Recommendation System Based on Collaborative Filtering

Zheng Wen

December 12, 2008

1 Introduction

Recommendation system is a specific type of *information filtering* technique that attempts to present information items (such as movies, music, web sites, news) that are likely of interest to the user. It is of great importance for the success of e-commerce and IT industry nowadays, and gradually gains popularity in various applications (e.g. Netflix project, Google news, Amazon). Intuitively, a recommendation system builds up a user's profile based on his/her past records, and compares it with some reference characteristics, and seeks to predict the 'rating' that a user would give to an item he/she had not yet evaluated. In most cases, the recommendation system corresponds to a large-scale *data mining* problem.

Based on the choice of reference characteristics, a recommendation system could be based on *content-based approach* or *collaborative filtering (CF) approach* (see [1]) or both. As their names indicate, content-based approach is based on the "matching" of user profile and some specific characteristics of an item (e.g.the occurrence of specific words in a document) while collaborative filtering approach is a process of filtering information or pattern based on the collaboration of users, or the similarity between items. In this project, we build a recommendation system based on multiple collaborative filtering (CF) approaches and their mixture, using part of Netflix project data as an example.

The remaining part of this report is organized as follows: in Section 2, we reformulate the Netflix project and use it to test the proposed algorithm in Section 3; in Section 3, we propose various CF algorithms to solve this problem; the experimental results are demonstrated in Section 4. We conclude the current results and propose future work at last.

2 Problem Formulation

We use the Netflix movie recommendation system as a specific example of recommendation system. Specifically, assume there are N_u users and N_m movies, given a set of training examples (i.e. a set of **triples** $(user, movie, rating)$), define the **user-movie matrix** $\mathbf{A} \in \Re^{N_m \times N_u}$ as:

$$A_{mu} = \begin{cases} R_{m,u}, \text{ User } u\text{'s rating on Movie } m & \text{if such rating exists} \\ \mathbf{?} & \text{if no such rating} \end{cases} \quad (1)$$

Our task is to predict all the users' ratings on all the movies based on the training set (i.e. existent ratings). In other words, we try to replace all the "question marks" in $\mathbf{A}$ by some *optimal* guesses. In this problem, each rating is an integer between 1 and 5, and the goal is to minimize the RMSE (root mean square error) when predicting the ratings on a test set (which is of course unknown

Implementation	Naive	Cinematch	Top in the current leaderboard of the contest
RMSE	around 1.25	0.9514	0.8604

Table 1: Performance of "Benchmarks"

during the training phase), that is, to minimize

$$rmse = \sqrt{\frac{1}{|S_{\text{test}}|} \sum_{(m,u)\in S_{\text{test}}} (R_{m,u} - P_{m,u})^2}, \tag{2}$$

where $(m,u) \in S_{\text{test}}$ if User u rates Movie m in the test set, $|S_{\text{test}}|$ is its cardinality, $R_{m,u}$ is the true rating and $P_{m,u}$ is the prediction based on the recommendation system.

Due to the recommendation system's importance in improving service quality, Netflix has started a contest with a grand prize to attract the researchers worldwide to work on this problem. Currently, many researchers, including many experts, are focusing on this problem and have made great progress. The performances of some important implementations can serve as the "measures" (or "benchmarks") for the quality of different algorithms, as shown in Table 1, where Naive implementation corresponds to using a constant rating (such as the average of all the available ratings), Cinematch is Netflix's original recommendation system (baseline).

Due to the limited computation power of PC and MATLAB, we only use part of the available data to build the recommendation system. Specifically, we use a data set include 20,000 users, and 1,500 movies.

