

A Recommender System

Weighted Alternating Least Squares

Sandra Giorgetta | Data Science Intensive | 5/2/2017

Capstone Final Report

Table of Contents

[1 Introduction - Implementing a Recommender System 2](#_Toc482608113)

[2 The Data Set 2](#_Toc482608114)

[3 Data Clean Up and Organization 3](#_Toc482608115)

[3.1 Clean up 3](#_Toc482608116)

[3.2 Data Organization 3](#_Toc482608117)

[4 The Algorithm – Model Based Recommender System 4](#_Toc482608118)

[5 Initial Findings 5](#_Toc482608119)

[5.1 Distributions of Ratings 5](#_Toc482608120)

[5.2 User Engagement 7](#_Toc482608121)

[5.3 Tags File Analysis 10](#_Toc482608122)

[6 Experiment 10](#_Toc482608123)

[7 Results 11](#_Toc482608124)

[8 Further Study 14](#_Toc482608125)

# Introduction - Implementing a Recommender System

### 

There’s no doubt that we live in a data driven world and to be competitive in it, a business must have digital access to its consumers via computer and hand-held devices. Through the collection of information regarding the consumer’s behavior, businesses can now provide a more personalized experience when targeting its customers. It can utilize this information to measure performance, enhance marketing efforts and drive business decisions. One of the methodologies used for targeting specific users with relevant material such as a recommended purchase, subscription suggestion, or online advertisement is the recommender system. With the implementation of a recommender system, a company can provide a personalized online experience that has the ability to drive consumer engagement, increase sales, and improve business outcomes.

With an average of 2,577 films produced per year around the world, it is no surprise that a movie recommendation system would be a popular idea. For companies like Netflix, a movie rental service, a movie recommendation engine is a must. With the average person watching 20 to 30 movies a year, there has been the advent of a plethora of movie recommendation services such as RottenTomatoes, Taste Kid, Jinni, Criticker, Flixter, and Movielens, to name a few. Taste Kid not only provides a movie recommendation but goes further by adding recommendations for books, bands, music, and other items related to the movie. The Movielens organization has generously provided access to datasets for educational purposes to encourage the development of better recommendation engines. The recommender system presented and studied in this report utilizes one of the datasets donated by Movielens.

# The Data Set

The stable Movielens dataset to be studied in this report can be found at the website: <http://grouplens.org/datasets/movielens/>. This dataset consists of movie ratings collected from the website <http://movielens.com> between the dates of January 9, 1995 and October 16, 2016. The data includes ratings by 671 users and information of 9123 movies, which have at least one rating or tag. Timestamps of the rating and tag are provided as well as link information for each movie for two movie websites, [www.imdb.com](http://www.imdb.com) and [www.themoviedb.org](http://www.themoviedb.org). Specifically, the dataset studied is named “ml-latest-small”, generated on October 17, 2016. This is a very sparse data set due to the sheer number of movies in comparison to the number of movies rated by a user.

The dataset is composed of four files in csv format: movies.csv, ratings.csv, tags.csv, and links.csv. The movies file includes a movieId, title, and genre field for 9125 movies. The ratings file includes userId, movieId, rating, and timestamp. The tags file includes userId, movieId, tag, and timestamp. And finally, the links file includes movieId, and identifiers for links to the movies as listed on the websites www.imdb.com and www.themoviedb.org.

# Data Clean Up and Organization

## Clean up

Due to the fact that some of the data is hand entered and some of the data is scraped from the web, there are bound to be some not so ideal entries. For this dataset, several irregularities were discovered. There were duplicate entries with the same movie title with different movie Id’s. There were four movies for which no release date was included in the title. And there were two movies that are in fact “made for TV” movies and not what would be considered your traditional box office movie. The two website url identifier columns from the links file were checked for null data. The column with data from the www.imdb.com site was complete. However, there were several null entries in the column for the www.moviedb.org identifiers.

Although going back into the original csv file and making manual edits is laborious, it is felt that in a realworld scenario, it would be advisable to have a “cleaned up” version of the file ready for use at a later time. If at some time in the future, the potential client had the desire to revisit the dataset, this would be most efficient. For these reasons, when possible changes were made to the original file. However, for some cases such as changing null values to zeroes, the changes were made in the pandas dataframe itself.

