Collaborative Filtering Using Matrix Approximation

Jihoon Kim
UCSD Division of Biomedical Informatics
j5kim@ucsd.edu

Hari Damineni
UCSD Computer Science & Engineering
hdaminen@cs.ucsd.edu

André Christoffer Andersen
NTNU Industrial Economics
andrecan@stud.ntnu.no

May 17, 2011

Abstract

The objective of collaborative filtering is to predict the user ratings for an item based on the ratings of the same user and also on the ratings of other users. As the quantity of data grows, it becomes imperative for the collaborative filtering tasks to be agile and scalable. In this article, matrix approximation methods are used to build a collaborative filtering model. A database of item ratings by users is input to the model and output is the predicted ratings which are missing in the database. The article also presents an optimization technique using stochastic gradient descent which yields a low-complexity, high-throughput and a scalable model.

1 Introduction

Recommender systems are ubiquitous in today's online experience. Users recommend different items based on his choices in collaboration with choices of other users. For example, news articles, products, and movies are recommended to users based on user's previous ratings and also induced by the ratings of other users. Collaborative filtering techniques estimate item ratings or preference scores for a user, based on the knowledge of the ratings of the same user as well as ratings of the other users. User X's rating on an item i is based on the ratings on item i given by other users with similar ratings as that of X on other items. Collaborative filtering algorithms learn and discover user preferences on items and form the basis of recommendation systems. As more users and items are added to the data, it is important to have collaborative filtering algorithms which are scalable and efficient. In this article, we build collaborative filtering models based on matrix approximation and optimized using stochastic gradient descent (SGD) approach. Our matrix approximation models have a run-time complexity of $O(n)$ per epoch where n is the number of available ratings during training phase and $O(1)$ to predict rating for each test example. On the Movielens data set of 100,000 users for 1682 movies by 943 users, the best mean absolute error (MAE) and the best mean squared error (MSE) we obtained are 0.6899 and 0.8841 respectively.

2 Collaborative Filtering Using Matrix Approximation

2.1 Background

The goal of collaborative filtering is to predict missing item ratings for users given a database of user ratings. Each user rating can be represented in a triplet $< i, j, x_{ij} >$ where i is the user, j is the item and x_{ij} is the rating of user i for item j . The database of user ratings can be represented as a matrix where each row, i, ($1 \leq i \leq m$, m users in total), represents a single user, and each column j, ($1 \leq j \leq n$, n items in total). Based on a provided set of ratings in the tuple form, the matrix representation contains ratings in positions found in the given data set, while other ratings in other positions are missing. The objective of the collaborative filtering taks is to predict the missing ratings.

2.2 Modeling

Ratings, which are usually integers in the range $[1, 5]$, are a result of contributions from latent user's properties and item's properties. If the user i's contribution is r_i and item j's contribution is c_j, then the additive approximation a_{ij} for x_{ij} can be written as, $a_{ij} = r_i + c_j$. Similarly, a multiplicative approximation can be written as $a_{ij} = r_i \cdot c_j$.

The optimal additive model can be arrived at by minimizing MSE noted by Equation 1.

$$\sum_{<i,j>\in I} (r_i + c_j - x_{ij})^2 \tag{1}$$

The additive model can be represented as shown in Equation 2

$$Az = b \tag{2}$$

where $z =< r_1, r_2, ..., r_m, c_1, c_2, ..., c_n >'$ are the values that need to be learned, b is a vector of $m + n$ ratings and A is a $m + n$ by $m + n$ is a sparse matrix with ones at positions i and $m + j$ for $< i, j >\in I$ where I is the set for which x_{ij} is non-zero. z needs to be calculated by minimizing $||Az - b||_2$. Though, Moore-Penrose pseudo-inverse of A yields $z = (A'A)^{-1}A'b$, the inverse operation of a $m + n$ by $m + n$ matrix is an expensive operation in order of $O((m + n)^3)$ using Gauss-Jordan elimination.

2.3 Optimization using SGD

SGD can be used to minimize an objective function, a $L2$ regularized loss function, E, in this case, with μ as the regularization strength is shown in Equation 3.

$$E = \mu(\sum_i r_i^2 + \sum_j c_j^2) + \sum_{<i,j>\in I} |f(r_i, c_j) - x_{ij}|^p \tag{3}$$

where, $f(r_i, c_j)$ is a prediction function which, for example, could be $f(r_i, c_j) = r_i + c_j$ or $f(r_i, c_j) = r_i \cdot c_j$ in additive and multiplicative models respectively.

