國　立　中　央　大　學

資訊工程學系軟體工程碩士班

碩　士　論　文

基於SVD模型之變形 – WSVD 與 PSVD

Variants of the SVD model – WSVD and PSVD

研 究 生：陳　璞

指導教授：施國琛

共同指導：陳弘軒

中 華 民 國　一百零七　年　六　月

摘要

隨著網際網路的發展，人們面臨越來越多的選擇，例如購物網站中商品的選擇，影音網站中該看哪些影片的選擇。推薦系統在這些網站中扮演幫人們迅速決定的重要角色。在這篇論文中，我們針對在推薦系統中的知名方法潛在因子模型(Latent factor model)進行分析與改良，並提出了加權潛在因子模型和多項式潛在因子模型。這兩個模型分別賦予了傳統潛在因子模型權重參數和非線性的特徵組合。我們將這兩個模型對五種開放資料集進行了許多實驗，發現相較於傳統模型，這兩個模型能夠有更好地預測效果。由於我們提出的模型是基於潛在因子模型的變體，我們的模型也可以應用於其他潛在因子模型上，如SVD ++模型和NMF模型。

關鍵字: 推薦系統、SVD模型、矩陣分解模型

Abstract

With the development of the Internet, people are faced with more and more choices, such as the choice of products in shopping websites and the choice of which videos to watch in video and audio websites. The recommendation system plays an important role in these sites to help people decide quickly. In this paper, we analyze the well known method – the latent factor model in the recommendation system, and propose the weighted latent factor model and the polynomial latent factor model. These two models respectively give the traditional latent factor model weights and nonlinear feature combinations. We conducted many experiments on these two models for the five open data sets and found that the two models have better predictive effects than the traditional models. Since our proposed model is based on the latent factor models, our model can also be applied to other latent factor models such as SVD ++ model and NMF model.

Keyword: recommender system, SVD model, matrix factorization method

Contents

List of Figures

List of Tables

1 Introduction

With the progress of the Internet in recent years, the information explosion has made people face more and more choices in our daily life. For example, with the growth of e-commerce, there are thousands of shopping websites on the Internet, and these websites have tons of items for selling. Another example, in the old days people have to go to a movie theater when they want to watch movies. Now there are many video streaming services such as Netflix, people can watch the movies they want anywhere, anytime on the Internet. The biggest difference between these services and traditional consumer behavior is that these services do not have the limitation of space and time. The customers can easily browse a lot of products on the website in a short time, but how to quickly find the desired products from a large number of products is the first consideration for users when selecting these services. From the business perspective, how to present the right products to the needy customers and increase the revenue and customer stickiness is the primary goal of operating these services. Therefore, the recommender system is very important in the modern days.

For recommendation, an intuitive way is to recommend the most popular item to the users, and if the user is viewing a specific item, recommend the items similar to it. Another way is finding users who are similar to our target user, recommend the items they were interested in our target user. These approaches are known as the content filtering, with enough information of the users and items, these strategies can be very successful. However, collecting external information may raise privacy concerns, it's might not be easy or available to collect in modern days.

An alternative to the content filtering relies only on past user behavior is the collaborative filtering. The most popular collaborative filtering technique is the matrix factorization method (MF, also known as SVD model and latent factor model), popularized by the Netflix Prize started in October 2006 [17]. MF discovers the vector of the latent factors for each user and each item from their interaction (e.g., give a rating, click link), and using their inner product to predict their interaction. MF can also incorporate with other information, such as external information [18], users' bias, and items' bias to improve the accuracy of prediction. Therefore, MF is widely used due to their effectiveness, easy implementation, and domain-independent.

Although MF is very powerful, the inner product operation in MF might be too simple. Recent year, Deep Learning is world famous because AlphaGo dominates the Chinese chess game - Go. In Deep Learning, the input feature pass through very complicated network structure, the activation function also bring the non-linearity to the interaction between the features. Bringing non-linearity into MF might improve the quality of the prediction. Another feature in MF is that each latent factor is equally weighted, this might not making sense in the reality.

In this thesis, we propose the polynomial latent factor model and weighted latent factor model. The polynomial latent factor model, bringing the non-linearity into traditional MF. In the polynomial latent factor model, we assume the interaction between each latent factor is not independent but pair to pair. The weighted latent factor model assigns different weight to each latent factor, we set the weight of each factor to one in the beginning and gradually learn the weights during the training phase.

We also found that the WSVD model often has problems with gradients when it comes to specific data. In response to this problem, we added a similar method to the clipping method to optimize the WSVD model and achieved good results. We also tried some variants of the WSVD model in the hope of solving the cold start problem.

The rest of the thesis is organized as follows. Chapter 2 first describes the preliminary knowledge about MF and an overview of the problem, then introduces our proposed method, the weighted latent factor model, and the polynomial latent factor model. Chapter 3 reports the experiments and results, we also reports the experiments of variants of the WSVD model, the GWSVD model. Chapter 4 briefly introduce the related works in the recommender system, and the hyper-parameter optimization method we used in the experiments. Chapter 5 conclude our work with our advancements.

2 Method

In this chapter, we first define the rating prediction problem and discuss the latent factor model which is a popular solution for collaborative filtering. After defining the problem, we present two methods which are based on the traditional SVD latent factor model. The first one is the weighted latent factor model, the WSVD model. Following next is the polynomial latent factor model, the PSVD model.

2.1 Preliminaries

2.1.1 Problem definition

Let m and n denote the number of users and items, respectively. We can define the user-item rating matrix as $R \in \mathbb{R}^{m \times n}$ which can hold all possible ratings between users and items. The $r_{ij} \in R$ denoted the rating value user i gave to item j. To predict the ratings, the basic idea is creating another matrix $\hat{R}$ to approximate matrix R, making the distance between $\hat{R}$ and R as close as possible. The distance between $\hat{R}$ and R can be measured by an error function, thus the recommendation problem becomes an optimization problem. Here, we use the mean square error function to measure the distance between $\hat{R}$ and R, the equation is as follows:

$$L(\hat{r}) = \frac{1}{|G|} \sum_{\forall (i,j) \in G} (r_{ij} - \hat{r}_{ij})^2 \tag{2.1}$$

Here, G denotes the user-item pairs we have observed, $\hat{r}_i j$ denotes the rating prediction between user i and item j. In practice, most of the users only interact with a few items, which means that matrix R is very sparse. Although matrix R is very sparse, we can't take the value of zero as the realistic elements in the matrix, these zero values are the values that need to predict. The distance only measured if the rating is already observed.

2.1.2 Latent factor model

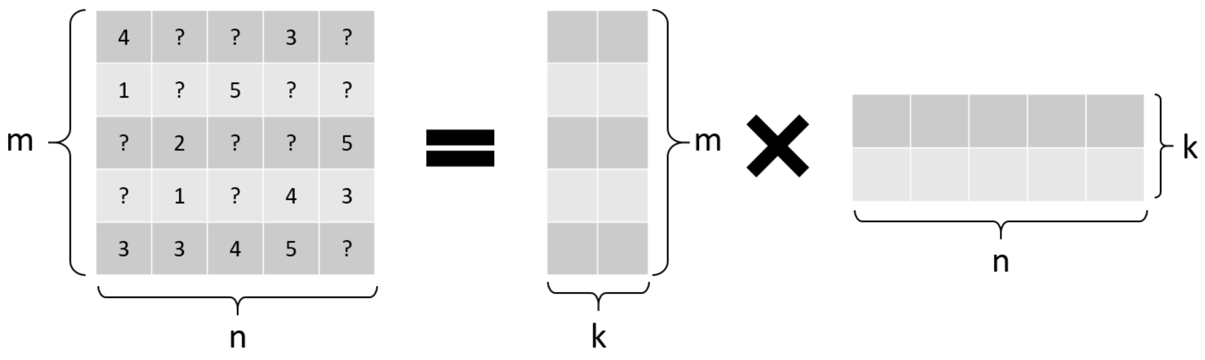


FIGURE 2.1: The illustration of SVD model

2.1.2.1 Basic latent factor model

In the real world, the number of users and items in recommender systems is very large. For example, on the online shopping website, hundreds of thousands of items are selling online, and the users can be all around the world, creating entire matrix $\hat{R}$ would cost too many memory spaces. The popular solution is the matrix factorization method, also known as the SVD model, and also known as the latent factor model lately in order to distinguish from the classic singular value decomposition in linear algebra.

