Capstone Project Rajesh Kumar Bandaru
Machine Learning Nano Degree June 28th, 2020

Report

1. Introduction:

Recommendation Systems are an important class of Machine Learning that offers relevant suggestions to users based on their previous activity. As it has a direct impact on the bottom line, Recommendation Systems are being used by most of the major corporations in the world. Online stores such as Amazon, Ebay use it to predict the purchase activities of their customers.

Netflix uses the Recommendation Systems to predict movie suggestions for viewers based on their preferences and history.

In this project, I will be building a hybrid movie recommendation system by studying the different recommendation systems.

I was fascinated by the way how the recommendation in the Netflix and Amazon work and wanted to try and work on it

2. Problem Statement:

The problems that are solved in here are as follows:

1. If user's movie watching history is given, which movie does the user is interested in watching?
2. If there is more than one movie to suggest (in most of the cases), what kind of ranking should be used to determine the order?

A hybrid recommendation system is built by combining item-based Collaborative Filtering and Content-based Recommender System to solve the above problems.

3. Metrics:

i. Personalization

It is a great way to assess if a model recommends many of the same items to different users. It is a dissimilarity between user's list of recommendations.

1. The predictions for a set of users are transformed into a binary indicator variable matrix.
2. The cosine similarity matrix is calculated over all the user's recommendation vectors.
3. The average of the upper triangle of the cosine matrix is calculated.
4. Finally, the personalization value is 1 – cosine similarity.

The highest personalization score suggests user's recommendations are different giving a personalized experience to each user.

ii. Intra-list similarity

It is a way to assess if a model recommends many of the same feature items to the users (user receiving only comedy movie recommendations). This calculation uses features of the recommended items to calculate the similarity.

1. The predictions for a set of users are transformed into a binary indicator matrix of a feature of a variable.
2. Intra-list similarity is calculated for each user and averaged across all the users to get an estimate of intra-list similarity for the model.

The lowest intra-list similarity indicates that the model is suggesting different feature recommendations to the user.

4. Dataset:

The datasets for this project are taken from Movie Lens containing information about the movies and user ratings for them. The secondary dataset is from Kaggle containing the plot of the movie.

Analysis

Reading the datasets required for analysis.

1. Downloading the data:

```
#the csv file containing the users data
users = 'Datasets/ratings.csv'
```

Fig. Reading the ratings csv file

```
#csv file containing the movies data
movies = 'Datasets/movies.csv'
```

Fig. Reading the movies csv file

```
#loading the movies_dataset from kaggle to get the content of the movies
content_df = pd.read_csv('Datasets_kaggle/movies_metadata.csv')
```

Fig. Reading the movies metadata file from Kaggle

2. Exploring the data:

Exploring each Data Frame.

i. Users Data Frame (users_df)

```
users_df.columns
```

```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
```

The users data frame contains userId, movieId, rating and timestamp columns. We can get rid of the timestamp column as it doesn't help in the analysis.

Description of users_df data frame

```
#describing the data
users_df.describe()
```

	userId	movieId	rating
count	100836.000000	100836.000000	100836.000000
mean	326.127564	19435.295718	3.501557
std	182.618491	35530.987199	1.042529
min	1.000000	1.000000	0.500000
25%	177.000000	1199.000000	3.000000
50%	325.000000	2991.000000	3.500000
75%	477.000000	8122.000000	4.000000
max	610.000000	193609.000000	5.000000

UserId, movieId, rating columns shouldn't contain any negative values. From the description of the data above, minimum and maximum values are not negative. So, they are no invalid values.

The information of the users_df data frame is as follows

```
users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   userId   100836 non-null  int64  
 1   movieId  100836 non-null  int64  
 2   rating   100836 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 2.3 MB
```

The first five rows of the users_df data frame.

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

ii. Movies Data Frame (movies_df)

```
movies_df.columns
```

```
Index(['movieId', 'title', 'genres'], dtype='object')
```

The movies data frame contains movieId, title, genres columns. The genres column is used for evaluation metrics.

The information of the movies_df is as follows

```
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

A look at the first five rows of the data frame

```
movies_df.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

The year mention in the title column is not needed for analysis as it doesn't help in any way. Year part in the title can be removed.

The format of data in the genre column is not analysis friendly. The genre column can be converted into a list.

