Matching The CineMatch: Analysis for Movie Recommender Systems

Shikhar Jaiswal
1601CS44
Department of Computer Science and Engineering
Indian Institute of Technology Patna
shikhar.cs16@iitp.ac.in

Abstract

Movie recommendation algorithms are one of the most prolific domains of study in the world of machine learning. Due to their wide area of application, recommendation systems have become an area of active research as well. Several machine learning related algorithms - Baseline Predictor, K-Nearest Neighbors, Singular Value Decomposition, Restricted Boltzmann Machines and so on are regularly used to predict the ratings for the users for which certain movies are unrated. These different models are compared using Root-Mean-Square-Error (RMSE) and Mean-Absolute-Error (MAE) criterion, to evaluate their performance.

1 Introduction

Recommender systems provide users with personalized suggestions for products or services. Generally speaking, a recommendation system builds up item's profile and user's profile based on their previous recorded behavior. Then it makes a prediction based on the rating given by a certain user on a certain item that they have not yet evaluated.Based on the predictions, the system makes recommendations. Various techniques for recommendation generation have been proposed over the years and have been successfully deployed in commercial environments, among which Collaborative Filtering (CF) and content-based methods were most commonly used [2, 3]. In this paper, we aim to use an ensemble combination of machine learning algorithms to build a full-scale movie recommendation system that offers comparable performance to the Netflix CineMatch (2009) Algorithm.

2 The Dataset

For training and model selection purposes, MovieLens 10M Dataset [1] is used. This contains the information about the movies and demographic data for the users, which we require for our content-based analysis. Approximately 10,000,000 ratings applied to 11,000 movies by 72,000 users between January, 1995 to January, 2009 are present in this dataset. Also, each of the user-item ratings has a timestamp as well. We have used nearly 8,000,000 ratings for training, and split the remaining 2,000,000 equally between validation and test sets.

Final RMSE benchmarking is done using the original Netflix Cinematch Dataset consisting of data from more than 480,000 users. Benchmarking is also done across different hyperparameter specifications and the best results from a model, or a combination of models, are be compared to the Netflix benchmark.

3 The Problem Statement

We prepare a large user-item matrix $M \in \mathcal{R}^{N_u \times N_i}$, where N_u is the number of users and N_i is the number of items, or in our case, movies:

$$M_{u,i} = \begin{cases} r_{u,i}, & \text{existent rating,} \\ 0, & \text{no rating} \end{cases}$$

Thus our work is to fill in all the zero-entries in the matrix based on the training set if existing ratings. Assume the predicted rating of user u in the test of item i is $t_{u,i}$. The widely used RMSE and MAE criteria are applied to evaluate performance:

$$RMSE = \sqrt{\frac{\Sigma_{(u,i)\in S_{test}}(r_{u,i} - t_{u,i})^2}{|S_{test}|}}$$

$$MAE = \frac{\Sigma_{(u,i)\in S_{test}} |r_{u,i} - t_{u,i}|}{|S_{test}|}$$

Here, S_{test} is the set of all ratings in the test set.

4 Challenges

4.1 Data Sparsity

Perhaps the biggest issue facing recommender systems is that they need a lot of data to effectively make recommendations. The more item and user data a recommender system has to work with, the stronger the chances of getting good recommendations. It nearly forms a virtuous cycle to get good recommendations, you need a lot of users, so you can get a lot of data for the recommendations.

4.2 Changing User Preferences

Systems are usually biased towards the old data and have difficulty showing new trends. Past behaviors of buyers often create problems since the trends in the consumer space are always changing.

4.3 Unpredictability

It has been noted, especially in the movie recommendation domain that there was an issue with eccentric movies, i.e., the type of movie that people either love or hate. These type of items are difficult to make recommendations on, because the user reaction to them tends to be diverse and unpredictable.

4.4 Scalability

Netflix operates at the level of more than 15 billion ratings and well over 115 million streaming channels. Moreover, Netflix's complex user personalization offerings, coupled with the need to cater to an ever-growing user base, only speaks aloud of the need for the most scalable techniques and approaches to achieve the best results.

