Improving regularized singular value decomposition for collaborative filtering

Arkadiusz Paterek
Institute of Informatics, Warsaw University
ul. Banacha 2, 02-097 Warsaw, Poland
paterek@mimuw.edu.pl

ABSTRACT

A key part of a recommender system is a collaborative filtering algorithm predicting users' preferences for items. In this paper we describe different efficient collaborative filtering techniques and a framework for combining them to obtain a good prediction.

The methods described in this paper are the most important parts of a solution predicting users' preferences for movies with error rate 7.04% better on the Netflix Prize dataset than the reference algorithm Netflix Cinematch.

The set of predictors used includes algorithms suggested by Netflix Prize contestants: regularized singular value decomposition of data with missing values, K-means, postprocessing SVD with KNN. We propose extending the set of predictors with the following methods: addition of biases to the regularized SVD, postprocessing SVD with kernel ridge regression, using a separate linear model for each movie, and using methods similar to the regularized SVD, but with fewer parameters.

All predictors and selected 2-way interactions between them are combined using linear regression on a holdout set.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning;
H.3.3 [**Information storage and retrieval**]: Information search and retrieval—*Information filtering*

General Terms

Algorithms, Experimentation, Performance

Keywords

prediction, collaborative filtering, recommender systems, Netflix Prize

1. INTRODUCTION

Recommender systems are very important for e-commerce. If a company offers many products to many clients, it can benefit substantially from presenting personalized recommendations. For example, Greg Linden, developer of Amazon's recommendation engine [6], reported that in 2002 over 20% of Amazon's sales resulted from personalized recommendations. There exist many commercial applications of recommender systems for products like books, movies, music and others. Also many applications not directly commercial have emerged: personalized recommendations for websites, jokes [4], Wikipedia articles, etc.

A difficult part of building a recommender system is, knowing preferences of users for some items, to accurately predict which other items they will like. This task is called collaborative filtering. Most approaches to this task, described so far in the literature, are variations of K-nearest neighbors (like TiVo [1]) or singular value decomposition (like EigenTaste [4]). Another approach is using graphical models [7, 8]. Articles [3, 7] are examples of comparisons of different collaborative filtering techniques.

In October 2006, the contest Netflix Prize was announced. The goal of the contest is to produce a good prediction of users' preferences for movies. Netflix released a database of over 100 million movie ratings made by 480189 users. The contest ends when someone submits a solution with prediction error RMSE (root mean squared error) 10% better than the Netflix Cinematch algorithm. For introduction to the Netflix Prize competition and description of Netflix Cinematch we direct the reader to the article [2].

This paper describes various collaborative filtering algorithms that work well on the Netflix Prize dataset. Using approach of combining results of many methods with linear regression we obtained 7.04% better RMSE than Netflix Cinematch on the Netflix Prize competition evaluation set.

In section 2 we describe our framework for combining predictions with linear regression. Most effective predictors from our ensemble are described in section 3, including approaches proposed by Netflix Prize contestants: regularized SVD of data with missing values, K-means, postprocessing results of regularized SVD with K-NN. In that section we describe also new (to our knowledge) approaches: regularized SVD with biases, postprocessing results of SVD with kernel ridge regression, building a separate linear model for each movie, and using two methods inspired by regularized SVD, but with lower number of parameters. In section 4 experimental results are presented, which show that combining the proposed predictors leads to a significantly better prediction than using pure regularized SVD. In section 5 we summarize our experiments and discuss possible further improvements.

2. COMBINING PREDICTORS

In this section we describe how in the proposed solution the training and the test set are chosen and how different prediction methods are combined with linear regression.

The Netflix Prize data consists of three files:

- training.txt contains $R = 100,480,507$ ratings on a scale 1 to 5, for $M = 17,770$ movies, made by $N = 480,189$ customers,
- probe.txt contains $1,408,395$ user-movie pairs, for which the ratings are provided in training.txt,
- qualifying.txt contains $2,817,131$ user-movie pairs, for which we do not know the ratings, but RMSE of a prediction is computed by the Netflix Prize evaluation system. We can assume probe.txt and qualifying.txt come from the same population of newest ratings.

Summarizing, the user-item matrix for this data has $N * M = 8,532,958,530$ elements – c.a. 98.9% values are missing.

