A movie recommendation system

Data Mining Final Project

Abstract

For this paper, we use two different approaches of the CF to make a movie recommendation engine. We compare the performance of a user oriented and an item oriented CF system. We use a data set from GroupLens Research provided by MovieLens. This dataset includes movies and user ratings. We use python to build the two different recommendation algorithms. We found that for both UBCF and IBCF, increasing the number of neighbors does not improve the performance of the recommendation engine. The UBCF algorithm seems to perform better when it comes to predicting the user rating for a movie.

LX16044 Ruben Spruit  
21620012 Cui Mengyan 崔梦焱  
21620018 Zhao Yeyu 赵叶宇

# Table of Contents

Table of Contents 1

1. Business understanding 2

2. Data understanding 4

2.1 Movies 4

2.2 Ratings 5

2.3 Descriptive statistics about movie data set 5

2.4 Descriptive statistics on the rating data set 7

3. Modeling the Recommendation System 11

3.1 Data preparation 11

3.2 Modeling the Recommendation System 11

3.2.1 Item-based collaborative filtering 11

3.2.2 User-based collaborative filtering 13

4. Model Evaluation 14

5. Summary 15

Appendix 16

# Business understanding

Modern consumers are inundated with choices. Online retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with most appropriate products is not trivial, yet it is a key in enhancing user satisfaction and loyalty. This emphasizes the prominence of *recommender systems*, which provide personalized recommendations for products that suit a user’s taste. Internet leaders like Amazon, Google, Netflix, TiVo and Yahoo are increasingly adopting such recommenders.

Recommender systems are often based on *Collaborative Filtering* (CF), which relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. Notably, collaborative filtering techniques require no domain knowledge and avoid the need for extensive data collection. In addition, relying directly on user behavior allows uncovering complex and unexpected patterns that would be difficult or impossible to profile using known data attributes. Consequently, collaborative filtering attracted much of attention in the past decade, resulting in significant progress and being adopted by some successful commercial systems, including Amazon, TiVo and Netflix.

To establish recommendations, collaborative filtering systems need to compare fundamentally different objects: items against users. There are two primary approaches to facilitate such a comparison, which constitute the two main disciplines of collaborative filtering: *the neighborhood approach* and *latent factor models*.

Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. An item-oriented approach evaluates the preference of a user to an item based on ratings of similar items by the same user. In a sense, these methods transform users to the item space by viewing them as baskets of rated items. This way, we no longer need to compare users to items, but rather directly relate items to items.

The collaborative filtering field has enjoyed a surge of interest since October 2006, when the Netflix Prize competition commenced. Netflix released a dataset containing 100 million movie ratings and challenged the research community to develop algorithms that could beat the accuracy of its recommendation system, Cinematch. A lesson learnt through this competition is that the neighborhood and latent factor approaches address quite different levels of structure in the data, so none of them is optimal on its own. This paper focusses on using neighborhood models to model the recommendation engine.

Neighborhood models are most effective at detecting very localized relationships. They rely on a few significant neighborhood-relations, often ignoring many ratings by a user. Consequently, these methods are unable to capture the totality of weak signals encompassed in all a user’s ratings. Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items. However, these models are poor at detecting strong associations among a small set of closely related items, precisely where neighborhood models do best.

For this paper, we will use two different approaches of the collaborative filtering algorithm to make a movie recommendation engine. We will compare the performance of a user oriented and an item oriented CF system. We use a data set from GroupLens Research[[1]](#footnote-1) provided by MovieLens. Main reason to for using this dataset Is because it is open source. This dataset includes movies and user ratings. We use python to build the two different recommendation algorithms.