## Data Organization

The data from the four csv files is merged into several different pandas dataframes depending on the requirements for the use of the data.

The first dataframe includes data from the ratings and movies files and is used to create a matrix, denoted as Q, where the columns are the movie Id’s, the rows are the user Id’s and the contents are the ratings. This matrix is the basis for the recommendation algorithm.

A second dataframe is the movies file merged with the links file. It contains the movie Id’s, the movie titles, the movie genres and the identifier information for creating the url link to information on the movie for the two different movie websites. The movieId in this dataframe is set as the index and this dataframe is used as a lookup table for retrieving the movie title and url that is the output of the recommender system.

The third dataframe is the ratings and movies files merged onto the tags file. It contains the userId, movieId, rating, tag, and timestamp of the tag. Some users provided multiple tags (in some cases, as many as 10) for the same movie. For this reason, the ratings and movies files needed to be merged onto the tags file as opposed to the other way around to preserve all of the tags. The use of the tag as a way of “inferring” user interest was considered as the act of creating a tag could indicate positive interest in the movie. A scan of the tags indicates that it would not be trivial to glean any information from the tags since they are user produced and somewhat random in how each user uses them. With some tags such as “dark”, “gloomy”, “violent”, it is clear that tagging the movie does not necessarily imply a positive interest in the movie. The information in the tags file is useful in a user-based or item-based collaborative filtering system. At this time, since the implementation of choice is a model based approach, the tags are not being used as part of the recommendation system.

# The Algorithm – Model Based Recommender System

Since there is not a unique way of implementing a recommender system, in what follows we briefly discuss the main possible approaches.

The two approaches for designing a recommender system are content-based filtering and collaborative filtering. The content-based approach requires specific knowledge of the features of an item and uses that information to recommend items with similar features. Collaborative filtering does not require prior knowledge of the features of an item. It builds a model based on users’ past preferences and uses that model to predict future preferences. There are two types of collaborative filtering algorithms: memory-based and model-based. Memory-based algorithms find a group of users that have similar preferences and use those preferences to make a prediction to a given user. A similarity measurement is used to find a correlation between users. The model-based algorithm uses a subset of the data as “training data” to learn a predictive model. There are advantages and disadvantages to the various approaches and algorithms. Some advantages of model-based collaborative filtering are:

1. No prior knowledge of the item or user is required
2. It handles the problem of sparsity (portion of matrix that is empty) better
3. Scalability. The model typically is smaller than the entire dataset so can be used even for very large datasets.
4. Are typically faster than memory-based
5. Can handle the problem of over-fitting

There are many model-based collaborative filtering algorithms. Several research articles indicate that the algorithm with the highest accuracy is matrix factorization. However, it has also been shown that the “best” algorithm for a given data set may depend on the nature of the data set itself. The sparsity of the MovieLens Data set lends it well to the use of the Alternating Least Squares Algorithm (See Appendix A for details).

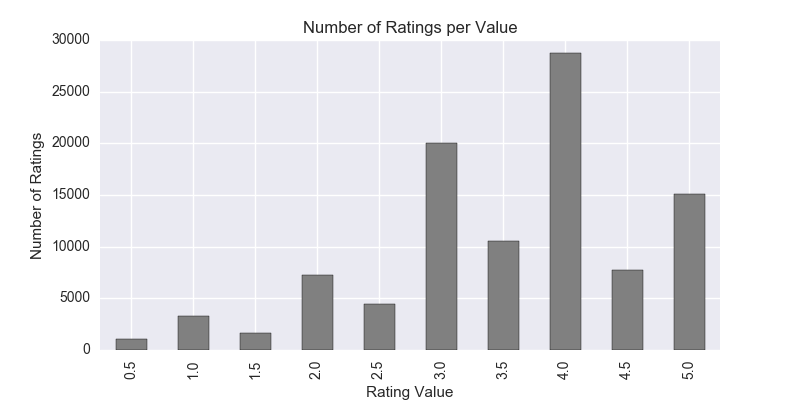
# Initial Findings

The data exploration phase of the project involved finding the answers to questions that would be helpful and relevant in making decisions regarding the design of the recommender system and in understanding trends in user engagement that would be helpful in driving business decisions for the potential client.

