Project 2 Machine Learning: Recommender System

They see me learnin', they hatin'
Christophe Windler, Adrian Pace, Louis Baligand
Department of Computer Science, EPFL, Switzerland

***Abstract*—This is the report of our approach to predict ratings given as Project 2 in the course Pattern Classification and Machine Learning at EPFL in Autumn 2016. We use Alternating Least Squares to predict our ratings. We achieve an error of 0.98029 by combining different results using this approach. This method proves to be reasonably effective but not optimal. Another approach using neural networks might be more suitable.**

I. Introduction

In 2009, Netflix held a competition that consisted in approximating the ratings that users will give to a set of movies. The teams were given a set of ratings to work on and the winners reached an improvement of 10.06% over the original algorithm of Netflix [2]. Similarly, the purpose of the project is to predict ratings on items as accurately as possible. In this report we will discuss step by step our approach to solve this prediction problem and describe our results.
Let a matrix $\mathbf{R}$ be our rating matrix, where $r_{i,j}$ be the i^{th} movie rated by the j^{th} user. The ratings given by each user are between 1 and 5. If the user does not give any ratings to the movie, the rating is equal to 0. Our matrix has 10'000 users and 1000 movies. Our goal is to predict the missing ratings in the matrix $\mathbf{R}$.

II. Methodology

In this part we describe our different methods and how we process the data.

A. Pre-Processing

Every user rates a certain amount of movies. If a user does not rate many movies, our predictions for this user might not be accurate, since there is not a lot of data to predict from. Therefore, we decide to remove every user that does not rate more than a certain amount of movies from our matrix $\mathbf{R}$. In a similar way, every movie that is not rated by more than a certain amount of users might mislead our predictions and is also removed from our matrix $\mathbf{R}$. Depending on the number of minimum ratings it might make our prediction better. Therefore, we use grid search by testing multiple number of minimum ratings to find the best fitting value.

We also split the matrix $\mathbf{R}$ randomly in such a way that we have a test rating matrix and a train rating matrix. Therefore we can test our approaches and find the best parameters on "unknown" data. Once we find the best fitting parameters to our approaches, we apply our learning algorithm to the whole matrix $\mathbf{R}$ to have even more data to learn from and therefore to get a better prediction.

We also try to normalize the matrix $\mathbf{R}$ in different ways. We try to subtract the mean of each user to its corresponding ratings (without affecting the ratings that are not given). This gives a negative weight to ratings that are below average, and positive one to the ones above average. Another method to normalize our matrix $\mathbf{R}$ is to subtract the mean of each user and divide by the standard deviation.

B. Approaches

Different approaches are used to find the best predictions possible. For each of these approaches, we use a cost function to estimate the correctness of our predictions. The cost function is used on the training set and on the test set.

1) Baseline Means: The baseline methods consists of three methods. The global mean method is averaging the whole matrix $\mathbf{R}$ and setting every element of our prediction matrix $\mathbf{P}$ to this value. The user mean is computing the average of each user and setting every corresponding column to this value. The item mean is the same process as the user mean but averaging over the items. All these methods result to our prediction matrix $\mathbf{P}$. For each of these methods, we use the following cost function called Root Mean Squared Error (RMSE):

$$RMSE(\mathbf{P}) := \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} \frac{1}{2} [\mathbf{R}_{(d,n)} - \mathbf{P}_{(d,n)}]^2$$

where Ω is the set of non-zero ratings and (d, n) is the indices of these non-zero ratings. These methods give us a better idea of the error we can achieve using a better approach.

