# Collaborative Filtering: Stacking Matrix Factorization and Neural Networks for Improved Recommendations

Benjamin Hahn, Kevin Klein, Lorenz Kuhn
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Online businesses face the challenge of recommending relevant products to users based on users' previous preferences and similar customers. This work explores the use of classic matrix factorization methods on the one hand and recent neural network-based methods on the other hand. Final predictions were further improved using ensembling methods such as bagging and stacking. We report similar, competitive scores for matrix factorization methods and slightly lower accuracy for neural network-based methods with a final ensemble RMSE of 0.964.**

## I. Introduction

Many online businesses face the difficult yet crucial challenge of finding the most relevant products out of an enormous set of options for each of their users. Providing better recommendations, such as suggesting a movie on Netflix or a product on Amazon that a user might like, has been found to be linked to increased sales, an increase in consumers and competitive advantages [**?**]. Given the gigantic set of options, more than 570 million products are currently available on Amazon.com [**?**], a wide variance in preferences between different users and relatively little knowledge about individual users, this is a challenging task. Collaborative filtering [**?**] is an approach to this problem in which users' preferences are modeled based on their past interactions with the system. These methods are based on the fundamental assumption that similarities between the users in terms of their past preferences can be exploited to make new recommendations. Whereas many applications in industry are based on implicit feedback, such as the number of times a user has clicked on or viewed a certain product, we work with explicit feedback, that is ratings of movies on an integer scale. Notably, we exclusively work with meta data, that is, we do not have access to any information about the users and movies besides their ratings.

Traditionally, k-nearest neighbor approaches have been applied to this problem. In these algorithms, the ratings of the most similar users or items are exploited to infer the rating for a given user and item [**?**].

Recently, the most popular approach has been to represent users and items as vectors in a shared, low-dimensional latent space. The driving intention is to capture few hidden factors, assumed to have the greatest influence on the users' preferences. These embeddings may be obtained and then combined in a linear fashion using matrix factorization techniques [**?**], or in a non-linear way using neural networks [**?**].

We draw on and improve previous work by proposing an approach which applies a state-of-the-art ensembling method to recommendations produced by both sophisticated matrix factorization techniques as well as neural network models.

Our main contributions are thus the following:

1) We adapt and improve established matrix factorization techniques for our purposes.
2) Following the idea of pre-training neural networks, we use existing embeddings as input for our neural network. We obtain embeddings using a number of different methods which are then used as input for a feed-forward neural network to model nonlinear interactions between the users and the items.
3) The predictions are then combined via an ensembling method called stacking which learns the optimal blend as defined by the reduction of the RMSE on a holdout set. The methods tested include standard linear regression, extreme gradient boosting and feed-forward neural networks, among others.

## II. Data and metric

We use a dataset [1] of movie ratings by anonymous users which contains the set of observed ratings $\mathcal{R}$. For $(i, j, r_{ij}) \in \mathcal{R}$, $i \in [1 \dots 10000]$ is a user id, $j \in [1 \dots 1000]$ is an item id and $r_{ij} \in [1 \dots 5]$ the rating of the user for this item. $\mathcal{R}$ contains $11.8\%$ of all ratings, i.e. the rest is unknown. Note that we have no knowledge about either movies or users other than their ids.

The mean number of movies users have rated is $1167 \pm 952$, and no user has rated fewer than 8 movies. The distribution of both the mean number of movies seen as well as the mean rating of movies is roughly Gaussian. This led us to use all the data at our disposal since we deemed the effect of individual outliers (e.g. users with only 8 ratings) to be negligible on the overall results.

To evaluate our predictions, we are given a test set $\mathcal{T}$, where $(i, j) \in \mathcal{T}$ are unobserved tuples of user and item ids for which we want to infer ratings.

[1] https://www.kaggle.com/c/cil-collab-filtering-2018

The goal of this task is to minimize the Root Mean Squared Error (RMSE) given by:

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(i,j) \in \mathcal{T}} (r_{i,j} - \widehat{r}_{i,j})^2} \tag{1}$$

where $\mathcal{T}$ is a test set of ratings in which $r_{u,i}$ is the actual rating of user $u$ for movie $i$ and $\widehat{r}_{i,j}$ the corresponding prediction. Intuitively, this is a measure of how much our predictions deviate from the true ratings on average, while penalising large errors more strongly than small errors.

For the purpose of testing the performance of our models, we randomly split the data into several parts, with the largest part used as training data for the predictors and the smaller parts as validation sets for our models. For individual models we need only one validation set, yet when we wish to use ensembling methods we require a distinct second validation set (see section III-B).

