Movie Recommendation System

PYTHON

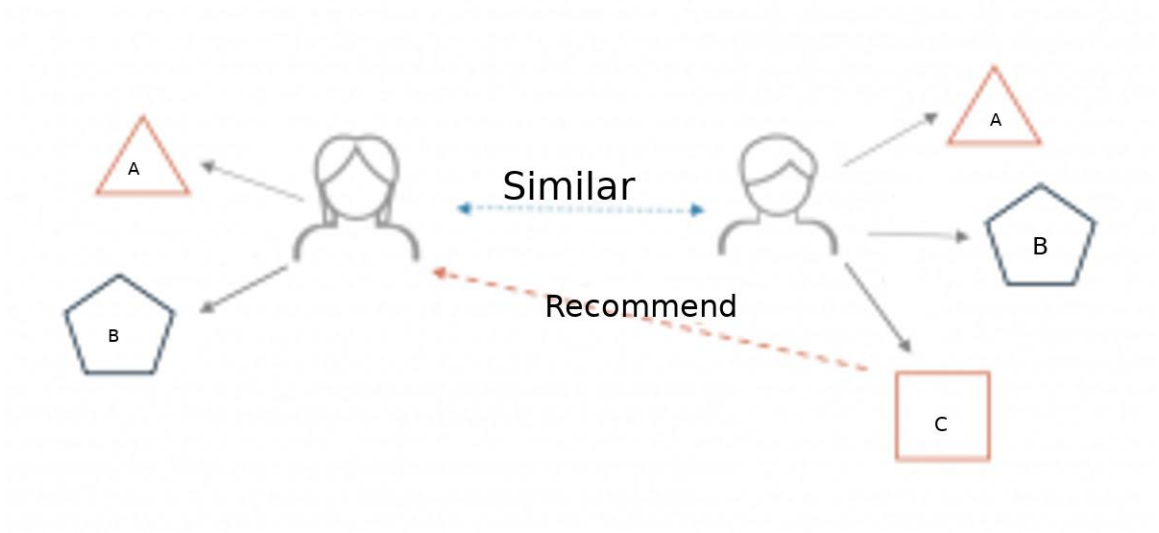
B. Rebecca Rejoice | Data Analytics and Machine Learning

PROJECT REPORT

What is a recommender system?

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.

Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.



Two users buy the same items A and B from an e-commerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

Different types of recommendation engines

The most common types of recommendation systems are **content-based** and **collaborative filtering** recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models **Memory-based** methods and **Model-based** methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

- **User-based collaborative filtering**: In this model, products are recommended to a user based on the fact that the products have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie come out that Derick like, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
- **Item-based collaborative filtering**: These systems identify similar items based on users' previous ratings. For example, if users A, B, and C gave a 5-star rating to books X and Y then when a user D buys book Y they also get a recommendation to purchase book X because the system identifies book X and Y as similar based on the ratings of users A, B, and C.

Model-based methods are based on Matrix Factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve accuracy. Examples of such model-based methods include Decision trees, Rule-based Model, Bayesian Model, and latent factor models.

Content-based systems use metadata such as genre, producer, actor, musician to recommend items say movies or music. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Similarly, you can get music recommendations from certain artists because you liked their music. Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.

Datasets to use for building recommender systems

In this tutorial, we are going to use the Movie Lens Data Set. This dataset was put together by the Group lens research group at the University of Minnesota. It contains 1, 10, and 20 million ratings. Movie lens also has a website where you can sign up, contribute reviews and get movie recommendations.

Walkthrough of building a recommender system

We are going to use the movie lens to build a simple item similarity-based recommender system. The first thing we need to do is to import pandas and numpy.

