Movie Rating Prediction Using Singular Value Decomposition

CMPS 242, Machine Learning Project Report

Serdar Sali
University of California, Santa Cruz
sali@soe.ucsc.edu

March 21, 2008

Abstract

In this paper, we explore the application of SVD for collaborative filtering. We employ the incremental SVD method for predicting movie ratings based on previous user preferences using the dataset provided by Netflix. Various experiments are performed to see the effect of different parameters on the performance of the algorithm. The results show that the method has potential, although it is prone to overfitting.

I. Introduction

Collaborative filtering is the method of making predictions about the interests of a user based on previous preferences by all users based on the idea that users who liked similar things in the past will tend to favor similar items in the future [4]. There has been extensive work in the literature on this problem, and currently Netflix runs a competition for collaborative filtering for movies. The competition sets as aim an increase of 10% in performance over its current algorithm. Singular Value Decomposition (SVD) has been shown to perform well for this goal by various teams in the competition, and almost all groups currently in top places in the leader board employ a form of SVD.

This paper proposes a method that employs the incremental SVD method proposed by Simon Funk [1]. In Section 2, we show how to formulate our problem as an SVD problem. The incremental SVD method is detailed in Section 3, the results of experiments are given in Section 4, and section 5 concludes the paper.

II. Problem Formulation

For our problem, ideally, we would have a set of features for each movie in our dataset, and a set of weights for each user indicating how likely a user is to enjoy a movie with such users. The task of predicting a rating then becomes simply multiplying the user preference vector with the movie feature vector.

However, it is extremely difficult to collect such data. First of all, it is by itself a difficult task to compute & rate features for each movie due to the subjective nature of the task. Second, this would require retrieving information from external resources and combining it with the user-movie rating, and this data would require tremendous cleaning-up effort. Notice, however, the features and the user preferences for these features are already embedded in the user-movie-rating triplets, it is just lumped together as a weighted sum. We just need a method to retrieve those features and related preference vectors. This is where Singular Value Decomposition (SVD) comes in.

Singular Value Decomposition states that every matrix A E $_{\mathrm{Rmxn}}$ can be decomposed as $A = USV^T$, where U and V are orthogonal and S is diagonal with singular values of A on the diagonal. In full singular value decomposition, U is mxm, S is mxn, and V is mxn. Since S is a diagonal matrix, we can remove rows from S and columns from U to obtain a more compact representation where U is mxn, S is nxn, and V is nxn. This is called the reduced singular value decomposition (Fig. 1).