5 The Work Approach

5.1 Pre-Processing

The training and test data for both the datasets is prepared as follows:

We make use of the database to build a U by I matrix, where U is the number of users, and I is the number of rated movies. Each element M_{ui} denotes the rating scored by the u^{th} user for the i^{th} movie. It is easy to find that the majority of the movies do not obtain a sufficient number of ratings, and also, there only exist common ratings for a general user. Hence, naturally, we obtain a sparse matrix.

5.2 Algorithms Implemented

5.2.1 Baseline Predictor

For the sake of reference, we denote the baseline prediction for user u and item i by $b_{u,i}$.
The simplest baseline is to predict the average of all the ratings in the system:

$$b_{u,i} = \mu$$

where, μ is the overall average rating. This can be improved upon by combining the mean for the users with the average deviation from user mean ratings for an individual item. Generally, a baseline predictor of the following form can be used:

$$b_{u,i} = \mu + b_u + b_i \tag{1}$$

where, b_u and b_i are the user and item baselines, respectively. They can be defined as:

$$b_u = \frac{1}{|I_u|}\Sigma_{i \in I_u}(r_{u,i} - \mu)$$

$$b_i = \frac{1}{|U_i|}\Sigma_{u \in U_i}(r_{u,i} - b_u - \mu)$$

where, I_u represents the set of all the movies rated by the user u and U_i represents the set of all the users that rated a particular movie i.

With Damping Terms

The baseline can be further regularized by providing a more reasonable estimate of user and item preferences to take care of data sparsity, with the incorporation of damping terms β_u and β_i, for user and item baseline damping respectively:

$$b_u = \frac{1}{|I_u| + \beta_u}\Sigma_{i \in I_u}(r_{u,i} - \mu) \tag{2}$$

$$b_i = \frac{1}{|U_i| + \beta_i}\Sigma_{u \in U_i}(r_{u,i} - b_u - \mu) \tag{3}$$

This adjustment causes the baseline predicted ratings to be closer to the global mean when the user has only a few ratings provided. This is based on the principle that the more ratings a user has provided or an item has received, the more is known about that user or item's true mean rating.

5.2.2 User-User Collaborative Filtering

To generate recommendations for a user, User-User Collaborative Filtering uses a similarity function $s(u, u')$ to compute a neighborhood $N \subseteq U$ of each user u in U. After this, the system combines the ratings of the users in N to generate u's

preferences for an item i. This is typically done by computing the weighted average of the neighboring user u''s ratings i, using similarity as the weights:

$$p_{u,i} = \overline{r}_u + \frac{\Sigma_{u' \in N} s(u,u')(r_{u',i} - \overline{r}_{u'})}{\Sigma_{u' \in N} |s(u,u')|}$$

Subtracting the overall mean rating provided by the user, $\overline{r}_u$, compensates for the difference in user' use of the rating scale (some users will tend to give higher ratings than the others). Equation above can also be extended to normalize the user ratings to z-scores by dividing the offset from mean rating by the standard deviation, σ_u of each user's ratings, thereby compensating for users differing in rating spread as well as mean rating:

$$p_{u,i} = \overline{r}_u + \sigma_u \frac{\Sigma_{u' \in N} s(u,u')(r_{u',i} - \overline{r}_{u'})/\sigma_{u'}}{\Sigma_{u' \in N} |s(u,u')|} \tag{4}$$

Similarity Function: Pearson Correlation

This method computes the statistical correlation (called Pearson's r) between two users' common ratings to determine their similarity as follows:

$$s'(u,u') = \frac{\Sigma_{i \in I_u \cap I_{u'}} (r_{u,i} - \overline{r}_u)(r_{u',i} - \overline{r}_{u'})}{\sqrt{\Sigma_{i \in I_u \cap I_{u'}} (r_{u,i} - \overline{r}_u)^2}}$$

$$s(u,u') = \frac{s'(u,u')}{\sqrt{\Sigma_{i \in I_u \cap I_{u'}} (r_{u',i} - \overline{r}_{u'})^2}} \tag{5}$$

Pearson correlation suffers from computing high similarity between users with few ratings in common. This can be alleviated by setting a threshold on the number of co-rated items necessary for full agreement (correlation of 1) and scaling the similarity when the number of items fall below this predefined threshold.

