G25: Netflix Recommendation Kaggle Competition

Abstract: The objective of this project is to analyze movie ratings provided by the Netflix Prize Competition and to build a recommendation system. The first challenge is the difficulty to handle a dataset with 100 million records, and we overcame this by working with subsampled and filtered data. We identified some interesting trends in analysis, such as customers being inclined to give higher but few ratings, and users that give fewer than 50 ratings tend to give higher ratings. We also performed frequent pattern mining, identifying movies that are frequently favored together. Additionally, the project involves building and testing the recommendation system itself.

Introduction: Recommendation systems are ubiquitous today. It is fascinating how large historical data can be transformed into a tool to help people with their future decision-making process. From doing groceries, deciding what to watch next and what content we might like - everything is the result of a comprehensive recommendation system. The Netflix Movies and Ratings dataset is a clear example of this historically recorded data being used to predict preferences. It includes information about user-movie interactions, which may bring insights into users' movie selection behavior. Our goals are to explore the data, apply the concepts learned in class, discover interesting user behavior, and build a recommendation system based on user ratings on different movies.

Motivation: The Netflix dataset is an excellent opportunity to apply the methods we have learned and will be learning in class. It allows us to use data mining techniques such as finding frequent patterns, calculating similarities between the data (each user/movie), and also using deep learning methods, which we will study. Building recommendation systems also is very applicable to real-world problems, and knowledge acquired can be further used in other fields like social media, grocery and online shopping.

Related work: Since the dataset we are using comes from the open competition, there is a lot of work previously completed on the task. We used these notebooks [1] as a reference for trimming the data so that it can fit the RAM. As for the pattern mining, additionally to the class materials, documentation of the fpgrowth function [2] was used. For the purpose of building a recommendation system, literature review on the topic was conducted. There is abundant literature in recommender systems [3 - 7] and the commonly used methods can be categorized into two paradigms: collaborative filtering and content based. Collaborative filtering methods are based solely on the observed interactions (the matrix) between users and items, while content based approaches rely on additional information about users and/or items.There are 2 classes of collaborative filtering methods: memory based methods (user-user and item-item) directly make use of the interaction matrix without assuming any model and deploy nearest neighbors search to find similar users or items; model based methods assume a latent model that explains the interaction matrix and employ matrix factorization to decompose the huge and sparse user-item interaction matrix into a product of two smaller and dense matrices, a user-factor matrix and an item-factor matrix.

Methodology:

EDA. We performed analyses and built models on the training dataset (over 100 million ratings that 480,189 users gave to 17,770 movies) provided by Netflix for their first competition in 2006. We also did preprocessing on the data, which involved cleaning the data and transforming it into a dataframe that can be easily and effectively used for further analysis. In order to get to know the data better, exploratory data analysis (EDA) has been performed. It involved descriptive statistical analyses, such as calculating the means, standard deviations, as well as visualizing distributions for different types of movies and users, as discussed during the class.

Data filtering and sampling. Since exploratory analyses do not require expensive computations, we were able to investigate the entire dataset (with over 100 million ratings that 480,189 users gave to

17,770 movies). Predictive analyses, however, involve much more complicated algorithms that are computationally intensive and memory demanding, which is the case for matrix factorization that we planned to do. Due to limited available computational resources, we decided to perform the movie recommendation (rating prediction) task on a subset of the original data. We cut out the data with two steps. First, we filtered out users who gave (relatively) few (<=50) ratings as well as unpopular movies (those that received <=200 ratings), for their sample sizes are small and variances due to sampling errors are large. After filtering them out, we were left with 324,471 users, 13,077 movies, and 95,726,056 ratings, which is still too large a dataset for us to handle with computing restrictions. So we continued the data removal process by subsampling 25% of the users in the filtered data. Specifically, we drew a random sample (with size = 25%×324,471 ≈ 81,117 users) from all users uniformly without replacement, and selected a subset of the filtered data containing those sampled users only. The resulting dataset is composed of 81,117 unique users, 13,077 movies, and 23,911,936 ratings/records (about 24% of original data), which we built the recommendation model upon.

Pattern Mining. Frequent Pattern Mining stage of the project implies applying FPGrowth algorithm to the (sampled) data and identifying the frequently highly rated movies. FPGrowth was favored over Apriori because of its efficiency. At this stage, we can apply the knowledge of Pattern Mining algorithms from class to real-life dataset, see how constraints and different support levels affect the results.