The dataset besides ratings contains other information, like the dates of the ratings, but we do not use for prediction any information besides the above mentioned rating data.

Our framework for combining predictions is simple: we draw random $1.5\% - 15\%$ of probe.txt as a test set (hold-out set). Our training set contains the remaining ratings from training.txt. We train all algorithms on the training set (some methods also occasionally observe test set error to make a decision when to stop optimization of weights). Then the predictions made by each algorithm for the test set are combined with linear regression on the test set. Adding to the regression selected two-way interactions between predictors gives a small improvement.

There is also a possibility of using data without ratings (qualifying.txt), which carry some information. The article [8] suggests that using this additional data can significantly improve prediction in the Netflix Prize task.

Because linear regression is made on a small set, the weights obtained are inaccurate. Also, using the test set for linear regression, feature selection and other purposes causes small overfitting. We can improve prediction using a cross-validation-like method: draw randomly a part of probe.txt as the test set, repeat the training and linear regression, do this a few times and average the results. However, each repetition means running again all algorithms on a massive dataset. Because training each one of our algorithms takes much time (0.5-20h), we did not perform cross-validation in experiments described in section 4.

The 7.04% submission to the Netflix Prize is a result of partial cross-validation. We ran part of our methods for the second time on a different test set and confirmed an improvement after merging results of two linear regressions.

In the next sections we describe the most effective predictors from our ensemble.

3. PREDICTORS

3.1 Simple predictors

In this section we describe six predictors which are used by methods from subsections 3.2 (RSVD) and 3.5 (SVD_KNN) and also in all experiments described in section 4.

For the given movie j rated by user i, first five predictors are empirical probabilities of each rating $1 - 5$ for user i. The sixth predictor is the mean rating of movie j, after subtracting the mean rating of each member.

We will refer to that set of six simple predictors as "BASIC".

3.2 Regularized SVD

Regularized SVD, a technique inspired by effective methods from the domain of natural language processing [5], was proposed for collaborative filtering by Simon Funk (Brandyn Webb) [9]. Simon Funk's description [9] includes proposition of learning rate and regularization constants, and a method of clipping predictions.

In the regularized SVD predictions for user i and movie j are made in the following way:

$$\hat{y}_{ij} = u_i^{\mathrm{T}} v_j \tag{1}$$

where u_i and v_j are K-dimensional vectors of parameters. The layer of k-th parameters of all vectors u_i, v_j is called the k-th feature.

Parameters are estimated by minimizing the sum of squared residuals, one feature at a time, using gradient descent with regularization and early stopping. Before training, from each rating a simple baseline prediction is subtracted – combination of six predictors described in section 2.1, with weights chosen with linear regression.

$$r_{ij} = y_{ij} - \hat{y}_{ij}$$

$$u_{ik} \mathrel{+}= lrate * (r_{ij} v_{jk} - \lambda u_{ik})$$

$$v_{jk} \mathrel{+}= lrate * (r_{ij} u_{ik} - \lambda v_{jk})$$

, where y_{ij} is the rating given by user i for movie j.

We stop training the feature when the error rate on the test set increases. After learning of each feature, the predictions are clipped to $< 1, 5 >$ range.

Parameters proposed by Simon Funk are difficult to improve, so we leave them unchanged: $lrate = .001$, $\lambda = .02$. We choose the number of features $K = 96$.

We will refer to this method as "RSVD".

3.3 Improved regularized SVD

We add biases to the regularized SVD model, one parameter c_i for each user and one d_j for each movie:

$$\hat{y}_{ij} = c_i + d_j + u_i^{\mathrm{T}} v_j \tag{2}$$

Weights c_i, d_j are trained simultaneously with u_{ik} and v_{jk}.

$$c_i \mathrel{+}= lrate * (r_{ij} - \lambda_2 (c_i + d_j - global\_mean))$$

$$d_j \mathrel{+}= lrate * (r_{ij} - \lambda_2 (c_i + d_j - global\_mean))$$

Values of parameters: $lrate = .001$, $\lambda_2 = .05$, $global\_mean = 3.6033$.

We will refer to this method as "RSVD2".

3.4 K-means

K-means and K-medians were proposed for collaborative filtering in [7].