5.2.3 Item-Item Collaborative Filtering

User-User Collaborative Filtering, while effective, suffers from scalability problems as the user base grows, as a neighborhood search for a user requires, O($|U|$) operations.

Item-Item Collaborative Filtering takes a major step in this direction and is one of the most widely deployed collaborative filtering techniques today. It is based on the principle that if two items tend to have the same users like and dislike them, then they are similar and the users are expected to have similar preferences for similar items.

Item-Item Collaborative Filtering generates recommendations by by using the user's own ratings for other items combined by with those items' similarities to the target item, rather than other users' ratings and user similarities as in User-User Collaborative Filtering. Much like the previous algorithm, this system needs a similarity function, $s : I \times I \rightarrow \mathcal{R}$, and a method to generate recommendations from ratings and similarities.

Computing Recommendations

After collecting a set K of movies similar to i, $p_{u,i}$ can be computed as:

$$p_{u,i} = \frac{\Sigma_{j \in K} s(i,j) r_{u,j}}{\Sigma_{j \in K} |s(i,j)|}$$

The above equation, as it stands, suffers from a deficiency when it is possible for similarity scores to be negative, as the ratings are constrained to be non-negative; some of the ratings averaged together to compute the predictions may be negative after weighting. While this will not affect the relative ordering of items by predicted value, it will bias them, and we might obtain values that do not map back to the user rating domain. This can be corrected either by thresholding similarities so that only items with non-negative similarities are considered, or by averaging distance from the baseline predictor:

$$p_{u,i} = \frac{\Sigma_{j \in K} s(i,j)(r_{u,j} - b_{u,i})}{\Sigma_{j \in K} |s(i,j)|} + b_{u,i} \tag{6}$$

Similarity Function: Cosine Similarity

$$s'(i,i') = \frac{\Sigma_{u \in U_i \cap U_{i'}} (r_{u,i} - \overline{r}_u)(r_{u,i'} - \overline{r}_u)}{\sqrt{\Sigma_{u \in U_i \cap U_{i'}} (r_{u,i} - \overline{r}_u)^2}}$$

$$s(i,i') = \frac{s'(i,i')}{\sqrt{\Sigma_{u \in U_i \cap U_{i'}} (r_{u,i'} - \overline{r}_u)^2}} \tag{7}$$

The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality (de-correlation), and in-between values indicating intermediate similarity or dissimilarity.

5.2.4 Singular Value Decomposition

Singular Value Decomposition, or SVD, is a factorization algorithm for collaborative filtering. This family of algorithms finds the features of users and items, and makes predictions based on these factors.

Formulation

Suppose $V \in \mathcal{R}^{n \times m}$ is the overall rating matrix for m movies and n users, and $I \in \{0,1\}^{n \times m}$. The SVD algorithm finds two matrices $U \in \mathcal{R}^{f \times n}$ and $M \in \mathcal{R}^{f \times m}$ as the feature matrix of users and objects, respectively. Hence, each user has an f-dimensional feature vector and f is called the dimension of SVD. A prediction function p is used to predict the values in V.The value of a score $V_{i,j}$ is estimated by $p(U_i, M_j)$, where U_i and M_j represent the feature vectors of user i and object j respectively. Once U and M are found, the missing scores in V can be predicted by the prediction function. Hence, the objective function is obtained by minimizing the sum of squares of difference between existing scores and their predicted values:

$$E = \frac{1}{2}\Sigma_{i \in n}\Sigma_{j \in m} I_{i,j}(V_{i,j} - p(U_i, M_j))^2 \quad (8)$$

The most common prediction function is the dot product of the feature vectors, which basically follows from *Matrix Factorization* technique:

$$p(U_i, M_j) = U_i^T M_j$$

In most applications, the scores are confined to be in an interval [a, b] where a and b are the minimal, so if the users rate a movie in the range 1 - 5 stars, the scores need to be bounded in interval [1, 5]. We modify the prediction function as:

$$p(U_i, M_j) = \begin{cases} a, & \text{if } U_i^T M_j < 0, \\ a + U_i^T M_j, & \text{if } 0 \leq U_i^T M_j \text{ and} \\ & U_i^T M_j \leq b - a, \\ b, & \text{if } U_i^T M_j > b - a \end{cases}$$

Though this prediction is non-differentiable at the end points, it does not cause problems during optimization as most of the prediction values are in the differentiable range. Plugging this in equation (8), we can now apply gradient descent for optimization of U and M matrices.

