CS229: The Netflix Project

Jane Huang, Jack Kamm, Conal Sathi

December 12, 2008

Abstract

This paper investigates the combination and application of a number of machine learning applications to the Netflix Challenge. The algorithm uses extra data in addition to the Netflix training set. Namely, it uses a mapping from Netflix to features gleaned from IMDB, such as the director and genre. Using k-means clustering, the algorithm first clusters the users together by the IMDB features each movie has. We tried making predictions through principally three methods: using Naive Bayes alone on the IMDB features, the average rating that a cluster gives to a movie, and through combining Naive Bayes with clustering.

Introduction

The Netflix Challenge

Given a user and a movie, we need to predict what rating the user gave me the movie. As training data, we are given a list of vectors (u, m, r, t), where u is a user ID, m is a movie ID, r is the rating u gave to m, and t is the date.

After training, we output predictions for a list of user-movie pairs. We measure error by using the root mean squared error (RMSE).

The Netflix Challenge is to get an RMSE of or below 0.8572. The data set comprises over 100 million movie ratings from over 480,000 customers.

General points on methodology

To make things faster, we ran our algorithms on a random set of 1000 users. We used simple cross-validation to test our results; we removed one movie for each user from the training data and placed it in our test set.

To augment the training set Netflix provided, we used a mapping that maps each movie to a list of features it contains, such as genre, director, and actors. The mapping only consisted of those features that appeared in 25 movies or more. There are 1071 such features. For comparison, a mapping with features that appear in 2 movies or more has 84000 features. We chose to use the smaller feature mapping because it would be faster to use and also because features that appear in very few movies wouldn't be very useful for making predictions.

Algorithms

Naive Bayes

The first algorithm we tried was simple naive Bayes. For a given user u, for a movie m with features $F = (f_1, f_2, \ldots, f_n)$, we predicted the rating r that maximized $p(r; u)p(F|r; u)$. Using the naive assumption that all features are independent, we get that $p(F|r; u) = \prod_{i=1}^{n} p(f_i|r; u)$. We used the Maximum Likelihood Estimates with Laplace Smoothing to estimate $p(f_i|r; u)$ and $p(r; u)$.

We tried using Naive Bayes because it could take each user's individual preferences for specific features into account. However, the RMSE we got was 1.2751, which is not very good. For comparison, if you simply predict the average rating of each user for each user-movie pair, we get an RMSE of 1.0635. One obvious problem with Naive Bayes is that it assumes each feature is independent, which is not true. For example, a director may prefer to work with particular actors. A bigger problem is that users have not seen each feature enough times to make good predictions: a single user will not have watched all that many movies, and each movie only has a handful of features.

To remedy the latter problem, we also tried using Naive Bayes with clusters. We clustered similar users together (see the following section), and then when making predictions we treated each cluster as a single user. So, for example, if user u belongs to cluster C, for a movie m with features $F = (f_1, f_2, \ldots, f_n)$, we predict the rating r that maximizes $p(r; C) \prod_{i=1}^{n} p(f_i|r; C)$, where we estimate the probabilities using the Maximum Likelihood Estimate with Laplace Smoothing. So, for example we estimate $p(f_i = 1|r; C)$ to be the number of times we have seen a movie with feature f_i get rating r by any user in C, divided by the number of times we have seen a movie get a rating r by any user in C, and then add one to the numerator and two to the denominator.

After clustering we got better results. Our best results were with 40 clusters: we got an RMSE of 1.1493. If we have too many clusters, clusters are too small and we don't have enough data per cluster to make accurate predictions. If clusters are too large, there are too many dissimilar users in a single cluster.

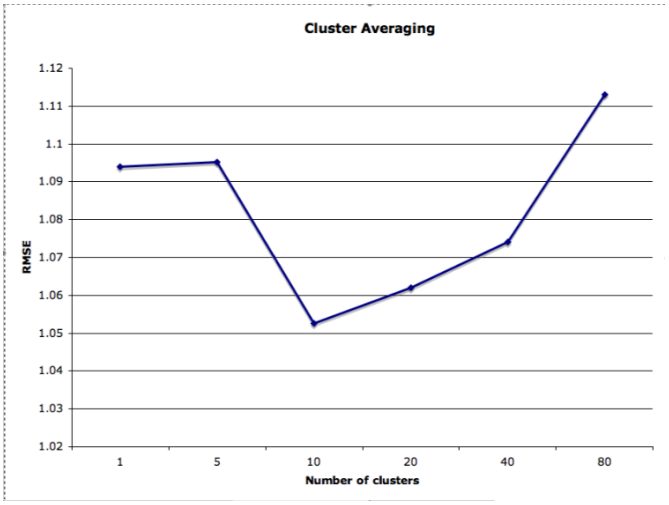
k-means Clustering

A typical way to predict how a user will rate a movie is to see how similar users rated the same movie. So we clustered similar users together, and then used information in the entire cluster to make predictions for any individual user in the cluster.