## Distributions of Ratings

After the recommender engine is implemented and the matrix of values for predicted ratings is defined, a decision needs to be made as to what values should be considered high enough for making purposeful recommendations. For this reason, it is helpful in investigating the distribution of ratings per value, the distribution of ratings per user, and the distribution of ratings per movie.

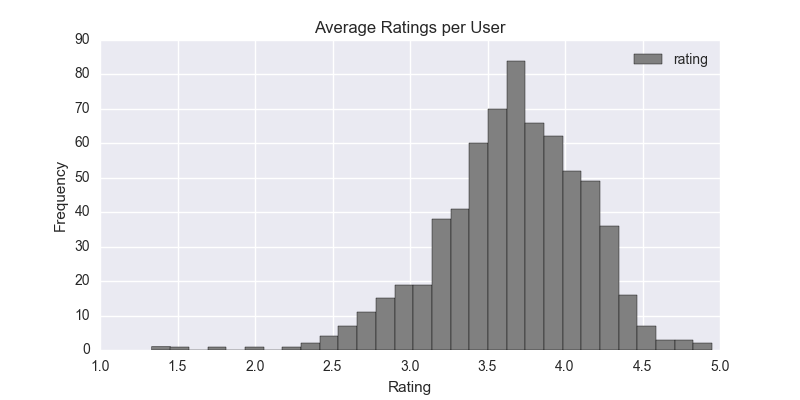
The rating values range from 0.5 to 5.0 in 0.5 increments. Figure 1 shows the distribution:



*Figure 1 Distribution of Ratings per Value*

The most frequent rating given was 4.0, followed by 3.0. The least most frequent rating was 0.5. More frequently than not, a higher rating is given than a lower rating. Is this because movie goers are not very discriminate in the way they rate movies? Or is it possibly the fact that a user is more likely to rate a movie they enjoy versus a movie they don’t enjoy?

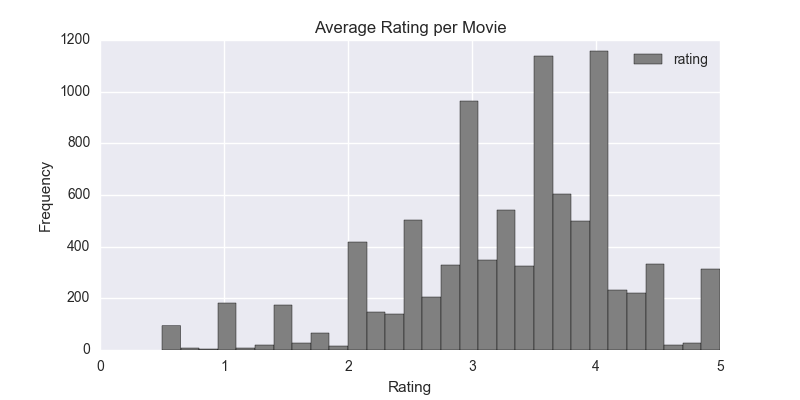
Knowing that there are more “high” ratings assigned than not, it may be helpful in knowing what the average rating is per user. To analyze the mean rating per user, the distribution was plotted and is shown in Figure 2:



*Figure 2 Average Rating per User*

As can be seen in the plot, the most frequent mean rating by a user was between 3.5 and 4.0.

Another piece of information that may be helpful is to look at the distribution of the average rating per movie. This distribution was plotted and can be seen in Figure 3. The distribution is not normal and is skewed to the left. More mean movie ratings tended to be higher than 2.5 than lower than 2.5.



*Figure 3 Distribution of Average Rating per Movie*

After the distribution of movie ratings has been analyzed, the “cut-off” point for deciding whether a predicted rating shall be considered positive or not can be made. One could choose an arbitrary value such as 2.5 to use as the “cut-off” in determining whether the predicted rating should be considered for a positive recommendation or not (only predicted ratings higher than the cut off will be output). Knowing that the model will be utilizing the data in the rating matrix and knowing that the ratings tend to be on the higher side, we can see that choosing a rating of 2.5 may not produce the most useful results. For this reason, we will use the mean as the cut-off point. Any movie with a predicted rating for the user that is greater than the mean rating will be considered for output.