The idea behind the SVD model is very simple. SVD breaks the large matrix R into the product of two small matrices, as shown in Figure 2.1.

Let us now explain the SVD model in detail with mathematics. Suppose we want to approximate matrix R as the inner product of two matrixes:

$$R \approx P \cdot Q^T, \tag{2.2}$$

where P is an $m \times k$ matrix, Q is an $n \times k$ matrix, and k is a number which is much smaller than the number of the users and items. These two matrixes represent low dimensional latent factors of the users and items. In this case, the rating of user i's rating r_{ij} on item j is evaluated base on the following equation:

$$\hat{r}_{ij} = p_i \cdot q_j^T = \sum_{l=1}^{k} p_{il} q_{jl}, \tag{2.3}$$

where $p_i \in \mathbb{R}^{1\times k}$ and $q_j \in \mathbb{R}^{1\times k}$ in matrix P and Q are the latent factors representation of user i and item j respectively. These factors of items can regard as items' attributes, and the latent factors of the users can regard as how much the user like on the corresponding item factor.

2.1.2.2 Biased latent factor model

In equation 2.3, the rating prediction only based on the interaction between users and items. However, it's often that some users tend to give higher or lower rating value than others, and some items is very popular making it easily receive higher rating value, these are known as bias. To add the bias into the SVD model, we first define the bias involved in r_{ij} as follow:

$$b_{ij} = \mu + b_i^{(U)} + b_j^{(I)} \tag{2.4}$$

The bias involved in rating r_{ij} is denoted by b_{ij}. The overall average rating is denoted by μ; the parameters $b_i^{(U)}$ and $b_j^{(I)}$ indicate the biases of user i and item j, respectively, from the average.

After defining the bias involved in rating r_{ij}, the equation 2.3 could be expanded by adding b_{ij} into it:

$$\begin{aligned} \hat{r}_{ij} &= p_i \cdot q_j^T + b_{ij} \\ &= p_i \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)} \end{aligned} \tag{2.5}$$

Now, our rating prediction contains four components: user-item interaction, global average, user bias, and item bias.

2.1.2.3 Learning algorithms

As mentioned in section 2.1.1, we want to minimize the loss function 2.1. To do this, we replace the $\hat{r}_{ij}$ in 2.1 with the SVD model as follow:

$$\begin{aligned} L(\hat{r}) &\equiv \frac{1}{|G|} \sum_{\forall(i,j)\in G} (r_{ij} - \hat{r}_{ij})^2 \\ &= \frac{1}{|G|} \sum_{\forall(i,j)\in G} (r_{ij} - (p_i \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)}))^2 \end{aligned} \tag{2.6}$$

After replaced the $\hat{r}_{ij}$, the learning process of the SVD model became an optimization problem: finding the parameters Θ that minimize the loss function L which can be defined by equation 2.7.

$$\begin{aligned} L(\Theta) &\equiv \sum_{\forall(i,j)\in G} (r_{ij} - (p_i \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)}))^2 + \lambda||\Theta||_2 \\ &= \sum_{\forall(i,j)\in G} (r_{ij} - (p_i \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)}))^2 \\ &\quad + (\lambda_f||P||_2 + \lambda_f||Q||_2 + \lambda_b||b^{(U)}||_2 + \lambda_b||b^{(I)}||_2), \end{aligned} \tag{2.7}$$

where $P = [p_1, p_2, ..., p_m] \in \mathbb{R}^{m\times k}, Q = [q_1, q_2, ..., q_n] \in \mathbb{R}^{n\times k}, b^{(U)} = [b_1^{(U)}, b_2^{(U)}, ..., b_m^{(U)}] \in \mathbb{R}^{1\times m}, b^{(I)} = [b_1^{(I)}, b_2^{(I)}, ..., b_n^{(I)}] \in \mathbb{R}^{1\times n}$, and $\Theta = (P, Q, b^{(U)}, b^{(I)})$ is the set of unknown parameters to learn. To avoid over-fitting, the Frobenius norm ($|| \cdot ||_2$) is used for regularization, $\lambda = (\lambda_f, \lambda_b)$ is the set of the regularization coefficients.

There are a lot of ways to solve the optimization problem, for the SVD model, the most popular way is the stochastic gradient descent (SGD). To find a local minimum of the loss function 2.7 using gradient descent, each parameter in Θ takes steps proportional to the negative of the gradient of the function 2.7 at the current point.

Let $p_i = [p_{i1}, p_{i2}, ..., p_{ik}], q_j = [q_{j1}, q_{j2}, ..., q_{jk}]$, the partial derivatives of the loss function with respect to the parameters based on user i's rating on item j are given in equation 2.8 to equation 2.11.

$$\frac{\partial L(\Theta)}{\partial b_i^{(U)}} = -(r_{ij} - p_i \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)}) + \lambda_b b_i^{(U)} \tag{2.8}$$

$$\frac{\partial L(\Theta)}{\partial b_j^{(I)}} = -(r_{ij} - p_i \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)}) + \lambda_b b_j^{(I)} \tag{2.9}$$

$$\frac{\partial L(\Theta)}{\partial p_i} = -(r_{ij} - p_i \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)})q_j + \lambda_f p_i \tag{2.10}$$

$$\frac{\partial L(\Theta)}{\partial q_j} = -(r_{ij} - p_i \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)})p_i + \lambda_f q_j \tag{2.11}$$

Algorithm 1 shows the optimization algorithm based on the SGD. The algorithm first generates two latent factor matrixes P and Q from the normal distribution with zero mean and 0.1 variance, two bias vectors $b^{(U)}$ and $b^{(I)}$ filling with zeros, then iterates over each (user, item) pair in $\boldsymbol{G}$ and updates the parameters with corresponding gradient for multiple epochs until the termination condition is met. The learning rate η is used to control how much we are adjusting the weights of our model with respect to the loss gradient.

Another learning method of the SVD model is alternating least squares. Although Equation 2.1 is non-convex since both p_i and q_j are unknowns, we can still fix one of them, making the optimization problem becomes quadratic and can be solved optimally. The learning process rotate between fixing the p_i's and fixing the q_j's. When all p_i's are fixed, the system recomputes the q_j's by solving a least-squares problem, and vice versa.

Algorithm 1: The SVD learning model based on SGD

```
Data: The rated (user, item) pairs $\boldsymbol{G}$, the regularization coefficients $\boldsymbol{\lambda} = (\lambda_f, \lambda_b)$, the
      learning rates $\boldsymbol{\eta} = (\eta_f, \eta_b)$
Result: Model parameters $\boldsymbol{\Theta} = (\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{b}^{(U)}, \boldsymbol{b}^{(I)})$
1  $\boldsymbol{P} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{0.1})$; $\boldsymbol{Q} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{0.1})$; $\boldsymbol{b}^{(U)} \leftarrow \mathbf{0}$; $\boldsymbol{b}^{(I)} \leftarrow \mathbf{0}$; $epoch \leftarrow 0$;
2  repeat
3      for $(i, j) \in \boldsymbol{G}$ do
4          $b_i^{(U)} \leftarrow b_i^{(U)} - \eta_b \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial b_i^{(U)}}$;
5          $b_j^{(I)} \leftarrow b_j^{(I)} - \eta_b \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial b_j^{(I)}}$;
6          $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta_f \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{p}_i}$;
7          $\boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta_f \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{q}_j}$;
8      end
9      $epoch \leftarrow epoch + 1$;
10 until termination condition is met;
```

2.2 Weighted latent factor model

In the SVD model, each of the latent factors of the users and the latent factors of the items are equally weighted, this might not be reasonable. For example, when predicting users' ratings on the movies, the latent factor that implicitly represents the genre of a movie is probably more important than the latent factor that represents the run-time of the movie, in this case, the weight of the genre should be more than the weight of the run-time.