We decided to cluster users together based on their tastes about IMDB features. In particular, for each user and each feature, we calculated the expected rating the user would give a movie with that feature. We then placed the user in a high dimensional space, where each dimension is a IMDB feature, and the coordinate of the user along that dimension is the expected rating the user gives that feature. We then used k-means (with the Euclidean norm) to cluster users.

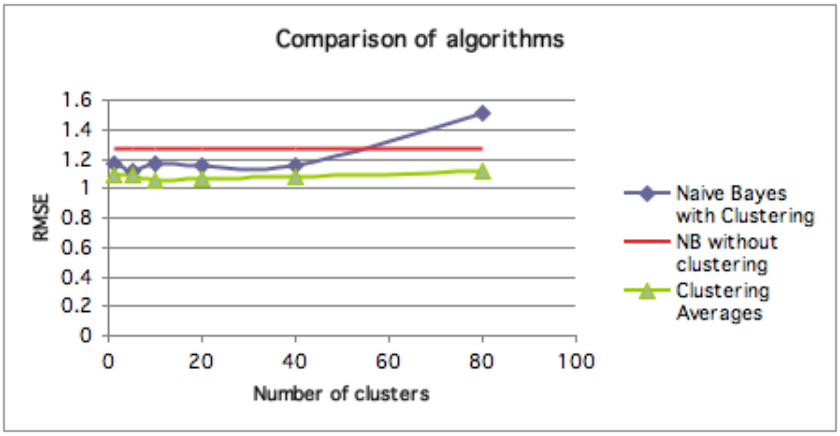
To make predictions with clusters, we use two methods. Firstly, we use cluster averaging. When we want to make a prediction for a movie, we look at the other members of the user's cluster to see how they rated the movie, and take the average. If we're predicting a rating for a movie that nobody else in the cluster has watched, then we use the user's own average

movie rating. We ran the cluster averaging algorithm on varying cluster sizes, and discovered that the best number of clusters to have was ten, which implies an average of about 100 users per cluster. The following graph shows how the RMSE varies with the number of clusters used.



The shape of the graph is convex, which makes sense. The fewer clusters you have, the more overfitting you do, because users within clusters becoming increasingly dissimilar. However, the more clusters you have, the less information you have with each cluster to work with. It appears that for a set of 1000 users, ten clusters finds the right balance between overfitting and a lack of information per cluster.

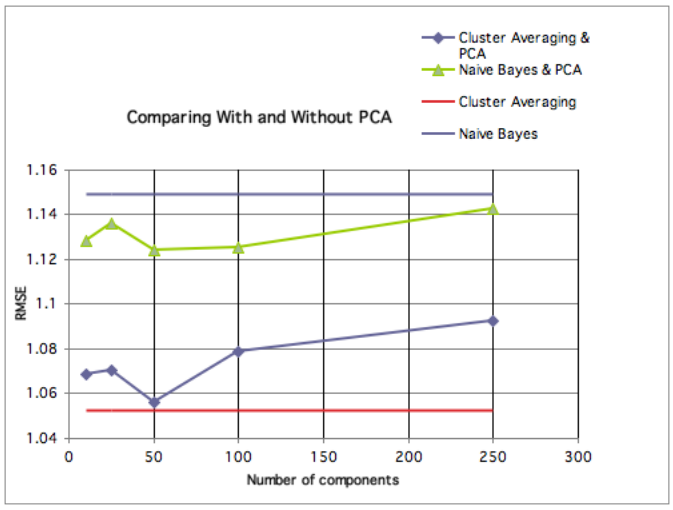
Surprisingly, combining the algorithms is not very helpful. Using cluster averaging yields a better RMSE than combining clustering with Naive Bayes.



