Collaborative Filtering: Picnic of Ensembled Matrix Factorization Models

Andreas Tsouloupas, Arad Mohammadi, Hsiu-Chi Cheng, Omkar Zade, Group: Picnic
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Collaborative Filtering (CF) is a common technique widely used and adopted by recommender systems. In this work, we approach CF by considering matrix factorization-based techniques. Such methods were successfully deployed in the past. First, we present two baseline methods, namely, Singular Value Decomposition and Alternating Least Squares. Then we introduce improved methods compared to the baselines, and finally, we combine them optimally with our 10-fold blending algorithm into a powerful system for recommendations.**

I. Introduction

Recommender systems are widely deployed on websites such as Netflix and Amazon. Considering their user tastes, those systems suggest specific items to specific users that (hopefully) match their interests. Using such recommender systems reportedly [1] has many benefits for a company, such as higher revenue and user satisfaction.

There are two broad categories for recommender systems, namely, Content-Based and Collaborative filtering. In the former, the system aims to build feature-based profiles for both users and items. These feature profiles are later used for recommendations. The drawback of this method is that finding the appropriate features is hard. In contrast, Collaborative filtering attempts to exploit the similarity between users' ratings to learn from the collective data provided by them.

In this work, we consider several CF matrix factorization-based techniques. We begin by presenting Singular Value Decomposition and Alternating Least Squares as our baselines. Then, we introduce some alternative methods, and finally, we combine them into a powerful system for recommendation by obtaining optimal blending weights.

II. Problem Setting

We are given a total of $N = 1,176,952$ ratings of $n = 10,000$ different users to $m = 1,000$ different movies/items. Let r_{ui} denote the rating assigned by user u to item i. The ratings are integer numbers between 1 to 5, which shows how much a specific user likes a specific item. Let $\mathcal{I} = \{(u,i) \mid r_{ui} \text{ is observed}\}$ be the set of pairs of user and item indices that are observed. We divide set $\mathcal{I}$ into 10 disjoint folds. Then, we consider all $\binom{10}{9} = 10$ different arrangements of folds into two groups. We denote by $T^{(f)}$ and $V^{(f)}$ the sets containing 9 folds and 1 fold, respectively, for all arrangements $f = 1, ..., 10$. Now we can define the following sets $\mathcal{I}_{train}^{(f)} = \bigcup_{fold \in T^{(f)}} fold$ and $\mathcal{I}_{val}^{(f)} = \mathcal{I} \setminus \mathcal{I}_{train}^{(f)}$. The former set (for all $f = 1, ..., 10$) contains 90% of the observed entries indices pairs, while the latter contains the remaining 10%. These sets will later be used for cross-validation and for learning optimal blending weights in Section IV.

Conventionally, the observed entries are represented as a sparse matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$. Formally, $\forall (u,i) \in \mathcal{I}$, $a_{ui} = r_{ui}$, and $\forall (u,i) \notin \mathcal{I}$, $a_{ui} = ?$. Duo to the sparseness of matrix $\boldsymbol{A}$, predicting missing values becomes particularly difficult. From the $n \times m = 10,000,000$ possible ratings, we know only as few as $N = 1,76,952$, which is just 11.77% of the whole matrix. Root mean squared error (RMSE) is used as the evaluation criterion to assess the performance of our models. RMSE calculates the root of the average distance between each predicted rating and true rating. Let $\mathcal{W}$ be an instance of observed entry pairs, then

$$RMSE = \sqrt{(\frac{1}{|\mathcal{W}|}) \sum_{(u,i) \in \mathcal{W}} (r_{ui} - \hat{r}_{ui})^2}$$

where $\hat{r}_{ui}$ is the predicted rating of user u for item i. The notation $\hat{r}_{ui}^{(j)}$ will be used later to denote the prediction of model j for the particular rating.

Our predictions are evaluated based on a public test set, which we denote by $\mathcal{I}_{test}$. The public test set contains 50% of the test data on which our predictions will finally be evaluated.

III. Individual Models and Methods

In our CF approaches, we focused on matrix factorization-based models. In this section, we will present seven individual algorithms we used for our recommender systems, starting from the two baselines.

