# An Ensemble Based Approach to Collaborative Filtering

Xiyi Chen, Pengxi Liu, Jiasong Guo, Chuhao Feng
Group: Juanwang, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Collaborative filtering plays an increasingly vital role in recommendation system to enable personalizing in product recommendation, and researchers tend to take attempt on ensemble-based models to achieve higher predictive performance. In this report, we propose a novel blending method for collaborative filtering based on linear regression, gradient boosting and XGBoost that is built upon Iterative SVD, BFM, NCF and etc. Our method finally achieves local CV score 0.9674 and public test score 0.9656, which could be regarded as a significant improvement comparing with our baseline method.**

## I. Introduction

Recommendation systems have become the core competitiveness of many large companies today, such as Amazon's product recommendation [1], Tiktok's video recommendation and Netflix's movie recommendation [2]. Most of the current mainstream recommendation systems use collaborative filtering as the core [3]. Collaborative filtering utilizes known browsing information, such as the current user's historical preferences and evaluations of the current transaction, to estimate the current user's liking for this data.

There are currently many approaches to estimate this problem using different models and from different aspects, however, many of them are just on par with the baseline.

In this paper, we propose an ensemble based method to perform collaborative filtering. We train different individual models and finally concatenate them to create an ensemble model. Experiments show that our ensemble model has better performance than the two baseline models and all individual models. Our implementation is publicly available at https://github.com/xiyichen/collaborative-filtering.

## II. Preliminaries

### A. Problem Settings

In this project, we are provided with $1,176,952$ integer ratings, from 1 to 5, given by $n = 10,000$ users to $m = 1,000$ movies. We convert these known ratings into a sparse matrix $A$ with $n$ rows and $m$ columns, where $A_{ij}$ represents the preference by user $i$ to movie $j$. It is a very sparse matrix with sparsity of only $11.77\%$. We denote the set of known ratings as $\mathcal{I} = \{(i,j) : A_{ij} \text{ is known}\}$. Our task is to predict the missing ratings in the user-movie matrix $A$, so we can provide movie recommendations to individual users. Root Mean Squared Error (RMSE) between observed and predicted ratings will be used to quantify the quality of our predictions. For a given set of observed ratings $\mathcal{O}$,

$$RMSE = \sqrt{\frac{1}{|\mathcal{O}|} \sum_{(i,j)\in\mathcal{O}} (A_{ij} - \hat{A}_{ij})^2} \tag{1}$$

where $\hat{A}_{ij}$ denotes the predicted value of $A_{ij}$.

### B. Data Preprocessing

Some of the presented models in this paper require preprocessing on the matrix $A$ before training, and the preprocessing methods are described here.

*1) Column-wise Standard Normalization:* Entries in each column of the matrix $A$ are normalized to have a mean of 0 and a variance of 1. Reversely, a matrix $A'$ can be denormalized using the column means and standard deviations calculated in the normalization process for $A$.

*2) Imputing Missing Ratings:* Some of the approaches presented in this paper operate on a matrix without missing entries. Since the observed matrix $A$ is sparse, missing entries of $A$ need to be imputed. Denote the imputed matrix by $\tilde{A}$. Then, $\forall (i,j) \in \mathcal{I}$, $\tilde{A}_{ij} = A_{ij}$. We consider the following imputing methods:

*i. Zero imputing:* For each $(i,j) \notin \mathcal{I}$, $\tilde{A}_{ij} = 0$.

*ii. Column mean imputing:* For each $(i,j) \notin \mathcal{I}$, $\tilde{A}_{ij} = \bar{A}_j$, where $\bar{A}_j$ represents the mean of the $j$th column of $A$.

## III. Models and Methods

In this section, we first present two baseline models, then other individual models, and finally the ensemble model used in our collaborative filtering approach. Optimal parameters for each non-baseline model are tuned by 10-fold cross-validation on $\mathcal{I}$.

### A. Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) [4] is a common method for matrix factorization. Any matrix $M \in \mathbb{R}^{n\times m}$ can be decomposed into $M = U\Sigma V^T$, where $U \in \mathbb{R}^{n\times n}$, $\Sigma \in \mathbb{R}^{n\times m}$, and $V \in \mathbb{R}^{m\times m}$. $U$ and $V$ are orthogonal matrices, and $\Sigma$ is a diagonal matrix with rank($A$) positive singular values on its main diagonal in decreasing order.