Optimization: Incremental Gradient Descent

The performance of this algorithm may suffer from overfitting, which can be tackled by introducing $L2$ regularization. Here, we follow an incremental learning approach, as the sum of E_i over all users would lead to the objective function:

$$\Sigma_{i \in n} E_i = \frac{1}{2}\Sigma_{i \in n}\Sigma_{j \in m} I_{i,j}(V_{i,j} - p(U_i, M_j))^2 + \frac{k_u}{2}\Sigma_{i \in n}\|U_i\|^2 + \frac{k_m}{2}\Sigma_{j \in m}\Sigma_{i \in n} K_{i,j}(\|M_j\|^2) \quad (9)$$

Here, k_u and k_m serve as regularization coefficients. Each object feature vector M_j has additional regularization coefficient $K_{i,j}$ which is equal to the number of existing scores for the object j. Therefore, an object with a higher number of scores has a larger regularization coefficient in this incremental learning approach. The objective function and the gradients have the following form:

$$E_{i,j} = \frac{1}{2}(V_{i,j} - p(U_i, M_j))^2 + \frac{k_u}{2}(\|U_i\|^2) + \frac{k_m}{2}(\|M_j\|^2) \quad (10)$$

$$-\frac{\partial E_{i,j}}{\partial U_i} = (V_{i,j} - p(U_i, M_j))M_j - k_u U_i \quad (11)$$

$$-\frac{\partial E_{i,j}}{\partial M_j} = (V_{i,j} - p(U_i, M_j))U_i - k_m M_j \quad (12)$$

The summation of $E_{i,j}$ over all existing ratings is:

$$\Sigma_{i \in n}\Sigma_{j \in m} E_{i,j} = \frac{1}{2}\Sigma_{i \in n}\Sigma_{j \in m} I_{i,j}(V_{i,j} - p(U_i, M_j))^2 + \frac{k_u}{2}\Sigma_{i \in n}\Sigma_{j \in m} K_{i,j}(\|U_i\|^2) + \frac{k_m}{2}\Sigma_{j \in m}\Sigma_{i \in n} K_{i,j}(\|M_j\|^2) \quad (13)$$

In this case, each user and movie has a regularization coefficient.

With Biases

Variants of the SVD algorithm can provide better accuracy. One method is to introduce a per-user $\in \mathcal{R}^{n \times 1}$ and a per-movie bias $\in \mathcal{R}^{m \times 1}$ on the prediction function. Hence the function becomes:

$$p(U_i, M_j, \alpha_i, \beta_j) = U_i^T M_j + \alpha_i + \beta_j \quad (14)$$

where α_i is bias for user i and β_j is bias for the movie j. Biases are updated like feature values. Hence evaluating the gradients of the above equation:

$$E_{i,j} = \frac{1}{2}(V_{i,j} - p(U_i, M_j, \alpha_i, \beta_j))^2 + \frac{k_u}{2}(\|U_i\|^2) + \frac{k_m}{2}(\|M_j\|^2) + \frac{k_b}{2}(\alpha_i^2 + \beta_j^2) \quad (15)$$

$$-\frac{\partial E_{i,j}}{\partial \alpha_i} = (V_{i,j} - p(U_i, M_j, \alpha_i, \beta_j)) - k_b \alpha_i \quad (16)$$

$$-\frac{\partial E_{i,j}}{\partial \beta_j} = (V_{i,j} - p(U_i, M_j, \alpha_i, \beta_j)) - k_b \beta_j \quad (17)$$