A. Overview

The idea behind matrix factorization-based models is that preferences of user u for item i can be estimated by a few hidden factors. Here we describe the association between user u and its factors as $\boldsymbol{p}_u \in \mathbb{R}^k$, while that between item i and its factors as $\boldsymbol{q}_i \in \mathbb{R}^k$. Under this structure, the estimation of a rating $\hat{r}_{ui}$ of user u for item i is modeled by the dot product of the corresponding two vectors.

$$\hat{r}_{ui} = \boldsymbol{p}_u^T \boldsymbol{q}_i = \Sigma_{t=1}^{k} p_{ut} q_{ti} \tag{1}$$

Suppose there are n users and m items; we can assemble all those user factors as a matrix $\boldsymbol{P} \in \mathbb{R}^{k \times n}$ and item factors as $\boldsymbol{Q} \in \mathbb{R}^{k \times m}$. The columns of $\boldsymbol{P}$ and $\boldsymbol{Q}$ correspond to the user and item features, respectively. Once we calculate the dot product of $\boldsymbol{P}$ and $\boldsymbol{Q}$, we are able to estimate all the user-item interactions by a matrix $\hat{\boldsymbol{A}} \in \mathbb{R}^{n \times m}$, where $\hat{\boldsymbol{A}} = \boldsymbol{P}^T\boldsymbol{Q}$.

Our goal of matrix factorization-based methods is to figure out the matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ from limited observed ratings and make estimations of unknown ratings.

B. Missing Values and Normalization

In our case and numerous other real-life applications, the vast majority of ratings are missing. In order to model our recommender systems, one issue is to deal with missing values, denoted by ? in the sparse matrix $\boldsymbol{A}$. Some methods, such as SVD, cannot operate on sparse matrices directly; therefore, imputation is an unfortunate necessity. Here, we made use of the following method:

- Zero imputations: Let $\mathcal{W}$ be an instance of observed entry pairs, then $\forall (u, i) \notin \mathcal{W}$, $a_{ui} = 0$.

Moreover, sometimes it is beneficial to normalize data before applying the methods. Thus, we consider the famous normalization method, namely item-item z-scores

$$a_{ui} = \frac{a_{ui} - \bar{\boldsymbol{a}}_i}{\sigma(\boldsymbol{a}_i)}$$

where $\boldsymbol{a}_i \in \mathbb{R}^n$ is the i-th column of matrix $\boldsymbol{A}$. $\bar{\boldsymbol{a}}_i$ and $\sigma(\boldsymbol{a}_i)$ are the mean and standard deviation, respectively, which take into consideration only the observed entries of each column. It is important to mention that whenever normalization is used the prediction is obtained by reverting it, i.e., $\hat{r}_{ui} = (\hat{r}_{ui} + \bar{\boldsymbol{a}}_i) \times \sigma(\boldsymbol{a}_i)$.

C. Singular Value Decomposition (SVD)

SVD [2] is a popular method in linear algebra for matrix factorization. SVD factorizes a given matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ into two orthogonal matrices $\boldsymbol{U} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{V} \in \mathbb{R}^{m \times m}$, and a rectangular diagonal matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times m}$, such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T,\ \boldsymbol{\Sigma} = diag(\sigma_1, ..., \sigma_{\min\{n,m\}})$$

where $\sigma_i \geq \sigma_{i+1} \geq 0$. The normalization described in Section III-B is applied to $\boldsymbol{A}$ before SVD. Our goal is to find a low-rank approximation $\hat{\boldsymbol{A}}$ such that $rank(\hat{\boldsymbol{A}}) \leq k$, with the hope that it can perform well on missing values. Eckart-Young Theorem [3] shows that pruning the first k singular values in the SVD representation results in the optimal rank k approximation of a matrix. We thus only utilize the largest k values in $\boldsymbol{\Sigma}$ and the first k columns of $\boldsymbol{U}$ and $\boldsymbol{V}$. We can then rewrite our estimation matrix into

$$\hat{\boldsymbol{A}} = \boldsymbol{U}_k^T\boldsymbol{\Sigma}_k\boldsymbol{V}_k$$

where $\boldsymbol{\Sigma}_k = diag(\sigma_1, ..., \sigma_k) \in \mathbb{R}^{k \times k}$, $\boldsymbol{U}_k = (\boldsymbol{u}_1, ..., \boldsymbol{u}_k) \in \mathbb{R}^{n \times k}$, $\boldsymbol{u}_i$ is the i-th row in $\boldsymbol{U}$; $\boldsymbol{V}_k = (\boldsymbol{v}_1, ..., \boldsymbol{v}_k) \in \mathbb{R}^{m \times k}$, $\boldsymbol{v}_j$ is the j-th row in $\boldsymbol{V}$.

D. Alternating Least Squares (ALS)

One of the common approaches to CF is ALS [4]. ALS iteratively optimizes the objective function

$$\frac{1}{2}\sum_{(u,i)\in\mathcal{W}} (r_{ui} - \boldsymbol{p}_u^T\boldsymbol{q}_i)^2 + \frac{\lambda}{2}(||\boldsymbol{P}||_F^2 + ||\boldsymbol{Q}||_F^2) \qquad (2)$$

Here $||\cdot||_F$ is the Frobenius norm, and $\mathcal{W}$ is an instance of observed entry pairs. ALS monotonically improves the objective in an alternating manner. It converges to a fix-point; however, there is no guarantee that it will converge to the global optimal. An essential factor that drastically affects the final result is the initialization of $\boldsymbol{P}$ and $\boldsymbol{Q}$, which in our approach is done by SVD. The same normalization as in SVD is applied before SVD and ALS. The algorithmic procedure is illustrated in Algorithm 5 in the Appendices.

