A Hybrid Blending Based Approach for Collaborative Filtering

Jakub Lichman, Rajasimha Narasimhakumar, Jack Clark
Group: Pragmatic Chaos, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—In this paper, we present our novel approach to the user-item collaborative filtering challenge based on a highly popular matrix factorization method. We have explored various baseline methods and their enhancements but in the end we came up with a novel solution through weight based blending of the Stochastic Gradient Descent results with decreasing learning curve. The results show that our blending techniques reduce the RMSE considerably. We present and evaluate our approaches along with some baseline methods, perform a comparative analysis and discuss future extensions to our approaches.**

I. Introduction

Websites typically have much more content than a user would ever wish to view and so a website needs to be able to make a decision as to what it should show a user. Websites want happy users, so they want to show things that they think users will like. Recommendation systems aim to solve the problem of deciding what item(s) a given user might want to see, given a set of all possible items. Adding a recommendation system to a website can increase revenue, improve user engagement and prevent user churn. This is why most websites have recommendation systems, with some prominent examples being Amazon, Netflix and Spotify.

Collaborative filtering is an important and widely used technique in recommendation systems. Collaborative filtering aims to make automatic predictions of a users interest based upon various information sources, which are typically comprised of the users past behaviour and the behaviour of other users. The main advantage of the collaborative filtering approach over other methods is its independence of actual understanding of the items involved in prediction of the outcome. Many collaborative filtering methods use matrix factorization techniques to try to better understand the relationships in the data. These relationships can then be used to predict ratings for user-item pairs that are missing in the matrix.

The rest of the paper is structured as follows: In Section II, we give an overview of the baseline models. In Section III, we discuss our basic approach to solving the collaborative filtering problem. In Section IV, we discuss how we extended our basic approach, with a novel blending of the results from our earlier approach, to get our best results. In Section V, we discuss our outcomes.

II. Baseline Models

The collaborative filtering problem can be defined as follows: Given an incomplete matrix $A \in \mathbb{R}^{m \times n}$ predict values for all missing ratings, based on existing entries. In this section we give a short overview of well known techniques based on matrix factorization.

A. SVD

SVD is one of the most famous matrix factorization techniques. SVD decomposes a matrix A into the product of three matrices:

$$A = UDV^T \tag{1}$$

where $U \in \mathbb{R}^{m \times m}$, $D \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times n}$. Matrices U and V are orthogonal while matrix D is diagonal with singular values on the main diagonal sorted in descending order, with the largest value in the top left entry. From the collaborative filtering point of view we can see the U as users-to-feature matrix, V as items-to-feature matrix and the values on the diagonal of matrix D as the strength of each feature. Therefore we can take k most significant features ($U \in \mathbb{R}^{m \times k}$, $D \in \mathbb{R}^{k \times k}$ and $V \in \mathbb{R}^{n \times k}$) and approximate matrix A through their product. The approximation error depends on the value of k and on the distribution of eigenvalues. A common technique for selecting the right value of k is to plot the scree plot and find a "knee" in it. Koren [1] further improved SVD by extending the models to exploit both explicit and implicit feedback by the users.

B. NMF

Non negative matrix factorization or NMF is a another decomposition technique where the data matrix is decomposed as the product of two matrices U and V such that their elements are always positive and are L_1 column normalized.

$$A = U^T V \tag{2}$$

$$a_{ij} = \sum_z u_{zi} v_{zj} \tag{3}$$

C. SGD

The stochastic gradient descent method optimizes a given loss function by moving in the opposite direction of the loss functions gradient. In the stochastic version of gradient descent, samples are randomly considered and the weights are updated whenever a sample is considered, rather than after considering all samples. In our case, we add regularization via penalty terms inn the loss function to prevent overfitting and suitable learning rates are experimented in order to achieve better and faster convergence. This method is proven to be simple and efficient in most cases as convergence is guaranteed. In SGD, firstly, the factor matrices U and V are initiated using random values. Thereafter, a random entry $\hat{a}_{ij}$ is chosen from the data matrix A and the gradient is calculated corresponding to the same index of U and V as shown below:

$$\nabla_{ij} = \hat{a}_{ij} - u_i^T v_j \tag{4}$$

Following this, the entries of the constituent matrices are updated using these equations:

$$u_i = u_i + \eta * (\nabla_{ij} * v_j - \lambda u_i) \tag{5}$$

$$v_j = v_j + \eta * (\nabla_{ij} * u_i - \lambda v_j) \tag{6}$$

where the hyperparameter η is the learning rate and λ is the regularization parameter to prevent overfitting.