Each validation set consists of 10% of the labeled data, with the training data equal to remaining amount.

## III. Models and Methods

### A. Individual models

Our models revolve around lower-dimensional representations of the initially sparse and high-dimensional data. Given a lower-dimensional representation $u_i$ of a user $i$ and $z_j$ movie $j$, also called embeddings, we can combine those to predict $r_{i,j}$. There are different approaches towards combining the latter. The simplest is a linear combination of both embeddings, the dot-product. More complex, non-linear combinations can be modeled by the activation functions used in neural networks (NNs). Our methods provide ways of generating such embeddings from the initially given ratings.

For those methods we assume that both embeddings stem from a shared embedding-space.

Observe that approximating $\widehat{r_{i,j}}$ by $u_i^T z_j$ can be expressed in matrix form: $\widehat{R} = UZ^T$ where row $i$ of $U$ represents the embedding of user $i$ and the row $j$ of $Z$ represents the embedding of movie $j$. Hence the typical reference to the term *matrix factorization*.

All models were evaluated via grid search of their hyper-parameters, most notably the embedding-dimension, on the validation set.

*1) Singular Value Decomposition (SVD):* We know that SVD can provide a matrix factorization for general matrices $M = U\Sigma V^*$. As $\Sigma$ is diagonal with non-negative entries, we can mutliply both $U$ and $V$ with the square root of $\Sigma$ to obtain implicit embeddings of users and movies. In order to obtain a *lower-dimensional* embedding, we simply drop all but the first $k$ dimensions, which minimizes the approximation error with regards to the Frobenius norm[2]. With those lower-dimensional embeddings we can generate sensible predictions $\hat{R}$. The careful reader might have noticed that the origin of our problem is that we do not dispose of such a matrix $M$. The ratings given to us only constitute a *sparse* matrix, constituted of many holes, as most users have rated but a small proportion of all movies. Hence we allow the brute compromise of filling up the holes of ratings matrix by heuristics, which we refer to as *imputation*. We have looked into different approaches towards imputation, ranging from simple row averages to more elaborate imputations based on the means and variances of the considered movie-user-pair. Imputation by row average perfomed best for SVD methods. Details can be found in our codebase. [3].

*2) Iterated SVD:* A simple yet powerful tweak on the previous approach is to reduce the importance of imputation by repeating the matrix factorization procedure. In particular, predictions from one round serve as imputation for the following round. We presume that this is an enhancement over single-round SVD as it decreases the importance of initialization.

$R$: Ratings matrix with holes, $k$ fixed rank
$M \leftarrow R$
Impute $M$ by initialization
**for** $i \in \{1 \dots n_{epochs}\}$ **do**
  $(U, \Sigma, D) \leftarrow SVD(M)$
  $U_{(k)} \leftarrow U[:, 1:k]$
  $\Sigma_{(k)} \leftarrow \Sigma[1:k, 1:k]$
  $D_{(k)} \leftarrow D[:, 1:k]$
  $M \leftarrow R$
  Impute $M$ by $U_{(k)}\Sigma_{(k)}D_{(k)}^T$
**end for**
**return** $M$

*3) Regularized 'SVD':* Both proposed methods suffer two obvious weaknesses:

- Due to the nature of SVD, some features will dominate others to an extent that might introduce numerical instabilities.
- Imputation dilutes our knowledge and gives equal importance to imputed values and to actual ratings.

By defining and optimizing for an explicit loss function we can tackle those problems:

- Regularization by the 2-norm reduces the difference of magnitude between features.
- We can explicitly only consider given ratings when calculating the approximation error with our loss function.

$$\mathcal{L} = \sum_{(i,j) \in T} ||r_{i,j} - \widehat{r}_{i,j}||^2 + \lambda_1(||U||_F^2 + ||Z||_F^2) + \lambda_2(||b_u||^2 + ||b_z||^2) \tag{2}$$

With $\widehat{r}_{i,j} = u_i z_j + b_{ui} + b_{zj}$, $b_{ui}$ representing a bias for user $i$, $b_{zj}$ representing a bias for movie $j$ and $\lambda_1, \lambda_2$

[2]Eckart–Young–Mirsky Theorem

[3]https://github.com/kkleindev/CILRecommender2018

serving as regularization factors. We found that regularizing the biases with a Lagrangian $b_{zj} + b_{ui} = \mu$ worked better in practice, where $\mu$ is the total average of ratings.