In order to learn r_i and c_j, we apply SGD as shown by the following equations

$$r_i \coloneqq r_i - \lambda(\frac{\partial E}{\partial r_i}) \tag{4}$$

$$c_j \coloneqq c_j - \lambda(\frac{\partial E}{\partial c_j}) \tag{5}$$

where λ is the learning rate. SGD approximates the true gradient at each training example. The algorithm sweeps through the training set and updates r_i and c_j. Several passes (epochs) of the training set are performed till the values converge. Parameters such as μ, λ and epochs need calibration. The advantages of SGD over matrix inversion or gradient descent are a) its inexpensive operations as gradient is approximated for a single example, b) ease of implementation and c) scalability as the convergence rate does not seem to increase with an increase in the size of the data set.

3 Data For Collaborative Filtering

For implementing our collaborative filtering models, we used the MovieLens dataset available at http://www.grouplens.org/node/73. The dataset contains 100,000 ratings given by 943 users for 1682 movies. The matrix represented by 943 rows (users) and 1682 columns (movies) contains 1,586,126 elements(ratings) out of which 100,000 ratings are provided. The sparsity of the given information matrix is almost 94%. The task of the collaborative learning models is to predict the remaining ratings in the matrix using different matrix factorization models. Data is presented in $< user, movie, rating >$ format where each user has rated at least 20 movies. Figure 1 displays exploratory analysis results on the whole 100K data. Rating 4 was most frequent. And histograms of both items-per-user and users-per-item decreased exponentially. Average rating did not show increasing pattern with an increase of rated items. While some users give systematically higher ratings, some items receive systematically higher ratings. These observations justify adoption of collaboration filtering for prediction of missing ratings.

Data Subsets. The provided data also contains 5 different training and test example subsets, named *ui.base* and *ui.test* respectively, $i \in 1, 2, 3, 4, 5$, constructed from the above data of 100,000 ratings after a 80-20 split. The subsets are used for 5-fold cross-validation in our experiments.

4 Collaborative Filtering Modeling Experiments

We will review multiple models of interest, then expand on the best model. For comparison we also look at some more trivial models based on arithmetical means.

Let $x_{ij} \in \{0, 1, 2, 3, 4, 5\}$ be the rating by user i for movie j if $x_{ij} > 0$ and define missing as $x_{ij} = 0$ where i and j range $\{1, 2, ..., m\}$ and $\{1, 2, ..., n\}$ respectively. In practical terms the training and test data are originally represented as a table of K examples, i.e., rows of triples $\langle i, j, x_{ij} \rangle_k, k \in \{1, 2, ..., K\}$. In this context an existing rating within a row k can be represented element wise as $x^k = x_{ij} > 0$ or vectorized as just x.

For the purpose of prediction and training we translate this in to two sparse matrices, $R \in \{0, 1\}^{K \times m}$ for the users and $C \in \{0, 1\}^{K \times n}$ for the items, where the respective elements $a^{(R)}_{ki}$ and $a^{(C)}_{kj}$ are defined as $a^{(R)}_{ki} = I(x_{ij} > 0), \quad j \in \{1, 2, ..., m\}$ and $a^{(C)}_{kj} = I(x_{ij} > 0), \quad i = \{1, 2, ..., n\}$. Here $I(\cdot)$ is the indicator function, equals to one if true and zero if false. Finally, a prediction, or approximation, of a rating $x^{(k)}$ will be denoted $\tilde{x}^{(k)}$ or vectorized as just $\tilde{x}$. If need be, the notation $\tilde{x}_{ij}$ will also be used for the prediction of x_{ij}.