The fpgrowth framework [2] was used to perform the experiments. The dataframe and minimum support levels were passed as arguments to the algorithm. The whole filtered dataset described above was tried first, however since fpgrowth performs a number of scans over the data to find patterns, it kept filling up available RAM resources. Therefore, the dataset was sampled further. As we are interested in sets of movies that are frequently liked by the users together, a data constraint to filter out only high ratings (rated 4 or 5 stars) was used. The final sampled dataset used for this stage had 1.5 million of ratings with 850 movies and 10k unique users.

Recommendation Model. To achieve the goal of recommending relevant movies to users, we decided to build a model to predict the rating score for a given user and movie combination. Due to lack of additional information about the users and movies, we adopted the collaborative filtering approach to building our prediction model. Specifically, we applied matrix factorization to predicting ratings. The movie ratings in our data can be represented in a user-item interaction matrix $R \in \mathbb{R}^{m \times n}$, where m is the number of users and n is the number of items/movies. Note that since each user only rated a limited number of movies, R can be extremely sparse with many empty values. The purpose of matrix factorization is to fill in those empty elements by leveraging information from observed values in R. The goal is achieved through expressing R in terms of two embedding matrices, one depicting users and the other characterizing movies, as follows:

$$R \approx UV^T, where\ U \in \mathbb{R}^{m \times d},\ and\ V \in \mathbb{R}^{n \times d}.$$

The value d (<< m and n) is the number of embeddings that represent latent factors/dimensions in the user-embedding matrix U and the item-embedding matrix V. We want to construct two embedding matrices such that their product UV^T approximates R . This can be realized by minimizing the squared Frobenius distance between R and UV^T:

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \| R - UV^T \|_F^2,$$

which is the objective (or loss) function that we need to optimize for. For this project, we used a generic algorithm for minimizing the loss function, the stochastic gradient descent (SGD).

Empirical results:

EDA. Users can rate a movie from one star to five stars on Netflix. In our preliminary EDA, we obtained the following information: distribution of ratings is left-skewed; that is to say, there are fewer low ratings (1 and 2 stars), and most ratings (>60%) are 3 and 4 stars. The distribution of the number of ratings each user gave is incredibly right-skewed. As the number of ratings increases, there is a prominent exponential decay of frequency/proportion. The number of ratings that appear most frequently is 2 per user. Most of the users gave few ratings, and few gave many. For movies that received fewer ratings (fewer than 10000), the number of ratings per movie peaks at 100-200 ratings and decays rapidly afterwards. Most movies that received many ratings (over 10000) received fewer than 25000 ratings. Overall, people tend to give higher but few ratings.

Based on the preliminary analysis, we wanted to expand the discussion by asking what are the rating distributions of users that gave few/many ratings. How do the rating distributions look for movies that received few ratings and those which received many? Many films that received no more than 200 ratings received mediocre scores. There are more lower ratings (1 and 2 stars) than higher ratings (4 and 5 stars), but overall there is a lot of variability in ratings for those movies. (Appx:Fig. 1.) Among the 10 most popular movies (which received the most ratings), *The Green Mile*, *Forrest Gump*, and *Pirates of the Caribbean: The Curse of the Black Pearl* have the highest mean and lowest variance for ratings. Users had mixed attitudes towards the other 7 movies. (APPX:Fig. 2., TABLE. 4.) Let us look at the rating distributions from users that gave fewer than 50 ratings. One can tell that the distribution is left-skewed, indicating that these users tend to give higher ratings. Which is intuitive, as people who watch few movies probably watch only the best (most recommended) ones, or those who do not rate movies on a regular basis typically only rate movies when they want to highly recommend them to others. How about the mean and median ratings each user gave? Each user's mean and median ratings are on the high end of the spectrum: the mean peaks around 3.75, while the median peaks at 4. Very few users gave low ratings of 1 or 2. (APPX:Fig. 3.) We can infer that users who gave fewer than 50 ratings cared more about giving high ratings to their favorite movies than giving lots of ratings to others. The distribution is left-skewed for the ratings from users that gave more than 1000 ratings. (APPX:Fig. 4.) Unlike the distribution for users who rated <=50 movies, the distribution for users who rated >=1000 movies peaks at 3. Users who rated many movies generally gave lower ratings than those who rated few. A reasonable interpretation is that users who rated >=1000 movies are like critics. They have watched many movies and are very strict about giving high ratings.