We tested two approaches towards minimization of this loss function:

- Stochastic Gradient Descent ('Reg SGD'): All features are trained and updated simultaneously. Given a rating $r_{i,j}$, all of the features from $u_i$ as well as $z_j$, $b_u i$ and $b_z j$ will be updated based on their relative partial derivative of the loss function.
- Coordinate descent ('SF'): One embedding dimension is learnt only after convergence of the previous dimension. This method has been popularized by Simon Funk [4].

We found that for both approaches it worked better to initialize the embeddings via SVD instead of using random matrices. In particular, we used SVD with imputation by variance-adjusted means[5]. Reg SGD gave the best results for an embedding-dimensionality of 17 and both regularization terms equaling 0.05.

*4) Iterated SF:* In order to further reduce the impact of the imputation used before the SVD initialization, we 'chained' several runs of SF. Hence embedding and bias vectors from previous runs were used as initialization for following runs. In addition, we found that smoothing via KNN accord to Bell et al. [**?**] slightly enhanced predictions. A grid-search over all hyperparameters yielded that 6 iterations of SF, 100 iterations over the dataset per SF application, an embedding-dimension of 20, $\lambda_1 = 0.02$ and $\lambda_2 = 0.05$ resulted in the best validation score.

*5) Neural Networks:* We leverage neural network models in two different ways. Firstly, we let the neural network learn lower-dimensional representations of users and items. Secondly, we apply neural networks to existing embedding vectors. Both approaches can be used to craft non-linear combinations for the sake of prediction.

The algorithm proposed in Neural Collaborative Filtering [**?**] uses the former approach. One-hot encodings for users and items serve as input to a feed-forward network. Hence, the vector defined by the activation of the first hidden layer can then be seen as the embedding learned by the neural network.

The second strategy exploits precomputed embeddings while also being able to model complex, nonlinear interactions between the users and the items. To this end, we train neural network models to predict ratings given user and item embeddings which are obtained through a number of different dimensionality reduction techniques: Locally Linear Embeddings [**?**], Non-Negative Matrix Factorization [**?**], Principal Component Analysis, SVD and iterated SVD, regularized SVD and the coordinate descent method as described above.

In particular the neural network model used is the following:

$$\begin{aligned} \mathbf{z_1} &= \phi_1(\mathbf{e_i}, \mathbf{e_u}) = [\mathbf{e_i}, \mathbf{e_u}] \\ \mathbf{z_2} &= a_2(\mathbf{W_2}^\mathbf{T}\mathbf{z_1} + \mathbf{b_2}) \\ &\dots \\ \mathbf{z_L} &= a_{L-1}(\mathbf{W_L}^\mathbf{T}\mathbf{z_{L-1}} + \mathbf{b_L}) \\ \hat{y}_{ui} &= \sigma(\mathbf{h}^\mathbf{T}\mathbf{z_L}) \end{aligned} \tag{3}$$

where $a_i, \mathbf{W_i}, \mathbf{b_i}$, denote the i-th layer's activation function, weights and bias respectively. With $\mathbf{e_i}$ and $\mathbf{e_u}$ we denote the embedding vectors for the users and items respectively. We use the rectifier (ReLU) activation function, which has been found to frequently yield superior results as compared to the traditionally used sigmoid and tanh functions [**?**]. Model parameters are then trained using the Adam optimizer [**?**] to minimize the squared loss.

Using grid search, we find the hyperparameters yielding the best results on the validation set (see table III-B).

### *B. Ensembling methods*

Ensembling techniques are based on the idea of combining multiple predictions to reduce their bias and the variance and, in turn, the overall prediction error.

This follows the intuition that different prediction methods may do better in different circumstances which is something we would like to exploit. Ever since the seminal Netflix prize they have played a prominent role where most of the best-placed teams achieved their final results by combining as many as 500 models [**?**], [**?**].

The most straightforward way to combine results is to take the uniform average over all results. This often yields an improvement but does not take into account strengths and weaknesses of individual models, which are all assigned equal weight.

To improve on this, we can split the data into three parts, creating two validation sets: the actual validation set for each predictor as well as a 'meta-validation' set for the stacking ensemble. Using the predictions of each predictor for the first validation set, we can fit a model to the ground truth ratings of the first validation set in a second stage. The resulting model can then be validated using the hold-out set. The final predictions are generated by training on all data and combining the predictions using the learned model.

This process is referred to as *stacking* or *blending*.

Any model can be used for this second stage in stacking that is capable of regression. For this project linear regression, ridge regression, extreme gradient boosted decision trees (xgboost) and neural networks were used to generate linear as well as nonlinear combinations of the predictions.

In the context of CF as opposed to classic regression we face the problem that predictions must be made for

[4]http://sifter.org/simon/journal/20061211.html

[5]https://github.com/kkleindev/CILRecommender2018

users with varying numbers of missing entries. As a result, the most straightforward implementation merely assigns the prediction matrix of each model a scalar weight.