3 Collaborative Filtering Algorithms

3.1 Item-Based K Nearest Neighbor (KNN) Algorithm

The first approach is the item-based K-nearest neighbor (KNN) algorithm. Its philosophy is as follows: in order to determine the rating of User u on Movie m, we can find other movies that are *similar* to Movie m, and based on User u's ratings on those similar movies we infer his rating on Movie m, see [2] for more detail. In order to determine which movies are "similar", we need to define a similarity function (similarity metric), as in [2], we use the **adjusted cosine similarity** between Movie a and b:

$$sim(a,b) = \frac{\sum_{u\in U(a)\bigcap U(b)} (R_{a,u} - \bar{R}_u)(R_{b,u} - \bar{R}_u)}{\sqrt{\sum_{u\in U(a)\bigcap U(b)} (R_{a,u} - \bar{R}_u)^2 \sum_{u\in U(a)\bigcap U(b)} (R_{b,u} - \bar{R}_u)^2}}, \tag{3}$$

where $R_{a,u}$ is User u's rating on Movie a, $\bar{R}_u$ is User u's average rating, $U(a)$ is the set of users that have rated Movie a and hence $U(a)\bigcap U(b)$ is the set of users that have rated both Movie a and b. The advantage of the above-defined adjusted cosine similarity over standard similarity is that the differences in the rating scale between different users are taken into consideration.

As its name indicates, KNN finds the nearest K neighbors of each movie under the above-defined similarity function, and use the weighted means to predict the rating. For example, the KNN algorithm for movies leads to the following formula:

$$P_{m,u} = \frac{\sum_{j\in N_u^K(m)} sim(m,j) R_{j,u}}{\sum_{j\in N_u^K(m)} |sim(m,j)|}, \tag{4}$$

where $N_u^K(m) = \{j : j \text{ belongs to the } K \text{ most similar movies to Movie } m \text{ and User } u \text{ has rated } j\}$, and $sim(m, j)$ is the adjusted cosine similarity defined in (3), $R_{j,u}$ are the existent ratings (of User u on Movie j) and $P_{m,u}$ is the prediction.

It should be pointed out that there also exists a user-based KNN algorithm. However, in most cases of the Netflix project, its performance is poorer than item-based KNN algorithm. This is because the user-based data appear to be sparser (i.e. it is very unlikely that a movie has only been rated by 1 or 2 users, and highly possible that a user only rates 1 or 2 movies). In addition, since typically there are much more users than movies, the user-based KNN also leads to a computational challenge.

3.2 Item-Based EM Algorithm

An alternative approach to solve this problem is Item-based EM algorithm. In this algorithm, we classify movies into G groups, and each Movie m belongs to Group g with probability $Q_m(g)$. For a given Group g of movies, we assume the ratings of different users are independent and Gaussian, that is, $P(R_{u,m}|m \in g) \sim N(\mu_{g,u}, \sigma_{g,u}^2)$. The conditional independence is known as **Naive Bayes** assumption in machine learning literature and is widely used in many different applications. In this project, intuitively, if we know Movie 1 is a horror movie, then whether User 1 likes this movie should be independent of whether User 2 likes this movie, hence their ratings on this movie are conditionally independent. The Gaussian assumption is assumed to simplify the calculation in the M-step. The unboundedness issue related to the Gaussian assumption can be solved by truncation.

To formulate the EM algorithm formula in this case, let latent random variable $G_m \sim Q_m(\cdot)$ denotes the group of Movie m. Define $U(m)$ as the set of users that have rated Movie m. Thus, the E-step formula is described as follows:

$$\begin{aligned} Q_m^{(t+1)}(g) &= P(G_m^{(t+1)} = g | R_{u,m}; \mu_{g,u}, \sigma_{g,u}^2) \\ &= \frac{Q_m^{(t)}(g) \Pi_{u \in U(m)} \frac{1}{\sqrt{2\pi}\sigma_{g,u}} exp\left(-\frac{(R_{u,m}-\mu_{g,u})^2}{2\sigma_{g,u}^2}\right)}{\sum_{g'} Q_m^{(t)}(g') \Pi_{u \in U(m)} \frac{1}{\sqrt{2\pi}\sigma_{g',u}} exp\left(-\frac{(R_{u,m}-\mu_{g',u})^2}{2\sigma_{g',u}^2}\right)}, \end{aligned} \tag{5}$$