Algorithm 1 Global Bias

Input: $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, $\lambda, \mu \in \mathbb{R}^+$, $\boldsymbol{B}_u \in \mathbb{R}^n$, $\boldsymbol{B}_i \in \mathbb{R}^m$ and $\mathcal{W}$

Output: Optimal $\boldsymbol{B}_u$ and $\boldsymbol{B}_i$

```
while not convergent do
    for $u$ in 1...$n$ do
        $b_u = (\lambda + \sum_{i:(u,i)\in\mathcal{W}} 1)^{-1} \sum_{i:(u,i)\in\mathcal{W}} (a_{ui} - \mu - b_i)$
    end for
    for $i$ in 1...$m$ do
        $b_i = (\lambda + \sum_{u:(u,i)\in\mathcal{W}} 1)^{-1} \sum_{u:(u,i)\in\mathcal{W}} (a_{ui} - \mu - b_u)$
    end for
end while
```

E. Global Bias (GBias)

Alternating least squares is a common method used to minimize an objective. In this method, we attempted to utilize the power of ALS in order to minimize a different objective function given by

$$\frac{1}{2}\sum_{(u,i)\in\mathcal{W}} (r_{ui} - (\mu + b_u + b_i))^2 + \frac{\lambda}{2}(||\boldsymbol{B_u}||^2 + ||\boldsymbol{B_i}||^2)$$

where $\mathcal{W}$ is an instance of observed entry pairs. In contrast with all other methods, the prediction of a rating given by user u to item i is given by $\hat{r}_{ui} = \mu + b_u + b_i$, where μ is the global rating mean, b_u is the bias of user u towards its ratings compared to other users, and b_i is the bias of ratings given to item i compared to other items. We can then assemble all those user biases as a vector $\boldsymbol{B}_u \in \mathbb{R}^n$ and item biases as $\boldsymbol{B}_i \in \mathbb{R}^m$. This method is illustrated in Algorithm 1.

As in every application of ALS, the initialization affects the final result, i.e., the fix-point to which it will eventually converge. Vectors $\boldsymbol{B}_u$ and $\boldsymbol{B}_i$ are initialized as shown in Eq. (4) and (5) in the Appendices.

F. Singular Value Projection (SVP)

SVP [5] is a projected gradient descent method. For a given matrix $\boldsymbol{A}$, SVP iteratively makes an orthogonal projection onto a set of k-rank matrices given by

$$\boldsymbol{A}^0 = 0,\ \boldsymbol{A}^{t+1} = [\boldsymbol{A}^t + \eta^{t+1}\Pi_{\mathcal{W}}(\boldsymbol{A} - \boldsymbol{A}^t)]_k,\ \eta^0 > 0$$

where the projection $[]_k$ to rank k matrices is done via SVD. The learning rate for each epoch is given by $\eta^{t+1} = \frac{\eta^0}{\sqrt{t+1}}$ which was suggested in [5]. Before SVP, we normalize our data as in the previous methods. The prediction $\hat{r}_{ui}$ is equal to the entry a^L_{ui}, of the last projection matrix $\hat{\boldsymbol{A}} = \boldsymbol{A}^L$.

G. Singualar Value Thresholding (SVT)

SVT [6] has been successfully used in many low-rank optimization problems. SVT is illustrated in Algorithm 2. Data normalization, as in the previous methods, is applied before SVT. The algorithm is based on the shrinkage operator, which iteratively shrinks the singular values of $\hat{\boldsymbol{A}}$ towards 0. The nuclear norm is the convex envelope of the $rank$ function on matrices; hence, we hope that the result will be a low-rank approximation of $\boldsymbol{A}$.

Algorithm 2 SVT

```
Input: $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, $\tau \in \mathbb{R}^+$, $Ep \in \mathbb{N}$ and $\mathcal{W}$
Output: $\hat{\boldsymbol{A}} = shrink_\tau(\boldsymbol{A}^{Ep})$
$\boldsymbol{A}^0 = 0$
for $t$ in $0, ..., Ep - 1$ do
    $\boldsymbol{A}^t = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
    $\tilde{\boldsymbol{A}} = shrink_\tau(\boldsymbol{A}^t) = \boldsymbol{U}diag(max(0, \sigma_k - \tau))\boldsymbol{V}^T$
    for all $(u, i) \in \mathcal{W}$ do
        $a^{t+1}_{ui} = a^t_{ui} + \eta(a_{ui} - \tilde{a}_{ui})$
    end for
end for
```