Several limitations of the collaborative filtering algorithms should be noted. First, when data are sparse, the correlations are based on few common items and therefore are unreliable. Breese, Heckerman, and Kadie (1998) show that prediction performance suffers dramatically in such a situation. Second, collaborative filtering algorithms can be used only when preference data for an item already exists in the database. In other words, these systems cannot handle queries that pertain to new items. For example, most collaborative filtering algorithms cannot help a user who needs to know whether a new movie is good. In such situations, the database has no information about the movie, and the system is therefore unable to process such requests. Third, these methods use ad hoc prediction algorithms, which are not based on a statistical model. Consequently, they do not account for uncertainty, which may be less important for such low-risk purchases as movies, but can be very important when the stakes are higher for a consumer or company. Fourth, collaborative filtering systems do not explicitly incorporate attribute information. Though they are bootstrapped by creating “virtual users” who represent tastes (e.g., a virtual action fan who has high ratings for all action movies). The implications of such indirect accounting of product features is not clear. Finally, because collaborative filtering methods are correlational, they provide little explanation for a recommendation, a feature that can be important for building trust and enhancing customer loyalty.

In this section, we discussed the business understanding of recommendation engines. The structure of the rest of the paper is as follows. Chapter 2 provides a description of the data set to understand the structure of the input. Also, this chapter visualizes the data to gain a more thorough understanding of the data set. Chapter 3 explains the two CF approaches, the user and item oriented neighborhood models. Chapter 4 compares the recommendation results of both CF approaches. Chapter 5 summarizes the findings in this paper.

# Data understanding

The data sets come from GroupLens Research[[2]](#footnote-2) provided by MovieLens. GroupLens is a research group in the Department of Computer Science and Engineering at the University of Minnesota. GroupLens Research operates a movie recommender based on collaborative filtering, MovieLens, which is the source of these data.

The dataset used in this paper describes movie ratings based on a 5-star rating scale. The users in this dataset were selected at random for inclusion. The only selection criteria to be selected was that a user must have rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information about the user is provided. It contains 20,000,263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. These files are divided into three parts, including movie information table, user information table and score information table. These three tables are in the following format:

|  |  |
| --- | --- |
| **Movies** | **Ratings** |
| MovieID | UserIDs |
| Titles | MovieIDs |
| Genres | Ratings |

## Movies

Movie information is contained in the file `movies.csv`. Each line of this file after the header row represents one movie. The dataset files are written as comma-separated values files with a single header row. Columns that contain commas (`,`) are escaped using double-quotes (`"`). These files are encoded as UTF-8.

Movie titles have been entered manually or imported from <https://www.themoviedb.org/>, and include the year of release in parentheses. Errors and inconsistencies may exist in these titles. Genres are a pipe-separated list, and are selected from the following:

* Action
* Adventure
* Animation
* Children's
* Comedy
* Crime
* Documentary
* Drama
* Fantasy
* Film-Noir
* Horror
* Musical
* Mystery
* Romance
* Sci-Fi
* Thriller
* War
* Western
* (no genres listed)

## Ratings

All ratings are contained in the file `ratings.csv`. Each line of this file after the header row represents one rating of one movie by one user, and has the following columns:

userId, movieId, rating, timestamp

The lines within this file are ordered first by userId, then, within user, by movieId. MovieLens users were selected at random for inclusion. Their ids have been anonymized. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

## Descriptive statistics about movie data set

The movie dataset collect the films that are released from 1981 to 2015 and has received at least one rating, including 27278 movies. Here is a simple summary of the movie dataset.

## # A tibble: 6 × 4  
## movieId title  
## <int> <chr>  
## 1 1 Toy Story (1995)  
## 2 2 Jumanji (1995)  
## 3 3 Grumpier Old Men (1995)  
## 4 4 Waiting to Exhale (1995)  
## 5 5 Father of the Bride Part II (1995)  
## 6 6 Heat (1995)  
## # ... with 2 more variables: genres <chr>, year <chr>

The first chart is a scatter plot about the number of movies released every year. This figure shows that the number of movies increased every year without no exception. Especially, during the period from 1990 to 2010, the number increase from 300/year to 900+/year, by almost three times, which is significant development.

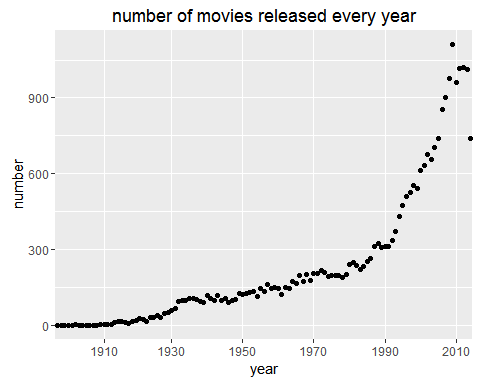


Figure 1. Number of Movies by Release Year.

