Algorithms for Collaborative Prediction

by
Todd Lipcon
Advisor: Philip N. Klein

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2007

Advisor: Philip N. Klein

Abstract

Collaborative filtering is a framework for providing recommendations of items in an e-commerce system tailored to a specific user based on information gleaned from past behavior of both that user and other users of the system. In this thesis, we focus on the problem of predicting the rating out a 5-star scale that a given user will assign to a new item based on all prior ratings in the system. Specifically, we investigate rating prediction performance on the Netflix Prize data set and the MovieLens-1M data set, both in terms of mean squared error and mean absolute error. We first introduce some improvements to neighborhood-based methods, and then introduce a new model based on gradient boosting machines. We then construct ensembles of gradient boosting machines by bootstrap aggregation, and produce results that improve upon the state-of-the-art on the MovieLens data set. We then introduce a new method for combining several distinct algorithms by means of support vector machines and support vector ordinal regression, and achieve results that improve upon the state of the art by an even larger margin.

Acknowledgements

I would like to thank my advisor, Professor Philip Klein, first for encouraging me to take on this project as an honors thesis, second for putting up with me sleeping through several meetings, and third for the many insights and suggestions throughout the meetings I didn't sleep through. I would also like to thank my reader Greg Shakhnarovich both for his help on this thesis and for teaching CS195-5, in which I learned the majority of what I know of machine learning. For spawning my interest in machine learning in the first place, I'd like to thank Andy Hyatt and Daryl Pregibon at Google. Lastly, I'd like to thank Benjamin Meyer and Brandyn Webb for their comments on early drafts of this work, as well as Allison Grubbs for acting as the proverbial rubber duck on top of my monitor [25].

On a less personal level, I'd like to extend thanks to the GroupLens research group at the University of Minnesota for making public the MovieLens data set, and also to Netflix for releasing the Netflix Prize dataset and offering a $1 million prize which really got me started on this project.

Contents

List of Tables **v**

List of Figures **vi**

1 Introduction **1**

1.1 Background 1

1.2 Machine Learning Fundamentals 1

1.2.1 Learning Regression 2

1.2.2 Generalization Performance and Regularization 2

1.2.3 Bias and Variance 2

1.3 Scoring Metrics 3

1.4 Notation 3

1.5 Data Sets, Statistics and Preparation 4

1.5.1 Rating Distributions 4

1.5.2 User and Movie Statistics 4

1.5.3 Rating Normalization 6

2 Naive Methods **8**

2.1 Movie Averages 8

2.2 Normalized Averages 9

2.2.1 Results 9

3 Movie-based Pearson Correlation KNN Method **10**

3.1 Prediction Model and Preprocessing Step 11

3.2 Prediction Step 12

3.3 Computational Complexity 13

3.4 Experimental Results 13

3.5 Data Mining from Neighbor Models 14

3.6 User-based Pearson Correlation 14

4 Linear Factor Models **15**
4.1 Singular Value Decomposition Model . . . 15
4.2 Webb's Factorization Algorithm . . . 17
4.2.1 Baseline Choices . . . 18
4.2.2 Correspondence to the true SVD . . . 18
4.2.3 Regularization . . . 18
4.2.4 Probabilistic Formulation . . . 19
4.2.5 Probabilistic Regularization . . . 19
4.3 Greedy Aggregation . . . 21
4.4 Relation to Trace Norm Regularization . . . 21
4.5 Improved Rank-1 Approximation Formulation . . . 21
4.6 Improved Regularization . . . 22
4.7 Choice of Regularization Parameters . . . 22
4.8 Stochastic Meta-Descent . . . 23
4.9 UVLearn Algorithm Summary . . . 23
4.10 Local Minima in the UVLearn Algorithm . . . 24

5 Gradient Boosting Machine Factorization **26**
5.1 Gradient Descent in Function Space . . . 28
5.2 Gradient Boosting as Convex Optimization . . . 28
5.3 Shrinkage Paramter . . . 28
5.4 GBMF Algorithm Summary . . . 30
5.5 Results . . . 30
5.6 Alternative Loss Functions . . . 31
5.7 Bagged GBMF . . . 31
5.7.1 Results . . . 32
5.7.2 Fast Parallel Learning with Bagged GBMF . . . 32

6 Combining Predictors **34**
6.1 Linear Least Squares Combination . . . 36
6.2 Support Vector Regression . . . 36

7 Conclusions and Future Work **38**
7.1 Contributions of this Thesis . . . 38
7.2 Future Work . . . 39
7.2.1 Alternate Loss Functions . . . 39
7.2.2 Faster UVLearn Optimization . . . 39
7.2.3 Iterated Bagging . . . 39
7.2.4 Stochastic Gradient Boosting . . . 39

Bibliography **41**

List of Tables

1.1 Summary of datasets used in this paper 4

2.1 Results for predicting future ratings as the simple priorized movie average 8
2.2 Naive predictor results 9

3.1 Optimal parameters for KNN algorithm determined by grid search 13
3.2 Best results for KNN algorithm 13
3.3 The 10 movies most similar to *Lord of the Rings: The Fellowship of the Ring* 14
3.4 The 10 movies most similar to *Finding Nemo (Full-Screen)* 14

5.1 Optimal parameters for GBMF determined by grid search 31
5.2 Best results for GBMF 31
5.3 Best parameters found for Bagged GBMF 32
5.4 Best results for Bagged GBMF 32

6.1 Best results for Least Squares Combination 36
6.2 Best results for Least Squares Combination with Interaction Terms 36
6.3 Best results for combination based on support vector machines 37

List of Figures

1.1 Histograms for the rating distributions of MovieLens (left) and Netflix (right) 5
1.2 Box plots of user rating distribution parameters for MovieLens and Netflix. 6
1.3 Skewness and kurtosis for artificial rating distributions drawn from our discretized and clamped Gaussian model 7

5.1 Training and probe error during training of the GBMF model. 29
5.2 Probe error during training of the gradient boosting machine model at different values of ν but with the same value of λ. 29

Chapter 1

Introduction

1.1 Background

In recent years, the popularity of e-commerce has grown tremendously. With this growth, customers are now able to shop online for practically any item, and purchase this item from a store anywhere in the world. As this mode of shopping as grown, online vendors increasingly know little about their customers but still want to personalize their shopping experiences. Thus, vendors have sought to mine customer's purchase and rating information in order to provide product recommendations catering to a user's individual tastes. The software systems that do this are called *recommendation systems* and have been employed with particular success in commercial situations where hard-to-specify personal tastes determine which items the user is most likely to purchase. For example, Amazon.com uses a recommendation system for book purchases, and Netflix for movie rentals.

As many vendors do not have any information on their users aside from some ratings or past purchases, most recommendation system algorithms are described as methods of *collaborative filtering.* This simply means that recommendations for one user are functions of data from other users. For example, users may be grouped by similar tastes, or products may be grouped by similar attributes, but the tastes and attributes are inferred rather than explicitly surveyed.

The *CineMatch* recommendation system developed by Netflix has been lucrative enough for their DVD rental business that in September, 2006, the company made available an anonymized subset of their rating database and offered a $1 million prize to any team who could beat their system by a margin of 10% on a held-out test set [4].

1.2 Machine Learning Fundamentals

As the audience of this thesis may not be familiar with some standard machine learning theory, we first briefly review some core concepts and terms.

1.2.1 Learning Regression

The general problem of regression is to learn a function $f(x_i)$ from some training set pairs $\{x_i, y_i\}$ such that some loss function $L(f(x_i), y_i)$ is minimized over the true distribution of the input variable. In most regression contexts, one makes the assumption that the examples x_i in the training set are independent and identically distributed (i.i.d) and that they are drawn from the same distribution as the examples which the learned function f will be required to predict.

The loss function itself varies depending on the specific problem. In this thesis, we concentrate on squared loss $L(f(x_i), y_i) = (f(x) - y_i)^2$ and absolute loss $L(f(x_i), y_i) = |f(x) - y_i|$.

1.2.2 Generalization Performance and Regularization

In learning a regression function $f(x)$, the ultimate goal is usually not to simply compress the training set X into a functional form. Rather, the goal is to learn a function that will make accurate predictions for new data points that were not in the training set. The ability of a regression function to make predictions on new data is its *generalization performance*. If a regression function exhibits low error on examples in its test set, but high error on new examples, it is said to have *overfit* or have been *overtrained*.

To combat the overfitting problem, *regularization* is introduced on the model. Regularization is the general term for any technique that reduces overfitting by introducing constraints or penalties on the complexity of the learned model. For example, in the case of linear regression (learning weights w to fit a function of the form $f(x) = w^T x$) regularization is often applied by penalizing either $\sum |w_i|$ (lasso regression) or $\sum w_i^2$ (ridge regression).

1.2.3 Bias and Variance

After the function $f(x)$ has been learned on a training set X, it will have some amount of loss $L(X)$ on that training set, as well as an expected loss $E[L]$ on new values. It can be shown that the expected loss is the sum of three terms: *noise*, *bias* and *variance*. The noise in the data is a random process that we cannot predict, even with perfect knowledge and the best possible algorithm. The bias is the structural error of the learning model that exists when the model is not flexible enough to completely fit the data. The variance, then, measures the amount that the model will change depending on the exact training set X provided to the learning algorithm. If the algorithm is unregularized, one can expect that the variance will be high, since the model will always change to predict the training data well. If the algorithm is over-regularized, the variance will be low, since the exact selection of examples for the training set should not significantly alter the predictions. Regularization, then, manages a trade-off between bias and variance; with an appropriate amount of regularization, the sum of the two terms will reach a minimum and the expected generalization accuracy will be optimal.

1.3 Scoring Metrics

In this paper, the focus of our algorithms is rating prediction. In this type of collaborative filtering, the algorithm is trained on triplets of the form {user, movie, rating} and then required to predict the missing rating for {user, movie} pairs in a held-out test set.

The scoring benchmark for the Netflix contest is root mean-squared-error (RMSE) of predictions. In this paper, RMSE values reported will be on the "probe" set as provided in the contest. As in [37], we find that our results on the Netflix "qualifying" set are generally 0.0050 better than our results on the probe set since we can include the probe set as additional training data.

The scoring benchmark reported more widely in the literature is NMAE (normalized mean absolute error), equal to MAE divided by a factor of 1.6, the expected absolute error assuming uniform rating and prediction distributions [32]. We therefore report both RMSE and NMAE scores for our algorithms. When reporting absolute error, we always implicitly round predictions to the nearest integer before calculating the error.

For all algorithms in this paper, we find and correct for bias terms by running the algorithm on the test set, and finding a constant additive term that minimizes training error. For the squared error metric, the correction term is simply the mean of the residuals from the uncorrected predictor. For the absolute error metric, the correction is the median of the residuals.

This paper seeks to apply several existing methods, as well as develop some novel approaches, showing experimental results both on Netflix and the well-known MovieLens-1M set. We also introduce a method of combining different prediction algorithms that attains lower error than any of the algorithms alone. Throughout, both RMSE and NMAE will be used as methods for comparing results.

1.4 Notation

Let M be defined as the set of movies in the data set, each assigned a unique numeric ID indexed beginning at 1. Similarly let U be the set of users.

