JMLR: Workshop and Conference Proceedings 80:1–16, 2017 ACML 2017

PHD: A Probabilistic Model of Hybrid Deep Collaborative Filtering for Recommender Systems

Jie Liu DAICOOLB@OUTLOOK.COM
Dong Wang WANGDONG@SJTU.EDU.CN
Yue Ding DINGYUE@SJTU.EDU.CN
Shanghai Jiao Tong University

Abstract

Collaborative Filtering (CF), a well-known approach in producing recommender systems, has achieved wide use and excellent performance not only in research but also in industry. However, problems related to cold start and data sparsity have caused CF to attract an increasing amount of attention in efforts to solve these problems. Traditional approaches adopt side information to extract effective latent factors but still have some room for growth. Due to the strong characteristic of feature extraction in deep learning, many researchers have employed it with CF to extract effective representations and to enhance its performance in rating prediction. Based on this previous work, we propose a probabilistic model that combines a stacked denoising autoencoder and a convolutional neural network together with auxiliary side information (i.e, both from users and items) to extract users and items' latent factors, respectively. Extensive experiments for four datasets demonstrate that our proposed model outperforms other traditional approaches and deep learning models making it state of the art.

Keywords: Collaborative Filtering, Stacked Denoising Autoencoder, Convolutional Neural Network, Recommender Systems

1. Introduction

In order to overcome the problem of information overload, applications of recommender systems have attracted a large amount of attention in recent years. Many companies have included recommender systems into their software such as Amazon[1], JD[2], Taobao[3], etc. One effective method of recommendation is to predict new ratings of different users and items. Currently, the most popular methods in this field are *Content-based Filtering* (CBF) and *Collaborative Filtering* (CF).

CBF uses the context of users or items to predict new ratings. For example, we can generate user preferences according to their age, gender, or graph of their friends, etc. In addition, genres and reviews of items can be analyzed and selected to make recommendation for different users. On the other side, CF uses the ratings of users for items in their feedback records to predict new ratings. For instance, a list of N users $\{u_1, u_2, ..., u_N\}$ and a list of M items $\{v_1, v_2, ..., v_M\}$ are given. Simultaneously, a list of items, v_{u_i}, has been rated by user u_i. These ratings can either be explicit feedback on a scale of 1-5, or implicit

1. https://www.amazon.com/
2. https://www.jd.com/
3. https://www.taobao.com/

feedback on a scale of 0-1. Our goal is to predict new feedback from users who have no records. Furthermore, CF is often superior to CBF because CF outperforms the agnostic vs. studied contest Ricci et al. (2011).

The most well-known approaches in CF are Probabilistic Matrix Factorization (PMF) Mnih and Salakhutdinov (2008) and Singular Value Decomposition (SVD) Sarwar et al. (2000). However, they are often faced with sparse matrices, a.k.a. cold start problem, so they fail to extract effective representations from users and items. To address the difficulty associated with this issue, an active line of research during the past decade and a variety of techniques have been proposed Zhou et al. (2011, 2012); Trevisiol et al. (2014). Almost all of these techniques consider employing the side information of users or items into CF to generate more effective features. Moreover, the most commonly used side information are documents, hence some approaches based on document modeling methods such as Latent Dirichlet Allocation and Collaborative Topic Modeling have been proposed Ling et al. (2014); Wang and Blei (2011). While prior studies have examined documents as a bag-of-words model, it may be preferable to contemplate the impact of sequences of words in documents; therefore previous methods are limited to some extent.

Because feature extraction of deep learning has strong characteristic, an increasing amount of studies have applied deep learning with side information to generate effective representations, such as Stacked Denoising AutoEncoder (SDAE) Wang et al. (2015a,b) and Convolutional Neural Network (CNN) Kim et al. (2016), a.k.a. ConvMF, which differentiates the orders of words in documents. However, ConvMF only considers item side information (e.g., document, review or abstract, etc.), so users' latent factors still have no effective representations. For another, the work of Dong et al. (2017) indicates that SDAE is adept at extracting remarkable features in users' latent factors without documents inside; this causes items' latent factors reside with some conventional approaches. In order to solve above issues, we combine SDAE and ConvMF into a probabilistic model to more effectively extract both users and items' latent factors. Our contributions can be summarized as follows:

- we propose a hybrid probabilistic model that we call PHD, which formulates users and items' latent factors meanwhile. To the best of our knowledge, PHD is the first model to seamlessly integrate two deep learning models into PMF under probabilistic perspective.
- we extensively demonstrate that PHD is a combination of several state-of-the-art methods but with a more effective feature representation.
- we conduct different experiments which show that PHD can alleviate the data sparsity problem in CF.

The rest of the paper is organized as follows. Section 2 defines the problem, background information and indispensable notations. Section 3 describes the PHD model and optimization method. Experimental results for the components analysis and performance comparisons are presented in Section 4. We conclude with a summary of this work and a discussion of future work in Section 5.

2. Preliminaries

In this section, we start with formulating the problems discussed in this paper, and then briefly review Matrix Factorization (MF), SDAE and CNN.

2.1. Problem Definition

Similar to some existing publications, this paper takes explicit feedback as training and testing data to complete the recommendation task. In a standard recommendation setting, we have n users, m items, and an extremely sparse rating matrix $R \in R^{n \times m}$. Each entry R_{ij} of R corresponds to user i's rating on item j. Likewise, the auxiliary information matrix of users and items are denoted by $X \in R^{n \times e}$ and $Y \in R^{m \times f}$, respectively. Let $u_i, v_j \in R$ be user i's latent factor vector and item j's latent factor respectively, where k is the dimensionality of the latent space. Therefore, the corresponding matrix forms of latent factors for users and items are $U = u_{[1:n]}$ and $V = v_{[1:m]}$, respectively. Given the sparse rating matrix R and the side information matrix X as well as Y, our goal is to learn effective users' latent factors U and items' latent factors V, and then to predict the missing ratings in R.