The following figure count movies by genres. An important note about a movie is that one movie can have one or more genres. Therefore, the total number of movies by genres is much larger than 27278-the total number of movies. It is obvious that the drama is the most common genre of movie while IMAX is the rarest genre in the dataset.

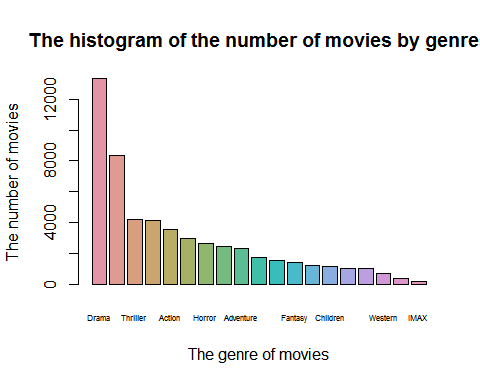


Figure 2. Number of Movies per Genre.

To specify whether the genres of released movies is different by year, a stack histogram of different genres movies by year is very useful. The following chart shows that there is no big difference among years after 1910 and before that Comedy、Drama、Musical and Romance are most common in the dataset.

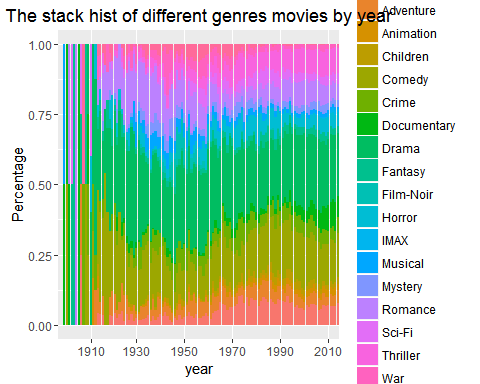


Figure 3. Percentage of Each Genre by Release Year.

## Descriptive statistics on the rating data set

All ratings are contained in the rating dataset. Each line of this file after the header row represents one rating of one movie by one user. Here is an overview about rating dataset.

summary(rating)

## userId movieId rating timestamp   
## Min. : 1 Min. : 1 Min. :0.500 Min. :7.897e+08   
## 1st Qu.: 34395 1st Qu.: 902 1st Qu.:3.000 1st Qu.:9.668e+08   
## Median : 69141 Median : 2167 Median :3.500 Median :1.104e+09   
## Mean : 69046 Mean : 9042 Mean :3.526 Mean :1.101e+09   
## 3rd Qu.:103637 3rd Qu.: 4770 3rd Qu.:4.000 3rd Qu.:1.226e+09   
## Max. :138493 Max. :131262 Max. :5.000 Max. :1.428e+09

Since a movie has been rated by plenty of users, the relationship between reviewers and the average rating is an interesting topic to figure out. The following image reveals that there is correlation between the two variables. That is, the average rating could reflect the real quality of this movie, which is not change with the change of number of reviewers. Besides, movies can be divided into three group based on the average rating: if the average rating is small than 3, the movie is bad, and if the rating is larger than 4, the movie must be good, and the others are medium ones.

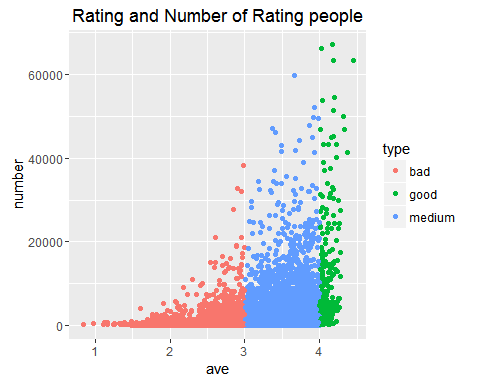


Figure 4. Number of Ratings by Average Score.

