

# The Netflix Prize: A 5.11% Solution



Johnathan Cree and Kylan Robinson  
{jcree,krobinso}@eecs.wsu.edu  
December 17, 2008

CptS 570: Machine Learning  
School of Electrical Engineering and Computer Science  
Washington State University  
Dr. Larry Holder

# 1. Introduction

This report describes our participation in the Netflix Prize contest. Using machine learning techniques, we were able to create a movie rating algorithm that is 5.11% more accurate than the proprietary Cinematch system employed by Netflix.

The Netflix Prize contest started in 2006 as a five-year challenge to find a movie recommendation system with a 10% improvement over Cinematch [1]. The grand prize is \$1 million, and there are \$50,000 yearly progress prizes as well. With this much money on the line, Netflix must expect to profit greatly from a new recommender. The solutions will do more than just improve their business, though. Since the winners are required to publish their methods, the contest serves to advance the field of machine learning as well.

The remainder of this report will be organized as follows: Section 2 describes the attributes and structure of the Netflix data. Section 3 discusses the methods we used to develop our algorithm. Section 4 reports our results. Section 5 reflects on our experience and proposes future enhancements to our algorithms.

## 2. Netflix Data

Netflix provides the same set of data to all contestants, made up of movies, users, ratings, and dates [2]. Figure 1 shows some of the attributes of the dataset.

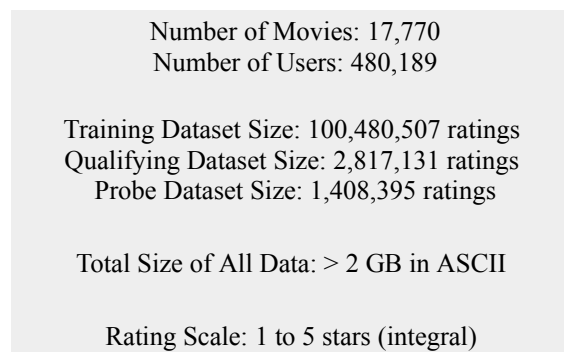


Figure 1: Netflix Prize data, by the numbers.

- The training dataset consists of one file per movie. Each movie file contains entries of the form `<customer_id, rating, date>`. This dataset can be used to train a learning algorithm.
- The qualifying dataset is a single file with movie identification numbers followed by a list of `<customer_id, date>` tuples for each movie. Contest submissions involve replacing the `<customer_id, date>` entries with a corresponding rating prediction.
- The probe dataset is a single file with movie identification numbers followed by a list of customer identification numbers. This resource can be used to calculate a rough RMSE

for an algorithm without having to make a submission to the Netflix Prize site. The Netflix Prize rules limit each team to one submission per day, so the probe set can be used for more frequent experimentation.

The Netflix Prize competition uses Root Mean Squared Error (RMSE) to measure the accuracy of submissions. This metric has been shown to be appropriate because it represents the distance between predicted and true values [3]. Lower RMSE values are better, with an RMSE of 0.0000 representing 100% accuracy. As a means of comparison, the Cinematch RMSE is 0.9514 and the RMSE needed to qualify for the grand prize is 0.8572 [1].

## 3. Methods

In this section, we describe the different methods we applied to try to improve our RMSE. Not all of the approaches worked as expected, but every one helped contribute to our understanding of the problem and encouraged us to explore other possible solutions.

The following are general descriptions of the various techniques that we used. Source code is available upon request. Our complete list of Netflix Prize submissions can be found in Appendix A: Summary of Attempts.

### *3.1 Simple Averages*

Since the sheer size of the Netflix data can be quite daunting, we decided to start with a few simple approaches. This allowed us to practice working with the data: reading it into memory, processing values, and generating prediction files. Even with naive algorithms, it was apparent that a major key to the competition was the ability to write optimal, fast-running code.

#### **3.1.1 Global Average**

Over all of the 100,480,507 ratings in the training set, the mean rating is 3.8 stars out of 5 possible. To practice creating a valid prediction file, we wrote a script that rated each movie in the qualifying file a value of 3.8. This approach served as the first baseline for our project.