where superscripts (t) and $(t+1)$ are used to denote distributions of latent random variables in different iterations. For M-step, we need to solve

$$\max_{\mu_{g,u}, \sigma_{g,u}^2} \quad \sum_m \sum_g Q_m(g) \left[log \left\{ \Pi_{u \in U(m)} \frac{1}{\sqrt{2\pi}\sigma_{g,u}} exp\left(-\frac{(R_{u,m}-\mu_{g,u})^2}{2\sigma_{g,u}^2}\right) \right\} - log\left(Q_m(g)\right) \right],$$

which is equivalent to

$$\max_{\mu_{g,u}, \sigma_{g,u}^2} \quad \sum_{m,g} Q_m(g) \sum_{u \in U(m)} \left[-log(\sigma_{g,u}) - \frac{(R_{u,m}-\mu_{g,u})^2}{2\sigma_{g,u}^2} \right].$$

Define $M(u)$ as the set of movies User u has rated, by changing the order of summation and a little bit of algebra, we have

$$\mu_{g,u} = \frac{\sum_{m \in M(u)} Q_m(g) R_{u,m}}{\sum_{m \in M(u)} Q_m(g)}$$

$$\sigma_{g,u}^2 = \frac{\sum_{m \in M(u)} Q_m(g) \left(R_{u,m} - \mu_{g,u}\right)^2}{\sum_{m \in M(u)} Q_m(g)} \tag{6}$$

We repeat E-step and M-step until convergence. And the final prediction is given by

$$P_{u,m} = \sum_g Q_m(g) \mu_{g,u}.$$

Similarly, there also exists a user-based EM algorithm. Due to the reasons discussed in the previous subsection about the advantage of item-based KNN over user-based KNN, for this project, the item-based EM will work better than user-based EM in most cases.

3.3 Sparse SVD Algorithm

Another algorithm to solve this problem is based upon **sparse matrix SVD**. This approach models both users and movies by giving them coordinates in a low dimensional feature space i.e. each user and each movie has a *feature vector*. And each rating (known or unknown) is modeled as the inner product of the corresponding user and movie feature vectors. In other words, we assume there exist a small number of (unknown) factors that determine (or dominate) ratings, and try to determine the values (instead of their meanings) of these factors based on training data. Mathematically, based on the training data (sparse data of a huge matrix), we try to find a low-rank approximation of the user-movie matrix A. This approach is called sparse SVD algorithm.

Let $\mathbf{u_i} \in \Re^{N_f}$, $i = 1, 2, \cdots, N_u$ be all the users' feature vectors, and $\mathbf{m_j} \in \Re^{N_f}$, $j = 1, 2, \cdots, N_m$ be all the movies' feature vectors. This problem can be formulated as the following optimization problem:

$$\min_{\mathbf{u_i}, \mathbf{m_j}} \sum_{(i,j) \in I} (R_{i,j} - \mathbf{u_i}^T \mathbf{m_j})^2 + \lambda \left(\sum_i n_{u_i} \|\mathbf{u_i}\|^2 + \sum_j n_{m_j} \|\mathbf{m_j}\|^2 \right), \tag{7}$$

where $I = \{(i,j) | \text{if user i has rated movie } j\}$, n_{u_i} is the number of movies user i has rated, and n_{m_j} is the number of users that have rated movie j. We notice the weighted L_2 regularization term is added to avoid potential overfitting.

One should notice that to solve (7) directly is nontrivial. As has been proposed in [3], this problem can be solved as follows: first, we fix the users' feature vectors $\mathbf{u_i}$, and solve for the movies' feature vectors $\mathbf{m_j}$; then we fix movies' feature vectors $\mathbf{m_j}$, and solve for users' feature vectors $\mathbf{u_i}$. We repeat this process until convergence. Notice in each step, we are solving a **regularized least square** problem, for which efficient algorithms exist and are handy to use.