4.1 Mean Models

All presented mean based model uses vectors $r \in \Re^m$ and $c \in \Re^n$, with elements r_i and c_j, as learning parameters. Vector r corresponds to all users and vector c corresponds to all movies. Prediction is done as a single matrix

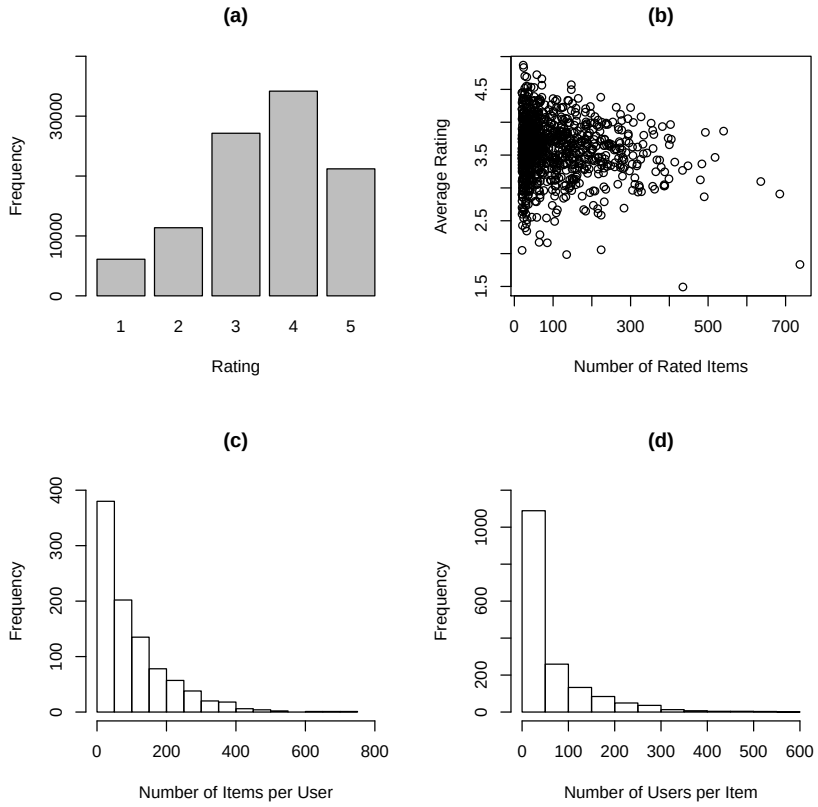

Figure 1: Histograms on 100K dataset. (a) Histogram of 5 ratings; (b) Average rating vs. Number of rated items; (c) Histogram of number of items per user; (c) Histogram of number of users per item.

multiplication of the following block matrix and block vector.

$$A = [R \quad C], \quad z = [r \quad c]^T \text{ where, } \tilde{x} = [R \quad C][r \quad c]^T = Az$$

Finally, let the following means be defined as

$$\bar{x}_i = \frac{\sum_{j=1}^{n} x_{ij}}{\sum_{k=1}^{K} a_{ki}^{(R)}} \text{ and } \bar{x}_j = \frac{\sum_{i=1}^{m} x_{ij}}{\sum_{k=1}^{K} a_{kj}^{(C)}}$$

When user ratings for a specific movie are zero or are not provided by the user, we use the global mean of all ratings, give by

$$\bar{x}_i = \bar{y_j} = \frac{\sum_{k=1}^{K} x_k^{(C)}}{K}$$

for initialization and bootstrapping purposes.

User Mean Model The user mean model just takes the average of all ratings a person has given and returns this as the prediction for unrated movies. This has no predictive power from the user's point of view. This is represented by,

$$r_i = \bar{x}_{i.}, \quad c_j = 0$$

Item Mean Model The item mean model is a little more reasonable and averages all ratings a movie has. This can be used for a reasonable prediction of a rating. This is represented by,

$$r_i = 0, \quad c_j = \bar{x}_{j.}$$

Bi-Mean Model The bi-mean is a naive way of combining the user mean and items mean. It takes the average of both user and item means. Although, a co-efficient of 1/2 is used to find the combination both the means, different co-efficients can also be used. This is represented by,

$$r_i = \tfrac{1}{2}\bar{x}_{i.}, \quad c_j = \tfrac{1}{2}\bar{x}_{.j}$$

Bias from Mean Model The bias from mean model works by finding the user's average rating r_i as a mean on which c_j can vary around. The interpretation of r_i and c_i is that each person tends to rate at generally high or generally low levels r_i and the amount c_i that deviates from this basis comes from all users who have rated the move. This model is represented by,

$$r_i = r_i, \quad c_j = \frac{\sum_{i=1}^{m}(x_{ij} - \bar{x}_{i.})}{\sum_{k=1}^{K} a_{kj}^{C}}.$$

In words, to find the deviation c_j of movie j, sum of all ratings less each user's average rating can be calculated which can then be divided by the number of raters.