2) Stochastic Gradient Descent: Another approach is the Stochastic Gradient Descent (SGD). Let's define two matrices $\mathbf{W}$ and $\mathbf{Z}$, where our prediction matrix $\mathbf{P} = \mathbf{W}^T\mathbf{Z}$, where $\mathbf{W} \in \mathbb{R}^{K\times D}$, $\mathbf{Z} \in \mathbb{R}^{K\times N}$ that represent respectively the item features and the user features. In this case, K is the number of features, D the number of items and N the number of users. The regularized function of the SGD algorithm used to derive the gradient is as follows:

$$\frac{1}{2|\Omega|} \sum_{(d,n)\in\Omega} [\mathbf{R}_{(d,n)} - \mathbf{P}_{(d,n)}]^2 + \frac{\lambda_w}{2}\|\mathbf{W}\|^2_{Frob} + \frac{\lambda_z}{2}\|\mathbf{Z}\|^2_{Frob}$$

The gradient follows by deriving the expression with respect to $w_{(d,n)}$ and $z_{(d,n)}$ respectively, $\nabla_{\mathbf{W},\mathbf{Z}}$:

$$[\mathbf{E}_{(d,n)}\mathbf{Z}_{(:,n)} - \lambda_w \mathbf{W}_{(:,d)}, \mathbf{E}_{(d,n)}\mathbf{W}_{(:,d)} - \lambda_z \mathbf{Z}_{(:,n)}]$$

where (d, n) are the non-zero indices and the prediction error is $\mathbf{E}_{(d,n)} = \mathbf{R}_{(d,n)} - \mathbf{W}^T_{(:,d)} \mathbf{Z}_{(:,n)}$. After each iteration of SGD, we update the item and user features by adding the item and user gradient multiplied by a factor γ. Therefore, there are 5 parameters that needs to be determined to find a good prediction: regularization term of item λ_w, regularization term of user λ_z, the step size γ, the number of features K and the number of iteration to have a good convergence.

3) Alternating Least Square: The last approach we use is the Alternate Least Square (ALS). More precisely we use the weighted regularized ALS:

$$f(\mathbf{W}, \mathbf{Z}) = \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} \frac{1}{2} [\mathbf{R}_{(d,n)} - (\mathbf{W}^T\mathbf{Z})_{d,n}]^2 + \sum_d n_{i_d} \frac{\lambda_w}{2} \|\mathbf{W}_{(:,d)}\|^2 + \sum_n n_{u_n} \frac{\lambda_z}{2} \|\mathbf{Z}_{(:,n)}\|^2 \quad (1)$$

where n_{i_d} and n_{u_n} are the number of non-zero values of the d^{th} item and the n^{th} user respectively. By deriving equation 1 with respect to w_d and z_n and setting it to zero, we obtain two equations:

$$\mathbf{W}_{(:,d)} = \mathbf{A}_d^{-1}\mathbf{V}_d, \quad \mathbf{Z}_{(:,n)} = \mathbf{A}_n^{-1}\mathbf{V}_n,$$
$$\text{with } A_d = \mathbf{Z}\,\mathbf{Z}^T + \lambda_w\, n_{i_d}\, \mathbf{I}_{K\times K}, \mathbf{V_d} = \mathbf{Z}\,\mathbf{R}_{(d,:)}$$
$$\text{and } A_n = \mathbf{W}\,\mathbf{W}^T + \lambda_z\, n_{u_n}\, \mathbf{I}_{K\times K}, \mathbf{V_n} = \mathbf{W}\,\mathbf{R}_{(:,n)}$$

where $\mathbf{I}_{K\times K}$ is the $K \times K$ identity matrix. Our implementation of ALS consists of initializing $\mathbf{W}$ and alternatively solving $\mathbf{W}$ by fixing $\mathbf{Z}$ and solving $\mathbf{Z}$ and fixing $\mathbf{W}$ until a certain stopping criterion is satisfied. Again, there are a couple of parameters we need to find to improve our predictions: λ_w, λ_z, the number of features K.

For both SGD and ALS, we use the simple $RMSE(\mathbf{P})$ stated in the baseline means to estimate the correctness of our predictions.

C. Post-Processing

In the pre-processing of the data, we decide to remove the users and items whose number of ratings is inferior to a certain threshold, to eliminate the user bias. We then calculate the features of the remaining users and items. However for the prediction, we need to calculate these removed users' and items' features. We insert in the features matrices the removed elements at their original position and put each of their features to the mean of the corresponding feature of the other elements. Then we perform a variant of the ALS where each newly added user's features are calculated based on item's features matrix and the newly added item's features from the user's features matrix.

