# Use of KNN for the Netflix Prize

Ted Hong, Dimitris Tsamis
Stanford University

{tedhong, dtsamis}@stanford.edu

## Abstract

This paper analyzes the performance of various KNNs techniques as applied to the netflix collaborative filtering problem.

## 1. Introduction

In the Netflix collaborative filtering problem, the goal is that given a set of training data $x = \{(u_i, m_i, t_i, r_i)\}$, consisting of a sample of prior movie ratings $r_i$ (an integer from 1 to 5), associated with user $u_i$, movie $m_i$, time $t_i$ to be able to accurately predict the rating that should be associated with a new point $(u,m)$. In this first pass, because we cannot easily ascertain the time associated with the new point we will ignore the time dimension. Furthermore, to simplify the analysis we will not take into consideration any features that could be associated with knowing the actual movie characteristics.

## 2. KNN

Our main premise is that similar users rate similar movies similarly. With KNN, given a point $(u,m)$ to predict, we compute the K most similar points and average the ratings of those points somehow to obtain our predicted rating $\hat{r}$. Different spaces, similarity metrics and different averaging techniques would affect the performance of KNN.

In the following sections we will consider primarily user similarity, ignoring movie similarity and saving that for future work. In essence, our KNN algorithm becomes: given a point $(u,m)$ to predict, compute the K most similar users and average the ratings of those users gave movie m to obtain our predicted rating $\hat{r}$.

We will consider approximations to KNN to obtain predictions in a reasonable amount of time, and several distance metrics.

## 3. Top Q Optimization

In order to calculate the similarity between two users, we consider the movies they have rated and combine them in some way. By looking at every single rating this takes roughly $O(M)$ time were M is the average ratings a user would have. In order to compute the KNN for every user, we need to compute the similarities between all the users for roughly $O(N^2 M \log K)$ time to finish computing KNN and $O(KN)$ space to store the K nearest neighbors for each user. However, many users don't rate a significant number of movies. For these users, it is unlikely that they would help in predicting the rating for a random movie. By this reasoning we should lose little information if we consider only the Top Q users with the most number of ratings. With this, KNN-TopQ would take $O(NQM \log K)$ time to complete and $O(KN)$ space. We will analyze the effect of K and Q after defining our initial distance metric.

### 3.1 Smaller Data Sets

The training data set provided by Netflix was huge, consisting of 100 million ratings. If we were to run our algorithms on that dataset, we would lose a significant amount of time waiting for the method to train. This would impair our ability to test small changes quickly. Therefore we created training sets that are 1000, 100 and 10 times smaller (meaning they have so many times fewer users), along with their respective testing sets. For each size 5 different datasets were created, by randomly selecting users.

## 4. Pearson's Correlation Coefficient

If we consider that a given user $u_i$ rates movies with a distribution $R_i \sim (\mu_i, \sigma_i)$ then a natural similarity metric between users $u_i$ and $u_j$ is the correlation coefficient between the two distributions $R_i$, and $R_j$: $\rho_{ij} = \frac{E[(R_i - \mu_i)(R_j - \mu_j)]}{\sigma_i * \sigma_j}$.

We estimate the covariance and variances by considering the M movies user i and j have in common and

$$E[(R_i - \mu_i)(R_j - \mu_j)] \approx \frac{1}{M}\sum_k (r_{ik} - \mu_i)(r_{jk} - \mu_j),$$

$$\sigma_i \approx \sqrt{\frac{1}{M}\sum_k (r_{ik} - \mu_i)^2}, \quad \sigma_j \approx \sqrt{\frac{1}{M}\sum_k (r_{jk} - \mu_j)^2}$$

But we estimate the means by considering all movies user i/j has rated irrespective of the other users. If a particular user as no ratings, then we consider the user's mean to be 0 (probably should be changed).

The values of the Pearson Correlation Coefficient lie in the interval [-1, 1]. At KNN the values of the similarity function typically lie in the [0,1] interval. A very simple way to convert from the first interval to the second is to consider $(\rho_{ij} + 1) / 2$.

The correlation coefficient $\rho_{ij}$ is a measure of how linearly related the two distributions are. We consider its use solely as a similarity metric and also as a component in the MSE Linear estimate of the rating $\hat{r}$.