Pattern Mining. As for the frequent pattern mining, the FPGrowth algorithm was applied to the filtered-then-sampled dataset. Before the algorithm could have been applied, the dataset was transformed into the transactions database type: a list of highly rated movies per user being considered as a transaction. Different relative support levels have been tried. With a more realistic dataset (random sampling across all movie ids), support levels down to 0.001 and less didn't give any frequent itemsets, which is intuitive. With the goal to explore the algorithm in mind, weighted random sampling was done favoring frequent movies. For the same data, different support levels were tested. Table 1 represents the number of frequent k-itemsets we got from these experiments.

	min_sup		
	0.2	0.3	0.4
Total number of k-itemsets	**24851**	**814**	**80**
Distribution for each k:	1: 325 2: 3849 3: 9571 4: 7855 5: 2789 6: 450 7: 12	1: 132 2: 436 3: 217 4: 29	1: 48 2: 30

TABLE. 1. Number of frequent itemsets generated for different support levels

As expected, the number of itemsets as well as their maximum length increase as min_sup decreases. For example, FPGrowth with min_sup=0.3 resulted in 814 frequent k-itemsets, with the max length of the pattern being 4. The most frequent 4-itemset are movies with ids - 14240, 1905, 2452, and 11521, they were given 4 or 5 rating by 37% of the users! See Appx:Fig.5 and 6. To evaluate the obtained numbers, we cross checked the titles for this list of movies. The most frequently liked together movies are - “Pirates of the Caribbean: The Curse of the Black Pearl”, “Lord of the Rings: The Fellowship of the Ring”, “Lord of the Rings: The Two Towers” and “Lord of the Rings: The Return of the King” . These results align with common sense, therefore proving that the experiments were conducted successfully.

Recommendation model. We built and trained the matrix factorization model using an existing set of Python functions provided by Google Developers in their machine learning advanced courses on recommender systems [8]. First, we randomly split the sampled data as described earlier into a training dataset made up of 90% of the sample and a testing dataset with the rest 10% of the sample. For efficient representation of rating matrices, we transformed the training rating matrix and the testing rating matrix into sparse tensors made available via TensorFlow. We evaluated model prediction performance by operationalizing the objective function as mean squared error (MSE):

$$MSE(R, UV^T) = \frac{1}{|R_{obs}|} \sum_{(i,j) \in R_{obs}} (R_{ij} - (UV^T)_{ij})^2$$

where R_{obs} is the set of observed ratings. We started training by initializing *U and V* using a normal distribution with $\mu = 0$, and $\sigma = 0.5$. Then, we employed SGD to find *U and V* that minimize MSE for the training rating matrix and used the fitted *U and V* to calculate the MSE for the testing rating matrix. For hyperparameter tuning, we followed a grid search approach to tune the number of latent factors (d) in *U and V* by looking at the 5 values: [10, 20, 50, 80, 100]. Since training a model on the mentioned dataset takes a significant amount of time, hyperparameter tuning was done on a smaller batch (25m ratings, 13k movies and 50k users) first. The following table displays the training and testing MSEs for each of the 5 values of d :

	init_stddev=0.5, num_iterations=1000, learning_rate=40.				
	embedding_dim				
	10	20	50	80	100
train_error	0.987	0.967	1.046	1.100	1.137
test_error	1.013	0.981	1.118	1.224	1.310

TABLE 2. Training and testing MSEs for different embedding dimensions

Based on the table, we decided to take d = 20 as our final choice for the embedding dimension. The plot below shows the training progress of our finalized model, with MSE values on the y-axis, number of iterations on the x-axis, a training MSE curve, and a testing MSE curve.

```
In [20]: # build and train another model based on tuned hyperparameters using a smaller dataset
         model2 = build_model(dt_sampled, embedding_dim=20, init_stddev=0.5)
         model2.train(num_iterations=1000, learning_rate=50.)

          iteration 1000: train_error=1.022531, test_error=1.0622330

Out[20]: [{'train_error': 1.0225307, 'test_error': 1.0622331}]
```