According the distinguish rules mentioned before, the stack histogram of the quality of movies by year can reveal whether the distribution of good movie or bad movie is concentrated. In fact, the rating data is incomplete since there will still be reviewers rating newest movie. Basically, the tendency of movies' quality is inconspicuous and the fluctuation of good movie is smaller than bad movies and medium movies.

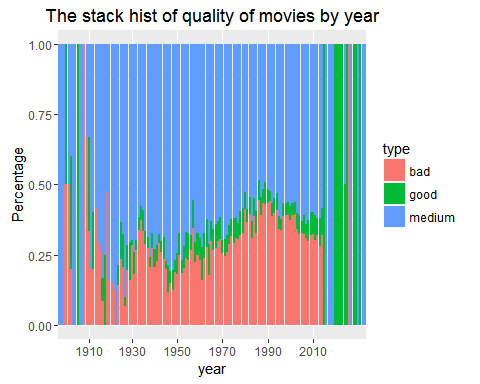


Figure 5. Percentage of Score Group by Release Year.

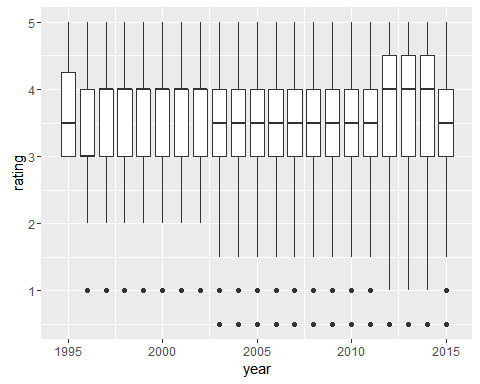
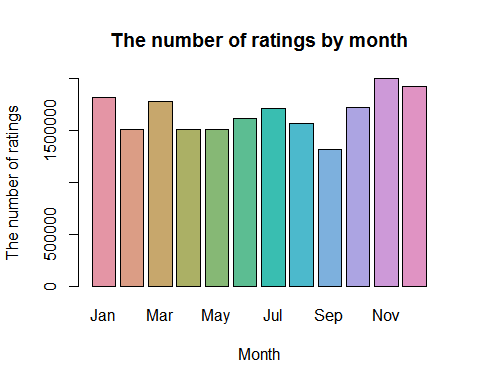
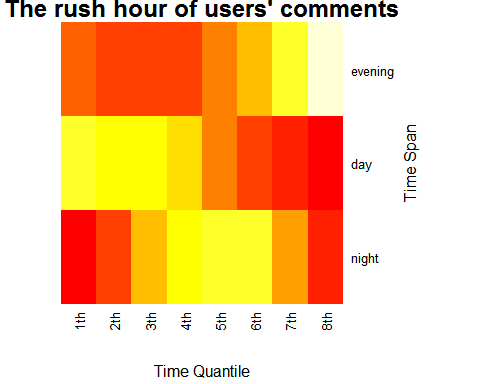


Figure 6. Boxplot of Rating by Release Year

Here is another chart to figure out the average quality of movies in every year. the bar plot is more intuitive. Considering that the timestamp is a variable recording comment time, we draw a histogram about the number of ratings by month to see if the rating was effected by month. The result seems no effect of month.



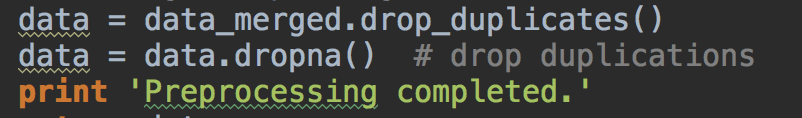
Similarly, to see whether the hour factor will influence rating, we plot a chart about the rush hour of users' comments. And there is no big fluctuation hour by hour.



# Modeling the Recommendation System

## Data preparation

Before building a recommendation system, we need to clean and remodel the data set. Unnecessary columns in the data are removed to increase the efficiency of the training model process. In the scoring data table, since we do not need to consider the user's comment time for the movie, so we get rid of the “timestamp” column. Use the following commands to remove the duplicate and null values from the data set, respectively.



Finally, we could get a cleaner data set.

