Data Science Project

Sivaram Jawahar

Top N Movie Recommender

1 Introduction

In today's online world, recommender systems have become a natural part of the user experience (Jannach et al., 2016). There is an enormous amount of research focused on recommender systems based on historical ratings and review text. Numerous recommender systems have been introduced over the last years. These methods work with two types of data: user-item interactions, such as ratings or buying behavior and attribute information about the items and users, such as keywords or textual profiles. Recommender systems based on the former data type are called Collaborative Filtering (CF) methods, whereas recommender methods based on the letter type of data are called Content-Based (CB) methods. In this study, we will be focused on the former type of recommnder systems. More specifically, we aim to build CF type of movie recommender

in order tofind the most related movies for a target movie and make recommendations based on this.

2 Data

In this challenge we use the MovieLens dataset consisting of 20000263 ratings and 465564 tag applications across 27278 movies spanning a period of 20 years, from January 1995 to March 2015 [1]. This data was generated and used by Harper and Konstan (2015). Each row in the *Rating* data is a tuple with 4 components or variables and each row in the *Movie* data is a tuple consisting of 3 components or variables. Table 1 presents these rating and movie tuples compositions, variables, and their corresponding descriptions. The datasets are initially supplied in csv format per product category. In the csv format observations are grouped together by the product (movie) ID which is defined as *movieId* in movie tuple, as presented by Table 1. We merge the rating data with movie data and then removed the unnecessary variables of the rating and movie tuples which will not be used in this analysis by keeping only the variables *movieId*, *userId*, *rating* and *title* for the purposes of our analysis. The reason of including the *title* variable is that we will need the actual names of the movies rather than the movie Id's for making movie recommendations.

Variable	Description
	Rating tuple
userId	The customer ID of the user who has provided the rating
movieId	The ID of the movie evaluated in the rating
rating	The rating of the movie which is evaluated in the review
timestamp	The time of the rating
	Movie tuple
movieId	The ID of the movie evaluated in the rating
title	The name of the movie
genre	The genre of the rated movie

Table 1: Rating and Movie Tuples Composition

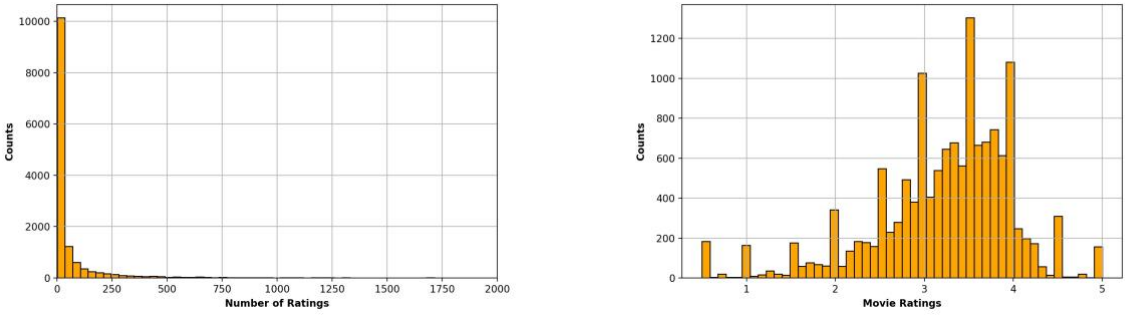
2.1 Descriptive Statistics

In order to get more insights in our data we group the dataset by the movies and take the average of all ratings per movie. We determine how many times each movie has been rated to create the *count* variable. Figure 1 (a) represents the histogram for the number of rating amounts in the rating dataset. We observe that most of the movies have received less than 50 ratings. Furthermore, we calculate the average rating per movie. From Figure 1 (b) we observe that the integer rating values have taller bars compared to the floating rating values, since most of the users often use integer value to rate a movie. Moreover, we observe that the rating data is approximately normally distributed with the mean of

[1] http://files.grouplens.org/datasets/movielens/ml-20m-README.html

approximately 3.5. Figure 2 presents the rating histogram in a log scale. From this figure we observe that most of the users have rated the movies with star ratings 3 and 4. From the scatter-plot presented in Figure 3, which plots the average ratings against the number of ratings, we observe that there is a positive relation between number average rating and number of ratings. Thus, this graph indicates that usually movies with large amount of ratings have high average rating. This can be reasoned by the fact that well-known movies are watched by a large amount of viewers and usually, are identified as *good* movies (rated highly).

