Revisiting the Netflix Prize the Decade After

Category: General Machine Learning

Victor Cheung
Computer Science Department
Stanford University
hoche@stanford.edu

Evan Huang
Electrical Engineering Department
Stanford University
evhuang@stanford.edu

Abstract

The Netflix Prize rewarded the top team with an incremental 10% improvement on their existing Cinematch algorithm with a million dollars. It turns out that improving the RMSE of predictions is difficult. Furthermore, the winning solution was cumbersome and infeasible for commercial use.

We will study the Netflix problem in a simplified setting under discretization, and combine prior information calculated on the dataset to make predictions on each individual movie. This method achieves nearly the performance of KNN at 75% accuracy in the discretized setting but can be up to 420x faster.

1. Introduction

The Netflix Competition caused quite a stir in the data science community when it was first announced. Products built on the back of recommendation systems were blooming — the likes of Amazon relied heavily on recommendation engines to suggest products for customers.

The problem presented by Netflix is as follows: given a dataset of 480,000 distinct users and their ratings from 1 to 5 over 17,000 movies, from October 1998 to December 2005, we want to predict ratings for movies a user hasn't watched. In this way, we can gauge which movies to recommend to which user.

The Netflix Prize was eventually won by an ensemble solution combining the techniques of the top teams. However, the system was sufficiently cumbersome that it was not implemented by Netflix for internal use. We will discuss the ensemble solution in the related work section.

Given these results, an interesting line of inquiry then is whether we can build an efficient system of inference with allowance for some time for precomputations. To do this, we will use two different approaches: Nearest Neighbour type methods and Bayesian inference. We will find i) that nearest neighbours suffers from runtime quadratic in the number of users and linear in the number of movies, and ii) that bayesian inference results in predictions under a discretized setting almost as good as nearest neighbours, and with only constant time inference for every movie and every user.

Thus, our input was a large sparse matrix of user ratings for movies. We then use nearest-neighbor methods and Bayesian inference to predict user ratings for unseen movies.

2. Related Work

Geoffrey Hinton tackled the netflix problem with a model he invented called restricted boltzmann machines. It's a type of generative model that tries to capture the joint distribution over visible data (units) and latent (hidden) variables. The latent variables hope to capture those underlying preferences for users and categories for movies. The joint distribution is learned by sampling techniques (Gibbs sampling) that aim to minimize the reconstruction loss when predicting the visible units from learned latent variables. He calls this "contrastive divergence". Hinton ultimately achieved a 6% improvement on the netflix benchmark after 20 epochs of training.

The "BellKors Pragmatic Chao" solution was an ensemble solution that achieved the desired 10% improvement over the Netflix baseline. It first poses a variety of models (what they call predictors) based on linear regression with different regularization weights, restricted boltzmann machines (as above), and nearest-neighbour methods with time-based covariates (ex. the number of ratings given by one user on a single day). It then uses gradient boosted trees (a kind of ensemble model that uses many decision stumps that iteratively penalizes outliers) to combine the some 474 different hand-crafted predictors to produce a "blended" output. Good results here depend on extracting and representing the same underlying trends in as many forms as possible and hoping that the gradient boosted trees will be able to extract the relevant signals and make good predictions.

3. Data Analysis and Data Infrastructure

The Netflix dataset contains more than 100 million observations, yet is sparse, since the matrix could hold more than 9 billion entries. Additionally, we are deprived of any explicit information about the movies or the users themselves. We might expect that certain demographic traits will inform the preferences of a user; alternately, we might expect that certain genres, stars, directors, and so on, might inform the likely audience of a movie. These are all missing from the dataset. It is this sense of sparsity that prevents the usage of various popular techniques in deep learning, since these learn representations based on visible dense features.

Given the large size of the dataset, naive representation as numpy matrix would result in a matrix holding approximately 100 million entries amongst approximately 9 billion slots. A matrix of this size is 34GB in memory assuming 32bit integer representations. Since numpy holds matrices in RAM, this is clearly infeasible. Furthermore, matrix operations are computationally costly, resulting in long run times. To efficiently operate on the data, the ratings corresponding to each user and movie are stored in a compressed sparse row (CSR) matrix. With this format, the rows represented the user, and the columns the corresponding movie. With an appropriate choice of integer representations, this results in a matrix of size approximately 200MB.