Let $R \in \Re^{|U| \times |M|}$ be defined as the rating data matrix for user and movie sets U and M respectively. The element R_{ij} is the rating that user i has assigned to movie j.

The training data (i.e. the subset of elements of R for which we have known values) is quite sparse. In the Netflix dataset, approximately 99% of the entries of R are unknown. We notate an unknown element as having the value $\emptyset$. This is in contrast to the usual usage of the term "sparse" for matrices, in which the majority of entries are simply 0. We formulate the collaborative filtering problem as one of matrix completion – we seek to "fill in" the missing values of R.

We also introduce notation for certain subsets of elements of R. We denote the set of known ratings by user i as $R_{u=i}$, and the set of known ratings for movie j by $R_{m=j}$.

We denote the set of movies rated by user u as M_u and the set of users that have rated movie m by U_m.

1.5 Data Sets, Statistics and Preparation

In this thesis, algorithms will be applied to two datasets: the Netflix Prize Competition dataset, and the MovieLens-1M dataset. The MovieLens data set has been released to the public for many years and is one standard for algorithm comparison in the collaborative filtering literature. Because the primary focus of this thesis is on the Netflix competition, we make a very slight modification to the MovieLens set and remove any movies that have been rated less than 5 times, thereby removing a total of 598 ratings (0.06% of the total).

In order to test generalization performance on MovieLens, we hold out a test set of 35,901 (approximately 3.6%) of the rating triplets. These ratings are selected in the same manner as the "probe" subset of Netflix data are selected: for each user, a number $c \sim \text{Binomial}(p = 0.66, n = 9)$ is chosen, and the c most recently dated ratings made by that user are moved to the test subset.

This test set construction is similar to Marlin's "weak generalization" experimental method [32] in which a single random rating is "held out" for each user. Our probe method is, however, a harder problem, since the expected hold-out per user is 6 ratings rather than 1. Because of this, we try our algorithms on both a probe-style test set as well as a "weak generalization" style test set. We list the "weak generalization" results as "MovieLens (weak)" and the Netflix-style test set as "MovieLens (probe)".

Due to time limitations, we do not calculate standard error for our scores across different train/test partitions, though Marlin notes that the MovieLens dataset is large enough that variance across different holdout sets is small. In his experimental results, the average ratio of standard error to mean accuracy is 0.6%.

A summary of the key statistics of the two datasets is shown in Table 1.1.

Dataset	**# Movies**	**# Users**	**Sparsity**	**# Training Examples**	**# Test Examples**
Netflix	17,770	480,189	98.84%	99,072,944	1,425,333
MovieLens (probe)	3,900	6,040	95.91%	963,710	35,901
MovieLens (weak)	3,900	5,000	95.8%	826,612	5,000

Table 1.1: Summary of datasets used in this paper

1.5.1 Rating Distributions

The rating data in both data sets is composed of integers in the range 1-5. We provide histograms of the rating distributions for each data set in Figure 1.1.

1.5.2 User and Movie Statistics

Many of the methods used on the dataset require some preprocessing of the ratings matrix R before they can function effectively. The preprocessing steps function to collect summary statistics across

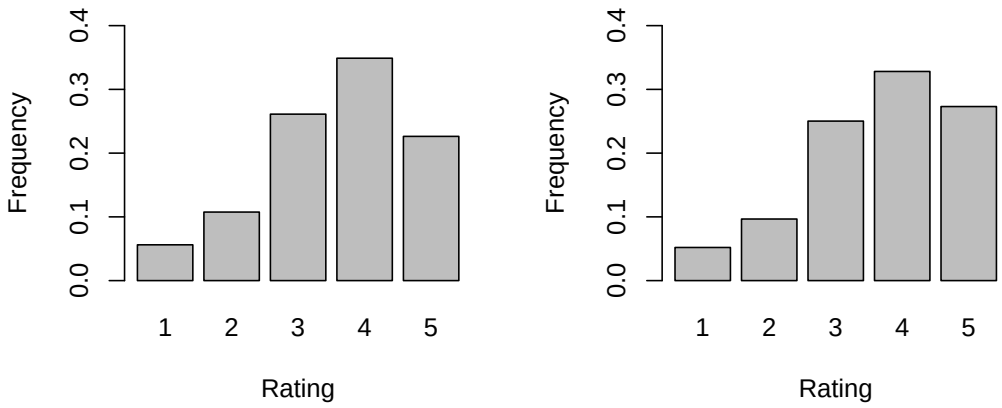

Figure 1.1: Histograms for the rating distributions of MovieLens (left) and Netflix (right)

rows and columns of the matrix. We also apply prepare normalized ratings to be used in some algorithms discussed later.

We first collect the following maximum likelihood estimates for parameters of Gaussian distributions describing ratings for each user i:

$$\hat{\mu}_i = \frac{1}{|R_{u=i}|} \sum_{r \in R_{u=i}} r \tag{1.1}$$

$$\hat{\sigma}_i = \sqrt{\frac{\sum_{r \in R_{u=i}} (r - \mu_{u=i})^2}{|R_{u=i}|}} \tag{1.2}$$

Because of the sparsity of the data, the maximum likelihood estimates for user parameters can have high variance when the number of known ratings for that user is small. We therefore also calculate priorized estimates $\hat{\mu}_i$, $\hat{\sigma}_i$ as follows:

$$\overline{\mu_u} = \frac{1}{|U|} \sum_{u \in U} \hat{\mu}_u \tag{1.3}$$

$$\overline{\sigma_u} = \frac{1}{|U|} \sum_{u \in U} \hat{\sigma}_u \tag{1.4}$$

$$\hat{\mu}'_i = \frac{|R_{u=i}|\mu_i + \lambda\overline{\mu_u}}{|R_{u=i}| + \lambda} \tag{1.5}$$

$$\hat{\sigma}'_i = \frac{|R_{u=i}|\sigma_i + \lambda\overline{\sigma_u}}{|R_{u=i}| + \lambda} \tag{1.6}$$

In our experiments, we choose $\lambda = 25$.

Following the same construction, we also calculate a priorized mean for each movie.

1.5.3 Rating Normalization

Normality Analysis

After calculating μ_i', σ_i' for each user i, we find it useful to make the simplification that each user's ratings are drawn from a normal distribution with those parameters. This assumption is clearly not perfect, since the ratings are chosen in a bounded interval $1 \leq r \leq 5$, whereas the normal distribution has infinite support. To investigate the validity of this assumption, we calculate the kurtosis and skewness of ratings made by all users with $|R_u| > 25$ for both the Netflix and MovieLens datasets. We provide box plots for these statistics in Figure 1.2.

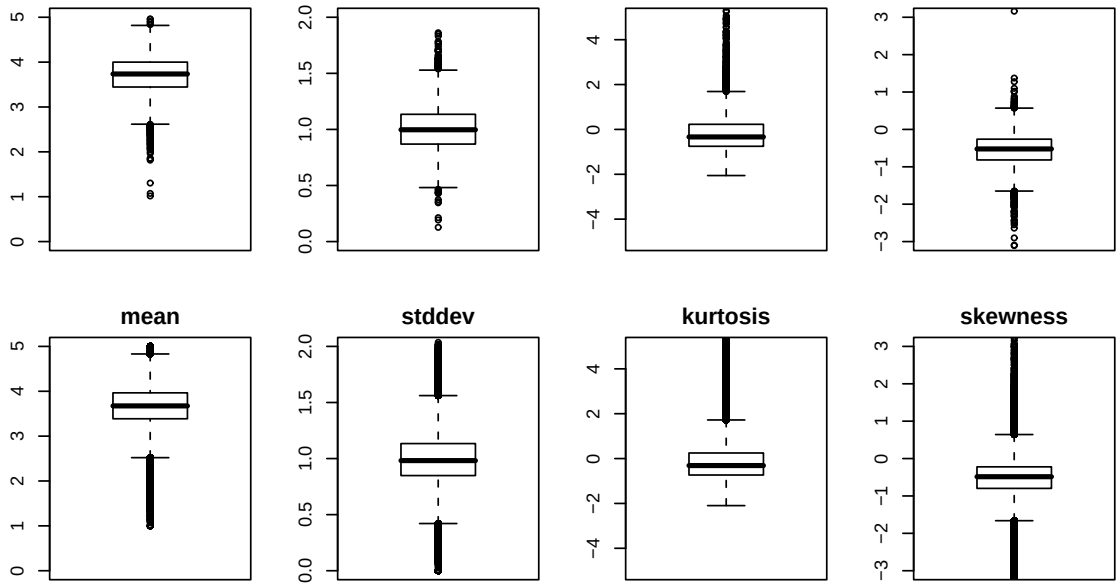

Figure 1.2: Box plots of user rating distribution parameters for MovieLens (top row) and Netflix (bottom row). Negative skewness indicates an elongated left tail. Kurtosis less than 0 indicates flatter topped distributions than the normal.

The analysis indicates that in both data sets, the average user's rating distribution is slightly less peaked (negative kurtosis) than the normal distribution. Additionally, it is skewed slightly left, indicating that users tend to rate movies with 3-5 stars more often than 1-2. This skew is clearly visible in the aggregate distribution of ratings as seen in the histograms in Figure 1.1.

These parameters, however, do not invalidate the analysis of ratings as if they were drawn from Gaussian distributions. Because the ratings are rounded to integers, kurtosis becomes negative, and because the mean rating is in the upper half of the clamping range, a left skew is inevitable. We show this by generating artifical rating distributions, sampling $r_i \sim \text{clamp}(\text{round}(\mathcal{N}(\mu = 3.6, \sigma = 1)))$ 100 times and calculating kurtosis and skewness. This experiment is repeated 1000 times and the distribution of results is shown in Figure 1.3. The distributions of skew and kurtosis observed in our

simulation experiment are quite similar to those seen in the real data, so we can reasonably conclude that our assumption well-founded.

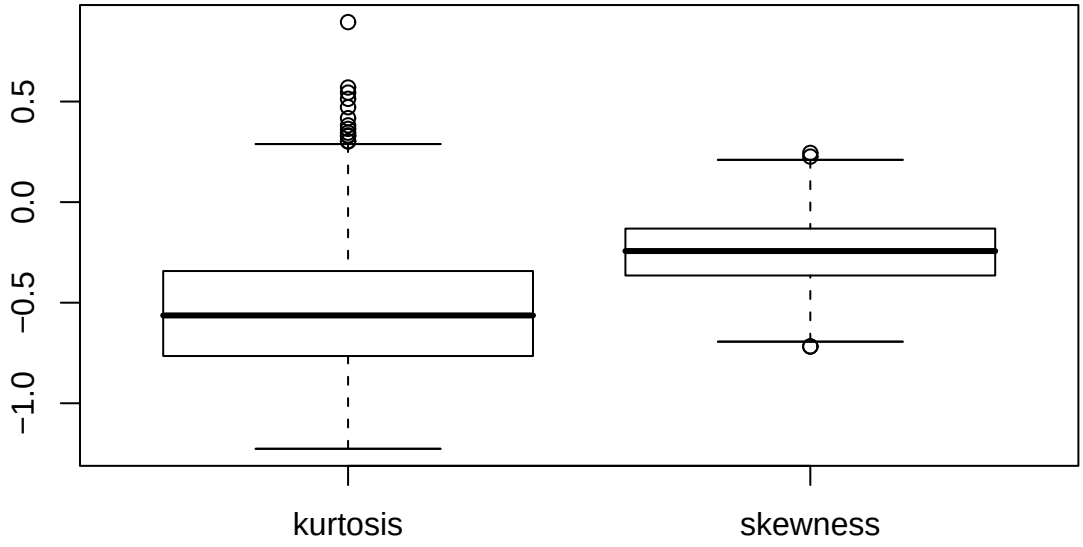

Figure 1.3: Skewness and kurtosis for artificial rating distributions drawn from our discretized and clamped Gaussian model

One area for future improvements on our methods is to assume that ratings are drawn from a distribution which can model the skew, kurtosis, and boundedness of the data set better. Two possibilities are the Beta distribution and the Kumaraswamy distribution [29].