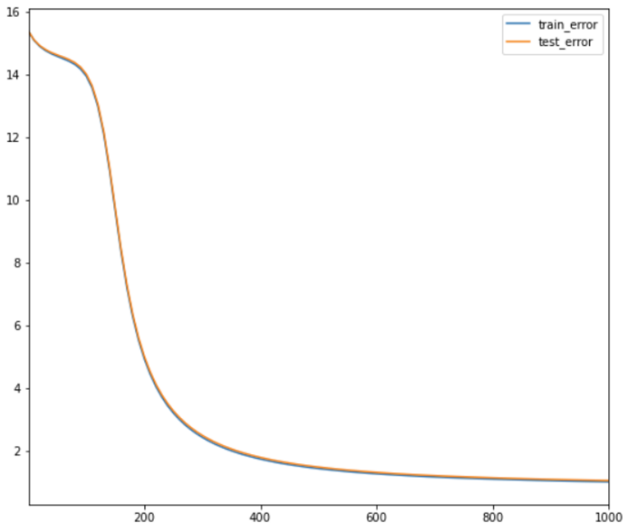

Fig. 1. Screenshot of training and testing MSEs.

From the plot, we observe that the two MSEs were high and dropped slowly at the beginning, after around 100 iterations they began to decline rapidly until about the 300th iteration, and kept decreasing slowly but steadily. The MSEs look to converge to a value close to 1, and we decided to stop the training process after 1000 iterations. It took us 35 minutes to train the final model (fortunately without crashing) using 24 CPUs and 32GB RAM.

Additionally, the model performance was evaluated by inspecting the embeddings. From possible techniques, computing the nearest neighbors according to cosine similarity and dot product was chosen. Dot product recommendations resulted in giving almost the same nearest neighbors for every movie. This is expected, and is explained by the fact that without any regularization norms of the embeddings (used to compute the dot product) are heavily correlated with their frequency. Whereas, cosine similarities tend to generate more or less meaningful results.

As we can see from Table 3, the nearest neighbors for one of the “Lord of the Rings” movies are other movies from the series. For movies “Carrie” (horror) and “Shrek” (family/comedy), neighbors also include some other movies of the same genre. We also observed that not all of the output movies have the exact same genre, which is expected because embeddings could be trained to catch not only genre but other related information too. In addition, our model was trained without any regularization, and therefore, favoring popular movies more.

Movie (Genre)	Lord of the Rings: The Two Towers
Nearest Neighbours	Lord of the Rings: The Return of the King The Incredibles Finding Nemo (Widescreen) Lord of the Rings: The Return of the King: Extended Version Lord of the Rings: The Fellowship of the Ring

Movie (Genre)	Carrie (**Horror**)	Shrek (**Family/Comedy/Fantasy**)
Nearest Neighbours	The School of Rock (Comedy/Music) Monster (Crime/Drama) The Omen (**Horror**) The Exorcist: Restored Version (**Horror**) Groundhog Day (Romance/Comedy)	Finding Nemo (**Family/Comedy/Fantasy**) Shrek 2 (**Family/Comedy/Fantasy**) The Bourne Supremacy (Action/Thriller) Spider-Man 2 (**Family**/Action/Adventure/**Fantasy**) Indiana Jones and the Temple of Doom (Adventure/Action)

TABLE. 3. Nearest Neighbors for different movies

As we can see from Table 3, the nearest neighbors for one of the “Lord of the Rings” movies are other movies from the series. For movies “Carrie” (horror) and “Shrek” (family/comedy), neighbors also include some other movies of the same genre. We also observed that not all of the output movies have the exact same genre, which is expected because embeddings could be trained to catch not only genre but other related information too. In addition, our model was trained without any regularization, and therefore, favoring popular movies more.

Conclusion / discussions:In summary, a wholesome analysis of the Netflix user-movie ratings dataset has been made. We have analyzed the data, identifying its distribution, both in general and for specific categories, by applying the concepts learned in the class. The experiments on Frequent Pattern Mining were conducted, and we were able to obtain the interpretable results where most frequent itemsets correspond to the movies from one series. Additionally, based on the collaborative filtering approach, a recommendation model has been trained. The resulting plot of decaying cost for the both training and test sets shows the successful training of the embeddings. The movie embeddings were further used to obtain the nearest neighbors for specific movies. It was concluded that the embeddings were able to catch important information such as movie genres. Adding regularization during the training is recommended for better results.