Usually, the rating data contains lots of missing ratings, because most of the users rate only small amount of items (movies). Figure 4 (a) presents the distribution of movie rating frequency and we observe that rating frequency is *long tail* distributed. Thus, only small fraction of movies are rated frequently, which we refer as *popular* movies, whereas, large amount of movies have not been rated at all. More specifically, 52.76% of all movies have received less than 10 ratings. From Figure 4 (b) which presents the rating frequency in log-scale, we observe that approximately 2200 out of 14026 movies received ratings by more than 100 users out of 7120 users. Thus, there is a large amount of inactive users as well. The same pattern can be observed in the rating frequency of users presented in Figure 5.



(a) Number of Ratings Histogram

(b) Average Rating Histogram

Figure 1: Movie Rating Histograms

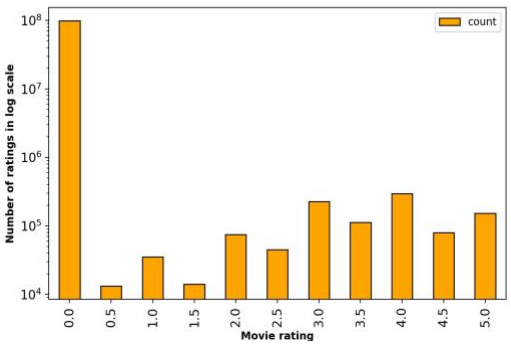


Figure 2: Movie Rating Histogram

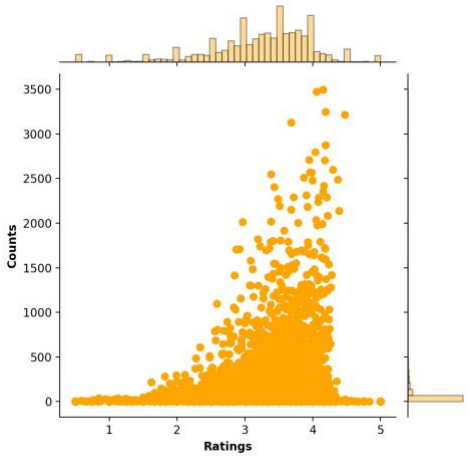
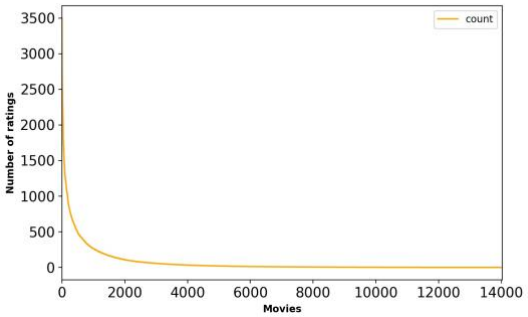
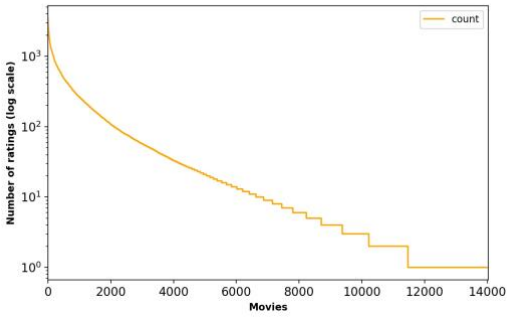


Figure 3: Scatter-plot of average movie ratings against the number of movie ratings.

3 Methodology

Collaborative Filtering (CF) approach proposed by Sarwar et al. (2001) uses the collaborative power of ratings of the user and the ratings of other similar users to predict whether the item, which is not rated yet, will interest this user or not. CF approach has two key advantages. Firstly, it performs well even in the domains where there is not a lot of associated content with items or where the content is difficult for computer analysis. Secondly, it is able to provide accurate recommendations regarding the items that are relevant to the user. However, CF encounters problems of matrix-sparsity and cold-start problem. CF methods are divided into two primary areas: Memory-based methods, which are also called neighborhood-based algorithms (Sarwar et al., 2001) and Model-based methods (Koren et al., 2009), also called Latent Factor Models (LFM). Neighborhood methods are focused on computing the relationships between users or the