In Equation 2.5, each latent factor in p_i and q_j is equally weighted. To apply different weight to each latent factor, we propose the weighted latent factor model (the WSVD model) in the following equation:

$$\hat{r}_{ij} = (w \odot p_i) \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)}, \tag{2.12}$$

where $w \in \mathbb{R}^{1 \times k}$ is the vector of the weights of the latent factors, the $\odot$ denotes the Hadamard product (i.e., element-wise multiplication) on the two vector w and p_i.

2.2.1 WSVD Learning algorithms

Since the only difference between the SVD model and the WSVD model is the w in Equation 2.12, the stochastic gradient descent can be simply applied to the WSVD model. The loss function of the WSVD model is defined as follow:

$$\begin{aligned}
L(\Theta) &\equiv \sum_{\forall(i,j)\in G} (r_{ij} - ((w \odot p_i) \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)}))^2 + \lambda||\Theta||_2 \\
&= \sum_{\forall(i,j)\in G} (r_{ij} - ((w \odot p_i) \cdot q_j^T + \mu + b_i^{(U)} + b_j^{(I)}))^2 \\
&\quad + (\lambda_w||w||_2 + \lambda_f||P||_2 + \lambda_f||Q||_2 + \lambda_b||b^{(U)}||_2 + \lambda_b||b^{(I)}||_2),
\end{aligned} \tag{2.13}$$

where $w = [w_1, w_2, ..., w_k] \in \mathbb{R}^{1\times k}, P = [p_1, p_2, ..., p_m] \in \mathbb{R}^{m\times k}, Q = [q_1, q_2, ..., q_n] \in \mathbb{R}^{n\times k}, b^{(U)} = [b_1^{(U)}, b_2^{(U)}, ..., b_m^{(U)}] \in \mathbb{R}^{1\times m}, b^{(I)} = [b_1^{(I)}, b_2^{(I)}, ..., b_n^{(I)}] \in \mathbb{R}^{1\times n}$, and $\Theta = (w, P, Q, b^{(U)}, b^{(I)})$ is the set of unknown parameters to learn. To avoid overfitting, the Frobenius norm ($||\cdot||_2$) is used for regularization, $\lambda = (\lambda_w, \lambda_f, \lambda_b)$ is the set of the regularization coefficients.

Let $w = [w_1, w_2, ..., w_k], p_i = [p_{i1}, p_{i2}, ..., p_{ik}], q_j = [q_{1j}, q_{j2}, ..., q_{jk}]$, the partial derivatives of the loss function with respect to the parameters based on user i's rating on item j are given in Equation 2.14 to Equation 2.18.

$$\frac{\partial L(\Theta)}{\partial b_i^{(U)}} = -(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)}) + \lambda_b b_i^{(U)} \tag{2.14}$$

$$\frac{\partial L(\Theta)}{\partial b_j^{(I)}} = -(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)}) + \lambda_b b_j^{(I)} \tag{2.15}$$

$$\frac{\partial L(\Theta)}{\partial w} = -(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)})p_i \odot q_j + \lambda_w w \tag{2.16}$$

$$\frac{\partial L(\Theta)}{\partial p_i} = -(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)})w \odot q_j + \lambda_f p_i \tag{2.17}$$

$$\frac{\partial L(\Theta)}{\partial q_j} = -(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)}) w \odot p_i + \lambda_f q_j \tag{2.18}$$

Algorithm 2 shows the optimization algorithm of the WSVD model. The vector w is initialized by filling with ones, indicate that the weight of each factor is equal before the training, the algorithm will try to learn a proper weight for each factor while iterates over each pair in the ratings set G.

Algorithm 2: The WSVD learning model based on SGD

```
Data: The rated (user, item) pairs $\boldsymbol{G}$, the regularization coefficients $\boldsymbol{\lambda} = (\lambda_w, \lambda_f, \lambda_b, )$,
      the learning rates $\boldsymbol{\eta} = (\eta_w, \eta_f, \eta_b)$
Result: Model parameters $\boldsymbol{\Theta} = (\boldsymbol{w}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{b}^{(U)}, \boldsymbol{b}^{(I)})$
1  $\boldsymbol{w} \leftarrow \mathbf{1}$; $\boldsymbol{P} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{0.1})$; $\boldsymbol{Q} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{0.1})$; $\boldsymbol{b}^{(U)} \leftarrow \mathbf{0}$; $\boldsymbol{b}^{(I)} \leftarrow \mathbf{0}$; $epoch \leftarrow 0$;
2  repeat
3      for $(i, j) \in \mathcal{G}$ do
4          $b_i^{(U)} \leftarrow b_i^{(U)} - \eta_b \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial b_i^{(U)}}$;
5          $b_j^{(I)} \leftarrow b_j^{(I)} - \eta_b \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial b_j^{(I)}}$;
6          $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_w \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{w}}$;
7          $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta_f \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{p}_i}$;
8          $\boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta_f \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{q}_j}$;
9      end
10     $epoch \leftarrow epoch + 1$;
11 until termination condition is met;
```

2.2.2 Vanishing gradient

During the experiment, we found that by using the stochastic gradient descent, the training loss of the WSVD model is easy to stick at a certain point if the dataset is dense enough. This phenomenon making it hard to train the WSVD model. To find out the reason, we analyzed the parameters in the WSVD model during the training, we find out that the values of latent factors of the users and items are shrinking during the training. In Figure 2.2, we pick user 405 and item 50 in the MovieLens-100K dataset and plot the value of their latent factors during each training epoch, they are the most active user and the most rated item in the MovieLens-100K dataset. As the figure shows, the value of the weights vector w quickly move to zero at the beginning of the training due to the regularization, the values of the latent factor P and Q refer to user 405 and item 50 respectively also gradually move toward zero. This means that as the training move on, the effect of the latent factors would gradually become smaller and smaller because the value of $(w \odot p_i) \cdot q_j^T$ shrinks close to zero, leaving the WSVD model only contains the effect of the biases and the mean rating.

This phenomenon is due to the gradient vanishing problem. From Equation 2.17 and 2.18, the gradient of the latent factors p_i and q_j is count on the value of w, and as the w quickly shrinks to close to zero, the gradients of p_i and q_j also shrink to close to zero, this means that the regularization would dominate the optimization process, making the value of the latent factors p_i and q_j shrinking quickly.

Theoretically, the regularization helps the generalization of the model, but the shrinking value of latent factors p_i and q_j actually destroy the model. Figure 2.3 compares the MSE scores between different λ_w in the WSVD model and the SVD model. When the value of λ_w is at 0.0001 and 0.0005, the WSVD model over-fits the training data, and as the value of λ_w goes higher, the loss of the testing data gradually goes down, but after λ_w reach 0.005, the error stuck after a few epochs, both in training and testing. The loss curve of $\lambda_w = [0.005, 0.01, 0.05]$ are overlapped in Figure 2.3, their loss is all stuck after certain training epoch.