D. ALS

ALS is a two-step iterative optimization technique for randomly initiated factor matrices U and V, where in every iteration, the first matrix is fixed and the second is solved and vice-versa. This method guarantees a minimal RMSE and in each step the error defined by the cost function can either decrease or remain unchanged. Alternating between the two steps guarantees reduction of the error, until convergence. Similar to gradient descent optimization, it is guaranteed to converge to a local minimum. The equations that update the factor matrices in each step are as follows:

$$U = (VV^T + \lambda I)^{-1} V A^T \tag{7}$$

$$V = (UU^T + \lambda I)^{-1} U A \tag{8}$$

We can use the analytical solution as it involves inverting a small $k * k$ matrix (generally ranging from 10 to 1000), and sparse matrix multiplication. The other hyperparameter λ is used for regularization.

III. Method

In this section we describe our progress towards the novel approach that yields the best results in predicting the missing ratings for the 10000 users and 1000 items matrix. There were 1176952 ratings given overall, which accounts for around 11.77% of all possible ratings. Our goal was to predict the missing entries in rating matrix $A \in \mathbb{R}^{10000 \times 1000}$.

A. Error function

As an indicator of performance, we employ Root Mean Square Error (RMSE) as the error function in this project (both in Cross Validation and Kaggle). Let the r_{ui} denote the actual rating provided by a certain user u for an item i, with i = $\{1, 2, ..., n_u\}$ ($n_u \leq n$, where n is the number of all available items) and let p_{ui} denote the prediction generated by a certain algorithm for the same user and the same item. RMSE, relating to user u, is then defined by:

$$RMSE_u = \sqrt{\frac{\sum_{i=1}^{n_u} (r_{ui} - p_{ui})^2}{n_u}} \tag{9}$$

The total RMSE can be obtained as an average of the RMSEs of all users:

$$RMSE = \sqrt{\frac{\sum_u \sum_{i=1}^{n_u} (r_{ui} - p_{ui})^2}{\sum_u n_u}} \tag{10}$$

B. Cross validation

To overcome the daily submission restriction on Kaggle, we split data randomly into a training set containing 90% of data and the remaining as testing set that we used for evaluation. We tracked RMSE for both sets since RMSE of the testing set approximates the result on Kaggle more accurately and too low of an approximation error on the training data can indicate overfitting. The models that performed the best were usually the ones where the test RMSE was low but the training one quite high (no overfitting).

C. Baseline Models and Grid search

We implemented all baseline algorithms described in Section II. Their performance was much worse than we expected. We could not achieve an RMSE better than 1.0 on Kaggle (except for SGD). The problem we found was that we did not search for parameters systematically. Therefore we adopted grid search where we tuned the hyperparameters in order to minimize test RMSE. The lowest achieved errors are summarized in Table I. From the results it is obvious that SGD clearly outperformed all the other baseline models and therefore we decided to focus our efforts on methods that can be built on top of it.

D. Tuning the SGD

The first observation we made was that both the train and test fit errors did not converge after 50 million iterations [1]. Therefore we decided to experiment with the number of iterations. We doubled the number of iterations to 100 million and observed a very small improvement in RMSE after 60 million iterations. The remaining 40 million iterations led to a negligible improvement (< 0.0005) in RMSE.

[1]One iteration in our case is the update of a single entry in a matrix together with the corresponding biases.

