Movie Recommendation Systems

Kushal Kejriwal
Electrical Engineering
IIT Bombay
190110040

Raghav Singhal
Electrical Engineering
IIT Bombay
19D070049

Rohit Vartak
Electical Engineering
IIT Bombay
190010058

***Abstract*—We use 2 different types of filtering -content and collaborative type to recommend users different product/services they may like.**

***Index Terms*—content, collaborative, similarity, recommendation**

I. Introduction

Recommender systems can be used to predict users' preferences and likings based on past behaviour and related data. This makes such systems extremely powerful, and are widely used by various platforms such as Amazon, Netflix and so on, to drive sales and improve profits substantially.

II. Datasets we have used

- *Credits.csv* : Has information regarding cast, crew etc
- *Movie_metadata.csv*:Contains information regarding Revenue, vote-average and vote-counts etc
- *Keywords.csv*: Dataset contains the genres of each movie.
- *ratings_csv*: Contains the userId, movieId, rating and timestamp for over 600+ users and 9000+ movies and has over 10,000+ entries. Used in collaborative filtering

III. Cleaning the Datsets

- The uncleaned data had stringed dictionaries, many numerical columns as string data etc Working with this data was impossible and so it had to be cleaned
- We used different classes to achieve this goal. The 'literal_eval' of the ast class was used to convert stringed dictionaries to python dictionaries. String formatting also came to rescue while handling and cleaning string data. For egs str.lower(), str.split() etc
- We convert all string data to lower case by using str.lower() function of python and strip each function str.strip() and delete any blank spaces within the string str.replace(" ","")
- Collaborative Filtering - After doing Exploratory Data Analysis we realised that many of the movies have been rated only once and many of the users had rated quite a few movies. We removed these entries so that we get better results.

IV. Types of Filtering

1) **Content Based**
    - In this method, the similarity between different movies is used to give recommendations.
    - The "similarity" between different movies is computed using "Cosine Similarity".
    - The personal taste of an individual is thereby not encapsulated in this method.
2) **Collaborative Filtering**
    - Content Based Filtering did not capture the personal taste of an individual. Thus, came the need for Collaborative Type Filtering.
    - In this method, each user effectively becomes a datapoint. Therefore the system predicts which movie to suggest to the desires user, depending on the preferences of other use
    - **Types of Collaborative Filtering**
        a) User-User Based: Determine similarity between users through the movies which they have already seen and then recommend movies to a user which are highly rated by the similar users.
        b) Item-Item Based: Determine similarity between each item pair through the user ratings and then recommend similar movies which are liked by the user in the past.

V. Building Content Based filtering System

- We first created a metadata dump for each movie, consisting of the genres, cast, director and keywords. To give more weight to the director, each director's name was repeated thrice. The top 3 cast members were chosen for each movie, and the keywords that occurred at least 10 times throughout the dataset were used.
- We used only a fraction of the dataset (1/5th) owing to our limited computational resources. In addition to this, only the movies lying in the top 25 percentile according to the weighted IMDB ratings were kept, to recommend higher quality movies.
- CountVectorizer was used to tokenize this metadata hump into a vocabulary of known words, and encode it. The cosine_similarity function was used from sklearn to compute the cosine similarities between movies.
- To make recommendations, we defined a function which essentially sorts the movies which are most similar to the required movie (on the basis on cosine similarity), and then suggests the best n depending on the user.

VI. Building Collaborative Based filtering System

A. From Scratch

- We implemented a basic collaborative filtering algorithm completely through scratch (using only numpy). We used the basic user-based KNN algorithm along with the pearson baseline similarity.
- The basic KNN algorithm simply takes the weighted average of the ratings given to a movie by the similar users (we chose the top 50 neighbours (users) having the highest similarity coefficient)
- We are going to use user-user collaborative filtering for this purpose. Let us now define the pearson correlation which will calculate the similarity between two users. After calculating the similarity, we will recommend movies to the user which have been rated highly by the other users having similar interests.
  The pearsons correlation is given by
  $$\frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum(x_i-\bar{x})^2\sum(y_i-\bar{y})^2}}$$
  where $\bar{x}$ is the mean of all ratings given by user 1 and $\bar{y}$ is the mean of all ratings given by user 2 and we sum over all the ratings x_i and y_i which have been rated by **both the users**
- This gave us a good rmse test score of around 0.85.

