Recommender Systems

Testing Recommender Systems with Movie Ratings

Fawad Ali
Computer Science
California State University, Fullerton
Fullerton, California
fawad.ali@csu.fullerton.edu

1 Introduction

Recommender Systems are algorithms used to recommend an item to a user. They can be used to recommend movies, tv shows, movies and various products. A Recommender Systems can be put into the category of collaborative filtering, continent-based filtering, or hybrid filtering.

In a collaborative filtering system items are recommended to a user based on other users who rated items similar to them. This is the most commonly used category [1].

In a content based filtering approach, items are recommended to a user based on the description of an item and the user's previous choices [1].

Hybrid recommender systems involve using more than one filtering method. This method is used to deal with problems such as the cold start problem, over specialization problem, and the sparsity problem [1].

For this project, I plan on researching many different types of recommender systems and then applying those methods on my dataset. After that, I plan on comparing the performance of those systems by using root mean squared error. The dataset I will be using movie ratings by users.

2 Dataset

The dataset I used to test each recommender system is from GroupLens, which is a research group from the University of Minnesota at the department of Computer Science and Engineering. The dataset contains 100836 ratings from 610 users and 9742 movies. Each user has at least 20 ratings. One of the files in the dataset is called ratings.csv. This file contains all the user ratings. The columns are userId, movieId, rating, and timestamp. The ratings are between 0.5 to 5.0 in .5 increments. Timestamp is the time the rating was made . It is represented in the seconds after January 1, 1970 midnight in UTC time [2].

The dataset contains another dataset called tags.csv. In this file, there are columns for userId, movieId, tag, and timestamp. The tag column is a word or phrase. The meaning of each tag is unique to the user [2].

The last file in the dataset is called movies.csv. The columns in this file are movieId, title, and genres. Genres include a list of genres of the movie. The genres include action, animation, comedy, drama, horror, and more.

The recommender systems I tested involved using a matrix with each row representing a user and each column representing an item. To get the ratings in this format, I used the pivot_table function from the pandas library in Python. For the parameters I set index equal to userId, columns equal to movieId, and values equal to rating. I ended up with a dataframe where the column is movieId and the index is userId. Each value in the dataframe is the rating the user gave to the movie. If a user did not rate the movie, then the value would be NaN. I replaced NaN values with 0 and saved the dataframe to a CSV file called user_rating.csv.

3 Recommender System Techniques

These are the recommender system techniques I have applied to my dataset. The techniques are K-Nearest Neighbor with users, K-Nearest neighbors with items, Basic Matrix Factorization, and Matrix Factorization with Bias. Each technique involves using a m x n matrix R. Where m is the number of users and n is the number of items. The value at row u and column i is the rating user u gave to item i.

3.1 K-Nearest Neighbors with Users

Using KNN with users involves using each row in matrix R as a vector. To predict the rating of item i with user u, the first step is to find the K most similar users to user u [1]. The method I used to calculate the similarity between two users is cosine similarity. This is given by the following equation.

$$(1)\ \text{sim}(x, y) = \frac{x \cdot y}{||x||_2 \times ||y||_2} = \frac{\sum_{s \in Sxy} r_{x,s} \cdot r_{y,s}}{\sqrt{\sum_{s \in Sxy} r_{x,s}^2} \sqrt{\sum_{s \in Sxy} r_{y,s}^2}}$$

In the function above x and y are vectors that contain the ratings the users gave to items. S_{xy} is the set of ratings that both x and y have given to a specific item [3].

Once the K most similar users are found, the next step is to predict the rating for item *i*. This equation is the approach I used to predict the rating.

$$(2)\ \hat{r}_{u,i} = \overline{r_u} + k \sum_{u' \in \hat{U}} sim(u, u') \times (r_{u',i} - \overline{r_{u'}})$$

In this equation, $\hat{U}$ is the set of users that are the K most similar to user u and that have also rated item *i*. The values $\overline{r_u}$ and $\overline{r_{u'}}$ are the average rating by user u and user u' respectively. The value k = $1 / \sum_{u' \in \hat{U}} \left| sim(u, u') \right|$.

