# Movie Rating Prediction and Recommendation

Abdulkadir Çelikkanat

2014700045

## Abstract

This paper demonstrates an implementation of various movie rating prediction algorithms for MovieLens 100k dataset[1]. The baseline predictor, kNN, SVD, asymmetric-SVD, SVD++ algorithms are used to estimate unknown ratings. For the evaluation of the algorithms, the Root Mean Squared Error(RMSE) is used.

## Introduction

The study of recommender systems has began to arise in the mid-1990s as an independent research area and it is new research area compared to other classical information system tools and techniques(e.g databases or search engines)[2]

In recent years, the interest in recommender systems has been dramatically increasing, and they has started to play an important role in highly rated web sites such as Amazon.com, YouTube, Netflix, and IMDb[2]. For instance, Netflix, the online movie rental service, awarded a million dollar prize to the team that first succeeded in improving substantially the performance of its recommender system.[3]

In this paper, I have implemeneted an item-based collaborative filtering algorithm and latent factor algorithms to predict movie ratings.

## Dataset Description and Preprocessing

In this project, I have prefered to use the MovieLens 100k dataset[1] which consists of 100,000 ratings between 1 and 5 from 943 users on 1682 movies, and each user has rated at least 20 movies.

The data sets ua.base, ua.test, ub.base, and ub.test split the u data into a training set and a test set with exactly 10 ratings per user in the test set. The sets ua.test and ub.test are disjoint. For the kNN and baseline predictors algorithm, ua/ub.base and ua/ub.test sets are used and for the other algorithims, only the ua.base and ub.base sets are preferred.

## Performance Evaluation

For the evaluation of algorithms' performance, I have used root *mean squared error (RMSE)* defined as:

$$RMSE = \sqrt{\frac{1}{S_{test}} \sum_{(u,i)\in S_{test}} (r_{ui} - \hat{r_{ui}})^2}$$

where $r_{ui}$ is the rating given by the user u for the movie i, and $\hat{r_{ui}}$ is the predicted rating.

## Algorithms

### k-Nearest Neighbour (kNN)

I have implemented kNN algorithm as an item-based collaborative filtering algorithm. Item-based approach looks into the set of items an user has rated and computes how similar they are to the item i and select k most similar items. For the calculation of similarity between items, the *Adjusted Cosine Similarity*[4] is used:

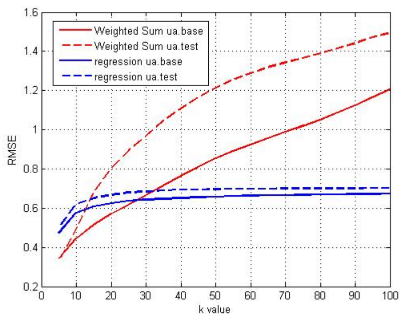
$$sim(i,j) = \frac{\sum_{u\in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u\in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u\in U}(R_{u,i} - \bar{R_u})^2}}$$

where U is the set of users, $R_{u,i}$ is a rating that user u has rated for movie i, and $\bar{R_u}$ is the average of the u-th user' ratings.

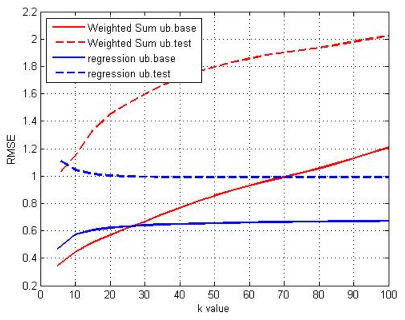
After finding similarities between items, or movies, *Weighted Sum* is used for prediction computation as

$$p_{u,i} = \frac{\sum_{k\ similar\ items}(s_{i,k} * R_{u,k})}{\sum_{k\ similar\ items}(|s_{i,k}|)}$$

where $p_{u,i}$ is prediction and $s_{i,j}$ corresponding similarity between i and j.



*Regression*
This approach is similar to the weighted sum method but instead of directly using the ratings of similar items it uses an approximation of the ratings based on regression model.[4]



## Baseline predictors

Most of the observed ratings are, independently of their interaction, due to effects associated with either users or items. For instance, for some movies receive higher rating than the other movies or some users have tendencies to give higher ratings than the others. Hence, the *baseline predictors*[2] is defined as

$$b_{ui} = \mu + b_u + b_i$$

where $b_{ui}$ is the baseline prediction for an unknown rating $r_{ui}$, $\mu$ is the overall average rating, and $b_u$ and $b_i$ indicate the observed deviations of user u and item i, respectively.

| | ua.base | ua.test |
|---|---|---|
| RMSE | 0.9345 | 0.9908 |
| | ub.base | ub.test |
| RMSE | 0.9333 | 1.0081 |

## Latent Factor Algorithms

Latent factor models, such as SVD, includes an alternative approach by transforming both items and users to the same latent factor space. The latent space tries to explain ratings by characterizing both movies and users on factors derived from user feedback.[2]