(a) Rating Frequency of All Movies **(b)** Average Rating Histogram

Figure 4: Rating Frequency Distributions

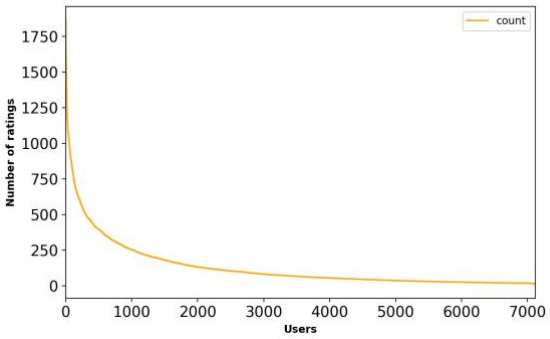


Figure 5: Rating Frequency of All Users

relationships between items, while Latent Factor Models are focused on both aspects simultaneously trying to explain the ratings by characterizing both users and items on some amount of factors inferred from the patterns of ratings. Nearest-Neighbor CF is based on either user-user or item-item similarities. Neighborhood-based CF based on user-user similarity is called User-User CF, whereas neighborhood-based CF based on item-item similarity is called Item-Item CF. In the case of User-based CF, the algorithm determines similar users based on their profiles and purchase behavior in the past and makes rating predictions. The idea behind this approach is that similar users are expected to assign similar ratings to the same item. In case of Item-based CF, the system finds similar items and assuming that similar items will be rated in a similar way by the same person, it predicts the rating corresponding to a user assigned to that item. Usually, the Item-based CF is preferred over the User-based CF because users can change their preferences and choices (aging, change of environment) whereas items (relatively) does not change over time.

3.1 Item-based Collaborative Filtering

As it was mentioned above, user-based CF algorithms suffer from few problems. Firstly, these type of algorithms suffer from scalability problems when dealing with millions of users and items. Secondly, user-based recommender algorithms suffer from sparsity problem because if the large limitation regarding the user and item information in the historical data. One way of overcoming these limitations is by focusing on the item similarities instead of user similarities. In case of item-based recommender algorithm, the historical information is used to identify relations between the items such that the purchase of an item often leads to the purchase of another *similar* item(s) Billsus and Pazzani (1998) Kitts et al. (2000). The key motivation behind Item-based CF is that a customer will more likely buy items that are similar or related to the items that she bought

already in the past. Item-based recommender algorithms first analyze the ratings matrix to identify different items relationships, and then use these relationships to indirectly compute item recommendations for users **?**. In this type of recommender algorithms, one usually builds an item-to-item matrix of similarities by iterating through all possible pairs of items. However, this is inefficient way of computing item similarities because many pairs have no common users, thus similarity can not be computed. In order to avoid this, one can better select only the pairs of items for which the similarity can be computed by initially scanning all the items and for all the users that bought an item, identify the other items bought by those customers. Then one can efficiently compute the similarities only for these item pairs.
Assuming that if a user likes a certain movie, then she may like similar movies in the future, we will focus on Item-Item CF recommender systems in order to find similar movies corresponding to a movie to make recommendations. But in order to find similar movies, we need to define the concept of *similarity* or neighborhoods. One can select all the neighbors or select random neighbors to be in the neighborhood. One of the most popular way of finding similar items in the field of recommender systems is in terms of K-Nearest Neighbor (KNN) approach, which selects top-K neighbors that are most similar to the target item.

3.1.1 Top-N Movie Recommender System

K-Nearest Neighbors (KNN) method is a non-parametric Machine Learning algorithm that separates all data points into fixed K number of clusters, neighborhoods. This method can be used for both classification and regression purposes. We will use KNN for the classification purposes in order to identify the K closest instances (movies) given one target instance in our database (particular movie). The idea behind the model is that the data points which are closest to each other, have the smallest *distance* from each other, will be assigned to one neighborhood. In the field of recommender systems, this algorithm calculates the *distance* between the target item (movie) and all the remaining items (movies) in the dataset. Then it ranks all those movies based on their corresponding distances and chooses the top K *nearest* movies (neighbors) in order to recommends those, as the most similar movies, to a user.

The recommender algorithm that we will use to find the N most similar movies given a particular movie title in our dataset is summarized in the Algorithm 1. What is important to note in this algorithm is that we use only the training set to fit the KNN model, compute the distances, and make movie recommendations. Thus, the training set is only used for the purposes of calculating the prediction accuracy of the algorithm.

