Abstract
In collaborative filtering, our task is to predict how a user will rate an item given a history of the user’s ratings of other items, as well as other users’ ratings of items. This paper considers collaborative filtering in the context of the Netflix dataset, a listing of approximately 480,000 users’ ratings of over 17,000 movies. We attempt to construct a novel algorithm for approaching this problem based on the idea that subsets of movies exist in the training set, and that movies within a subset have higher predictive power for one another than for movies outside the subset. We show that such movie sets do in fact exist, although finding high-quality sets is not easy. Further, we demonstrate that predictions can be improved by taking into account a user’s prior rating tendencies.

Our results show that our algorithm makes improvements over some of the baseline schemes, and more importantly, that several of the features implemented do improve prediction accuracy of the algorithm. We expect that the insights gained in the course of this paper will lead to better methods for identifying movie subsets and therefore to better predictions in the future.

1. Introduction
1.1 Collaborative Filtering This paper deals with the general problem of collaborative filtering applied to a specific problem domain, the Netflix dataset. Collaborative filtering refers to algorithms which take into account a set of users’ ratings of certain items in order to come up with predictions for users’ ratings of new or unrated items. For the purposes of this paper, the user for whom we are making a prediction will be called the predictee user and the item for which we are predicting will be called the predictee item or predictee movie.

The primary difficulty in constructing effective collaborative filtering algorithms is finding the ratings in the training set which relate to the prediction at hand. Solutions to this challenge can include looking at all the past ratings by the predictee user, and finding users who seemed to exhibit similar rating tendencies to the predictee user. The methods employed often depend on the problem domain itself. In some cases, real-time predictions are needed so speed is a primary issue, while in other cases, elaborate models can be constructed and finessed before making predictions.

The method of combining the relevant ratings to come up with a new prediction can vary widely, from simple averaging, to weighted averaging based on various factors, to computing score differentials between users for common items.

Many collaborative filtering algorithms have been described in the literature. The two main families of algorithms are memory-based algorithms and model-based algorithms.
Memory-based algorithms search the training set to find relevant ratings, and compute a prediction based on an average of the germane ratings. Model-based algorithms first determine a model of prediction, or a rule that best describes the user’s rating pattern, and then predict according to the rule. Memory-based approaches are generally slower and less suited for online prediction.

1.2 The Netflix Dataset In the interest of improving their current algorithm, Netflix has released a substantial dataset of their customers’ ratings of movies. This dataset is comprised of the ratings of over 480,000 individual users for a collection of 17,770 movies, a total of over 1 million ratings. Each rating is an integral value between (and including) one and five stars. Netflix has offered a monetary prize to anyone who is able to make a 10% improvement over their current prediction algorithm, called Cinematch. The primary goal of using this dataset was to have access to a large, complex set of ratings that serve as a suitable test bed for the algorithms to be explored in this paper.

The dataset is distributed along with a “probe set” of user/movie pairs upon which algorithms can be tested. The actual ratings for these pairs are known, so an algorithm’s output for the set can be compared to the actual ratings in order to measure the error rate. For this paper, error will be measured using root mean squared error, or RMSE. For a vector of actual ratings of movies $\theta$, and a vector of our algorithm’s predictions for these movies $\hat{\theta}$:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i} (\theta_i - \hat{\theta}_i)^2} \quad [6]$$

1.3 Goals of the Project The purpose of this project was to explore the domain of collaborative filtering and to develop an original algorithm which worked well on the Netflix dataset. Future work will be guided based on the successes and failures of this project. The performance of the algorithm was benchmarked against several other known algorithms to determine absolute performance; variants of the algorithm were also compared to one another to determine the features which are most successful in predicting.

2. The Slope One Algorithm As stated before, there are two basic issues in collaborative filtering: identifying relevant ratings in the training set and combining those ratings. The Slope One family of algorithms is designed to deal with both of these issues [1]. For this paper, the Slope One algorithm was used to deal with the second problem while the first problem was dealt with in a different manner.