5.2.5 Restricted Boltzmann Machines

Restricted Boltzmann Machines are a type of artificial neural networks, characterized as a class of two-layer undirected graphical models. We use a unique RBM for each user. Every RBM has the same number of hidden units, but an RBM only has visible softmax units for the movies rated by that particular user, so an RBM has few connections if that user rated few movies. Each RBM only has a single training case, but all of the corresponding weights and biases are tied together, so if two users have rated the same movie, their two RBMs must use the same weights between the softmax visible unit for that movie and the hidden units. The binary states of the hidden units, however, can be quite different for different users. From now on, to simplify the notation, we will concentrate on getting the gradients for the parameters of a single user-specific RBM. The full gradients with respect to the shared weight parameters can then be obtained by averaging over all n users.

Formulation

Let V be a $K \times m$ observed binary indicator matrix with $v_i^k = 1$ if the user rated movie i as k and 0 otherwise. We also let h_j, $j = 1...f$, be the binary values of hidden (latent) variables, that can be thought of as representing stochastic binary features that have different values for different users. Here K is the maximum rating that can be assigned to a movie by a user.

We use a conditional multinomial distribution for modeling each column of the observed visible binary rating matrix V and a conditional Bernoulli distribution for modeling hidden user features h. There is a more subtle source of information in the Netflix database that cannot be captured by the standard multinomial RBM. Netflix tells us in advance which user/movie pairs occur in the test set, so we have a third category: movies that were viewed but for which the rating is unknown.

The conditional RBM model takes this extra information into account. Let $r \in \{0,1\}^M$ be a binary vector of length M (total length of the movies), indicating which movies the user rated (even if these ratings are unknown). The idea is to define a joint distribution over (V, h) conditional on r.

$$p(v_i^k = 1|h) = \frac{exp(b_i^k + \Sigma_{j \in f}(h_j W_{i,j}^k))}{\Sigma_{l \in K} exp(b_i^l + \Sigma_{j \in f}(h_j W_{i,j}^l))} \quad (18)$$

$$p(h_j = 1|V, r) = \sigma(b_j + \Sigma_{i \in m}\Sigma_{k \in K}(v_i^k W_{i,j}^k) + \Sigma_{i \in M}(r_i D_{i,j})) \quad (19)$$

where $D_{i,j}$ is an element of a learned matrix that models the effect of r on h. $\sigma(x) = 1/(1 + e^{-x})$ is the logistic function, $W_{i,j}^k$ is a symmetric interaction parameter between feature j and rating k of movie i, b_i^k is the bias of rating k for movie i, and b_j is the bias of feature j.

Learning

The parameter updates required to perform gradient ascent in the log-likelihood can be obtained from above:

$$\Delta W_{i,j}^k = \epsilon(< v_i^k h_j >_{data} - < v_i^k h_j >_{model})$$

where ϵ is the learning rate. The expectation $< v_i^k h_j >_{data}$ defines the frequency with which movie i with rating k and feature j are on together when the features are being driven by the observed user-rating data from the training set, and $< . >_{model}$ is an expectation with respect to the distribution defined by the model. The expectation $< . >_{model}$ cannot be computed analytically in less than exponential time. To avoid computing it, we follow an approximation to the gradient of a different objective function called Contrastive Divergence (CD) [7].

$$\Delta W_{i,j}^k = \epsilon(< v_i^k h_j >_{data} - < v_i^k h_j >_T) \quad (20)$$

The expectation $< . >_T$ represents a distribution of samples from running the Gibbs sampler, initialized at the data, for T full steps. Learning D using CD is similar to learning biases and takes the form:

$$\Delta D_{i,j} = \epsilon(< h_j >_{data} - < h_j >_T) r_i \quad (21)$$

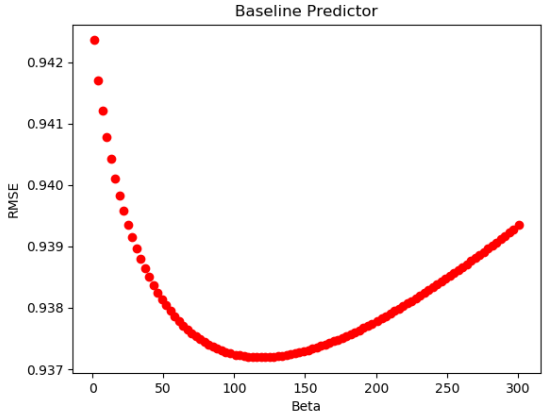
5.2.6 Hybrid Combinational Approach

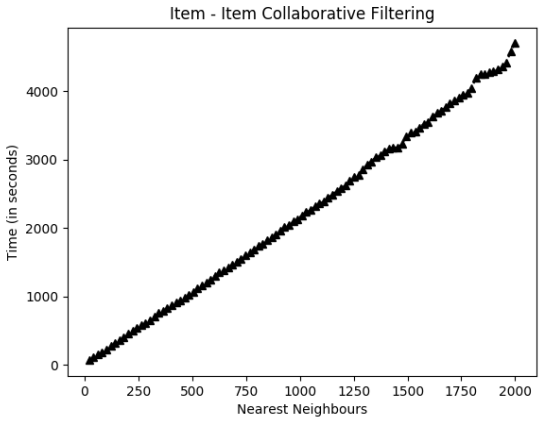
Here, we simply take the convex combination of predicted scores of the different models, that is:

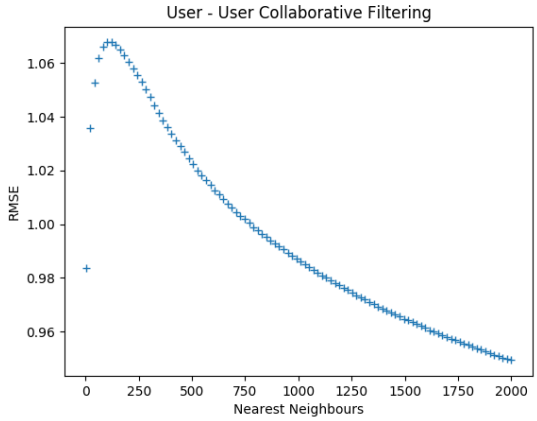
$$p(i, j) = \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_n x_n \quad (22)$$

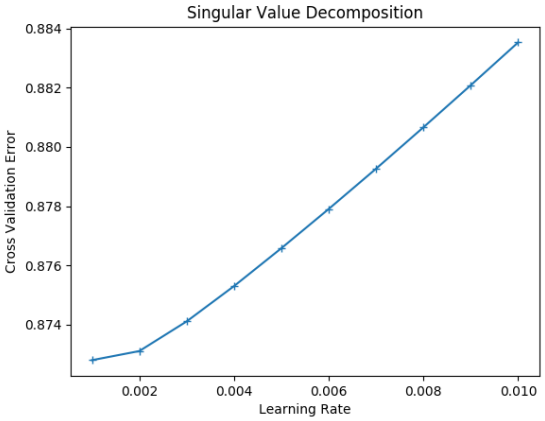
where, the real numbers α_i satisfy $\alpha_i \geq 0$ and $\Sigma_{i \in n} \alpha_i = 1$. Here the $p(i, j)$ corresponds to the predicted rating for all users i to the movies j. Hence, we optimize the coefficient values according to the minimization objective of the RMSE.