H. Improved Regularized SVD (IRSVD)

Improved regularized SVD was introduced in [7], and it leverages the power of Stochastic Gradient Descent (SGD) in order to minimize the following objective

$$\frac{1}{2}\sum_{(u,i)\in\mathcal{W}}(r_{ui} - (\mu + b_u + b_i + \boldsymbol{p}_u^T\boldsymbol{q}_i))^2 + \frac{\lambda_1}{2}(||\boldsymbol{P}||_F^2 + ||\boldsymbol{Q}||_F^2) + \frac{\lambda_2}{2}(||\boldsymbol{B}_u||^2 + ||\boldsymbol{B}_i||^2)$$

The improvement of this method compared to its predecessor Regularized SVD (RSVD), suggested by Simon Funk [8], is the introduction of user-specific and item-specific biases. Each user u has a corresponding bias term denoted by $b_u \in \mathbb{R}$ and each item i a bias term denoted by $b_i \in \mathbb{R}$. These bias terms model the effects associated with specific users or items. Consequently, we increase the modeling power; hence, the predictions will be closer to reality. The prediction of the rating given by user u to item i is now defined as $\hat{r}_{ui} = \mu + b_u + b_i + \boldsymbol{p}_u^T\boldsymbol{q}_i$ (μ is the global rating mean), which is an improvement of Eq. (1). Moreover, the regularization terms are introduced in order to avoid overfitting by controlling the magnitude of the vectors.

Algorithm 3 illustrates the update rules for IRSVD. It is worth mentioning that the set of observed entries is shuffled after each epoch. This enables better training since shuffling imposes different orders for indices pairs per epoch. Furthermore, we decided to initialize the bias terms according to Eq. (4) and (5) in the Appendices.

Algorithm 3 Improved Regularized SVD

```
Input: $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, $\lambda_1, \lambda_2, \eta, \mu \in \mathbb{R}^+$, $k, Ep \in \mathbb{N}$, $\boldsymbol{B}_u \in \mathbb{R}^n$, $\boldsymbol{B}_i \in \mathbb{R}^m$ and $\mathcal{W}$
Output: Optimal $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{B}_u$ and $\boldsymbol{B}_i$
Initialize $\boldsymbol{P} \in \mathbb{R}^{k \times n}$ and $\boldsymbol{Q} \in \mathbb{R}^{k \times m}$ by drawing numbers from a normal distribution with $\mu = 0$ and $\sigma = \frac{1}{k}$
for $t$ in $1...Ep$ do
    Randomly shuffle $\mathcal{W}$
    for all $(u, i) \in \mathcal{W}$ do
        $e = a_{ui} - \mu - b_u - b_i - \boldsymbol{p}_u^T\boldsymbol{q}_i$
        $b_u = b_u + \eta(e - \lambda_2 b_u)$
        $b_i = b_i + \eta(e - \lambda_2 b_i)$
        $\boldsymbol{p}_u = \boldsymbol{p}_u + \eta(e\boldsymbol{q}_i - \lambda_1\boldsymbol{p}_u)$
        $\boldsymbol{q}_i = \boldsymbol{q}_i + \eta(e\boldsymbol{p}_u - \lambda_1\boldsymbol{q}_i)$
    end for
end for
```

IV. Ensemble of Methods

So far, we have presented various individual models. A natural question to ask is whether we can combine the predictive power of these methods. *Ensembling* is a simple technique to achieve this. We used a variant of ensembling, commonly referred to as *Blending*, to combine our models (baselines included). Assuming we have x models and the reconstructed matrix by the j-th model is denoted by $\hat{\boldsymbol{A}}^{(j)}$. Then the final prediction is the weighted linear combination

$$\hat{\boldsymbol{A}} = w_0 + \sum_{j=1}^{x} w_j\hat{\boldsymbol{A}}^{(j)} \tag{3}$$

where $\hat{\boldsymbol{A}}^{(j)}$ are fixed, and we learn the parameters w_j for $j = 0, 1, ..., x$. This reduces to solving a least-squares problem. However, if we use the training data on which $\hat{\boldsymbol{A}}^{(j)}$ are trained, the ensemble is prone to overfitting. Hence, the following strategy is employed:

1) Split the dataset into training and a holdout set.

2) Train j-th model on the training set for all $j = 1, ..., x$, learn w_j on the holdout set for all $j = 0, ..., x$.

In order to further improve the performance with more accurate weights, a 10-fold cross-validation style process is employed. The ensemble algorithm is then modified to an averaging of obtained weights of 10 repetitions of the original blending procedure. Algorithm 4 illustrates in detail all steps of our new 10-fold ensembling function.

