Ensembling Latent Factor Models for Collaborative Filtering

Sotiris Anagnostidis, Adamos Solomou, Ioannis Sachinoglou, Konstantinos Andrikopoulos
Group: sasglentame, Department of Computer Science, ETH Zürich, Switzerland

***Abstract*—Recommender systems often rely on collaborative filtering to provide users with personalised product recommendations. Latent factor models have proven to be a very successful approach to collaborative filtering. In this work, we consider several latent factor models, including classical techniques relying on matrix factorization as well as modern approaches based on deep learning. Our system, combines the predictions of several models in an optimal fashion, surpassing the performance of each individual model.**

I. Introduction

In recent years, the tremendous growth of e-commerce and social media platforms has established the need for *recommender systems* capable of providing personalised product recommendations to individual users. Such systems are often based on *Collaborative Filtering* [1], a technique relying on past user behaviour, aiming to identify relationships between users and products.

Collaborative filtering breaks down into two primary approaches, namely neighbourhood methods and latent factor models. Neighbourhood methods focus on computing the relationships between either the users or the items [2], [3]. In this setting, recommendations for a user are made based on recorder preferences of like minded users. Instead, latent factor models aim to explain the observed user-item interactions by representing users and items in terms of latent variables [4]. Once these variables are inferred they can be used to provide users with personalised recommendations.

In this project, we consider several latent factor models to improve upon baseline methods such as the Singular Value Decomposition and Alternating Least Squares. Thereafter, we employ a linear blending technique to combine individual model predictions by learning optimal blending coefficients. Doing so, we achieve an absolute improvement in performance compared to each individual model.

II. Preliminaries

We are given a set of $N = 1,176,952$ integer movie ratings, ranging from 1 to 5, that are assigned by $m = 10,000$ users to $n = 1,000$ movies. A rating r_{ui} indicates the preference by user u of item i. Let $\mathcal{I} = \{(u, i) : r_{ui} \text{ is known}\}$ be the set of user and movie indices for which the ratings are known. The provided data $\mathcal{I}$ are split into two disjoint subsets, namely $\mathcal{I}_{\text{train}}$ and $\mathcal{I}_{\text{blend}}$. The former consists of 90% of the data and is used for training the individual models whereas the latter consists of the remaining 10% of the data and is used for learning optimal blending weights.

Typically, a sparse matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is used to represent the observed ratings. Formally, $\forall\,(u, i) \in \mathcal{I}_{\text{train}}$ let $A_{ui} = r_{ui}$. In this setting, the task of collaborative filtering is to predict as accurately as possible the missing entries in the user-item matrix. Doing so, we can then provide item recommendations to individual users. A major challenge in collaborative filtering is the sparseness of the observed data. While there exist a total of $m\times n = 10,000,000$ ratings, we only observe 11.8% of them, making the problem particularly challenging. We will quantify the quality of our predictions based on the root mean squared error (RMSE) function between the true and observed ratings. For a given set of observations $\mathcal{J}$, let

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

where $\hat{r}_{ui}$ denotes the estimate of r_{ui}.

III. Models and Methods

In this section, we first present two baseline models, followed by the latent factor models used in our collaborative filtering approach. Optimal hyperparameters for each model are determined using 10-fold cross-validation on $\mathcal{I}_{\text{train}}$.

A. Overview

A latent factor model associates each user u with a set of user factors $\boldsymbol{p}_u \in \mathbb{R}^k$ and each item i with a set of item factors $\boldsymbol{q}_i \in \mathbb{R}^k$. In the case of movies, some of these factors could correspond to movie genres such as comedy, drama, action etc. For each item i, the elements of $\boldsymbol{q}_i$ quantify the extent to which the item possesses these factors. Similarly, for each user u, the elements of $\boldsymbol{p}_u$ measure the level of interest that the user has to each of these factors. In this framework, user-item interactions are modeled by inner products in the latent space, leading to the following prediction rule

$$\hat{r}_{ui} = \boldsymbol{p}_u^T\boldsymbol{q}_i. \tag{2}$$

We can collectively denote the user factors by a matrix $\boldsymbol{P} \in \mathbb{R}^{m\times k}$, such that the row of $\boldsymbol{P}$ indexed by u corresponds to $\boldsymbol{p}_u^T$. Similarly, we may represent the item factors by a matrix $\boldsymbol{Q} \in \mathbb{R}^{k\times n}$, where the i-th column of $\boldsymbol{Q}$ corresponds to the vector $\boldsymbol{q}_i$. In this setting, the prediction rule (2) is given by $\hat{\boldsymbol{A}} = \boldsymbol{PQ}$. The goal of collaborative filtering is to infer the matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$ from the observed ratings. Once they are learnt, they can be used to predict the missing entries in the original user-item matrix.