3.4 Tricks in Postprocessing

In addition to the collaborative filtering algorithms discussed in the previous subsections, multiple "tricks" in data postprocessing could also be applied simultaneously to enhance the performance of the recommendation system. Some of the "tricks" we have used in our implementation include *newcomer prediction*, *prediction truncation*, *item-based correction* and *near-integer round-off*.

Newcomer Prediction

When there exists a *newcomer*, i.e. a user without any existent ratings, it is very difficult to predict his/her rating on any item. In order to minimize the RMSE in this case, we use the *item mean* as his/her prediction. That is, the prediction of User u's rating on Movie m is the average value of existent ratings on Movie m given by other users,

$$P_{m,u} = \frac{1}{|U(m)|} \sum_{u' \in U(m)} R_{m,u'},$$

as defined above, $U(m)$ is the set of users that have rated Movie m.

Prediction Truncation

For some algorithms such as sparse SVD, it is possible that the prediction $P_{m,u}$ is above 5 or below 1. In this case, we "truncate" the prediction. That is, we set $P_{m,u} = 5$ if $P_{m,u} > 5$ and $P_{m,u} = 1$ if $P_{m,u} < 1$. Of course, prediction truncation will strictly improve the performance of the recommendation system and reduce the RMSE.

Item-Based Correction

The third and most important "trick" is item-base correction. Specifically, define the **item-based rating mean** as

$$S_m = \frac{1}{|U(m)|} \sum_{u' \in U(m)} R_{m,u'},$$

where $U(m)$ is defined above. For a given collaborative filtering algorithm, define the **item-based prediction mean** as

$$\tilde{S}_m = \frac{1}{N_u} \sum_{u'} P_{m,u'},$$

where N_u is the number of users. We correct the prediction by $\Delta_m = S_m - \tilde{S}_m$. That is

$$P_{m,u} := P_{m,u} + \Delta_m. \tag{8}$$

The above equation gives the formula for item-based correction. Based on our experiment, the item-based correction improves the RMSE by about 0.01.

Near-Integer Round-off

For Netflix project, one interesting question is that whether *rounding off* the prediction will reduce the RMSE. While reducing the *errors* of some predictions to 0, this approach will also increase the errors of other predictions. In practice, it is observed that in most cases the naive round-off will increase the RMSE.

In this project, we use an approach called near-integer round-off. Specifically, we round off the prediction if it is close enough to an integer. During the implementation, we round off the prediction if its distance to the nearest integer is less than or equal to 0.1. It has been observed that this approach can improve the performance of the recommendation system.

Sequence of Implementation

In our implementation, given predictions as outcomes of some collaborative filtering algorithms, we further enhance the performance by first applying the newcomer prediction algorithm, then carry out the item-based correction, then prediction truncation, and at last near-integer round-off. It has been observed that with this sequence of implementation of postprocessing tricks, the RMSE of the recommendation system can be reduced by approximate 0.02.

3.5 Parameter Adjustment and Algorithm Mixture

As is classical in machine learning and collaborative filtering, we need to adjust the parameters in order to achieve the optimal performance of the recommendation system. For example, in the item-based KNN algorithm, we need to adjust the number of neighbors (i.e. K); in the item-based EM algorithm, we need to adjust the number of groups of movies; in the sparse SVD algorithm, we need to adjust the dimension of the feature vectors and the regularization weight. All the parameter adjustment is carried out through **K-fold cross-validation**.

It is also noticed that as expected, "blending" the algorithms could improve the performance. For "blending", we mean that we use the *convex combination* of the predictions from different collaborative filtering algorithms as the final predictions. For example, assume $P_{m,u}^{KNN}$ is the prediction from the item-based KNN algorithm, and $P_{m,u}^{SVD}$ is the prediction from the sparse SVD algorithm, then the final prediction is

$$P_{m,u}^{final} = \lambda P_{m,u}^{SVD} + (1-\lambda) P_{m,u}^{KNN}, \tag{9}$$

where $\lambda \in (0,1)$ is the weight of sparse SVD algorithm in the "blending".