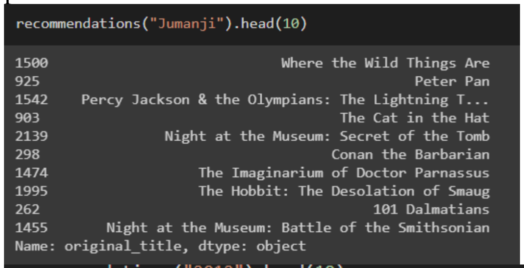
B. Using Python's Surprise library

- We then used the Surprise Python Library [1] which is specifically built for designing recommendation systems, did a cross validation analysis over some of the popular algorithms such as Singular Value Decomposition, KNN with Z score, KNN Basic, Baseline Only and KNN Baseline. After analysing these algorithms, we selected Baseline Only and KNN with means and did a grid search on these algorithms to tune the hyperparameters. We also compared user - user based and item-item based in the KNN with means algorithm.
- **Baseline Estimate**
  - Some movies may recieve higher ratings than other and some users might rate higher than others. It is customary to adjust the data by accounting for these effects, which we encapsulate within the **baseline estimates**. Let μ be the overall average rating. A baseline estimate for an unknown rating r_{ui} is
    $$b_{ui}=\mu+b_u+b_i$$
    where b_u accounts for user effects and b_i accounts for item(movie) effects. [2]
  - This now becomes a standard regression problem and to solve we minimise the least square error (with regularisation) given by
    $$min\sum_{u,i}(r_{ui}-(\mu+b_u+b_i))^2+\lambda_1(\sum_u b_u^2+\sum_i b_i^2)$$
    This is implemented using grid search where we tune the hyperparameters to yield the best result
- **KNN with means**
  - Now we will look at a different approach, which was similar to what we implemented from scratch, only with a few minor differences. There are mainly 2 types of sub categories in this algorithm:
    1) **User - User based** - Estimate unknown ratings based on recorded ratings of like-minded users
    2) **Item - Item based** - A rating is estimated using known ratings made by the same user on similar items.
  - We will focus on the second one (the reasons are discussed later)
    The rating $\hat{r_{ui}}$ which is the predicted rating given by user u to item i is given by
    $$\hat{r_{ui}}=\mu_i+\frac{\sum_j sim(i,j)\cdot(r_{uj}-\mu_j)}{\sum_j sim(i,j)}$$
    where j are the items belonging to the neighbourhood of i (ie having the highest similarity) and μ_i and μ_j are the mean ratings received for those items.
  - The advantage of item-item based filtering is that Item-item neighborhood models can provide updated recommendations immediately after users enter new ratings. This includes handling new users as soon as they provide feedback to the system, without needing to retrain the model and estimate new parameters. (we assume that the item - item similarity is stable over time)
  - Also, we note that we implemented the basic KNN algorithm from scratch which is slightly different from the KNN with means and as the name suggests, a basic version. Here, we dont bother about the mean average ratings given to an item (movie). We simply take the weighted average of the ratings given to the movie by similar users (user - user based) or the ratings given different similar movies by the same user (item - item based). We had implemented user - user based collaborative filtering. Mathematically, it is given by
    $$\hat{r_{ui}}=\frac{\sum_v sim(u,v)\cdot r_{vi}}{\sum_v sim(u,v)}$$
    where v are the similar users to user u
- This gave us a rmse test score of around 0.7 - 0.75. [3]

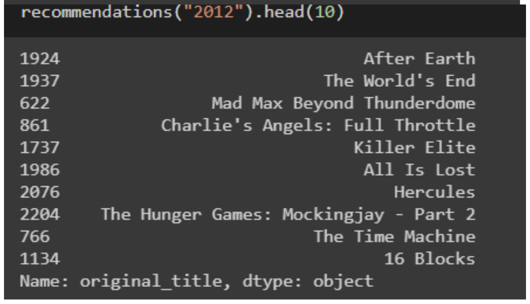
VII. Results

- **Content Based Filtering**
  We summarise the results in form of

preditions for 2 movie titles as follows.