We apply SVD to the standard-normalized and zero-imputed user-movie matrix $\tilde{A}$ to decompose it into $\tilde{A} = U\Sigma V^T$. The SVD approach is to approximate $\tilde{A}$ by another low-rank matrix, $\hat{A}$, so that the missing ratings can be predicted by corresponding entries of $\hat{A}$. The Eckart-Young theorem states that the optimal (in terms of Frobenius norm

objective) rank $k$ approximation of a matrix $\tilde{A}$ is given by $\tilde{A}_k = U_k \Sigma_k V_k^T$, where $U_k \in \mathbb{R}^{n\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, and $V_k \in \mathbb{R}^{m\times k}$[5]. $U_k$ and $V_k$ is the first $k$ columns of $U$ and $V$, respectively. $\Sigma_k$ is a diagonal sub-matrix of $\Sigma$ containing the $k$ largest singular values. Then, we denormalize $\tilde{A}_k$ to obtain our approximation. The optimal $k$ is selected as 9 by cross-validation, and the SVD method serves as the first baseline in our approach.

### B. Alternating Least Squares (ALS)

Considering factored parameterization via $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{k\times m}$ such that $A \approx UV$, Alternative Least Squares (ALS) [6] is an iterative algorithm minimizing the following non-convex objective function:

$$\mathcal{L}(U,V) = \frac{1}{2}||\Pi_\Omega(A-UV)||_F^2 + \frac{\lambda}{2}(||U||_F^2 + ||V||_F^2) \quad (2)$$

where $\lambda > 0$ and $\Pi_\Omega$ selects the observed entries in $A$. Since the parameter dependencies in Eq. (2) for matrix factorization form a bipartite graph between the rows of $U$ and the columns of $V$ [7], we can separate out the part of the objective depending on a column $v_j$ of $V$ (completely analogous a row $u_i$ of $U$) as follows

$$\mathcal{L}_U(v_j) = \frac{1}{2} v_j^T (\sum_i \omega_{ij} u_i u_i^T + \lambda I) v_j - (\sum_i \omega_{ij} a_{ij} u_i^T) v_j \quad (3)$$

where $\omega_{ij}$ denotes if $a_{ij}$, the entry of $A$ at $i$th row and $j$th column, is known. Treating $U$ fixed, the update rule for $v_j$ in each iteration can be derived as

$$v_j^* = (\sum_i \omega_{ij} u_i u_i^T + \lambda I)^{-1} (\sum_i \omega_{ij} a_{ij} u_i) \quad (4)$$

We apply ALS with $k = 3$ and $\lambda = 0.1$ for 20 iterations to the standard-normalized and zero-imputed user-movie matrix $\tilde{A}$ to obtain $\hat{U}$ and $\hat{V}$. $U$ and $V$ are initialized as the $U_k$ and $V_k^T$ matrix from SVD with $k = 9$ on $\tilde{A}$, respectively. We then denormalize $\hat{A}$, where $\hat{A} = \hat{U}\hat{V}$, to obtain our approximation. The ALS method serves as the second baseline in our approach.

### C. Iterative SVD

Iterative SVD is an iterative variant of SVD. The procedure is adapted from the *singular value shrinkage* operator introduced by Cai et al. [8]. At each iteration, SVD is applied to a matrix $A'$ to decompose it into $A' = U'\Sigma' V'^T$. Then, diagonal entries of $\Sigma'$ are shrunken by a shrinkage parameter $\tau$ as follows

$$\Sigma'_{ii} = max\{0, \Sigma'_{ii} - \tau\} \quad (5)$$

to obtain $\Sigma'_s$. Matrix $A'_s$ is reconstructed as $A'_s = U'\Sigma'_s V'^T$, and the observed entries of $A'_s$ are set to their original ratings to become the $A'$ in next iteration.

We apply Iterative SVD to the column-mean-imputed matrix $\tilde{A}$ with optimal $\tau = 38$ for 15 iterations to obtain our approximation.