B. Imputing Missing Ratings

Some of the methods presented in this paper operate on a user-item matrix with no missing entries. Since the observed matrix is extremely sparse, in order to apply those methods we must first impute the missing entries of matrix $\boldsymbol{A}$. We will denote the imputed user-item matrix by $\widetilde{\boldsymbol{A}}$. Thus, $\forall\,(u,i) \in \mathcal{I}_{\text{train}}$, let $\widetilde{A}_{ui} = A_{ui}$. We then consider the following imputation methods:

1) Zero values: For each $(u,i) \notin \mathcal{I}_{\text{train}}$ let $\widetilde{A}_{ui} = 0$.

2) Mean rating: For each $(u,i) \notin \mathcal{I}_{\text{train}}$, let $\widetilde{A}_{ui} = \mu$ where $\mu = \frac{1}{|\mathcal{I}_{\text{train}}|}\sum_{(u,i)\in\mathcal{I}_{\text{train}}} r_{ui}$.

3) Mean per item: Instead of using the global average, we impute missing ratings per item using the mean item rating. Formally, for each $i \in \{1,...,n\}$, let

$$\mu_i = \frac{1}{N_i}\sum_{u:(u,i)\in\mathcal{I}_{\text{train}}} r_{ui}$$

where N_i is the number of observed ratings for item i. Then, for each $i \in \{1,...,n\}$ and $\forall\, u : (u,i) \notin \mathcal{I}_{\text{train}}$, let $\widetilde{A}_{ui} = \mu_i$.

C. Vanilla SVD

Singular Value Decomposition (SVD) [5] is a widely used technique for matrix factorization. Any matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ can be decomposed into $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{m\times m}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times n}$. Matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal, whereas $\boldsymbol{\Sigma}$ has $\text{rank}(\boldsymbol{M})$ positive entries on the main diagonal[1] sorted in decreasing order of value.

We apply the SVD on the imputed user-item matrix to decompose it into $\widetilde{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. The modelling assumption introduced in Section III-A that $\hat{\boldsymbol{A}} = \boldsymbol{P}\boldsymbol{Q}$ implies that $\text{rank}(\hat{\boldsymbol{A}}) \leq k$. This motivates us to approximate $\widetilde{\boldsymbol{A}}$ by another matrix of low rank. The Eckart-Young theorem [6] states that the optimal (in terms of the Frobenius norm objective) rank k approximation of the matrix $\widetilde{\boldsymbol{A}}$ is given by $\widetilde{\boldsymbol{A}}_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k\boldsymbol{V}_k^T$, where $\boldsymbol{U}_k \in \mathbb{R}^{m\times k}$, $\boldsymbol{\Sigma}_k \in \mathbb{R}^{k\times k}$ and $\boldsymbol{V}_k \in \mathbb{R}^{n\times k}$. $\boldsymbol{U}_k$ and $\boldsymbol{V}_k$ correspond to the first k columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ respectively and $\boldsymbol{\Sigma}_k$ to the $k\times k$ sub-matrix of $\boldsymbol{\Sigma}$ containing the k largest singular values. Equivalently, we can write $\widetilde{\boldsymbol{A}}_k = (\boldsymbol{U}_k\boldsymbol{\Sigma}_k^{1/2})\cdot(\boldsymbol{\Sigma}_k^{1/2}\boldsymbol{V}_k^T) = \boldsymbol{P}\boldsymbol{Q} = \hat{\boldsymbol{A}}$.

D. Alternating Least Squares (ALS)

Alternating Least Squares (ALS) [4] is an iterative algorithm minimizing the following objective function

$$\sum_{(u,i)\in\mathcal{I}_{\text{train}}} (r_{ui} - \boldsymbol{p}_u^T\boldsymbol{q}_i)^2 + \lambda\sum_{u=1}^{m}\|\boldsymbol{p}_u\|^2 + \lambda\sum_{i=1}^{n}\|\boldsymbol{q}_i\|^2 \quad (3)$$

where $\lambda > 0$ is the regularization strength. The procedure is illustrated in Algorithm 3 in the Appendix.

[1]The main diagonal is defined by the elements Σ_{ij} such that $i = j$.