The algorithm itself is based on the idea of a “popularity differential” between items. For example, if User A rates Item J 0.5 stars higher than Item I, then User B might also like Item J by about 0.5 stars more than Item I. This is demonstrated in Figure 1.

Figure 1. The Slope One algorithm is based on the notion of a “popularity differential”. Adapted from [1].

Formally, define $P(u)$ as a prediction vector, where each component $P(u)_i$ is the contribution one item in the training set, item $i$, makes to the prediction. We are predicting on item $j$. $R_j$ is the set of relevant items, i.e. the items other than $j$ that the predictee user has rated ($R_j$ is later changed in this project – see Section 3), and $card(R_j)$ is the size of the set $R_j$. Let $u_i$ be the rating that the predictee user gave item $i$. Define $dev_{j,i}$ to be the average
difference between the ratings for items \( i \) and \( j \) across all users who have rated both \( i \) and \( j \). Then, the Slope One algorithm can be summarized as follows:

\[
P(u) = \frac{1}{\text{card}(R_j)} \sum_{i \in R_j} (\text{dev}_{j,i} + u)
\]

This defines the value of each component of the prediction vector. To get the overall prediction, we do a weighted average of the components, either with equal weights for Standard Slope One, or with different weights for Weighted Slope One.

The Weighted Slope One algorithm is the one actually used in this project. This version takes into account the number of users who rated a specific item in \( R_j \). If more users rated an item, the accuracy of that component of the prediction is assumed to be greater, so it is weighted more heavily. The weights increase linearly with the number of users who rated the item. So, for each component of the prediction vector \( P(u) \), there is an associated weight value contained in the vector \( W(u) \), where each entry is the number of users rating both the predictee item and the training item. The overall prediction \( P \) is then:

\[
P = \frac{\sum_{i \in R_j} P(u_i)W(u_i)}{\sum_{i \in R_j} W(u_i)}
\]

The name of the algorithm, Slope One, derives from the fact that we are essentially estimating an equation of the form \( f(x) = x + b \), where \( f(x) \) is the prediction and \( x \) is an average of ratings for a training movie. We estimate \( b \) for many different training movies, and combine the results of the functions to get a prediction. Functions with additional coefficients, like \( f(x) = ax + b \) or \( f(x) = ax^2 + bx + c \) may be used, but previous work has shown no significant increase in performance over \( f(x) = x + b \) [2].

3. Critical Movie and User Sets
The problem of finding relevant ratings was one of the main issues dealt with in this project. The insight behind the methods employed was that there are probably subsets of movies in the training set, and that the movies in a subset have higher predictive power for one another than for the movies outside the subset. These sets are henceforth referred to as critical movie sets. The same can be said for users; subsets of users with this property are referred to as critical user sets.

The purpose of finding these sets was to be more selective about the valid inputs to the Slope One algorithm. Rather than allowing \( R_j \) to be the set of all movies rated by the predictee user, we only allowed movies in the same cluster as the predictee movie to be in \( R_j \). Similarly, rather than allowing all users who rated both the predictee movie and a movie in \( R_j \) to be included in the calculation of \( \text{dev}_{j,i} \), we only allowed users in the same cluster as the predictee user to be included.

There are many possible methods for coming up with candidate movie and user sets. The difficulty was that the data in the training matrix is very sparse – each user has rated only a small fraction of the 17,770 movies, and some users have very few ratings. This made it difficult to employ traditional clustering methods, or even just to compare users who had few, if any, overlaps in movies seen.

To find critical movie sets in this project, we computed a 5-dimensional vector for each movie. Each feature of the new vector contained the fraction of ratings of the movie that were a given value:

\[
\nu(m) = \left[ \frac{f(1)}{f(\text{all})}, \frac{f(2)}{f(\text{all})}, \frac{f(3)}{f(\text{all})}, \frac{f(4)}{f(\text{all})}, \frac{f(5)}{f(\text{all})} \right]
\]

\( f(n) \) = # ratings of value \( n \) for the given movie
\( f(\text{all}) \) = total # ratings for the given movie

Rather than considering individual ratings as features, we considered a movie in terms of its distribution of ratings – what percentage of its
ratings were one star, two stars, etc. Once this vector was computed for each movie, we performed k-means clustering on the vectors. The resulting clusters were, for example, movies with a high proportion of 5-star ratings; movies with a high proportion of 1-star ratings; movies with a nearly equal proportion of all ratings; and movies with a high proportion of 1- and 5-star ratings, but a low proportion of 3-star ratings.

