Collaborative Filtering Using Matrix Factorization with Neural Post-processing

Ezeddin Al Hakim, Cyril Stoller, Tobias Verhulst
Group: *The Slow Learners*, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—In this paper we present a novel approach to the collaborative filtering challenge that combines the highly popular technique of matrix factorization with neural networks as a post-processing step. To create features for the neural network, we optimize basic models by using extensive cross-validation, and explore different matrix initialization and learning rate functions. Then, we re-train parts of the matrix factors using neural networks. We explore very basic networks and show that the score indeed improves, creating opportunity for further work with more complex network architectures.**

I. Introduction

Collaborative filtering has become more and more present in our interconnected world, leveraging the large number of users of a site to give more targeted and accurate recommendations to individual visitors. These recommendations are especially powerful in multimedia streaming websites, for example suggestions for the next movie that a user might like to watch, based on his past activity and movie ratings. In fact, a lot of competitions like the Netflix prize have been organized to try to create the best possible recommender system, with first place prizes as high as 1 million dollars.

In this paper, we present our own recommender system. We use a very popular approach to the problem, which is based on matrix factorizations to extract the most relevant information out of the dataset. We try different basic models, constantly seeking to maximize their performance by tuning all the relevant parameters. On top of those models, we then construct a neural network that uses the matrix factorizations as the input and tries to predict with greater accuracy the relevant user ratings.

In section 2, we formally define the collaborative filtering problem as well as some basic models. We then show how we implemented those models and optimized their performance in section 3. In section 4, we then present our novel addition, the neural network post-processing, which acts as an addition to the basic models. We present the results of all models in section 5, and finally conculde in section 6.

II. Basic Models

The collaborative filtering problem that we explore is formally defined is as follows : Given is a matrix $A \in \mathbb{R}^{m \times n}$ of ratings, an entry $A_{i,j}$ represents a rating between 1 and 5 given to movie i by user j. Empty values (zeroes) in the matrix correspond to missing ratings. The goal is to predict those missing ratings from the known ones.

In this section we present the different simple models that we tried, each being an improvement on the previous ones.

A. SVD: Singular Value Decomposition

The first trivial model is a simple singular value decomposition of the matrix. The factorization usually uses 3 matrices, but can equivalently be written as finding two matrices U and V such that $A = U \cdot V^T$, where $U \in \mathbb{R}^{m \times K}$ and $V \in \mathbb{R}^{n \times K}$. Truncating the least important features (corresponding to reducing K), extracts the most important information, which in turn can be used to make predictions : a predicted rating $\hat{A}_{ij}$ for a movie i given by a user j is calculated as:

$$\hat{A}_{ij} = u_i^T v_j$$

m where u_i is the i-th row of U, and v_j is the j-th row of V.

B. SGD: Stochastic Gradient Descent

The problem with the singular value decomposition is that it requires a full matrix with ratings for every movie/user combination in the first place. One solution is to use a stochastic gradient descent for calculating the U and V matrices, which only considers the movie ratings that are known: For an observed rating $A_{i,j}$ (chosen randomly), first calculate the prediction error:

$$\delta_{i,j} := A_{i,j} - \hat{A}_{i,j} = A_{i,j} - u_i^T v_j$$

and then update u and v according to the following formula:

$$u_{i,k} \mathrel{+}= \eta * (\delta_{i,j} v_{j,k} - \lambda u_{i_k})$$
$$v_{j,k} \mathrel{+}= \eta * (\delta_{i,j} u_{i,k} - \lambda v_{j_k})$$

In this basic model, the hyper-parameters to optimize are the following:

- K: The number of features to use (rank of the U and V matrices).
- λ: The regularizer for the U and V vectors, which is used to avoid overfitting.
- η: The learning rate used, which can be constant or dependent on the iteration number.

This method was popularized by Simon Funk in [1].