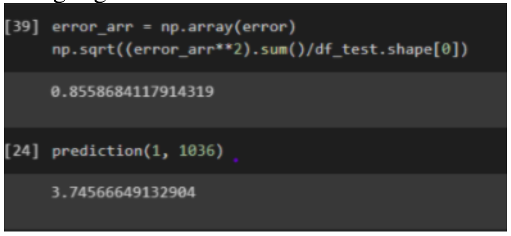
```
recommendations("Jumanji").head(10)

1500                             Where the Wild Things Are
925                                               Peter Pan
1542    Percy Jackson & the Olympians: The Lightning T...
903                                     The Cat in the Hat
2139             Night at the Museum: Secret of the Tomb
298                                   Conan the Barbarian
1474                  The Imaginarium of Doctor Parnassus
1995                  The Hobbit: The Desolation of Smaug
262                                         101 Dalmatians
1455       Night at the Museum: Battle of the Smithsonian
Name: original_title, dtype: object
```



```
recommendations("2012").head(10)

1924                              After Earth
1937                          The World's End
622              Mad Max Beyond Thunderdome
861            Charlie's Angels: Full Throttle
1737                              Killer Elite
1986                              All Is Lost
2076                                 Hercules
2204    The Hunger Games: Mockingjay - Part 2
766                          The Time Machine
1134                                16 Blocks
Name: original_title, dtype: object
```

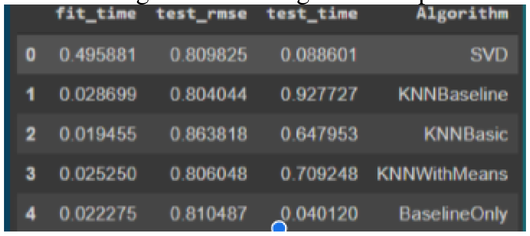
- **Collaborative Based Filtering**

    1) From the KNN basic algorithm we got a respectable rmse score of 0.85. We also defined a function which will predict the rating given a user id and a movie id.



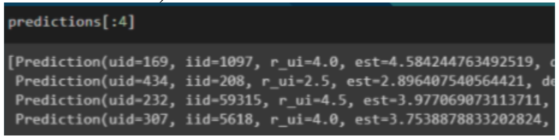
```
[39] error_arr = np.array(error)
     np.sqrt((error_arr**2).sum()/df_test.shape[0])

     0.8558684117914319

[24] prediction(1, 1036)

     3.74566649132904
```

    2) Following are the cross validation scores for different algorithms using the surprise library



	fit_time	test_rmse	test_time	Algorithm
0	0.495881	0.809825	0.088601	SVD
1	0.028699	0.804044	0.927727	KNNBaseline
2	0.019455	0.863818	0.647953	KNNBasic
3	0.025250	0.806048	0.709248	KNNWithMeans
4	0.022275	0.810487	0.040120	BaselineOnly

    3) Using the surprise library, we got an rmse score of 0.79 in baseline only estimate and 0.75 using KNN with means algorithm (item - item based) and 0.77 in user - user based



```
predictions[:4]

[Prediction(uid=169, iid=1097, r_ui=4.0, est=4.584244763492519,
 Prediction(uid=434, iid=208, r_ui=2.5, est=2.896407540564421, de
 Prediction(uid=232, iid=59315, r_ui=4.5, est=3.977069073113711,
 Prediction(uid=307, iid=5618, r_ui=4.0, est=3.7538878833202824,
```

References

[1] "https://surprise.readthedocs.io/en/stable/index.html."

[2] Y. Koren, "Collaborative filtering with temporal dynamics," *https://courses.ischool.berkeley.edu/i290-dm/s11/SECURE/p89-koren.pdf*.

[3] "https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/."