Normalization

Having shown that the assumption of Gaussian distributions for ratings is somewhat applicable, we can normalize the ratings r made by each user by converting them to standard scores $\tilde{r} = \frac{r - \hat{\mu}'_u}{\hat{\sigma}'_u}$.

This normalization serves to make a set of ratings invariant with respect to scale and translation. Intuitively, some users may be generally more positive than others, or may tend to give ratings with a bigger spread. After normalization, positive numbers correspond to "liking" a movie, and negative numbers correspond to "disliking" a movie. A higher magnitude models a stronger feeling about the movie.

We note that all of the above normalizations and statistics can be done once during preprocessing and then kept up-to-date in an online environment as more data becomes available.

Chapter 2

Naive Methods

2.1 Movie Averages

The simplest method investigated is simply predicting future ratings to be equal to the average priorized rating for the movie, with no regard to the specific customer. Results are shown in Table 2.1.

It should be noted that the original dataset R consists of integers between 1 and 5 inclusive, and therefore we must clamp our prediction r to that range:

$$r_{\text{clamped}} = \begin{cases} 1 & \text{if } r \leq 1 \\ 5 & \text{if } r \geq 5 \\ r & \text{otherwise} \end{cases} \tag{2.1}$$

We do not, however, constrain our predicted ratings to be integral. The Netflix competition does not require this, and we note that in practical recommendation systems, the fractional parts of a prediction affect ranking order. Additionally, when the scoring metric is squared error, rounding predictions almost always increases error.

It should be assumed that all further predicted ratings r in this paper are implicitly subjected to the clamping described in Equation 2.1.

Dataset	**RMSE**	**NMAE**
Netflix	1.0516	0.5050
MovieLens (probe)	1.0065	0.4813
MovieLens (weak)	1.0030	0.4795

Table 2.1: Results for predicting future ratings as the simple priorized movie average

2.2 Normalized Averages

The next method investigated is a fairly simple approach in which predictions are made using only summary statistics for users and movies. Intuitively, this approach models a user's rating behavior as a single Gaussian distribution (with parameters μ_u and σ_u) and models a movie as an average of normalized ratings $\tilde{\mu}_m$. The assumption is that a movie is generally liked or disliked, and that that attitude can be mapped to a score using the user's rating distribution. Mathematically, a prediction for user i and movie j is made as:

$$\tilde{\mu}_j = E_{R_{m=j}}(\tilde{r}) \tag{2.2}$$

$$\tilde{r} = \tilde{\mu}_j \tag{2.3}$$

$$r = \hat{\mu}_u + \hat{\sigma}_u \tilde{r} \tag{2.4}$$

where E is the expectation operator.

Equation 2.4 is a generic unnormalization step that maps a standard score back to the original space of ratings.

2.2.1 Results

Our results for this method are shown in Table 2.2. Although prediction performance is fairly reasonable, and predictions are user-dependent, it is important to note that movie-ordering is not user-dependent. Because Equation 2.4 is a monotonically increasing function of $\tilde{r}$, the ordering of predictions is entirely determined by that value which is user-independent. Therefore the values $\tilde{\mu}_j$ determine a ranking order for every user in the system. Because most applications of collaborative filtering seek to provide a list of "top personal recommendations" to a user, this method does not achieve good results in practical usage.

Dataset	RMSE	NMAE
Netflix	0.9888	0.4615
MovieLens (probe)	0.9578	0.4503
MovieLens (weak)	0.9548	0.4454

Table 2.2: Naive predictor results

Chapter 3

Movie-based Pearson Correlation KNN Method

The next approach used is a k-nearest-neighbor method using Pearson correlation as a similarity metric. The sample Pearson correlation $\hat{\rho}$ measures the amount of linear dependence between two vectors X and Y, and is defined:

$$\hat{\rho}(X,Y) \equiv \frac{\sum_{i=0}^{N}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=0}^{N}(X_i-\bar{X})^2\sum_{i=0}^{N}(Y_i-\bar{Y})^2}} \tag{3.1}$$

If the two vectors are linearly dependent (i.e. $Y = aX + b$ for some a and b) then $|\hat{\rho}| = 1$. A positive value of $\hat{\rho}$ indicates a positive slope on a scatter plot of Y vs X, and a negative value indicates a negative slope. A value of $\hat{\rho} = 0$ indicates no correlation between the two vectors. It is important to note that, in all cases, $-1 \leq \hat{\rho} \leq 1$.

We apply Pearson correlation to the rating matrix R in order to determine similarity between two movies i, j. A high correlation indicates that there is a strong linear dependence between ratings on movie i and movie j. We interpret this as a similarity between the two movies. Recall that the set of users U_m is defined such that $\forall u \in U_m, R_{m,u} \neq \emptyset$. We then define the intersection set U_{ij} as $U_i \cap U_j$. This set includes all users who have made a rating for both movies i and j.

To calculate similarity between a given pair of movies i, j we first find the intersection set U_{ij}. If $|U_{ij}| \leq t$ for some t (we use $t = 3$), we determine that the two movies i, j cannot be similar since so few users have rated both. Assuming $|U_{ij} > t|$ we then calculate the Pearson correlation of the two vectors as:

$$\hat{\rho}_{i,j} = \frac{\sum_{u\in U_{ij}}(R_{i,u}-\mu_i)(R_{j,u}-\mu_j)}{\sqrt{\sum_{u\in U_{ij}}(R_{i,u}-\mu_i)^2\sum_{u\in U_{ij}}(R_{j,u}-\mu_j)^2}} \tag{3.2}$$

For this calculation, we take μ_i and μ_j only over the intersection set U_{ij} rather than over the entire training set.

The Pearson correlation $\hat{\rho}$ is only an estimate of the true population correlation between the two movies i, j based on a sample size $|U_{ij}|$. We note that a small sample size is unreliable for determining a correlation. For example, if there are 6 ratings in common between i and j with a correlation of 1.0, that does not necessarily mean that (i, j) have higher similarity than another pair (i, k) with $\hat{\rho} = 0.8$ and $|U_{ik}| = 3000$.

To counteract this effect, we calculate a confidence interval around the estimate as in [1]. It can be shown that, although the error distribution of this estimate $P(\rho = x|\hat{\rho} = y)$ is not Gaussian, it can transformed into a Gaussian distribution with standard error $\sigma_z = (|U_{ij}| - 3)^{-1/2}$ using Fisher's z'-transformation [16]:

$$z' = \operatorname{atanh}(\hat{\rho}) \tag{3.3}$$

We note that in this calculation, $z' = \pm\infty$ when $\hat{\rho} = \pm 1$. In order to simplify the calculations, we add a maximum correlation magnitude parameter τ and clamp $\hat{\rho}$ to the range $[-\tau, \tau]$ before computing Equation 3.3.

Because z' has an error distribution that is Gaussian, we can calculate a new value $\hat{\rho}'$ such that we can assert with some degree of confidence that the magnitude of the actual population correlation $|\rho| \geq |\hat{\rho}'|$.

$$\sigma_z = \frac{1}{\sqrt{|U_{ij}| - 3}} \tag{3.4}$$

$$z'_{\text{inner}} = \begin{cases} \min(z' + \varepsilon\sigma_z, 0) & \text{if } \hat{\rho} < 0 \\ \max(z' - \varepsilon\sigma_z, 0) & \text{if } \hat{\rho} > 0 \end{cases} \tag{3.5}$$

$$\hat{\rho}' = \tanh(z'_{\text{inner}}) \tag{3.6}$$

We could choose ε from a table of the normal distribution to achieve a specified confidence value. Not knowing a priori what confidence value will be optimal for generalization performance, we instead determine the best value of ε by grid search.

3.1 Prediction Model and Preprocessing Step

For each movie i, we calculate $\hat{\rho}'_{ij}$ for all $j \neq i$, and then choose the indices j with the N highest-magnitude correlations. We define N_m as the set of N nearest neighbors for movie m. In a binary file we store for each i the list of up to N neighbors along with their associated $\hat{\rho}'$ values.

Once we have established that movie i is a neighbor of movie j, we seek to be able to predict a rating on movie j given a rating on movie i. Although $\hat{\rho}$ models the tightness of a true linear dependence, we choose to follow the Slope One Predictors model of Lemire and Maclachlan [30] and

constrain the linear dependence to have a slope of 1. That is to say, for each neighbor i of j, we seek to learn the optimal intercept δ_{ij} in the equation:

$$r_j = r_i + \delta_{ij} \tag{3.7}$$

Lemire and Maclachlan introduce this constraint in order to reduce the number of free parameters in the model. This in turn decreases the variance of the predictor and improves generalization performance.

Our analysis of the data sets indicates that the assumption of positive slope is a fair one. In general, when ρ is a positive value, the slope of the best fit linear predictor will also be positive. In our data sets, although $\hat{\rho}$ is sometimes negative, we find that $\hat{\rho}'$ is almost always driven to 0 when $\hat{\rho} < 0$. We explain this by noting that, if there were a negative correlation between ratings of two movies, it is unlikely that many people would choose to rent and rate both of them. In most cases, users will rent movies they assume they will like, so it is rare that $|U_{ij}|$ would be large when $\rho < 0$.

In order to discover the optimal value δ_{ij} for Equation 3.7, we simply find the difference between the means of the two vectors:

$$\delta_{ij} = \frac{1}{|U_{ij}|} \sum_{u \in U_{ij}} (R_{j,u} - R_{i_u}) \tag{3.8}$$

We hypothesize that using the difference in medians of the two vectors may be more robust, but leave exploration of this idea for further research.

3.2 Prediction Step

In order to make a prediction using the neighborhood data gathered above, we make the assumption that a user will assign similar ratings to movies that have high rating correlation. That is to say, if user u has rated movie i with score r_i then he will assign movie j a rating $r_j = r_i + \delta_{ij} \pm \epsilon$ where the error term ϵ is smaller for larger ρ'_{ij}.