C. SGD+: Stochastic Gradient Descent Improved with biases

Some movies have better average ratings than others and some users usually give higher ratings than other users. This can best be captured by biases. The new prediction for a given user / movie combination is now modeled as $\hat{A}_{i,j} = u_i^T v_j + c_i + d_j$, where c are the movie biases and d are the user biases. The update formulas for U and V remain the same, and the biases are updated in the following way:

$$c_i \mathrel{+}= \eta * (\delta_i, j - \lambda_2(c_i + d_j - \mu))$$
$$d_i \mathrel{+}= \eta * (\delta_i, j - \lambda_2(c_i + d_j - \mu))$$

where μ is the average observed prediction. This model uses the same hyper-parameters as for SGD, with the additional regularizer λ_2.

III. Method

In this section we describe how the optimal parameters were chosen in order to reach the best score for the basic models on the given dataset. The data was given as a matrix of 1000 movies times 10000 users filled with $1.1e6$ known ratings. The score of a prediction was based on the rooted mean squared error (RMSE) of the selected entries.

A. Main hyperparameters

To fine-tune the parameters we used a grid search with cross validation: The data can be split into parts of equal size, with one part being ”held back” as a validation set. The best parameters can then be estimated using the held back data as scoring data.

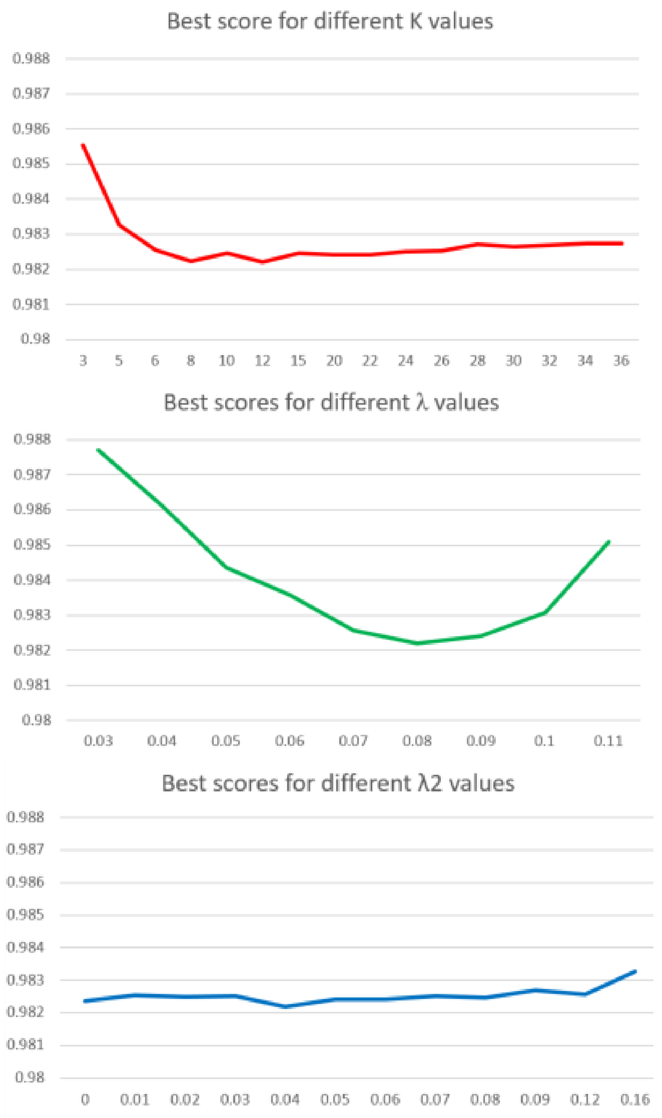

Figure 1. Best possible scores when varying hyperparameters

We ran grid searches for each of the basic models. In this section, we discuss the results and plots for the SGD model with biases (SGD+), as it contains all the hyper-parameters that the other models contain.

The three graphs in Figure 1 show the maximal scores that were obtained when varying one of the hyper-parameters, independently of the other two hyper-parameters.