2.2. Matrix Factorization

Due to the required accuracy and scalability, matrix factorization has been the most high-profile method in Collaborative Filtering, and was first developed in the Netflix contest Koren et al. (2009). Generally, MF model can learn low-rank representations (i.e., latent factors) of users and items in the user-item matrix, which are further used to predict new ratings between users and items. For clarity, we include the most common formulation of MF as follow:

$$\mathcal{L} = arg \min_{U,V} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} (R_{ij} - u_i^T v_j)^2 + \frac{1}{2} \lambda_U ||U||_F^2 + \frac{1}{2} \lambda_V ||V||_F^2 \tag{1}$$

in which I_{ij} is an indicator function that is equal to 1 if $R_{ij} > 0$, otherwise 0. In addition, $||U||_F$ and $||V||_F$ denote the Frobenius norm of the matrix, and λ_U and λ_V are regularization parameters that are usually set to alleviate model overfitting.

2.3. Stacked Denoising Autoencoder

As a specific form of neural network, an autoencoder (AE) takes a given input and maps it or encodes it to a hidden representation via deterministic mapping. Denoising autoencoders reconstruct the input from a corrupted version of data with the motivation of learning a mapping method from data. Various types of autoencoder have been developed in the literature and have shown promising results in several domains Lee et al. (2009); Kavukcuoglu et al. (2009). Moreover, denoising autoencoders can be stacked to construct a deep network also known as a stacked denoising autoencoder which allows learning higher level representations Vincent et al. (2008). Certainly, there is some research that employs SDAE to enhance the performance of recommendation. For example, the autoencoder of Wang et al. (2015a) generates a relational SDAE model with tags, and that of Li et al. (2015) utilizes a

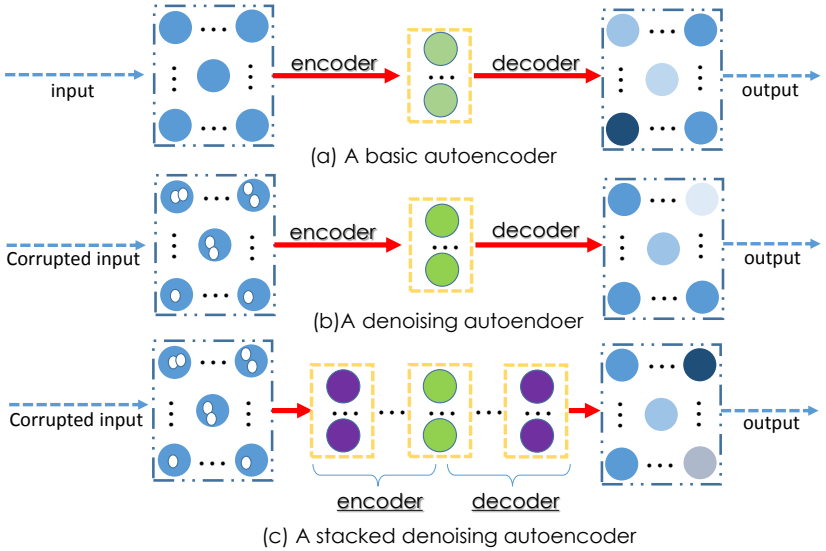

Figure 1: Overview of three kinds of autoencoder

marginalized denoising autoencoder with PMF to compose latent factors. Because SDAE has the characteristic of feature extraction, we adopt it with auxiliary side information to generate users' latent factors in our PHD model, which will be introduced in Section 3.2. In order to have an intuitive understanding of AE, Figure 1 illustrates the most popular variant of AEs from three different aspects: a) a basic AE, b) a denoising AE, c) a stacked denoising AE. Specifically, learning of a SDAE involves solving the following regularized optimization problem:

$$\min_{W_l, b_l} ||X_{corrupted\_input} - X_{output}||_F^2 + \lambda \sum_l (||W_l||_F^2 + ||b_l||_2^2) \tag{2}$$

where λ is a regularization parameter and W_l as well as b_l is weight parameter of SDAE.

2.4. Convolutional Neural Network

A convolutional neural network is a class of deep, feed-forward neural network that has successfully been applied to Computer Vision Krizhevsky et al. (2012), Natural Language Processing Kim (2014) and Audio Signal Processing Piczak (2015). Similarly, there are some studies have used CNN in recommender systems, such as content-based music recommendation van den Oord et al. (2013), which compares a traditional approach using a bag-of-words representation of the audio signals with deep CNN, and ConvMF, which employs CNN to generate items' latent factors but ignores users' latent factors because of the user privacy problem. In other words, their work only considers one sided latent factors (i.e., item side information), which makes predicted ratings not equal to SVD through gradient descent. Consequently, we use items' latent factors constructed by ConvMF and users' latent factors constructed by a variant of SDAE (auxiliary SDAE that we call aSDAE), which will be introduced in Section 3.2.