Algorithm 1 Top-N Movie Recommender System

Input: Rating data, Movie data, Number of top-N recommendations, Title of target movie **Output:** List of most similar N movie titles **Step 1:**

- Pre-processing the rating and movie data
- Create a mapper between movie id's and movie titles
- Split the data into train and test sets
- Transform the rating (train/test) data to a sparse movie-user matrix **Step 2:**
- Fit the model using the movie-user sparse matrix based on training set
- Use the input title to find the matching movie index
- Choose the best match from all matching movie indices
- Predict the distances of neighbor movies from the target movie with number of KNN neighbors K equal to number of top recommendations N
- Sort these N neighbor(closest) movie indices in ascending order
- Transform these movie indices back to the movie titles
- Print these N movie titles
- Use the test set movie-user sparse matrix to calculate the prediction accuracy of the model

---

4 Evaluation and Results

There are few key aspects in this type of recommender systems that should be studied and taken into account in order to obtain qualitative recommendations. A key aspect that should be taken into account while using top-N type of recommender systems is the selection of users and items that should be used while making recommendations. Namely, we need to consider the *candidate item* and *candidate user* selection. As it was mentioned earlier, large amount of movies have not been rated at all and there is also a large amount of inactive users. Therefore, to avoid noisy pattern caused by the absence of user and movie ratings, we will use only the most popular movies and most active users, while making recommendations. We will use as movie popularity threshold and as active membership threshold value 50. That is, all the movies that have more than 50 ratings and all the users that have rated more than 50 movies will stay in the main dataset, while the remaining movies and users will be removed from it.
Another key aspect for this type of analysis is train/test separation procedure. In order to evaluate the performance of recommender system one should split the data into training and test set. However, this splitting procedure should be done with care, otherwise one can create a test model with users that have no or very few rated movies in the training set or vice versa (Said et al., 2013). To avoid this problem we perform personalized data separation. Firstly, we sort the rating data per user with respect to the *timestamp* variable from the rating data as introduced in Table 1. Then we put the last 30% of this users rating data in the test set and the remaining data we put in the training set, such that 30% (most

recent) of ratings of each user is moved to the test set and the remaining 70% is moved to the training set. Furthermore, the number of top recommendations, the number of neighborhood in the KNN method, has also a large impact on the prediction accuracy of the KNN based recommender systems (Said et al., 2013). Therefore, we will use 6 different values of N = {3, 5, 10, 20, 30, 50} in order to investigate the impact of N on the accuracy of top-N movie recommender system. The final key aspect in the Itembased CF type of recommender systems, is the choice of the method used to compute the similarity between the items. There are various measures that can be used as *distance* measures in KNN. The most popular distance metrics are *Cosine Similarity*, *Euclidean* or *Mahalanobis*, *Jaccard Similarity*, and *Minkowsky* (Aggarwal, 2016). Following the approach taken in Sarwar et al. (2001), will use the *Cosine similarity* as a distance measure for computing the similarity between two items. The idea behind this measure is to treat each movie as a vector in the space of customers and use the cosine measure between these vectors to measure the similarity. Let us define by A and B the ndimensional vectors corresponding too movies A and B in the n *x* m user-movie matrix, then the *cosine similarity* between these two vectors (movies) is defined as follows:

$$sim(A,B) = cos(\vec{A},\vec{B}) = \frac{\vec{A}\vec{B}}{\left\|\vec{A}\right\|_2 \left\|\vec{B}\right\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (4.1)$$

where both A and B are non-zero vectors. We observe that this similarity measure will be high if each customer that buys movie A also buys the movie B. One of the reasons of the increased popularity of this measure in the field of recommender systems, is that *cosine similarity* takes into account the buying frequency of different items, this is done by the denominator in Equation 4.1. Thus, the frequently bought items will tend to be similar to the infrequent bought items which will avoid the obvious recommendations of very popular items (Karypis, 2001). Figure 6 presents the idea behind Cosine Similarity when determining the distance between two movies. Since, the rating data contains only the movie indices but not the titles, while our goal is to build a recommeder system that takes as an input the movie title and gives as an output N most related movie titles. Therefore, for finding the matching titles of the movies corresponding to each movie Id, we use the *Fuzzy String Matching* technique in *Python* in order to transform the movie title to the corresponding *movieId* and vice versa.