Because R is sparse, however, we cannot assume that a given user u has rated all of the movies in N_m when trying to predict u's rating on i. We therefore calculate an intersection set $N_{iu} \equiv N_i \cap M_u$ which consists of all neighbors of movie i that user u has rated in the past. As in standard nearest-neighbor methods, we choose a parameter k for the maximum number of "nearest neighbors" to include in the intersection set N_{iu}. We then calculate a weighted average of the ratings as predicted by the N neighbors as:

$$r_{ui} = \frac{\sum_{j \in N_{iu}} \hat{\rho}_{ij} \left(R_{ju} + \delta_{ij}\right)}{\sum_{j \in N_{iu}} \hat{\rho}_{ij}} \tag{3.9}$$

Because the set N_{iu} may be arbitrarily small, and even empty, the denominator of the above weighted average may be equal to 0. In order to combat this, we factor in the naive result as

described in Section 2.2. This can be included in Equation 3.9 by averaging in the naive prediction r_n with some weight w_n:

$$r_{ui} = \frac{\sum_{j \in N_{iu}} \hat{\rho}_{ij} \left(R_{ju} + \delta_{ij} \right) + w_n r_n}{\left(\sum_{j \in N_{iu}} \hat{\rho}_{ij} \right) + r_n} \tag{3.10}$$

We experimentally determine the optimal values for w_n and k jointly with the two neighborhood parameters ε and τ.

3.3 Computational Complexity

It can readily be seen that the choice of N does not significantly impact the runtime of the preprocessing step, since correlations are computed for all $|M|(|M|-1)$ pairs of movies, and N only affects the storage requirements. Practically, although the $O(n^2)$ nature of this step seems expensive, it runs on a single Athlon 3800+ in approximately 5 hours for the Netflix dataset. It is also embarassingly parallelizable, and could easily be run as a nightly batch job in a production setting for any size database.

3.4 Experimental Results

In terms of NMAE, our neighbor-based algorithm far outperforms the Pearson-based algorithm investigated by Marlin [32] as well as Lemire and Maclachlan's fairly similar "Slope One Predictors" algorithm [30]. Part of our improvement can be attributed to our rounding and clamping of predictions – Marlin does not do this [33] and it is unclear whether Lemire and Maclachlan do. However, our best unrounded NMAE is 0.4493 which still outperforms both of the other results.

Dataset	**RMSE**	**NMAE**
Netflix	$k = 15, \varepsilon = 4.8, \tau = 0.86, w_n = 1$	$k = 10, \varepsilon = 4.8, \tau = 0.86, w_n = 1$
MovieLens (probe)	$k = 30, \varepsilon = 2.4, \tau = 0.98, w_n = 0.75$	$k = 45, \varepsilon = 2.4, \tau = 0.9, w_n = 0.4$
MovieLens (weak)	$k = 20, \varepsilon = 2.4, \tau = 0.80, w_n = 0.5$	$k = 10, \varepsilon = 2.4, \tau = 0.8, w_n = 1$

Table 3.1: Optimal parameters for KNN algorithm determined by grid search

Dataset	**RMSE**	**NMAE**
Netflix	0.9411	0.4274
MovieLens (probe)	0.9166	0.4240
MovieLens (weak)	0.9126	0.4208

Table 3.2: Best results for KNN algorithm

3.5 Data Mining from Neighbor Models

Although KNN methods do not provide state-of-the-art prediction performance, the resulting neighborhood data is simple to mine and easy to interpret. For example, given a query movie, a list of the N most similar other movies can be returned in linear time. This has useful applications in e-commerce situations when providing recommendations for users who are browsing without a rating or purchase history, since an automatically generated list of "Similar Items" can encourage the user to continue shopping and increase conversion rates. Two examples of this type of query are shown in Tables 3.3 and 3.4.

$\hat{\rho}'$	Movie Title
0.809	Lord of the Rings: The Two Towers
0.755	Lord of the Rings: The Return of the King
0.687	The Lord of the Rings: The Fellowship of the Ring: Extended Edition
0.652	Lord of the Rings: The Two Towers: Extended Edition
0.623	Lord of the Rings: The Return of the King: Extended Edition
0.395	Lord of the Rings: The Fellowship of the Ring: Bonus Material
0.358	Lord of the Rings: The Two Towers: Bonus Material
0.308	Lord of the Rings: The Return of the King: Bonus Material
0.244	The Matrix
0.240	X-Men

Table 3.3: The 10 movies most similar to *Lord of the Rings: The Fellowship of the Ring*

$\hat{\rho}'$	Movie Title
0.650	Finding Nemo (Widescreen)
0.316	Monsters, Inc.
0.281	A Bug's Life
0.259	Toy Story
0.256	Toy Story 2
0.251	The Incredibles
0.243	Shrek 2
0.226	Ice Age
0.218	The Lion King: Special Edition
0.215	Harry Potter and the Prisoner of Azkaban

Table 3.4: The 10 movies most similar to *Finding Nemo (Full-Screen)*

3.6 User-based Pearson Correlation

The above algorithm for finding nearest neighbors for each movie and then learning predictors from those neighbors can be modified to find neighbors for each user. Given the $O(n^2)$ nature of the data preparation, the user-based correlation method is far less efficient, since $|U| \gg |M|$ in most applications of collaborative filtering. Additionally, the user-based correlation algorithm has been found to be less accurate than the item-based (i.e. movie-based) method [2].

Chapter 4

Linear Factor Models

As noted in Section 1.4, the matrix R is sparse, with most entries $R_{ij} = \emptyset$. The problem of collaborative prediction can be seen as a matrix completion problem – for each missing element in R, we seek to find the value that has highest probability under some model. One such class of models that has been explored in the literature is the linear factor model. This model hypothesizes that there is a vector $w_u \in \Re^k$ for each user, modeling that user's preferences for movies with different traits, and a vector $w_m \in \Re^k$ for each movie, modeling the strength with which that movie has those traits. For example, factors may capture the amount of violence in a movie, or the strength with which a movie belongs to a certain genre like comedy. A rating of movie m by user u is then simply determined by the inner product $w_u \cdot w_m$. The length k of the vectors is a model parameter that must be selected with regard to the usual tradeoff between bias and variance. A high value of k can lead to overfitting, whereas a low value of k does not adequately capture the variance of the training data.

In matrix form, the above model can simply be written:

$$R = UV^T$$

where $U \in \Re^{|M| \times k}$ and $V \in \Re^{|U| \times k}$ and M^T denotes the matrix transpose.

This model includes such approaches as: Hofmann's pLSA [23], in which the matrix U is constrained to be a stochastic matrix; Weighted Low Rank Approximation [10], in which the number of columns of U and V is bounded; and Canny's "factor analysis" method [11]. We introduce a new method, which we call Gradient Boosting Matrix Factorization (GBMF) that is related to these approaches but significantly improves upon their performance.

4.1 Singular Value Decomposition Model

In order to introduce GBMF, we first review the truncated singular value decomposition [15, 22]. Generally, the singular value decomposition of a matrix M is formulated:

$$M = U\text{diag}(\sigma)V^T \tag{4.1}$$

where $M \in \Re^{m \times n}$, orthonormal $U \in \Re^{m \times r}$, $\sigma \in \Re^r$, orthonormal $V \in \Re^{n \times r}$, and $r = \text{rank}(M)$. The rows of U are the left singular vectors of M. The elements of σ are the singular values of M, listed in descending order by convention. The rows of V are the right singular vectors of M.

The *truncated* singular value decomposition is a low-rank matrix approximation such that the vector σ is truncated to a number of elements $k < r$, and U and V are truncated to their first k columns. After these truncations, the reconstructed matrix M' is an approximation of M that has rank k. In fact, it is well known that M' is the *optimal* rank-k approximation of M with respect to the Frobenius norm of the residual matrix $M' - M$. That is to say:

$$M' = \underset{\{M' : \text{rank}(M') = r\}}{\text{argmin}} ||M - M'||_{Fro} \tag{4.2}$$

where the Frobenius norm of a matrix $A \in \Re^{m \times n}$ is equal to $\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2}$

By collapsing the diagonal matrix of singular values into either U or V, we see that the SVD formulation is indeed a linear factor model with k factors as described above. Indeed, it is the linear factor model that has the smallest squared error.

Given that our ultimate goal is the recovery of missing elements of R with minimal squared error, the result that the truncated SVD provides the optimal rank-k approximation is enticing. For this reason, many researchers have applied the SVD to such fields as noise suppression [27], dimensionality reduction (PCA) [26] and soft clustering [14] with success.

The major drawback to the SVD is that it only applies to matrices that are fully filled-in. The most common methods for its fast computation (e.g. Lanczos methods [20]) cannot be easily modified to compute an SVD given only a small number of observations from the full matrix. We therefore must formulate a different strategy to deal with the missing elements. Some researchers have applied the SVD to collaborative filtering problems after substituting either 0 or the movie average μ_m for the missing values $\emptyset$ in the matrix [3, 19]. The drawback to these methods is that, as we increase the rank of our approximation towards $\min\{m, n\}$, we lose all predictive power from the decomposition – we simply recover the same values we substituted.

Another solution for the computation of the SVD of a matrix with missing values is to do so column-by-column while maintaining statistics for a Gaussian that models the distribution of column vectors [6]. Imputation of the missing elements is done by maximum likelihood estimation; however, this method requires computing the Moore-Penrose pseudo-inverse of a dense and large matrix for every added column. This method therefore is computationally infeasible for large datasets such as ours. Additionally, this method is not invariant with respect to column permutation, and performs significantly worse than our algorithm on the MovieLens set.

4.2 Webb's Factorization Algorithm

Recently, Webb introduced a collaborative filtering model [43] based on an incremental SVD technique using the Generalized Hebbian Algorithm [21]. The method aims to solve for the linear factor model UV^T that minimizes the sum of squared error across only the training set elements. The computation ignores the missing elements completely. Webb's objective is equivalent to the "weighted low-rank approximation" problem [10]:

$$\min_{\{U \in \Re^{m \times r}, V \in \Re^{n \times r}, r \ll \min\{m,n\}\}} ||W \odot \left(UV^T - M\right)||_{\text{Fro}} \tag{4.3}$$

where W is a binary weight matrix and $\odot$ is the element-wise multiplication operator.

We reproduce Webb's algorithm here with the intent of describing it with greater mathematical rigor than was given in its initial publication, and later in this thesis propose improvements to this method that translate to faster training speed and higher prediction accuracy.

Webb optimizes Equation 4.3 by greedily finding best-fit rank-1 approximations of the form uv^T and subtracting them from the data matrix M to find a new data matrix M'. That is to say, for each step of the algorithm, it solves:

$$\{u, v\} = \underset{\{u \in \Re^{|U|}, v \in \Re^{|M|}\}}{\text{argmin}} \sum_{\{i,j\} \in R_{\text{observed}}} (u_i v_j - R_{ij})^2$$

Each u, v pair is solved for using stochastic gradient descent. In each training epoch (i.e. sweep over the training data) the error gradient is calculated:

$$\begin{aligned}
E_{ij} &= (u_i v_j - R_{ij})^2 && (4.4)\\
\frac{\partial E_{ij}}{\partial u_i} &= 2u_i v_j^2 - 2v_j R_{ij} && (4.5)\\
&= 2v_j(u_i v_j - R_{ij}) && (4.6)\\
\frac{\partial E_{ij}}{\partial v_j} &= 2v_j u_i^2 - 2u_i R_{ij} && (4.7)\\
&= 2u_i(u_i v_j - R_{ij}) && (4.8)\\
\text{Let } \epsilon &= u_i v_j - R_{ij} && (4.9)\\
\nabla E_{ij} &= < 2v_j\epsilon, 2u_i\epsilon > && (4.10)
\end{aligned}$$

We construct update equations for u_i and v_j at each iteration with learning rate η absorbing the constant factor:

$$\begin{aligned}
u_i' &:= u_i - \eta v_j \epsilon && (4.11)\\
v_j' &:= v_j - \eta u_i \epsilon && (4.12)
\end{aligned}$$

The algorithm iteratively makes sweeps through the training data to sequentially minimize the error objective until the improvement in objective value between two consecutive epochs is smaller

than some threshold value which we call τ. When the u and v vectors stabilize (usually after 100-200 epochs), they are stored as the next rows of U and V respectively. The result uv^T is subtracted from the rating matrix R and the algorithm repeats to calculate the next most significant pair.