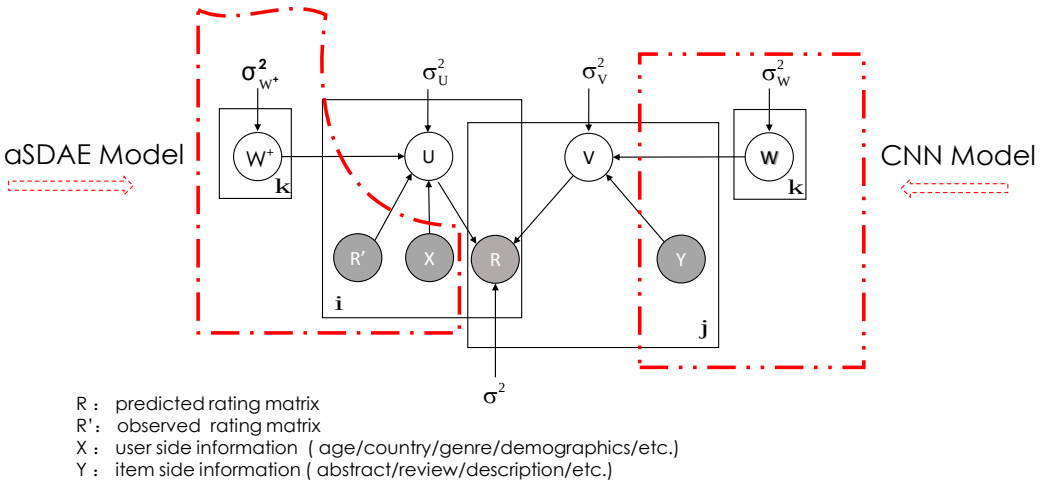

Figure 2: Overview of the PHD model

3. PHD Model

In this section, we provide details of the proposed PHD model through three steps: 1) First, we introduce the probabilistic model, and describe the main idea used to combine PMF, aSDAE and CNN tactfully in order to utilize ratings, user side information and item side information simultaneously. 2) Next, we explain the detailed architecture of our aSDAE, which generates users' latent factors by exploiting user side information, and CNN, which generates items' latent factors by analyzing item description documents. 3) Finally, we describe how to optimize our PHD model.

3.1. Probabilistic Model of PHD

Figure 2 shows the overview of the probabilistic model for PHD, which integrates aSDAE and CNN into PMF. From a probabilistic point of view, the conditional distribution over predicted ratings can be given by

$$p(R|U,V,\sigma^2) = \prod_{i}^{N}\prod_{j}^{M} N(R_{ij}|u_i^T v_j, \sigma^2)^{I_{ij}} \tag{3}$$

where $N(x|\mu,\sigma^2)$ is the probability density function of the Gaussian normal distribution with mean μ and variance σ^2. As for users' latent factors, we assume that a user's latent factor is formed by three variables: 1) internal weights W^+ in aSDAE, 2) X_i representing the side information of user i, and 3) varepsilon variable as Gaussian noise, which is applied to optimize the user's latent factor for the rating. Thus, the final user's latent factor can be generated by the following equations.

$$u_i = asdae(W^+, X_i) + \varepsilon_i \tag{4}$$

$$\varepsilon_i = N(0, \sigma_U^2 I) \tag{5}$$

where $asdae()$ represents the L/2 layer's output of aSDAE architecture. For each weight w_k^+ in W^+, we place zero-mean spherical Gaussian prior, the most commonly used prior.

$$P(w^+|\sigma_{W^+}^2) = \prod_k N(w_k^+|0, \sigma_{W^+}^2) \tag{6}$$

Accordingly, the conditional distribution over users' latent factors is given by

$$p(U|W^+, X, \sigma_U^2) = \prod_i^N N(u_i|asdae(W^+, X_i), \sigma_U^2) \tag{7}$$

Just like the user's latent factor, an item's latent factor consists of three variables: 1) internal weights W in CNN, 2) Y_j representing the side information of item j, and 3) epsilon variable as Gaussian noise, which is used to further optimize an item's latent factor for rating. Hence, the final item's latent factor can be given by the following equations.

$$v_j = cnn(W, Y_j) + \epsilon_j \tag{8}$$

$$\epsilon_j = N(0, \sigma_V^2 I) \tag{9}$$

where $cnn()$ represents the output of CNN architecture. In the same way, for each weight w_k in W, we set zero-mean spherical Gaussian prior.

$$P(w|\sigma_W^2) = \prod_k N(w_k|0, \sigma_W^2) \tag{10}$$

$$p(V|W, Y, \sigma_V^2) = \prod_j^M N(v_j|cnn(W, Y_j), \sigma_V^2 I) \tag{11}$$

3.2. Auxiliary Stacked Denoising Autoencoder of PHD

The work of Vincent et al. (2008) has shown that multiple layers stacked together can generate rich representations in hidden layers. We extend this idea as did Dong et al. (2017) with auxiliary information. Figure 3 shows the generative process with a user's latent factor, which we employ to compose the matrix U in latent factors. Furthermore, we give a detailed generative process of a user's latent factor as follows:

- For each hidden layer $l \in 1, ..., L-1$ of the aSDAE model, the hidden representation h_l is computed as:

  $$h_l = g(C_l h_{l-1} + Q_l \tilde{X} + b_l)$$

  in which C_l as well as Q_l is the weight parameter in each layer, and b_l is the bias vector for each layer. $g()$ is a nonlinear activation function. Recall that h_0 is a corrupted version of R_i and $\tilde{X}$ is a corrupted version of X.