There exist many evaluation metrics for measuring the prediction accuracy of the Recommender Systems. In case of the top-N recommender systems for a user, three most popular evaluation metrics are *Pecision*, *Recall* and F_1 measures. We will use the *Precision* for evaluating the accuracy of our model. In case of top–N recommendations, this

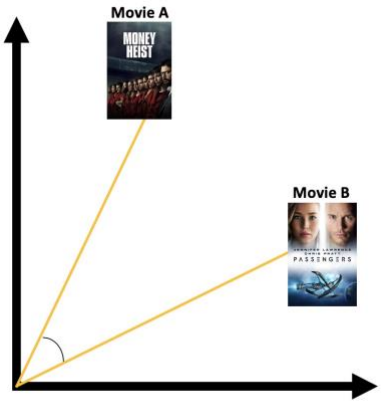


Figure 6: Cosine Similarity between two movies

measure is called precision–at–N or P@N measure and is determined as follows:

$$P@N = \frac{|\ hits\ |}{N} = \frac{|\ test \cap topN\ |}{N} \tag{4.2}$$

where *hits* represent the number of movies in the test set of each user that where also present in the N recommended movies. For a give *target* movie, we will calculate the precision value per user. Once we have determined the prediction accuracy for all users, we will take the average over all users in the test set.
We perform the experiment for the following randomly chosen 6 different *target* movies: *Babe (1995)*, *Babysitter (1995)*, *(500) Days of Summer*, *Things I Hate About You*, and *Three Amigos*.

Figure 7 presents the prediction accuracy's in terms of PN of the top-N movie recommender system discussed in the Section 4 for 6 different model settings and movies. We can see that for 5 out of 6 movies the model with N = 3 outperforms the remaining models. So, we see that smaller N results is better performing top_N recommender system. Only in case of the movie *Babysitter* it holds that the model with number of most similar recommendations equal to 5 performs better than N = 3 case. For all movies it holds that the model with N = 50 performs the worst. Figures 8, 9, 10, 11, 12, and 13 presents the outputs of our top-N movie recommender system with most 3, 5, 10, 20, 30, and 50 similar movies, respectively, for the input movie *Things I Hate About You*. We can see that the recommended movies are quite different in case of each model setting. Thus, the choice of number of recommendations (number of neighbors in KNN) has a large impact on the list of recommended movies. What we also observe is that the movie *Pink Floyd: The Wall* is present in all 6 settings.