4.2.1 Baseline Choices

In his original proposal of this method, Webb initializes the predictions at the start of the algorithm based on priorized estimates of movie means and each customer's average offset from those means. We propose that this is unnecessary and adds unnecessary hyperparameters (e.g. regularization tradeoffs or thresholding values for mean estimates as seen in Section 1.5.2).

In our investigation of this method, we experimented with two different choices for baseline predictions:

- **One-baseline** - we set the initial prediction for each rating to be the constant value 1.
- **Average-baseline** - we set the initial prediction for each rating to be the priorized mean rating for that movie

The advantage to the "one-baseline" approach is that no regularization parameters affect the baseline predictions. We find that the "one-baseline" method exhibits slightly better generalization performance than the average-baseline method across a wider range of regularization parameters, and therefore concentrate on this approach in the following sections.

4.2.2 Correspondence to the true SVD

It can be shown that, in the special case where all of the entries in the matrix R are observed, Webb's technique does give a result identical to the true SVD as found by standard techniques. In the case of a partially observed matrix, however, the factorized solution UV^T is not truely an SVD – there is no constraint that the matrices U and V be orthogonal. Therefore, Webb's method is not truely a "sparse SVD," but rather another example of a linear factor model.

4.2.3 Regularization

Webb notes that the u and v vectors must be regularized in some manner to prevent overfitting phenomena for customers or movies with low amounts of data. For example, if a user has given a movie a score of 5, but that movie's v_j value has very low magnitude (e.g. .01), u_j will be learned to be extremely high in order to lower the training error on the test set. Webb suggests the following alteration to the update assignments:

$$u_i' := u_i - \eta(v_j\epsilon + Ku_i) \tag{4.13}$$

$$v_j' := v_j - \eta(u_i\epsilon + Kv_j) \tag{4.14}$$

where K is a small constant (he uses 0.02). Webb shows empirically that this regularization reduces the effect of overfitting and makes the expected tradeoff of higher training error for lower test error. He hypothesizes that it is related to Tikhonov Regularization (ridge regression), which applies penalties to model probability that are directly proportional to the ℓ^2 norm of the model weights.

4.2.4 Probabilistic Formulation

We add rigor to Webb's algorithm by deriving the same formulation under a probabilistic framework and showing that it is in fact a maximum likelihood estimator under the assumption of Gaussian noise. We start with that the rating made by user i on movie j is sampled from a Gaussian distribution with mean $u_i v_j$ and some standard deviation σ. From this we derive the log likelihood of the training data:

$$p(R|u,v) = \prod_{ij} p(R_{ij}|u_i,v_j) \tag{4.15}$$

$$\log p(R|u,v) = \sum_{ij} \log p(R_{ij}|u_i,v_j) \tag{4.16}$$

$$\text{Let } p(R_{ij}|u_i,v_j) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(R_{ij}-u_i v_j)^2}{2\sigma^2}\right) \tag{4.17}$$

$$\log p(R_{ij}|u_i,v_j) = -\frac{(R_{ij}-u_i v_j)^2}{2\sigma^2} - \log(\sigma\sqrt{2\pi}) \tag{4.18}$$

$$\log p(R|u,v) = -\frac{1}{2\sigma^2}\sum_{ij}(R_{ij}-u_i v_j)^2 - N\log(\sigma\sqrt{2\pi}) \tag{4.19}$$

We then seek to assign u and v to be the maximum likelihood estimates for the observed training data. Because the log function is monotonically increasing, optimizing the log likelihood in turn optimizes the likelihood. We therefore derive:

$$\{u,v\} = \underset{\{u,v\}}{\text{argmax}}\ p(R|u,v) \tag{4.20}$$

$$= \underset{\{u,v\}}{\text{argmax}}\left[-\frac{1}{2\sigma^2}\sum_{ij}(R_{ij}-u_i v_j)^2 - N\log(\sigma\sqrt{2\pi})\right] \tag{4.21}$$

$$= \underset{\{u,v\}}{\text{argmin}}\ \sum_{ij}(R_{ij}-u_i v_j)^2 \tag{4.22}$$

4.2.5 Probabilistic Regularization

Now that the model has probabilistic underpinnings, we can regularize it by applying *maximum a posteriori* (MAP) estimates rather than maximum-likelihood. The MAP estimate models some prior belief we hold about the probability distribution of u and v. For example, we may assume

that u and v are sampled from multivariate Gaussian distributions with covariance matrices of the form σI. Under this assumption, we can confirm Webb's hypothesis that his regularization tradeoff corresponds to Tikhonov Regularization. We start by penalizing the log-likelihood of the data by adding the log-likelihood of the model:

$$\begin{aligned}
\log p(R|u,v)p(u)p(v) &= \sum_{ij} \log p(R_{ij}|u_i,v_j) + \log p(u) + \log p(v) && (4.23)\\
\log p(u) &\propto -\sum_{i=1}^{|U|} (u_i - \mu_u)^2 && (4.24)\\
\log p(v) &\propto -\sum_{j=1}^{|M|} (v_j - \mu_v)^2 && (4.25)\\
\{u,v\} &= \underset{\{u,v\}}{\operatorname{argmin}} \left[\sum_{ij} (R_{ij} - u_i v_j)^2 + \lambda_u \sum_{i=1}^{|U|} (u_i - \mu_u)^2 \right. && (4.26)\\
&\qquad \left. + \lambda_v \sum_{j=1}^{|M|} (v_j - \mu_v)^2 \right]
\end{aligned}$$

After the algorithm has learned one $\{u,v\}$ pair, there should be no constant bias term in the residuals, so we further assume that $\mu_u = \mu_v = 0$.

$$\{u,v\} = \underset{\{u,v\}}{\operatorname{argmin}} \sum_{ij} \epsilon^2 + \lambda_u ||u||^2 + \lambda_v ||v||^2 \qquad (4.27)$$

where $||\cdot||_2$ signifies the ℓ^2 norm. This penalty on the ℓ^2 norms of u and v is by definition Tikhonov Regularization on those vectors.

From this result we rederive the error gradient as above:

$$\begin{aligned}
E_{ij} &= (u_i v_j - R_{ij})^2 + \lambda_u u_i^2 + \lambda_v v_j^2 && (4.28)\\
\frac{\partial E_{ij}}{\partial u_i} &= 2v_j\epsilon + 2\lambda_u u_i && (4.29)\\
\frac{\partial E_{ij}}{\partial v_j} &= 2u_i\epsilon + 2\lambda_v v_j && (4.30)\\
& && (4.31)
\end{aligned}$$

and new update equations:

$$\begin{aligned}
u_i' &:= u_i - \eta(v_j\epsilon + \lambda_u u_i) && (4.32)\\
v_j' &:= v_j - \eta(u_i\epsilon + \lambda_v v_j) && (4.33)\\
& && (4.34)
\end{aligned}$$

4.3 Greedy Aggregation

After each $\{u, v\}$ pair has been learned, it is aggregated into the linear factor model by simply appending columns $U' = [U\ u]$ and $V' = [V\ v]$. A model prediction is then made by simply taking the inner product of the customer's row in U with the movie's row in V:

$$R_{ij} = b_{ij} + U_i \cdot V_j = bij + \sum_{s=1}^{r} U_{is} V_{js} \tag{4.35}$$

where b_{ij} is the baseline prediction (in our case equal to 1).

Equation 4.35 presents a slight modification from Webb's algorithm proposition in which predictions are made with nonlinear clamping such that the sum is held within the valid output prediction range after each term:

$$R_{ij} = \text{Clamp}\left(\text{Clamp}\left(\text{Clamp}\left(b + U_{i1}V_{j1}\right) + U_{i2}V_{j2}\right) + ...\right) \tag{4.36}$$

We find that this clamping provides slight improvements in Webb's model, though it confuses the analysis and slows down prediction performance. We also find that, in our modifications of the algorithm, the benefits of clamping are made insignificant.

4.4 Relation to Trace Norm Regularization

An alternative interpretation of the penalization on the ℓ^2 norms of each $\{u, v\}$ pair is the imposition of regularization on the trace norm of the rating matrix. The trace norm of a matrix X is defined the sum of the singular values of X, and can equivalently be expressed as either: [36]

$$\min_{\{U,V\}:U'V=X} |U|_{Fro}|V|_{Fro}$$

or

$$\min_{\{U,V\}:U'V=X} \frac{1}{2}\left(|U|^2 + |V|^2\right)$$

Additionally, Rennie shows that the second quantity is minimized when $|U| = |V|$ and both U and V are orthogonal matrices.

Because of these facts, we can see that Webb's regularization method penalizes an upper bound on the trace norm of the prediction matrix. As trace norm can be seen as a measure of the complexity of a matrix, as well as a convex surrogate for rank, this regularization method should limit the amount of overfitting of our model and increase generalization accuracy.

4.5 Improved Rank-1 Approximation Formulation

We start with Webb's rank-1 base learner and make improvements that increase generalization performance while retaining and sometimes improving on the efficiency of the original formulation.

4.6 Improved Regularization

First, we investigate our previous assumption that $\bar{u} = \bar{v} = 0$ (Equation 4.27). We note that, in the first feature trained, our choice of a constant baseline means that the mean of the residuals will be nonzero. Therefore, a high regularization factor under Webb's framework will prevent the residual bias from being learned away. We formulate an improved regularization factor that makes the assumption that we do have enough data for most users and movies to find appropriate u and v values that do not overfit. We then use these "appropriate" values as an "empirical prior", and constrain that all u and v values should fall close to the average. With this regularization, our new objective becomes:

$$\{u, v\} = \underset{\{u \in \Re^{|U|}, v \in \Re^{|M|}\}}{\operatorname{argmin}} \sum_{\{i,j\} \in R_{\text{observed}}} (u_i v_j - R_{ij})^2 + \lambda_u \sum_{i=0}^{|U|} (u_i - \bar{u})^2 + \lambda_v \sum_{i=0}^{|M|} (v_i - \bar{v})^2 \tag{4.37}$$

noting that $\bar{u}$ and $\bar{v}$ are simultaneously optimized along with u and v. Taking the gradient of this new objective for a single training example yields:

$$\frac{\partial E_{ij}}{\partial u_i} = 2v_j(u_i v_j - R_{ij}) + 2\lambda_u(u_i - \bar{u}) \tag{4.38}$$

$$\frac{\partial E_{ij}}{\partial v_j} = 2u_i(u_i v_j - R_{ij}) + 2\lambda_v(v_j - \bar{v}) \tag{4.39}$$

$$\frac{\partial E_{ij}}{\partial \bar{u}} = -2\lambda_u(u_i - \bar{u}) \tag{4.40}$$

$$\frac{\partial E_{ij}}{\partial \bar{v}} = -2\lambda_v(v_j - \bar{v}) \tag{4.41}$$

4.7 Choice of Regularization Parameters

In our regularization formulation, we have kept λ_u and λ_v, the regularization parameters on movie features and user features respectively, separate from each other. We hypothesized that we may want to constrain users more than movies since $|R_u|$ tends to be smaller on average than $|R_m|$. In our experiments, we performed grid search across values of λ_u and λ_v and found that the best generalization result was no different than the best result for constraining $\lambda_u = \lambda_v$. We assume this is because, for any pair uv^T, we can multiply u by some constant k and divide v by that same constant and achieve the same result. Therefore the lowest regularization penalty $||u||^2 + ||v||^2$ is achieved when $||u|| = ||v||$. We therefore follow Webb in simply using a single regularization parameter λ. In our experiments, we choose this parameter by running the algorithm for many different values of λ and picking the one that exhibits the best generalization performance.