We begin our analysis with summary statistics of the training data provided. With a mean of 3.604 and a standard variation of 1.085, the data seems centered around a rating of around 3, showing that an average rating is in fact on the higher end. A median of 4 also expresses a higher number of 4s and 5s versus 1s, 2s, and 3s. Through visualization of a histogram showcasing frequency of ratings, we can see that a majority of users rate movies as either a 2 or 3. And 4s are more common than 1s.

Our initial hypothesis that users tend to be extreme in their ratings is thus proven incorrect — instead, users seem to be ambivalent (3 or 4) most of the time. This complicates the situation, since extreme behavior is easier to predict and model. Ambivalence implies the possibility of significant noise included in the data. That is, information that has little to do with the movie itself is factored into the ratings. Concretely, the difference between a 3 star or 4 star may be due to cognitive biases (the availability heuristic of what movies a user has recently watched, etc). This will likely bias any methods that relies on the assumption that users rate movies due entirely to the quality of the movie.

3.1. Discretization

Given the above, instead of trying to capture the ambivalence of user, we use strong-thresholding to discretize ratings into two bins. Thus, for discretization, we choose a threshold T such that for all ratings $R \leq T$, $R' = 0$ and $R' = 1$ otherwise. Given above analysis, we choose $T = 3$. This corresponds to users either "liking" or "disliking" a movie. In this simplified setting, we can apply certain assumptions that we will discuss in the methods section that allows us to efficiently learn a set of latent population parameters that characterize how different users give out different ratings.

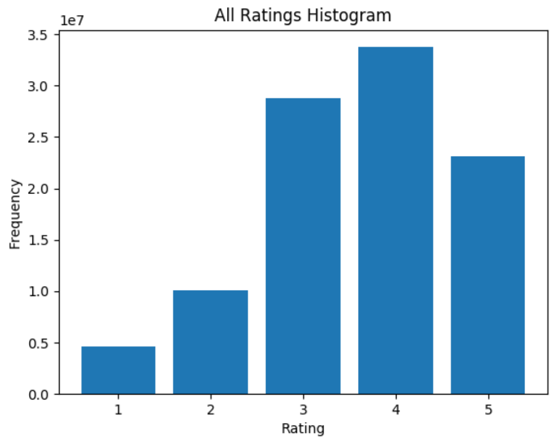

Figure 1. Histogram of Score Distributions

4. Proposed Methods

We ran variations of the following models on an appropriate choice of the dataset.

4.1. Baseline: Neighborhood Based Models

Given a user u, we will define the $neigh_k^{C'}(u)$ as the set of users $\{v_1, \ldots, v_k\}$, $v_i \neq u$, most similar to u in the l_2 metric over a minimum set $C = \{m_1, \ldots, m_c\}$ where we have ratings from all k neighbors on the set C. Furthermore, let $C' = C \cup \{m\}$ where m is the movie we wish to make a prediction on for u. We also require that $neigh_k^{C'}$ contain ratings for m. Then the prediction for u is simply the average of ratings in the neighborhood of u. With test user u, neighbor j, and respective ratings R_u and R_j for $|C| = N$, the distance metric is:

$$l_{uj}^2 = \frac{||R_u - R_j||_2^2}{N}. \quad (1)$$

This choice of architecture is informed by the following considerations. We understand that users which have not watched the same movies as our test user will not be suitable neighbors. That is, there is little to compare unless we considered genre, director, etc of the movie, which would complicate the model. Thus, we will instead omit this data under a baseline minimum number of 5 common movies, and will rely on the large amount of data to make up for this loss. We also wanted to consider the number of movies watched. Given the formula for euclidean distance, we normalize the distances between our test user and a neighbor by dividing by number of movies watched.

4.2. KNN with Variable Distance Scaling

We also propose the use of various distance scalings for more accurate predictions. We classify our additional methods as *constant scaling*, *distance weighted scaling*, and *cosine distance*. Constant scaling weights each neighbor observation j by $(k - j)$ to higher weight closer neighbors. The calculation for each prediction of user u is then

$$\hat{R} = \sum_{j=1}^{k} \frac{(1-k) * R_j}{N}. \tag{2}$$

Distance weighted scaling uses a weighted average of u's neighbors to again higher weight closer neighbors. This methodology uses the inverse l^2 metric (l^2 similarity) as the weight, and creates predictions based on

$$\hat{R} = (\sum_{i} l_{ui}^2) * (\sum_{j=1}^{k} \frac{R_j}{l_{uj}^2 * N}). \tag{3}$$

The last KNN methodology is similar to the baseline but with the cosine similarity distance instead of the l^2 norm. With test user u, neighbor j, and respective ratings R_u and R_j for N movies watched, the cosine similarity metric is:

$$cos_{uj} = \frac{R_u \cdot R_j}{||R_u||_2 ||R_j||_2}. \tag{4}$$

With these alternative methods of KNN, we expect better RMSE as nearer neighbors should have provide more accurate predictions.