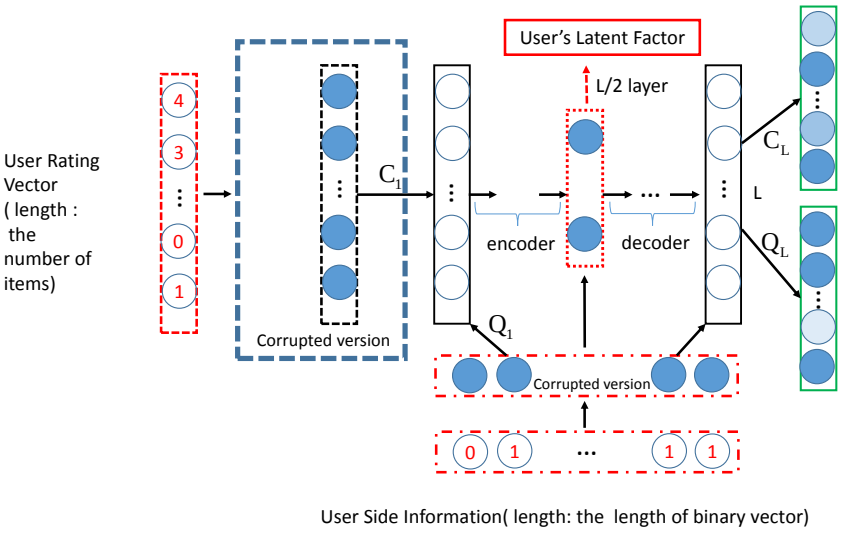

Figure 3: Auxiliary stacked denoising autoencoder architecture of the PHD model

- For the output layer L, the outputs can be computed as:

$$\hat{R}_i = f(C_L h_L + b_{\hat{R}_i})$$
$$\hat{X} = f(Q_L h_L + b_{\hat{X}})$$

where $f()$ is also a nonlinear activation function.

Note that the first $\frac{L}{2}$ layers of aSDAE serve as an encoder and the last $\frac{L}{2}$ layers serve as a decoder. Accordingly, we can learn C_l, Q_l and b_l for each layer using the back-propagation algorithm. In our PHD model, we only use the $\frac{L}{2}$ layer as the user's latent factor. For user side information, according to the content of information, we encode the information into a binary vector whose length can be up to 500 in our experiment. For the input layer, we only regard the rating vector of each user as the source input.

3.3. Convolutional Neural Network of PHD

The objective of our CNN architecture is to obtain documents' latent vectors from documents of items, which are used to compose the items' latent factors with epsilon variables. Figure 4 reveals our CNN architecture that contains four layers: 1) embedding layer, 2) convolution layer, 3) pooling layer, 4) output layer.

Embedding layer

The function of the embedding layer is to transform a raw document into a numeric matrix according to the length of words, which will be conducted a convolution operation in the next layer. For instance, if we have a document whose number of words is l, then we can concatenate a vector of each word into a matrix in accordance with the sequence of words. The word vectors are initialized with a pre-trained word embedding model such as

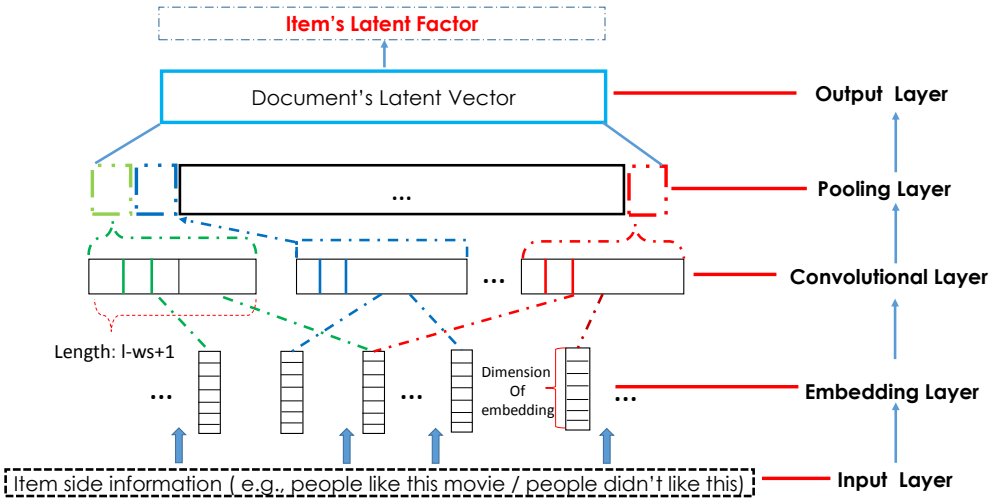

Figure 4: Convolutional neural network architecture of the PHD model

Glove Pennington et al. (2014). Next, the document matrix $D \in \mathbb{R}^{p \times l}$ can be visualized as follow:

$$\begin{bmatrix} w_{11} & \dots & w_{1i} & \dots & w_{1l} \\ w_{21} & \dots & w_{2i} & \dots & w_{2l} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{p1} & \dots & w_{pi} & \dots & w_{pl} \end{bmatrix}$$

in which p stands for the dimension of word embedding and $\boldsymbol{w}_{[1:p,i]}$ represents raw word i in the document.

Convolution Layer

The convolution layer extracts contextual features. We employ the convolution architecture to dispose documents. A contextual feature $c_i^j \in R$ is extracted by jth shared weight $W_c^j \in \mathbb{R}^{p \times ws}$ whose window size ws determines the number of surrounding words:

$$c_i^j = f(W_c^j * D(:, i : (i + ws - 1)) + b_c^j) \tag{12}$$

where * is a convolution operator, $b_c^j \in R$ is a bias for W_c^j and $f()$ is a nonlinear activation function. Among nonlinear activation functions such as sigmoid, tanh and rectified linear unit(ReLU). We use ReLU to avoid the problem of vanishing gradient, which causes slow optimization convergence and may lead to a poor local minimum. Then, a contextual feature vector $c^j \in \mathbb{R}^{l-ws+1}$ of a document with W_c^j is constructed by:

$$c^j = [c_1^j, c_2^j, ..., c_i^j, ..., c_{l-ws+1}^j] \tag{13}$$

However, one shared weight captures one type of contextual features. Thus, we use multiple shared weights to capture multiple types of contextual features, which enable us to generate contextual feature vectors as many as the number n_c of W_c. (i.e., W_c^j where j=1,2,...,n_c)