3.6 Our New Contribution

It should be noticed that most of the algorithms used in this project have been presented in previous literatures (e.g. [2, 3]) in this field. To the best of our knowledge, our new contributions include:

1. We use the item-based correction and near-integer roundoff in both sparse SVD and EM algorithm.
2. We change the EM algorithm from user-based to item-based.

4 Experimental Results

We carry out the algorithms proposed in Section 3 on part of Netflix data with $20,000$ users and $1,500$ movies. For each CF algorithm and their mixture, we use the postprocessing "tricks" presented above to enhance the performance and use K-fold cross-validation to choose the best parameters (models). In particular, we choose $K = 10$ in our implementation. The results are listed as follows:

4.1 Item-Based KNN Algorithm

For the item-based KNN algorithm, we use the approach described in Subsection 3.1. The cross-validation RMSE is shown in Figure 1. From the cross validation result, we notice the optimal number of nearest neighbors is $K = 600$. With this particular choice of K, the RMSE of test data is **0.9508**.

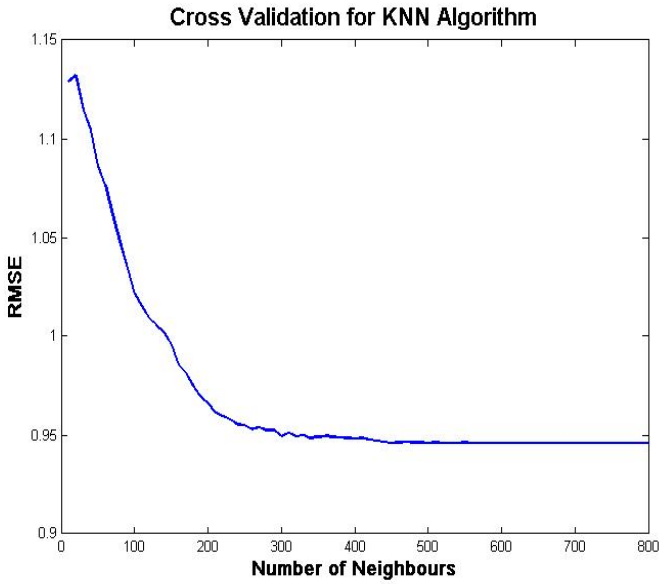

Figure 1: Cross Validation for Item-Based KNN Algorithm

Although the performance of the item-based KNN algorithm is NOT as good as alternative algorithms, such as the sparse SVD and item-based EM algorithm, it does reach a performance similar to Cinematch. In addition, the prediction results of the KNN algorithm can be used to initialize other algorithms, which will usually lead to a much better performance and faster convergence than random initialization.

4.2 Item-Based EM Algorithm

For the item-based EM algorithm, we apply the algorithm described in Subsection 3.2 and use the cross validation approach to choose the optimal number of groups of movies. The cross validation indicates the optimal number of groups is 30, and the resulting RMSE for test data is **0.9140**.

During the experiment, we notice that item-based EM algorithm has a better performance than user-based EM algorithm. However, its performance is poorer than the sparse SVD algorithm.

4.3 Sparse SVD Algorithm

For the sparse SVD algorithm, we implement the algorithm as described in Subsection 3.3. The cross-validation RMSE for different feature vector dimensions N_f and L_2 regularization weights λ is shown in Figure 2,3 and 4. From the cross validation, we notice the optimal feature space dimension is $N_f = 150$ and the associated optimal weight is $\lambda = 0.106$. With these parameters, the RMSE on test data is **0.89674**.

We notice that sparse SVD has a better performance than item-based KNN algorithm and item-based EM algorithm. This is primarily due to: (1) the sparse SVD simultaneously treat both user and item features, while other algorithms always concentrate on one aspect (hence can be classified as "user-based" or "item-based"); (2) L_2 regularization is implemented to avoid overfitting.