Algorithm 4 10-fold Ensemble

```
Input: $model^{(1)}, ..., model^{(x)}$ and $\mathcal{I}$
Output: Obtain $\hat{\boldsymbol{A}}$ as in Eq. (3) for models trained on $\mathcal{I}$ (entire dataset) with computed averaged weights $w_0, ..., w_x$
Obtain $\mathcal{I}_{train}^{(f)}$ and $\mathcal{I}_{val}^{(f)}$ for all $f = 1, ..., 10$ as in Sec II
for $f$ in $1...10$ do
    for $j$ in $1...x$ do
        Train $model^{(j)}$ on $\mathcal{I}_{train}^{(f)}$ to obtain $\hat{\boldsymbol{A}}^{(j)}$
        Let $\hat{r}_{ui}^{(j)}$ be the predictions of $\hat{\boldsymbol{A}}^{(j)}$
    end for
    Solve the linear system of equations
        $$r_{ui} = w_0^{(f)} + \sum_{j=1}^{x} w_j^{(f)} \hat{r}_{ui}^{(j)}$$
    for all $(u, i) \in \mathcal{I}_{val}^{(f)}$, with unknowns $w_0^{(f)}, ..., w_x^{(f)}$
end for
Compute $w_j = \sum_{f=1}^{10} w_j^{(f)}/10$ for all $j = 0, ..., x$
```

V. Experimental Results

As explained in Section II, we split the set of observed entries $\mathcal{I}$ into 10 disjoint folds. Each model is trained on set $\mathcal{I}_{train}^{(f)}$. The other set $\mathcal{I}_{val}^{(f)}$ is held to estimate the score of the model. The same process is repeated 10 times for all $f = 1, ..., 10$ in order to obtain the 10-fold cross-validation score for each model, i.e., the mean RMSE and its standard deviation. The same technique (with set $\mathcal{I}$) is used to identify the optimal hyperparameters by performing 10-fold cross-validation for different sets of parameters. In Appendix A we present the experimental results of this process. The optimal hyperparameters for each model are summarized in Table I.

Table I
Experimental Optimal Hyperparameters for Each Method

Method	Optimal Parameters
SVD	$k = 8$
ALS	$k = 3$, $Ep = 3$
GBias	$\lambda = 0.001$, $Ep = 5$
SVP	$k = 3$, $\eta = 5$, $Ep = 10$
SVT	$\eta = 1.2$, $\tau = 800$, $Ep = 12$
RSVD	$k = 325$, $\eta = 0.01$, $\lambda = 0.02$, $Ep = 15$
IRSVD	$k = 325$, $\eta = 0.01$, $\lambda_1 = 0.02$, $\lambda_2 = 0.05$, $Ep = 15$

Finally, we re-train each model on the entire set of observed entries $\mathcal{I}$ in order to leverage the power of the entire dataset. The obtained predictions $\hat{r}_{ui}^{(j)}$ for all $(u, i) \in \mathcal{I}$ and $j = 1, ..., 7$ are also provided to the blending algorithm. The blending algorithm will combine the predictions as discussed in detail in Section IV and Algorithm 4. The RMSE we received for each model on both cross-validation and public test set is illustrated in Table II. The public test score on $\mathcal{I}_{test}$ is obtained after submitting to Kaggle the predictions for the models trained on the entire set $\mathcal{I}$.

Table II
RMSE for different models

Model	Cross-Validation		Public Score
	Mean	Std	
	Baseline Models		
SVD	1.00799	0.00178	1.00511
ALS	0.98866	0.00195	0.98609
	New Models		
GBias	0.99959	0.00217	0.99823
SVP	0.99244	0.00199	0.99088
SVT	0.98979	0.00179	0.98688
RSVD	0.98996	0.00196	0.98686
IRSVD	0.98111	0.00198	0.97772
Ensemble	-	-	**0.97417**

VI. Evaluation

As shown in Table II, the blending algorithm had the best score by significantly outperforming both baseline and the best individual methods. It is worth mentioning that although only IRSVD was an improvement of both baselines, while the rest outperformed only SVD (baseline), blending their results contributed to a more powerful model. This reinforces the belief that different models can contribute their own merit to the final solution. Moreover, comparing the cross-validation and public scores, we conclude that all methods generalize well on previously unavailable entries, i.e., on $\mathcal{I}_{test}$. Additionally, we noticed an improvement in the score on the public set compared to cross-validation, which we attribute to the fact that models were trained on the entire set of observed entries $\mathcal{I}$, while cross-validation always had a holdout fold.

As mentioned at the beginning, the sparseness of an observed matrix imposes a monumental challenge for reconstruction. Our final observation indicates that denser matrices enhance the usefulness of our models.

VII. Summary

In summarizing this work, we approached the recommender system problem using various matrix factorization methods. Our best individual method outperformed both baseline methods, while the power of the blending algorithm managed to further boost the effectiveness of our system.