Before applying K-means we subtract from each rating the user's mean rating. K-means algorithm is used to divide users into K clusters C_k, minimizing the intra-cluster variance.

$$\sum_{k=1}^{K} \sum_{i \in C_k} ||y_i - \mu_k||^2 \tag{3}$$

, where

$$||y_i - \mu_k||^2 = \sum_{j \in J_i} (y_{ij} - \mu_{kj})^2 \tag{4}$$

, where J_i is the set of movies rated by user i.

For each user belonging to cluster C_k the prediction for movie j is μ_{kj}.

Our predictor is mean prediction of ensemble of 10 runs of K-means with K ranging from 4 to 24.

We will refer to this method as "KMEANS".

3.5 Postprocessing SVD with KNN

The following prediction method was proposed by an anonymous Netflix Prize contestant.

Let's define similarity between movies j and j_2 as cosine similarity between vectors v_j and v_{j_2} obtained from regularized SVD:

$$s(v_j, v_{j_2}) = \frac{v_j^T v_{j_2}}{||v_j|| ||v_{j_2}||} \tag{5}$$

Now we can use k-nearest neighbor prediction using similarity s.

We use prediction by one nearest neighbor using similarity s and refer to this method as "SVD_KNN".

We also obtained good quality clustering of items, using single linkage hierarchical clustering with the similarity s. Though it was not useful for improving prediction, we mention it, because clustering of items can be useful in applications in recommender systems, for example to avoid filling recommendation slots with very similar items.

3.6 Postprocessing SVD with kernel ridge regression

One idea to improve SVD is to discard all weights u_{ik} after training and try to predict y_{ij} for each user i using v_{jk} as predictors, for example using ridge regression.

Let's redefine y in this section as a vector: i-th row of matrix y, with missing values omitted (now y is vector of movies rated by user i). Let X be a matrix of observations - each row of X is normalized vector of features of one movie j rated by user i: $x_{j_2} = \frac{v_j}{||v_j||}$. For efficiency reasons we limit the number of observations to 500. If a user rated more than 500 movies, we use only 500 most frequently rated movies.

We can predict y using ridge regression:

$$\hat{\beta} = (X^T X + \lambda I)^{-1} X^T y \tag{6}$$

$$\hat{y_i} = x_i^{\mathrm{T}} \hat{\beta} \tag{7}$$

Equivalent dual formulation involving Gram matrix XX^T:

$$\hat{\beta} = X^T (XX^T + \lambda I)^{-1} y \tag{8}$$

By changing Gram matrix to a chosen positive definite matrix $K(X, X)$ we obtain the method of kernel ridge regression. It is equivalent to performing ridge regression in a possibly much higher dimensional space, implicitly defined by kernel K. Predictions in this method are made in the following way:

$$\hat{y}_i = K(x_i^{\mathrm{T}}, X)(K(X, X) + \lambda I)^{-1} y \tag{9}$$

We can look for a kernel (defining similarity or distance between observations) which will result in better prediction than $K(x_i^{\mathrm{T}}, x_j^{\mathrm{T}}) = x_i^{\mathrm{T}} x_j$. We obtained good results with Gaussian kernel $K(x_i^{\mathrm{T}}, x_j^{\mathrm{T}}) = exp(2(x_i^{\mathrm{T}} x_j - 1))$ and the parameter $\lambda = .5$.

For each user we perform kernel ridge regression. Observations are $\frac{v_j}{||v_j||}$, for the first at most 500 most frequently rated movies, rated by the user. We name the method with Gaussian kernel "SVD_KRR".

3.7 Linear model for each item

For a given item (movie) j we are building a weighted linear model, using as predictors, for each user i, a binary vector indicating which movies the user rated.

$$\hat{y}_{ij} = m_j + e_i * \sum_{j_2 \in J_i} w_{j_2} \tag{10}$$

where J_i is the set of movies rated by user i, constant m_j is the mean rating of movie j, and constant weights $e_i = (|J_i| + 1)^{-1/2}$. The model parameters are learned using gradient descent with early stopping.

We name this method "LM".

3.8 Decreasing the number of parameters

The regularized SVD model has $O(NK + MK)$ parameters, where N is the number of users, M is the number of movies, K is the number of features. One idea to decrease the number of parameters is, instead of fitting u_i for each user separately, to model u_i as a function of a binary vector indicating which movies the user rated. For example $u_{ik} \approx e_i \sum_{j \in J_i} w_{jk}$, where J_i is the set of movies rated by user i (possibly including movies for which we do not know ratings, e.g. qualifying.txt) and constant weights $e_i = (|J_i| + 1)^{-1/2}$, like in the previous section. This model has $O(MK)$ parameters.

$$\hat{y}_{ij} = c_i + d_j + e_i \sum_{k=1}^{K} v_{jk} \sum_{j_2 \in J_i} w_{j_2 k} \tag{11}$$

where J_i is the set of movies rated by user i.