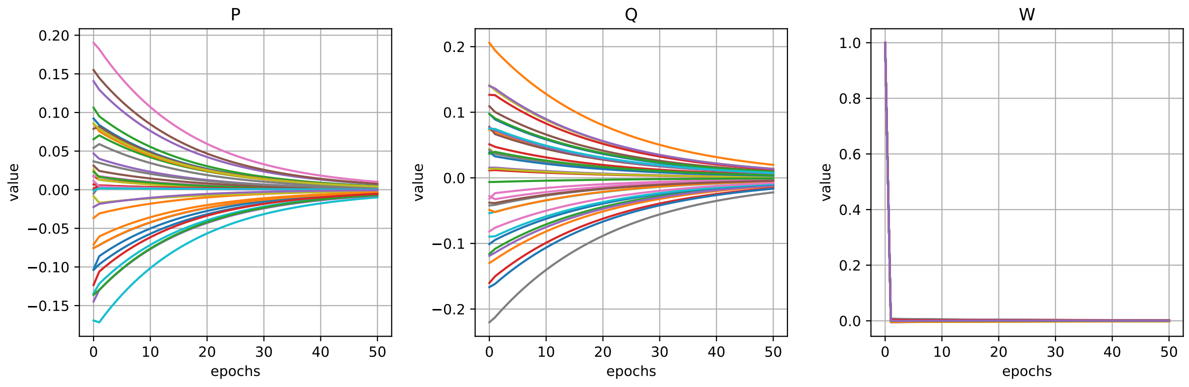


FIGURE 2.2: The latent factor is shrinking quickly during the training process.

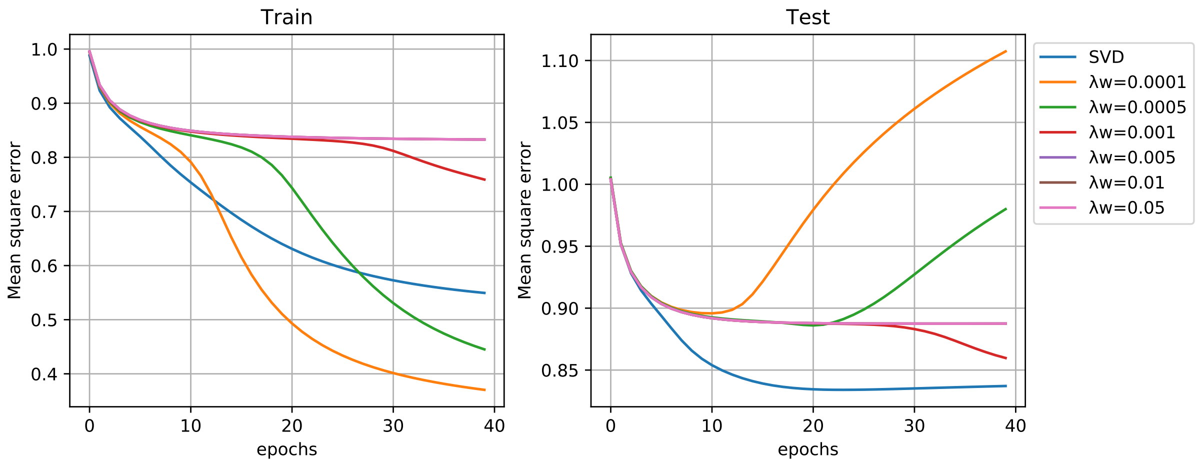


FIGURE 2.3: The loss curve between different λ_w of the WSVD model and the SVD model.

2.2.3 Gradient clipping

To avoid the vanishing problem, set the λ_w lower is an effective way, but making the model easily over-fit the training data. In our experiment, we found that apply sign function 2.19 on the w in the gradient of p_i and q_j can effectively avoid the vanishing problem and improve the generalization performance. In Equation 2.20, 2.21 show the new optimization method for the latent factors p_i and q_j in the WSVD model. The performance of gradient clipping will show in Section 3.

$$sgn(x) = \begin{cases} -1 & : x < 0 \\ 0 & : x = 0 \\ 1 & : x > 0 \end{cases} \tag{2.19}$$

$$p_i \leftarrow p_i - \eta_f(-(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)})sgn(w) \odot q_j + \lambda_f p_i) \tag{2.20}$$

$$q_j \leftarrow q_j - \eta_f(-(r_{ij} - (w \odot p_i) \cdot q_j^T - \mu - b_i^{(U)} - b_j^{(I)})sgn(w) \odot p_i + \lambda_f q_j) \tag{2.21}$$

2.2.4 Comparison between the WSVD model and the SVD model

With m denote the number of users and n denote the number of items, the WSVD model has $(m+n)(k+1)+k$ parameters to optimized, and the SVD model has $(m+n)(k+1)$ parameters. Although the parameters of the WSVD model is more than the SVD model, the expressiveness of these two models are actually identical. For any p_i in the Equation 2.3, p_i can be decomposed into $w \odot p'_i$ in Equation 2.12, and for any $w \odot p'_i$ in Equation 2.12, we can replace with p_i in Equation 2.3. Turns out, the WSVD model is an over-parameterized version of the SVD model. Usually, over-parameterization is believed to lead over-fitting, however, [1] points out over-parameterization may boost the optimization speed and help the model escape from falling into local minimums. In their experiment, the over-parameterization model with simple SGD performs even faster than two well known acceleration method - Adagrad[7] and Adadelta[19].

In the SVD model, non-active users and items have less frequent updating of latent factors, the predictions of them are usually poor. The WSVD model might be able to solve this problem, because the weight vector w might bring addition information from the latent factors that frequently updating. We discuss this hypothesis in Section 3.7.

2.3 Polynomial latent factor model

In the SVD model, the rating is evaluating through the weighted sum of latent factors, which means that each of the latent factors is linear independent. This assumption might not be reasonable in the recent. The popular machine learning model in recent years may be the Support Vector Machine and the Artificial Neural Network. These two methods have one thing in common: the relation between the input features is nonlinear. In the SVM, kernel trick is used to map the input into high-dimensional feature spaces. In Artificial Neural Network, the non-linearity is shown in the architecture of the network and the activation functions. For the recommendation task, Factorization machines [15] is a very powerful method which brings the pairwise feature into model, the pairwise interaction of the features has significant effect on the optimization. Hence, we propose the polynomial latent factor model (the PSVD model) in the following equation:

$$\hat{r}_{ij} = \sum_{l=1}^{k}(p_{il}q_{jl}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)} \tag{2.22}$$

In Equation 2.22, the rating prediction $\hat{r}_{ij}$ is not only based on the sum of the product of the latent factors, but also the two-by-two product of the latent factors. We assume that there are two by two interaction between the latent factors.

2.3.1 PSVD Learning algorithms

The PSVD model is based on the SVD model with nonlinear interaction between each latent factor, so the stochastic gradient descent can be applied on the PSVD model. The loss function of the PSVD model is as follow:

$$\begin{aligned}
L(\Theta) &\equiv \sum_{\forall(i,j)\in G} (r_{ij} - (\sum_{\ell=1}^{k}(p_{i\ell}q_{j\ell}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)}))^2 + \lambda||\Theta||_2 \\
&= \sum_{\forall(i,j)\in G} (r_{ij} - (\sum_{\ell=1}^{k}(p_{i\ell}q_{j\ell}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)}))^2 \\
&\quad + (\lambda_f||P||_2 + \lambda_f||Q||_2 + \lambda_b||b^{(U)}||_2 + \lambda_b||b^{(I)}||_2) \\
&\quad + \lambda_{pf}\sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}^2p_{it}^2 + q_{is}^2q_{it}^2),
\end{aligned} \tag{2.23}$$

where $P = [p_1, p_2, ..., p_m] \in \mathbb{R}^{m\times k}, Q = [q_1, q_2, ..., q_n] \in \mathbb{R}^{n\times k}, b^{(U)} = [b_1^{(U)}, b_2^{(U)}, ..., b_m^{(U)}] \in \mathbb{R}^{1\times m}, b^{(I)} = [b_1^{(I)}, b_2^{(I)}, ..., b_n^{(I)}] \in \mathbb{R}^{1\times n}$, and $\Theta = (P, Q, b^{(U)}, b^{(I)})$ is the set of unknown parameters to learn. To avoid over-fitting, the Frobenius norm ($||\cdot||_2$) is used for regularization, $\lambda = (\lambda_f, \lambda_b, \lambda_{pf})$ is the set of the regularization coefficients.