Best RMSE: 1.1357

### 3.1.1 Movie Average

Since the training dataset is organized by movie, it was easy to find the average rating for each individual movie. If one movie is considered to be better than another by the general population of customers, it will have a higher average star rating. Here is the algorithm we used to make predictions based on the mean rating of each movie:

```
for each movie_id in qualifying_dataset:
    movie_avg = mean_rating(movie_id)
    for each prediction under movie_id:
        prediction = movie_avg
```

Best RMSE: 1.0533

### 3.1.2 Customer Average

Not all customers will give the same rating to a particular movie. If this were the case, the problem would be easy and the movie average approach would show 100% accuracy. Individuals have different sources and degrees of bias. To rate each movie we assume that each individual customer has their own mean rating and that mean rating is close to what they will rate all movies.

Our Customer Average method started by finding the mean rating for each customer. Then the qualifying dataset was processed, replacing each <customer\_id, date> tuple with the mean rating value corresponding to that customer identification number.

Best RMSE: 1.0651

### 3.1.3 Movie and User Double Average

Our Movie Average method attempted to measure the quality of a movie, but ignored the preferences of individual customers. Our Customer Average method did the opposite, focusing on the overall disposition of a customer while ignoring movie quality. We decided to combine the Movie Average and Customer Average methods by simply averaging the corresponding entries in both prediction files.

We tried weighting this combination to favor one method over the other, but a simple 50/50 split seemed to work best. However, we didn't think about weighting based on the variance of each mean until after we had achieved RMSE scores with other approaches. The Movie and User Double Average method provided the lowest error of all of our Simple Average methods, with an RMSE of 1.0147.

Best RMSE: 1.0147

### ***3.2 Artificial Neural Networks (ANNs)***

We were curious as to how an artificial neural network would perform as a recommendation system. Seeing how neither of us had used an ANN before we just picked a few different number of nodes at each hidden layer. We tried a small network 25x5, then a larger 100x25, and one a little bigger 500x100. However, when we submitted these scores something really weird happened which was that all of the ANNs received the same RMSE score. So we figured either we must have submitted the same file multiple times or there was a bug that made it so each ANN outputted the same results. However, after checking our md5 sum which needs to be submitted with all submissions we found that we didn't submit the same file multiple times. Next we performed a diff on the files and 85% of the ratings differed from one file to the next. This really surprised us and I still think there is some bug in the code. However, since we were only getting a RMSE score of 2.0193 which is significantly worse than using the global average and we had reports from others in the class that their ANNs were not performing as well as simple averages we decided to give up on this approach and try something different.

Best RMSE: 2.0193

### ***3.3 Nearest Neighbor***

Given that the customer average and movie average worked well, we decided maybe groups of customers existed that rated movies similar to each other. To predict a rating for a customer X and movie Y, we find the customers who rated movie Y and also rated many of the same movies as customer X. Then we can use the similar customers to find a rating for customer X.

To do this, we used a Euclidean distance between each customer's rating of a given movie and the customer we wanted to find a rating for. Then we summed up this difference for every movie they both had rated (had in common) and divided by the number of movies they had in common. Finally, to calculate the ratings we found the nearest neighbors (the smallest distance) and averaged their ratings as well as voted on their ratings.

The first problem that we ran into is that we couldn't fit the entire data set into memory. This led to the problem of having to search through each movie file to figure out if a customer had rated it or not. To solve this problem we decided to manipulate the data and create files such that there was a file for each customer that listed the movies they rated, the rating, and the date it was rated.

With the transformed data we started to run a nearest neighbor program. It was calculated to take 25 days to complete so we had to optimize the program and finally we got it down to about 8 days. That still was not enough time but then we realized that when we tried to store the data into memory it was in ASCII format so we created a data structure. A vector that held a class Customer. The Customer contained an int customerID and a vector of Movie classes. The Movie class contained a short int for the movieID and a char for the rating value. Now we were able to store all the data in less than half a gigabyte.

Now that we could hold all of the data, we re-wrote our nearest neighbor program to read in all

of the data before performing calculations. The problem we ran into is that there were quite a few movies that had been rated by a large number of customers so in order to calculate nearest neighbor on all of them it could take a few minutes for each prediction.

The next approach was to limit the number of customers that were used in the nearest neighbor calculation. To get it done fast we decided that if a movie had more than 2500 customers that had rated it then we would randomly select 2500 customers to calculate the nearest neighbor. This took one day to complete the calculations and since it was so time consuming we had it calculate 1NN, 3NN, 5NN, 10NN with averaging the nearest neighbor's ratings and having the nearest neighbors vote. Since this was completed towards the end of our project and we had to get other submissions we decided to submit the 3NN average which got us a RMSE of 1.2988 which is worse than just using the global average. After finding the 2500 randomly selected customer approach didn't do so well we decided to increase the number of randomly selected customers to 5000. However, this didn't work out as expected and we received a worse RSME score.