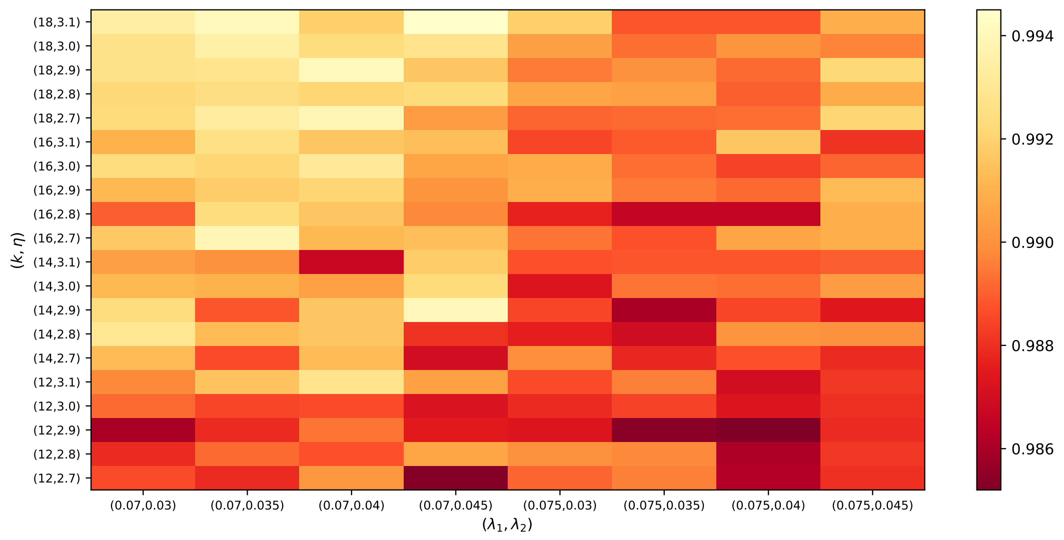

Figure 1. RMSE scores for different k, η, λ_1 and λ_2 values produced by a grid search.

In addition to the number of iterations, our SGD algorithm requires 4 other hyperparameters: k, λ_1, λ_2 and η. Parameter k indicates the number of features to use. Matrices U and V are then of shape $U \in \mathbb{R}^{10000\times k}$ and $V \in \mathbb{R}^{1000\times k}$. λ_1 is the regularization term for matrices U and V, and λ_2 is the regularization term for biases b_u and b_v. In order to find the best performing configuration we ran a grid search and the visualization of it can be seen in Figure 1.

E. Decreasing learning curve

Choosing the right learning rate was challenging since a very small learning rate generally means we are more likely to find a local minimum rather than the global one. On the other hand, a higher learning rate is not very accurate in later iterations and can lead to suboptimal results. Therefore we decided to split the iterations into 12 parts and assign a decreasing learning rate to each of them. This enhancement brought an additional improvement in the accuracy of predictions, as can be seem in Table I with name SGD$_{lr}$.

Model	k	η	Public RMSE
ALS	900	0.5	1.10867
SVD	7	–	1.05471
NMF	8	0.0055	0.99807
Baseline[2]	–	0.005	0.99768
SVD*	1	0.005	0.99566
SVD++*	1	0.007,	0.99507
SGD	12	0.00638	0.98288
SGD$_{lr}$	12	<0.09450,0.00005>	0.97949

Table I
PERFORMANCES OF BASELINE ALGORITHMS. MODELS MARKED WITH * WERE RUN WITH HELP OF THE SURPRISE [2] LIBRARY.

IV. BLENDING

After successfully tuning SGD to reach an RMSE smaller than 0.98, we wanted to further improve the accuracy of our predictions, using ideas from [3]. Therefore we decided to apply a post-processing blending step.

A. Blending function

In order to mix different predictors we can calculate a linear combination of their predictions. Thus, we adopt the *weighted average* as our blending function. Let p_{mui} be the prediction generated by predictor Pr_m with weight w_m for the user u and the item i. The final prediction p_{ui} can be defined by:

$$p_{ui} = \frac{\sum_{m=1}^{M} w_m p_{mui}}{\sum_{m=1}^{M} w_m} \tag{11}$$

The total RMSE becomes:

$$RMSE = \sqrt{\frac{\sum_u \sum_{i=1}^{n_u} (r_{ui} - \frac{\sum_{m=1}^{M} w_m p_{mui}}{\sum_{m=1}^{M} w_m})^2}{\sum_u n_u}} \tag{12}$$

We look for the vector of weights $w = [w_1, w_2, ..., w_M]$ which would minimize the error. An important remark is that the weights are the decision variables and equation 12 is the objective function; thus ratings and predictions are known parameters. Moreover, it is important to point out that if we had used a different vector of weights for each user, we would have to solve as many specific optimization

[2]Prediction in form: $r_{ui} = \mu + bu_u + bi_i$ where bu and bi are user and item biases.

problems as the total number of users; we call it personalized blending but we never tried it. On the other hand, a different approach is global blending. We look for a single vector of weights that is independent from the users. We used this approach since it requires much less computational resources and tuning time.

B. Predictions blending

The goal of the predictions blending was to find vector of weights w and subset of predictions Pr to minimize the RMSE. We started by blending the baseline methods' predictions. Naturally we wanted to include the SGD$_{lr}$ predictions since it performed the best. The results are reported in Table II. We can see that their performance is actually worse than performance of SGD$_{lr}$ by itself. We also observed that the less weight we put on SGD$_{lr}$ the worse the result blending produces. It means that other methods had nothing to contribute and the only effect they made was increasing the RMSE of the blended predictions. Therefore we had to include predictions with similar RMSE to SGD$_{lr}$.