4.2 Additive Model

Our additive model is based on the formulation of the prediction function $f(r_i, c_j)$ as shown by the following equation.

$$f(r_i, c_j) = r_i + c_j \tag{6}$$

r_i and c_j are learned by minimizing the square-loss and employing regularized SGD as shown in the following equations where λ is learning rate and μ is regularization strength .

$$E = \mu(\sum_i r_i^2 + \sum_j c_j^2) + \sum_{<i,j>\in I} (r_i + c_j - x_{ij})^2 \tag{7}$$

$$r_i \coloneqq r_i - 2\lambda\mu r_i - 2\lambda(r_i + c_j - x_{ij}) \tag{8}$$

$$c_j \coloneqq c_j - 2\lambda\mu c_j - 2\lambda(r_i + c_j - x_{ij}) \tag{9}$$

4.3 Multiplicative Model

Our multiplicative model is based on the formulation of the prediction function $f(r_i, c_j)$ as shown by the following equation.

$$f(r_i, c_j) = r_i \cdot c_j \tag{10}$$

r_i and c_j are learned by minimizing the square-loss and employing regularized SGD as shown in the following equations.

$$E = \mu(\sum_i r_i^2 + \sum_j c_j^2) + \sum_{<i,j>\in I} (r_i \cdot c_j - x_{ij})^2 \tag{11}$$

$$r_i \coloneqq r_i - 2\lambda\mu r_i - 2\lambda(r_i + c_j - x_{ij})c_j \tag{12}$$

$$c_j \coloneqq c_j - 2\lambda\mu c_j - 2\lambda(r_i + c_j - x_{ij})r_i \tag{13}$$

4.4 Combined Model

Our combined model is built by incorporating the additive and multplicative models simultaneously.

$$\tilde{x}_{ij} = r_i^{(1)}c_j^{(1)} + r_i^{(2)}c_j^{(2)} + ... + r_i^{(L)}c_j^{(L)} \tag{14}$$

where $L \in \mathcal{N}_1$ is a meta-parameter that needs to be learned. Given $l = 1, 2, ..., L$ and $i = 1, ..., m$, we can, for purposes of implementation, enter all $r_i^l \in \Re$ into a vector $r^l \in \Re^m$ or matrix $r \in \Re^{m\times L}$, likewise, we can enter $c_j^{(l)} \in \Re$ into a vector $c^{(l)} \in \Re^n$ or matrix $c \in \Re^{n\times L}$. We thus have a way to simultaneously make predictions $\tilde{x} \in \Re^K$ for all K presented by the test examples $\langle i, j\rangle_{k\in\{1,...,K\}}$ as follows.

$$\tilde{x} = \sum_{l=1}^{L}(Rr^l \bullet Cc^l) \tag{15}$$

where we define $u \bullet v = diag(u)v \in \Re_q$ be the notation for component wise multiplication for some $u, v \in \Re^q$.

Stochastic Gradient Descent. SGD is applied to learn r_i and c_j. For the combination model, following are the SGD update rules for $r_i^{(l)}$ and $c_j^{(l)}$.

$$r_i^{(l)} \coloneqq r_i^{(l)} - 2\lambda\mu r_i^{(l)} - 2\lambda(\tilde{x}_{ij} - x_{ij})c_j^{(l)} \tag{16}$$

$$c_j^{(l)} \coloneqq c_j^{(l)} - 2\lambda\mu c_j^{(l)} - 2\lambda(\tilde{x}_{ij} - x_{ij})r_i^{(l)} \tag{17}$$

Learning rate λ and regularization strength μ are fixed meta-parameters that need to be optimized separately. Learning process is terminated when the change in mean squared error MSE_e for epoch e falls below a threshold δ during training phase. Learning is continued if $MSE_e - MSE_{e-1} > \delta$. Learning will also stop when maximum number of epochs, $epoch_{max}$ are completed.