### D. Regularized+Biased SVD (RBSVD)

Regularized SVD (RSVD) is another variant of SVD that tries to find low-rank approximation using regularized factor matrices $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{m\times k}$ and it has been demonstrated to substantially improve upon the results of SVD in collaborative filtering [9]. [10] proposes to additionally apply separate biases $b_u$ for users and $b_v$ for movies. The estimated rating $\hat{r}_{ij}$ for the combination of user $i$ and movie $j$ is calculated as:

$$\hat{r}_{ij} = u_i^T v_j + b_{u_i} + b_{v_j}, \quad (6)$$

and the objective function is formulated as:

$$\mathcal{L}_{ij} = (r_{ij} - \hat{r}_{ij})^2 + \lambda_1(\|u_i\|^2 + \|v_j\|^2) + \lambda_2(b_{u_i} + b_{v_j} - \mu)^2, \quad (7)$$

where $\mu$ is the global mean of movie ratings on the train set. This objective can then be minimized by Stochastic Gradient Descent We set $k = 12$, $\lambda_1 = 0.075$ and $\lambda_2 = 0.04$ and learn $U$, $V$, $b_u$, $b_v$ for 50 epochs. The initial learning rate is set to 0.05 with a learning rate decay of 0.7 every 5 epochs.

### E. Factorization Machines (FM)

Factorization machine is method that combines the advantages of Support Vector Machines (SVM) with factorization models and is capable of estimating model parameters accurately even under very sparse data [11]. The most commonly used second-order FM is formulated as a simple quadratic regression for $n$ input variables:

$$f(x) = w_0 + \sum_{p=1}^{n} w_p x_p + \sum_{p=1}^{n-1} \sum_{q=p+1}^{n} \langle v_p, v_q \rangle x_p x_q \quad (8)$$

where $w_0 \in \mathbb{R}, \mathrm{w} = \{w_1, ..., w_n\} \in \mathbb{R}^n, \mathrm{V} = \{v_1, ..., v_n\} \in \mathbb{R}^{n\times k}$ are trainable parameters. $\langle \cdot, \cdot \rangle$ denotes the dot product of two vectors of size $k$, where $k \in \mathbb{N}_0^+$ is a hyperparameter that defines the dimensionality of the factorization. The use of the second order allows to model the interactions among different input variables.

Freudenthaler [12] proposes that FMs can be enhanced by introducing Bayesian Inference as Bayesian Factorization Machines (BFM), which are faster and more scalable in applications such as recommendation systems. We use python myFM [13] library to train BFM models and follow its tutorials as well as [14] to build features.

*1) BFM with Base Features:* We start with base features which are simply one-hot encodings of the user and movie indices. As a common practice in probabilistic matrix factorization, user and movie vectors are assumed to be drawn from separate normal priors: $u_i \sim \mathcal{N}(\mu_U, \Sigma_U), v_i \sim \mathcal{N}(\mu_M, \Sigma_M)$. We train a Bayesian Linear Regression (BLR) model with Gibbs Sampling as in [12] on these base features and refer to this model as **BFM Base** in later experiments.

*2) BFM with Implicit Features:* In addition, we include implicit features for users and movies. User implicit features are a binary vector recording if each of the movies has been rated by the user. The vector is normalized by $\frac{1}{\sqrt{\max(L,1)}}$ where $L$ represents the number of non-zero entries in the vector. Similarly, movie implicit features are a normalized vector recording if each of the users has rated this movie. We assign separate variances to the two sets of implicit features as for the base features. We combine the base and the implicit features and train our **BFM Base + Implicit** models. In addition to BLR with Gibbs sampler, we also try ordinal probit regression [15] [16].

We also attempt to handcraft additional auxiliary features about the users and movies such as their statistical features and encoded latent vectors from our autoencoders. However, none of these attempts helps further. The only 2 hyperparameters for all our Bayesian FM models are the aforementioned factorization dimensionality $k$ and the number of iterations $N$. We set them to 10 and 500 respectively.

### F. Autoencoders

Autoencoder is a type of symmetric neural network to learn efficient, low-dimensional data representations in an unsupervised manner [17]. It learns a latent space of dimension $k$ to represent a nonlinear dimension reduction of the input vector. Consisting of an encoder $\phi : \mathbb{R}^d \to \mathbb{R}^k$ and a decoder $\theta : \mathbb{R}^k \to \mathbb{R}^d$ (where $k < d$), the network is learned to minimize the reconstruction error between the input x and the output $\hat{\mathrm{x}} = \theta(\phi(\mathrm{x}))$. Autoencoder has been proven effective in collaborative filtering in [18] [19] [20]. In this work, we first test a vanilla deep autoencoder similar to [20] and then extend it to its probabilistic variant, variational autoencoder [21].