From these graphs we can draw the following conclusions:

- Since the vertical axis range of the three graphs is the same, we see that varying the λ_2 bias regularizer did not affect the scores a lot, compared

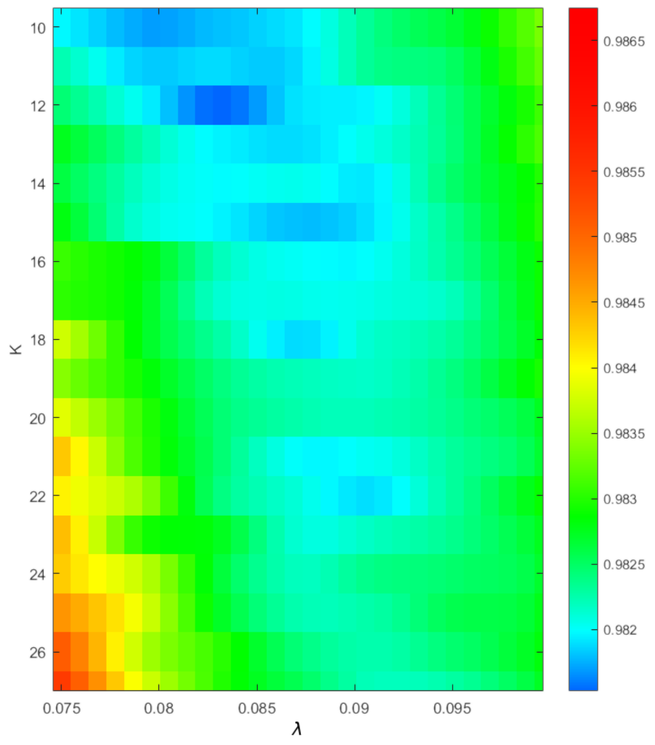

Figure 2. Scores for different K and λ values

to varying K and λ.

- Varying K did play an important role for small values, which is explained by the fact that too few factors do not allow to capture the essential information contained in A. For bigger Ks, the difference is not as significant, but the score gets worse as the decomposition includes more and more noisy information.
- Fine-tuning the λ regularizer was most important, as it was the main way to prevent over-fitting.

The following graph shows in more detail the relationship between the K and λ:

In the end, the combination of hyper-parameters that produced the best results for SGD+ were $K = 12$, $L = 0.083$, and $L2 = 0.04$.

B. *Other parameters*

- **Learning rate:** Choosing the right learning rate was a challenging task. A learning rate too small generally meant that local optima could be reached rather than the global one. A high learning rate also limited the precision of the solution and lead to suboptimal results. In the end, an adaptive learning rate dependent on the iteration number was chosen : Starting with $\eta = 0.09$, it would decrease to 0.03 after 30% of the iterations, continuing to decrease until a learning rate of $6 \cdot 10^{-6}$ was reached for the last iterations.
- **Matrix initialization:** We also investigated different methods of initializing the U and V matrices. One option was to initialize then with small normally distributed values. We tried different means and standard deviations. In general this did not have influence on the score, as long as values were random (to break symmetries) and non-zero.

IV. Neural post-processing

Now that we had thoroughly optimized some basic models, we applied a post-processing step to create the "SGDnn" model:

A. *The model*

After having trained the matrices U and V, as well as the vectors c and d using the SGD+ model, such that a prediction can be made with the formula $\hat{A}_{i,j} = u_i^T v_j + c_i + d_j$, the idea of the post-processing is the following : Discard the U matrix, as well as the c bias vector, and then retrain them using a neural network, e.g., find an α vector and β scalar such that $\hat{A}_{i,j} = \alpha \cdot v_j + \beta + d_j$. This can be done sequentially for each movie i, and ideally only trained on observed user-movie pairs (i, j). The formula can then be modified to $y = \alpha \cdot v_j + \beta$, where $y = m_{i,j} - v_j$. Finding such an α and β that minimize the error over all known user ratings for a given movie i, can be seen as a simple linear regression problem, where the y values are the observations, and v_j are the feature vectors (one for each user j). Using a neural network, we can model such a simple linear relationship, which was the focus of our work. However the approach has the potential to model arbitrarily complex relationships depending on the architecture of the neural network.