REFERENCES

[1] B. Smith and G. Linden, “Two decades of recommender systems at amazon.com,” *IEEE Internet Computing*, vol. 21, pp. 12–18, 05 2017.

[2] V. Kelma and A. Laub, “The singular value decomposition: Its computation and some applications,” *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[3] C. Eckar and G. Young, “The approximation of one matrix by another of low rank,” *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[4] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[5] P. Jain, R. Meka, and I. Dhillon, “Guaranteed rank minimization via singular value projection,” *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[6] J.-F. Cai, E. J. Candès, and Z. Shen, “A singular value thresholding algorithm for matrix completion,” *SIAM Journal on optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[7] A. Paterek, “Improving regularized singular value decomposition for collaborative filtering,” *Proceedings of KDD Cup and Workshop*, 01 2007.

[8] “Netflix update: Try this at home,” https://sifter.org/simon/journal/20061211.html, (Accessed on 07/22/2022).

APPENDIX A. HYPERPARAMETER TUNING

An important aspect of building a powerful system is to find its best hyperparameters. This section presents some of our experimental results that led to selecting specific parameters for each model. The results are based on the technique of 10-fold cross-validation on the entire set $\mathcal{I}$.

A. *Singular Value Decomposition (SVD)*

We found that performing SVD on normalized $\boldsymbol{A}$ as described in Section III-B and then reverting normalization yields better results than the unnormalized version. Figure 1 shows the cross-validation score for different selection of k (number of singular values to keep). It was found that the optimal value for k is 8.

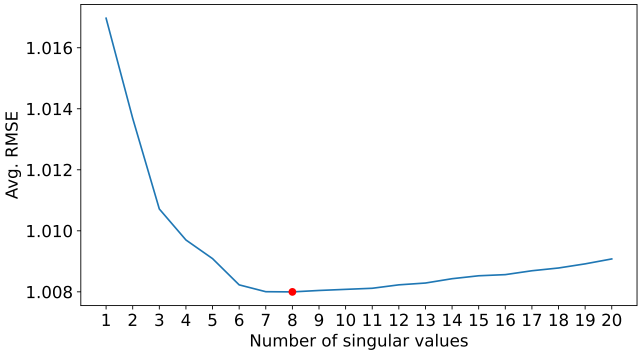

Figure 1. Hyperparameter k tuning for SVD.

B. *Alternating Least Squares (ALS)*

We fixed the proposed baseline parameters k and λ, i.e., $k = 3$ and $\lambda = 0.1$. The experiment illustrated in Figure 2 was contacted in order to find the number of epochs required until the algorithm converges. It was found that $Ep = 3$ is sufficient, while for more than three epochs, the RMSE error increases (overfitting phenomenon). The algorithmic procedure of ALS to minimize its objective (Eq. (2)) is shown in Algorithm 5.

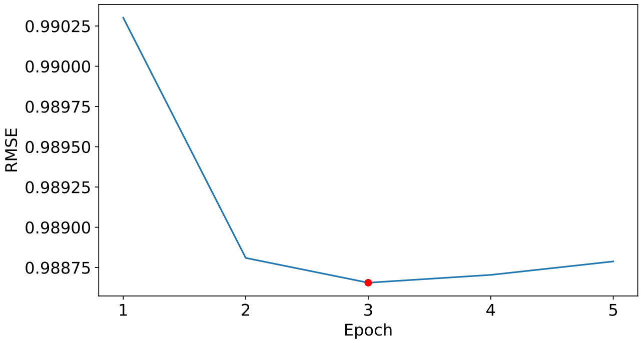

Figure 2. Hyperparameter Ep tuning for ALS.

Algorithm 5 ALS

```
Input: $\boldsymbol{A} \in \mathbb{R}^{n\times m}$, $k \in \mathbb{N}$, $\boldsymbol{P} \in \mathbb{R}^{k\times n}$, $\boldsymbol{Q} \in \mathbb{R}^{k\times m}$ and $\mathcal{W}$
Output: Optimal $\boldsymbol{P}$ and $\boldsymbol{Q}$
while not convergent do
    for $u$ in $1 \ldots n$ do
        $\boldsymbol{p}_u = (\sum_{i:(u,i)\in\mathcal{W}} \boldsymbol{q}_i \boldsymbol{q}_i^T + \lambda \boldsymbol{I}_k)^{-1} \sum_{i:(u,i)\in\mathcal{W}} a_{ui} \boldsymbol{q}_i$
    end for
    for $i$ in $1 \ldots m$ do
        $\boldsymbol{q}_i = (\sum_{u:(u,i)\in\mathcal{W}} \boldsymbol{p}_u \boldsymbol{p}_u^T + \lambda \boldsymbol{I}_k)^{-1} \sum_{u:(u,i)\in\mathcal{W}} a_{ui} \boldsymbol{p}_u$
    end for
end while
```

C. *Global Bias (GBias)*

The GBias method that is inspired by ALS to minimize its objective can be considered as the least powerful method that we introduced. Even when matrix $\boldsymbol{A}$ is fully observed, this method may not be able to achieve exact reconstruction since it attempts to reconstruct an entry based only on the information of the biases of the user and the item. Furthermore, experiments have shown that keeping the regularization factor as small as possible improves its performance. The improvement was negligible for regularization factor $\lambda \leq 0.001$; therefore, we decided to select 0.001 as its value.