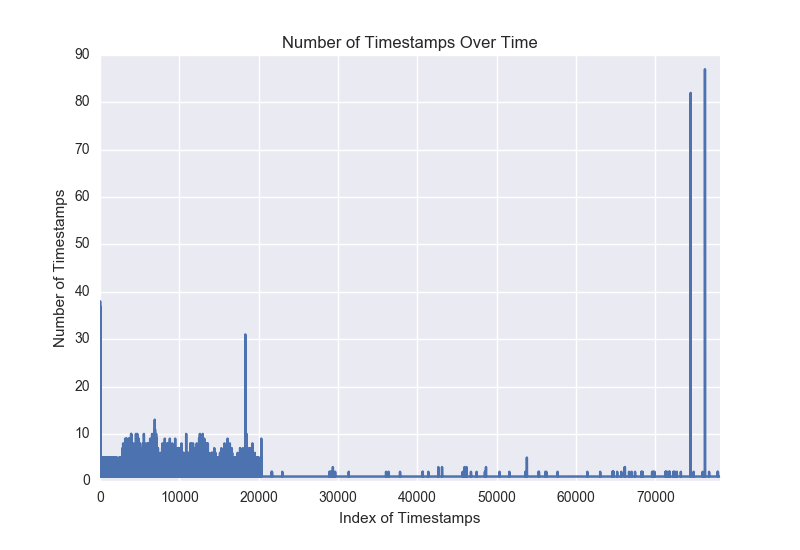
Knowing the sample size or the number of ratings a movie received is useful. If the movie has received at least 30 ratings, then the mean rating of the movie is likely to be the true mean of the population. The analysis showed that the most ratings received by a movie was 341 and the mean number of ratings was 11. For a larger data set where the mean number of ratings is larger, say above 30, then we could utilize statistical analsyis to compute probabilities of a movie receiving a given rating. These probabilities can be used to initialize the Y vector for a more hybrid approach.

## User Engagement

User engagement is a topic of concern to any business. The first item considered concerning user engagement is to know how many movies on average a user rated. We found that the largest number of movies rated by a user was 2391. The smallest number of movies rated by a user was 20 and the average number of movies rated by a user was 149. If the minimum number of movies rated were 30, then having a sample size of 30 we would use statistical analysis to determine the probability of a user assigning a given rating value. This information could be useful in creating a hybrid approach where the X vector is initialized with those probabilities as opposed to random assignments.

The next item of concern is the duration of engagement. The timestamp entries for the ratings were analyzed. The timestamp is the number of seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. The number of timestamps entered over time was plotted and is shown in Figure 4. The x-axis is the index of timestamps and the y-axis is the number of timestamps per timestamp value. There were a total of 78,141 timestamps over the approximately 21 years of data collection. As evidenced in the plot, the user engagement dropped off considerably around the timestamp corresponding with 20,000 which corresponds to the date of May 1, 2003. It was also observed that near the end of the data collection period, there was a large surge of engagement.

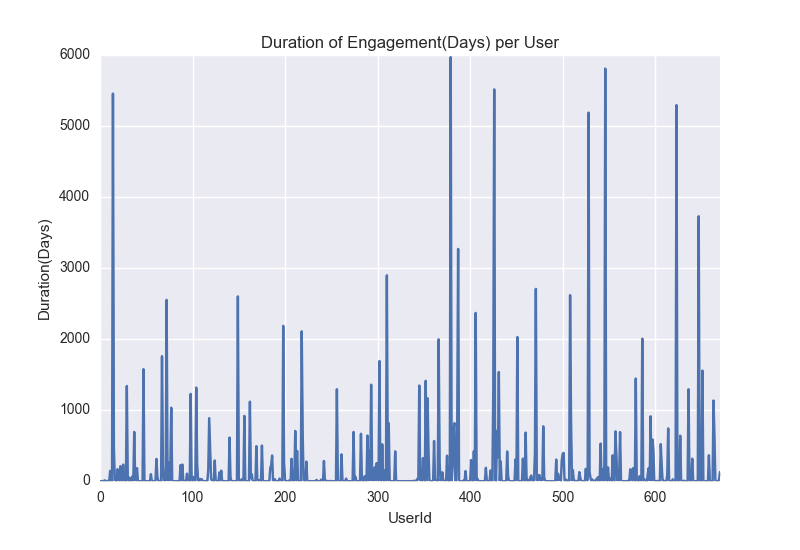
Another way of analyzing user engagement is to look at the duration of engagement by user. The minimum timestamp was subtracted from the maximum to determine the time of engagement in days per user. The data was plotted and can be seen in Figure 5. UserId 35 had the least duration of engagement which was 29 seconds while userId 380 had the longest duration of engagement which was 16.36 years. The average duration of engagement was 201 days which is approximately 7 months. The average length of