We would liked to have had time to work with using different ways to calculate the distance between the neighbors as well as use different formulas to calculate the rating based on the nearest neighbors. We also thought about randomly selecting about 10 groups of customers around size 250 then vote on the probe set using these customer and pick the group with the best RSME to calculate the rating for the movie.

In all of the approaches above we wanted to make it so that if a customer only has a couple of movies in common with another customer then we wouldn't use that customer as a nearest neighbor and if there were not enough nearest neighbors or the nearest neighbor was too far away then we would use some other approach so calculate that rating. We believe that this would really help solve our problem with kNN because there are many customers that have rated fewer than 10 movies.

We believe that the reason our current approach is not performing as well as we had hoped is because of the simplifications we made and the fact that many customers haven't rated very many movies.

Best RMSE: 1.2988

### ***3.4 Singular Value Decomposition***

One of the most popular methods applied to the Netflix Prize problem is called Singular Value Decomposition (SVD). This requires conceptualizing the Netflix data as a sparsely populated m-by-n matrix M of movies, customers, and ratings. We 'decompose' this matrix by defining it as

$$M = U\Sigma V^*$$

where U is an m-by-m unitary matrix,  $\Sigma$  is an m-by-n diagonal matrix with non-negative entries, and  $V^*$  is the conjugate transpose of a n-by-n unitary matrix [4]. By sampling the eigenvalues of  $\Sigma$ , we obtain a compressed representation of the matrix [5]. The resulting compression causes

movies to be grouped by common features. These feature groups allow higher predictive accuracy.

### **3.4.1 Simon Funk**

The first Netflix Prize contestant to successfully implement SVD was team Simon Funk. Surprisingly, Simon Funk chose to post their algorithm on the Internet, even though it gave up the secrets of their third place position on the leaderboard [6].

Our first submission using the SVD approach was an adaptation of the Timely Development implementation of the Simon Funk algorithm [7].

Best RMSE: 0.9261

### **3.4.2 BellKor/KorBell**

Currently, the BellKor team from AT&T labs is the world leader in the Netflix Prize competition [8]. Recently, they were awarded the 2008 Progress Prize [9]. It seemed reasonable, then, that imitating their approach might be a good way to lower our own RMSE. However, the BellKor team's solution is a linear combination of more than 100 different learner results [10]. This forced us to focus on a subset of these learners, and we chose to investigate their implementation of SVD.

BellKor's SVD approach is different from the Simon Funk algorithm in a couple of ways [11]. First, it incorporates the concepts of the Nearest Neighbor model described earlier. Essentially, the BellKor algorithm performs kNN at the local level and SVD at the global level. Second, the BellKor approach is iterative, allowing scalability and increased performance. As a result, the BellKor implementation is a full order of magnitude faster than Simon Funk's [12].

We used an implementation of the BellKor SVD algorithm that also removed the baseline movie average error from the predictions. This gave a better RMSE than the BellKor SVD algorithm on its own [12].

Best RMSE: 0.9066

### **3.4.3 Arek Paterek**

Another team that has done very well with the SVD method is Arek Paterek from Poland. As with the two other SVD algorithms we studied, his algorithm has some unique features [13]. Contributions include post-processing using the Nearest Neighbor method and kernel ridge regression. Also, the number of parameters are decreased by using gradient descent to learn a generalized model for each user.

Our implementation of the Arak Paterek algorithm was executed in two steps [14][15]. First the SVD was calculated, using a smaller number of features than the BellKor method. Then the results were postprocessed using calculated weights. This algorithm gave the best RMSE of any single learner we tried.

Best RMSE: 0.9048

### ***3.5 Ensembles***

Once we had established a small group of different learners with good RMSE values, we began thinking about ways to combine our results. All of the leading teams in the competition use "blending" or "mixing" to achieve their scores. Since we were short on time, we decided to implement the most elementary ensemble method possible [16]. We simply took the prediction files from the learners we wanted to blend and averaged their prediction values.

In most cases, the resulting prediction file gave an RMSE somewhere between the best and worst learner. This is not what we had hoped for. However, a simple ensemble worked very well when applied to our SVD learners. Since the algorithms were sufficiently independent of one another, the RMSE value of the resulting prediction file was 0.002 lower than our previous best.