Algorithm 1: SVD Shrinkage

Input : $\mathcal{I}_{\text{train}}$, $L \in \mathbb{Z}_+$, $\tau \in \mathbb{R}_+$, $\widetilde{\boldsymbol{A}} \in \mathbb{R}^{m\times n}$
Output: Predicted user-item matrix $\hat{\boldsymbol{A}} \in \mathbb{R}^{m\times n}$

$\hat{\boldsymbol{A}} \leftarrow \widetilde{\boldsymbol{A}}$
for $l = 1$ **to** L **do**
  $\hat{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  $\boldsymbol{\Sigma} \leftarrow \mathcal{D}_\tau(\boldsymbol{\Sigma}) = \text{diag}(\max\{0, \Sigma_{jj} - \tau\})$
  $\hat{\boldsymbol{A}} \leftarrow \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$
  foreach $(u,i) \in \mathcal{I}_{train}$ **do**
    $\hat{\boldsymbol{A}}_{ui} = \widetilde{\boldsymbol{A}}_{ui}$

E. SVD Shrinkage

We can improve upon the vanilla SVD by applying an iterative variant of the algorithm. The procedure is based on the *singular value shrinkage* operator introduced by Cai et al. [7] and is illustrated in Algorithm 1. At every iteration of the algorithm, we apply the soft-thresholding operator $\mathcal{D}_\tau$ to the singular values of $\hat{\boldsymbol{A}}$, effectively shrinking them towards zero. Before repeating, we set the observed ratings to their original value.

F. Bias Stochastic Gradient Descent (B-SGD)

The prediction rule given by Eq. (2) tries to capture the user-item interactions in terms of the inner product between their latent representation. According to Koren et al. [4] this model is insufficient to capture these interactions since a significant portion of the observed variation in the rating values comes from effects associated with either the user or the items. Such effects are commonly referred to as *biases* and are independent of any user-item interactions. We follow the proposal of Arkadiusz Paterek [8] and associate each user u and item i with a bias term denoted by $b_u \in \mathbb{R}$ and $b_i \in \mathbb{R}$ respectively. The prediction rule is now given by $\hat{r}_{ui} = \boldsymbol{p}_u^T\boldsymbol{q}_i + b_u + b_i$.

To learn the latent factor vectors and biases, we follow the technique introduced in [8] and minimize a regularized squared error on the set of observed ratings:

$$\min_{\boldsymbol{p}_*,\boldsymbol{q}_*,b_*} \sum_{(u,i)\in\mathcal{I}_{\text{train}}} \big(r_{ui} - \boldsymbol{p}_u^T\boldsymbol{q}_i - b_u - b_i\big)^2 + \lambda_1\big(\|\boldsymbol{p}_u\|^2 + \|\boldsymbol{q}_i\|^2\big) + \lambda_2\big(b_u + b_i - \mu\big)^2 \quad (4)$$

where μ is the average value of the training ratings and $\lambda_1 > 0, \lambda_2 > 0$ are hyperparameters controlling regularization. In the objective above, $\sum_{(u,i)\in\mathcal{I}_{\text{train}}}(r_{ui} - \hat{r}_{ui})^2$ is the data fitting term that strives to find model parameters that fit the observed ratings. The two remaining components are regularization terms that penalize the magnitude of the parameters to avoid overfitting.

A popularized technique to solve the aforementioned optimization problem is Stochastic Gradient Descent (SGD) [9].

Algorithm 2: Bias-SGD Training Process

Input : $\mathcal{I}_{\text{train}}$, $k \in \mathbb{R}_+$, $L \in \mathbb{Z}_+$, $\mu \in \mathbb{R}$, $\eta \in \mathbb{R}_+$, $\lambda_1 \in \mathbb{R}_+, \lambda_2 \in \mathbb{R}_+$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$

Output: Optimal model parameters $\boldsymbol{p}_u$, $\boldsymbol{q}_i$, b_u, b_i

1 Randomly initialize $\boldsymbol{q}_i, \boldsymbol{p}_u \in \mathbb{R}^k$ for all i and u
2 $\forall u : b_u \leftarrow 0, \forall i : b_i \leftarrow 0$
3 **for** $l = 1$ **to** L **do**
4 Randomly shuffle $\mathcal{I}_{\text{train}}$
5 **if** $l (\text{mod } 5) = 0$ **then** $\eta \leftarrow \eta \times \frac{2}{3}$
6 **foreach** $(u, i) \in \mathcal{I}_{train}$ **do**
7 $e_{ui} = A_{ui} - \boldsymbol{p}_u^T \boldsymbol{q}_i - b_u - b_i$
8 $\boldsymbol{p}_u \leftarrow \boldsymbol{p}_u + \eta(e_{ui}\boldsymbol{q}_i - \lambda_1 \boldsymbol{p}_u)$
9 $\boldsymbol{q}_i \leftarrow \boldsymbol{q}_i + \eta(e_{ui}\boldsymbol{p}_u - \lambda_1 \boldsymbol{q}_i)$
10 $b_u \leftarrow b_u + \eta(e_{ui} - \lambda_2(b_u + b_i - \mu))$
11 $b_i \leftarrow b_i + \eta(e_{ui} - \lambda_2(b_u + b_i - \mu))$