Let $p_i = [p_{i1}, p_{i2}, ..., p_{ik}], q_j = [q_{1j}, q_{j2}, ..., q_{jk}]$, the partial derivatives of the loss function with respect to the parameters based on user i's rating on item j are given in Equation 2.24 to Equation 2.27.

$$\frac{\partial L(\Theta)}{\partial b_i^{(U)}} = -(r_{ij} - (\sum_{\ell=1}^{k}(p_{i\ell}q_{j\ell}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)})) + \lambda_b b_i^{(U)} \tag{2.24}$$

$$\frac{\partial L(\Theta)}{\partial b_j^{(I)}} = -(r_{ij} - (\sum_{\ell=1}^{k}(p_{i\ell}q_{j\ell}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)})) + \lambda_b b_j^{(I)} \tag{2.25}$$

$$\frac{\partial L(\Theta)}{\partial p_{i\ell}} = (r_{ij} - (\sum_{\ell=1}^{k}(p_{i\ell}q_{j\ell}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)}))(-q_{j\ell} \\ - \sum_{t=1}^{\ell-1}(p_{it}q_{j\ell}q_{jt} - \sum_{t=\ell+1}^{k}(p_{it}q_{j\ell}q_{jt}))) + \lambda_f p_{i\ell} + \lambda_{pf} p_{i\ell}(\sum_{t=1}^{\ell-1} p_{it}^2 + \sum_{t=\ell+1}^{k} p_{it}^2) \tag{2.26}$$

$$\frac{\partial L(\Theta)}{\partial q_{j\ell}} = (r_{ij} - (\sum_{\ell=1}^{k}(p_{i\ell}q_{j\ell}) + \sum_{s=1}^{k-1}\sum_{t=s+1}^{k}(p_{is}p_{it}q_{js}q_{jt}) + \mu + b_i^{(U)} + b_j^{(I)}))(-p_{i\ell} \\ - \sum_{t=1}^{\ell-1}(p_{i\ell}p_{it}q_{jt} - \sum_{t=\ell+1}^{k}(p_{i\ell}p_{it}q_{jt}))) + \lambda_f q_{j\ell} + \lambda_{pf} q_{j\ell}(\sum_{t=1}^{\ell-1} q_{jt}^2 + \sum_{t=\ell+1}^{k} q_{jt}^2) \tag{2.27}$$

Algorithm 3 shows the optimization algorithm based on the SGD. The algorithm is actually the same as the optimization algorithm of the SVD model in algorithm 1, since the PSVD model is the polynomial version of the SVD model.

Algorithm 3: The PSVD learning model based on SGD

```
Data: The rated (user, item) pairs $\boldsymbol{G}$, the regularization coefficients $\boldsymbol{\lambda} = (\lambda_f, \lambda_b, \lambda_{pf})$,
      the learning rates $\boldsymbol{\eta} = (\eta_f, \eta_b)$
Result: Model parameters $\boldsymbol{\Theta} = (\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{b}^{(U)}, \boldsymbol{b}^{(I)})$
1  $\boldsymbol{P} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{0.1})$; $\boldsymbol{Q} \leftarrow \mathcal{N}(\mathbf{0}, \mathbf{0.1})$; $\boldsymbol{b}^{(U)} \leftarrow \mathbf{0}$; $\boldsymbol{b}^{(I)} \leftarrow \mathbf{0}$; epoch $\leftarrow 0$;
2  repeat
3      for $(i, j) \in \boldsymbol{G}$ do
4          $b_i^{(U)} \leftarrow b_i^{(U)} - \eta_b \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial b_i^{(U)}}$;
5          $b_j^{(I)} \leftarrow b_j^{(I)} - \eta_b \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial b_j^{(I)}}$;
6          $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta_f \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{p}_i}$;
7          $\boldsymbol{q}_j \leftarrow \boldsymbol{q}_j - \eta_f \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{q}_j}$;
8      end
9      epoch $\leftarrow$ epoch $+ 1$;
10 until termination condition is met;
```

2.3.2 Comparison between the PSVD model and the SVD model

With m denote the number of users and n denote the number of items, the PSVD model has $(m+n)(k+C_2^k+1)$ parameters to optimized, and the SVD model has $(m+n)(k+1)$ parameters. The number of parameters of the PSVD model is much larger than the SVD model, we compare the the PSVD model with the SVD model which has the same number of parameters with the PSVD model in Section 3.5 in order to ensure the performance of the PSVD model.

2.4 Library implementation

We implemented our proposed model in Python and packaged them into a package for easy experimentation. To improve performance, we use the Cython library. We have also tried to use the deep learning framework to construct the model, but since our model differentiation is relatively simple, using Cython with SGD is actually faster. Figure 2.4 shows the class diagram of our implementation. To find the best model, we can call the random search in library to find the best model for specific dataset. Figure 2.5 shows the flow chart of finding process.