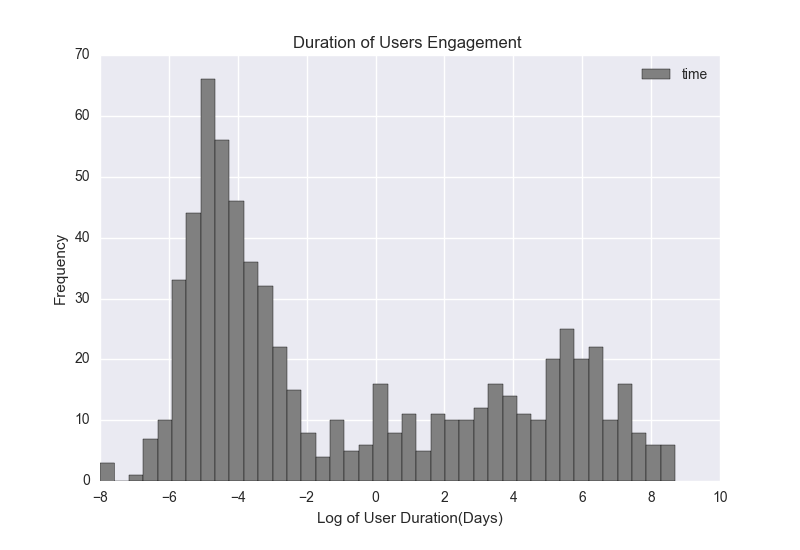
*Figure 4 User Engagement over Time*

engagement is most likely related to the fact that the movielens project is conducted in an educational setting where students are required to use the system during an academic calendar year.

Because there are some users whose duration was less than a minute and other users whose duration was over a twenty year period, it is useful to plot the log of the distribution to be able to view more detail. As a reminder, when looking at the plot, log0 corresponds to 1, log1 corresponds to 10, log2 corresponds to 100, and so on. The distribution of the log of the user duration in days was plotted and can be seen in Figure 6. There is a large proportion of users whose duration of engagement was less than 1 day (values <= 0).

The number of users that dropped out before the end of one year was 588. The number of users that stayed for at least one year and up to two years was 36, at least two years and up to three was 10, at least three years and up to four was 13, at least four years and up to five years was 5, and those staying over five years was 19. 

*Figure 5 Duration of User Engagement by UserId*



*Figure 6 Log of User Duration in Days*

## TAGS FILE ANALYSIS

The tags file was analyzed and considered as a vehicle for inferring user interest. There are 1296 tags, 582 of which are unique. There are 689 movies that have at least one tag provided by 61 users over a 10 year period beginning in January of 2006 and ending at the end of the collection process which was October 2016. Of the movies that were tagged, 266 were not rated. The maximum number of tags a movie received was 25 and the mean was 1.88. The maximum number of tags submitted by a user was 401 and the mean was 21.25. The tags information may be useful in a user-based collaborative filtering system. With the mean number of tags per movie less than two, using tags for an item-based collaborative filtering is not advised.

# Experiment

In addition to implementing the algorithm, five-fold cross validation (see Appendix B) is used for tuning the regularization parameter λ and the K value that determines the sizes of the X and Y factor vectors. The root mean squared error (see Appendix C) is calculated after each fold and then the parameters are re-adjusted. After the final test set run, the X and Y vectors will be used to create the Q\_hat matrix with the predicted ratings. The program looks for the predicted ratings that are above the average rating and have release dates within the last five years. If no predicted rating is found, the year is reduced by one until a rating is found that is above average. For each user, three items are printed; the title of one of the user’s top rated films (a user could have multiple movies with their top rating), the title of the user’s top recommended film regardless of release date, and the title of the user’s top recently released recommended film. For the recommended films, in addition to the title, a link to the www.imdb.com website is provided.