The procedure is illustrated in Algorithm 2. We train the model for $L = 50$ training epochs. At each epoch, we randomly shuffle the training set $\mathcal{I}_{\text{train}}$ and iterate over the training ratings, updating model parameters according to lines $8 - 11$. Moreover, every five epochs, we decrease the learning rate by a factor of $2/3$ (line 5) starting with an initial learning rate of $\eta = 0.05$.

G. Neural Collaborative Filtering (NCF)

So far, we focused on learning the latent factors of users and movies and used a simple rule, namely the inner product between their latent representation possibly with some additive biases, to predict user-item interactions. Given the function approximation capabilities of artificial neural networks [10], [11], we adopt the Neural Collaborative Filtering (NCF) approach proposed by He at al. [12] to jointly learn both the user-item latent representations as well as the prediction rule of user-item interactions.

The neural network architecture is illustrated in Figure 1. Input to the network is the user and item indices denoted by u and i respectively. Next, is an embedding layer that associates each user and item with their respective latent representation. We pass the user and item representations through two fully-connected hidden layers with ReLU activations [13] and dropout [14]. Subsequently, we branch and perform both a concatenation as well as an element-wise multiplication between the transformed user and item representations. Finally, the two resulting vectors are concatenated and provided as input to a feedforward network with a single hidden layer and an output layer with five units and a softmax activation.

The j^{th} softmax output, denoted by y_j, can be thought of as the probability that the user u would assign a rating j to item i. For a given user u and item i, the final rating prediction $\hat{r}_{ui}$ is computed as a weighted sum between the output probabilities y_j and the rating values $j \in \{1, ..., 5\}$.

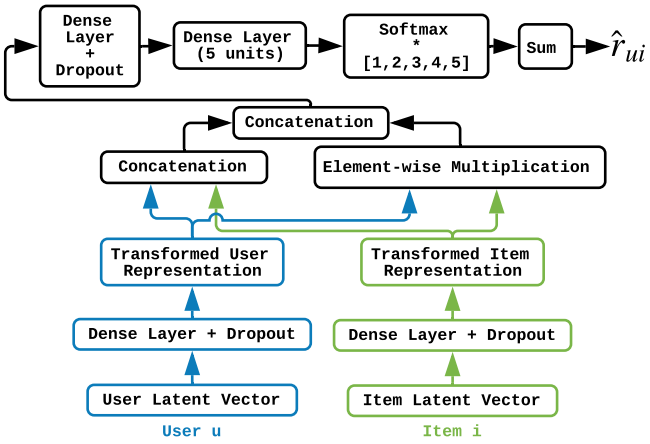

Figure 1. Architecture of neural collaborative filtering.

Although training the model by means of minimizing the Mean Squared Error (MSE) between the true and predicted ratings seems to be a natural choice, we found that applying and minimizing a Cross-Entropy loss at the output of the softmax unit leads to better results. Recent research on matrix completion for recommender systems has also shown competitive performance while treating the recommendation problem as a classification task during training [15].

H. Autoencoder

An autoencoder is a type of neural network that is used to learn efficient, low-dimensional data representations in an unsupervised manner [16]. It implements two transformations: first, the encoder $g_e : \mathbb{R}^d \rightarrow \mathbb{R}^k$ and secondly, the decoder $g_d : \mathbb{R}^k \rightarrow \mathbb{R}^d$, where $k < d$. The network is trained by minimizing the squared reconstruction loss between the input $\boldsymbol{x}$ and output $\hat{\boldsymbol{x}} = g_d(g_e(\boldsymbol{x}))$, thereafter learning an intermediate $k-$dimensional representation for $\boldsymbol{x}$.

Autoencoders have proven to be very effective in the context of collaborative filtering [17], [18], [19]. We adopt the user-based autoencoder approach presented in [17], [19] to learn a latent representation for each user. This approach differs from the ones presented previously in the sense that we no longer learn a representation for items. Both the encoder and decoder consist of feedforward neural networks with ReLU activations. We found that use of regularization techniques such as dropout and L_2 regularization is necessary to avoid overfitting. We apply dropout at the input of the autoencoder as well as at the output of every hidden layer in both the encoder and decoder networks. To further imporve the performance of the autoencoder we apply different bias for each user at the final layer.