References:

[1] Netflix Movie Recommendation Notebook - https://www.kaggle.com/code/laowingkin/netflix-movie-recommendation

[2] “fpgrowth: Frequent itemsets via the FP-growth algorithm” documentation - http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/fpgrowth/

[3] Aggarwal, C. C. (2016). Recommender systems (Vol. 1). Cham: Springer International Publishing.

[4] Breese, J. S., Heckerman, D., & Kadie, C. (2013). Empirical analysis of predictive algorithms for collaborative filtering. arXiv preprint arXiv:1301.7363.

[5] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017, April). Neural collaborative filtering. In Proceedings of the 26th international conference on world wide web (pp. 173-182).

[6] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.

[7] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001, April). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web (pp. 285-295).

[8] Colab: Build a Movie Recommendation System.

https://developers.google.com/machine-learning/recommendation/labs/movie-rec-programming-exercise

Appendix:

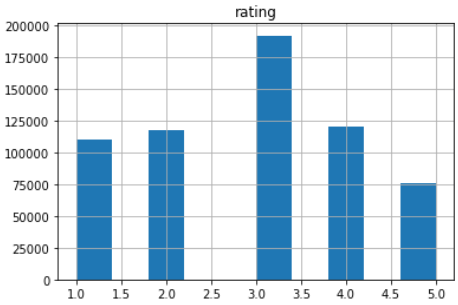

Fig. 1. Rating distribution of movies that received few ratings (<=200)