5 Experiment Setup

Performance Measure. Mean Squared Error (MSE), Mean Absolute Error (MAE) are used as the performance measure. The lower the values the better the model. MSE is calculated as follows.

$$MSE = \frac{1}{K}\sum_{k=1}^{K}(\tilde{x}^{(k)} - x^{(k)})^2 \tag{18}$$

MAE is calculated by the following equation.

$$MAE = \frac{1}{K}\sum_{k=1}^{K}|\tilde{x}^{(k)} - x^{(k)}| \tag{19}$$

Meta-parameters. $\lambda, \mu, L, \delta, \text{epoch}_{\max}$ need to be calibrated. We conduct a three-dimensional grid search for λ, μ and L while setting δ and $\text{epoch}_{\max}$ to computational tractable sizes by human intuition and by trial-and-error. The following grid-search windows are chosen for parameters,

$$\lambda \in \{2^{-i}\}_{i=5}^{11}, \quad \mu \in \{2^{-i}\}_{i=3}^{11}, \quad L \in \{1, 2, 3\}$$

The grid was initially smaller, but expanded until optimal border cases were uncovered. In addition to these values we used the following fixed meta-parameters,

$$\delta = 2^{-15}, \quad \text{epoch}_{\max} = 2^{10}$$

The above settings let the models converge until sufficiently small change in MSE is repeatedly observed under the predetermined epoch number.

Cross Validation. In order to get fair measurements we performed 5-fold cross-validation for all measurements. That is, we generate five sets of performance metrics, i.e., MSE, and average over the folds. This allows us to use more data, without overfitting. The training data was randomized then split evenly among 20% partitions yielding $u1, u2, u3, u4$ and $u5$ training and test data subsets. Training was done using, for example, $u1.base$ and the resultant model is used to test $u1.test$. This is repeated for each set of training and test data. Each fold was used exactly once for training (per meta-parameter triplet). The remaining four folds were used to train the actual model that did the prediction for this fold of test data.

6 Results and Analysis

Trivial Mean Models. The performance of the mean models is reported in Table 1. As can be seen, the performance increases as the models get more 'clever' with the best one being the bias from mean model.

Model	**Average MSE on Test Data**
User Mean	1.0895
Item Mean	1.0498
Bi-Mean	0.9662
Bias From Mean	0.9264

Table 1: MSE of Mean Models

Combination Model. We conducted a broad grid search for the three main meta-parameters of the combined model, $\lambda, \mu and L$. Model length $L = 2$ performs unequivocally better than models based on $L = 1$ and $L = 3$ which have

L=2	$\mu = 2^{-8}$	$\mu = 2^{-9}$	$\mu = 2^{-10}$	$\mu = 2^{-11}$
$\lambda = 2^{-8}$	0.906	0.925	0.922	0.946
$\lambda = 2^{-9}$	0.891	0.886	0.886	0.9077
$\lambda = 2^{-10}$	0.885	0.885	**0.884**	0.8961
$\lambda = 2^{-11}$	0.899	0.888	0.886	0.9072

Table 2: Subset of the Combination Model's MSE grid search

MSE of 0.9034 and 0.9033 respectively. From the MSE results of models based on $L = 2$, which are shown on Table 2, we see that the best performing meta-parameters are $\lambda = 2^{-10}$ and $\mu = 2^{-10}$. Furthermore, we can also see that the regularization strength has negligible sensitivity among $\mu, \lambda \in \{2^{-8}, 2^{-9}, 2^{-10}\}$ x $\{2^{-8}, 2^{-9}, 2^{-10}\}$.

The table presented in Table 3 compares MSE and MAE of all the models (mean and combination models). From the results, it can be seen that the combination model performs significantly better than the mean models.

Model	**Average MSE on Test Data**	**Average MAE on Test Data**
User Mean	1.0895	0.8362
Item Mean	1.0498	0.8174
Bi-Mean	0.9662	0.7951
Bias From Mean	0.9264	0.7590
Combination Model	0.8841	0.6899

Table 3: Final resulting MSE and MAE of all models

Computational Complexity. For our implementation of the combination model using SGD, the space complexity is $O(K)$ and run-time complexity of the training phase is $O(nK)$ where n is the number of epochs consumed and K is the number of training examples. The complexity of the test phase is constant time per example. The number of training examples is the dominant contributor to the complexity as number of epochs tend to stabilise at low 50s and do not increase with increase in dataset size. In Figure 2, we see the learning progress over time, i.e, training MSE versus epochs. Of the 560 epochs used on the final model we see a significant decrease in MSE for the first 50 epochs. For $L = 2$ and $K = 80,000$ each epoch took about 70 ms. This bodes well for performance on a larger dataset, however, it is unclear if the full dataset $K = 10^8$ of Netflix is manageable, with regard to memory, on a computer desktop. A parallel and distributed implementation of SGD would handle larger datasets with considerable ease as a single SGD updates can be divided into smaller updates based on smaller subsets with each update parallelized. Such an implementation can lead to better a performing model with high degree of scalability.