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

Next, we load in the data set using pandas `read_csv()` utility. The dataset is tab separated so we pass in `\t` to the `sep` parameter. We then pass in the column names using the `names` parameter.

```
df = pd.read_csv('u.data', sep='\t',
names=['user_id','item_id','rating','titmestamp'])
```

Now let's check the head of the data to see the data we are dealing with.

```
df.head()
```

It would be nice if we can see the titles of the movie instead of just dealing with the IDs. Let's load in the movie titles and merge it with this dataset.

```
movie_titles = pd.read_csv('Movie_Titles')
movie_titles.head()
```

Since the `item_id` columns are the same we can merge these datasets on this column.

```
df = pd.merge(df, movie_titles, on='item_id')
df.head()
```

Let's look at what each column represents:

- `user_id` - the ID of the user who rated the movie.
- `item_id` - the ID of the movie.
- `rating` - The rating the user gave the movie, between 1 and 5.
- `timestamp` - The time the movie was rated.
- `title` - The title of the movie.

Using the `describe` or `info` commands we can get a brief description of our dataset. This is important in order to enable us to understand the dataset we are working with.

```
df.describe()
```

We can tell that the average rating is 3.52 and the max is 5. We also see that the dataset has 100003 records.
Let's now create a data frame with the average rating for each movie and the number of ratings. We are going to use these ratings to calculate the correlation between the movies later. Correlation is a statistical measure

that indicates the extent to which two or more variables fluctuate together. Movies that have a high correlation coefficient are the movies that are most similar to each other. In our case, we shall use the Pearson correlation coefficient. This number will lie between -1 and 1. 1 indicates a positive linear correlation while -1 indicates a negative correlation. 0 indicates no linear correlation. Therefore, movies with a zero correlation are not similar at all. In order to create this data frame we use pandas `groupby` functionality. We group the dataset by the `title` column and compute its mean to obtain the average rating for each movie.

```
ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings.head()
```

Next we would like to see the number of ratings for each movie. We do this by creating a `number_of_ratings` column. This is important so that we can see the relationship between the average rating of a movie and the number of ratings the movie got. It is very possible that a 5-star movie was rated by just one person. It is therefore statistically incorrect to classify that movie has a 5-star movie. We will, therefore, need to set a threshold for the minimum number of ratings as we build the recommender system. In order to create this new column, we use pandas `groupby` utility. We group by the `title` column and then use the `count` function to calculate the number of ratings each movie got. Afterward we view the new data frame by using the `head()` function.

```
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()
ratings.head()
```

Let's now plot a Histogram using pandas plotting functionality to visualize the distribution of the ratings

```
import matplotlib.pyplot as plt
%matplotlib inline
ratings['rating'].hist(bins=50)
```

We can see that most of the movies are rated between 2.5 and 4. Next, let's visualize the `number_of_ratings` column in as similar manner.

```
ratings['number_of_ratings'].hist(bins=60)
```

From the above histogram, it is clear that most movies have few ratings. Movies with most ratings are those that are most famous.
Let's now check the relationship between the rating of a movie and the number of ratings. We do this by plotting a scatter plot using seaborn. Seaborn enables us to do this using the `jointplot()` function.

```
import seaborn as sns
sns.jointplot(x='rating', y='number_of_ratings', data=ratings)
```

From the diagram we can see that there is a positive relationship between the average rating of a movie and the number of ratings. The graph indicates that the more the ratings a movie gets the higher the average rating it gets. This is important to note especially when choosing the threshold for the number of ratings per movie.
Let's now move on swiftly and create a simple item-based recommender system. In order to do this, we need to convert our dataset into a matrix with the movie titles as the columns, the `user_id` as the index and the ratings as the values. By doing this we shall get a data frame with the columns as the movie titles and the rows as the user ids. Each column represents all the ratings of a movie by all users. The rating appears as NAN where a user didn't rate a certain movie. We shall use this matrix to compute the correlation between the ratings of a single movie and the rest of the movies in the matrix. We use pandas `pivot_table` utility to create the movie matrix.

```
movie_matrix = df.pivot_table(index='user_id', columns='title',
values='rating')
movie_matrix.head()
```

Next let's look at the most rated movies and choose two of them to work with in this simple recommender system. We use
pandas `sort_values` utility and set `ascending` to false in order to arrange the movies from the most rated. We then use the `head()` function to view the top 10.

```
ratings.sort_values('number_of_ratings', ascending=False).head(10)
```