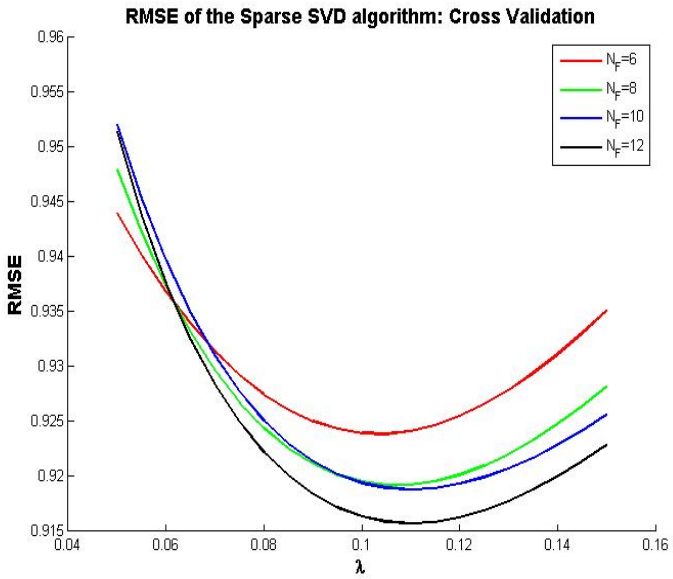

Figure 2: Cross Validation for Sparse SVD Algorithm: $N_F = 6, 8, 10, 12$

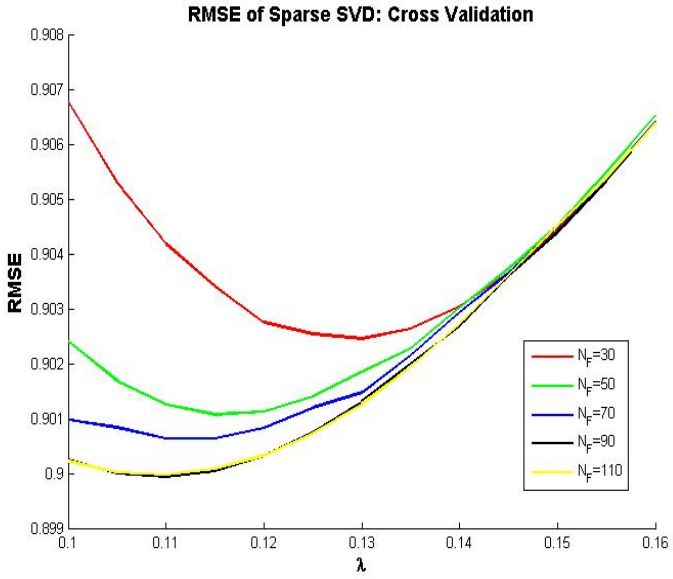

Figure 3: Cross Validation for Sparse SVD Algorithm: $N_F = 30, 50, 70, 90, 110$

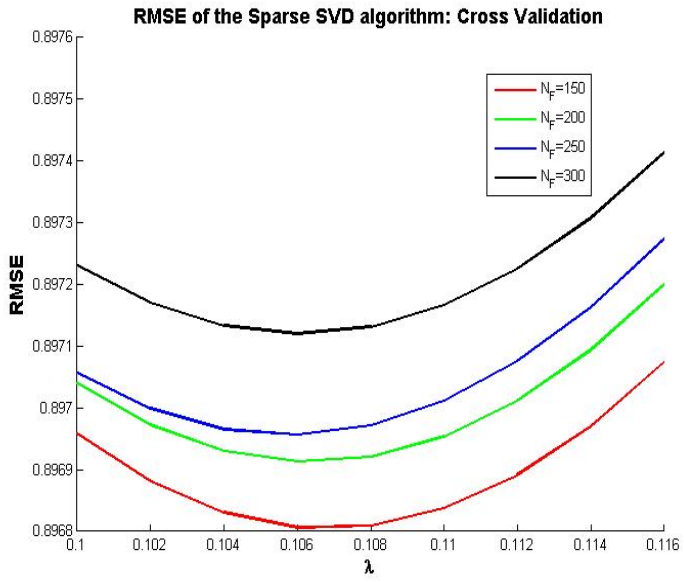

Figure 4: Cross Validation for Sparse SVD Algorithm:$N_F = 150, 200, 250, 300$

Algorithm	Item-Based KNN	Item-Based EM	Sparse SVD	Blending of Item-Based EM and Sparse SVD
RMSE	0.9508	0.9140	0.89674	0.8930

Table 2: Performance of Different Collaborative Filtering Algorithms

4.4 Blend Item-Based KNN and Sparse SVD

At last, we "blend" the predictions of item-based KNN and sparse SVD, as described in Equation (9). We use the cross validation to choose the optimal weight λ, and the cross-validation RMSE is shown in Figure 5. From the cross validation, the optimal "blending" weight is $\lambda = 0.78$. With this optimal weight, the RMSE for test data is **0.8930**.

In summary, the performance of different algorithms are illustrated in Table 2.

5 Conclusion and Future Work

In this project, we present several collaborative filtering algorithms for recommendation system and test the performance of each algorithm and their mixtures on part of the Netflix data. At last, a RMSE of 0.8930 is achieved, which corresponds to a 6.14% improvement of the performance of Cinematch (baseline).