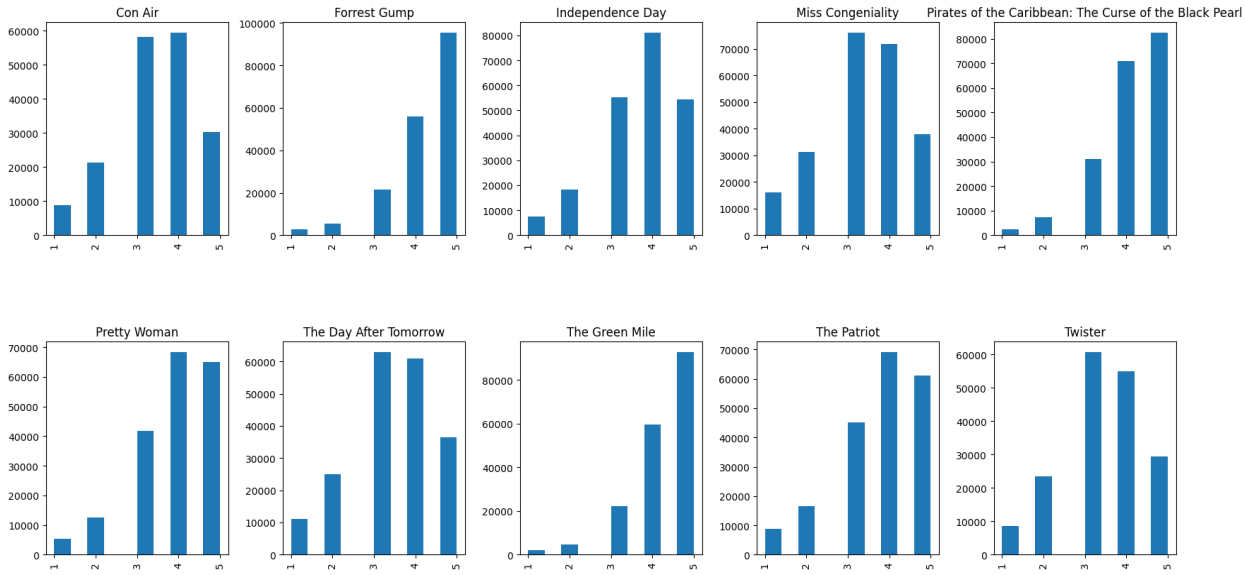

Fig. 2. Rating distributions of the top 10 most popular (most rated) movies

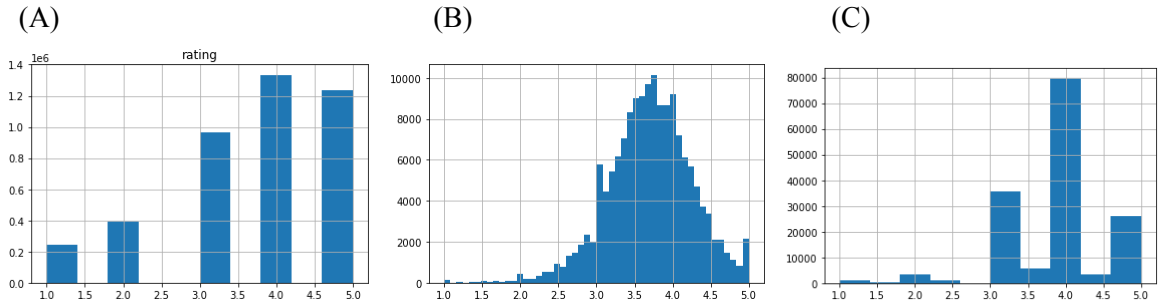

Fig. 3. (A) Distributions from users who rated a few movies (<=50). **(B)**Distribution of mean ratings from users who rated a few movies (<=50). **(C)**Distribution of median ratings from users who rated a few movies (<=50).

(A) (B) (C)

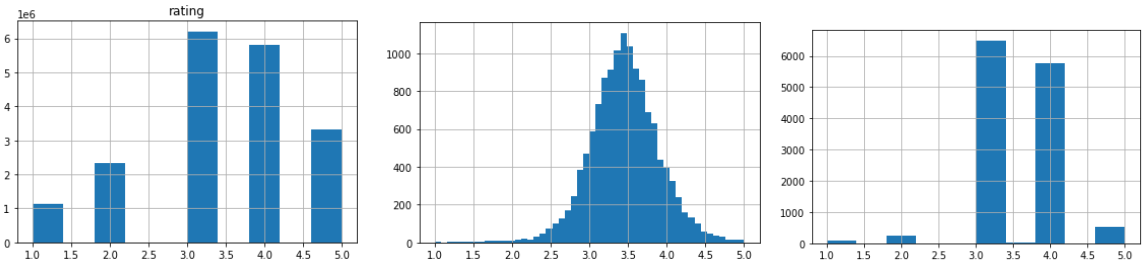

Fig. 4. (A) Distributions from users who rated many movies (>=1000). **(B)**Distribution of mean ratings from users who rated many movies (>=1000). **(C)**Distribution of median ratings from users who rated many movies (>=1000).

```
[53] from mlxtend.frequent_patterns import fpgrowth
     patterns03 = fpgrowth(trans, min_support=0.3)
     patterns03
```

	support	itemsets
0	0.626215	(518)
1	0.624885	(81)
2	0.587435	(191)
3	0.565333	(765)
4	0.549678	(182)
...	...	...
809	0.322419	(689, 502)
810	0.302261	(104, 502, 527)
811	0.322010	(780, 518)
812	0.306047	(780, 191)
813	0.302875	(675, 780)

814 rows × 2 columns

Fig. 5. Screenshot of the FPGrowth execution

```
[54] #this given not movie_ids but indices
     patterns03[patterns03.itemsets.map(len)==4].sort_values(by='support', ascending=False).head(3)
```

	support	itemsets
618	0.374706	(656, 81, 104, 527)
196	0.346055	(656, 689, 104, 527)
605	0.345237	(656, 104, 527, 191)

```
[55] freq_m = []
     for elem in [656, 81, 104, 527]:
       freq_m.append(int(list(trans.columns)[elem]))
     print(freq_m)

     [14240, 1905, 2452, 11521]
```

Fig. 6. Screenshot of the most frequent 4-itemset (min_sup=0.3)

top 10 most popular movies			
movie title	movie year	movie_id	Mean rating
Forrest Gump	1994	11283	4.29991
Twister	1996	12470	3.41187
The Patriot	2000	14313	3.783854
Independence Day	1996	15124	3.724238
The Day After Tomorrow	2004	15205	3.442166
Con Air	1997	16242	3.454411
The Green Mile	1999	16377	4.306941
Pirates of the Caribbean: The Curse of the Black Pearl	2003	1905	4.153908
Miss Congeniality	2000	5317	3.361267
Pretty Woman	1990	6287	3.905047

Table. 1. Statistics of the rating distributions of top 10 most popular movies