The vectors $\boldsymbol{B}_u \in \mathbb{R}^n$ and $\boldsymbol{B}_i \in \mathbb{R}^m$ in GBias are initialized as follows. Let $\mathcal{W}$ be an instance of observed entries, then

$$\mu = \frac{\sum_{(u,i)\in\mathcal{W}} a_{ui}}{|\mathcal{W}|}$$

$$\mu_{(u,.)} = \frac{\sum_{i:(u,i)\in\mathcal{W}} a_{ui}}{|\mathcal{W}_{i:(u,i)}|}$$

$$\mu_{(.,i)} = \frac{\sum_{u:(u,i)\in\mathcal{W}} a_{ui}}{|\mathcal{W}_{u:(u,i)}|}$$

$$b_u = \mu_{(u,.)} - \frac{\sum_{t=1}^{n} \mu_{(t,.)}}{n}, \text{for } u = 1, \ldots n \tag{4}$$

$$b_i = \mu_{(.,i)} - \frac{\sum_{t=1}^{m} \mu_{(.,i)}}{m}, \text{for } i = 1, \ldots, m \tag{5}$$

where μ is the global rating mean, $\mu_{(u,.)}$ is user's u rating mean, $\mu_{(.,i)}$ is item's i rating mean, b_u is user's u rating bias and b_i is item's i rating bias.

D. *Singular Value Projection (SVP)*

As in the case of SVD, we found that using normalized $\boldsymbol{A}$ yields better results. Figure 3 depicts the cross-validation score for different selection of k (the projection rank). In contrast with SVD, the best value for k was found to be 3, and after 20 epochs, the algorithm converges. Furthermore, note that the value of learning rate η was fixed to 5 as suggested in [5]

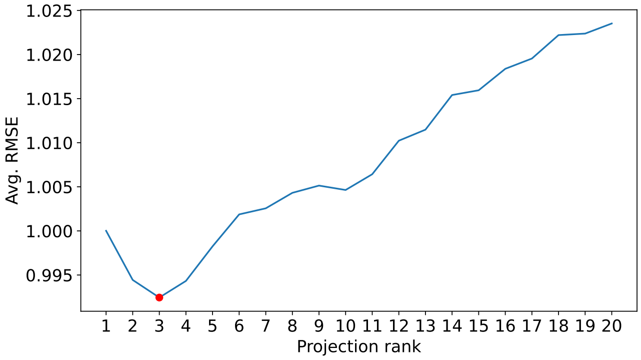

Figure 3. Hyperparameter k tuning for SVP.

E. *Singular Value Thresholding (SVT)*

For this method, we had to tune the parameter τ of the shrinkage operator. As in the methods mentioned above, we found that using normalized $\boldsymbol{A}$ yields better results. Figure 4 shows the cross-validation score for different selection of τ (the shrinkage value). We found that if we fix the learning rate η to 1.2 as suggested in [6], the best value for τ is 800, and it converges after 12 epochs.

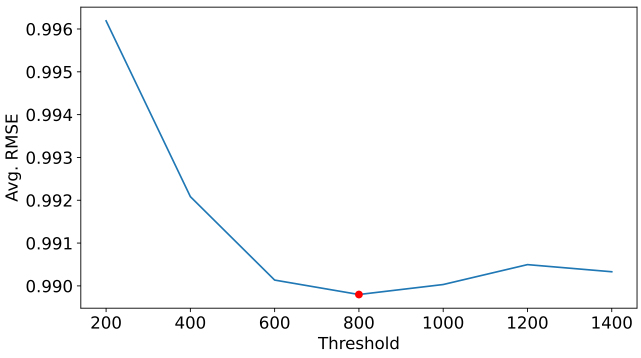

Figure 4. Hyperparameter τ tuning for SVT.

F. *Regularized SVD (RSVD)*

Figure 5 presents the cross-validation score for different selection of k. The model with 325 features had the best score. For this experiment, we fixed the learning rate η and regularization factor λ to 0.01 and 0.02, respectively. The regularization factor value was suggested in [7]. Moreover, the number of epochs required to converge was found to be 15.

Algorithm 6 illustrates the update rules for RSVD, which are very similar to IRSVD presented in Algorithm 3.