### 4.1 Baseline Measure: Average

If the user distributions are all independent, then the best we can do is to consider every user individually and estimate the rating as the mean of all prior ratings of a particular user. With this baseline (user average) we get a RMSE error of around 1.07.

| 1000x Set | RMSE of training Data | RMSE of testing Set |
|---|---|---|

| | | |
|---|---|---|
| 1 | 0.987118 | 1.09584 |
| 2 | 0.983495 | 1.04112 |
| 3 | 0.994448 | 1.06747 |
| 4 | 0.98941 | 1.06254 |
| 5 | 0.979823 | 1.09453 |
| average | 0.986859 | 1.0723 |

## 4.2 ρ as a Similarity Metric

We consider a user j to another user i if $\rho_{ij}$ is close to 1. In other words, when user j rates a movie high (relative to his mean), user i would do similarly. Our KNN algorithm creates a list of K neighbors with high correlation coefficients; with a cap on the minimum similarity it would consider at 0.1 even if the KNN isn't met. Then when it is time to estimate the rank user i would give to movie m we consider the other users in the KNN set that have ranked movie m and compute the weighted average of the rankings: $\hat{r} = \frac{\sum_k \rho_{ik}(r_k - \mu_k)}{\sum_k abs(\rho_{ik})} + \mu_k$.

With this algorithm our RMSE is around 1.09 for K=100 and Q=entire training set (480)

| 1000x Set | RMSE of training Data | RMSE of testing Set |
|---|---|---|
| 1 | 0.884406 | 1.0862 |
| 2 | 0.88278 | 1.05615 |
| 3 | 0.894878 | 1.08438 |
| 4 | 0.904519 | 1.08758 |
| 5 | 0.908141 | 1.11352 |
| average | 0.894945 | 1.085566 |

Despite performing significantly better on the training data set, the algorithm performs slightly worse on testing.

### *4.2.1 Statistics on KNN*

One thing we thought would be interesting to see is along with how many nearest neighbors are we actually using both because of the similarity floor of ignoring those with similarities less than 0.1 and because the nearest neighbors might not have rated the movie being predicted as well.

For the testing data with K=100 and Q=480 as before

| 1000x Set | NN w/ ρ>0.1 | NN actually used | Percentage when NN=0 |
|---|---|---|---|
| 1 | 99.0079 | 5.21094 | 23.8301 |
| 2 | 99.6228 | 5.06192 | 24.622 |
| 3 | 99.4937 | 4.93718 | 25.8684 |
| 4 | 99.0126 | 5.20632 | 22.4561 |
| 5 | 99.7602 | 4.89915 | 24.5416 |
| average | 99.37944 | 5.063102 | 24.26364 |

We also noted by how much did the predicted rankings change from the average baseline

| Set | Average Delta from baseline | Correct Delta from baseline |
|---|---|---|
| 1 | 0.400599 | 0.873973 |
| 2 | 0.396282 | 0.833844 |
| 3 | 0.386784 | 0.84684 |
| 4 | 0.411089 | 0.850112 |
| 5 | 0.395535 | 0.869042 |
| average | 0.398058 | 0.854762 |

Ws can be seen, the prediction is too conservative, its predictions are too close to the mean.

We then considered the same metrics but applied to the same testing data as it was trained on.

| 1000x Set | NN w/ ρ>0.1 | NN actually used | Percentage when NN=0 |
|---|---|---|---|
| 1 | 99.9553 | 5.12469 | 19.2942 |
| 2 | 99.9765 | 5.12071 | 19.4698 |
| 3 | 99.9712 | 5.22575 | 19.2644 |
| 4 | 99.945 | 4.53416 | 21.0865 |
| 5 | 99.9811 | 3.77212 | 27.9462 |
| average | 99.96582 | 4.755486 | 21.41222 |

| Set | Average Delta from baseline | Correct Delta from baseline |
|---|---|---|
| 1 | 0.464651 | 0.795025 |
| 2 | 0.465521 | 0.789292 |
| 3 | 0.45914 | 0.796511 |
| 4 | 0.47082 | 0.797497 |
| 5 | 0.426288 | 0.774789 |
| average | 0.457284 | 0.790623 |

Though the prediction is still too conservative, the prediction spread is closer to the actual spread. However, there still isn't too much of a difference between the behavior on training data and the testing data so the difference has to be in the makeup of the two sets (users and movies to be predicted)