3.2 K-Nearest Neighbors with Items

To apply K-Nearest Neighbors we use the columns of the matrix R as vectors. The first thing that needs to be done is to find the K most similar items and their similarities [1]. I used equation 1 to calculate the similarities between different items.

Once the K most similar items are found and their similarities, we can predict a rating. To predict a rating I used the following equation:

$$(3)\ \hat{r}_{u,i} = \frac{\sum_{x \in \hat{I}} sim(x, i) \times r_{u,x}}{\sum_{x \in \hat{I}} |sim(x, i)|}$$

In this equation $\hat{I}$ is a subset of K similar items. The items in $\hat{I}$ only include the items that user *u* rated [4].

3.3 Basic Matrix Factorization

In basic matrix factorization, we have to find a matrices $P \in \mathbb{R}^{m \times f}$ and $Q \in \mathbb{R}^{n \times f}$ such that $R \approx PQ^T$. To find P and Q, we would have to minimize the following function:

$$(4)\ \min_{P,Q} \sum_{(u,i) \in S} (r_{u,i} - p_u q_i^T)^2 + \lambda \left(\|p_u^T\|^2 + \|q_i^T\|^2 \right)$$

In this equation $r_{u,i}$ is the element in R that is in the u-th row and i-th column, p_u is the u-th row in P, and q_u is the u-th row of Q. The λ is the regularization term, it is used to prevent overfitting. S is the set of known ratings in R [5].

To find P and Q from equation 4, we loop through the indices in S multiple times while updating the values for p_u and q_u at each index by using these equations [5]:

$$e_{ui} = r_{ui} - p_u q_i^T$$
$$p_u \Leftarrow p_u + \alpha(e_{ui}\, q_i - \lambda p_u)$$
$$q_i \Leftarrow q_i + \alpha(e_{ui}\, p_u - \lambda q_i)$$

The α is a constant that is used to move each row proportional to it in the opposite direction of the gradient [5].

3.4 Matrix Factorization with Bias

An issue with basic matrix factorization is that it does not take user and item bias into account. Different users rate movies differently [5]. For example, some users are really generous with their ratings and rate many movies 5 out of 5. Other users have really high standards for movies they rated 5 out of 5. Item biases are needed because some items are considered better or worse than others [5]. The equation used to predict the rating of item *i* by user *u* is given by:

$$(5)\ \hat{r}_{ui} = \mu + b_i + b_u + p_u q_i^T$$

This is similar to how we would predicate a rating for basic matrix factorization, except we add three values to each rating prediction. The new values are μ, which is the average rating of all items in the dataset. b_i is the bias for item *i* and b_u is the bias for user *u* [5]. To find the bias values and the p_u and q_i values, we need to minimize the following equation [5]:

$$(6)\ \min_{P, Q, b_i, b_u} \sum_{(u,i) \in S} \left(r_{ui} - \mu - b_i - b_u - p_u q_i^T \right)^2 + \lambda \left(\|p_u\|_2 + \|q_i\|_2 + b_i^2 + b_u^2 \right)$$

3.5 Matrix Factorization with Temporal Dynamics

A problem with other matrix factorization techniques is that they do not take into account how people change overtime. For example, people's interests change. The way they rate items can also change. For example a person may regularly rate movies 10/10 stars, but in the future, they may not be so lenient with their ratings. Also some items may be rated very differently at some point due to popularity. To take time into account we use Matrix

Factorization with bias, but the P matrix, item bias and user bias are turned into a function of time. The Q matrix is not turned into a function of time because items do not change like people do [5].

$$(7)\ \hat{\mathrm{r}}_{ui}(t) = \mu + b_i(t) + b_u(t) + p_u(t)q_i^T$$

3.6 Naive Bayes

A way to recommend an item is to use Naive Bayes Classifier. The equation is given below [6].

$$(8)\ \max_{c \in C} p(c) \prod_{i=1}^{n} p(f_i|c)$$

In this equation C is all the possible classes, and f_i represents a feature. The classes can be ratings and features can be the genres of movies, keywords, and actors [6].

