand of item that the user normally buys and can help give efficient suggestions on what the user needs to buy, sellers can also use this information to research which item is in more demand by the users.

Thus, the recommendation system can be used by both producer and consumer and make the process more efficient by providing accurate suggestions.