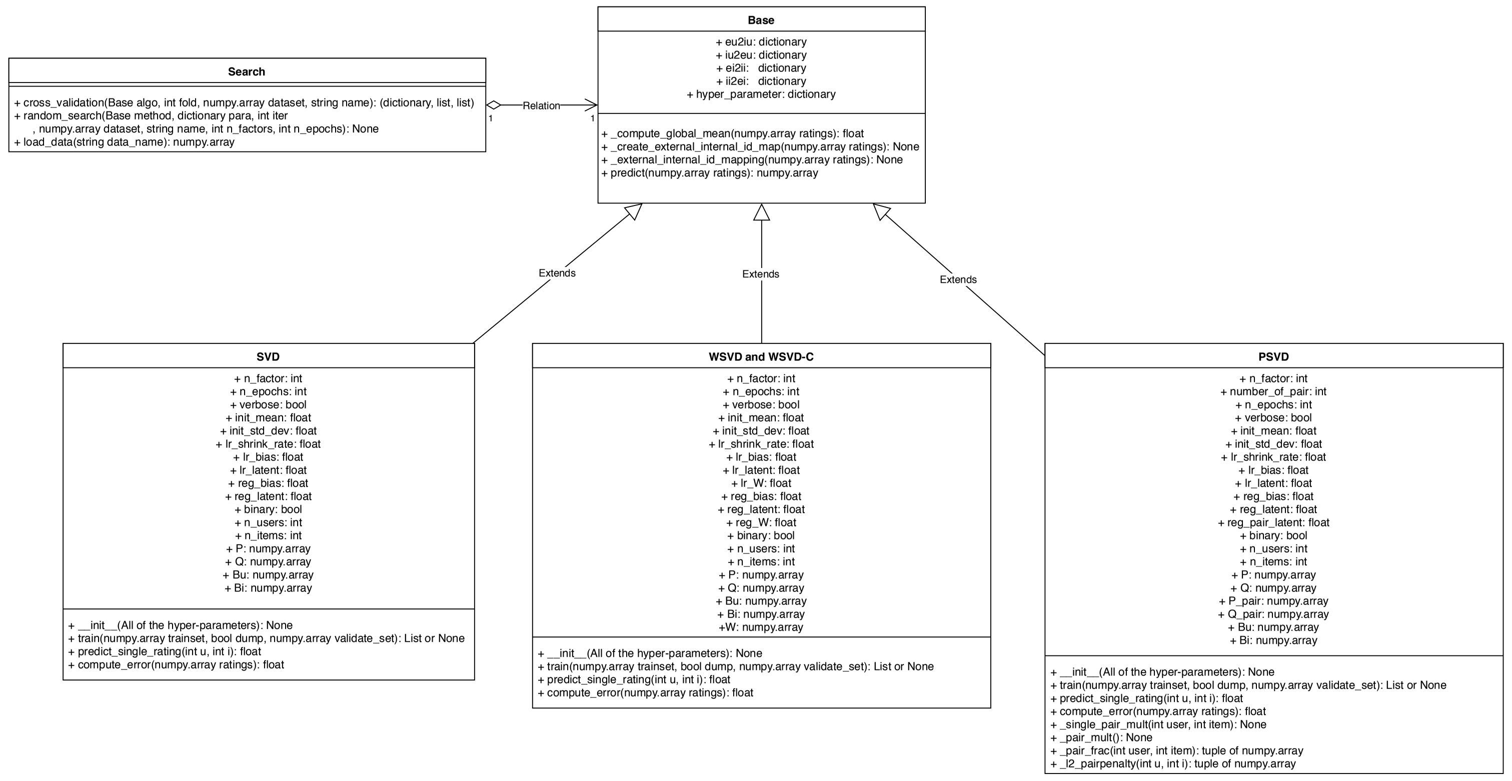

Search
+ cross_validation(Base algo, int fold, numpy.array dataset, string name): (dictionary, list, list)
+ random_search(Base method, dictionary para, int iter
, numpy.array dataset, string name, int n_factors, int n_epochs): None
+ load_data(string data_name): numpy.array
1
Relation
1
Base
+ eu2iu: dictionary
+ iu2eu: dictionary
+ ei2ii: dictionary
+ ii2ei: dictionary
+ hyper_parameter: dictionary
+ _compute_global_mean(numpy.array ratings): float
+ _create_external_internal_id_map(numpy.array ratings): None
+ _external_internal_id_mapping(numpy.array ratings): None
+ predict(numpy.array ratings): numpy.array
Extends
Extends
Extends
SVD
+ n_factor: int
+ n_epochs: int
+ verbose: bool
+ init_mean: float
+ init_std_dev: float
+ lr_shrink_rate: float
+ lr_bias: float
+ lr_latent: float
+ reg_bias: float
+ reg_latent: float
+ binary: bool
+ n_users: int
+ n_items: int
+ P: numpy.array
+ Q: numpy.array
+ Bu: numpy.array
+ Bi: numpy.array
+ __init__(All of the hyper-parameters): None
+ train(numpy.array trainset, bool dump, numpy.array validate_set): List or None
+ predict_single_rating(int u, int i): float
+ compute_error(numpy.array ratings): float
WSVD and WSVD-C
+ n_factor: int
+ n_epochs: int
+ verbose: bool
+ init_mean: float
+ init_std_dev: float
+ lr_shrink_rate: float
+ lr_bias: float
+ lr_latent: float
+ lr_W: float
+ reg_bias: float
+ reg_latent: float
+ reg_W: float
+ binary: bool
+ n_users: int
+ n_items: int
+ P: numpy.array
+ Q: numpy.array
+ Bu: numpy.array
+ Bi: numpy.array
+W: numpy.array
+ __init__(All of the hyper-parameters): None
+ train(numpy.array trainset, bool dump, numpy.array validate_set): List or None
+ predict_single_rating(int u, int i): float
+ compute_error(numpy.array ratings): float
PSVD
+ n_factor: int
+ number_of_pair: int
+ n_epochs: int
+ verbose: bool
+ init_mean: float
+ init_std_dev: float
+ lr_shrink_rate: float
+ lr_bias: float
+ lr_latent: float
+ reg_bias: float
+ reg_latent: float
+ reg_pair_latent: float
+ binary: bool
+ n_users: int
+ n_items: int
+ P: numpy.array
+ Q: numpy.array
+ P_pair: numpy.array
+ Q_pair: numpy.array
+ Bu: numpy.array
+ Bi: numpy.array
+ __init__(All of the hyper-parameters): None
+ train(numpy.array trainset, bool dump, numpy.array validate_set): List or None
+ predict_single_rating(int u, int i): float
+ compute_error(numpy.array ratings): float
+ _single_pair_mult(int user, int item): None
+ _pair_mult(): None
+ _pair_frac(int user, int item): tuple of numpy.array
+ _l2_pairpenalty(int u, int i): tuple of numpy.array

FIGURE 2.4: Class diagram of our lib

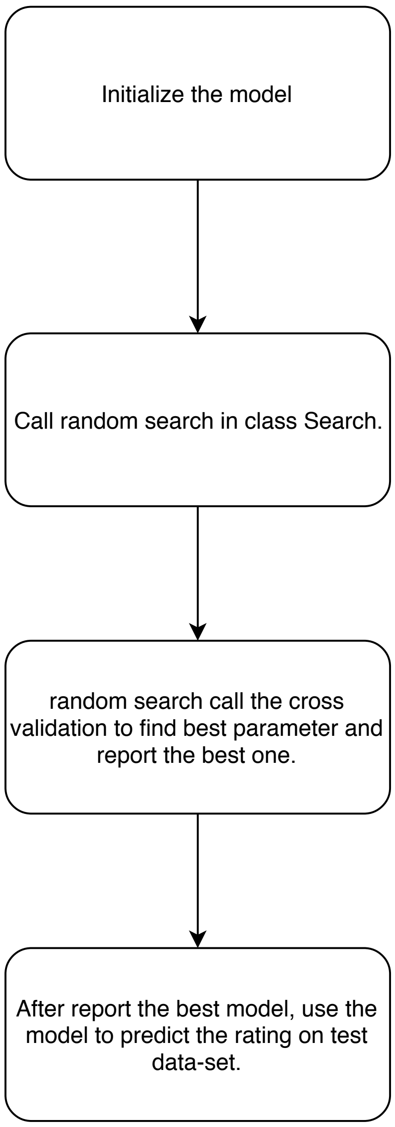

Initialize the model
Call random search in class Search.
random search call the cross validation to find best parameter and report the best one.
After report the best model, use the model to predict the rating on test data-set.

Figure 2.5: The flow chart for finding the best model using our library.

3 Experiment

3.1 Dataset

We compared our models with the conventional SVD model on five public datasets, MovieLens-100K [9], MovieLens-1M [9],FilmTrust [8], Movietweetings [6], and Book-Crossing [20]. The MovieLens dataset was collected by the GroupLens Research Project at the University of Minnesota. The data was collected through the MovieLens website, the dataset consists ratings from users on movies, each user has rated at least 20 movies, the notation 100K and 1M denote the number of ratings. The FilmTrust dataset was collected through the FilmTrust website, the dataset consists ratings from users on movies, it is the smallest dataset in our experiment. The MovieTweetings dataset is a dataset consisting of ratings on movies that were contained in well-structured tweets on Twitter, the MovieTweetings is still updating from Twitter, and the number of ratings is about 686 thousand when we download it. The Book-Crossing dataset is collected by Cai-Nicolas Ziegler from the Book-Crossing community, the Book-Crossing consists ratings and implicit feedback from users on books, we remove the implicit feedback from the dataset since our experiment is based on explicit feedback (i.e. ratings). Table 3.1 shows a summary of the statistics of these datasets.