The code representing the first four folds of the five-fold cross validation will be timed. At a later date, the code will be run using Amazon Web Services and the results will be compared.

# Results

The initial values for the lambda and K values were chosen. Lambdas = [0.01, 0.1, 1.0]. K\_values = [10, 13, 16]. The middle value of K was chosen to be 1/10 the size of the training sets which contain 134 samples. The number of iterations was chosen as 12 to allow the program to complete overnight. The five-fold cross validation was executed. The time of execution of the first four folds was 23 hours, 38 minutes, and 12 seconds. After the final test run, the best parameters were identified as K=19 and lambda=0.0025. The RMSE results were shown to be better with larger K values and the smaller lambda values. The lowest validation run RMSE was 0.0967 while the lowest RMSE for the test run was 0.089. Here are the RMSE outputs of the four validation test runs:

{0: [0.7582043920272664,

0.755395539204102,

0.6713266084958215,

0.19451375975325327,

0.186776698089775,

0.18248000592477814,

0.17934165162478113,

0.17677361081544005,

0.1745493885045151,

0.172618474958735,

0.17095047026730062,

0.1695103326886946],

1: [0.7622835665077554,

0.7357778514055999,

0.20191656539527153,

0.18789439957714643,

0.18058554995131337,

0.17591945699237463,

0.17253370788217884,

0.16986044154521426,

0.1676252449324854,

0.16569036735300816,

0.16397997105459913,

0.1624435172999324],

2: [0.729335588507427,

0.690065409239731,

0.16067541359170606,

0.13870950864145629,

0.12794587698147677,

0.1210871463119129,

0.11606592964250761,

0.1120587916736094,

0.10866190789018039,

0.1056864732635389,

0.10304006712334209,

0.100670012881258],

3: [0.7280762871420026,

0.6671884048258214,

0.15898850233829687,

0.13881880796409754,

0.12750627627090477,

0.11957266756974279,

0.113569782907759,

0.10880259042373822,

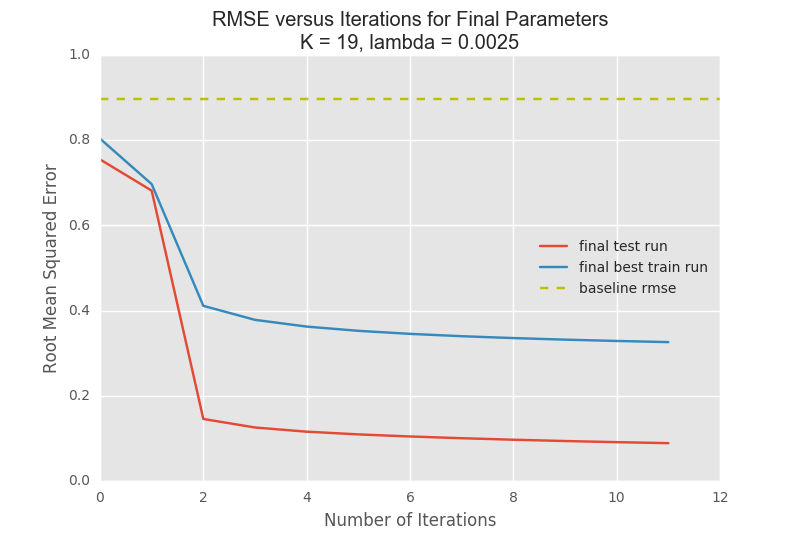
0.10494284501933879,

0.10174850129723564,

0.09904626188131788,

0.09671655129378601]}

RMSE was plotted versus the number of iterations for the final test run and the final best training run. For reference, a baseline rmse is also indicated on the plot. The baseline rmse is the result that occurs for a prediction matrix consisting of the means of each rated movie. Refer to Figure 7.