Let's assume that a user has watched Air Force One (1997) and Contact (1997). We would like to recommend movies to this user based on this watching history. The goal is to look for movies that are similar to Contact (1997) and Air Force One (1997 which we shall recommend to this user. We can achieve this by computing the correlation between these two movies' ratings and the ratings of the rest of the movies in the dataset. The first step is to create a data frame with the ratings of these movies from our `movie_matrix`.

```
AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']
```

We now have the data frame showing the `user_id` and the rating they gave the two movies. Let's take a look at them below.

```
AFO_user_rating.head()
contact_user_rating.head()
```

In order to compute the correlation between two dataframes we use pandas `corwith` functionality. `Corrwith` computes the pairwise correlation of rows or columns of two data frames objects. Let's use this functionality to get the correlation between each movie's rating and the ratings of the Air Force One movie.

```
similar_to_air_force_one=movie_matrix.corrwith(AFO_user_rating)
```

We can see that the correlation between Air Force One movie and Till There Was You (1997) is 0.867. This indicates a very strong similarity between these two movies.

```
similar_to_air_force_one.head()
```

Let's move on and compute the correlation between Contact (1997) ratings and the rest of the movies ratings. The procedure is the same as the one used above.

```
similar_to_contact = movie_matrix.corrwith(contact_user_rating)
```

We realize from the computation that there is a very strong correlation (of 0.904) between Contact (1997) and Til There Was You (1997).

```
similar_to_contact.head()
```

As noticed earlier our matrix had very many missing values since not all the movies were rated by all the users. We therefore drop those null values and transform correlation results into data frames to make the results look more appealing.

```
corr_contact = pd.DataFrame(similar_to_contact,
columns=['Correlation'])
corr_contact.dropna(inplace=True)
corr_contact.head()
corr_AFO = pd.DataFrame(similar_to_air_force_one,
columns=['correlation'])
corr_AFO.dropna(inplace=True)
corr_AFO.head()
```

These two data frames above show us the movies that are most similar to Contact (1997) and Air Force One (1997) movies respectively. However, we have a challenge in that some of the movies have very few ratings and may end up being recommended simply because one or two people gave them a 5-star rating. We can fix this by setting a threshold for the number of ratings. From the histogram earlier we saw a sharp decline in a number of ratings from 100. We shall, therefore, set this as the threshold, however, this is a number you can play around with until you get a suitable option. In order to do this, we need to join the two data frames with the `number_of_ratings-` column in the `rating` data frame.

```
corr_AFO = corr_AFO.join(ratings['number_of_ratings'])
corr_contact = corr_contact.join(ratings['number_of_ratings'])
corr_AFO .head()
corr_contact.head()
```

We shall now obtain the movies that are most similar to Air Force One (1997) by limiting them to movies that have at least 100 reviews. We then sort them by the `correlation` column and view the first 10.

```
corr_AFO[corr_AFO['number_of_ratings'] >
100].sort_values(by='correlation', ascending=False).head(10)
```

We notice that Air Force One (1997) has a perfect correlation with itself, which is not surprising. The next most similar movie to Air Force One (1997) is Hunt for Red October, The (1990) with a correlation of 0.554. Clearly, by changing the threshold for the number of reviews, we get different results from the previous way of doing it. Limiting the number of rating gives us better results and we can confidently recommend the above movies to someone who has watched Air Force One (1997).

Now let's do the same for Contact (1997) movie and see the movies that are most correlated to it.

```
corr_contact[corr_contact['number_of_ratings'] >
100].sort_values(by='Correlation', ascending=False).head(10)
```

Once again, we get different results. The most similar movie to Contact (1997) is Philadelphia (1993) with a correlation coefficient of 0.446 with 137 ratings. So, if somebody liked Contact (1997) we can recommend the above movies to them.

Obviously, this is a very simple way of building a recommender system and is nowhere close to industry standards.

How to improve the recommendation system

This system can be improved by building a Memory-Based Collaborative Filtering based system. In this case, we'd divide the data into a training set and a test set. We'd then use techniques such as cosine similarity to compute the similarity between the movies. An alternative is to build a Model-based Collaborative Filtering system. This is based on matrix factorization. Matrix factorization is good at dealing with scalability and sparsity than the former. You can then evaluate your model using techniques such as Root Mean Squared Error (RMSE).