## Modeling the Recommendation System

In this part, we will use two different algorithms to build a recommendation engine for movies. There are many approaches to build a recommendation engine. A popular approach for recommendation is collaborative filtering. User-based collaborative filtering assumes that users will probably like a product if it is liked by a similar user. A popular collaborative filtering algorithm is the k-Nearest Neighbor algorithm, which calculates the similarity between the current user and all other users (based on rating behavior). While this algorithm is simple and intuitive, it is not useful for users and products with few data (sparsity). Also, it can’t be used for new users and products (cold start). Other approaches include is content-based filtering and item-based collaborative filtering.

### Item-based collaborative filtering

（1）Construction of Item similarity

In terms of item similarity, the implicit feedback data set has higher algorithm efficiency in calculating similarity, so we use implicit feedback dataset to construct the similarity matrix. Implicit feedback data refers to the data that cannot clearly reflect the behavior of user preferences, the most representative of the hidden feedback data is the user's browsing behavior. The algorithm ignores the user's rating record for the movie and does not predict the user's rating for the unviewed video, but only focuses on the user's past browsing history of the movie. The film similarity calculation formula is shown in following formula.



The denominator |N(*i*)| denotes the number of users who liked the movie i, the numerator |N(*i*)∩N(*j*)| denotes the number of users who liked the movie i and the movie j at the same time.

There is a certain problem in above formula, if the film j is preferred by a lot of people, then the calculated similarity wij will be close to 1, indicating that many movies will have a high similarity with the popular movies. Based on the above analysis, we change the formula to the following one.



changed formula penalizes the weight of the movie j, thus reducing the likelihood that a popular movie will be similar to many other movies.



（2）Collaborative filtering recommendation

After building the movie similarity data set, users can be recommended based on the similarity degree and their historical behavior. First, we should select the movies which are rated higher by the certain user, thereby constructing a weighted list, including the other movies which are most similar to the selected movies. The following table is an example diagram based on item recommendations.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Movie** | **Rating** | **Sim.D** | **R.D** | **Sim.E** | **R.E** |
| **A** | 4.0 | 0.9 | 3.6 | 0.4 | 1.6 |
| **B** | 3.0 | 0.3 | 0.9 | 0.5 | 1.5 |
| **C** | 1.0 | 0.5 | 0.5 | 0.2 | 0.2 |
| **Sum** |  | 1.7 | 5.0 | 1.1 | 3.3 |
| **Normalization** |  |  | 2.94 |  | 3.0 |

As shown in the table, A, B, C are movies which are watched by the user. This user scored A, B and C for 4.0, 3.0 and 1.0. D and E are movies that have not been watched by the user. For videos that have not been watched, the corresponding column Sim.X will record how close it is to the movies being watched. For example, the similarity of D and A is 0.9, and Sim.D is the similarity between A and D, R.D is the evaluation value of the movie( 4.0 \* 0.9 = 3.6). The table predicts that this user may score a film D for 2.94 and a film E for 3.0. And so on, and ultimately in accordance with the forecast scores to recommend movies for this user.

### User-based collaborative filtering

（1）Construction of User similarity

We use the same method to calculate the similarity between users, that is, the implicit feedback data set are used to calculate the user similarity.

|  |  |  |
| --- | --- | --- |
| Movie | User1 | User2 |
| A | 1 | 1 |
| B | 0 | 1 |
| C | 1 | 0 |
| D | 1 | 1 |
| E | 0 | 1 |
| F | 0 | 1 |

For example, User1 has watched A、C and D, User2 has watched A、B、D、E and F, so that the similarity between User1 and User2 equals to 2/() = 0.5164.

（2）Collaborative filtering recommendation

After building the user similarity data set, users can be recommended based on the similarity degree and their historical behavior. We will use the same algorithm which we used in IBCF to make recommendation for users.

# Model Evaluation

（1）Description of model evaluation parameters

We have construct the IBCF and UBCF recommendation models respectively. To select the more accurate prediction models, we need to test the two models. In the model evaluation, the data sets will be split into two parts, including training set and test set. We write an algorithm to divide the original data set randomly into M copies (M = 8 in this experiment) using the uniform distribution. One of them is the test set, and the remaining M-1 is the training set. The training set data is used to train the model, and then the testing index is tested by using the test data. The recall rate, the accuracy rate and the coverage rate are used as the evaluation indexes of the model to test the two models respectively. The recall rate describes the weighting of the user-rating record contained in the final recommendation list, as shown in following equation, where R(*u*) denotes the movie recommended for user u, and T(*u*) denotes the set of movies that the user likes on the test set.