## 5. Sparseness of Training Data

From section 4, though we set K=100, we actually only use around 5 NN when predicting. This shows that preselecting the KNN without regard for the movies might not be such a good

idea. It would be possible to get better NN if we select the KNN off those persons that have rated the movie. This is confirmed by the RMSE of this algorithm which is 1.067, slightly less than that of the average baseline

| 1000x Set | RMSE of training Data | RMSE of testing Set |
|---|---|---|
| 1 | 0.719601 | 1.09166 |
| 2 | 0.717532 | 1.01548 |
| 3 | 0.72067 | 1.07631 |
| 4 | 0.722906 | 1.05834 |
| 5 | 0.718044 | 1.09424 |
| average | 0.7197506 | 1.067206 |

For the testing data:

| 1000x Set | NN (that rated movie) | NN actually used (ρ>0.1) | Percentage when NN=0 |
|---|---|---|---|
| 1 | 31.4464 | 23.5724 | 11.1591 |
| 2 | 31.5335 | 23.5508 | 8.85529 |
| 3 | 30.9727 | 22.9239 | 11.0865 |
| 4 | 32.767 | 24.4133 | 8.49123 |
| 5 | 31.8865 | 23.9147 | 7.54584 |
| average | 31.72122 | 23.67502 | 9.427592 |

| Set | Average Delta from baseline | Correct Delta from baseline |
|---|---|---|
| 1 | 0.304105 | 0.873973 |
| 2 | 0.333018 | 0.833844 |
| 3 | 0.326745 | 0.84684 |
| 4 | 0.335144 | 0.850112 |
| 5 | 0.331236 | 0.869042 |
| average | 0.32605 | 0.854762 |

For the training data:

| 1000x Set | NN (that rated movie) | NN actually used (ρ>0.1) | Percentage when NN=0 |
|---|---|---|---|
| 1 | 34.6919 | 25.8505 | 2.28853 |
| 2 | 33.9762 | 25.0654 | 2.46564 |
| 3 | 35.6775 | 25.3181 | 2.52856 |
| 4 | 33.1333 | 24.1437 | 2.23688 |
| 5 | 30.6005 | 21.9693 | 2.90325 |
| average | 33.61588 | 24.4694 | 2.484572 |

| Set | Average Delta from baseline | Correct Delta from baseline |
|---|---|---|
| 1 | 0.436832 | 0.795025 |
| 2 | 0.436285 | 0.789292 |
| 3 | 0.433401 | 0.796511 |
| 4 | 0.441996 | 0.797497 |
| 5 | 0.433108 | 0.774789 |
| average | 0.436324 | 0.790623 |

## 6. Cosine Similarity

Instead of using the Pearson's correlation coefficient, we could use the cosine similarity to determine the similarity between two users: $s_{ij} = \frac{E[(R_i)(R_j)]}{\sigma_i * \sigma_j}$ which is the same as Pearson's but without normalizing the means of the distribution to 0. We ran several experiments comparing the RMSE of cosine similarity and Pearson's on various K and Q (on 2 larger training/testing set)

| | Pearson's with p [0,1] | Cosine | Pearson |
|---|---|---|---|
| Q=2000 K=50 T1 | 1.00318 | - | - |
| Q=2000 K=100 T1 | 0.986039 | 0.991617 | 1.0049 |
| Q=4000 K=100 T1 | 1.00505 | 1.00639 | 1.0091 |
| Q=1000 K=100 T2 | 1.06067 | 1.06248 | 1.08059 |
| Q=2000 K=100 T2 | 1.05787 | 1.06045 | 1.09093 |

As we can see, cosine similarity performs worse. However, we are at a loss to explain why converting person's inequality to be from [0,1] instead of from [-1,1] would be better.

## 7. Default Values

As we have seen in the past sections, the actual number of nearest neighbors is much less than K. We can fill in these gaps if we also predict the ratings the missing KNN users would give. We call these default values and they are essentially a weighted average of the average rating the particular user rates his movies, the average rating the particular movie received from all users, and the average rating in the whole system:

AvgRating
+ (AvgUserRating(i) - AvgRating) * 1.0 / (1.0 + exp (-NumUserRatings(i) / A))

+ (AvgMovieRating(j) - AvgRating) * 1.0 / (1.0 + exp (-NumMovieRatings(j) / B))

The above baseline has the advantage that it predicts different ratings per user and per movie. The constants A and B were specified by members of another team, based on hand tests.