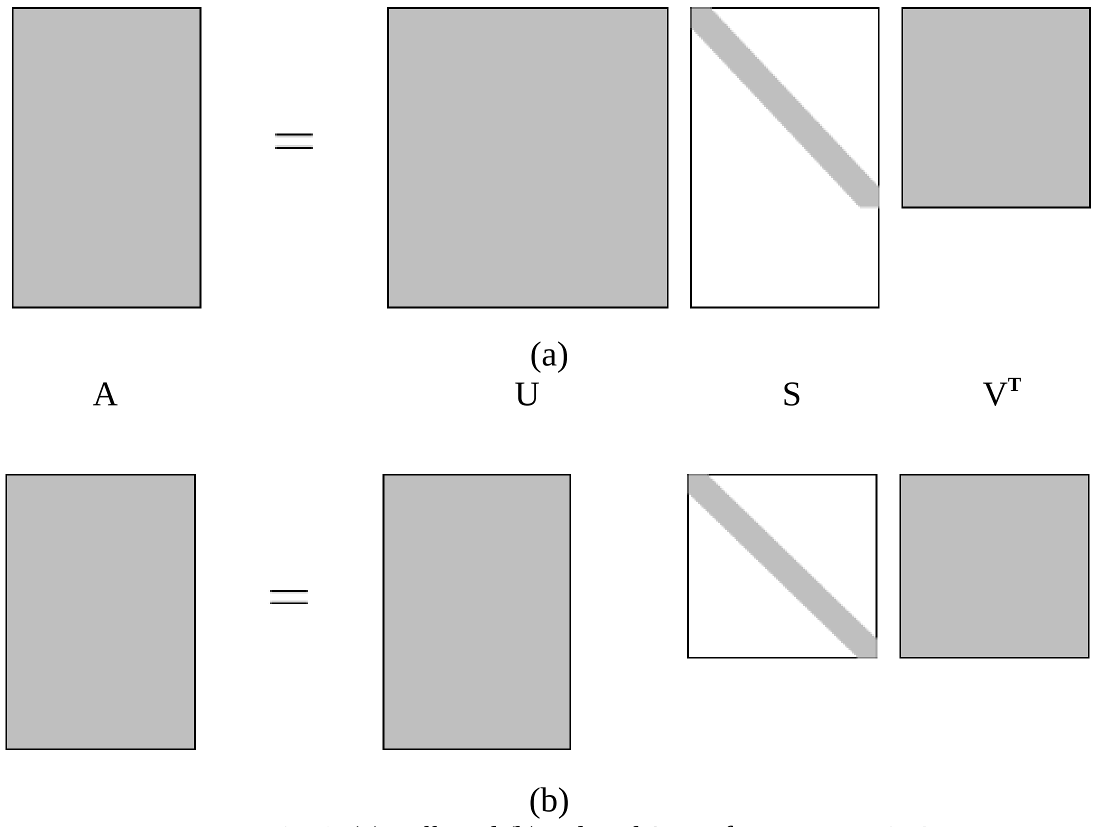


Fig. 1. (a) Full, and (b) reduced SVD of an mxn matrix A

SVD has many applications, but of particular interest to us is the application of the SVD method to find a rank-r approximation to a given matrix. This problem can simply be stated as follows: Given a matrix R, compute a rank-r approximation R_{app} to this matrix such that the Frobenius norm of $R - R_{app}$ is minimized. Frobenius form ($||R - R_{app}||_F$) is defined as simply the sum of squares of elements in $R - R_{app}$. We can achieve such an approximation by only considering the first r most significant singular values in the singular value decomposition of R.

Returning to our domain, we can formulate our problem as follows: Suppose we have an uxm matrix R which contains the actual ratings by the users, where u is the number of users and m is the number of movies. Assume that we want to consider f features, regarding the rest as insignificant. We want to compute an approximation R_{app} to this matrix R, such that $||R\text{-}R_{app}||_F$ is minimized, and $R_{app} = P_{uxf}(F_{mxf})^T$. Notice that the i^{th} row of P vector is the preference vector for user i, and the k^{th} row of F is the feature vector for movie k. Therefore we have extracted the an approximation to the desired data, which we can then use to fill unknown entries of R by computing the dot product of user preference and movie feature vectors.

III. Incremental SVD Method

Unfortunately, serious problems arise in applying the SVD method to our domain: Whatever dataset is used, R is highly sparse, and traditional SVD does not work for sparse matrices. Various methods to overcome this issue have been proposed, such as filling in the unknown entries with normalized averages, or starting out with averages and then improving the unknown entries by using the computed features during the iteration of the algorithm. A second problem is that R is typically huge, and computing the SVD of huge matrices requires too much computation power.

The incremental SVD method proposed by Simon Funk seems to perform really well and it is also easy to compute, addressing both of those problems. It is simply a gradient descent algorithm to compute the approximation using only the known entries of R. Notice that the error in a prediction for a user i for movie j is simply $(R - R_{app})_{ij}$. To calculate the gradient, we take the derivative of the square of this with respect to p_{ik} and then f_{jk} , and since R is a constant, and R_{app} is a function of p_{ik} and f_{jk} in just one factor, the updates for p and f then become:

$$p_{ik}(t+1) = p_{ik}(t) + learning\_rate*(R-R_{app})_{ij}*f_{jk}(t)$$
$$f_{jk}(t+1) = f_{jk}(t) + learning\_rate*(R-R_{app})_{ij}*p_{ik}(t)$$

This method seems to work really good, and almost all teams in the competition use this method in one form or another. There are of course various modifications and adjustments to make this perform slightly better, including regularization, using different functions for rating prediction instead of just taking a dot product of the preference and feature vector, doing the rounding in a clever method, etc. Although it is a little slow to converge as with all gradient descent algorithms, the method performs really good even in its raw form. Various experiments have shown this method to be competitive with Restricted Boltzmann Machines and neighborhood-based estimation methods [2, 3, 5, 6].