To measure the probabilities we use these equations [6].

$$(9)\ P(c) = \frac{n_c + n_f}{n_t}$$

$$(10)\ P(o|c) = \frac{n_o + 1}{n_c + n_f}$$

The variable n_c is the number of times n has been observed, n_f is the number of features, n_t is total number of observations, and n_o is the time o has been observed with class c. P(c) and P(o|c) are in a form that prevents a probability of 0 [6].

3.7 TF-IDF

TF-IDF is short for term frequency inverse document frequency. It is used to show how important a term is for a document. TF-IDF is given by the following equations [3].

$$(11)\ TF_{t,d} = \frac{f_{t,d}}{\sum_{i=0}^{n} f_{t_i,d}}$$

$$(12)\ IDF_t = \frac{N}{n_t}$$

$$(13)\ TF - IDF_{t,d} = IDF_t \times TF_{t,d}$$

$f_{t,d}$ is the number of times the term t appears in document d. N is the total number of documents and n_t is the number of documents that contain the term t. Each item will have a vector in which each element in the vector corresponds to the TF-IDF of a term [3]. Equation 1, which is cosine similarity can be used to compare similarity between items [3].

4 Results

In this section I will be going over the results of the recommender systems in the previous section. All of my work was done using Python.

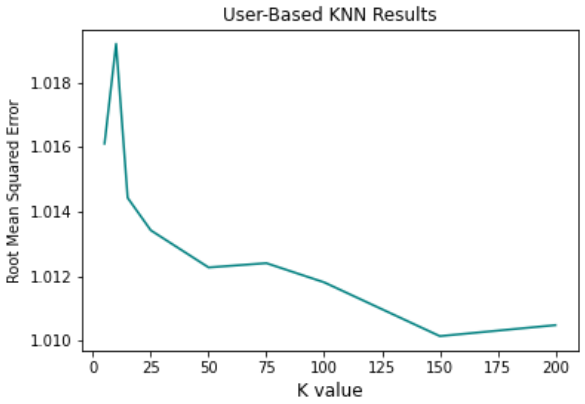
The training set was 80 percent of the known values and the testing set was the remaining 20 percent of the data. To split the data into a training and testing set, I first put the indices of the known ratings in a list. Then I used the shuffle function from the random library to shuffle the data. After that I put the first 80 percent of the data in a training list and the remaining data was assigned to a testing list.

I test each recommender system by trying different parameter values of each system. I graphed the results using matplotlib.

4.1 KNN Based on Users

I implemented user-based KNN by making a copy of the rating matrix and replacing each value in the training set with 0 to make it look like a rating that does not exist. I used the cosine_similarity from sklearn on the copied matrix to get the cosine similarity matrix of each user. Then I changed the diagonal values to 0 and used the matrix to find the K most similar users of each user in the testing set. After that, I used equation 2 to predicate each index in the testing set.

To test K-Nearest Neighbors with users, I used K values of 5, 10, 15, 25, 50, 75, 100, 150, and 200.



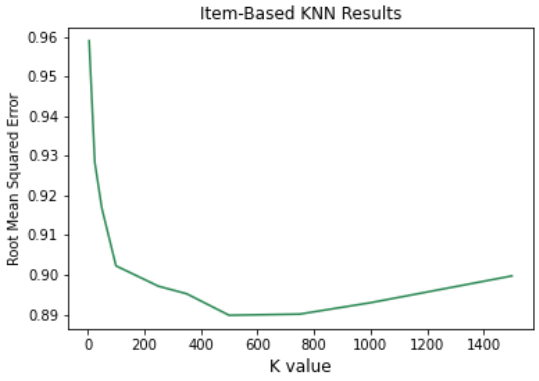
From the graph, we can see that the root mean squared error is decreasing from k=5 to k=150. Afterwards, there is a slight increase in the RMSE. The RMSE was the lowest at k=150 with a RMSE of 1.010139.