4.8 Stochastic Meta-Descent

In addition to improving the regularization terms in Webb's method, we investigate improvements to the optimization method by using Stochastic Meta-Descent [39] to increase convergence rate as well as robustness against local optima. SMD requires calculation of the Hessian matrix of second derivatives of the error function:

$$H\left(E(u_i, v_j, \bar{u}, \bar{v})\right) = \begin{bmatrix} 2v_j^2 + 2\lambda_u & 4u_i v_j - 2R_{ij} & -2\lambda_u & 0 \\ 4u_i v_j - 2R_{ij} & 2u_i^2 + 2\lambda_v & 0 & -2\lambda_v \\ -2\lambda_u & 0 & 2\lambda_u & 0 \\ 0 & -2\lambda_v & 0 & 2\lambda_v \end{bmatrix}$$

We then maintain a local learning rate η as well as a *gradient trace* ν for each parameter, with updates governed by the SMD equations:

$$\eta' = \eta \max(0.9, 1 + \mu \nabla E_{ij} \odot W) \tag{4.42}$$

$$W' = W - \eta \odot \nabla E_{ij} \tag{4.43}$$

$$\nu' = \lambda v + \eta \left(\nabla E_{ij} - \lambda H \nu\right) \tag{4.44}$$

where $W =< u_i, v_j, \bar{u}, \bar{v} >$ and η, ν index specific SMD parameters for those weights.

In our experimentation, however, the algorithm exhibited issues with divergence. This is a known problem with stochastic meta-descent when μ and λ are chosen improperly. In our algorithm, however, the optimal parameters $\{\mu, \lambda\}$ change as the greedy algorithm progresses, since the norm of the matrix to be approximated is steadily reduced. Therefore, a setting of μ that is appropriate for the first round of gradient descent will usually cause divergence later on in the algorithm.

Given these factors, we revert to using the standard stochastic gradient descent method and leave continued exploration of SMD optimization of the rank-one approximation objective for further work.

4.9 UVLearn Algorithm Summary

We now summarize the rank-1 approximation learning algorithm, which we dub UVLearn:

Input:

- Rating matrix $R \in \Re^{m \times n}$ with missing values
- Regularization parameter λ
- Early stopping parameter τ
- Learning rate η

Output:

- Vectors $u \in \Re^m, v \in \Re^n$ such that uv^T is a good approximation of R in the Frobenius norm sense, and such that $\lambda(||u||^2 + ||v||^2)$ is small.

UVLearn(R, λ, τ, η)

$\eta \leftarrow 0.01$
$\nu \leftarrow 0$
$\bar{R} \leftarrow \frac{1}{N} \sum_{ij \in R} R_{ij}$
$\bar{u} \leftarrow \sqrt{\bar{R}}$
$\bar{v} \leftarrow \bar{R}/\bar{u}$
$\{u, v\} \leftarrow$ UniformDistribution(0 ± 0.01)
last_objective $\leftarrow \sum_{ij \in R} (R_{ij} - u_i v_j)^2 + \lambda \left(\sum_{i=1}^m u_i^2 + \sum_{j=1}^n v_n^2 \right)$
$\delta \leftarrow 0$
while $\delta > \tau$ or epoch $<$ `min-epochs`
do for $j \in$ RandomPermutation(M)
    do for $i \in U_j$
        do Update weights using stochastic gradient descent given training example R_{ij}
    objective $\leftarrow \sum_{ij \in R} (R_{ij} - u_i v_j)^2 + \lambda \left(\sum_{i=1}^m u_i^2 + \sum_{j=1}^n v_n^2 \right)$
    $\delta \leftarrow 0.8\delta + 0.2($ last_objective $-$ objective $)$
return $\{u, v\}$

4.10 Local Minima in the UVLearn Algorithm

Because the UVLearn algorithm is optimized by stochastic gradient descent, and because the objective function is not convex, there is no guarantee that it will converge to a solution that is a global optimum. However, our experience indicates that, despite not being convex, the surface of the objective function either has many local optima that have the same objective value, or the likelihood of falling into a bad local optimum is extremely low.

In order to show this, we first generate a random matrix where the known entries are placed in the same locations as the MovieLens data set. The values of the entries are replaced with random values sampled from a uniform distribution on the interval from -2.5 and 2.5. We then run the UVLearn algorithm on this matrix with the parameters $\eta = 0.0001, \lambda = 0.03, \tau = 0$. We let the

algorithm run for a maximum of 5000 training epochs, and record the value of the objective function after training completes. This experiment is repeated 10 times. The difference between the best and worst objective values reached is 0.03%.

The same experiment is also run 25 times for the unaltered MovieLens (probe) data set. The difference between the worst objective value achieved and the best objective value achieved was 0.007%.

From these results, it appears that local minima do not significantly affect the optimization surface in a mathematical sense. In practice, though, the very small value of η used in this experiment requires too many iterations of gradient descent to be useful in real systems. The actual objective values achieved for more reasonable values of η do tend to vary more, but the gain in optimization speed is worth the slight loss in convergence to a minimum. Additionally, the seeming lack of local minima for the small-η experiment indicates that work on improved optimization methods for this same objective should be fruitful.

Chapter 5

Gradient Boosting Machine Factorization

Further investigation of Webb's factorization method reveals that it is in fact a slightly modified version of the Gradient Boosting Machine described by Friedman [17]. A Gradient Boosting Machine is an greedy additive regression model with properties similar to classification boosting algorithms such as the well-known AdaBoost [38].

We reproduce the Gradient Boosting Machine algorithm here in order to be self-contained:

Input:

- A training set X made up of N individual examples (x_i, y_i)
- A base learner $h(x; a)$ for which the parameters a can be learned to minimize squared loss between $h(x_i)$ and some values $\tilde{y}_i$.
- A convex differentiable loss function $\Psi(y, F(x))$ to be minimized
- Shrinkage parameter $0 < \nu \leq 1$
- A depth M which controls how many times to boost the model.

Output:

- $F(x) = F_0(x) + \sum_{m=1}^{M} \nu \rho_m h(x; a_m)$

GBMLearn$(h(x), \Psi(y, F), X, \nu, M)$

1 $F_0(x) \leftarrow \underset{p}{\text{argmin}} \sum_{i=1}^{N} \Psi(y_i, \rho)$
2 **for** $m \leftarrow 1$ **to** M
3 **do for** $i \leftarrow 1$ **to** N
4 **do** $\tilde{y}_i \leftarrow -\left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$
5 $a_m \leftarrow \underset{a, \beta}{\text{argmin}} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(x_i; a)]^2$
6 $\rho_m \leftarrow \underset{\rho}{\text{argmin}} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(x_i) + \rho h(x_i; a_m))$
7 $F_m(x) \leftarrow F_{m-1}(x) + \nu \rho_m h(x; a_m)$

We make the following modifications to the framework:

- In line 1, we explore different functions for the baseline F_0 rather than always setting it to the constant which minimizes the loss function Ψ.
- When minimizing squared error, we remove line 6, since it has the effect of undoing the regularization on the base learner. Instead, we just set $\rho_m = 1$.

In our gradient boosting machine collaborative filtering model, we use the UVLearn algorithm as the "base learner" $h(x; a)$. Boosting has been shown to function best with base learners that have high bias but low variance; for example, in classification contexts, the usual base learner is the single-variable decision stump. Because the rank-1 approximation of a mostly-missing matrix has high variance, and boosting does not reduce variance, we expect best performance from regularized base learners. The resulting increase in bias is reduced by the boosting framework, yielding superior performance of the aggregate learner compared to simply calculating a smaller number of $\{u, v\}$ pairs without regularization.

5.1 Gradient Descent in Function Space

An important insight gained from Friedman's framework is that gradient boosting machines perform a sort of gradient descent in function space [34]. Each iteration of boosting finds a base learner which moves the aggregate learner in the direction of the negative gradient as determined by any loss function that is a convex function of the training error.

5.2 Gradient Boosting as Convex Optimization

Another way to look at the gradient boosting framework is as a convex optimization method. The objective of this problem is the convex loss on our training examples, and the set of permissible solutions is the set of all possible linear combinations of base learners $h(x; a)$. At each iteration of gradient boosting, we make a call to an oracle method to find a direction in which to move the current solution. We then move our current solution an incremental amount in that direction such that the convex loss is reduced. Assuming that parameters a for the function $h(x; a)$ can always be learned such that $h(x_i; a) \cdot y_i > 0$, the solution X will take steps in the direction of the negative gradient. Because the objective is convex, the algorithm will always converge to 0 training error in the limit.

5.3 Shrinkage Paramter

Paralleling the idea of a learning rate in standard gradient descent optimization, gradient boosting employs a *shrinkage* parameter ν which determines the amount each additional base learner should be incorporated into the aggregate. This is particularly important for our learning machine, as it has been shown that greedy aggregation of rank-1 approximations does not yield the optimal rank-k approximation when the matrix is not fully observed [41].

In experiments with other base learners (both pruned and unpruned CART [9]), Friedman shows that shrinkage in the range $\nu < 0.1$ can prevent overfitting and improve generalization performance of the model. We show similar results for our algorithm in Figure 5.1. Lower values for ν decrease the value of λ required to achieve optimal performance. Additionally, lower values of ν decrease the error of the best boosted predictor. In Figure 5.2 we show the generalization performance of three predictors with the same value of λ, but with different values of the shrinkage parameter ν. Although it is evident that smaller values of ν achieve better results, the tradeoff is one of training speed, as the optimal depth M increases with smaller ν, and the increase in prediction accuracy is a case of diminishing returns.