In order so that these default ratings do not drown out the real ratings, we weight these default values by 1/5. We get slightly better results on the same training sets as in section 6.

| | Pearson's with p [0,1] | Cosine | Pearson |
|---|---|---|---|
| Q=2000 K=50 T1 | .975016 | - | - |
| Q=2000 K=100 T1 | 0.972373 | .999309 | .972449 |
| Q=4000 K=100 T1 | .97344 | .999309 | .975443 |
| Q=1000 K=100 T2 | 1.05698 | 1.05596 | 1.0564 |
| Q=2000 K=100 T2 | 1.05206 | 1.05483 | 1.05121 |

## 8. Confidence Intervals

So far most of our tests were on the small 1000x dataset, which only contains 480 users. We were able to test KNN on larger data sets to get a clearer view of how its performance correlates to the number of neighbors that are used during the prediction phase. We tried various values for Q, which lead to different sets of k-nearest neighbors being chosen each time.

100x data set (k = 100)

| | RMSE | NN (that rated the movie) |
|---|---|---|
| **Q = 500** | 1.01729 | 34.1386 |
| **Q = 1000** | 1.0284 | 25.1022 |
| **Q = 2000** | 1.06434 | 13.4119 |
| **Q = 5000** | 1.14393 | 3.00694 |

10x data set (k = 100)

| | RMSE | NN (that rated the movie) |
|---|---|---|
| **Q = 500** | 1.0129 | 48.3191 |
| **Q = 1000** | 1.01 | 44.1402 |
| **Q = 2000** | 1.01093 | 38.8236 |
| **Q = 5000** | 1.0258 | 28.5866 |
| **Q = 8000** | 1.04374 | 22.0923 |

We observe that as the number of nearest neighbors that are actually used decreases, the RMSE increases. What is more interesting to observe is that NN decreases as Q increases. This is explained as follows: the smaller the Q, the more movies the neighbors have rated and thus it is more probable that they have rated the movies in the testing set. As Q increases, we discover users with higher similarity, but this turns out to be negative since they have rated less movies. The fact that users with higher similarity are discovered is not only due to the larger search space, but also because they have rated less movies and the similarity is computed upon less common ratings.

As a counter-measure for this phenomenon, we decided to penalize the similarity of users that have rated few common movies. From the results we can see that this feature does pay off, since the number of neighbors used gets higher and the RMSE gets better.

100x data set (k = 100)

| | RMSE | NN (that rated the movie) |
|---|---|---|
| **Q = 500** | 1.01697 | 35.0486 |
| **Q = 1000** | 1.02005 | 28.8928 |
| **Q = 2000** | 1.03313 | 21.0512 |
| **Q = 5000** | 1.05468 | 14.182 |

10x data set (k = 100)

| | RMSE | NN (that rated the movie) |
|---|---|---|
| **Q = 500** | 1.01132 | 48.7446 |
| **Q = 1000** | 1.00582 | 45.8686 |
| **Q = 2000** | 1.00316 | 41.82 |
| **Q = 5000** | 1.00395 | 34.4633 |
| **Q = 8000** | 1.00836 | 29.6732 |

It is interesting to note that for the 100x data set the lowest Q still gives the best results, while for the 10x data set the best Q is 2000. This indicates that there needs to be large enough Q to contain the "working set" of the training data but too large Q decreases the quality of the predictions by having not enough ratings in the nearest neighbor set.

In order to discount similarities based off the number of ratings, we decrease the Pearson's similarity by one standard deviation in the method of [1]. Essentially we convert the similarity to a z-score via Fischer's z' transformation [2].

$$z' = \frac{1}{2}\left(\log(\rho + 1) - \log(1 - \rho)\right)$$

And then decrease the value by one standard deviation:

$$\sigma = \frac{1}{\sqrt{CommonRatings - 3}}$$

And convert back to a similarity metric

$$\rho = \frac{e^{2z'} - 1}{e^{2z'} + 1}$$

In the end, if there are not enough common ratings, the standard deviation would be large, forcing the z-score to be low and close to zero, and hence the similarity of the two users will be low and close to 0.

## 9. Movie Similarity and K-NN

One problem mentioned before is that running KNN on the entire data set is too prohibitive. Top-Q was used as an optimization but past 2000 users, taking more users on is actually harms the results. If the poor results are due to these similar users not having enough ratings, perhaps we can artificially introduce ratings. Default values based on averages were analyzed in Section 7 and were

found to perform only slightly better. Instead of using baseline approach we can also the similarity of movies to help improve the ratings. Essentially, if a nearest neighbor happens to have not rated the movie, we would find a similar movie and make the prediction based off that. We implemented movie similarity just like we did user similarity: pearson's similarity of two movies based off of common user ratings. However, we found that using movie similarity as default values actually hurt us. Furthermore, we created a movie similarity baseline: instead of rating based off the average, we take the weighted average of similar users the user to be predicted has already seen. But saw that though the average baseline on the 1000x set gives around 1.07 RMSE, using movie similarity as a baseline gets 1.09 RMSE. While using movie similarity would help with problem of sparseness the method still needs tweaking.