Pooling Layer

The pooling layer not only extracts representative features from the convolution layer but also deals with variable lengths of documents via pooling operation that constructs a fixed-length feature vector. After the convolution layer is created, a document is represented as n_c contextual feature vectors, where each contextual feature vector has variable length (i.e., $l - ws + 1$ contextual feature). However, such representation imposes two problems: 1) there are too many contextual c_i, in which most contextual features might not help enhance the performance of the model, 2) the length of contextual feature vector varied, which makes it difficult to construct the following layers. Therefore, we utilize max-pooling, which reduces the representation of a document into an n_c fixed-length vector by extracting only the maximum contextual feature from each contextual feature vector as follow.

$$d_f = [max(c^1), max(c^2), ..., max(c^j), ..., max(c^{n_c})] \tag{14}$$

where c^j is a contextual feature vector of length $l - ws + 1$ extracted by jth shared weight W_c^j

Output Layer

Generally, at output layer, high-level features obtained from the previous layer should be converted for a specific task. Thus, we project d_f on a k-dimensional space of users and items' latent factors for our recommendation task, which finally produces a document's latent vector by using conventional nonlinear projection:

$$s = tanh(W_{f_2}\{tanh(W_{f_1}d_f + b_{f_1})\} + b_{f_2}) \tag{15}$$

in whcich $W_{f_1} \in \mathbb{R}^{f \times n_c}, W_{f_2} \in \mathbb{R}^{k \times f}$ are projection matrices, and $b_{f_1} \in \mathbb{R}^f, b_{f_2} \in \mathbb{R}^k$ is a bias vector for W_{f_1}, W_{f_2} with $s \in \mathbb{R}^k$. Eventually, through the above processes, our CNN architecture becomes a function that takes a raw document as input, and returns a latent vector of each document as output:

$$s_j = cnn(W, Y_j) \tag{16}$$

where W denotes all the weight and bias variables to prevent clutter, Y_j denotes a raw document of item j, and s_j denotes a document's latent vector of item j.

3.4. Optimization

To optimize the variables such as users' latent factors, items' latent factors, weight and bias parameters of aSDAE and CNN, we use maximum a posteriori estimation as follow.

$$\begin{aligned} &\max_{U,V,W^+,W} p(U, V, W^+, W | R, X, Y, \sigma^2, \sigma_U^2, \sigma_V^2, \sigma_{W^+}^2, \sigma_W^2) \\ &= \max_{U,V,W^+,W} [p(R|U, V, \sigma^2) p(U|W^+, X, \sigma_U^2) p(W^+|\sigma_{W^+}^2) p(V|W, Y, \sigma_V^2) p(W|\sigma_W^2)] \end{aligned} \tag{17}$$

If we give a negative logarithm on Equation (17), it can be reformulated as follow.

$$\mathcal{L}(U,V,W^+,W) = \sum_i^N \sum_j^M \frac{I_{ij}}{2}(R_{ij} - u_i^T v_j)^2 + \frac{\lambda_U}{2}\sum_i^N ||u_i - asdae(W^+, X_i)||_F^2 \tag{18}$$
$$+ \frac{\lambda_V}{2}\sum_j^M ||v_j - cnn(W, Y_j)||_F^2 + \frac{\lambda_{W^+}}{2}\sum_k^{|w_k^+|} ||w_k^+||_2^2 + \frac{\lambda_W}{2}\sum_k^{|w_k|} ||w_k||_2^2$$

in which λ_U is σ^2/σ_U^2, λ_V is σ^2/σ_V^2, λ_W^+ is $\sigma_2/\sigma_{W^+}^2$ and λ_W is σ^2/σ_W^2. We adopt coordinate ascent,which iteratively optimizes a latent variable while fixing the remaining variables. Specifically, Equation (17) becomes a quadratic function with respect to U (or V) while temporarily assuming W and V (or W^+ and U) can be analytically computed in a closed form by simply differentiating the optimization function $\mathcal{L}$ with respect to u_i (or v_j) as follows.

$$u_i = (VI_iV^T + \lambda_U I_K)^{-1}(VR_i + \lambda_U asdae(W^+, X_i)) \tag{19}$$
$$v_j = (UI_jU^T + \lambda_V I_K)^{-1}(UR_j + \lambda_V cnn(W, Y_j)) \tag{20}$$

where I_i is a diagonal matrix with I_{ij},$j = 1, ..., M$ as its diagonal elements and R_i is a vector with $(R_{ij})_{j=1}^M$ for user i. For item j, I_j and R_j are similarly defined as I_i and R_i, respectively. Equations (19) and (20) show the updated formula of user's latent factor u_i and item's latent factor v_j, respectively, where λ_U and λ_V are balancing parameters as in Wang and Blei (2011). Unfortunately, W^+ and W cannot be optimized by an analytic solution as we do for U and V because W^+ and W are closely related to the features in aSDAE and CNN architecture. Nonetheless, we observe that $\mathcal{L}$ can be interpreted as a squared error function with L_2 regularized terms as follows when U and V are temporarily constant.

$$\Phi(W^+) = \frac{\lambda_U}{2}\sum_i^N ||u_i - asdae(W^+, X_i)||_F^2 + \frac{\lambda_{W^+}}{2}\sum_k^{|w_k^+|} ||w_k^+||_2^2 + constant \tag{21}$$

$$\Phi(W) = \frac{\lambda_V}{2}\sum_j^M ||v_j - cnn(W, Y_j)||_F^2 + \frac{\lambda_W}{2}\sum_k^{|w_k|} ||w_k||_2^2 + constant \tag{22}$$

To optimize W^+ and W, we use the back .propagation algorithm.