As to the future work, we plan to

1. Try other important algorithms on this project, such as Bayes Network;
2. Work more on the algorithm "blending". Since different algorithms might characterize different aspects of the problem, and cooperation between multiple algorithms should lead to a better performance. Of course, model selection approaches such as cross validation will be widely used in this scenario.

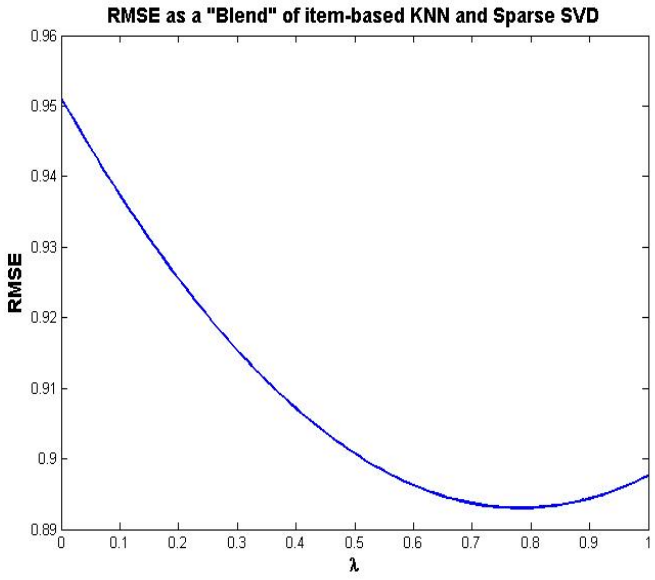

Figure 5: Cross Validation for "Blending" Weight of Item-Based KNN and Sparse SVD

Acknowledgment

We are grateful for Prof. Ng and TAs' help during this course and project. We also thank Mr. Robbie Yan for helping us get started on this project.

References

[1] J. Breese, D. Heckerman and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering", *Technical Report of Microsoft Research, 1998.*

[2] B. Sarwar, G. Karypis, J. Konstan and John Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms", *Proceedings of the 10th international conference on World Wide Web 2001: 285-295.*

[3] Y. Zhou, D. Wilkinson, R. Schreiber, R. Pan. "Large-Scale Parallel Collaborative Filtering for the Netflix Prize", *AAIM 2008: 337-348.*

[4] Andrew Ng, CS229 Lecture Notes.