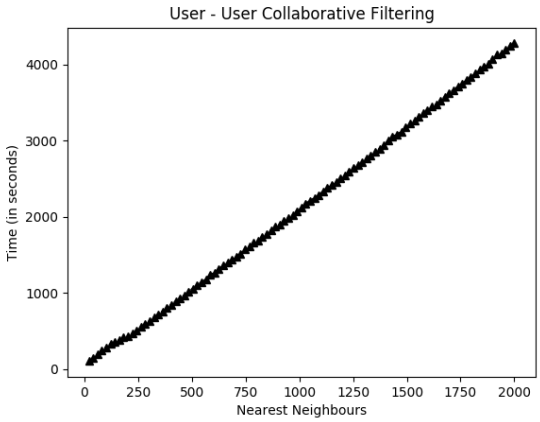
6 Benchmarks

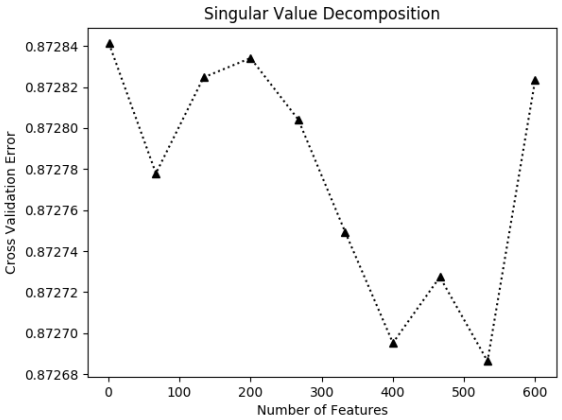


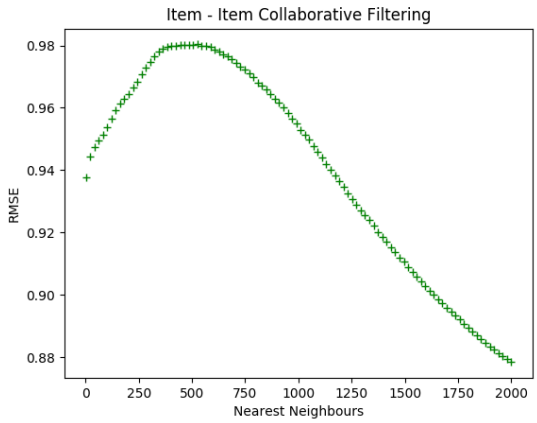


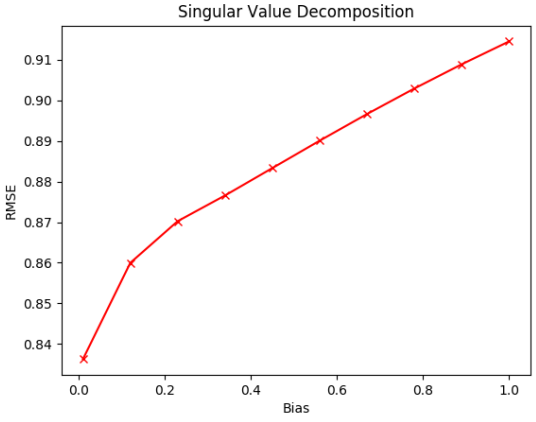


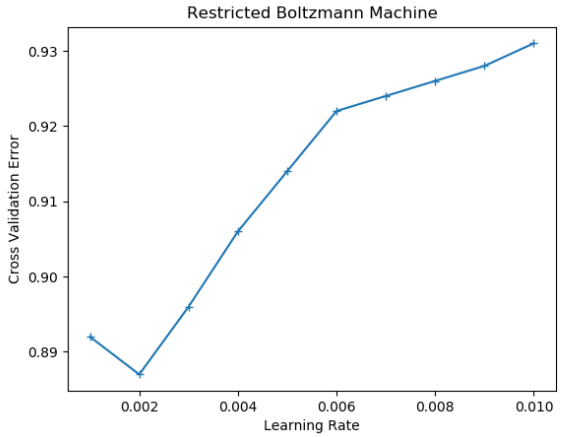


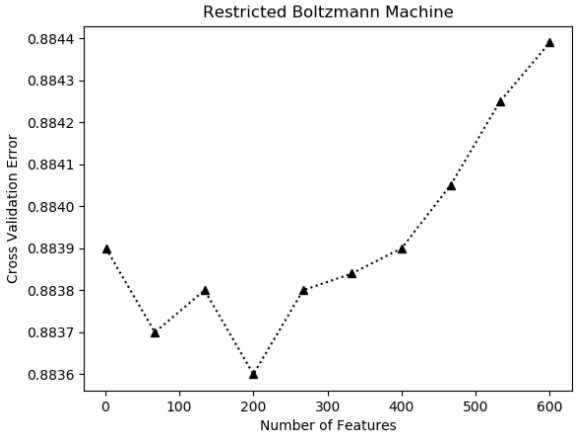


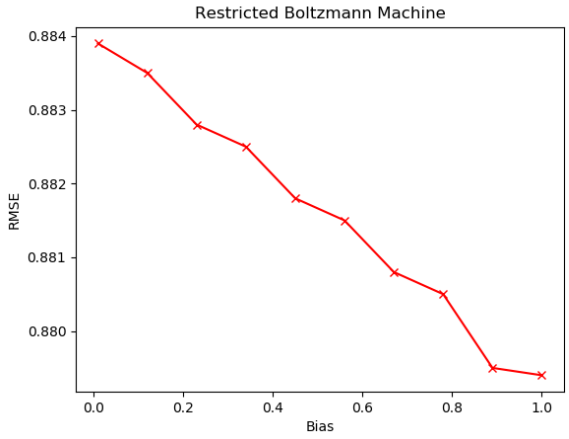












7 Future Work and Conclusion

This project was my first formal academic project, and as such, a great learning experience. This endeavor presented me with the opportunity to look under the hood and see for myself how the various black boxes in major ML libraries function. Also, it made me realize how difficult it is to fine tune ML models and obtain meaningful results from them, which I had only heard about. This project can be extended in many ways:

7.1 Learning Autoencoders

An alternative way of using an RBM is to treat this learning as a pre-training stage that finds a good region of the parameter space [6]. However, overfitting becomes an issue and more careful model regularization is required.

7.2 Learning Deep Belief Networks

Greedy learning of Deep Belief Networks (DBNs), one layer at a time, with the top two layers forming an undirected bipartite graph which acts as an associative memory, was derived recently [8].

The learning procedure consists of training a stack of RBMs each having only one layer of latent (hidden) feature detectors. The learned feature activations of one RBM are used as the data for training the next RBM in the stack. DBNs have been really useful in reducing error on many standard datasets.

For faster computation, one may also consider a faster implementation (in C++ or Cython), and may use a good virtual server, if available.

Acknowledgments