IV. Experiments

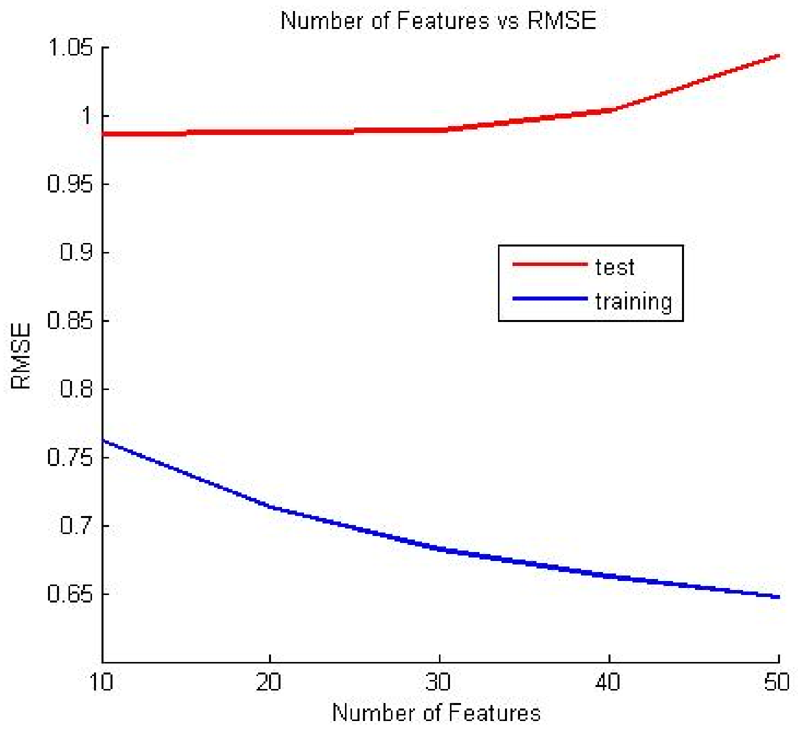
The method proposed above has many free parameters. Among those, I experimented with the number of features used, the learning rate, and the regularization parameter. I also tried two different models using different methods for calculating the base prediction. RMSE is used as the performance measure.

The preference multipliers for all users were set to 0.1 initially. A feature is trained until the improvement in RMSE is smaller than 0.0001, or at least 120, at most 200 passes are made over the training data.

Since the dataset is really large, all tests were performed on a smaller dataset consisting of ratings for 2000 movies by 480189 users. There are a total of 10314269 ratings in this smaller dataset, making it about 10% in size of the original dataset. A single run using all the data was also done to show that the method performs better than Netflix's current architecture Cinematch.

IV.1. Number of Features vs RMSE

For this experiment, we use a constant learning rate of 0.015. We also use the average rating for the movie as the base prediction. Any feature added after the 10th results in an overfit.