Best RMSE: 0.9028

## **4. Results**

In the end, our ensemble of SVD predictors achieved the highest accuracy, with an RMSE of 0.9028. This value represents a 5.11% improvement over Netflix's Cinematch. As of December 17<sup>th</sup>, 2008, we are currently in 297<sup>th</sup> place in the world, which is in the top 1% of 35,529 teams competing for the Netflix Prize.

## **5. Discussion and Conclusion**

### ***Tools***

All of our algorithms were implemented in Python or C/C++. Python was convenient because it was able to parse the data with ease. It also allowed for quick implementation of algorithms. However, as the algorithms became increasingly complex, the slow performance became an issue.

For simple algorithms and general data exploration, we found a Python package named Pyflx to be extremely helpful [17]. It allowed us to implement each of our Simple Average methods in

just a few lines. One of the major advantages of Pyflix is the ability to test against the probe set with a simple commandline option. Of course, since Pyflix is implemented in Python, it faced the same performance issue as above.

We used C/C++ for the implementation our more complicated algorithms. This allowed us to have more control over the memory management of our processes. Much of our success is attributable to the availability of open source implementations of the methods described above. In particular, nprize [18] and Timely Development [19] were especially useful.

### *Future Work*

To further improve our RMSE, there are a number of steps we would like to try. First, it would be interesting to explore the Nearest Neighbor approach a little more. It seems like this method most closely resembles the general idea of a movie recommendation system. We had expected our implementations of Nearest Neighbor algorithms to achieve better RMSE scores, and maybe they still can with some modifications.

Since the SVD method produced such good results, we would also like to pursue more optimizations for those algorithms. Comparing the performance of the Simon Funk approach with that of BellKor, it is obvious that not all SVD implementations are equal. Particularly, we are interested in improving the utilization of Nearest Neighbor principles in the BellKor and Arek Paterek algorithms. We obtained permission to become contributors to the nprize repository and have already started contributing improvements.

Finally, we need to develop a better method for creating ensembles from groups of learners. The approach we used above was able to improve our RMSE, but it was also very naive. Instead of simply taking the average of two or three prediction files, we would like the combination of learners to take place during the prediction process itself. This way, the learners could be weighted on a per-prediction basis if needed.

### *Challenges*

When we first set out address this project, we assumed that the main challenge would be in algorithm design. After all, there are a number of methods and approaches that seem promising. Choosing the right mix of learners can be an intimidating task. We ended up with many more ideas than we had the opportunity able to try.

The real limiting factors were related to the immense size of the datasets. Computer memory, processing power, and time were all in short supply. In one sense, this was very frustrating. Many times, we spent hours and days waiting for processes to complete. On the other hand, it helped add to the overall experience. We were working on a dataset with realistic size and complexity. It is a real-world problem that gave us valuable experience. We were forced to write quality code that was aware of both memory and CPU limitations.

Another challenge was the interdisciplinary nature of the Netflix Prize contest. As is true with most machine learning problems, this involved elements of statistics, pattern recognition, psychology, and culture [20]. We had many conversations that were technical as well as philosophical. One of the best parts of this competition is the fact that it requires contestants to approach the problem broadly, without any preconceptions.

## 6. References

- [1] Netflix Prize rules. <http://www.netflixprize.com/rules>
- [2] Dataset README file. <http://www.netflixprize.com/community/viewtopic.php?id=68>
- [3] Herlocker et al. "Evaluating Collaborative Filtering Recommender Systems."  
[http://web.engr.oregonstate.edu/~herlock/papers/eval\\_tois.pdf](http://web.engr.oregonstate.edu/~herlock/papers/eval_tois.pdf)
- [4] Singular Value Decomposition. [http://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](http://en.wikipedia.org/wiki/Singular_value_decomposition)
- [5] Rashidi and Jakkula. "CptS 570 Machine Learning Project: Netflix Competition."  
<http://eecs.wsu.edu/~vjakkula/MLProject.pdf>
- [6] Netflix Update: Try This at Home. <http://sifter.org/~simon/journal/20061211.html>
- [7] Timely Development – Netflix Prize Results and Source Code.  
<http://www.timelydevelopment.com/demos/NetflixPrize.aspx>
- [8] Netflix Prize Leaderboard. <http://www.netflixprize.com/leaderboard>
- [9] Netflix Progress Prize 2008 awarded to team BellKor in BigChaos.  
<http://www.netflixprize.com/community/viewtopic.php?id=1193>
- [10] Bell, Koren, and Volinsky. "The BellKor 2008 Solution to the Netflix Prize."  
<http://www.research.att.com/~volinsky/netflix/Bellkor2008.pdf>
- [11] Bell, Koren, and Volinsky. "Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems."  
<http://public.research.att.com/~volinsky/netflix/NetflixKDD07.pdf>
- [12] nprize README. <http://code.google.com/p/nprize/source/browse/trunk/README.txt>
- [13] Paterek. "Improving regularized singular value decomposition for collaborative filtering."  
<http://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/Regular-Paterek.pdf>
- [14] BellKor's NSVD1. <http://www.netflixprize.com/community/viewtopic.php?id=898&p=1>