We found critical user sets subsequent to the critical movie sets, for reasons explained below. To do this, we computed each user’s vote distribution on each subset of movies. Then, for a given critical movie set, we clustered the users according to their vote distributions on that movie set via k-means. We computed different critical user sets for each critical movie set. The logic behind this approach was that for a given subset of movies, there are probably some users who rate them similarly to the norm, and other users who are outliers and rate differently from the norm. For example, for a set of movies where 80% of the ratings are five stars, there are clusters of users who rate many of them five (the “norm” users) and clusters of users who rate many of them one (the “outliers”).

Then, when making predictions, we considered which group the predictee user was in based on his/her past ratings, and only used ratings from other users in their critical user set. Continuing the above example, if the predictee user rated many of the movies in that cluster 1 star (an outlier), we considered only the other outliers who had rated many of them 1 star in making our prediction.

An alternative method we tested for finding relevant users took into account the structure of the Slope One algorithm. Because the algorithm computes an average rating differential between two users, the ideal user input to this algorithm has a high consistency in vote differential from the predictee user. If the two users have been different by a consistent amount in the past, they will likely continue the trend in the future. Thus, users were selected on the basis of their “differential consistency” with the predictee user. In this project, consistency between two users was measured by examining the following ratio:

\[
\frac{\text{frequency(highest occurring rating differential)}}{\text{total number of ratings in common}}
\]

This gave us a rough estimate of how consistent the differential was between two users, and only those users with a consistency ratio above a certain threshold were included in the prediction calculation.

It is important to keep in mind that we were actually looking for users with similar voting patterns to the predictee user. For example, this approach identified users who consistently had a 0 rating differential from the predictee user – i.e. a consistently similar user. Additionally, it is important to note that users with high rating differentials were generally not selected via this approach. Intuitively, it seems unlikely that there are pairs of users such that one always rates movies three stars higher than the other, especially considering the small range of possible ratings. It is more likely that either the difference is small or that there is no consistency in the rating differential. The users identified by this method generally had a small differential from the predictee user.

4. Score Tweaking
Combining the relevant ratings via the Slope One algorithm gave a general prediction of the rating for the predictee user/movie. However, it was important to consider that each user may have individual rating tendencies which were not reflected in a prediction which was an average of other users’ ratings. For example, some users may be bimodal in their ratings, giving only one or five stars to movies they have seen. In this case, it makes no sense to predict a 4.2 for them. If we know that they rate all movies they like five stars, then we should predict a five, or at least something
much closer to it. Likewise, if a user gives out all ratings, but rates movies four stars much more often than five stars, then a prediction somewhere between four and five should probably be moved closer to four to take into account the user’s past rating history.

Essentially, we were making the assumption that the Slope One algorithm yielded a ballpark prediction which could then be fine-tuned to fit a user’s historical tendencies.

The adjustment used in this project was only a first step toward fully utilizing the information gleaned from the user’s rating history. It took into account the prior distribution on the two integral ratings surrounding the Slope One prediction, referred to as book-end values, e.g. the distribution on 4 and 5 if the prediction is a 4.3. As the difference in distribution between the two book-end values increased, the rating was shifted more and more towards the appropriate book-end value (see Figure 2). If the difference in distribution between the two was small, more weight was given to the Slope One prediction and less to the book-end values. The actual formula used was:

\[
pred = \text{Slope One prediction} \\
d(x) = \text{distribution on integral score } x \\
diff = \text{difference in distribution between } \lfloor \text{pred} \rfloor \text{ and } \lceil \text{pred} \rceil \\
\frac{2 \times \text{diff} \times \lfloor \text{pred} \rfloor \times d(\lfloor \text{pred} \rfloor) + \lceil \text{pred} \rceil \times d(\lceil \text{pred} \rceil) + (1 - \text{diff}) \times \text{pred}}{2 \times \text{diff} \times d(\lfloor \text{pred} \rfloor) + d(\lceil \text{pred} \rceil) + (1 - \text{diff)}}
\]

Figure 2. Tweaking the prediction to take into account the user’s past rating tendencies. Because the user rates movies 5 much more often than 4, the first prediction of 4.37 will be shifted closer to 5.