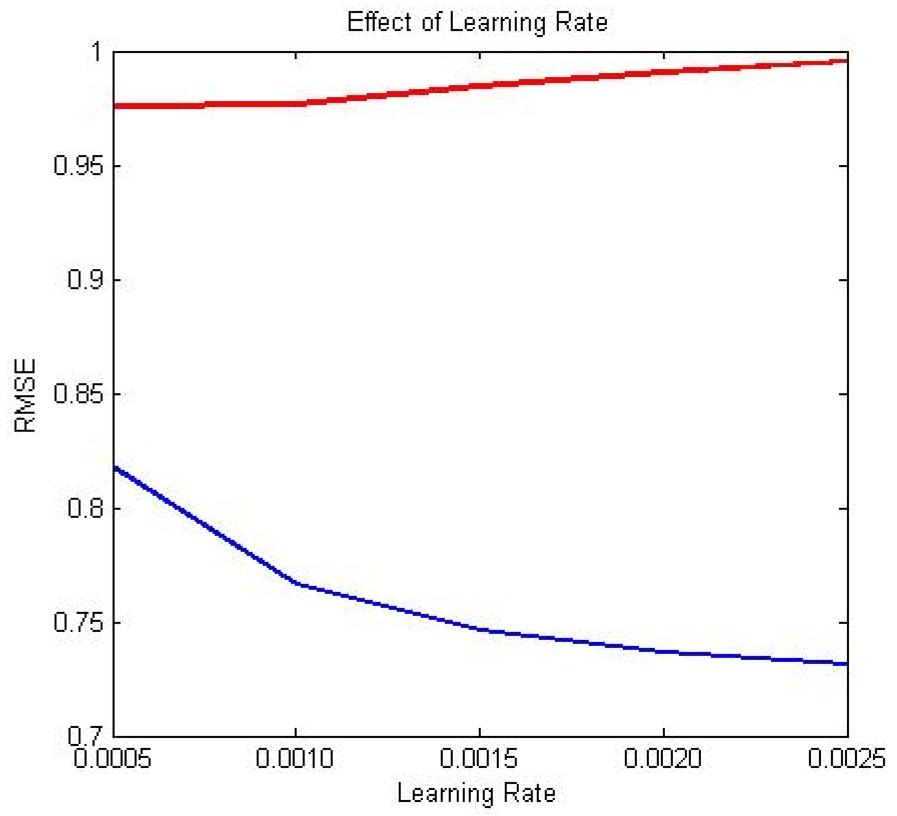


	10 features	20 features	30 features	40 features	50 features
Training	0.7623	0.7128	0.6820	0.6619	0.6468
Test data	0.9858	0.9869	0.9882	1.0028	1.0438

Fig. 2. RMSE values vs. number of features

IV.2. Learning Rate

For this experiment, the learning rate was assigned values 0.0005, 0.0010, 0.0015, 0.0020 and 0.0025 respectively. All experiments were performed using 10 features, and a fixed value of 0.015 for the regularization parameter. Fig. 3 shows the RMSE values for the test and training data.



	L = 0.0005	L = 0.0010	L = 0.0015	L = 0.0020	L = 0.0025
Training	0.8180	0.7666	0.7464	0.7365	0.7314
Test	0.9755	0.9765	0.9843	0.9904	0.9955

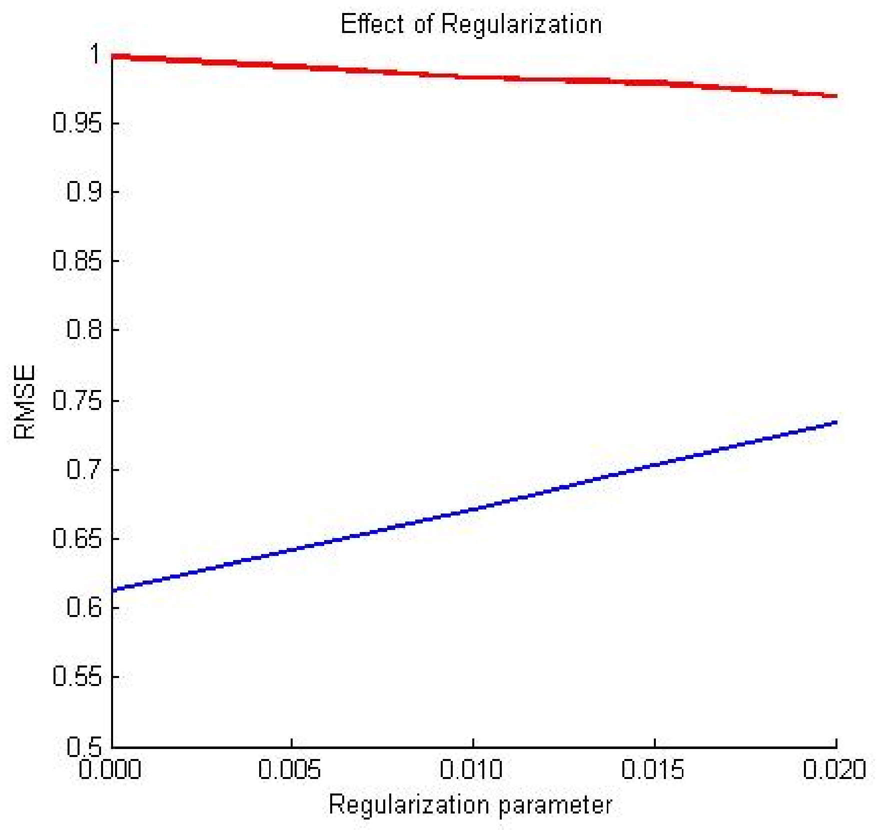
Fig. 3. RMSE values vs. different learning rates

As we can clearly see in the plot, increasing learning rate causes overfitting even if we employ heavy regularization. The same general trend also applies here: The models overfit the training data very easily when the training set is small.