After the processing of data, the movies_df data frame is as follows

```
movies_df.head()
```

	movieId	title	genres
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]
1	2	Jumanji	[Adventure, Children, Fantasy]
2	3	Grumpier Old Men	[Comedy, Romance]
3	4	Waiting to Exhale	[Comedy, Drama, Romance]
4	5	Father of the Bride Part II	[Comedy]

iii. Movie metadata (content_df)

```
content_df.columns
```

```
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count'],
      dtype='object')
```

Movie metadata data frame contains so many unnecessary columns. The only column needed is the ‘overview’ column which is the plot of the movie.

The information of the content_df data frame is as follows

```
content_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   title     45460 non-null  object
 1   overview  44512 non-null  object
dtypes: object(2)
memory usage: 710.5+ KB
```

A look at the first five rows of the data frame

```
content_df.head()
```

	title	overview
0	Toy Story	Led by Woody, Andy's toys live happily in his ...
1	Jumanji	When siblings Judy and Peter discover an encha...
2	Grumpier Old Men	A family wedding reignites the ancient feud be...
3	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...
4	Father of the Bride Part II	Just when George Banks has recovered from his ...

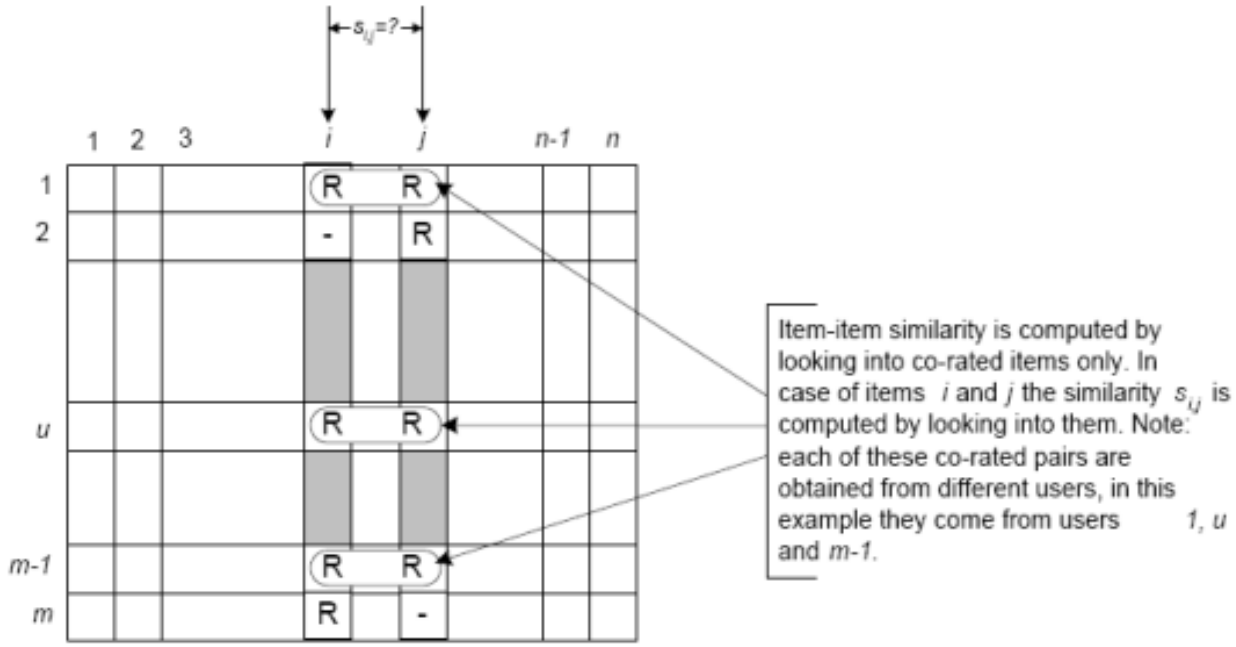
3. Algorithms and Techniques

For this project, we focused on two main algorithms Item-based collaborative filtering and Content-based Filtering.

Item-based Collaborative Filtering:

Item-based collaborative filtering is a model-based algorithm for making recommendations. In the algorithm, the similarities between different items in the dataset are calculated by using one of a number of similarity measures, and then these similarity values are used to predict ratings for user-item pairs not present in the dataset.

The similarity values between items are measured by observing **all the users who have rated both the items**. As shown in the diagram below, the similarity between two items is dependent upon the ratings given to the items by users who have rated both of them:



There are different mathematical formulae that can be used to calculate the similarity of the items. The one used here is the Cosine-based similarity also known as vector-based similarity.

Cosine-based Similarity:

This takes two items and their ratings as vectors and defines the similarity between them as the angle between these two vectors.