5. Implementation

Most of the algorithm was written in C++ within an open-source data management framework developed specifically for the Netflix dataset [4]. The framework allowed for efficient storage of the large dataset in memory and provided functionality for accessing the data. The k-means clustering code was written in Python.

K-means clustering requires a number of clusters to compute as input. In some domains, \( k \) is known a priori; in this case, there was no clear way to determine an optimal \( k \). After analyzing results, we chose to compute 25 critical movie sets and 25 critical user sets per movie set. Having more than 25 clusters generally resulted in several clusters which were too close to be distinct; having fewer clusters made them too far apart to be specific enough in splitting the data. However, this number was somewhat arbitrary and better methods of determining \( k \) may result in better prediction performance.

Run-time performance was a significant issue in this project because of the sheer size of the dataset. Preprocessing took up the majority of the computation time, as was intended. Clustering of the 480,000 users in five dimensions for each critical movie set took approximately 12 hours using sequential k-means clustering. User clustering was done once for each critical movie set so three Zoo machines were used to perform the clusterings at the same time. Each clustering took between 45 and 120 minutes. Clustering of the movies was a less substantial task because of the smaller point set, and took roughly 30 minutes to complete. Once this preprocessing was done, predictions could be made fairly quickly. The complete Weighted Slope One algorithm using user and movie sets and score tweaking made 2000 predictions in about 17 minutes on an Intel Pentium M 1.5 GHz machine with 1Gb of memory. This gives an average prediction time of approximately 0.51 seconds/prediction.
6. Experimental Results
We measured the algorithm’s performance on the probe set included with the Netflix dataset. For comparison, we also ran several benchmark algorithms on the same data. These results are summarized in Table 1. The algorithms are:

1. **Global Average** – predict a constant: the average rating given to any movie by any user. The average rating is 3.60.
2. **Movie Average** – for a predictee movie, predict the average of all its ratings so far.
3. **Double Average** – for a predictee movie and user, compute the movie’s average rating so far and the user’s average rating of all movies so far. Predict the average of these two values.
4. **Cinematch** – Netflix’s proprietary prediction algorithm. The exact details are not specified but it is described as using “Straightforward statistical linear models with a lot of data conditioning” [3]. We used the RMSE given on the website for the probe set for comparison.

Table 1. Comparison of the performance of the complete algorithm to benchmark algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slope One with Movie/User Sets</td>
<td>1.0442</td>
</tr>
<tr>
<td>Global Average</td>
<td>1.1298</td>
</tr>
<tr>
<td>Movie Average</td>
<td>1.0519</td>
</tr>
<tr>
<td>Double Average</td>
<td>1.0033</td>
</tr>
<tr>
<td>Cinematch</td>
<td>0.9474</td>
</tr>
</tbody>
</table>

Additionally, several different versions of the adapted Slope One algorithm were compared to one another to judge the relative merits of each feature. The different versions allow different groups of movies and/or users to be input into the Slope One prediction. These results are summarized in Table 2. The versions tested were:

1. **Weighted Slope One**
2. **Movie sets** – (1) plus only using movies in the predictee movie’s critical movie set.
3. **Movie sets + score tweaking** – (2) plus tweaking the prediction to account for the predictee user’s past rating tendencies.
4. **Movie + user sets + score tweaking** – (3) plus only using users who are in the predictee user’s critical user set.
5. **Movie sets + consistency screening + score tweaking** – (3) plus only using users who have a consistent “rating differential” with the predictee user.