*1) Vanilla Autoencoder (AE):* For a user vector $\mathrm{x}_u \in \mathbb{R}^m$, we minimize its masked mean square error:

$$\mathcal{L}_{\mathrm{RECON}_{\mathrm{x}_u}} = \frac{1}{k}\sum_{i=1}^{k}((\mathrm{x}_{u_i} - \hat{\mathrm{x}}_{u_i})^2 \cdot \mathcal{M}_u) \tag{9}$$

where $\mathcal{M}_u \in \mathbb{R}^m$ contains binary values to represent if the rating of each movie in the user vector is recorded.

Our autoencoder has an architecture of [$m$-256-32-256-$m$] with a latent space $\mathrm{z}_u$ of size $k = 32$. We find it benefitial to apply a learnable user bias matrix $b \in \mathbb{R}^n$ to the output of the decoder so that $\hat{\mathrm{x}}_u = \theta(\phi(\mathrm{x}_u)) + b_u$. We also attempt to learn a movie bias matrix but it does not give any improvement. We use LeakyReLU [22] as activation function and initialize all layer weights with Kaiming Uniform Initialization [23] since it particularly considers rectifier non-linearlities. Dropout of 0.5 is applied to all hidden layers as well as the input to the encoder to avoid over-fitting. We zero-impute the user-movie matrix and train the network for 1000 epochs using Adam Optimizer [24] with an initial learning rate of 0.025 and an exponential learning rate decay with rate 0.992.

*2) Variational Autoencoder (VAE):* A common problem of a vanilla autoencoder is that it only learns to reconstruct the input with as small loss as possible and disregards the properties of its latent space. Variational autoencoder [21], a probabilistic variant of autoencoder, aims to provide better generative properties by encoding the input as a distribution over the latent space instead of a single vector. For each user vector $\mathrm{x}_u$, we approximate the intractable posterior $p(\mathrm{z}_u|\mathrm{x}_u)$ with a simpler one $q(\mathrm{z}_u)$ that is "as close as possible". Here we set $q(\mathrm{z}_u)$ as a diagonal Gaussian distribution $q(\mathrm{z}_u) = \mathcal{N}(\mu_u, \mathrm{diag}(\sigma_u^2))$ and minimize the Kullback-Leiber (KL) divergence [25] $\mathrm{KL}(q(\mathrm{z}_u)||p(\mathrm{z}_u|\mathrm{x}_u))$. We follow [26] and formulate the overall objective function as:

$$\begin{aligned}\mathcal{L}_\beta(x_u;\theta,\phi) = \mathbb{E}_{q_\phi(\mathrm{z}_u|\mathrm{x}_u)}\left[\log p_\theta(\mathrm{x}_u|\mathrm{z}_u)\right] \\ -\beta \cdot \mathrm{KL}(q_\phi(\mathrm{z}_u|\mathrm{x}_u)||p(\mathrm{z}_u))\end{aligned} \tag{10}$$

where $\mathbb{E}_{q_\phi(\mathrm{z}_u|\mathrm{x}_u)}\left[\log p_\theta(\mathrm{x}_u|\mathrm{z}_u)\right] = \mathcal{L}_{\mathrm{RECON}_{\mathrm{x}_u}}$ represents the reconstruction error.

Assuming $p(\mathrm{z}_u) = \mathcal{N}(0, \mathrm{I}_k)$, we have:

$$\mathrm{KL}(q_\phi(\mathrm{z}_u|\mathrm{x}_u)||p(\mathrm{z}_u)) = \frac{1}{2}\sum_{i=1}^{k}(\sigma_{\phi_i}^2 + \mu_{\phi_i}^2 - 1 - \ln\sigma_{\phi_i}^2) \tag{11}$$

Note that $\beta$ in equation 10 is a tunable hyperparameter. We compare linear annealing schedule as in [26] and cyclic annealing schedule as in [27], and find that cyclic annealing more effectively reduces the reconstruction error while mitigating the KL-vanishing problem. Formally, $\beta$ is scheduled as:

$$\beta = \min(\frac{(e \mod M)\cdot\beta_{max}}{M \cdot R}, \beta_{max}) \tag{12}$$

where $e$ is the current epoch, $M = 100$ denotes the number of epochs for an annealing cycle, $\beta_{\max} = 0.2$ represents the maximum possible value for $\beta$ (selected by taking the optimal value from the linear annealing schedule), and $R = 0.5$ denotes the portion of epochs within a cycle to increase $\beta$.