We also combine two predictions from ALS and SGD or both ALS. To do so, we take a weighted average of the two predictions using the following formula:

$$\mathbf{P_{mean}} = \mathbf{P_0}\, x + \mathbf{P_1}\, (1 - x) \quad (2)$$

where x, $\mathbf{P_0}$, $\mathbf{P_1}$ and $\mathbf{P_{mean}}$ are respectively the weight, the first prediction, the second prediction and the weighted average of our prediction.

III. Results

We try every approach individually using the best parameters we obtain the results in TableI on Kaggle.

TABLE I
$RMSE$ ON TEST DATA SET FOR DIFFERENT APPROACHES

Approaches	RMSE
Baseline global mean	1.11786
Baseline user mean	1.09268
Baseline item mean	1.02982
SGD	0.98888
ALS	0.98172
ALS Normalized	0.98822
ALS Combined	0.98029

SGD parameters: iteration step $\gamma = 0.05$, $K = 20$, $\lambda_z = 0.15$, $\lambda_w = 0.04$.
ALS parameters: convergence value: 10^{-4}, $K = 20$, $\lambda_z = 0.8$, $\lambda_w = 0.01$.
ALS Normalized parameters: convergence value: 10^{-4}, $K = 20$, $\lambda_z = 0.8$, $\lambda_w = 0.01$.
ALS Combined parameters: convergence value: 10^{-4}.
Prediction 1: $K = 10$, $\lambda_z = 0.4$, $\lambda_w = 0.015$
Prediction 2: $K = 20$, $\lambda_z = 0.8$, $\lambda_w = 0.01$

We are especially interested to find the best parameters for ALS, since it yields better results. First we fix λ_w and λ_z and try to find the best fitting number of features K. We obtain the test and train graph presented in Fig. 1

The computation time of the best number of features is long, therefore we decide to apply our grid search to find λ_w and λ_z on a reasonable number of features that yields a good result too. We pick $K = 20$. We plot the obtained results in Fig. 3. As we can observe, the best fitting λ_w and λ_z are 0.08 and 0.01 respectively.

We also try a grid search with $K = 10$ features. We find that the best fitting λ_w and λ_z is equal to 0.04 and 0.15 respectively. By curiosity, we try a couple of λ_w and λ_z values for $K = 120$ (including the values that are best fitting for $K = 20$). The best result we obtain is 0.98166 with a smaller convergence, which is not much better than the obtained results for 20 features considering the computation time. The best fitting λ_w and λ_z for $K = 120$ are the 0.01 and 0.9 respectively.