4.2 KNN Based on Items

The implementation of item-based KNN involved copying the rating matrix and setting each value in the testing set to 0. I used

the cosine_similarity function with the transpose of the new rating matrix to get the cosine similarity matrix based on items. I set the diagonal values to 0 and then found K most similar movies to each of the movies in my testing set. I then used equation 3 to predicate the ratings of the testing set.

I tested item-based K-nearest neighbors with K set to 5, 25, 50, 100, 150, 250, 350, 500, 750, 1000 and 1500.

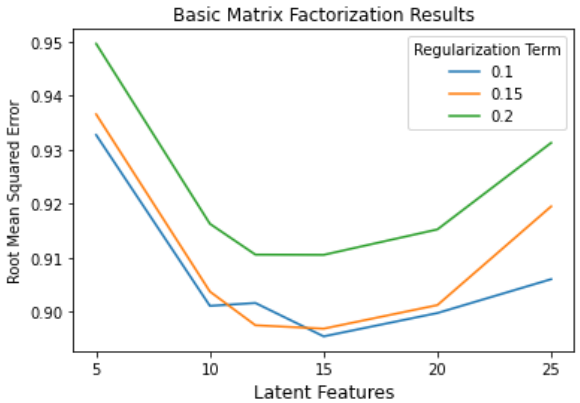


In the graph we can see that there is a big drop in RMSE from K=5 to K=100 and then there is a smaller drop from K=100 to K=500. After that, the RMSE increases. The RMSE is the lowest at K=500, with a value of 0.889785.

4.3 Basic Matrix Factorization

Implementing basic matrix factorization involved using the equations in section 3.3. I trained the model by looping through the indices in the training set and used those equations to update each row.

To test basic matrix factorization, I used 5, 10, 12, 15, 20, and 25 for the latent features and I used 0.1, 0.15, and 0.2 for regularization terms.

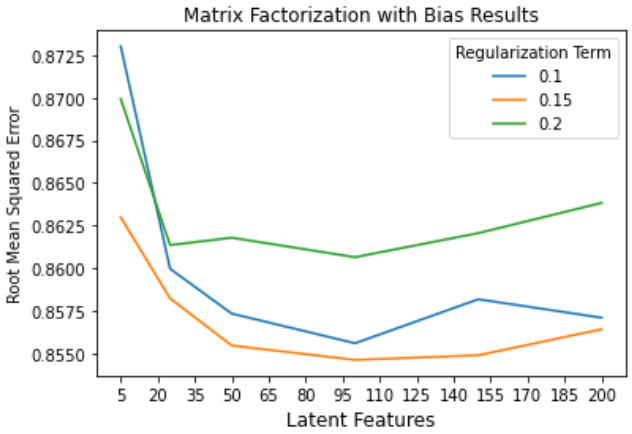


From the graph we can see that a regularization term of 0.2 did not perform as well as the others. The lowest RMSE was with a latent feature of 15 with a regularization term of 0.1. The value was 0.895426.

4.4 Matrix Factorization with Bias

I implemented matrix factorization with bias similar to the way I implemented basic matrix factorization. The difference was that e_{ui} contained the biases and the mean ratings and the biases were updated as well.

To test matrix factor factorization with bias, I used latent features of 5, 25, 50, 100, 150, and 200. The regularization terms that were used were 0.1, 0.15, and 0.2.



From the graph we can see that a regularization term of 0.2 had the highest RMSE after 25 latent factors. A regularization term of 0.15 performed the best at each K value. The lowest RMSE was 0.854609. This was at 150 latent features and with a 0.15 regularization term.