TABLE 3.1: Statistic of the experimental datasets

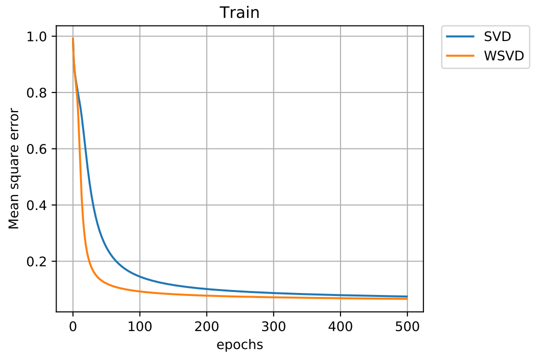
	#users	#items	#ratings	Density	Rating Scale
MovieLens 100K	943	1682	100,000	6.30%	[1, 2, 3, 4, 5]
MovieLens 1M	6040	3706	1,000,209	4.46%	[1, 2, 3, 4, 5]
FilmTrust	1,508	2,071	35,497	1.14%	[0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4]
MovieTweetings	52897	30395	686,954	0.04%	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Book-Crossing	77805	185973	433,671	0.002%	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

3.2 Experiment environment

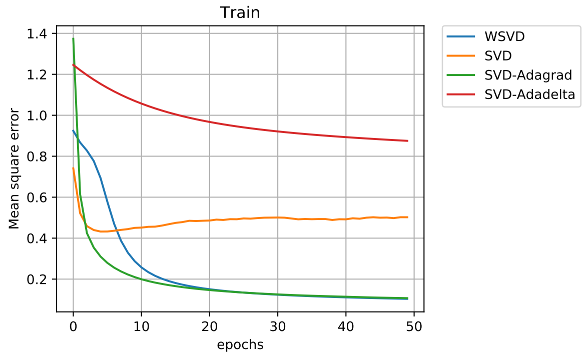
We use the mean-square error (MSE) as our metric to compare the prediction between the SVD model, WSVD model, WSVD model with gradient clipping (the WSVD-C model), and the PSVD model. The datasets are split into training and test sets using a random 80%-20% split. We set the number of factors $k = 35$, fixed the regularization weight for the biases (i.e., λ_b) to 0.02 and use the 5-fold cross-validation with the random search on the training set to find the rest hyper-parameters for every model and every dataset. We implemented all the methods in Python 3.6.2 with Cython [2] which is a C-extension for Python to speed up the computation. All the experiments reported here were performed on a server with the Intel E5-2603 v4 CPU 1.70GHz and 64GB RAM. The OS is Ubuntu Linux 16.04.

3.3 Optimization speed of the WSVD model

In Section 2.2.4, we mentioned that the over-parameterzaion model with simple SGD is faster than two well known acceleration method - Adagrad and Adadelta. To verify this phenomenon exist in the WSVD model, we set every regularization weights of the SVD model and the WSVD model to zero and trained both models on the MovieLens-100K in order to compare the optimization speed between the SVD model and the WSVD model. Figure 3.1(a) shows at the beginning of the training phase, the WSVD model reaches much lower MSE than the SVD model, and since the expressiveness of the SVD model and the WSVD model is identical, at the end of the training, the MSE of the SVD model eventually catch up the MSE of the WSVD model. We also compare the optimization speed between the WSVD model, SVD model, SVD model with Adagrad method and SVD model with Adadelta method. We use the grid search to find the best hyper-parameters for each model on the MovieLens-100K dataset. To be specific we set the regularization to zero and learning rate to $0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5$. Figure 3.1(b)



(a) WSVD *v.s.* SVD



(b) WSVD *v.s.* AdaGrad and AdaDelta over SVD

FIGURE 3.1: Speed compared between SVD model and WSVD model

shows the result, the WSVD model is faster than all the other model, although the MSE of the WSVD model only lower than Adagrad very slightly (about 0.003), the experiment still shows that the WSVD model indeed has a faster optimization speed than the SVD model.

3.4 Overall MSEs result

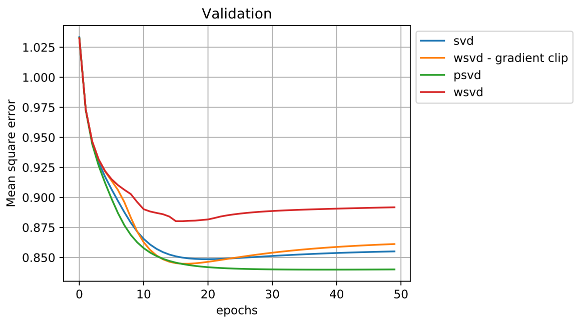
To measure the performance of our models, we apply our models to the test data of each dataset. As explained in Section 3.2, we apply 5-fold cross-validation and random search to find the best hyper-parameters for each model on each dataset to ensure we get the best performance on each model. The result is shown in Table 3.2, the PSVD model performs better than the SVD model in 4 out of the 5 test datasets, and for the WSVD model, when the dataset is not so sparse, in our dataset is MovieLens 100k, MovieLens 1M, and FilmTrust, the WSVD-C model perform much better, as the density go lower as in MovieTweetings and Book-Crossing, the WSVD-C model start to over-fitting and the original WSVD model get better result. This phenomenon is probably due to the gradient vanishing problem we mentioned in Section 2.2.2, as the dataset gets denser, the vanishing problem occurred, the clipping method achieve our desired effect.

Figure 3.2 shows the relationship between the MSE scores on the 5-fold cross-validation and the number of training epochs of the compared methods on the benchmark datasets. As can be seen, the WSVD model tends to stick at a certain point in the MovieLens 100K, MovieLens 1M, and FilmTrust dataset, which are the denser data in our five datasets, and as the data getting sparse, in the MovieTweetings and Book-Crossing, vanishing problem doesn't occur, thus the WSVD model gets a better result.

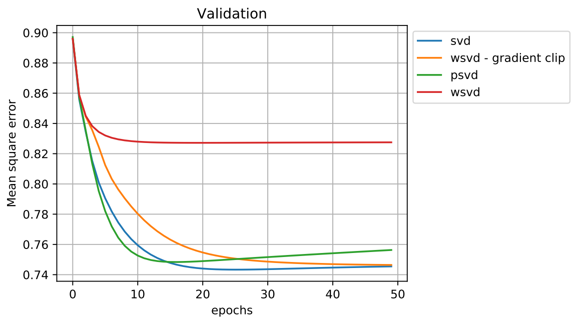
However, although our model wins the SVD model in each dataset, the difference is not significant. We report the p-values of the t-tests of each model in Table 3.2. It can be seen that all models except the Movietweetings are not performing significantly well.

TABLE 3.2: The test MSEs of benchmark datasets

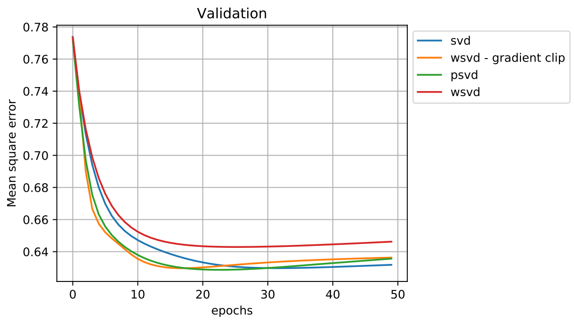
	SVD	WSVD	p-value	WSVD-C	p-value	PSVD	p-value
MovieLens 100K	0.8310	0.8881	<<0.01	0.8279	0.923	**0.8238**	0.679
MovieLens 1M	0.7428	0.8280	<<0.01	0.7356	0.401	**0.7355**	0.439
FilmTrust	0.6230	0.6424	0.264	0.6214	0.917	**0.6164**	0.809
MovieTweetings	2.2290	**2.1493**	<<0.01	2.2556	0.222	2.1724	<<0.01
Book-Crossing	2.6299	**2.6252**	0.85	2.8491	<<0.01	2.8215	<<0.01



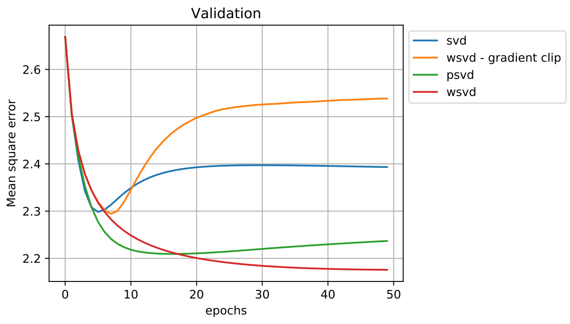
(a) MovieLens-100K



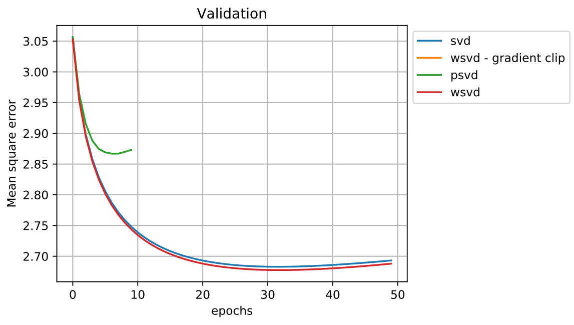
(b) MovieLens-1M



(c) FilmTrust



(d) MovieTweetings



(e) Book-Crossing

FIGURE 3.2: The MSE scores on the 5-fold cross-validation vs training epochs of different methods on the benchmark datasets.