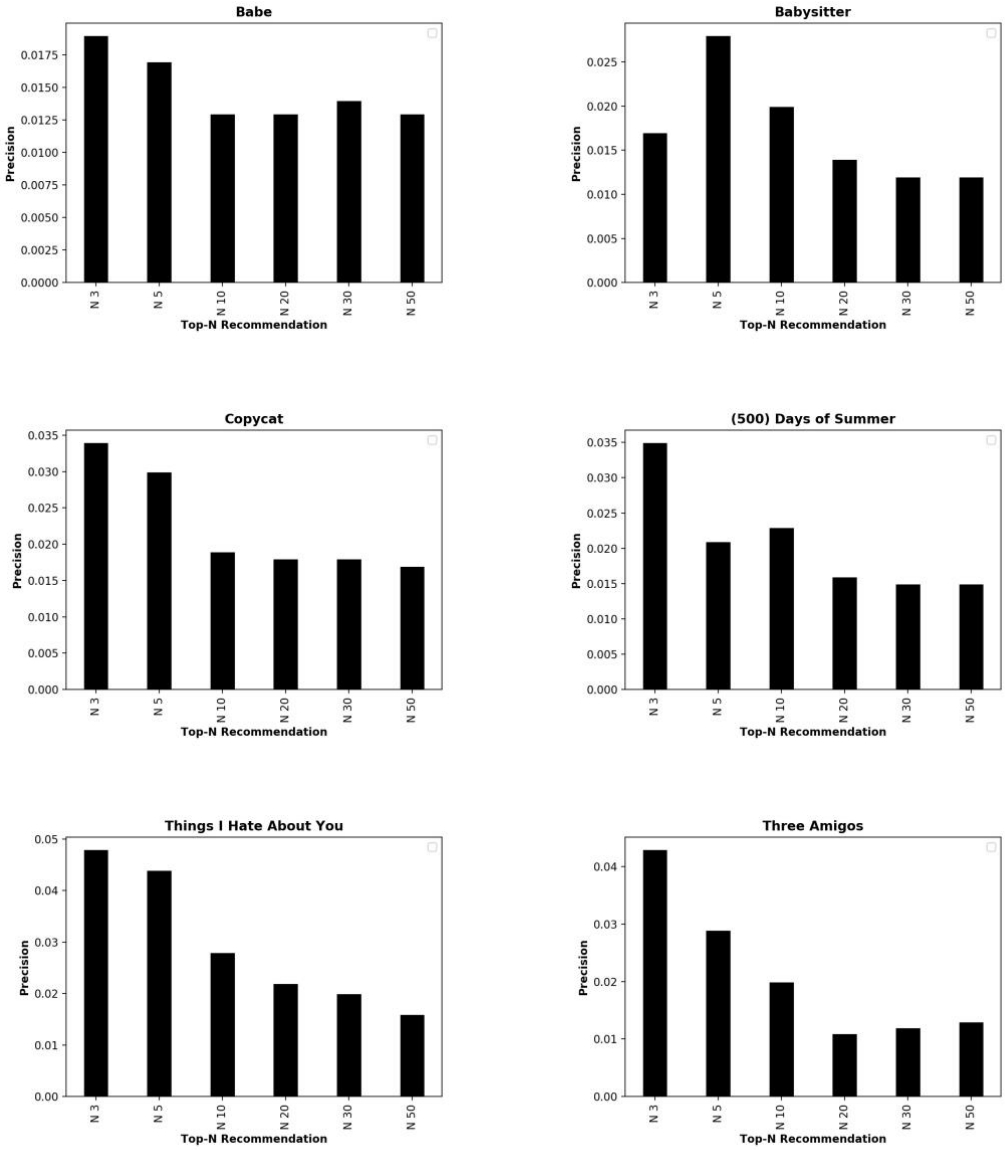


Figure 7: Prediction accuracy of Top-N Movie Recommender. *N x* represents the model with number of top recommendations equal to x, etc.

```
Recommendations for Things I Hate About You:
--------------------------------------------
1: Mars Attacks! (1996)
2: Georgia (1995)
3: Pink Floyd: The Wall (1982)
```