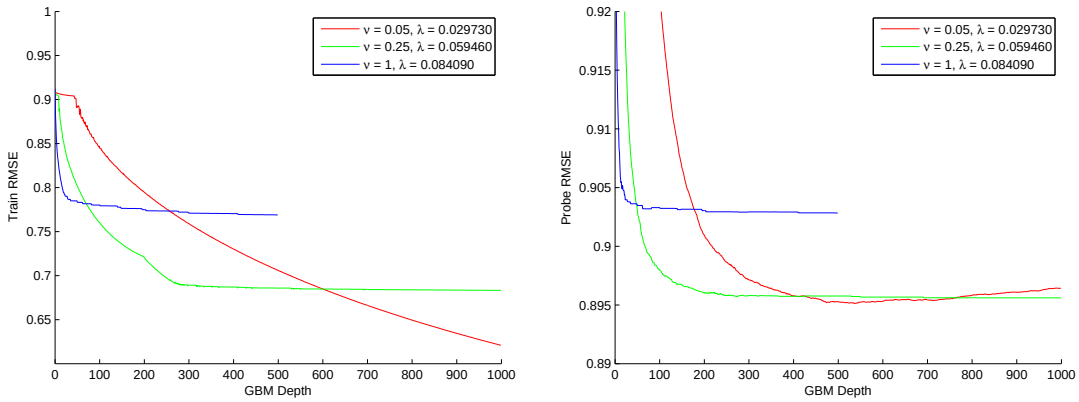

Figure 5.1: Training and probe error during training of the GBMF model. The GBM depth refers to the number of iterations taken by the loop at line 2 in the GBMLearn algorithm. Each curve represents the best choice of λ for that value of ν. Data shown is for the MovieLens (probe) dataset.

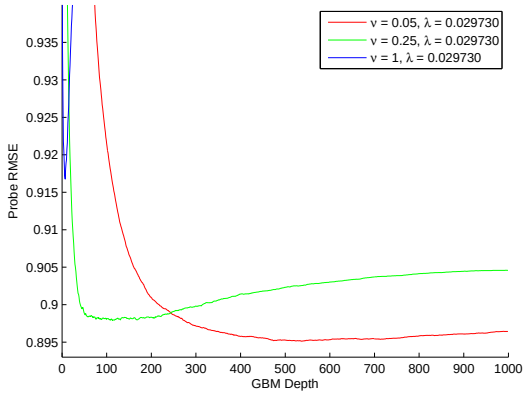

Figure 5.2: Probe error during training of the gradient boosting machine model at different values of ν but with the same value of λ. The models with small ν overfit later in training and achieve lower error.

5.4 GBMF Algorithm Summary

We now summarize the GBMF algorithm:

Input:

- Rating matrix $R \in \Re^{m \times n}$ with missing values
- Regularization parameter λ
- Gradient descent early stopping parameter τ
- Gradient descent learning rate η
- Shrinkage parameter ν
- Number of iterations M

Output:

- Matrices U and V such that UV^T is a good approximation of R in the Frobenius norm sense, and such that $\lambda(||U||^2 + ||V||^2)$ is small.

GBMFLearn$(R, \lambda, \tau, \eta, \nu, M)$

```
1  R_c ← Baseline(R)
2  U ← [], V ← []
3  for m ← 1 to M
4  do ΔR ← R − R_c
5      {u, v} ← UVLearn(ΔR, λ, τ, η)
6      U ← [U νu], V ← [V νv]
7      R_c ← R_c + νuv^T
8  return {U, V}
```

5.5 Results

We tabulate our best results for the GBMF model in Table 5.2 and the associated training parameters in Table 5.1. It is important to note that the optimal value of λ is highly dependent on the gradient descent stopping threshold τ. When we set $\tau = 0$, a higher degree of regularization λ on the norms of u and v is needed in the base learner. When we set $\tau > 0$, it encourages "early stopping" of the gradient descent, which acts as an alternative form of regularization.

We assume that the best possible results would be for $\tau = 0$, but this setting significantly increases the number of iterations which each subroutine call to UVLearn requires. For example, the optimal value on the MovieLens (probe) set with $\tau = 0$ is achieved after a total of 407,583 UVLearn iterations, whereas the optimum value on MovieLens (weak) with $\tau = 0.00001$ is achieved after a total of 62,277 iterations.

Dataset	RMSE				NMAE			
	ν	λ	τ	M	ν	λ	τ	M
Netflix	0.1	0.000433	0.00001	400	0.1	0.000433	0.0001	400
MovieLens (probe)	0.05	0.029730	0	540	0.05	0.29730	0	1000
MovieLens (weak)	0.05	0.007433	0.00001	586	0.05	0.007433	0.00001	596

Table 5.1: Optimal parameters for GBMF determined by grid search. The parameter τ affects runtime and the value of the other parameters, but was simply chosen for reasonable experimental run time.

Dataset	RMSE	NMAE
Netflix	0.9139	0.4099
MovieLens (probe)	0.8933	0.4100
MovieLens (weak)	0.8859	0.4051

Table 5.2: Best results for GBMF

5.6 Alternative Loss Functions

Because gradient boosting performs gradient descent in function space, any convex differentiable loss function can be substituted for squared loss. Because this thesis uses absolute error as one of its scoring metrics, we investigate the absolute error loss function and its derivative:

$$\Psi(F(x), y) = |F(x) - y| \tag{5.1}$$

$$\frac{\partial \Psi(F(x), y)}{\partial F(x)} = \begin{cases} 1 & \text{if } F(x) > y \\ -1 & \text{if } F(x) < y \end{cases} \tag{5.2}$$

Suprisingly, optimizing using the absolute loss objective yielded solutions with higher absolute loss than optimizing with the squared error metric. Additionally, we found that several other loss functions, including the smooth hinge loss, all-thresholds logistic loss [35], and Huber's robust M-estimator [24] all yielded poor results. We are still investigating why these other loss functions do not improve the model.

5.7 Bagged GBMF

One of the main problems in designing an algorithm for collaborative filtering is that there is so little data for many of the users. If a user only has a few entries in the training set, it is very easy to overfit the model for that user. Regularizing too strongly then underfits the users who have a lot of entries in the training set. We therefore would like to find a method of increasing generalization performance (reducing predictor variance) that does not cause an associated increase in bias on users for whom we can make good predictions.

One variance-reduction method that fits this criterion is bootstrap aggregation, also known as *bagging* [7]. Bagging is an ensemble learning method that trains multiple parallel replicas of

the learning algorithm, where the training set given to each replica is a bootstrap sample of the full set. If a training set X has N elements, a bootstrap sample is taken by simply choosing N elements at random from X with replacement. Therefore the expectation is that approximately $\lim_{N\to\infty}(1-1/N)^N \approx 37\%$ of the training examples will be missing from each bootstrap training set, and other randomly selected examples will be weighted multiple times. Each member of the ensemble is then fully trained, and the resulting predictions are averaged together using the mean when optimizing squared error and the median when optimizing absolute error.

Taniguchi and Tresp [42] show that bagging works best when the variance of each member of the ensemble is high, but not when the members are completely unregularized. We therefore train our ensemble members with lower values of λ and higher values of GBM shrinkage ν compared to our single predictors. Due to the computationally intensive nature of bagging, we were unable to do a full grid search for these parameters; we show our best hand-picked choices in Table 5.3.

It has also been shown that the benefits of bagging begin to level off as the size of the ensemble increases. We choose an ensemble size of 60 predictors, which our experiments show to be around the point at which adding additional ensemble members give diminishing returns.

In our bagging experiments, we use simple averaging when making predictions to optimize squared error, and we use the median of the member predictors when optimizing absolute error.

5.7.1 Results

Our best results for bagging where the ensemble members are made of a single type of GBM are shown in Table 5.4.

Dataset	ν	λ	τ	η	M
Netflix	0.25	3×10^{-6}	3×10^{-6}	0.003	450
MovieLens (probe)	0.25	1×10^{-6}	1×10^{-5}	0.003	800
MovieLens (weak)	0.25	1×10^{-6}	1×10^{-5}	0.003	800

Table 5.3: Best parameters for Bagged GBMF determined by experimentation. In all cases, ensembles were made of 60 predictors.

Dataset	**RMSE**	**NMAE**
Netflix	0.9121	0.4086
MovieLens (probe)	0.8916	
MovieLens (weak)	0.8850	0.4045

Table 5.4: Best results for Bagged GBMF

5.7.2 Fast Parallel Learning with Bagged GBMF

Aside from advantages in terms of generalization accuracy, bagged GBMF models have the ability to learn extremely quickly when parallel processing capability is available. Training 60 predictors

to just a depth of 10 GBM iterations on the MovieLens (weak) dataset yields an NMAE of 0.4183 and can be achieved in about a minute on a cluster of 2.80Ghz Xeons.

Chapter 6

Combining Predictors

Given the various formulations available for the collaborative filtering problem, we posture that each algorithm has strengths and weaknesses for different sorts of data that may appear within a single set. For example, one algorithm may have performance advantages on new users with little rating history, and another may work better on niche movies that have only been rated a few times. Indeed, even within a single algorithm, different values for tuning parameters may be optimal on different subsets of the data. We also assume that, because of our simplifying assumptions of Gaussian noise and error homoscedasticity, our predictions can be further improved by a second layer of training.

We hypothesize that the effectiveness and structural error of a predictor $\hat{r}$ is a function of the following variables:

- μ_u - the user's average assigned rating
- μ_m - the movie's average assigned rating
- $|U_m|$ - the number of users who have rated the movie
- $|M_u|$ - the number of movies the user has rated

For a given unknown rating, we craft a vector s composed of the above statistics. We then augment that vector with the predictions made by any number of algorithms as described above. For example, we may include two KNN predictions with different tuning parameters, as well as a few predictions made by GBMF models at various depths of training and regularization levels.

We denote this created statistics-prediction vector x and then then seek to learn a function $f : x \rightarrow \Re$ that makes predictions on unknown ratings. Because we can fashion the x vector for any known or unknkown rating, this becomes a simple regression problem for which there are a multitude of well known techniques.

In order to learn this combination function f, we create a training set X made up of ratings that the individual learners have not been trained on. The ratings that make up X must not be training data for the component learners, or else components that have overtrained will be assigned

too much weight in the combination predictor. In our experiments, we use the held-out test datasets as training sets for f, and then use the standard technique of k-fold cross-validation [28] to verify the performance of our combination algorithm.

For the MovieLens (probe) we use the following predictors:

- GBMF with movie-average baseline and Webb's regularization, $\nu = 0.05, \lambda = 0.001, M = 250$
- GBMF with movie-average baseline and Webb's regularization, $\nu = 0.05, \lambda = 0.005657, M = 140$
- GBMF with one baseline and our regularization, $\nu = 0.05, \lambda = 0.008839, M = 200$
- GBMF with one baseline and our regularization, $\nu = 0.05, \lambda = 0.029730, M = 720$
- Bagged GBMF with one baseline and our regularization, $\nu = 0.25, \lambda = 0.005, M = 500$
- Bagged GBMF with one baseline and our regularization, $\nu = 1, \lambda = 0.03, M = 450$
- KNN predictor with $\varepsilon = 2.4, \tau = 0.98, k = 30, w_n = 0.75$
- KNN predictor with $\varepsilon = 4.8, \tau = 0.80, k = 10, w_n = 0.1$

These predictors were selected to encompass a wide variety of approaches. For each GBMF method, we selected the predictor that gave the best generalized RMSE for customers with more than 25 ratings, and the predictor that generalized best for customers with less than 25 ratings. In all cases these two predictors were different, which supports our hypothesis that a combination of predictors will be better. As a baseline, we simply average the 8 component predictors and achieve an RMSE of 0.8898 and NMAE of 0.4147. For reference, we calculate that, if for every data point we could always select the component with the closest prediction, we would achieve an RMSE of 0.7192.

For the MovieLens (weak) set we try the even simpler combination:

- Bagged GBMF with $\lambda = 3 \times 10^{-5}, \nu = 0.25, M = 800$
- GBMF $\nu = 0.05, \lambda = 0.025, M = 10$

By choosing a deep model and a shallow model we can strike a good balance between bias and variance by weighting between them. The shallow model on its own has an RMSE of 1.85, but we will see that the combination algorithm can extract useful data.