*Figure 7 Final Test and Training Runs*

The recommendations for each user were printed to a file. Here’s an example of the output:

userId: 18.0:  
 Top Rated Movie was: Twelve Monkeys (a.k.a. 12 Monkeys) (1995).  
 Top Recommended Movie is: Clockwork Orange, A (1971).  
 Link:<http://www.imdb.com/title/tt66921/>  
 Top Recommended Recent Release Movie is: Grand Budapest Hotel, The (2014).  
 Link:<http://www.imdb.com/title/tt2278388/>  
  
userId: 19.0:  
 Top Rated Movie was: Raising Arizona (1987).  
 Top Recommended Movie is: Lethal Weapon (1987).  
 Link:<http://www.imdb.com/title/tt93409/>  
 Top Recommended Recent Release Movie is: Django Unchained (2012).  
 Link:<http://www.imdb.com/title/tt1853728/>  
  
userId: 21.0:  
 Top Rated Movie was: Man Who Would Be King, The (1975).  
 Top Recommended Movie is: Willy Wonka & the Chocolate Factory (1971).  
 Link:<http://www.imdb.com/title/tt67992/>  
 Top Recommended Recent Release Movie is: Her (2013).  
 Link:<http://www.imdb.com/title/tt1798709/>  
  
userId: 38.0:  
 Top Rated Movie was: Fireworks Wednesday (Chaharshanbe-soori) (2006).  
 Top Recommended Movie is: Gattaca (1997).  
 Link:<http://www.imdb.com/title/tt119177/>  
 Top Recommended Recent Release Movie is: Avengers, The (2012).  
 Link:<http://www.imdb.com/title/tt848228/>

# Further Study

Some analysis was done before final submittal of the project. A second look at the code was done looking for any areas where we could find improvement in performance regarding computation time. For example, could we use built-in functions such as enumerate as opposed to for loops, or could we use list comprehensions? A few minor changes were made with negligible improvement in performance.

We chose one value for K, 13, and only tuned lambda. The time savings is significant in that the five-fold cross validation process only requires three training runs during each fold as opposed to nine. The results were similar as shown in the rmse plot in Figure 8. However, the rmse was not as low when only tuning lambda.

A small experiment was conducted to evaluate the performance cost of increasing the size of K which determines the size of the X and Y facto vectors. For the following four values of K, 20,30 ,40, and 50, the function called train was timed. The results indicated that increasing values of K do not impact the computation time. The times for each loop ranged from 38 minutes 36 seconds to 38 minutes 49 seconds. Therefore, it was determined the computation cost is mostly due to the size of the test set and the number of iterations. Each iteration takes approximately 1 minute for a sample size of 134.

In order to provide a basis of comparison for explaining our root mean squared error results, we decided to create a Q\_hat matrix filled with the mean ratings of each movie and the rmse for this matrix was found to be 0.897.

We decided to populate the Y vector with the overall mean rating of each movie utilizing the entire database as opposed to the mean based on just the ratings provided by the

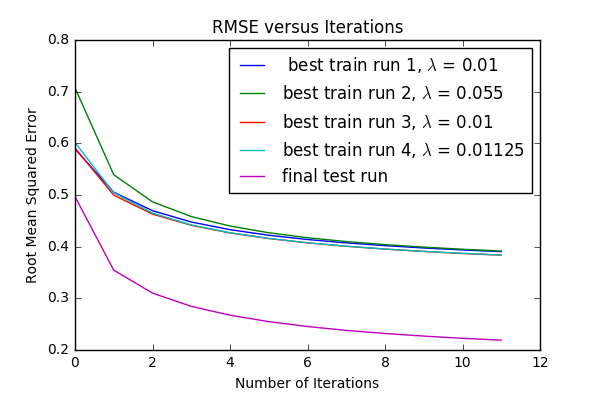


Figure 8 RMSE with K=13

smaller sample size of the test set. This would insure that the mean was created from a larger sample size and would be closer to the true population mean. By doing this, we noticed that the measured rmse was much lower after fewer iterations. Thus, we decided the additional computation time required to generate 12 iterations was not worth it, and re-ran the program setting the number of iterations to 7. The test\_set rmse after 7 iterations was 0.113 with a savings in computation time of approximately half. The results are stored in the jupyter notebook named Algorithm-Final-Copy1.