4.3. Binomial KNN

Given the clear separation between movies with ratings above the threshold T, and movies below T, a simpler question of interest is whether we can predict a user likes or dislikes a movie. We have then changed the problem to a binomial classification problem. We can then change the KNN method by reclassifying predictions in the same way.

4.4. Bayesian Inference

Suppose we have a joint distribution $J(M, U, R)$ that captures the interaction between movies, users and ratings. Then for a given user and a given movie, we can predict R as

$$argmax_R P(R|M, U) \propto P(M)P(U)$$

where M is a prior capturing the likelihood of a movie being highly rated, and U is a prior capturing the likelihood of the user of interest to like any movie. We can infer these priors from data using maximum likelihood given sufficient observations per movie and per user. Therefore, for movies/users with a sufficiently large number of observations, we use the standard MLE estimate where for a dataset of size m,

$$\hat{M} = \sum_{i=1}^{m} \frac{1[M_i = 1]}{1[M_i = 1] + 1[M_i = 0]}$$

$$\hat{U} = \sum_{i=1}^{m} \frac{1[U_i = 1]}{1[U_i = 1] + 1[U_i = 0]}$$

However, given data sparsity, there are movies and users for which we have very few observations. Maximum likelihood estimates fail here since estimates will be highly biased. In particular, this problem generalizes to settings whenever a new user arrives, and we have no or very little information about the user's prior preferences. The same goes for new movies. One way to resolve this problem is to learn a population prior that we assume holds for the new user/movies. How we do this is described in the following section.

4.4.1 Optimally Learning Population Parameters

In the discretized setting, our estimation problem can be precisely formulated. Suppose there are n users, each with an unknown parameter p_i, and we observe n independent random variables $X_1, \ldots, X_n$ with $X_i \sim Binomial(t, p_i)$. If we further discretize the ratings into "like" and "dislike", we may be able to make a statement about each individual's latent parameter by using information from the entire set of observations across all n individuals.

Our estimate for the set of the population parameters is an ϵ-net of size m, where we solve a linear (or quadratic) program that minimizes the distance between the empirical moments and the population moments (from the ϵ-net). The intuition for method of moments is given by our knowledge about moment generating functions. We know that two distributions P, Q are the same if their moment generating functions are the same. Then, if their moments match up to the t^{th} moment, we can make some reasonable statement about how "close" the two distributions are, given by the Wasserstein distance

$$||P - Q||_W = sup_{f \in Lip_1} \int f(x)(P(x) - Q(x))dx$$

where P, Q is supported on $[0, 1]$ where Lip_1 refers to the class of functions where there exists a constant $K \geq 0$ such that $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$ for any x_1, x_2. Valiant's approach gives the bound $O(\frac{1}{t})$ for the distance between the CDF of true population parameters and the estimated parameters through moment-matching.

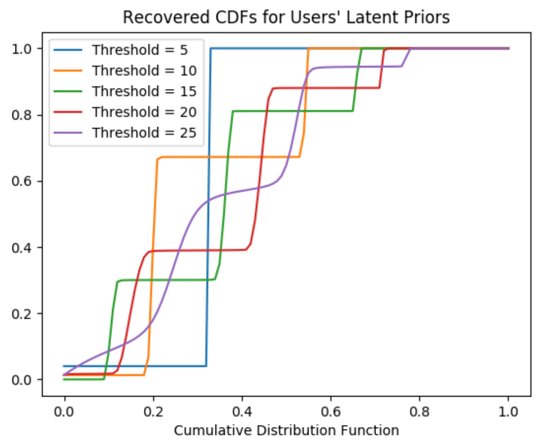

Figure 2. Recovered CDF for Users' Latent Parameters; threshold refers to the cut-off for the assumption that we have enough observations for a robust maximum-likelihood estimate.