The accuracy rate describes how many items in the recommended list are accurately predicted. The calculation formula is shown following. As with the variables in the recall, R(*u*) denotes the item recommended for user u, and R(*u*) denotes the set of items that the user likes on the test set.



Coverage reflects the ability of the proposed algorithm to mine long-tail items. The higher the coverage, the better the recommendation algorithm can recommend the items in the long tail to the user. The coverage indicates the proportion of items in the final recommendation list. If all items are recommended to at least one user, the coverage is 100%. The formula for calculating the coverage is shown as follows. Where |I| denotes the number of movies.



（2）Model evaluation result

To test the two models from multiple dimensions, we employ the recommendation based on the implicit feedback dataset and score prediction models for the two algorithms, respectively, and set different K values. In the user similarity data set, the K value represents the K users who are most similar to a certain user. In the item similarity data set, the K value represents the K items which are most similar to a certain item.

And the results are as follows:

Table 1. Model evaluation results

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| UBCF | | | | | | | |
| algorithm | parameters | K=5 | K=10 | K=20 | K=40 | K=80 | K=160 |
| Implicit feedback dataset | recall | 24.85% | 24.72% | 23.61% | 23.94% | 24.02% | 24.94% |
| precise | 7.07% | 6.93% | 6.77% | 6.76% | 6.65% | 6.83% |
| coverage | 4.64% | 5.01% | 5.01% | 5.03% | 5.05% | 5.01% |
| Score prediction | recall | 25.54% | 26.63% | 25.01% | 24.81% | 25.13% | 26.39% |
| precise | 7.33% | 7.11% | 7.01% | 6.85% | 6.73% | 6.95% |
| coverage | 4.79% | 5.08% | 5.04% | 4.99% | 4.96% | 5% |
| IBCF | | | | | | | |
| algorithm | parameters | K=5 | K=10 | K=20 | K=40 | K=80 | K=160 |
| Implicit feedback dataset | Recall | 13.67% | 13.58% | 13.7% | 12.95% | 12.98% | 12.36% |
| Precise | 14.55% | 14.47% | 14.7% | 14.11% | 13.9% | 12.96% |
| coverage | 8.29% | 6.42% | 5.66% | 5% | 4.77% | 4.57% |
| Score prediction | Recall | 4.26% | 2.3% | 0.92% | 0.23% | 0.13% | 0.02% |
| Precise | 2.58% | 1.6% | 0.79% | 0.27% | 0.2% | 0.08% |
| coverage | 22.62% | 24.15% | 24.3% | 23.39% | 23.05% | 21.42% |