Grid search was performed to find the best set of parameters for stacking using neural networks and xgboost. Moreover, different combinations of predictions were tested as an input to stacking as previous work has shown that models performing poorly on their own may still yield an improvement when included in an ensemble [**?**].

The previously described methods all use the output from various predictors to generate a combined prediction. Bootstrapped aggregation, or bagging, on the other hand, is a variance-reducing technique that runs a single method on a sample of the same data several times.

Several variants exist, with the one used in this work being row-wise bootstrapping [**?**, Chapter 6.3.2]. Here, the order of the rows (or columns) of the data is randomly selected with replacement to form a sampled matrix of the same size as the original ratings matrix. A chosen collaborative filtering algorithm is run on each sampled matrix and a uniform average over all results is taken to reduce the results to a single matrix.

## IV. Results

Table III-B shows an overview of the best performances for different predictors individually as well as different ensembling methods.

Table II
MODELS USED IN FINAL ENSEMBLE

| Model | Individual RMSE |
|---|---|
| Iterated SF with rank 8 | 0.981 |
| Iterated SF with rank 10 | 0.983 |
| Iterated SF with rank 13 | 0.982 |
| Iterated SF with rank 15 | 0.989 |
| Iterated SF with rank 17 | 0.988 |
| Neural net with PCA embeddings | 0.992 |
| Reg SGD | 0.978 |
| Reg SGD | 0.981 |
| Bagging SVD for 30 epochs | 0.979 |

In general, all matrix factorization methods achieved a similar performance and outperformed neural network-based methods. Regularized SGD achieved the best performance among the base predictors, with an improvement of $3\%$ compared to vanilla SVD.

All ensembling techniques led to similarly good results, lowering the final error to beneath the best values of individual predictors. Neural network ensembles led to the best score on Kaggle, albeit by a small margin. Note that the ensemble predictions were generated by training the base predictors on 100% of the available data to improve the RMSE beyond what was achieved when using 90% of the data as training data. Nonlinear methods in general have the potential to outperform linear models but this is highly dependent on the variety of input used - if the similarity is great, non-linearity will provide less of a benefit.

The best final score was achieved using an ensemble of 10 methods, listed in table IV.

## V. Discussion

In [**?**], letting the network learn its own embeddings as well as combining the predictions of this network with matrix factorization predictions yielded an improvement over standard matrix factorization techniques. We do not find the same results which might be attributed to the task in the paper being binary classification rather than multi-class classification.

Ensembling methods consistently improved scores beyond the base predictors although less difference was observed between different stacking methods than expected. When compared to [**?**] we used a smaller set of predictions which may explain why linear models perform similarly well as non-linear methods in our case.

Furthermore, the winners of the Netflix prize used several layers of stacking, i.e. using blends of blends which we did not explore in this work and could potentially bring about further improvement and explain our comparatively smaller performance gains due to ensembling.

One limitation of our stacking method is that it assigns each predicted ratings matrix a weight, thus disregarding user or item specific characteristics. We could extend the method to assign weights to individual users or items instead, although it would become more difficult to reliably make predictions for users or items with few ground truth ratings.

## VI. Conclusion

We were able to adapt existing models, most notably Simon Funk's matrix factorization technique and thereby improve the prediction RMSEs.

In particular the repetition of the coordinate descent algorithm as well as a non-random initialization of embeddings and postprocessing by smoothing helped improve the scores. Combining the latter with neural network models, although not performing astonishingly by themselves, allowed for an enhancement of 1.4 % percent. We found that a stacked ensemble using a neural network led to the best final results.

Further work could include investigating other neural network based collaborative filtering approaches, more sophisticated ensembling as well as using a greater variety of predictors.

Table I
VALIDATION SCORES OF INDIVIDUAL MODELS

| Linear Model | RMSE |
|---|---|
| SVD | 1.008 |
| Iterated SVD | 0.992 |
| Reg SGD | 0.978 |
| SF | 0.984 |
| Iterated SF | 0.980 |

| NN initialization | RMSE |
|---|---|
| None | 1.116 |
| SVD | 0.993 |
| Iterated SVD | 0.999 |
| Reg SGD | 1.043 |
| SF | 1.011 |
| LLE | 1.070 |
| NMF | 0.994 |
| PCA | 0.993 |

| Ensembling | RMSE |
|---|---|
| uniform average | 0.980 |
| linear regression | 0.975 |
| ridge regression | 0.976 |
| neural network | 0.964 |
| xgboost | 0.972 |