Content-based filtering:

Content-based filtering uses item features to recommend other items similar to the user's likes. The feature used in this project is the plot of the movie. This type of recommender compares the descriptions of different movies and provides recommendations that have most similar plot descriptions. There will be less diversity in the recommendations. Advantages of Content Based approach is that data of other users are not required and the recommender engine can recommend new items which are not rated currently, but the recommender algorithm doesn't recommend the items outside the category of items the user has rated.

4. **Benchmark Model:**

The benchmark model here is the knowledge-based recommender system, content-based recommender system and a simple recommender system. These models are implemented and their metrics are compared to the Hybrid Model built.

Knowledge-based recommender: Knowledge-based recommender for movie recommendation uses the input data from the users such as Genre, runtime, director to recommend movies to the users.

Content-based recommender: Content-based recommender for movie recommendation can be done in two different ways.

1. Recommendations can be given by analyzing the plot of the movie. **(plot-based)**
2. Recommendations can be given by analyzing the metadata (cast, crew, genre) of the movie. **(meta-data based)**

For plot-based filtering, the overview (plot) column of the df data frame is analyzed by natural language processing to recommend movies to the users.

For meta-data based, the 'genre' column of df data frame, the 'cast' and 'crew' columns of credits data frame and the 'keywords' column of keywords data frame are analyzed.

Simple recommender system: A metric is chosen to rate the movies on. Decide on the prerequisites for the movie to be featured on the chart. Calculate the score for every movie that satisfies the conditions. The movies are listed according to the decreasing order of their scores.

Methodology

1. Data Pre-processing

Merging the movies_df and content_df data frames on title.

```
movies_df = pd.merge(movies_df, content_df, left_on='title', right_on='title')
```

After merging the data frames there are lot of duplicate values in the title column of the merged data frame. By dropping the duplicates and the NaN values the information of the data frame is as follows.

```
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6316 entries, 0 to 8069
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   movieId   6316 non-null   int64 
 1   title     6316 non-null   object
 2   overview  6316 non-null   object
 3   genres    6316 non-null   object
dtypes: int64(1), object(3)
memory usage: 246.7+ KB
```

A total of 6316 movies are present in the data frame.

A look at the first five rows of final movies_df data frame

```
movies_df.head()
```

	movieId	title	overview	genres
0	1	Toy Story	Led by Woody, Andy's toys live happily in his ...	[Adventure, Animation, Children, Comedy, Fantasy]
1	2	Jumanji	When siblings Judy and Peter discover an encha...	[Adventure, Children, Fantasy]
2	3	Grumpier Old Men	A family wedding reignites the ancient feud be...	[Comedy, Romance]
3	4	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...	[Comedy, Drama, Romance]
4	5	Father of the Bride Part II	Just when George Banks has recovered from his ...	[Comedy]

The users_df data frame has more movies than in the movies_df data frame. So to make proper suggestions using collaborative filtering the movies in those two data frames are to be equal. Merging movies_df data frame and users_df data frame.

```
users_df = users_df.merge(movies_df,indicator=True,how='outer')
```

The number of users in the data frame are

```
users_df.userId.nunique()
```

```
610
```

For Content-based filtering, the overview column is used to construct a tf-idf matrix.

```
#Define a TF-IDF Vectorizer Object. Remove all english stopwords
tfidf = TfidfVectorizer(stop_words='english')

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

#Construct the required TF-IDF matrix by applying the fit_transform method on the overview feature
tfidf_matrix = tfidf.fit_transform(movies_df['overview'])
```

2. Implementation

Item-based Collaborative Filtering using the users_df data frame.

```
item_matrix = users_df.pivot_table(values='rating', index='movieId', columns='userId')
```

The matrix is constructed with userId in columns, movieId as rows and rating as values.

A look at the first five columns of the item matrix

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN

The NaN values in the data frame is filled with the mean of the values in the row.

```
item_matrix_filled = item_matrix.fillna(item_matrix.mean(axis=0))
```

The similarity of the movies is calculated using the cosine similarity rule of the matrix.

```
cos_sim = cosine_similarity(item_matrix_filled, item_matrix_filled)
```