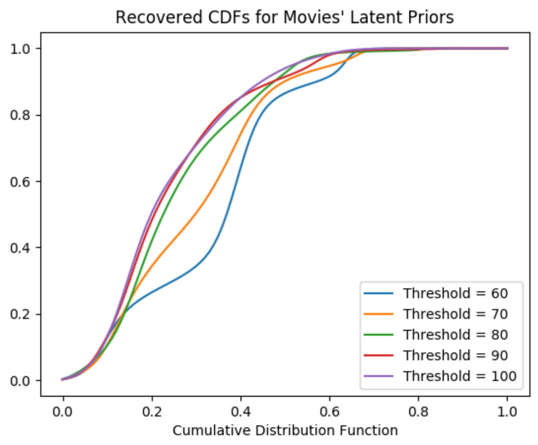

Figure 3. Recovered CDF for Movies' Latent Parameters; threshold is defined similarly as before.

For $1 \le k \le t$, let the empirical estimate of the k^{th} moment be

$$\beta_k = \frac{1}{n}\sum_{i=1}^{n} \frac{\binom{X_i}{k}}{\binom{t}{k}}$$

Then for a vector $q = (q_0, \ldots, q_m)$ of length $m+1$ representing the ϵ-net where q_i represents the amount of mass for the value $\frac{i}{m}$, we solve the linear program

$$\min \sum_{k=1}^{t} |\hat{\beta}_k - \beta_k|$$

where $\hat{\beta}_k = \sum_{i=1}^{m} q_i(\frac{i}{m})^k$ subject to the constraints that $q^T\mathbf{1} = 1$ and $q_i \ge 0$.

We use this technique to infer the latent parameters of users/movies with few observations with $m = 100$. We will then assign the argmax of the ϵ-net to be the population prior that we assume for users and movies without sufficient observations for a robust maximum-likelihood estimate.

5. Results and Discussion

Let's first consider the results of the estimation procedure described in section 4.1.1.

First consider figure 1. We find that the population prior is tightly concentrated around 0.05 to 0.4 for a sufficiently large prior. Intuitively, this means that movies tend to be disliked unless there are specific redeeming factors. That is, users tend to be harsh when rating movies unless the movie is good. We also find that there are two distinct segments in the CDF for threshold = 60, corresponding to the sharp rises in the CDF around specific points on the support; this implies that amongst the population of movies with few observations, that some movies are considerably worse-performing than others. However, as we increase the threshold, we see these distinct segments disappear, likely because they're smoothed over during moment-matching.

Now consider figure 2. It's clear that estimates of the underlying priors are very noisy and heavily contingent on the choice of threshold. It's not immediately clear how to choose a threshold without referring to their performance in inference. Searching over suitable thresholds then becomes important. It's worth noting however that amongst users with very few observations that the prior found assumes they're biased towards disliking movies. We also find that by increasing the threshold that the CDF begins to interpolate between the extremes of lower thresholds resulting in smoother curves.

Our hyperparameters include the thresholds for the latent-parameter estimation for Bayesian inference models, and different notions of distances and weighting methods for the nearest-neighbour models. For Bayesian inference, we chose hyperparameters via grid search over a set of predefined thresholds, where our dev set is a set of randomly sampled users from the entire dataset. For nearest neighbours,

In the discretized setting, we now have a binary classification problem. The relevant metrics will tell us how many true positives and true negatives we've captured, how many incorrect classifications of either type (i.e. false negatives and false positives), how sensitive we are to positive and negative signals, and so on. These are quantitatively captured by the notions of accuracy, precision, recall. To evaluate overall model performance, we use a geometrically weighted metric called the F1-score. Suppose we have true labels y, and predicted labels $\hat{y}$, both of length m, then

$$Accuracy(y, \hat{y}) = \frac{1}{m}\sum_{i=1}^{m} 1[y_i = \hat{y}_i]$$

$$Precision(y, \hat{y}) = \frac{tp}{tp + fp}$$

$$Recall(y, \hat{y}) = \frac{tp}{tp + fn}$$

$$f1 - score(y, \hat{y} = \frac{Precision(y, \hat{y}) * Recall(y, \hat{y})}{Precision(y, \hat{y}) + Recall(y, \hat{y})}$$

where $tp = \sum_{i=1}^{m} 1[y_i = 1, \hat{y}_i = 1], fp = \sum_{i=1}^{m} 1[y_i = 0, \hat{y}_i = 1], tn = \sum_{i=1}^{m} 1[y_i = 0, \hat{y}_i = 0], fn = \sum_{i=1}^{m} 1[y_i = 1, \hat{y}_i = 0]$.