[15] nprize NSVD1 Source. <http://code.google.com/p/nprize/source/browse/trunk/usv dns1b.c>

[16] Holder. "Ensembles of Classifiers."  
<http://eecs.wsu.edu/~holder/courses/CptS570/fall08/slides/ensemble.pdf>

[17] Pyflix. <http://pyflix.python-hosting.com/>

[18] nprize. <http://code.google.com/p/nprize/>

[19] Free source code :). <http://www.netflixprize.com/community/viewtopic.php?id=481>

[20] Holder. "Conclusions."  
<http://eecs.wsu.edu/~holder/courses/CptS570/fall08/slides/conclude.pdf>

# Appendix A: Summary of Attempts

The following table summarizes the submissions made to the Narimus account for the Netflix Prize competition. The Date column lists the date of submission (based on UTC time). The RMSE column contains the root mean squared error of the submission, as reported by Netflix. The Description field explains the algorithm used to prepare the submission's prediction file.

Date	RMSE	Description
10/21/2008	1.1357	Global Average
11/15/2008	1.0533	Movie Average
11/16/2008	1.0953	Movie Average Rounded
11/18/2008	2.0193	Artificial Neural Network 25x5 hidden layers
11/19/2008	2.0193	Artificial Neural Network 100x25 hidden layers
12/2/2008	1.0651	Customer Average
12/3/2008	1.0147	$0.5 * \{\text{Customer Average}\} + 0.5 * \{\text{Movie Average}\}$
12/5/2008	2.0193	Artificial Neural Network 1000x100 hidden layers
12/8/2008	0.9671	Simon Funk SVD MAX_FEATURES=20
12/9/2008	0.9261	Simon Funk SVD MAX_FEATURES=64
12/10/2008	1.0153	$0.46 * \{\text{Customer Average}\} + 0.55 * \{\text{Movie Average}\}$
12/11/2008	0.9066	BellKor SVD
12/12/2008	1.2988	3NN, 2500 randomly selected users used to calculate 3NN
12/13/2008	0.9028	Average of Simon Funk SVD, BellKor SVD, and Arek Paterek SVD
12/15/2008	0.9048	Arek Paterek SVD
12/16/2008	1.3027	3NN, 5000 randomly selected users used to calculate 3NN

Table A.1: Submissions made to the Narimus Netflix Prize account.

# Appendix B: Reproducing the Best Result

These instructions detail the process for reproducing our best RMSE value of 0.9028.

A copy of our best prediction file is located at:

[http://www.dupyshon.com/netflix/best\\_submission.txt.gz](http://www.dupyshon.com/netflix/best_submission.txt.gz)

## *Prerequisites*

We assume a Linux environment (this was tested on Ubuntu). The following utilities must be installed:

- bash shell
- GCC and G++
- Python
- lapack

Also, the Netflix data archive must be downloaded from:

<http://www.netflixprize.com/download>

## *Instructions*

Start in your home directory:

```
cd ~
```

Download the source code:

```
wget http://www.dupyshon.com/netflix/netflix.tar
```

Extract the contents:

```
tar -xvf netflix.tar
```

Place the Netflix data tarball in the netflix directory:

```
mv <netflix data tar.gz file> ~/netflix/download.tar.gz
```

Enter the netflix directory:

```
cd netflix
```

Run the script (it will take several hours to complete):

```
sh reproduce_rmse.sh
```

Submit the file 'ensemble.txt.gz' to the Netflix Prize website.

See the file 'reproduce\_rmse.sh' and the source code for more information on how the algorithms are executed.