Again, a blind averaging of the results give an RMSE of 0.9756, which is surprisingly close to the best performing model. Again, this suggests that errors made by different models tend to cancel each other.

IV.3. Effect of Regularization

For this experiment, the regularization parameter was assigned values 0 (corresponding to no regularization), 0.005, 0.010, 0.015 and 0.02 respectively. All experiments were performed using 25 features. The learning rate was set to 0.001. The following plot shows the results.



	K = 0	K = 0.005	K = 0.010	K = 0.015	K = 0.020
Training	0.6118	0.6418	0.6705	0.7027	0.7333
Test data	0.9973	0.9899	0.9823	0.9787	0.9686

Fig. 4. RMSE values vs. different regularization parameters

The results clearly reveal that a low regularization value results in overfitting the training data, and RMSE goes down as more regularization is enforced.

A blind averaging of results gives an RMSE of 0.9741, which again shows that the methods indeed compensate for each other's mistakes.

IV.4. Different Initialization Methods

Another interesting parameter is how we make the base prediction for a movie. For this experiments, three different models were compared: using a base guess of 1 for all movies, using the average rating for the movie by all users, and using the overall movie average + average user offset. The following table gives the results for 0.001 learning rate and 0.015 as the regularization parameter with 10 features.

	Base = 1	Base = Average	Base = Average + Offset
Training	0.7651	0.7638	0.7638
Test data	1.0144	1.0156	1.0158

Using the averages of the results of three methods in this case performs better than all three models with an RMSE of 0.9899.

Combining Best Results

Although intuitively combining the results of the best models will not necessarily result in the optimum combination, simply taking the average of predictions of the best models for each case yields an RMSE of 0.9673, which is the best result obtained among all results. A more clever way of combining the results, as well as using results from more methods is likely to result in better performance.

Performance on the whole dataset

A single run of the algorithm takes about 20 hours using the original dataset. Using the method with a learning rate of 0.001, a regularization parameter of 0.02, and using averages as the base predictions, the algorithm achieved a 0.9261 RMSE, which is about 3% better than Cinematch.

V. Conclusions & Future Work

The experiments performed reflect the competitive nature of the SVD method for the problem. Despite its sheer simplicity, the incremental SVD method performs very good when compared to Netflix's own architecture Cinematch despite the limited data availability. Furthermore, it is computationally feasible with respect to the sheer size of the task at hand.

The main drawback of the method is that there is an abundant number of free parameters which makes finding a good set of parameters that will work good for all cases is next to impossible. Although this is typical of machine learning problems, especially in this case, the experiments showed that even a single blind combination of results from very similar models gives performance almost close to the best model, which suggests that different models work good for different cases. Therefore, the only way to achieve the goal set by Netflix seems to use combinations of different models. Not coincidentally, this is what the top contenders in the competition currently do.

Experiments have clearly shown that overfitting is a serious problem. Different regularization methods may help overcome this problem.

Experimenting with entirely different algorithms and combining results, which remains as future work, seems to be the best approach to take. Using the features computed by the incremental SVD method as input to other algorithms might also prove useful. Also, more clever ways of combining the results are likely to improve the results.

References

[1] Incremental SVD method, Simon Funk.
http://sifter.org/~simon/journal/20061211.html

[2] Bell, R., Koren, Y., Volinsky, C. BellKor solution to the Netflix Prize,
http://www.research.att.com/~volinsky/netflix/

[3] Salakhutdinov, R., Mnih, A., and Hinton, G. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international Conference on Machine Learning* (Corvalis, Oregon, June 20 - 24, 2007). Z. Ghahramani, Ed. ICML'07, vol. 227. ACM, New York, NY, 791-798.

[4] Collaborative Filtering: A Machine Learning Perspective, Benjamin Marlin, MS Thesis, Department of Computer Science, University of Toronto.

[5] Bell, R. Koren, Y. Improved neighborhood-based collaborative filtering. In *Proceedings of the KDDCup 2007.* August 12, 2007, San Jose, California. ACM.

[6] Bell, R. Koren, Y. and Volinsky, C. Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems, *Proceedings of the 13th ACM Int. Conference on Knowledge Discovery and Data Mining (KDD'07)*, ACM press, 2007.