## 10. Clustering

One idea we had was that if we split up similar users, we could run KNN separately for each cluster, speeding up the calculation and potentially increasing accuracy as only similar users would ever be considered in the first place. In order to test this out, we used the K-Means Clustering results from Brian Sa and Patrick Shih for 10 clusters. For the 1000x smaller data set we get RMSE of around 1.14 and use around 2 neighbors. For the 100x we get RMSE of around 1.08 and use around 10 neighbors. We also ran the algorithm for the entire training set and found similar results: with clustering we are able to find fewer neighbors and our RMSE is harmed. Just like high topQ, a simple clustering approach hurts RMSE by decreasing the available nearest neighbors.

## 11. Conclusion

As shown by its performance on the training rather than testing set, KNN has the potential to perform good predictions: with RMSE lower than 0.80. However, the RMSE for testing hovers around 1.00. We found that techniques that required higher more accurate statistics or higher nearest neighbors to performed worse though they has the potential to perform better; vector MMSE linear estimators, and clustering were two examples. Techniques that attempted to increase the accuracy of the statistics, or increase nearest neighbor count like default values, and clustering improved RMSE by a small amount. Movie similarity was a hybrid approach, it required more statistical data but its purpose was to increase nearest neighbor count; it just happened to perform worse. More complex predictors would not help unless the first problem, sparseness of the data, is taken care of.

## 12. Future Work

The way to attack problems like the Netflix prize is to create some hybrid method, combining more than one model. We understand that other teams are working on different approaches on the problem and once they are finished we could use their results to enhance the KNN model.

KNN as it is does not exploit extra information about the movies, like their genre, director, actors, etc. This information can be extracted from the IMDB and could be used a simple content-based prediction model. We could then use this content-based model to create the default values of the prediction stage of KNN. We could also go as far as filling up the entire user-movie matrix before computing the user similarities. Melville et al [3] propose one way such an approach could take.

We think clustering on users could still be very useful. We could search for the nearest neighbors only within the cluster that a user belongs to, finding neighbors that are more strongly correlated. Given reasonably sized clusters we could abandon the topQ optimization and search all the users in the cluster. However, we'll need some method to incorporate users with high number of ratings into multiple clusters so that the nearest neighbor count does not drastically decrease due to clustering.

Another approach is to cluster on movies and create different similarities for every movie cluster. For example, when we examine movie cluster 1, we would only consider only movies that belong to this cluster during the phase of the similarity generation. Then, for each test we would use the similarities that belong to the same cluster as the movie of the test.

Finally, considering that the rankings are integers, we should examine ways to round off our results to an integer value. Real values would be useful if we wanted to implement a recommendation system, but since for the competition we only wish to predict the ratings, integer predictions could improve the RMSE. Of course, this would first require to improve our algorithm, so that the classification would get us closer to the test value and not create larger errors. Much like how logistic regression generally outperforms linear regression on discrete data; we expect a increase in accuracy if we're able to come up with a smart method to discretize the results.

## 13. Acknowledgements

Thuc Vu coded the initial KNN algorithm and helped with the direction of the project; so did Chuong Do. Brian Sa and Patrick Shih created the K-Means clustering program and clustering results we used. We also thank the rest of the Netflix team for sharing their ideas and successes.

## 14. References

[1] Ali, K. and van Stam, W. 2004. TiVo: making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the Tenth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining* (Seattle, WA, USA, August 22 - 25, 2004). KDD '04. ACM Press, New York, NY, 394-401. DOI= http://doi.acm.org/10.1145/1014052.1014097

[2] http://davidmlane.com/hyperstat/A98696.html

[3] Melville, P., Mooney, R. J., and Nagarajan, R. 2002. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth National Conference on Artificial intelligence* (Edmonton, Alberta, Canada, July 28 - August 01, 2002). R. Dechter, M. Kearns, and R. Sutton, Eds. American Association for Artificial Intelligence, Menlo Park, CA, 187-192.