The overall optimization process (U,V,W^+ and W are alternatively updated) is repeated until convergence. With optimized U,V,W^+ and W, finally we can predict ratings of users on items:

$$\hat{R}_{ij} = \mathbb{E}[R_{ij}|u_i^T v_j, \sigma^2] = u_i^T v_j = (asdae(W^+, X_i) + \varepsilon_i)^T (cnn(W, Y_j) + \epsilon_j) \tag{23}$$

in which $\hat{R}_{ij}$ is the predicted value of rating.

4. Experiment

In this section, we evaluate the performance of our PHD model with four real-word datasets from different domains and compare our PHD model with five state-of-the-art algorithms.

Table 1: Statistics of datasets

Dataset	User side information	Item side information	Users	Items	Ratings	Density
ML-100K	tags	movie descriptions	943	1,546	94,808	6.503%
ML-1M	gender/age/occupation/zipcode	movie descriptions	6,040	3,544	993,482	1.413%
ML-10M	tags	movie descriptions	69,878	10,073	9,945,875	4.641%
Amazon	demographic characteristics	reviews	81,339	18,203	238,352	0.016%

4.1. Experiment Setting

Our experiment environment is in Google Cloud Platform[4] with 24 CPU (Xeon(R), 2.2GHz), 2 GPU(K80, 24G memory), 90G RAM and 128G SSD. We use keras as the deep learning framework and employ tensorflow as the background. Our implementation is available at https://github.com/daicoolb/PHDMF

Datasets

To manifest the effectiveness of our model in terms of rating prediction, we employ four datasets, three from Movielens[5] and one from Amazon[6] for our experiment. The first three datasets, MovieLens-100k (ML-100K), MovieLens-1M (ML-1M) and MovieLens-10M (ML-10M), are universally used for evaluating the performance of recommender systems. The ML-100K includes more than 90 thousand ratings from 943 users on 1,546 items, ML-1M contains over 1 million ratings from 6,040 users on 3,706 movies, and ML-10M contains 69,878 users and 10,073 items with nearly 10 million ratings. Amazon instant videos has 81,339 users and 18,203 items with 9,945,875 ratings after selecting the user whose minimum rating is 2 and by ignoring items without reviews. Table 1 shows the side information[7] and statistics of datasets we used in the experiment.

Baselines

To demonstrate the performance of our model, we compare it with three traditional methods, which are NMF, PMF and SVD as well as two deep learning methods, which are aSDAE and ConvMF. As far as that our model is based on PMF, we compare it with NMF Zhang et al. (2006) and SVD, which is the most common method in production environment. Note that some traditional approaches, such as Collective Matrix Factorization (CMF) Singh and Gordon (2008) and Kernelized Probabilistic Matrix Factorization (KPMF) Zhou et al. (2012) which adopt side information, are superior than PMF and NMF. However, this fact proves trivial when compared with SVD. Hence, we employ SVD instead of CMF and KPMF. Furthermore, in order to show that our model can extract effective representations from both sides (i.e., users and items), we contrast it with aSDAE Dong et al. (2017) which ignores item side information (i.e., abstract, review or description, etc.) and ConvMF Kim et al. (2016) which neglects user side information (i.e., age, country, genre or demographics, etc.).

Evaluation Metrics

Evaluation measures for recommender systems are usually divided into three categories Davoudi and Chatterjee (2016): 1) predictive accuracy measures (such as Mean Absolute

4. https://cloud.google.com/
5. https://grouplens.org/datasets/movielens/
6. http://jmcauley.ucsd.edu/data/amazon/
7. Note that movie descriptions can be collected in http://www.imdb.com/

Table 2: Parameter settings of different methods

Method	ML-100K	ML-1M	ML-10M	Amazon
NMF	$\lambda_U = 0.06, \lambda_V = 0.06$	$\lambda_U = 0.06, \lambda_V = 0.06$	$\lambda_U = 0.08, \lambda_V = 0.08$	$\lambda_U = 0.07, \lambda_V = 0.07$
PMF	$\lambda_U = 0.01, \lambda_V = 0.01$	$\lambda_U = 0.01, \lambda_V = 0.01$	$\lambda_U = 0.01, \lambda_V = 0.01$	$\lambda_U = 0.05, \lambda_V = 0.05$
SVD	$\lambda_U = 0.005, \lambda_V = 0.005$	$\lambda_U = 0.01, \lambda_V = 0.01$	$\lambda_U = 0.01, \lambda_V = 0.01$	$\lambda_U = 0.01, \lambda_V = 0.01$
aSDAE	$\lambda_U = 5, \lambda_V = 200$	$\lambda_U = 10, \lambda_V = 100$	$\lambda_U = 10, \lambda_V = 100$	$\lambda_U = 1, \lambda_V = 100$
ConvMF	$\lambda_U = 80, \lambda_V = 20$	$\lambda_U = 200, \lambda_V = 5$	$\lambda_U = 100, \lambda_V = 1$	$\lambda_U = 1, \lambda_V = 150$
PHD	$\lambda_U = 2.5, \lambda_V = 250$	$\lambda_U = 3, \lambda_V = 250$	$\lambda_U = 15, \lambda_V = 80$	$\lambda_U = 250, \lambda_V = 1$