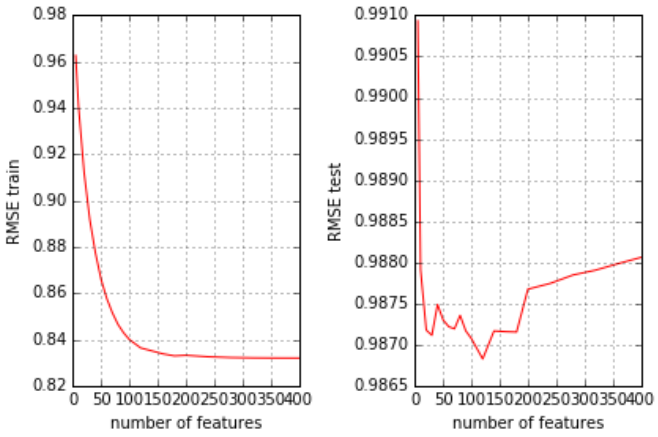

Fig. 1. $RMSE$ train and test for different number of features

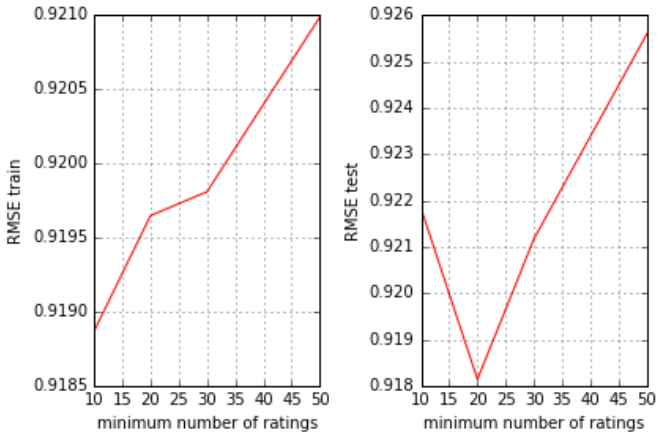

Fig. 2. $RMSE$ train and test for different minimum number of ratings

Moreover, we can observe on Fig. 2 that setting the minimum number of ratings per item and user to 20 yields the best prediction.

We obtain the best result on the submission website Kaggle by combining multiple predictions using equation 2. We combine the best fitting result for 10 and 20 features with a weight of 0.54 and 0.46 respectively.

IV. Discussion

Various methods are tested to get the best prediction possible. In this section we discuss the obtain results and compare the different parameters to find the optimal prediction.

A. Preparing the data set

Firstly we remove the user and items which has a number of ratings below a certain threshold. The idea is to eliminate the users whose prediction are unreliable since they depend on a few ratings. We try with a minimum number of ratings ranging from 0 to 200 with ALS with default parameters. We observe that removing this bias does not make a noticeable difference when the threshold is higher than 20. On the contrary, it worsen our prediction since we have less ratings to train on. We hence decide to choose 20 as a minimum number of ratings.

Then we try normalizing the data set. The idea is to neutralize the user bias. Indeed, each user rates differently, some higher than others. By normalizing the data, we have each ratings independent of how high a user generally rates. However we observe that normalizing the data does not yield us a better prediction so we decide not to normalize the data.

Also, we decide not to use cross validation, but to only use a train and test data split. This is motivated by the high number of ratings and therefore high computational needs.

B. Training methods

1) Baseline means: Once the data set is pre-processed, we have to choose which method and parameters we use. First we try the baseline means. The baseline item mean predicts better than the baseline user mean which predicts better than the baseline global mean. The baseline item mean has the smallest $RMSE$, which can be explained by the fact that a movie is good or not. Even though each user may rate differently, there is still a general consensus on the quality of a movie so we can predict that other users have a similar opinion on the movie. Next, the user mean is better than the global mean because the user mean reflects the way a user rates movies. He can rate higher or lower than average, so if we want to predict the rating of a user on an item, we can give it the rating he gives in average.

2) Stochastic Gradient Descent: For this method, we have to find good λ_w, λ_z, γ and number of features K. With a number of features of 20 and λ_w and λ_z of 0.065, we try a range of γ values. We also realize that after around 20 iterations, the SGD is not improving our prediction much. Furthermore, it soon becomes clear that this method is less accurate than the alternating least square. We decide then not to focus on this method.

3) Alternating Least Square: For this method we have to choose the best parameters λ_z, λ_w and K (number of features). The stopping criterion is when the difference of the last two $RMSE$ lies below a certain threshold. For testing different parameters, we decide to pick a threshold equal to 10^{-3} to get better computation time. For our prediction, we pick a smaller threshold, 10^{-4}, to get a better estimation. We realize that a smaller threshold does not improve our prediction much and worsen the performance of our approach.

We first fix λ_w and λ_z to 0.09 and search the number of features that yields the best prediction. We can see that the higher the number of features, the better the predictions on the train data. However, if the number of features is too high,