Table 2. Comparison of the relative performance of different features of the algorithm. The reference scheme for each comparison is Algorithm (1).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>0.0000%</td>
</tr>
<tr>
<td>(2)</td>
<td>-0.6858%</td>
</tr>
<tr>
<td>(3)</td>
<td>0.1835%</td>
</tr>
<tr>
<td>(4)</td>
<td>0.9079%</td>
</tr>
<tr>
<td>(5)</td>
<td>-0.0676%</td>
</tr>
</tbody>
</table>

As shown in Table 1, the algorithm we developed showed improvement over some of the baseline schemes but failed to achieve the same level of performance as the Netflix Cinematch algorithm.

Table 2 shows some more interesting results regarding the various features with which we experimented. The score tweaking function made a small but consistent improvement to all of the various schemes. It generally improved results by about 0.8%. This is significant considering the limited effect it currently has on adjusting the scores.

Adding movie sets alone slightly decreased the prediction performance of the algorithm. However, because of the way user sets were determined based on movie sets, we expected their addition would have a more significant impact on the results. The results show that this was true. Adding user sets increased prediction accuracy by about 0.8%.

We also tested the method described above for finding users with a consistent rating differential from the predictee user. Because the calculation was performed as a prediction is being made, run-time performance is hurt. However, the accuracy of predictions made using these users did not increase; the predictions were worse than those made using critical user sets.
We note that fairly significant changes in prediction mechanisms resulted in only small changes in prediction performance. It is likely that this reflects the fact that the intuitions behind our methods were valid, but that fulfilling the goals of these intuitions is hard. Movie sets and user sets do exist, and we made progress towards finding them. The small gain in prediction accuracy indicates that our sets also contained many erroneous movies/users which were no more helpful than the average movie or user in making predictions.

7. Conclusions and Future Work

The algorithm implemented showed some success in increasing prediction accuracy over the basic Slope One algorithm. The results imply that it is possible to identify critical movie and user sets with some amount of accuracy. Additionally, the results showed that taking into account a user’s past rating tendencies has a consistent, significant effect on the accuracy of the predictions made.

The project has inspired many ideas for future work on this problem. Of particular interest is the development of better methods for identifying critical movie and user sets. Although clustering based on score distribution is a step in the right direction, it seems that doing so creates subsets of movies which are too general: good movies, for example, are all clustered together, although there are probably specific subsets of good movies which are liked by different types of users. Identifying groups of users who rate certain movies similarly is an effective strategy in principle, but it needs to be refined to have a more significant effect.

In the methods employed in this project, we basically tried to estimate correlation between movies and users. By finding critical movie and user sets, we attempted to find groups of movies and users that were highly correlated without explicitly doing pair-wise correlation computations across the entire training set. This was done primarily because of performance considerations.

Ideally, to make an accurate prediction for a user/movie pair, the algorithm would be able to quickly identify users who were similar to the predictee user, as well as movies that had a strong correlation (either positive or negative) to the predictee movie. Then, using only the appropriate movies, the algorithm would compare the relevant users’ past ratings to the predictee user’s ratings to determine whether they rated very similarly or differently. Based on this information, each relevant user would make a contribution to the overall prediction.

Essentially, we would need to compute a giant correlation matrix between users (~480,000 x 480,000) and movies (17,770 x 17,770). This problem seems well-suited for a parallel program. Because each calculation is independent of the other, this problem could be infinitely parallelized. As the training set is static, this would only have to be computed once as a pre-processing step. The run-time of this step could be significantly reduced via parallelism with enough processors, as there are no data interdependencies. Predictions would then be made by looking up the most highly correlated users and movies to the predictee user and movie and combining those ratings to make a prediction. The correlation matrices would not necessarily construct closed subsets of critical movies/users, but would list for a given user or movie the most relevant users and movies to consider.

The Netflix dataset has provided a substantial test bed for our collaborative filtering algorithm. We have shown that although it is possible to selectively identify ratings which are relevant to a specific prediction, it is not easy to do so with high confidence. Particularly promising was the consideration of users’ past rating tendencies in adjusting predictions. Future work towards better identifying relevant users and ratings for a given prediction will hopefully improve the accuracy of this algorithm.
8. References


