**First Capstone Project**

**Compare Different Methods and Similarity Functions to Build a Minimal Movie Recommendation Engine**

1. **Problem Description**

There’ve been a huge amount of movies and movie lovers since the first movie was in theaters. But even the professional movie critics can’t go through all the movies. So it’s a good idea to recommend movies by some algorithms according to audience's interest. So goal of this project is to build a movie recommendation system according to movie datasets.

And some entertainment companies such as Netflix and movie review website such as IMDB also want to build such a system to find out proper movies for their users and target users for their movies. In addition, recommendation system can not only offer movies suggestion, but also be used for content and product recommendation for social websites and online stores, such as Facebook, Instagram, Amazon, etc.

In this project, a movie dataset from https://grouplens.org/datasets/movielens/ will be utilized for analysis. And it contains several sub-datasets, including movie ID, user ID, movie name, genre, tag, movie links and ratings. Totally there are 100,000 ratings and 1,300 tag applications applied to 9,000 movies by 700 users. As it says on the web, it’s not a benchmark dataset and may be updated over time, so it’s only used for education and development. And this one is last updated in 10/2016.

1. **Methods**

There're two well-known recommendation methods:

Content-based filtering: recommend based on the user's own rating history on one movie.

-- Content-based filtering methods are based on a description of the item and a profile of the user’s preferences.

(1)

--- : rating on movie i by user u,

--- i’ : one of other movies,

--- sim(): similarity value, which is scaled by minmaxscaler.

Collaborative filtering: recommend based on other user's rating history on one movie.

-- Collaborative filtering methods are based on collecting and analyzing a large amount of information on users’ behaviors, activities or preferences and predicting what users will like based on their similarity to other users.

(2)

--- u’ : one of other users,

1. **Data Wrangling and Exploratory**

There are 4 sub-datasets, and they are merged together at first. And there exist 100006 ratings on 9066 movies by 671 users. Then timestamp objects are transferred into date time object in order for visualization. Next, year of movies are extracted as a new column in movielens dataset.

In the interest of running time, a smaller dataset with 10000 ratings is generated randomly from the original one. Year of movies are explored, so that we know the top five years with most movies and the number of movies: [(1993, 451), (1996, 494), (1999, 521), (1994, 546), (1995, 665)]. And Figure 1 is the distribution of year of movies. Genres of movies are also explored, so we know there are 19 genres totally and many moves have multiple genres. And the 19 genres are: {'Fantasy', 'Crime', 'Western', 'Animation', 'War', 'Documentary', 'Thriller', 'Adventure', 'Film-Noir', 'Children', 'Action', 'Romance', 'Horror', 'Mystery', 'IMAX', 'Musical', 'Sci-Fi', 'Comedy', 'Drama'}.

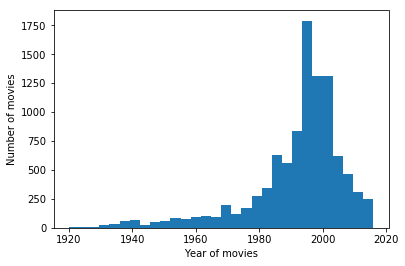


Figure 1 Distrubition of year of movies

Additionally, inferential statistics are applied to answer these two questions:

1. If mean of one user's rating is 3.5, does this user tend to overestimate or underestimate a movie?

Null hypothesis for this question: This user neither tends to overestimate, nor tends to underestimates movies. The mean of users’ rating follows the normal Gaussian distribution, so z test is a proper choice. Finally, z score is: -5.02151248569, and p value is: 5.12661706198e-07. So a user tends to underestimate a movie, if mean rating of the user is 3.5.

1. If one movie's rating is 3.5, is this movie beyond the average or below the average?

Null hypothesis: This movie's rating, 3.5, is just average. The mean of movies’ rating follows the normal Gaussian distribution, so z test is also a proper choice. Finally, z score is: 7.42267829289, and p value: 1.14797060746e-13. According to the z score and p value, if a movie's rating is 3.5, it's significantly higher rated.

**4. Modeling and Evaluation**

At first, users who only rate once are removed from the set, because we want to have at least 2 ratings per user in this set. And the movielens dataset are split into train and test set, and test set size is 33%. Though there are several performance criterions for evaluation, such as: RMSE, Precision /Recall /F-scores, ROC curves, Cost curves, RMSE is selected as the criterion in this project. (add one example)

There exist a few custom similarity functions, such as Euclidean similarity function, Pearson r function, cosine function and Jaccard function. And all these four functions are incorporated with both content-based filtering and collaborative filtering. One example for each filtering is shown below.

* Euclidean 'similarity':

sim(x, y) = (3)

* Cosine similarity':

sim(x, y) = (4)

* Pearson correlation:

sim(x, y) = (5)

* Jaccard similarity:

sim(x, y) = (6)

Content-based filtering using Euclidean sim function:

In this case, missing values are replaced by the mean ratings of movies. And Euclidean similarity between movies’ ratings are scaled by sklearn.MinMaxScaler, and it is used as the weights of movies’ ratings, so the unknown ratings are predicted. And root mean square error is 0.819.

Collaborative filtering using Pearson sim function:

In this example, missing values are replaced by users’ mean ratings. And Pearson correlation between movies’ ratings are scaled by sklearn.MinMaxScaler, and it is used as the weights of users’ ratings, so the unknown ratings are predicted. And root mean square error is 1.083.

Table 1.  RMSE of Content-based and collaborative filtering using sim functions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RMSE | EUCLIDEAN | PEARSON | COSINE | JACCARD |
| CONTENT-BASED | 0.819 | 0.924 | 0.985 | 0.975 |
| COLLABORATIVE | 1.143 | 1.083 | 1.154 | 1.140 |

1. **Conclusion**

In this project, content-based filtering and collaborative filtering are compared, while both of these two methods are incorporated with similarity of items. And according to the root mean square errors in table 1, content-based filtering has a better performance. Additionally, a few similarity functions are implemented in this project, and Euclidean similarity function in content-based filtering provides the best performance.

1. **Reference**
2. Hands-on with PyData: How to Build a Minimal Recommendation Engine, <http://unatainc.github.io/pycon2015/>
3. Andreas C. Muller and Sarah Guido, Introduction to Machine Learning with Python, 2017,06,09