Table 3: Average RMSE on ML-100K and ML-1M of different methods

	Ratio of training set (ML-100k)				Ratio of training set (ML-1M)			
Method	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8
NMF	1.06720	1.01020	0.97920	0.97400	0.96690	0.92970	0.92360	0.91250
PMF	1.12140	1.03940	0.97970	0.95220	0.99830	0.93100	0.91220	0.88900
SVD	**0.97470**	0.96100	0.94660	0.94530	0.93330	0.91140	0.89300	0.86840
aSDAE	1.06475	0.98455	0.95495	0.94193	0.93778	0.90409	0.89151	0.87190
ConvMF	1.08427	1.00367	0.96441	0.95215	0.99910	0.91540	0.88706	0.85909
PHD	0.99019	**0.96517**	**0.94185**	**0.93259**	**0.91696**	**0.88070**	**0.86543**	**0.84895**

Error (MAE), Root Mean Squared Error (RMSE)) which evaluate how accurately the recommender system is in predicting rating values, 2) classification accuracy measures (such as precision and recall) which measure the frequency with which a recommender system makes correct/incorrect decisions, and 3) rank accuracy measures (such as discounted cumulative gain and mean average precision) which evaluate the correctness of the ordering of items performed by the recommendation system. Since our purpose is to conduct rating prediction, we employ RMSE and MAE as the evaluation metrics. Generally, RMSE and MAE can be formulated as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i,j} Z_{i,j}^{P}(R_{ij} - \hat{R}_{ij})^2} \tag{24}$$

$$MAE = \frac{\sum_{i,j} Z_{i,j}^{P}|R_{ij} - \hat{R}_{ij}|}{N} \tag{25}$$

where N is the total number of ratings in the test set, and Z_{ij}^{P} is a binary matrix that indicates test ratings.

Parameter Settings

For all the compared methods, we train them with different percentages (20%, 40%, 60%, 80%), which means that we use the specific ratio to randomly generate a training set and the remaining data is used as the test set. Table 2 shows the parameters of different methods. For the latent dimension of U and V, we all set 50 according to previous work Wang et al. (2015a). The corrupted ratio is 0.4 and layer l is 4 in aSDAE. Recall that other parameters such as the learning rate in NMF, PMF and SVD can be selected according to the regularization parameters (i.e., λ_U, λ_V). We repeat the evaluation five times with different randomly selected training sets and the average performance is reported.

Table 4: Average RMSE on ML-10M and Amazon of different methods

	Ratio of training set (ML-10m)				Ratio of training set (Amazon)		
Method	0.2	0.4	0.6	0.8	0.4	0.6	0.8
NMF	0.90481	0.88182	0.87413	0.86912	1.42063	1.36050	1.31092
PMF	0.93842	0.88161	0.85072	0.82703	1.37080	1.37091	1.37462
SVD	0.87920	0.83712	0.81283	0.79521	1.14239	1.05667	1.0139
aSDAE	0.83710	0.80740	0.79232	0.78014	1.53697	1.31685	1.15460
ConvMF	0.86602	0.83894	0.82795	0.80742	1.68026	1.37892	1.24843
PHD	**0.83530**	**0.80651**	**0.79220**	**0.77991**	**1.13340**	**1.04011**	**0.97354**

Table 5: Impacts of λ_u and λ_v on four datasets

	Density 6.503%			ML_100K					
λ_U	1	2.5	3	3	3	3	10	10	15
λ_V	150	250	50	80	100	150	150	180	180
RMSE	0.94014	**0.93259**	0.95483	0.94687	0.94108	0.93883	0.95478	0.95218	0.95218
	Density 1.413%			ML_1M					
λ_U	1	1	1	3	3	3	3	3.5	3.5
λ_V	150	200	250	150	200	250	300	250	300
RMSE	0.86057	0.86832	0.85528	0.85092	0.85065	**0.84895**	0.85066	0.85023	0.84976
	Density 4.641%			ML_10M					
λ_U	0.1	1	15	20	20	20	20	20	30
λ_V	100	100	80	100	120	150	350	500	100
RMSE	0.91775	0.82130	**0.77991**	0.78004	0.78070	0.78061	0.78259	0.78750	0.78183
	Density 0.016%			Amazon					
λ_U	80	100	100	100	100	150	200	250	350
λ_V	1	1	2	3	5	1	1	1	1
RMSE	0.99895	0.98014	0.98684	0.99606	0.99989	0.99218	0.99187	**0.97354**	0.99151

4.2. Experiment Results

Tables 3 and 4 show the average RMSE of NMF, PMF, SVD, aSDAE, ConvMF and our PHD model with different percentages of testing data on the four datasets. First, we can observe from Tables 3 and 4 that aSDAE and ConvMF achieve better performance than NMF and PMF. This implies the effectiveness of incorporating side information. However, when the ratio of training set is below 0.4, we see a different trend emerging in that NMF and PMF have lower RMSE value when compared with aSDAE and ConvMF. It demonstrates that when the data matrix is too sparse, aSDAE and ConvMF cannot effectively extract latent factors. Nonetheless, in almost locations, our PHD model outperforms the above methods except that when raito is 0.2 in ML-100K when compared with SVD. That is, deep structures can create better feature quality of side information especially combining aSDAE and CNN together to extract more effective latent factors. As for the ratio with 0.2 in ML-100K, we believe that the side information is inadequate in a small dataset to effectively extract features. Furthermore, from Tables 3 and 4, we can see that our PHD model obtains lower RMSE than aSDAE and ConvMF, which validates the strengths of the latent factors learned by our model. At the same time, we can conclude that *even when the data is too sparse, the PHD model still has robust and good performance when compared with traditional approaches and other deep learning models.* In addition, Figure 5 shows the