During inference, the model takes as input a user u represented by its imputed vector of ratings. We use zero value imputation, which in combination with the ReLU activation drives the network to focus only on reconstructing the known ratings and not the missing ones. While the input of the

network is very sparse, the output of the decoder, is dense and contains the user's rating predictions for all the items. We train the model by minimizing the squared reconstruction loss considering only the contribution of the observed ratings of each user. To further boost the performance and account for randomness, we repeat the training process ten times and average the predictions.

I. Blending Predictions

Our final rating predictions are produced by combining the predictions of the four main methods introduced in this section, namely SVD Shrinkage, B-SGD, NCF and the Autoencoder model. Such *blending* approaches have proven to be very effective in the context of collaborative filtering [20]. The motivation behind this approach is that different methods might be better than others at predicting different parts of the user-item matrix. We consider the following blending scheme. Given a set of predicted ratings $\hat{r}_{ui}^{(j)}$ for $j = 1, ..., 4$ (one for each method), we compute the final prediction according to

$$\hat{r}_{ui} = w_0 + \sum_{j=1}^{4} w_j \hat{r}_{ui}^{(j)}. \tag{5}$$

Hence, our goal is to infer optimal blending weights w_j. To that end, we use a simple linear regression model which receives a matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{I}_{\text{blend}}| \times 4}$ and a vector $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{I}_{\text{blend}}|}$ and computes $w_0, ..., w_4$. Each row of $\boldsymbol{X}$ holds the predicted ratings $\hat{r}_{ui}^{(j)}$ for a pair $(u, i) \in \mathcal{I}_{\text{blend}}$ and $j = 1, ..., 4$. On the other hand, the vector $\boldsymbol{y}$ holds the known ratings r_{ui} for $(u, i) \in \mathcal{I}_{\text{blend}}$. We train the linear regression model using the holdout dataset $\mathcal{I}_{\text{blend}}$ to obtain optimal blending weights. Once these are inferred they can be used to produce final rating predictions for other user-item pairs.

IV. Experiments and Results

We train and evaluate each individual model on the training split specified in Section II. The training set $\mathcal{I}_{\text{train}}$ is also used to perform 10-fold cross-validation for hyperparameter selection. Experimental results from this process are presented in Appendix A and the optimal configuration for each model is outlined in Table I. Finally, we re-train each model (with its optimal configuration) on the entire training set $\mathcal{I}_{\text{train}}$ and compute predictions $\hat{r}_{ui}^{(j)}$ for all $(u, i) \in \mathcal{I}_{\text{blend}}$ and $j = 1, ..., 4$, that are subsequently provided as input to the blending algorithm. The RMSE achieved by each model on both cross-validation and public test set is presented in Table II. The public test error for each model is obtained from the Kaggle website after re-training using the entire training split $\mathcal{I}_{\text{train}}$.

Table I
OPTIMAL CONFIGURATION FOR EACH MODEL.

Model	Optimal Parameters
SVD	$k = 12$
ALS	$k = 12, \lambda = 25$
SVD Shrinkage	$\tau = 38, L = 15$
B-SGD	$k = 9, L = 50, \eta = 0.05, \lambda_1 = 0.079, \lambda_2 = 0.05$
NCF	k=96, layer sizes=[1024, 5]
Autoencoder	$k = 12$, dropout=0.5

Table II
RMSE ACHIEVED BY DIFFERENT MODELS.

Model	Cross-Validation	Public Test
Baseline Models		
SVD	1.00812	1.00432
ALS	0.99176	0.98644
Our Models		
SVD Shrinkage	0.98595	0.98320
B-SGD	0.98317	0.97976
NCF	0.98613	0.98326
Autoencoder	0.98218	0.97915
Ensemble	N/A	**0.97231**

V. Discussion

As Table II indicates, all models generalize well to the public test set and the four models taking part in the ensemble perform better compared to the two baseline models. The improvement resulting from using the variant of the vanilla SVD method is also evident. Moreover, we observe that blending predictions leads to a better RMSE on the public test set compared to any individual method. Effectively, blending individual model predictions allows to combine the individual merits of each method into a single model, increasing its predictive capabilities. Finally, the error difference between the cross-validation and the public test set can be attributed to the increased training set.

As mentioned in the beginning, a great challenge in collaborative filtering is the sparseness of the observed data. Many users have registered very few ratings, resulting in a very sparse representation. While imputation provides an initial estimate to these missing entries having more data would be very beneficial for all of our methods.