By observing the differences in the generated SGD$_{lr}$ predictions when run with different configurations, we noticed that the prediction values vary but their RMSEs are very similar. Thus, we decided to blend various configurations of SGD$_{lr}$. Let $SGD_{lr}(k, \lambda_1, \lambda_2, \eta)$ be the configuration of SGD$_{lr}$ and the default configuration be $SGD_{lr} = SGD_{lr}(12, 0.08, 0.04, 2.9)$. If one of the parameters is a vector then we treat it as a vector of configurations with only one parameter changing. The results of blending various configurations are reported in Table II.

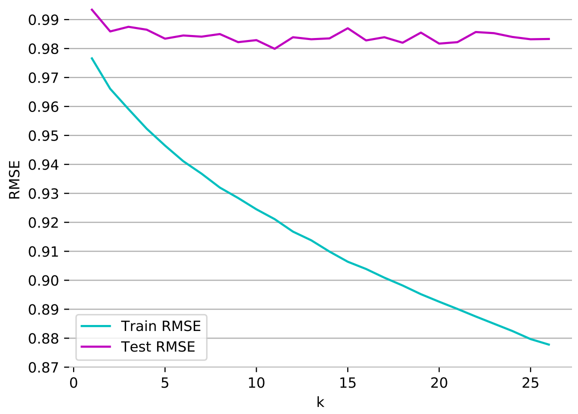

Figure 2. Best SGD$_{lr}$ scores for different values of k. We can see that the train RMSE decreases with increasing k, but the test RMSE does not improve if k is bigger than 12. Therefore we chose parameters $k \in <10, 17>$ for blending, as the RMSE converges but different values of k expose different features.

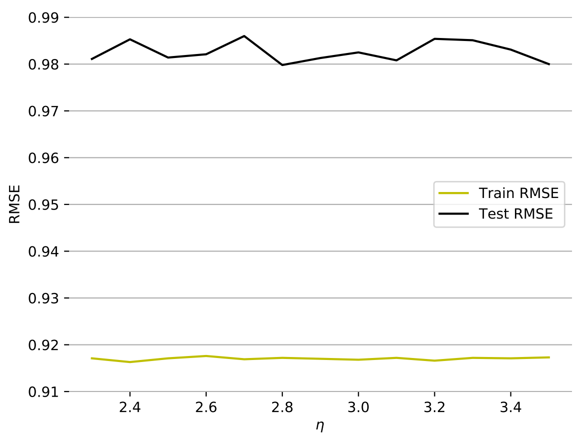

Figure 3. Best performing SGD scores for different η values. We can see that the learning rate has very limited influence on the RMSE but it can expose different features.

V. Results

As mentioned in previous sections we chose SGD$_{lr}$ as the baseline model to build upon since it performed much better than other baselines. In this section we present the results of our blending method that brought further RMSE improvements.

Our first RMSE improvement was achieved by blending two SGD$_{lr}$ configurations that differ only in the λ_1 parameter by a value of 0.005. Even such a small variation in one of the parameters was able to expose different features and therefore blending brought an improvement in the RMSE of 0.00132. We continued to experiment, and found the best results came from a blend of configurations with parameter k varying and configurations with parameter η varying. The results are reported in Table II.

Blended models	Weights	Public RMSE
SGD$_{lr}$/SVD++/Baseline/SVD	4/1/1/4	0.98842
SGD$_{lr}$)/SVD++/Baseline	4/1/1	0.98126
SGD$_{lr}$/SGD$_{lr}$(12,0.075,0.04,2.9)	5/4	0.97817
SGD$_{lr}$([10..17],0.08,0.04,2.9)	1/1../1	0.97723
SGD$_{lr}$([10..17],,,)/SGD$_{lr}$(,,,[2.5,2.6,..3.5])	1/1../1	0.97718

Table II
BEST PERFORMING COMBINATIONS OF SGD$_{lr}$ AND BASELINE MODELS.

VI. Summary and Future Work

In this paper we have presented our novel approach to the collaborative filtering challenge based on a blending of different SGD configurations. We have also described which baseline approaches we used and how we tuned them in order to get the most accurate predictions. As a possible enhancement to our approach, one can run SGD to find optimal vector of weights which can lead to further improvements in accuracy.

References

[1] Y. Koren, “Factorization meets the neighborhood: A multi-faceted collaborative filtering model,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[2] N. Hug, “Surprise, a Python library for recommender systems,” http://surpriselib.com, 2017.

[3] F. Roda, A. Costa, and L. Liberti, “Optimal recommender systems blending,” in *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, 2011.