The second proposed model is following:

$$\hat{y}_{ij} = c_i + d_j + \sum_{k=1}^{K} v_{jk} \sum_{j_2 \in J_i} v_{j_2 k} \tag{12}$$

Parameters v_{jk} and w_{jk} are merged and there are no constant weights e_i.

In both models parameters are learned using gradient descent with regularization and early stopping, similarly to the regularized SVD.

We name the first method "NSVD1" and the second "NSVD2".

4. EXPERIMENTAL RESULTS

Table 1 summarizes the results of experiments with methods described in the previous sections.

Combining results of RSVD2 method with six simple predictors called BASIC gives RMSE .9039 on the test set and

Predictor	Test RMSE with BASIC	Test RMSE with BASIC and RSVD2	Cumulative test RMSE
BASIC	.9826	.9039	.9826
RSVD	.9094	.9018	.9094
RSVD2	.9039	.9039	.9018
KMEANS	.9410	.9029	.9010
SVD_KNN	.9525	.9013	.8988
SVD_KRR	.9006	.8959	.8933
LM	.9506	.8995	.8902
NSVD1	.9312	.8986	.8887
NSVD2	.9590	.9032	.8879
SVD_KRR * NSVD1	—	—	.8879
SVD_KRR * NSVD2	—	—	.8877

Table 1: Linear regression results - RMSE on the test set

.9070 (4.67% improvement over Netflix Cinematch) on qualifying.txt, as reported by the Netflix Prize evaluation system. Linear regression with all predictors from the table gives RMSE .8877 on the test set and .8911 (6.34% improvement) on qualifying.txt.

The predictors described in this paper are parts of a solution which scores .8844 on the qualifying dataset – that is 7.04% improvement over Netflix Cinematch. The solution submitted to the Netflix Prize is the result of merging in proportion 85/15 two linear regressions trained on different training-test partitions: one linear regression with 56 predictors (most of them are different variations of regularized SVD and postprocessing with KNN) and 63 two-way interactions, and the second one with 16 predictors (subset of the predictors from the first regression) and 5 two-way interactions. In the first regression the test set is random 15% of probe.txt, and in the second – 1.5% of probe.txt.

All experiments were done on a PC with 2GHz processor and 1.2GB RAM. Running times varied from 45min for SVD_KNN to around 20h for RSVD2.

5. SUMMARY

We described a framework for combining predictions and described methods that combined together give a good prediction for the Netflix Prize dataset.

Possible further improvements of the solution presented:

- apply cross-validation-like solution described in chapter 2 – repeat calculations on different training-test partitions and merge the results.
- add different efficient predictors to the ensemble. Good candidates are methods already applied with success to collaborative filtering: Restricted Boltzmann Machines [8] and other graphical models [7].

6. ACKNOWLEDGMENTS

Thanks to Netflix for releasing their data and organization of the Netflix Prize. Thanks to Simon Funk for sharing his approach of using regularized singular value decomposition. Also, I would like to thank Piotr Pokarowski for the course Statistics II.

7. REFERENCES

[1] K. Ali and W. van Stam. Tivo: making show recommendations using a distributed collaborative filtering architecture. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, editors, *KDD*, pages 394–401. ACM, 2004.

[2] J. Bennett and S. Lanning. The Netflix Prize. *Proceedings of KDD Cup and Workshop*, 2007.

[3] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In G. F. Cooper and S. Moral, editors, *UAI*, pages 43–52. Morgan Kaufmann, 1998.

[4] K. Y. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151, 2001.

[5] G. Gorrell and B. Webb. Generalized hebbian algorithm for incremental latent semantic analysis. *Proceedings of Interspeech*, 2006.

[6] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[7] B. Marlin. Collaborative filtering: a machine learning perspective. *M.Sc. thesis*, 2004.

[8] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann Machines for collaborative filtering. *Proceedings of the 24th International Conference on Machine Learning*, 2007.

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