VI. Summary

In this work, we approached the recommendation problem using a variety of latent factor models. We consider both "shallow" (SVD Shrinkage, B-SGD) as well as deep learning methods (NCF, Autoencoder), outperforming the two baseline models. Finally, we leverage the power of ensemble methods to optimally combine individual model predictions and further boost the performance of our system.

References

[1] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, no. 12, pp. 61–70, Dec. 1992. [Online]. Available: http://doi.acm.org/10.1145/138859.138867

[2] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: ACM, 2008, pp. 426–434. [Online]. Available: http://doi.acm.org/10.1145/1401890.1401944

[3] Z. Zhang, Y. Kudo, and T. Murai, "Neighbor selection for user-based collaborative filtering using covering-based rough sets," *Annals of Operations Research*, vol. 256, no. 2, pp. 359–374, Sep 2017. [Online]. Available: https://doi.org/10.1007/s10479-016-2367-1

[4] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1109/MC.2009.263

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

[6] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, Sep 1936. [Online]. Available: https://doi.org/10.1007/BF02288367

[7] J. Cai, E. Cands, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010. [Online]. Available: https://doi.org/10.1137/080738970

[8] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering." Proc. KDD Cup and Workshop, ACM Press, 2007, pp. 39–42.

[9] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186.

[10] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec 1989. [Online]. Available: https://doi.org/10.1007/BF02551274

[11] K. Hornik, M. B. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[12] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural Collaborative Filtering," *arXiv e-prints*, p. arXiv:1708.05031, Aug 2017.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudk, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313

[15] R. van den Berg, T. N. Kipf, and M. Welling, "Graph Convolutional Matrix Completion," *arXiv e-prints*, p. arXiv:1706.02263, Jun 2017.

[16] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006. [Online]. Available: http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google

[17] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15 Companion. New York, NY, USA: ACM, 2015, pp. 111–112. [Online]. Available: http://doi.acm.org/10.1145/2740908.2742726

[18] F. Strub and J. Mary, "Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs," in *NIPS Workshop on Machine Learning for eCommerce*, Montreal, Canada, Dec. 2015. [Online]. Available: https://hal.inria.fr/hal-01256422

[19] O. Kuchaiev and B. Ginsburg, "Training Deep AutoEncoders for Collaborative Filtering," *arXiv e-prints*, p. arXiv:1708.01715, Aug 2017.

[20] Y. Koren, "The bellkor solution to the netflix grand prize," 2009.

[21] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

APPENDIX A.
HYPERPARAMETER SELECTION

We present some of the results obtained during hyperparameter selection for individual models. The results are based on 10-fold cross-validation on the training split.

A. *Vanilla SVD*

We found that using the mean per item imputation method to initialize the missing entries in $\boldsymbol{A}$ yields better results compared to the two other approaches. Figure 2 shows the cross-validation RMSE for different values of k, the number of singular values to be preserved during truncation of the SVD. We found $k = 12$ to be optimal.

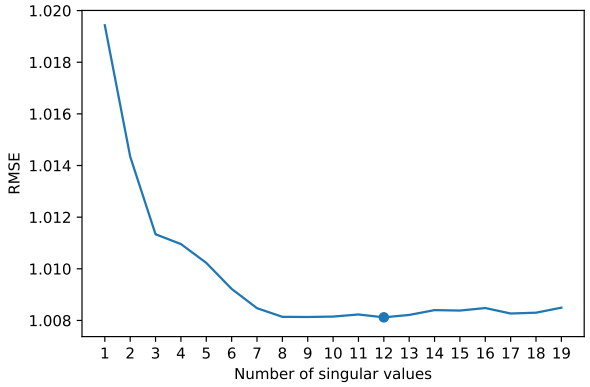

Figure 2. Hyperparameter selection for vanilla SVD.

B. *Alternating Least Squares*

Algorithm 3 illustrates the process for minimizing the objective in Eq. (3). We used the optimal k derived from the vanilla SVD (i.e. $k = 12$) and optimize for different values of the regularization strength λ. Results from this process are presented in Figure 3. We found $\lambda = 25$ to be optimal.