Classifiers	Accuracy	Precision	Recall	F1-Score
KNN - Bernoulli	0.751	0.735	0.694	0.712
KNN - L2	(0.957)			
KNN - CS	(0.930)			
KNN - Cosine	(1.034)			
M60 U5	0.748	0.683	0.748	0.714
M70 U10	0.710	0.667	0.685	0.676
M80 U15	0.743	0.702	0.735	0.718
M90 U20	0.703	0.629	0.697	0.661
M100 U25	0.730	0.683	0.680	0.682
M80 U15	**0.743**	**0.702**	**0.735**	**0.718**

Table 1. Classification metrics under discretized setting with KNN as comparison. M80 refers to threshold for movies set at 80 observations. Similarly for U15, with threshold set at 15. CS refers to constant scaling. Accuracy for KNN-L2 and KNN-CS refers to RMSE. Results are evaluated over a uniformly randomly sampled subset of the dataset.

For Bayesian inference, we see the F1-score peaks for the M80 U15 model. The resulting accuracy (and F1-score) is comparable to the K nearest-neighbour under the discretized setting. This is surprising. KNN methods explicitly remove noisy data points from the by finding a similarity score between users, and making an inference based on the most similar users alone. The resulting estimates are likely to be more robust. However, we see that Bayesian inference alone, based on precalculated priors for movies and users that are likely noisy, can match the classification performance of KNN, which is much more computationally heavy. This may be because inference from priors incorporates all available knowledge about a specific user and a specific movie.

With KNN, we see comparable results for all methodologies described in section 4. Under equal weighted KNN with l^2 distances, we achieved precise results, obtaining an RMSE of 0.957. This matches results of others attempting the same classification on the Netflix dataset. Using a distance weighting leads to an RMSE similar to the baseline KNN methodology, as the distance measurements with top neighbors are similar. However, a constant scaling produces a significantly lower RMSE of 0.930. This reduced RMSE suggests that the nearer the neighbor, the more strongly his/her rating acts as an indicator for the interested user's rating. Cosine distances led to a higher RMSE of 1.034, suggesting a less optimal distance measure for the finding of nearest neighbors.

With regards to computation time, we find that precomputing priors scales well with number of users ($O(1)$ inference time) when compared to KNN ($O(n)$ inference time). Then to compute a set of preferences for the entire user set over some movie, Bayesian inference takes time $O(n)$ and and KNN takes time $O(n^2)$. Consider also the time complexity of incorporating additional information. Updating priors is $O(m)$ in the number of new data points m, whereas KNN will be $O(N + m)$ for incorporating new data. This implies that Bayesian inference is more efficient and may be more suitable in industrial, online settings.

6. Conclusion and Future Directions

We showed that KNN models achieve a relatively low RMSE, with constant scaling KNN performing best. These baseline models suggest that a large amount of predictive power in estimating user ratings lie in the ratings of peers. Furthermore, these models achieve a high accuracy when predicting whether a user will simply like or dislike a movie. Nonetheless, high computation times and needed processing power limit these methodologies. We showed that a simple model using strategic parameter estimation achieves similar accuracy with much lower computation times. Additionally, these parameter estimation methods create priors that give new information regarding the quality of a movie, and rating tendencies of a user.

For future directions, we would want to extend to the multinomial setting for estimating priors. This calls for a generalization of the procedure described in [8]. Furthermore, given that Netflix now collects considerably more data on users, we would like to enrich the dataset with covariates about movies and users (browsing behavior, completion rate, completion time, etc). This might open up the possibility of training deep neural networks over dense features. Finally, to capture changes in preference over time, a many-to-many recurrent neural network (and its variants LSTM, GRU, etc.) on the enriched dataset that takes in a sequence of movies, users and the resulting ratings might produce better results.

7. Team Contributions

The team has worked together on this project following a set schedule. Both members have identified potential strategies in reducing MSE of test estimates. Victor has researched the modern day strategies that could be applied, and implemented the algorithm described in [8], experimented with different thresholds for hyperparameter searches. Evan has worked on the implementation of the KNN baseline, its variants and its analysis. Both have worked additionally on the initial data analysis, and further research.

References

[1] N. N. S. S. Altman and N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[2] R. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize.

[3] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[5] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, Jan. 2003.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[8] K. Tian, W. Kong, and G. Valiant. Optimally learning populations of parameters. 09 2017.