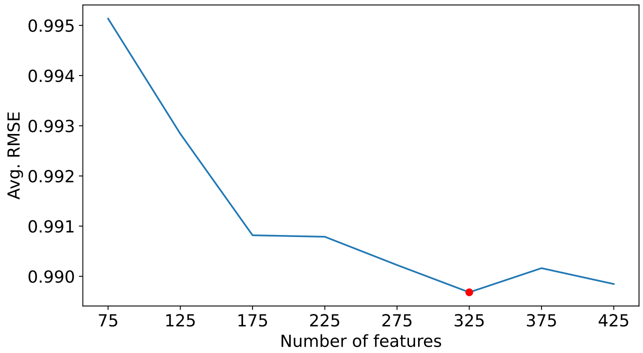

Figure 5. Hyperparameter k tuning for RSVD.

Algorithm 6 Regularized SVD

Input: $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, $\lambda, \eta \in \mathbb{R}^{+}$, $k, Ep \in \mathbb{N}$ and $\mathcal{W}$

Output: Optimal $\boldsymbol{P}$ and $\boldsymbol{Q}$

```
Initialize $\boldsymbol{P} \in \mathbb{R}^{k \times n}$ and $\boldsymbol{Q} \in \mathbb{R}^{k \times m}$ by drawing numbers from a normal distribution with $\mu = 0$ and $\sigma = \frac{1}{k}$
for $t$ in $1 \ldots Ep$ do
    Randomly shuffle $\mathcal{W}$
    for all $(u, i) \in \mathcal{W}$ do
        $e = a_{ui} - \boldsymbol{p}_u^T \boldsymbol{q}_i$
        $\boldsymbol{p}_u = \boldsymbol{p}_u + \eta(e\boldsymbol{q}_i - \lambda\boldsymbol{p}_u)$
        $\boldsymbol{q}_i = \boldsymbol{q}_i + \eta(e\boldsymbol{p}_u - \lambda\boldsymbol{q}_i)$
    end for
end for
```

G. Improved Regularized SVD (IRSVD)

In this method, we made an unorthodox decision regarding the selection of the hyperparameter k, which represents the features of users and items. Figure 6 depicts the cross-validation score for different selection of k. While we observed that having more features gives a small boost to the model's performance, we decided to stop at 325 features. With this number of features and onwards, the RMSE starts stabilizing. Since the goal is also the training speed of our model and the number of features is getting rapidly closer to $m = 1000$, (the number of items) the decision was to stop at $k = 325$. For this experiment, we fixed the learning rate η, the first regularization factor λ_1 and the second regularization factor λ_2 to 0.01, 0.02 and 0.05, respectively. Both regularization factor values were suggested in [7]. Finally, the number of epochs required to converge was found to be 15, as is shown in Figure 7.

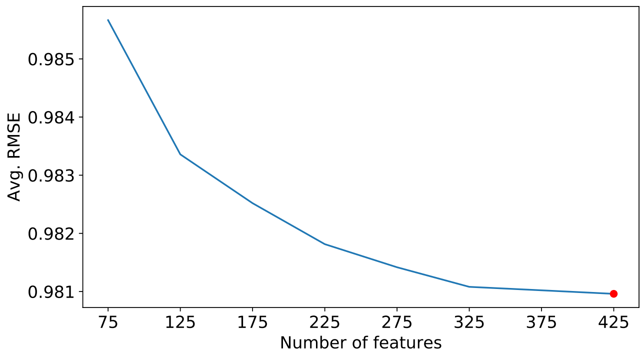

Figure 6. Hyperparameter k tuning for IRSVD.

Moreover, we decided to present the overfitting phenomenon on the training data for this method. Overfitting occurs when a model learns how to reproduce training data very well at the cost of missing the ability to generalize on unknown entries. Figure 7 illustrates overfitting during training of IRSVD with the parameters suggested above. It is clear that after epoch 15, the test RMSE increases while the train RMSE rapidly decreases.

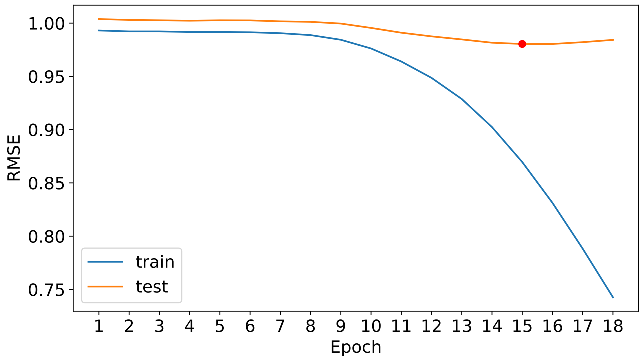

Figure 7. Overfitting phenomenon on IRSVD when $k = 325$.