Algorithm selection:

After getting acquainted with the dataset and understanding the problem, I did some literature survey [1],[2] to realize that the problem was from the domain of collaborative filtering. Papers related to some of the best practices [3] mentioned the use of more than one algorithm to create a recommendation system that is robust to handle fluctuation due to temporal change in data as well as serve with good recall. Since the dataset given for this project was static I chose to go with a single algorithm that was suitable for the problem.

After understanding the theory behind collaborative filtering [1, 2] it was clear that the dataset given could be solved using either User-User collaborative filtering or UUCF; Item-item collaborative filtering or IICF or combination of both. Though the popular sentiment [4], [5], [6] was to use IICF since it is more robust towards changes in data and that the dimension of movies (Number of users = 3000) was less than the dimension of Users (Number of movies = 7000), I still choose to go with UUCF approach because of the following reasons:

1. The predictions were to be made for users and hence it made sense to take user similarity and user biases in account to decide the rating for the movie
2. In case of IICF I was not sure if for some pair of movies there were not enough similarity based data to make a reliable prediction. I observed from preliminary observations of the dataset that these will not be the case if UUCF approach is used.

Intuition of Algorithm and general outline:

The general intuition on which the idea of collaborative filtering rests is , "Reaction of item (or person) A to any unknown item (or person) B can be predicted reasonably well, by learning from how items similar to A reacted when they interacted with item B or similar to B."

Developing this intuition for the dataset at hand, I decided to make predictions for user U about Movie M that she has not seen by:

- First determining the behavior profile of user U. How she rates movies in general. The average of all the ratings done by U was a good starting point for determining the same.
- Find users similar to U in their rating behaviour of movies that are already rated by U. Cosine similarity was used to compute the distance function between the users.
- I preprocessed the given data and collected users that have seen a particular movie. This was more of a transpose matrix of the given dataset but without the values.
- Prediction of movie 'm' for user 'u' can be calculated using the following formula:

$$P_{um} = U_{avg} + (\sum \{N_{simscore} \times N_{rating)} / \sum N_{simscore}) \quad \text{--- equation 1}$$

P_{um} : predicted rate of movie 'M' for user U U_{avg} : Average of all ratings done by user U
$N_{simscore}$: Similarity score between the user in question and a neighboring user who had previously seen the movie.
N_{rating} : Neighbour's rating of movie 'M'

- The formula means that the predicted rating will be the sum of the target user's average rating of the movies plus the cumulative score of the rating given by his 'k' neighbors that is adjusted and normalized by their similarity score.

Data Preprocessing and Prediction:

1. **Adjusting for user Bias:**

For every rating in the dataset, adjust it by deducting the average rate of the user.

1. It will normalize the ratings and balance out the noise of inherent bias of users. For example, say a very enthusiastic user gives every movie a higher rating like a rating of 11 for an average movie. His average rating will be usually on a higher side. While a miserly rater who gives a rating of 11 even to the movie she finds to be very good will have a lower average rating. Deducting out the average from their score will remove this biases.
2. Due to the deductions, the user rating will have now positive and negative values. Users who have opposite opinion about the same movie will have higher difference in their corresponding ratings and hence will get smaller similarity score. This will help in finding 'truly' similar users.

2. **Calculating user similarity: (adjusted cosine similarity)**

When dimensions of vectors are both positive and negative values, the range of cosine similarity is [-1, 1] with 1 for exactly same vector and -1 for exactly opposite vectors.
Due to the preprocessing in step 1, the neighborhood for each user reduced drastically. Most neighbors shared a similarity score in the range of 60 to 64 %. There were very few data points below 60 % and rarely any above to 70 %.

Predicting Values:

Preprocessed data was collected into 5 separate .csv files each for the newratings, user average ratings, the similarity score, list of users per movie and list of neighbors per user.
For each prediction, the algorithm would do as follows:

1. The 'mapping.csv' provides with target movie-id and user-id for which the recommender system will predict the rating.
2. Using the target user id fetch its average rating from preprocessed data stored in 'avgUserRating.csv'

3. For the target movie id, the file 'movieNeighbors.csv' will provide with list of users who have seen the movie, while the file 'userNeighbors.csv' will provide with the list of users similar to the target user. Users common to these two sets will be used to compute the prediction.
4. Users with similarity rating less than 60 % are not considered as neighbors because I thought that choosing users with lower similarity score would add more noise than information and thus deteriorate the quality of the results.
5. For all the users in the common set obtained from step 3, fetch their similarity score for target user from 'newSimScore.csv' and their adjusted rating to for the target movie id from 'newUserRating.csv'.
6. Calculate the prediction as per formula given in equation 1

Optimization tricks:

1. **Movie popularity vs User Bias:** Calculate the average over user rating and average over each movie. Taking average of al l these averages will give the average rating of any movie by any user, also known as total system average.

From the average rating of the system two things can be determined:

1. All popular movies will have their average ratings above this total system average and vica versa. **Movie popularity = (Avg rating of movie – total system average)**. For popular movies this term will be > 0 and it will be less than zero or close to zero for non-popular movies, which will a majority of dataset according to the the big tail phenomenon.
2. Rating prediction for every movie can be adjusted for its popularity = Rating from equation 1 + movie popularity.

2. **Learning the effect of movie popularity on users:** Using damping coefficient (α). Learning how much the popularity of the movie may affect the user rating. Create different results with varying rate of (α).

Formula: {(Rating from eq. 1) x (α)} + {(1- α) (Average rating of the movie by all users)}

Varying alpha from 0.1 to 0.9 and testing the results against the MAE would help fine tune the recommendation system.

Computation complexity | Choosing the right number of "k"

Poor judgment in understanding the vastness of output led me into choosing wrong parameters which adversely affected the rate of output for recommendations. During my initial attempt, each result was taking 25 to 45 seconds per result. At this rate, it would take years to generate results. After tweaking the algorithm further and distributing the processing load into batches, I was able to generate the results for less than a second per result. The most important take away from this project is the appreciation of the fact that simply having an algorithm that accurately solves the problem is not enough. It has to be able to do that in reasonable amount of time.

Appendix:

Preprocessing source code:
01-preProcessing-newUserRating.py : will generate Averate rating of user as stored in 'avgUserRating.csv' and adjust raw rating and populate it into file 'newUserRating.csv'
02-CalculateUserSimilarity.py : will calculate pairwise similarity score for each user and store it in 'newSimScore.csv'
03-MovieNeighbors.py : for each movie, it will fetch the list of users and populate it in ' movieNeighbors.csv'
04-userNeighbors.py : for each user, list all users similar to it from 'newSimScore.csv' and store in ' newNeighbors.csv'
PREDICTIONS.py : It uses information from all preprocessed files above to generate recommendations for each userid, movie pair as given in 'mapping.csv'.

Generated results:
I could generate only partial results, due to paucity of time.
All the source code and the preprocessed data is uploaded using gsubmit.

References:

1. Shai Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
2. Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.
3. Robert M. Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *SIGKDD Explor. Newsl.* 9, 2 (December 2007), 75-79. DOI=10.1145/1345448.1345465 http://doi.acm.org/10.1145/1345448.1345465
4. Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. 2011. Collaborative Filtering Recommender Systems. *Found. Trends Hum.-Comput. Interact.* 4, 2 (February 2011), 81-173. DOI=10.1561/1100000009 http://dx.doi.org/10.1561/1100000009
5. http://www.hindawi.com/journals/aai/2009/421425/
6. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web* (WWW '01). ACM, New York, NY, USA, 285-295. DOI=10.1145/371920.372071 http://doi.acm.org/10.1145/371920.372071

7.