It makes sense that Naive Bayes combined with clustering does better than Naive Bayes alone. See Naive Bayes section. It is surprising that simply using cluster averaging works better than Naive Bayes combined with clustering, however, since Naive Bayes uses more information about users' tastes. Naive Bayes may perform worse because it fails to take into account the "intrinsic value" of a movie. For example, The Matrix and The Matrix 3 probably have similar features, but The Matrix is a far superior movie. By using cluster averaging, however, we use the average rating given by the cluster, which is going to be a lot higher for The Matrix than for The Matrix 3.

Principal Components Analysis

Since we had 1000 data points that each had over a thousand features, it seemed wise to prune back the number of features through principal components analysis before making any clusters. We hoped that though applying PCA would cause some information loss as it compressed 1071 dimensions into k dimensions, the information loss would be compensated for by reducing a lot of noise at the same time. In addition, it might have helped the k-means clustering algorithm run faster. In our experiments, we ran PCA on the optimal cluster sizes we found. So for cluster averaging, where the optimal number of clusters was ten, we used ten clusters, and for clustering combined with Naive Bayes, we used forty clusters. The following graph shows the results we got while varying the number of components used.

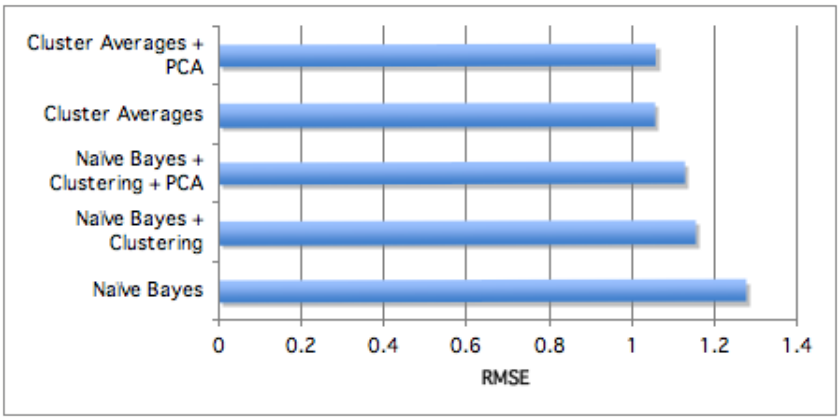


Running PCA prior to running cluster averaging alone does not improve RMSE. Despite tweaking the number of components for ten clusters, we are unable to achieve a lower RMSE than for ten clusters without PCA. This may be because PCA encourages overfitting and user tastes are highly idiosyncratic. Once we chose the best number of components however, RMSE was nearly the same as the best RMSE for cluster averaging, so choosing an optimal number of components might not hurt results.

However, running PCA to form the clusters before running Naive Bayes combing with clustering does seem to help the RMSE a little bit. These results are unexpected, however, since for using Naive Bayes we use all the features, not the set of compressed features created by PCA. It does not seem like PCA should have an effect at all. The benefit seen from PCA could simply be random noise.

Conclusion

The following graph displays the best RMSEs achieved with each algorithm:



We used a few different approaches to try and make predictions for the Netflix Challenge. Unfortunately, our algorithms had limited success. The least successful approach was Naive Bayes, in large part because we did not have enough data for an individual user. When we combined clustering with Naive Bayes, we got better results. However, simply taking averages across each cluster does a better job than Naive Bayes. This may be because Naive Bayes fails to take into account the "intrinsic value" of a movie; even though a movie has features a user likes, the movie might still be poorly made. Finally, we tried using PCA to help with clustering. PCA made clustering faster but made the clusters less accurate when we just took averages across clusters. The clusters were slightly more accurate for Naive Bayes.

There are a couple of things we could do to improve our algorithms. For Naive Bayes, we could try to take into account the "intrinsic value" of a movie by offsetting our prediction by the average rating of the movie. For clustering, we could take into account more information than just the expected rating that a user gives a feature. For example, the variance of a rating that a user gives a feature shows how important a feature is to the user (a low variance implies that the feature is more important to the user). To improve clustering, we could turn each feature into 5 dimensions, one for each rating, and along each dimension plot the probability that the user will give the feature that rating.

References

Grigorik, Ilya. "Correlating Netflix and IMDB Datasets." January 1, 2007.
Ng, Andrew. Lecture notes. December 2008.