4.5 Matrix Factorization with Temporal Dynamics

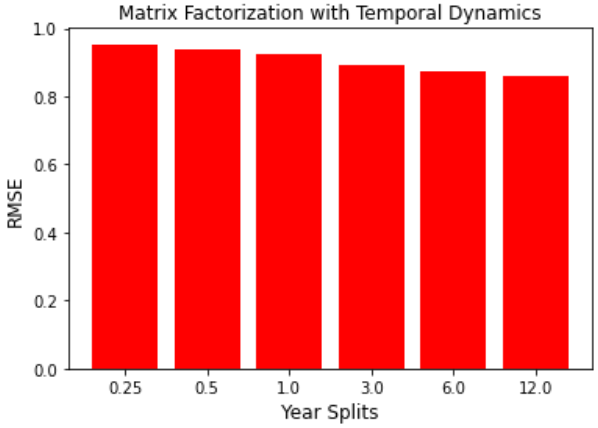
The dataset I used contained the time in when the rating was made. The format it was in were the seconds after midnight January 1, 1970 [2]. To get the t values, I split the data into different time periods and each time period had its own biases and P matrix. For example if I split the data into 365 days, then the ratings that occurred in the from the first rating to the next 365 days had its own biases and P matrix and the next 365 days had different biases and P matrix and so on.

I created a 2-dimensional numpy array, in which the rows were the users and the columns were the movies and the value at (i,j) was the timestamp user j rated movie i. If the rating did not exist then the value was zero.

To get the t values, I first got the lowest timestamp in the data and subtracted every timestamp by it. The function I created for this algorithm had a days parameter which was the amount of days I wanted to split the data by. I created a new array by using floor division to divide each timestamp by the days in seconds. I then created lists of biases and P matrices of the size of the maximum number in the new array + 1. Each value (i, j) in the new array was used as an index for each list.

To train each parameter I kept the regularization term at 0.15, which had the best results with Matrix Factorization with Biases. I used latent factors of 15, 25, 50. 100, and 150. For days I used 0.25, 0.50, 1, 3. 6, and 12 years. The years were converted to days when imputed to the function.

The graph below shows the best result at each time period. The best result was at 12 years and a latent factor of 50 with a RMSE of 0.858889. From the graph we can see that the RMSE increases as the time period decreases. This was not the results I expected. This algorithm performed the best from the paper I got it from [5]. I think the result could have been something to do with my dataset, because it only has 100 thousand ratings and the ratings are from a span of over 22 years. Since I used 20 percent of the data for testing, I only used 80 thousand ratings for training and the smaller the time period I use, the smaller the datasets get for each bias and P matrix. Also, the paper did not specify how they used time, so they may have done something different than I did.



4.6 Naive Bayes Classifier

For Naive Bayes Classifier, I used the possible ratings for the classes. The ratings ranged from 0.5 to 5 in 0.5 increments. For the features I used the genres. There were 20 possible genres and a movie could have had one or multiple genres.

To predict a rating user u gave movie i, n_c was the number of times user u rated a movie with a rating c, n_f was the number genres was 20, n_t was the total number of ratings the user made, and n_o was the number of times the user rated an item with a rating of c when the feature was o.

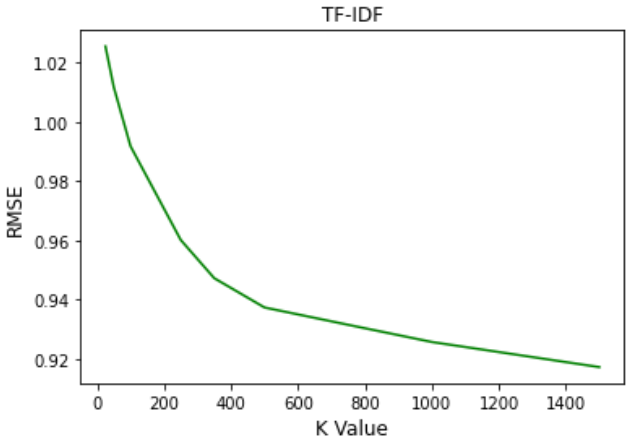
To create the algorithm, the first thing I did was make a pandas dataframe in which the index were the users and columns were the possible ratings. Each cell was the approximate probability a user rated a movie the column rating. For each user, I made a dataframe in where the index were the possible genres and the columns were the possible ratings and cells calculated p(o|c) for the user. These dataframes were stored in a dictionary. I used these probabilities to predict a rating using equation 8. The result was a RMSE of 1.0695.