The VAE has double output layers of shape 256-32 in the encoder to predict $\mu_\phi(\mathrm{x}_u)$ and $\log\sigma_\phi^2(\mathrm{x}_u)$ (log variance is predicted instead of standard deviation) separately. We can then perform reparametrization trick [21] to sample $\epsilon \sim \mathcal{N}(0, \mathrm{I}_k)$ and reparameterize $\mathrm{z}_u = \mu_\phi(\mathrm{x}_u) + \epsilon \odot \sigma_\phi(\mathrm{x}_u)$ so that the gradient with respect to $\phi$ can be back-propagated through sampling $\mathrm{z}_u$. We change the activation function to tanh as in [26] and initialize all layer weights with Xavier Initialization [28]. For our VAE, the exponential decay rate for the learning rate is increased to 0.997 and the number of epochs trained is increased to 2000.

### G. Neural Collaborative Filtering (NCF)

Inspired by the Neural Collaborative Filtering (NCF) method proposed by He et al. [29], we use artificial neural networks to learn the relationship between the user latent space, the movie latent space and the ratings. One key task

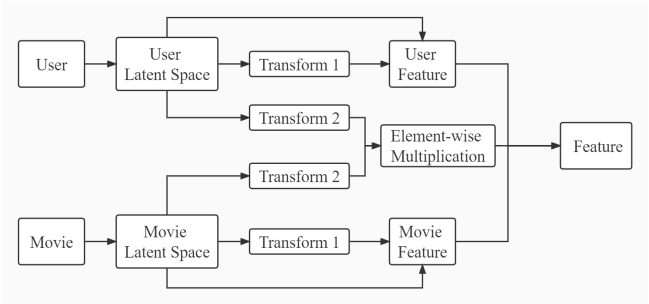


Figure 1. Feature Generation of NCF model

is to generate informative features. Our feature is combined as shown in Figure 1.

We first embed users and movies to their latent space, then use 'transform 1' to fit the spatial relationship between latent spaces and the final feature. Besides, we also use 'transform 2 to change two latent vectors and then do a element-wise multiplication. Experiments show that this multiplication feature really improves the testing performance. The user feature, the movie feature and the multiplication feature are concatenated to form the final feature.

We also use a structure with 'shortcuts' from movie latent space to movie feature, which is inspired by ResNet [30]. Experiments prove that these two shortcuts decrease the loss.

After cross validation, we take the embedding dimension as 128 for both users and movies. 'Transform 1' and 'transform 2' are both defined as two dense layer, the first from 128 to 256 and the second from 256 to 128, all with ReLU layer and dropout layer of rate 0.1. Note that there is also a dropout layer of rate 0.05 between the embedding layer and the first fully-connected layer of transformers. Our dataset is very easy to be overfitting, so we take a large weight decay of 1e-4.

After we get the feature, we simply use a hidden layer and RMSE loss to train our regressor. We also try considering the task as a 5-category classification task and then use the probabilities to do final prediction, however, direct regression has better performance. So we finally use a simple regressor to predict the ratings.

### *H. Blending Multiple Models*

Due to the possibility that different models may have better generative ability for different parts of the user-movie matrix, we attempt to blend multiple models. We empirically find that some models that produce higher test errors have lower train errors, so it is unfair to simply perform cross-validation using predictions from the models learned on the entire data. Therefore, we preserve all models from our 10-fold cross validation test and train the blending models based on the 10% holdout set in each fold. We train linear regression, XGBoost [31], and Gradient Boosting [32] models on the 10 folds and average the predictions from all models to get our blended matrix. We then clip the values into $[1, 5]$ to get the final predictions. We include all models we describe above but the two baseline models.

## IV. Experiments and Results

| Methods | Local CV | Public Test |
|---|---|---|
| SVD (baseline 1) | 1.0081 | 1.0049 |
| ALS (baseline 2) | 0.9899 | 0.9874 |
| Iterative SVD | 0.9840 | 0.9816 |
| RBSVD | 0.9820 | 0.9788 |
| BFM Base (Gibbs) | 0.9779 | 0.9749 |
| BFM Base + Implicit (Gibbs) | 0.9715 | 0.9687 |
| BFM Base + Implicit (Ordinal Probit) | 0.9694 | 0.9672 |
| AE | 0.9791 | 0.9758 |
| VAE | 0.9769 | 0.9749 |
| NCF | 0.9889 | 0.9856 |
| Blending (Linear Regression) | 0.9682 | 0.9659 |
| Blending (XGBoost) | 0.9676 | **0.9656** |
| Blending (Gradient Boosting) | **0.9674** | **0.9656** |

Table I
Experiment results of all our models and blending methods.