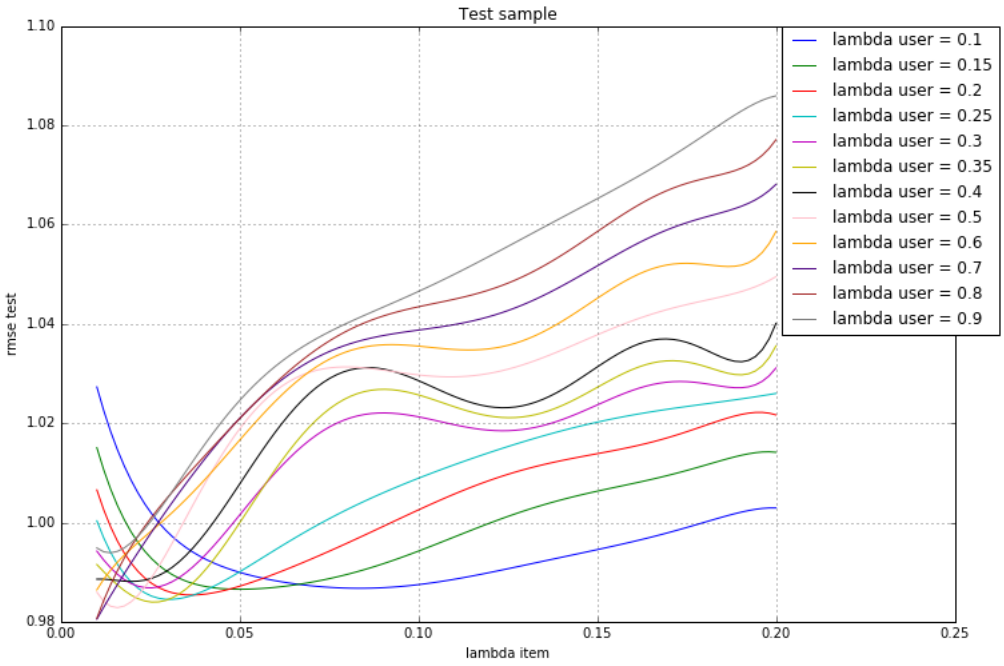

Fig. 3. $RMSE$ test for combinations of λ_w and λ_z

the test error increases. This is due to over-fitting. We conclude that the best number of features we use is either 20 or 180 features. We choose 20 features as it speeds up our future calculations. Interestingly we once try to use 10 features, and it has a better result on Kaggle than we are supposed to obtain according to our test data set. We decide to also look at 10 features.

After, we estimate λ_w and λ_z. We analyze the train and test $RMSE$ with different λ_w and λ_z using Grid search by fixing the number of features (in our case we fix it to 10 and 20). With this analysis, we are able to choose the λ_w and the λ_z that minimizes our prediction error on the test sample. For 10 features, we pick λ_z of 0.15 and a λ_w of 0.04 and for 20 features λ_z of 0.8 and a λ_w of 0.01. It is interesting to note that high λ_z and λ_w increases our error because it regularizes the ALS too much and low λ_z and λ_w does not regularize the ALS enough and the prediction is over-fitting.

C. Post-Processing

After the features for each item and user are calculated, the removed users and items (which has a low number of ratings) are added back. These newly added elements are then trained according to the existing feature matrix with a variant of ALS. We however observe that the number of iterations we execute this update yields no significant improvement. This is because the update is executed on only the few newly added elements and it does not produces a lot of changes. We decided to execute the update only once.

Then we calculate our predictions by multiplying the item's features with the user's features. We however notice that by taking a weighted average of two different predictions, we can improve our predictions. So we decide to find the best weights between different ALS prediction with 10 and 20 features. The cost-minimizing weights for the average is 0.54 and 0.46.

V. Conclusion

Our prediction yields a kaggle score of 0.98029 with the combination of two Alternating Least Square computations with different parameters.

Various methods are used on this project to minimize our prediction error, each with different accuracy and computational needs. While our focus is mostly on Stochastic Gradient Descent and Alternating Least Squares, it would have been interesting to look at others implementations such as Neural Networks. Also, it is important to note that our predictions are calculated with limited computational power and that better predictions can be computed with our implementation given superior processing power.

References

[1] Moritz Haller. Predicting user preferences in python using alternating least squares. http://online.cambridgecoding.com/notebooks/mhaller/predicting-user-preferences-in-python-using-alternating-least-squares.

[2] Yehuda Koren. Netflix prize, the bellkor solution. http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf.