## HW4: Movie Recommender Systems Report

Date: 11/4/2020

Binhui Xu(G01172196)

Username on miner2: WhitneyXu

LeadBoard Score: 0.91

## Approach:

* **Read data from .dat file and preprocessing data for correlative filtering:** I *used* pd.read\_table() read all data from .dat file, and all data become dataframe directly. For train data(rating matrix), first convert it to a pivot table which consists of rows as movieID and columns as userID. For collaborative filtering we'll need to create a pivot table of users on one axis and tv show names along the other. The pivot table will help to define the similarity between users and movies to better predict who will like what. In the step of normalization, I subtracted the mean from each rating to standardize. All users with only one rating or who had rated everything the same will be dropped. Then convert the data to a sparse matrix format to. be read by the cosine function.

Functions: read\_data(); preprocess()

* **Read data from .data file and preprocessing data for content based:**loaded and merged the actor, director, genre and tag data into a main dataframe and dropped the redundant features, which kept ‘actorName’, ‘directorName’, ‘genres’, ‘tags’and ‘usertags’ five features. Next, convert all those string instances into lowercase and strip all the spaces between them. Finally, joined all the required columns by space. And this step created an output that will be fed the word vector model.

Functions: get\_movie\_details(); clean\_data(); merge\_details()

* **Compute Term Frequency-Inverse Document Frequency (TF-IDF):** Before build the recommender model, used [sklearn.feature\_extraction.text.TfidfVectorizer](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html). get a matrix where each column represents a word in the movie details, and each row represents a movie.

Function: tfidf()

* **Calculate similarity between each movies or users**: I got the similarity by calculate the cosine similarity and pearson correlation. And I used sklearn.metrics.pairwise.cosine\_similarity get the cosine distance, used scipy.stats.pearsonr computed the pearson correlation.

Function: get\_similarity()

* **Recommender:**
* *User-based collaborative filtering:* construct a dataframe (user\_sim\_df) consisted with the value of similarity between all users. For each user-movie pair in test data, get its userid and movieid. Find the most k similarity users for this userid, and record their rating and weight(value of similarity) to this movieid, and use them to make prediction. In the function of prediction, initialize the predict value to 3.0, calculated the weighted average of similar users to determine a potential rating for an input user and movie.
* *Content-based recommender: basically same with method above.*

Functions: collaborative(); content\_based(); cf\_predictor(); cb\_predictor()

* **Cross\_validation:** use cross validation to find the best k. I used the sklearn.model\_selection.KFold to split data in train/test sets. Split dataset into 10 consecutive folds. Each fold is then used once as a validation while the k - 1 remaining folds form the training set. And compute the RMSE of the recommender model.

Functions: cross\_validation(); get\_rmse()

Table 1 shows the result of k to be 25, 455, and 1655:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 25 | 455 | 1655 |
| Correlative filtering | 0.9837 | 0.9117 | 0.9154 |
| Content-based | 1.1974 | 1.0251 | 0.9936 |

table 1

## Conclusion:

Collaborative filtering requires rating from other users to find similarities between users and then provide recommendations. In contrast, the content-based approach only needs to analyze the item and individual user’s personal data to make recommendations.  Collaborative filtering provides recommendations based on other unknown users who have the same taste as a given user, while content-based filtering items are recommended on a feature-level basis. In contrast to collaborative filtering, new items can be recommended before a large number of users rate them by content-based method. If the content does not contain enough information to accurately distinguish the items, then the recommendation itself is likely to be imprecise. In addition, content-based filtering provides only limited novelty because it must match the functionality of a user's profile with the items available. In the case of item-based filtering, only item profiles are created and users are advised of items similar to the ones they rate or search for, rather than their past history.

## Reference:

1. <https://www.kaggle.com/ajmichelutti/collaborative-filtering-on-anime-data>
2. <https://www.datacamp.com/community/tutorials/recommender-systems-python>
3. <https://realpython.com/build-recommendation-engine-collaborative-filtering/>
4. <https://heartbeat.fritz.ai/recommender-systems-with-python-part-i-content-based-filtering-5df4940bd831>