Figure 8: Top 3 movie recommendations for movie *Things I Hate About You*

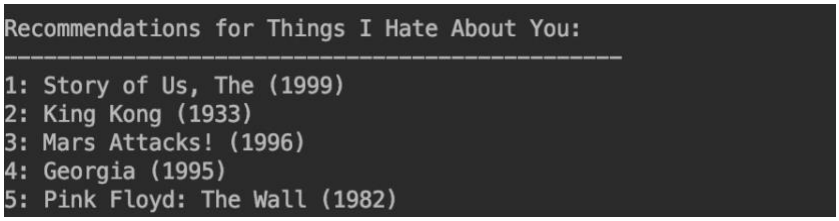


Figure 9: Top 5 movie recommendations for movie *Things I Hate About You*

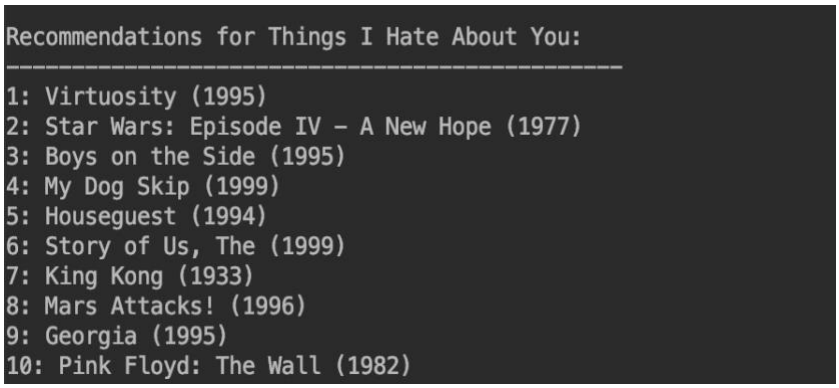


Figure 10: Top 10 movie recommendations for movie *Things I Hate About You*

Figure 11: Top 20 movie recommendations for movie *Things I Hate About You*

Figure 12: Top 30 movie recommendations for movie *Things I Hate About You*

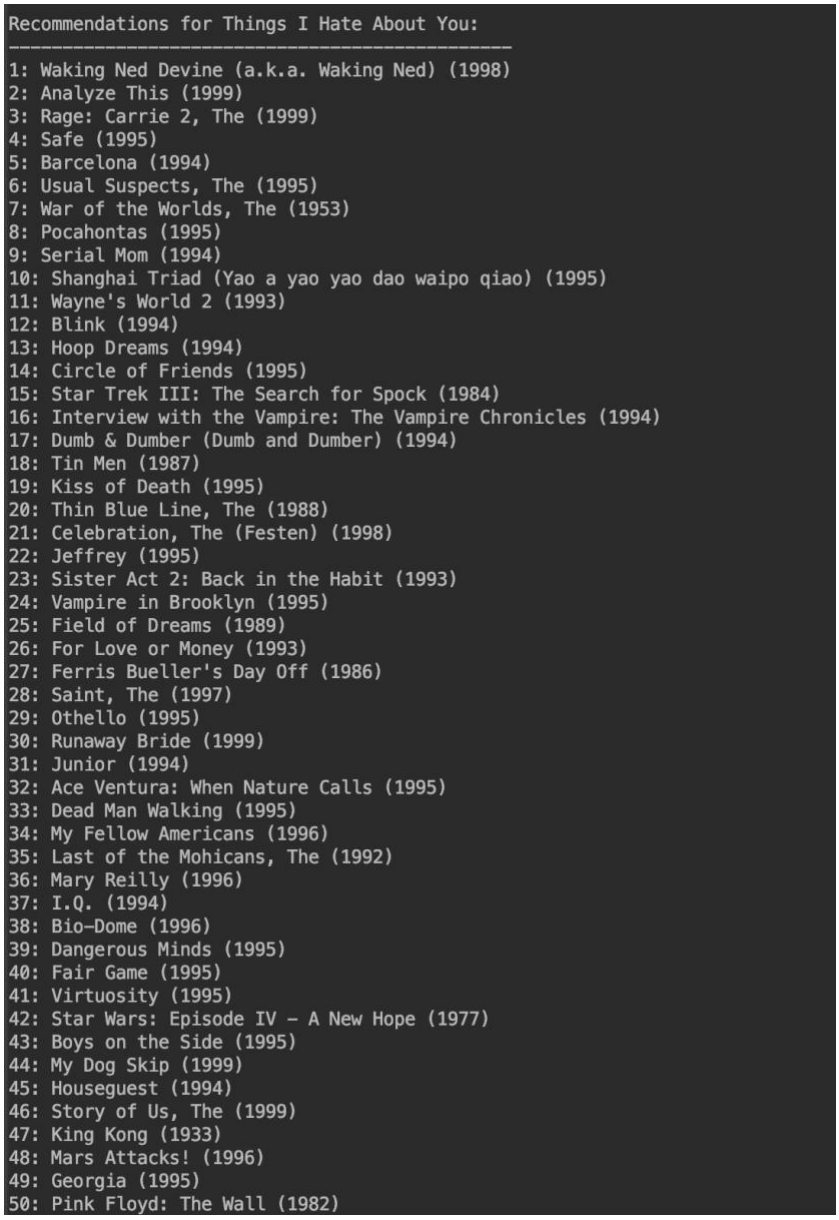


Figure 13: Top 50 movie recommendations for movie *Things I Hate About You*

References

Aggarwal, C. (2016). *Recommender Systems*. Thomas J. Watson Research Center.

Billsus, D. and Pazzani, M. J. (1998). Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 46–54. Morgan Kaufmann Publishers Inc.

Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *ACM*, 5(4):1–19.

Jannach, D., Resnick, P., Tuzhilin, A., and Zanker, M. (2016). Recommender systems: beyond matrix completion. *Communications of the ACM*, 59:94–102.

Karypis, G. (2001). Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 247–254. ACM.

Kitts, B., Freed, D., and Vrieze, M. (2000). Cross-sell: A fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Said, A., Bellog´ın Kouki, A., and de Vries, A. (2013). A top-n recommender system evaluation protocol inspired by deployed systems. In *Proceedings of the 2013 ACM RecSys Workshop on Large-Scale Recommender Systems*.

Sarwar, B., Karypis, G., Konstant, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithm. In *10th International Conference on World Wide Web (WWW 2001)*, pages 285–295. ACM.