## Singular Value Decomposition (SVD)

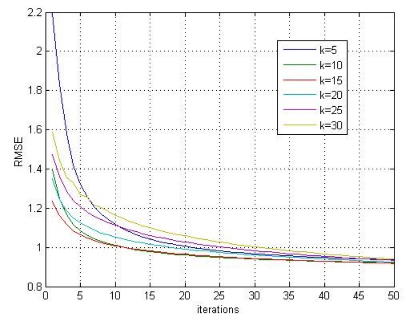
Each item i is associated with a vector $q_i \in \Re^f$, and each user u is associated with a vector $p_u \in \Re^f$. For a given item i, the elements of $q_i$ measure the extent to which the item possesses those factors, positive or negative. A rating can be predicted by adding baseline predictors as

$$\hat{r_{ui}} = \mu + b_i + b_u + q_i^T p_u$$

Standart SVD asssumes all missing values are zero, so this leads bad prediction accuracy when the matrix is highly sparse. Hence the *stochastic gradient method* is used to learn factor vectors as decribed in [4]

$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$
$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$
$q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i)$
$p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u)$

where $e_{ui} := r_{ui} - \hat{r_{ui}}$. $q_i$ and $p_u$ has been initialized as a random vectors, and learning rates $\lambda$, $\gamma$ are choosen as $\gamma = 0.0002$ and $\lambda = 0.005$ after several trials.

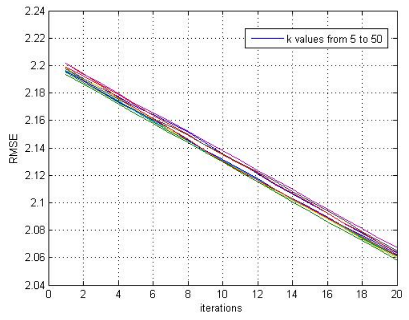


### Asymmetric SVD

The SVD model can be extended by considering also implicit information. Instead of providing an explicit parameterization for users, users can be represented through the items that they prefer.[5]

$\hat{r_{ui}} = b_{ui} + q_i^T(|R(u)|^{-1/2}\sum_{j \in R(u)}(r_{ui} - b_{uj})x_j + |N(u)|^{-1/2}\sum_{j \in N(u)} y_j)$

Here $R(u)$ contains all the items for which ratings by u are available, N(u) all items for which u provided an implicit preference (movies that are rented/purchased/watched), and each item i is associated with three factor vectors $q_i, x_i, y_i \in \Re^f$ For this model, learning parameters has been chosen as $\lambda = 0.02$ and $\gamma = 0.00002$.

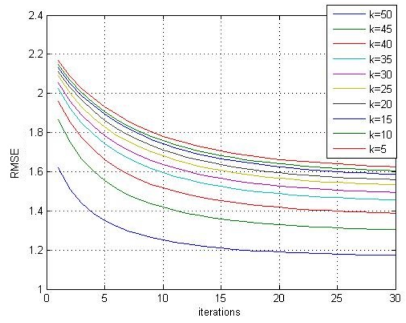


### SVD++

Integration of implicit feedback is concerned, we could get more accurate results by a more direct modificaiton of $b_{ui} + q_i^T p_u$, leading to the following model.[5]

$\hat{r_{ui}} = b_{ui} + q_i^T(p_u + |R(u)|^{-1/2}\sum_{j \in R(u)} y_j)$

A second set of item factors are used to characterize users based on the set of items that they rated. The set $R(u)$ contains the items rated by user u. Learning parameters has been chosen as $\lambda = 0.02$ and $\gamma = 0.0002$



## Conclusion

In this paper, I have applied kNN algorithm with regression and weighted sum techniques, baseline predictors, SVD, asymmetric SVD, and SVD++ algorithms to the MoviLens 100k dataset. kNN algorithm with regression model has given better results than the weighted sum technique on the ua.base, ub.base subsets and on the disjoint ua.test and ub.test sets. Then, the latent factor models have shown that SVD++ is better than asymmetric SVD, and also the results of the SVD++ and SVD model are close to each other. For the performance evaluation of the algorithms, I have used root mean squared error.

## References

[1] http://grouplens.org/datasets/movielens/

[2] Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor, Recommender Systems Handbook

[3] http://www.netflixprize.com/

[4] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., Item-based Collaborative Filtering Rec-

ommendation Algorithms, Proc. 10th International Conference on the World Wide Web, pp. 285-295, 2001.

[5] Koren, Y., Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model, Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008.

[6] Koren, Y., Factor in the Neighbors: Scalable and Accurate Collaborative Filtering , ACM Transactions on Knowledge Discovery from Data (TKDD),4(2010):1-24.

[7] Zhouxiao Bao, Haiying Xia, Movie rating estimation and recommendation,2012

[8] Hans Bystrm, Movie Recommendations from User Ratings,2012

[9] stat.wikia.com/wiki/Matrix factorization

[10] mahout.apache.org/users/recommender/matrix-factorization.html

[11] quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/