7 Conclusion

Collaborative filtering with high accuracy and scalability are important both to the businesses to create user satisfaction and also to the user to have a richer experience. It is becoming imperative for recommender systems to produce high quality of predictions and at the same time be seamlessly scalable. With the inundation of 'Big Data', collaborative filtering algorithms would be under stress without efficient algorithms. In this article, we implemented collaborative filtering algorithms, ranging from trivial to higher-accuracy combination models. Our additive, multiplicative and combination models employ SGD technique and have a linear run-time and space complexity and can scale well to millions of data records provided convergence occurs in few epochs. Applying parallel and distributed strategies for storage and computation, we believe that the collaborative filtering system becomes all the more scalable and responsive.

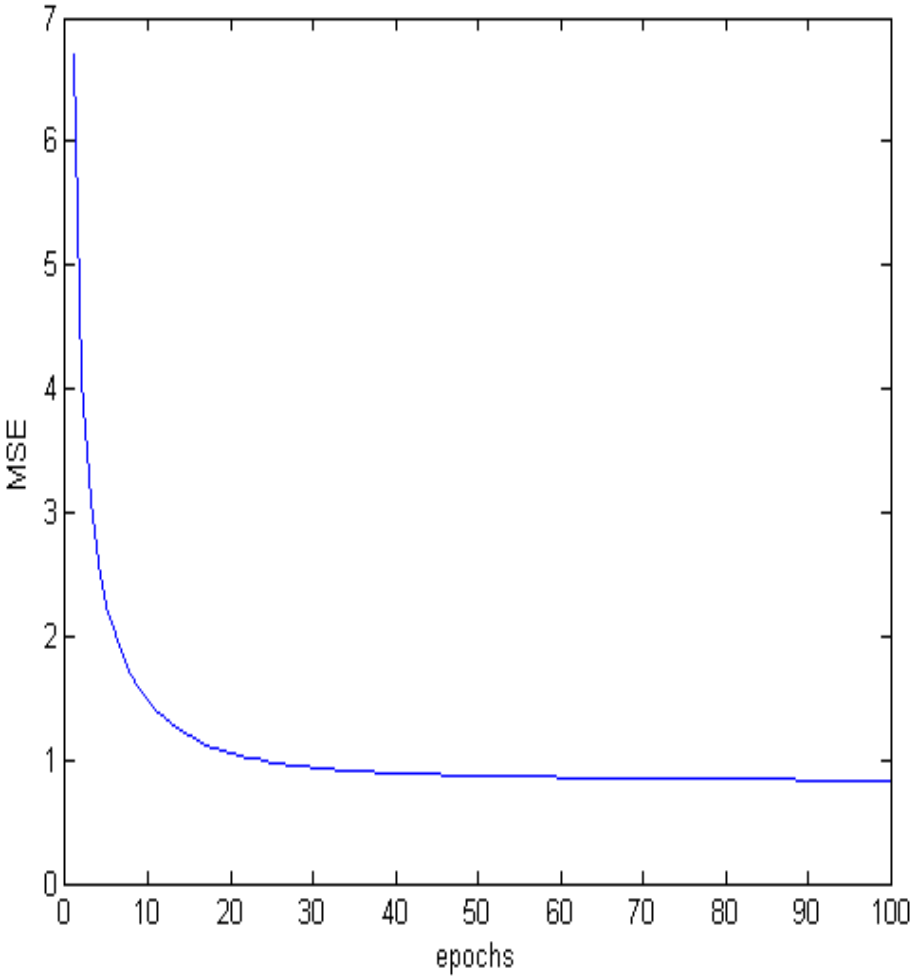

Figure 2: MSE vs Epochs for Combination Model