The method was inspired by [2], which uses a standard Ridge Regression instead of a neural network in one of the predictors.

B. *Implementation*

The evaluation of the neural network was made directly on the validation set, instead of using cross-validation as for the simple models. This was done because the number of relevant parameters suitable

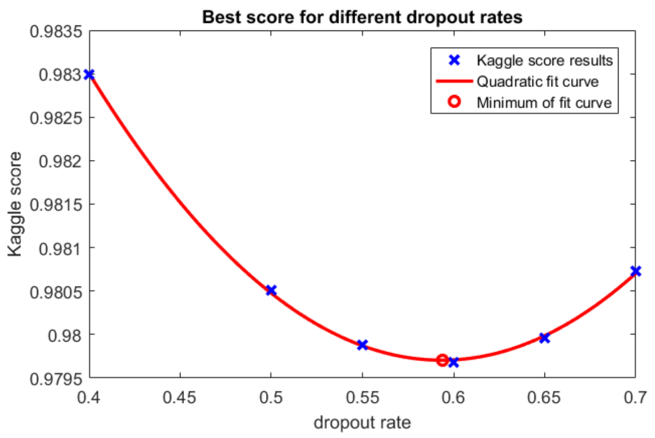

Figure 3. Test scores for different dropout rates

for cross-validation were limited, and more accurate scoring could be obtained this way.

Out of the simple networks that we trained, the best scores were obtained when using an trivial network corresponding to the linear regression case: A single layer with linear activation. This can be explained by the fact that the features were created by the SGD+ model for a simple linear relation.

Overfitting was avoided by regularizing in two different ways : Early stopping and dropout rates.

Early stopping consists in using a part of the training data as validation data, and stopping the neural network training as soon as the prediction error on this validation data increases. However, this means that part of the training data is lost.

Better regularization was therefore obtained when using so-called dropouts: Randomly discarding training datapoints with a probability q, where $q = 0$ means no dropout. Different dropout rates were tried, as indicated by the following figure :

We also briefly tried more complex neural networks, with multiple layers and different activation functions. Seeing that a lot of those networks performed poorly because of overfitting, we decided to focus on the more simple ones, to show that good results could indeed be achieved with such a post-processing approach. However we believe that further work could potentially make use of more complex networks to achieve even better results.

V. Results

In this section we compare the different models we used for our predictions. We also list the commands used so that they can be reproduced.

Model	RMSE	Parmeters	Command
SVD	1.00432	K=12	`python train.py --model=SVD`
SGD	0.98330	K=32, λ=0.084	`python train.py --model=SGD --K=32`
SGD+	0.97980	K=12, λ=0.084, $\lambda_2 = 0.04$	`python train.py`
SGDnn	0.97968	K=12, λ=0.084, $\lambda_2 = 0.04$, $q = 0.6$	`python train.py --model=SGDnn`

As we can see, the neural network post-processor managed to achieve a better score than the SGD+ model. Even though the difference might not be significant, even similar scores show that post-processing the matrix factorization has the potential to perform well, and that overfitting in the post-processing can successfully be avoided using dropouts.

VI. Summary

In this paper we presented an iterative implementation of a simple, yet powerful predictor for the collaborative filtering challenge. We described how we thoroughly optimized basic predictors in order to achieve their full potential. Upon these highly optimized models, we added our own post-processing methodology that combines matrix decomposition with neural networks. We then exposed the results of our experiments which show that the post-processing successfully increased the performance of our predictor when using very basic neural networks.

References

[1] S. Funk, "Netflix update: Try this at home," 2006. [Online]. Available: http://sifter.org/~simon/journal/20061211.html

[2] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, pp. 5–8.