Unfortunately, due to time constraints and computation costs, we have not yet investigated the combination algorithm's performance on the Netflix data set, though there is no reason to believe we cannot extract similar improvements.

6.1 Linear Least Squares Combination

The simplest solution for f is to let f be a standard linear equation:

$$f(x) = w^T x + b$$

where we optimize w and b to minimize the squared error. This problem can be solved quite efficiently and requires no regularization since we assume that the number of examples in our training set X is far larger than the length of the feature vectors x.

We learn the least squares model and test it using 10 repititions of 10-fold cross-validation on the MovieLens (probe) set and 10 repetitions of 100-fold cross-validation on the MovieLens (weak) data set due to its smaller size. We then also learn a least squares model with variable interaction terms and show that error is reduced further. Results are tabulated in Tables 6.1 and 6.2. The NMAE results from even this simplistic combination method compete with the state-of-the-art Ensemble MMMF [13] technique.

Dataset	**RMSE**	**NMAE**
MovieLens (probe)	0.8842	0.4055
MovieLens (weak)	0.8817	0.4062

Table 6.1: Best results for Least Squares Combination

Dataset	**RMSE**	**NMAE**
MovieLens (probe)	0.8824	0.4039
MovieLens (weak)	0.8758	0.4043

Table 6.2: Best results for Least Squares Combination with Interaction Terms

6.2 Support Vector Regression

Because we are able to get such good improvements from even a simple linear combination of predictors, it makes sense to also consider nonlinear regression techniques. One popular technique in the machine learning literature is the Support Vector Regression [40]. Specifically, we use the nu-SVR [5] variant of this technique as implemented in LibSVM [12] for squared-error minimization. For absolute error minimization, we use a form of ordinal support vector regression that uses a reduction to a classification problem [31]. In both cases we use radial basis function kernels and use grid search with k-fold cross-validation to select tuning parameters ν, C, γ in the case of SVR and C, γ in the case of ordinal regression. For the absolute loss case, we achieve a significant improvement by using support vector ordinal regression rather than the simple rounded least squares solution.

We present our results in Table 6.3.

Dataset	**RMSE**	**NMAE**
MovieLens (probe)	0.8788	0.3984
MovieLens (weak)	0.8810	0.4003*

Table 6.3: Best results for combination based on support vector machines. * indicates that training is still being undertaken for this value as of 5/1

.

Chapter 7

Conclusions and Future Work

7.1 Contributions of this Thesis

In this thesis, we have contributed several things to the field of collaborative filtering. First, we examined the newly released Netflix Prize dataset in comparison to the well-known MovieLens dataset with regard to several important statistical measures. Next, we provided evidence that supports the oft-made assumption that ratings can be treated as drawn from Gaussian distributions.

In Chapter 3, we introduced a method of rating prediction based on movie neighorhoods determined by a confidence-interval adjusted Pearson Correlation. We improve upon the best known results for correlation-based algorithms by providing smooth fallback to a naive prediction method as well as adding a parameter τ which controls the maximum amount of unadjusted correlation that two movies can have with each other.

In Chapter 4, we investigated Webb's "SVD-like" matrix factorization method and proved that each $\{u, v\}$ pair that is solved for is in fact the maximum likelihood estimate under a Gaussian noise model. We additionally have shown that Webb's regularization term corresponds to Tikhonov Regularization on the vectors u and v. We then suggested several improvements to Webb's feature learner, including improved regularization and the possibility of using Stochastic Meta-Descent to encourage fast convergence to global optima. We have shown that, in a simulation study, and on the MovieLens data set, our resulting UVLearn base learner does not exhibit characteristics of local optima.

In Chapter 5, we related Webb's factorization technique to Friedman's Gradient Boosting Machine framework. We utilized this insight to improve the algorithm further. We added a shrinkage parameter ν which provides additional regularization and increases generalization performance. We also explored the possibility of alternative loss functions including absolute loss and robust loss techniques such as Huber's M-estimation, though initial exploration of these possibilities has proven unsuccessful. We name the improved algorithm Gradient Boosting Matrix Factorization (GBMF).

In Section 5.7, we improved further upon the results of the GBMF model by constructing ensembles of predictors trained on bootstrap samples of the training set, following Breiman's *bagging* framework. We showed that these ensemble methods can learn extremely quickly in parallel learning situations and that accuracy is increased from singular predictors.

In Chapter 6, we introduced a method of combining various distinct prediction algorithms. We employed ν-SVR to optimize squared error and employed a reduction method for ordinal regression to optimize absolute error. After feeding the algorithms discussed earlier in this work into the combination learning algorithm, we achieved results that far surpass the current state of the art on the MovieLens data set.

7.2 Future Work

7.2.1 Alternate Loss Functions

The GBMF method can optimize training error for any function of the training errors which is convex and differentiable. We expect that some more robust loss function may increase generalization performance by decreasing the influence of outliers on the learned model.

7.2.2 Faster UVLearn Optimization

Although we experimented with Stochastic Gradient Descent for the UVLearn subroutine, we were unable to prevent divergence from occurring. We believe that there may be a way to set the SMD parameters μ and λ specifically for each iteration of GBMF such that the subroutine does not diverge, either by annealing the SMD parameters on some decreasing schedule, or by adaptively setting it based on some measurements of the current pseudo-residual matrix ΔM.

Alternatively, since the optimization objective of UVLearn seems to have few local minima, we hypothesize that some optimization methods other than gradient descent may be able to optimize it equally well but with faster convergence.

7.2.3 Iterated Bagging

Although the Bagged GBMF model is a hybrid of boosting and bagging, we do not combine the two processes in a unified framework. Breiman has introduced a model called Iterated Bagging [8] with promising experimental results. We experimented briefly with this method but were unable to achieve results better than Bagged GBMF, despite its better theoretical underpinnings.

7.2.4 Stochastic Gradient Boosting

Another method which deserves further investigation is Friedman's Stochastic Gradient Boosting [18] in which each subroutine call to UVLearn would be passed a fractional sample of the training set. Friedman shows strong experimental results as well as an increase in training speed associated with

having to train only on a fraction of the training set at each boosting iteration. Like iterated bagging, we experimented with this method but were unable to extract improvements.

Bibliography

[1] Kamal Ali and Wijnand van Stam. Tivo: making show recommendations using a distributed collaborative filtering architecture. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 394–401, New York, NY, USA, 2004. ACM Press.

[2] Joseph Konstan Badrul Sarwar, George Karypis and John Reidl. Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA, 2001. ACM Press.

[3] Joseph Konstan Badrul Sarwar, George Karypis and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference on Computer and Information Technology (ICCIT)*, 2002.

[4] James Bennett and Stan Lanning. The Netflix Prize. In *Proceedings of KDD Cup and Workshop*, 2007. To appear.

[5] Robert C. Williamson Bernhard Schölkopf, Alex J. Smola and Peter L. Bartlett. New support vector algorithms. *Neural Computations*, 12(5):1207–1245, 2000.

[6] M. Brand. Fast online SVD revisions for lightweight recommender systems. In *Proceedings of the SIAM International Conference on Data Mining (SDM'03)*, 2003.

[7] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[8] Leo Breiman. Using iterated bagging to debias regressions. *Machine Learning*, 45(3):261–277, 2001.

[9] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, January 1984.

[10] Nicoletta Del Buono and Tiziano Politi. A continuous weighted low-rank approximation for collaborative filtering problems. In *ICAPR (1)*, pages 45–53, 2005.

[11] John Canny. Collaborative filtering with privacy via factor analysis. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 238–245, New York, NY, USA, 2002. ACM Press.

[12] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. Technical report, Computer Science and Information Engineering, National Taiwan University, 2001-2004. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[13] Dennis DeCoste. Collaborative prediction using ensembles of Maximum Margin Matrix Factorizations. In *ICML '06: Proceedings of the 23rd International Conference on Machine Learning*, pages 249–256, New York, NY, USA, 2006. ACM Press.

[14] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Journal of Machine Learning*, 56(1-3):9–33, 2004.

[15] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[16] R. A. Fisher. Frequency distribution of the values of the correlation coefficient in samples of an indefinitely large population. *Biometrika*, 10:507–521, 1915.

[17] J. Friedman. Greedy function approximation: a gradient boosting machine. Technical report, Dept of Statistics, Stanford University, 1999.

[18] J. Friedman. Stochastic gradient boosting. Technical report, Dept of Statistics, Stanford University, 1999.

[19] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[20] Gene H. Golub, Franklin T. Luk, and Michael L. Overton. A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix. *ACM Transactions on Mathematical Software*, 7(2):149–169, 1981.

[21] Genevieve Gorrell. Generalized Hebbian Algorithm for incremental singular value decomposition in natural language processing. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics: EACL 2006*, 2006.

[22] Per C Hansen. The truncated SVD as a method for regularization. Technical report, Stanford University, Stanford, CA, USA, 1986.

[23] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):89–115, 2004.

[24] P.J. Huber. *Robust Statistics*. Wiley, New York, 1981.

[25] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, 1999.

[26] I. T. Jolliffe. *Principal Component Analysis*. Springer, 1986.

[27] J. Kamm. *Singular value decomposition-based methods for signal and image restoration*. PhD thesis, Southern Methodist University, Dallas, TX, 1998.

[28] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, pages 1137–1145, 1995.

[29] P. Kumaraswamy. A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, 46(1/2):79–88, March 1980.

[30] D. Lemire and A. Maclachlan. Slope One predictors for online rating-based collaborative filtering. In *Proceedings of the SIAM International Conference on Data Mining (SDM'05)*, 2005.

[31] Ling Li and Hsuan-Tien Lin. Ordinal regression by extended binary classification. In *Advances in Neural Information Processing Systems 19*. MITPress, 2007. To appear.

[32] Benjamin Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, 2004.

[33] Benjamin Marlin, 2007. Personal correspondence.

[34] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent in function space. Technical report, RSISE, Australian National University, 1999.

[35] Jason Rennie and Nathan Srebro. Loss functions for preference levels: Regression with discrete ordered labels. In *IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling*, July 2005.

[36] Jason D. M. Rennie. Equivalent ways of expressing the trace norm of a matrix. http://people.csail.mit.edu/jrennie/writing/traceEquivalence.pdf, September 2005.

[37] Andriy Mnih Ruslan Salakhutdinov and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24nd International Conference on Machine Learning*, 2007. To appear.

[38] Robert E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[39] Nicol N. Schraudolph. Local gain adaptation in stochastic gradient descent. Technical Report IDSIA-09-99, ISDIA, 1999.

[40] A.J. Smola and B. Schoelkopf. A tutorial on support vector regression. Technical Report CS2-TR-1998-030, NeuroCOLT2, 1998.

[41] N. Srebro and T. Jaakkola. Weighted low rank approximation. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, 2003.

[42] Michiaki Taniguchi and Volker Tresp. Averaging regularized estimators. *Neural Computation*, 9(5):1163–1178, 1997.

[43] Brandyn Webb. Netflix update: Try this at home, 2006.
http://sifter.org/~simon/journal/20061211.html.