In this chapter, we introduce the methods and details of the experiments. We utilize 10-fold cross validation on the training set and also predict the output on the public test set. "Local CV" on the first column of table I is the RMSE loss on cross validation and the "Public Test" is the RMSE loss on public test set.

All single models outperform the 2 baseline models. Regarding similar models, the VAE slightly outperforms the vanilla AE thanks to probabilistic modeling and cyclic annealing KL loss. The introduction of user and movie implicit features substantially boosts the performance of BFM that is trained only with the base features. The replacement of BLR model with ordinal probit further improves the performance.

Meanwhile, all of our 3 blending models further reduces the local CV score of the best single model by more than 0.001, with XGBoost and Gradient Boosting regressor producing equally best results on the public test set. Although the weights assigned to RBSVD and Bayesian FM by the linear regressors are negative, we still find that including them would produce slightly lower CV scores for all of our blending methods. Both local CV and Public Test prove that our blending methods are robust and outperform all individual models and baselines.

## V. Summary

In this work, we propose an ensemble-based method for collaborative filtering, which aggregates approaches like literative SVD, RBSVD, BFM, NCF and etc via linear regression, XGBoost and Gradiet Boosting. Judging from our local 10-fold CV score and public test score, we could draw a safe conclusion that our ensemble model is reliable and effective, which also have a significant improvement compared with baseline models (SVD and ALS).

## References


[1] B. Smith and G. Linden, "Two decades of recommender systems at amazon. com," *Ieee internet computing*, vol. 21, no. 3, pp. 12–18, 2017.

[2] J. Bennett, S. Lanning *et al.*, "The netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007. Citeseer, 2007, p. 35.

[3] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[4] 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, 1980.

[5] C. Eckart and G. M. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, pp. 211–218, 1936.

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

[7] T. Hofmann, "Computational intelligence lab lecture notes," 2022.

[8] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM J. Optim.*, vol. 20, pp. 1956–1982, 2010.

[9] S. Zheng, C. Ding, and F. Nie, "Regularized singular value decomposition and application to recommender system," 2018. [Online]. Available: https://arxiv.org/abs/1804.05090

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

[11] S. Rendle, "Factorization machines," in *2010 IEEE International Conference on Data Mining*, 2010, pp. 995–1000.

[12] C. Freudenthaler, "Bayesian factorization machines," 2011.

[13] "myfm: A python/c++ implementation of bayesian factorization machines," https://github.com/tohtsky/myFM.

[14] S. Rendle, L. Zhang, and Y. Koren, "On the difficulty of evaluating baselines: A study on recommender systems," 2019. [Online]. Available: https://arxiv.org/abs/1905.01395

[15] J. Albert and S. Chib, "Bayesian analysis of binary and polychotomous response data," *Journal of The American Statistical Association - J AMER STATIST ASSN*, vol. 88, pp. 669–679, 06 1993.

[16] J. H. Albert and S. Chib, "Sequential ordinal modeling with applications to survival data," *Biometrics*, vol. 57, no. 3, pp. 829–836, 2001. [Online]. Available: http://www.jstor.org/stable/3068422

[17] 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

[18] 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: Association for Computing Machinery, 2015, p. 111–112. [Online]. Available: https://doi.org/10.1145/2740908.2742726

[19] 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

[20] O. Kuchaiev and B. Ginsburg, "Training deep autoencoders for collaborative filtering," 2017. [Online]. Available: https://arxiv.org/abs/1708.01715

[21] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[22] M. Khalid, J. Baber, M. K. Kasi, M. Bakhtyar, V. Devi, and N. Sheikh, "Empirical evaluation of activation functions in deep convolution neural network for facial expression recognition," in *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, 2020, pp. 204–207.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[25] J. M. Joyce, *Kullback-Leibler Divergence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722. [Online]. Available: https://doi.org/10.1007/978-3-642-04898-2_327

[26] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 689–698. [Online]. Available: https://doi.org/10.1145/3178876.3186150

[27] H. Fu, C. Li, X. Liu, J. Gao, O. Celikyilmaz, and L. Carin, "Cyclical annealing schedule: A simple approach to mitigating KL vanishing," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 240–250. [Online]. Available: https://aclanthology.org/N19-1021

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[29] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[31] T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. [Online]. Available: https://doi.org/10.1145%2F2939672.2939785

[32] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.