4.7 TD-IDF

To calculate TD-IDF, I used the movies as a document and genres as the terms. I made a dataframe in where index were movies and the columns were genres. Each cell was the TD-IDF for a movie and genre. I used cosine similarity on the dataframe to calculate the similarities between the movies.

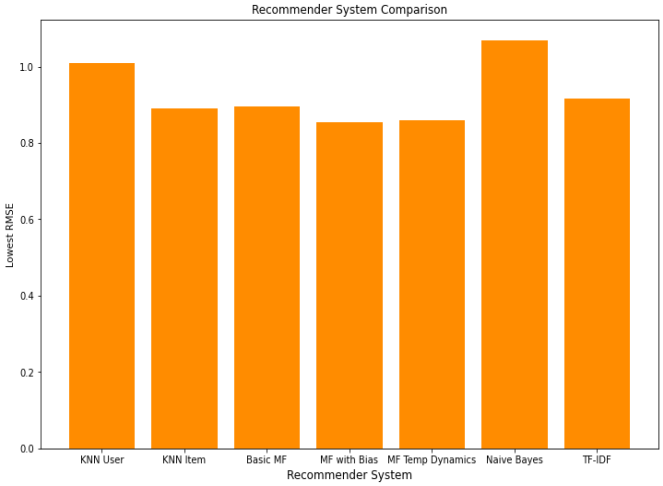
To predict a rating, I used KNN to find the K most similar movies. I then used the subset of the K movies the user rated to predict the rating. The equation I used to predict the rating was equation 3, which is the weighted average.

The K values I used were K=25, 50, 100, 250, 350, 500, 750, 1000, and 1500. From the graph we can see that the RMSE for the testing set kept decreasing as K increased, which I found surprising. The lowest RMSE was 0.917247, which is pretty good considering other user ratings were not used to make the prediction.



4.8 Comparison

The following graph shows the lowest RMSE produced by each recommender system.



From the graph, we can see that Naive Bayes performed the poorest and it's RMSE value was decently higher than the others. Matrix factorization with bias performed the best and the RMSE for temporal dynamics was a close second, but this model was just like two different models of matrix factorization with bias. Although TF-IDF did not perform as well as the other methods, it did well considering it did not use the ratings of other users.

5 Future Work

For CPSC 597, I would like to continue working on Recommender Systems, but I would like to work on it with some other item other than movies. I learned a lot from this project and I think I could learn more using a different item, because I think there might be other methods that work better for different types of items.

REFERENCES

[1] Thorat, P. B., Goudar, R. M., & Barve, S. (2015). *Survey on collaborative filtering, content-based filtering and hybrid recommendation system.* International Journal of Computer Applications, 110(4), 31-36.

[2] F. Maxwell Harper and Joseph A. Konstan. 2015. *The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems* (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872

[3] G. Adomavicius and A. Tuzhilin, *Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions*, in IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 6, pp. 734-749, June 2005, doi: 10.1109/TKDE.2005.99.

[4] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, *Item-based collaborativefiltering recommendation algorithms* in Proc. 10th Int. Conf. on WorldWide Web, 2001, pp. 285–295

[5] Y. Koren, R. Bell and C. Volinsky, *Matrix Factorization Techniques for Recommender Systems*, in *Computer*, vol. 42, no. 8, pp. 30-37, Aug. 2009, doi: 10.1109/MC.2009.263.

[6] Rombouts J., Verhoef T.,. A Simple Hybrid Movie Recommender System.http://www.fon.hum.uva.nl/tessa/Verhoef/Past_projects_files/Eind_Rombouts_Verh oef.pdf.