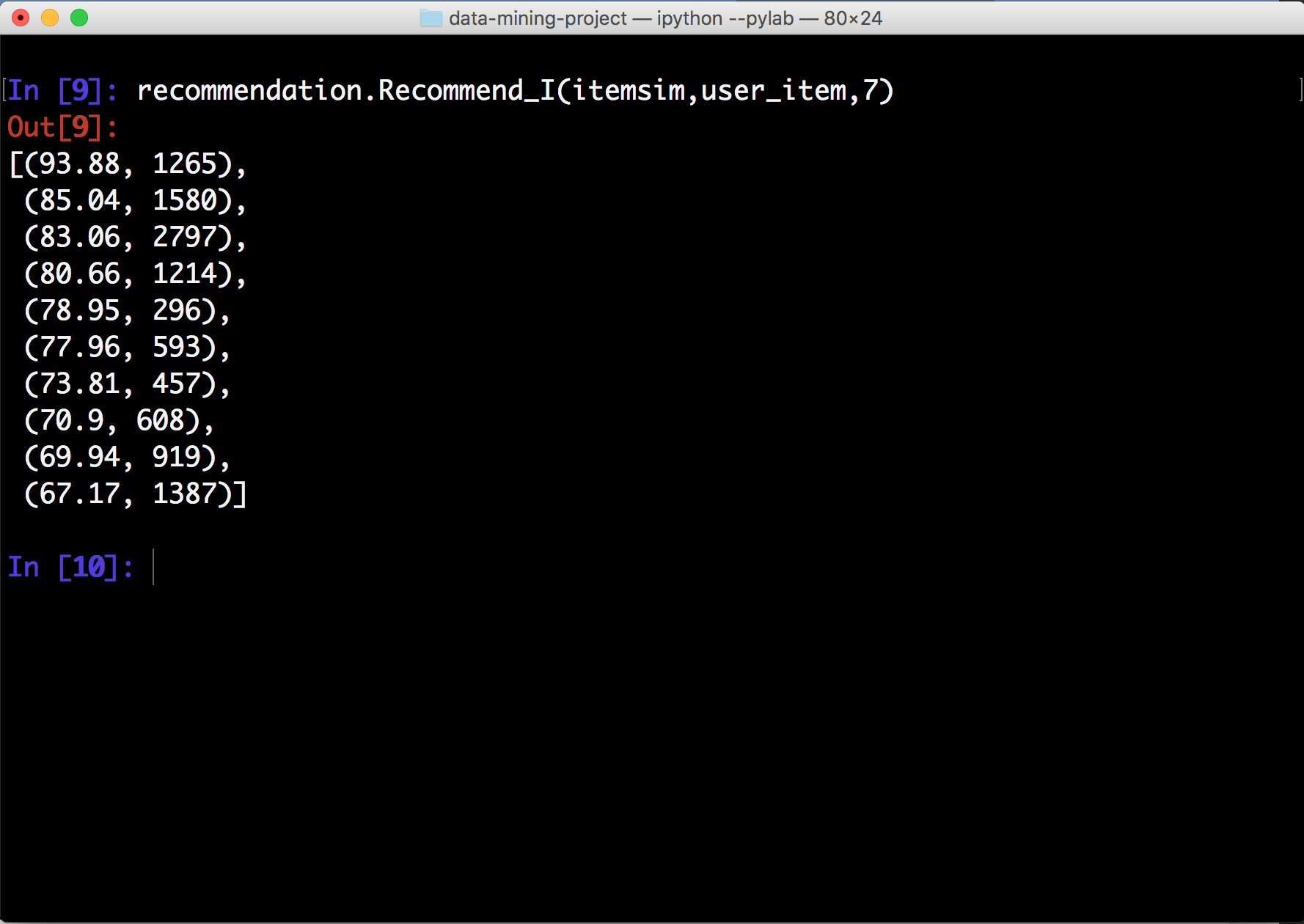
Table 1 shows the results of the implicit feedback and the score prediction for both algorithms using different values for K. The results show that increasing the number of neighbors in the UBCF model marginally affects the recall, accuracy and coverage rates. For the IBCF on the other hand, increasing the number of neighbors decreases the recall and accuracy rate for the score prediction. Remarkable, increasing K has a negative influence on the accuracy of the predictions. The UBCF algorithm has a higher recall rate for the implicit feedback as well as the score prediction. It also seems to perform better at prediction the user ratings for a movie.

# Summary

Modern consumers are inundated with choices. Recommendation engines match consumers with most appropriate products. Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. For this paper, we used two different approaches of the collaborative filtering algorithm to make a movie recommendation engine. We compared the performance of a user oriented and an item oriented CF system. We found that for both UBCF and IBCF, increasing the number of neighbors does not improve the performance of the recommendation engine. The UBCF algorithm seems to perform better when it comes to predicting the user rating for a movie.

# Appendix

A demo for our model



In this demo, itemsim represents the similarity matrix of items, user\_item represents the dictionary (one of the data structure in Python) which has the struction of “user:item”. When we want to make recommendations for user7, we can input the command in Terminal, and get 10 recommendation movies.

And If you want to access the source code, please click on <https://github.com/rubensprt/data-mining-project/blob/master/recommendation.py> to view our GitHub group.

1. (http://grouplens.org/datasets/movielens) [↑](#footnote-ref-1)
2. (http://grouplens.org/datasets/movielens) [↑](#footnote-ref-2)