We opted not to implement any sentiment classification on the tags for unrated movies at this time. Neither did we utilize the Amazon AWS service. We look forward to investigating these options in the future as well as exploring the larger movielens dataset.

Appendix A

Weighted Alternating Least Squares Algorithm

From the MovieLens Dataset, an *m* x *n* matrix Q will be formed that contains the user ratings of *n* movies by *m* users. **Q***ui*is the rating by user *u* of movie item *i*. The problem is to find, create, or “learn” two factor matrices, one for the users, **X**, and one for the items(movies), **Y**. The rating of an item by a user is “modeled” by the product of the user and item vectors contained in **X** and **Y** respectively. Because we have two unknowns, we will use the alternating least squares approach to solve for **X** and **Y**. First, estimate **Y** using **X**, and then **X** using **Y** and continue iterating until **X** and **Y** are not changing or the change is small. Specifically, the formulas for the factor vectors, **x***u*and **y***i*, are:

**x***u* **= (YW*u*YT + λI)-1YW*u*q*u***  and **y*i* = (XTW*i*X + λI)-1XTW*i*q*i***

where **W** is a weight matrix, **I** is the identity matrix, and **λ** is a small scalar multiplier used for regularization.

Because not all of the movies have been rated, the original Q matrix constructed from the MovieLens dataset will contain null data. The null data will be set to zero. The weight matrix **W** will contain a value of 1 for items rated and a value of 0 for items not rated. Therefore, the zero values will not unduly influence the results since a 0 is in actuality an unrated movie and not a movie with a rating of 0. The **λI** term is used for regularization to help reduce overfitting.

After a certain number of iterations, the values in the X and Y factor vectors can be used to create the prediction matrix, **Q\_hat. Q\_hat = X\*Y**

Here is a pictorial illustration:

**m x n**

**matrix**

**m users**

**n items**

**m x K**

**matrix**

**K x n**

**matrix**

**Q\_hat**

**X**

**YT**

=

\*

**K << m, n**

Appendix B

Five-Fold Cross Validation

Five-Fold Cross Validation: Splitting the Data

Validation

Exp 1

Training

Set 2

Training

Set 3

Training

Set 4

Training

Set 1

Validation

Exp 2

Training

Set 3

Training

Set 4

Training

Set 1

Training

Set 2

Validation

Exp 3

Training

Set 4

Training

Set 1

Training

Set 2

Training

Set 3

Validation

Exp 4

Test Set

Training

Set 1

Training

Set 2

Training

Set 3

Training

Set 4

Training

Set 1

Training

Set 2

Training

Set 3

Training

Set 4

Test

Set

Fold 1

Experiment 1

Fold 2

Experiment 2

Fold 3

Experiment 3

Fold 4

Experiment 4

Fold 5

Experiment 5

Split the Data into Five Parts:

Concatenate green training sets to use for training and use the red sets for the validation test:

Select Best Parameters

Validation Run 1

Tune Parameters

Experiment

Run 1

n2 times

2n repititions

1

Select Best Parameters

Validation Run 2

Tune Parameters

Experiment

Run 2

n2 times

2n repititions

1

Select Best Parameters

Validation Run 3

Tune Parameters

Experiment

Run 3

n2 times

2n repititions

1

Select Best Parameters

Validation Run 4

Experiment

Run 4

n2 times

2n repititions

1

Identify Best Run and Chose Best Parameters

Experiment

Run 5

2n times

2n repititions

1

Test Run

Five-Fold Cross Validation Process ProcessPPPProcess

Appendix C

Root Mean Squared Error

We chose root mean squared error for evaluating the algorithm. The equation is as follows:

RMSE =

Where Wui, Qui, and Q\_hatui are the locations in the matrices where ratings occurred and N is the number of ratings.

For every rating in the matrix, the difference is found between it and the predicted rating (locations where W = 1). The difference is squared to make negative differences positive. The result is summed and then divided by the number of ratings (averaged). Lastly, we take the square root since we had squared the result earlier.

The result is basically the average of the differences between predicted and actual values.

The baseline rmse of 0.897 means that if we used the mean rating of a movie as the predicted value, we would be “off” by approximately 1 or 1 star.