Algorithm 3: ALS Training Process

Input : $\mathcal{I}_{\text{train}}$, $k \in \mathbb{R}_+$, $\lambda \in \mathbb{R}_+$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$
Output: Optimal model parameters $\boldsymbol{P}, \boldsymbol{Q}$

Initialize $\boldsymbol{P} \in \mathbb{R}^{m \times k}, \boldsymbol{Q} \in \mathbb{R}^{k \times n}$
while *not convergent* **do**
  for $u = 1$ **to** m **do**
    $\boldsymbol{p}_u = \Big(\sum_{i:(u,i) \in \mathcal{I}_{\text{train}}} \boldsymbol{q}_i \boldsymbol{q}_i^T + \lambda \boldsymbol{I}_k \Big)^{-1}$
      $\times \sum_{i:(u,i) \in \mathcal{I}_{\text{train}}} A_{ui} \boldsymbol{q}_i$
  for $i = 1$ **to** n **do**
    $\boldsymbol{q}_i = \Big(\sum_{u:(u,i) \in \mathcal{I}_{\text{train}}} \boldsymbol{p}_u \boldsymbol{p}_u^T + \lambda \boldsymbol{I}_k \Big)^{-1}$
      $\times \sum_{u:(u,i) \in \mathcal{I}_{\text{train}}} A_{ui} \boldsymbol{p}_u$

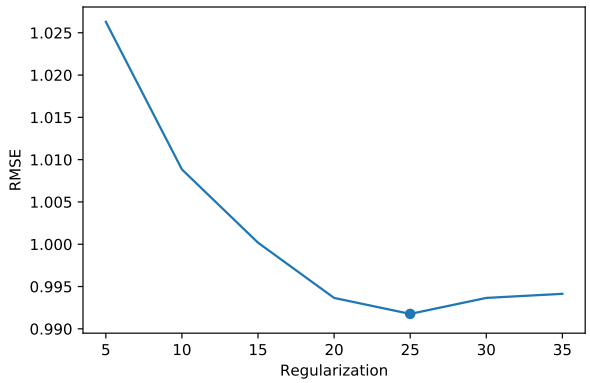

Figure 3. Cross-validation RMSE for different values of the parameterλ in ALS.

C. *SVD Shrinkage*

As in the case of vanilla SVD, we found that using the mean per item imputation method to initialize the missing entries in $\boldsymbol{A}$ yields better results compared to the two other approaches. Figure 4 shows the cross-validation RMSE as a function of the threshold level τ. We found $\tau = 38$ to be optimal. The best results for this best configuration are obtained using $L = 15$ iterations.

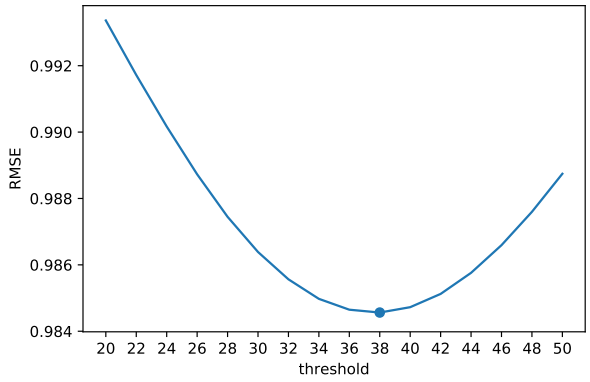

Figure 4. Hyperparameter selection for threshold SVD.

D. *Bias Stochastic Gradient Descent*

We use a grid search to find optimal values for the number of latent variables k and regularization strength λ_1. The results of this process are illustrated in Figure 5. Each square in the figure corresponds to a pair of (k, λ_1) values and the square colour indicates the RMSE achieved by this pair of

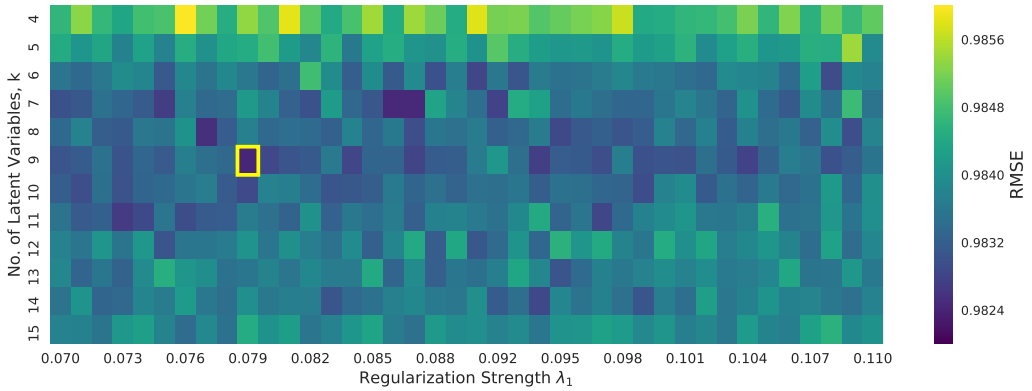

Figure 5. RMSE for different values of k and λ_1. Darker colours correspond to a lower cross-validation error whereas lighter colours correspond to a higher cross-validation error.

values. Darker colours correspond to a lower cross-validation error whereas lighter colours correspond to a higher cross-validation error. We found $k = 9$ and $\lambda_1 = 0.079$ to be optimal.