The first five movies of the data frame are as follows

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573
movieId															
1	1.000000	0.988961	0.989857	0.991037	0.988225	0.988483	0.988566	0.991272	0.990515	0.986610	...	0.991758	0.991736	0.991749	0.991749
2	0.988961	1.000000	0.994237	0.995461	0.994383	0.992315	0.993489	0.996021	0.995607	0.990762	...	0.996427	0.996399	0.996425	0.996425
3	0.989857	0.994237	1.000000	0.996728	0.995412	0.993572	0.994708	0.996766	0.996639	0.992125	...	0.997468	0.997440	0.997467	0.997467
4	0.991037	0.995461	0.996728	1.000000	0.996857	0.995429	0.996287	0.998825	0.998565	0.995151	...	0.999357	0.999329	0.999355	0.999355
5	0.988225	0.994383	0.995412	0.996857	1.000000	0.993387	0.995141	0.996899	0.996935	0.993022	...	0.997469	0.997440	0.997469	0.997469

A function is written with user id as input. The movies watched by the user is taken from the item_matrix. The movies which are similar to the movies watched by the user is taken from the similarity matrix. The first 15 movies are selected from the similar movie list on the decreasing order of the similarity. The indices are of the top 15 movies are selected and the titles are extracted from the movies_df data frame.

```
def user_watched(user_id):
    item = list(item_matrix[item_matrix[user_id]>0].index)
    df = pd.DataFrame()
    for i in item:
        df = df.append(item_sim_matrix[i][:])
    df = df.nlargest(15,user_id)
    ind = list(df.index)
    watched_df = pd.DataFrame()
    for i in ind:
        watched_df = watched_df.append(movies_df[movies_df['movieId'] == i])
    return watched_df[['movieId','title']]
```

Content-based filtering uses the movies_df data frame using the overview(plot) column.

From the data preprocessing section, tf-idf matrix is taken and a similarity matrix is constructed.

```
cosine_sim1 = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

A function is written with the movies data frame, title of the movie, plot similarity matrix and indices of the movies as input. The index of title of the movie is taken. The similarity scores of all the movies is extracted from the tf-idf similarity matrix. The scores are sorted in decreasing order. The top 10 movies are selected from the list. It returns a data frame containing the title and genre of the movies.

```
def content_recommender(df, title, cosine_sim, indices):

    idx = indices[title]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda df: df[1], reverse=True)

    sim_scores = sim_scores[1:11]

    movie_indices = [i[0] for i in sim_scores]

    return pd.DataFrame(df[['title','genres']].iloc[movie_indices])
```

A function with the titles as input is written. The content_recommender function is applied to every title in the list and all the suggested titles are stored in a data frame and returned.

```
def result(titles):
    final_df = pd.DataFrame()
    for i in titles:
        final_df = final_df.append(content_recommender(movies_df, i, cosine_sim1, indices))
    return final_df
```

A function implementing both the item-based collaborative filtering and content-based filtering is written with user id as input and the top 5 movies are suggested to the user

```
def recommend_movies(user_id):

    sim_mov = user_watched(user_id)

    titles = list(sim_mov['title'])

    movies = result(titles)

    return movies.head()
```

Metrics used to evaluate the model are personalization and intra list similarity.

A function to implement the Personalization metric from the predicted movies.

```
def personalization(predicted):

    users_rec = pd.DataFrame(predicted)

    users_binary_rec = pd.get_dummies(users_rec,columns=users_rec.columns)

    similarity = cosine_similarity(X=users_binary_rec, dense_output=False)

    upper_right = np.triu_indices(similarity.shape[0], k=1)

    personalization = np.mean(similarity[upper_right])

    return 1-personalization
```

A function to implement the intra list similarity metric.

```
def intra_list_similarity(predicted, user):

    user['genre'] = get_genre(list(user['genres']))
    feature_df = pd.get_dummies(user['genre'],columns=user.index)

    recs_content = feature_df.loc[predicted]

    similarity = cosine_similarity(X=recs_content.values, dense_output=False)

    upper_right = np.triu_indices(similarity.shape[0], k=1)

    ils_single_user = np.mean(similarity[upper_right])

    return ils_single_user
```

Results

1. Model Evaluation and Validation

Personalization Scores:

Users	*Score*
User 1, user 5 and user 610	1.0
User_6 and user_42	1.0
user_1, user_6, user_42, user_610	1.0

Intra list similarity scores:

User	*Score*
User_1	0.14
User_5	0.18
User_42	0.20

2. Justification

Personalization score of the recommender system should be in between 0 to 1. '0' indicating the lowest and '1' indicating the highest. For all the tested cases, the personalization score is '1' which indicated the user is getting a personalized experience in the suggestions.

Intra list similarity score of a recommender system should also be in between 0 to 1. '0' indicates a good score and '1' indicates a bad score. For all the tested cases, the intra list similarity scores are close to '0' indicating the recommender system is not biased to a single genre as genre is used as a feature to determine the score.