During the course of this project, I had faced a gargantuan number of issues and dilemmas with regards to the overall development workflow. I am thankful to the Netflix CineMatch team for providing the official benchmark as well as the test data for comparison. I am also thankful to Xavier Amatriain, former Director of Recommender Systems at Netflix, for his insightful blogs. I am also thankful to the GroupLens team for hosting the MovieLens 10M Dataset for educational and research purposes. I am thankful to my project advisors, Prof. Jimson Mathew, Prof. Arijit Mondal and Prof. Raju Halder, at Indian Institute of Technology Patna, for the motivation and moderation of my thoughts and ideas. Lastly, I'm grateful to the Google Cloud team for providing me with a $300 line of credit and access to their computational engines.

References

[1] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems (TiiS) 5.4 19:1-9:19*

[2] Francesco Ricci, Lior Rokach, Bracha Shapira and Paul B. Kantor. "Recommender Sys-

tems Handbook." *1st. New York, NY, USA: Springer-Verlag New York, Inc., 2010. ISBN 0387858199, 9780387858197*

[3] John S. Breese, David Heckerman and Carl Kadie. "Empirical Analysis of Predictive Algorithms for Collaborative Filtering." *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence. UAI'98. Madison, Wisconsin: Morgan Kaufmann Publishers Inc., 1998, pp. 43-52. ISBN: 1-55860-555-X*

[4] Simon Funk. "Netflix Update: Try This At Home"

[5] Ruslan Salakhutdinov, Andriy Mnih and Geoffrey Hinton. "Restricted Boltzmann Machines for Collaborative Filtering" *Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, 2007.*

[6] Ruslan Salakhutdinov and Geoffrey Hinton. "Reducing the Dimensionality of Data with Neural Networks" *Science Magazine Report, Volume 313, 28 July 2006.*

[7] Geoffrey Hinton, Simon Osindero and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets" *Journal of Neural Computation, Volume 18 Issue 7, July 2006*

[8] Yoshua Bengio, Pascal Lamblin, Dan Popovici and Hugo Larochelle. "Greedy Layer-Wise Training of Deep Networks" *Proceedings of the 19th International Conference on Neural Information Processing Systems, Canada, 2006*