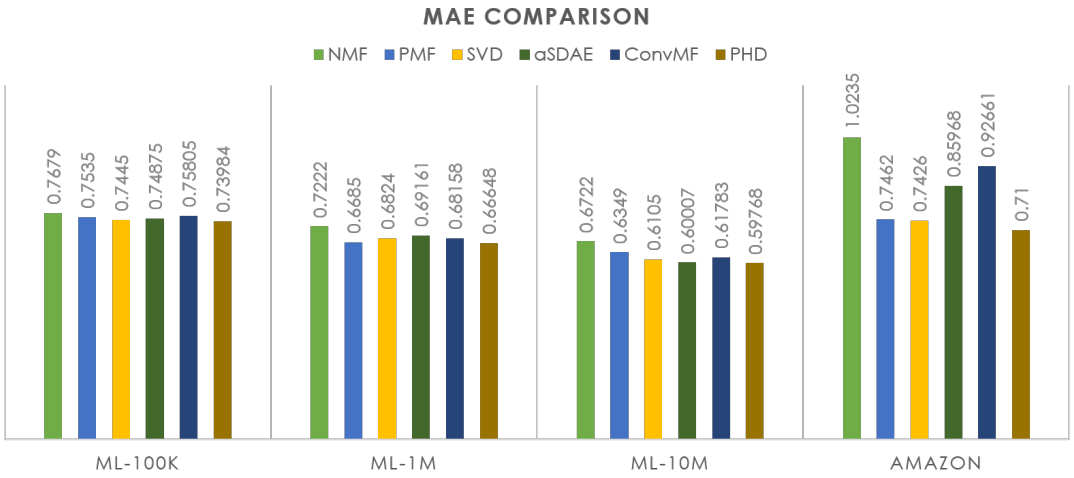

Figure 5: Average MAE of four datasets on training sets whose ratio are 0.8

average MAE, compared with different methods, on the entire training sets whose ratio are 0.8. From the above, our model PHD still achieves good performance even when compared with the most popular method SVD. Therefore, the RMSE and MAE metrics demonstrate the effectiveness of our PHD model.

Table 5 shows the impacts of λ_U and λ_V on four datasets. Regarding the changes from different datasets, we observe that when the rating data becomes sparse, λ_U increases while λ_V decreases to produce the best results. At the same time, compared with the aSDAE and ConvMF's RMSE in different ratios of training sets from Tables 3 and 4, we can conclude that it is more effective to extract users' latent factors from aSDAE and extract items' latent factors from ConvMF, respectively. In other words, it implies that *in different situations with sparse data, our PHD model can still effectively extract features according to both user and item side information, and this can alleviate the data sparsity problem.* Moreover, based on the above comparison, we can also conclude that our PHD model has a more effective feature representation by combining aSDAE and ConvMF.

5. Conclusion and Future Work

In this paper, we present a hybrid collaborative filtering model that combines aSDAE and CNN into a probabilistic matrix factorization. Our proposed model can learn effective latent factors from both user-item rating matrix and side formation for both users and items. Furthermore, our model is based on aSDAE and ConvMF and can effectively learn users and items' latent factors, respectively. Our experimental results present that our model outperforms five state-of-the-art algorithms. As for part of future work, we will think about how to reduce the fine-tuning time in deep learning models with recommender systems. In addition, we want to use a recurrent neural network to extract time features composing users' latent factors in Collaborative Filtering.

References

Anahita Davoudi and Mainak Chatterjee. Modeling trust for rating prediction in recommender systems. In *SIAM Workshop on Machine Learning Methods for Recommender Systems, SIAM*, pages 1–8, 2016.

Xin Dong, Lei Yu, Zhonghuo Wu, Yuxia Sun, Lingfeng Yuan, and Fangxi Zhang. A hybrid collaborative filtering model with deep structure for recommender systems. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1309–1315, 2017.

Koray Kavukcuoglu, Marc'Aurelio Ranzato, Rob Fergus, and Yann LeCun. Learning invariant features through topographic filter maps. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1605–1612, 2009.

Dong Hyun Kim, Chanyoung Park, Jinoh Oh, Sungyoung Lee, and Hwanjo Yu. Convolutional matrix factorization for document context-aware recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 233–240, 2016.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751, 2014.

Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Honglak Lee, Peter T. Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems*, pages 1096–1104, 2009.

Sheng Li, Jaya Kawale, and Yun Fu. Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 811–820, 2015.

Guang Ling, Michael R. Lyu, and Irwin King. Ratings meet reviews, a combined approach to recommend. In *Eighth ACM Conference on Recommender Systems, RecSys '14*, pages 105–112, 2014.

Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

Karol J. Piczak. Environmental sound classification with convolutional neural networks. In *25th IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–6, 2015.

Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011. ISBN 978-0-387-85819-7.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

Ajit P. Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 650–658. ACM, 2008. ISBN 978-1-60558-193-4.

Michele Trevisiol, Luca Maria Aiello, Rossano Schifanella, and Alejandro Jaimes. Cold-start news recommendation with domain-dependent browse graph. In *Eighth ACM Conference on Recommender Systems, RecSys '14*, pages 81–88, 2014.

Aäron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013.*, pages 2643–2651, 2013.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 1096–1103, 2008.

Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 448–456, 2011.

Hao Wang, Xingjian Shi, and Dit-Yan Yeung. Relational stacked denoising autoencoder for tag recommendation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 3052–3058, 2015a.

Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244, 2015b.

Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, pages 549–553, 2006.

Ke Zhou, Shuang-Hong Yang, and Hongyuan Zha. Functional matrix factorizations for cold-start recommendation. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 315–324, 2011.

Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *Proceedings of the Twelfth SIAM International Conference on Data Mining*, pages 403–414, 2012.