Finally, we search for optimal λ_2 setting the values of k and λ_1 to 9 and 0.079 respectively. As indicated in Figure 6 we found $\lambda_2 = 0.05$ to be optimal.

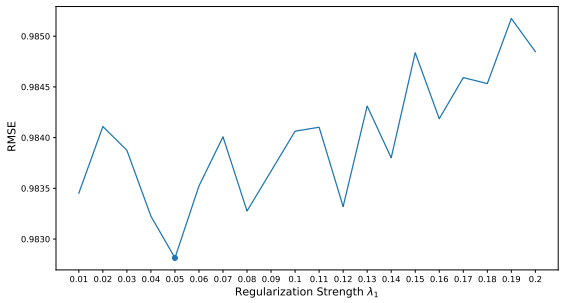

Figure 6. RMSE for different values of the regularization strength λ_2.

E. Neural Collaborative Filtering

At this section, we provide some additional details regarding our NCF approach. We elaborate upon some architectural choices and in particular discrepancies with the originally proposed "Neural Collaborative Filtering" framework [12].

We set the dimensionality of the latent space to $k = 96$ for both users and movies. We found that transforming these latent representations by passing them through two fully-connected layers with ReLU activation and dropout yields better results, possibly due to the robustness introduced by dropout. The first hidden layer increases the dimensionality to 256 and the second reduces it back to 96. We apply a dropout rate of 0.2 on the initial user and movie latent vector representations and a dropout rate of 0.1 to the output of each fully-connected layer.

Table III
RMSE ACHIEVED BY DIFFERENT NCF CONFIGURATIONS.

Model Configuration	**CV RMSE**
Dim. of Latent Space, $k = 16$	0.98800
Dim. of Latent Space, $k = 32$	0.98650
Dim. of Latent Space, $k = 64$	0.98661
Dim. of Latent Space, $k = 96$	0.98636
Non-linear transformation of initial latent variables through dense layers with ReLU activations & Dropout (final configuration)	**0.98613**
Same architecture with scalar output trained with MSE objective	0.99393

Moreover, we found that using both the latent vectors of the user and movies alone, as well as their (element-wise) multiplication leads to better results. He et al. [12] use two different embedding representations for the concatenation and multiplication layers in Figure 1. On the contrary, we use the same latent vectors for both functions, leading to better results. Finally, as stated in Section III-G, we treat the recommendation problem as a 5-class classification problem during training, minimizing a Cross-Entropy loss at the output of the softmax unit instead of using a MSE objective. We also experimented with modifying the Cross-Entropy loss to take into account the "distance" between different classes. However, this did not lead to further improvement in performance. We train the model using the ADADELTA [21] optimizer and an exponentially decay learning rate. Table III outlines same of the configurations used and their impact on the final score achieved.

F. Autoencoder

The most important parameter influencing the accuracy of the Autoencoder is the size of the hidden layer holding the k-dimensional latent representation for a user. In Figure 7 we demonstrate how this parameter affects the overall performance of the model. Note that the results presented in Figure 7 correspond to a single repetition of the experiment. The observed fluctuation is due to randomness introduced during the experiment. Nonetheless, we noticed that using a hidden layer of size 6 was persistently found to be optimal amongst the rest of the parameter values.

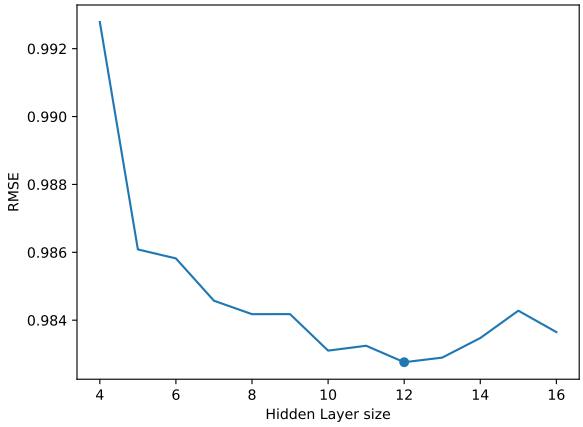

Figure 7. Hyperparameter selection for Autoencoder.