3.5 Overall MSEs result – SVD with comparable number of learnable parameters

Although our proposed models outperform the SVD model most of the time, the improvement might comes from the growth of model parameters. To ensure the improvement of our models is not due to the growth of model size, we set the number of latent factor of the SVD model equal to the PSVD model, which is $35 + C_2^{35} = 630$. We also apply 5-fold cross-validation and random search to find the best hyper-parameters as section 3.2 on the MovieLens-100K, FilmTrust, and Book-Crossing to keep the fairness of the experiment.

The result shows in Table 3.3. We named the size growth SVD model as SVD-Alter in Table 3.3. The PSVD model performs better than SVD-Alter in 2 out of 3 test data, means that the non-linearity indeed helps the generalization of the PSVD model. The WSVD model still only get better performance when the dataset is sparse as we mention in 3.4. The WSVD-C model doesn't outperform the SVD-Alter in any dataset, but the gap of MSE between them are quite small, in MovieLens 100K and FilmTrust the SVD-Alter only outperform less than 0.01.

TABLE 3.3: The test MSEs of SVD-Alter and our methods

	SVD-Alter	WSVD	WSVD-C	PSVD
MovieLens 100K	0.8261	0.8881	0.8279	**0.8238**
FilmTrust	0.6203	0.6424	0.6214	**0.6164**
Book-Crossing	2.7012	**2.6252**	2.8491	2.8215

3.6 Training time

This section shows the training time for our compared methods. We used FilmTrust, MovieLens-100K, and MovieLens-1M as the benchmark datasets. The experiments environment is the same as we mentioned in Section 3.2. Table 3.4 list the training time of an epoch for the compared method. The training time of the WSVD and WSVD-C model is almost the same as the SVD model and about 66% faster than the SVD-Alter. The PSVD model costs about 135% times of the SVD-Alter and 220% of the SVD model. Although in Section 3.5, the SVD-Alter outperforms the WSVD-C model a little bit, the WSVD-C model has apparently better time and memory efficiency. In a large-scale recommender system with a huge number of users and items, the WSVD-C model might be a more reasonable choice than the SVD-Alter.

Table 3.4: Average training time(in seconds) of one epoch

	SVD	WSVD	WSVD-C	PSVD	SVD-Alter
FilmTrust	0.55	0.56	0.56	1.22	0.90
MovieLens 100K	1.47	1.47	1.47	3.53	2.18
MovieLens 1M	14.5	14.6	14.6	33.2	21.0

3.7 Non-active users and long-tailed items

In the SVD model, non-active users and long-tailed items have less frequent updating of latent factors, and the rating predictions of these users and items are usually poor. In the WSVD model and WSVD-C model, we assume that since the weight w is updated in each round of parameter updates, non-active users and long-tailed items may have better prediction results.

In order to verify our hypothesis, we divide users and items into ten groups according to their activity and popularity. Specifically, users are grouped according to the number of times they gave rating, and items are grouped according to the number of times they receive the rating. We use heat maps to show the difference in rating predictions for each group of users and each group of items between the SVD model and the WSVD-C model on dataset MovieLens-100K, and the SVD model and the WSVD model on dataset Book-Crossing. The numbers in the figure show how much the WSVD model or WSVD-C outperforms the SVD model, the negative numbers indicate that SVD outperforms the WSVD or WSVD-C model.

Figure 3.3 shows the result on MovieLens-100K, not as we expected, the performance of the WSVD-C model on the non-active users and long-tailed items does not better than the SVD model. Figure 3.4 shows the result on Book-Crossing, still, the performance of the WSVD model on the non-active users and long-tailed items does not better than the SVD model. It's seems that the WSVD model and WSVD-C model doesn't help the non-active user or long-tailed items, but sacrifices them to get the best performance on active user and popular item.

Although the overall effect of the WSVD model is better than the traditional SVD model. However, if WSVD is to sacrifice less active items and users for the overall effect, it may cause the recommendation system to only tend to recommend popular items. To solve this problem, we have another new model GWSVD model.

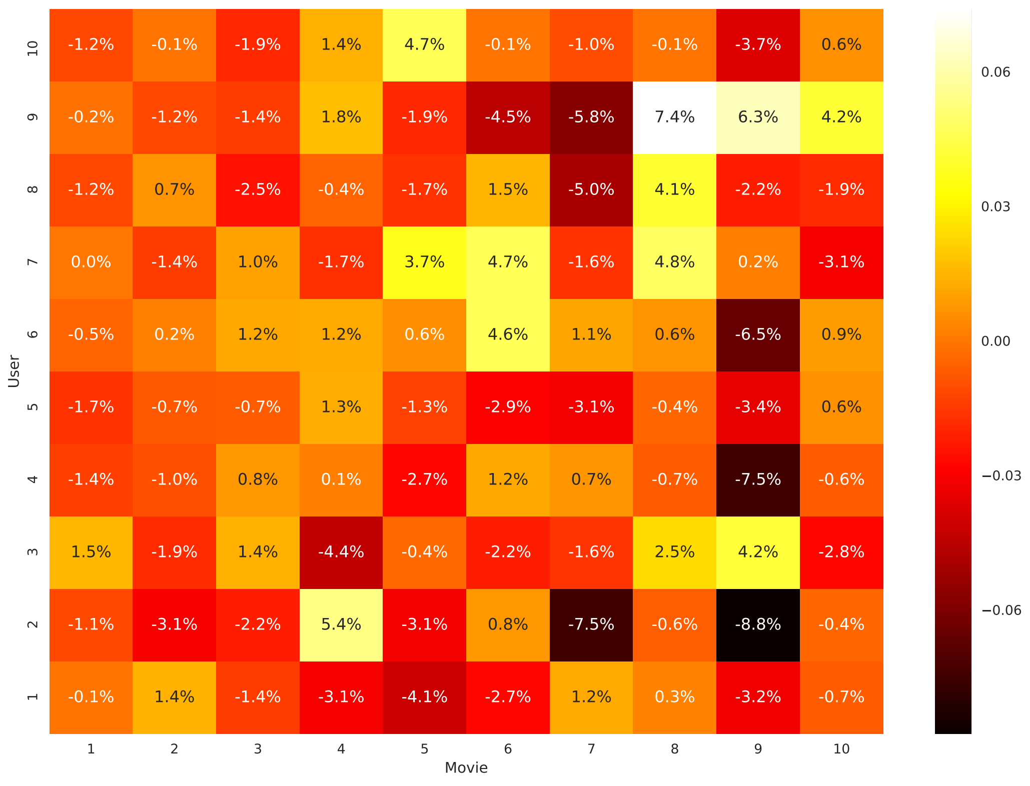


FIGURE 3.3: The heat map shows the performance of the WSVD-C model over the SVD model on MovieLens-100K dataset in percentage. The brighter the color, the better the WSVD-C model is than the SVD model. The result shows that WSVD-C model does not get better performance than SVD model on non-active user and long items.

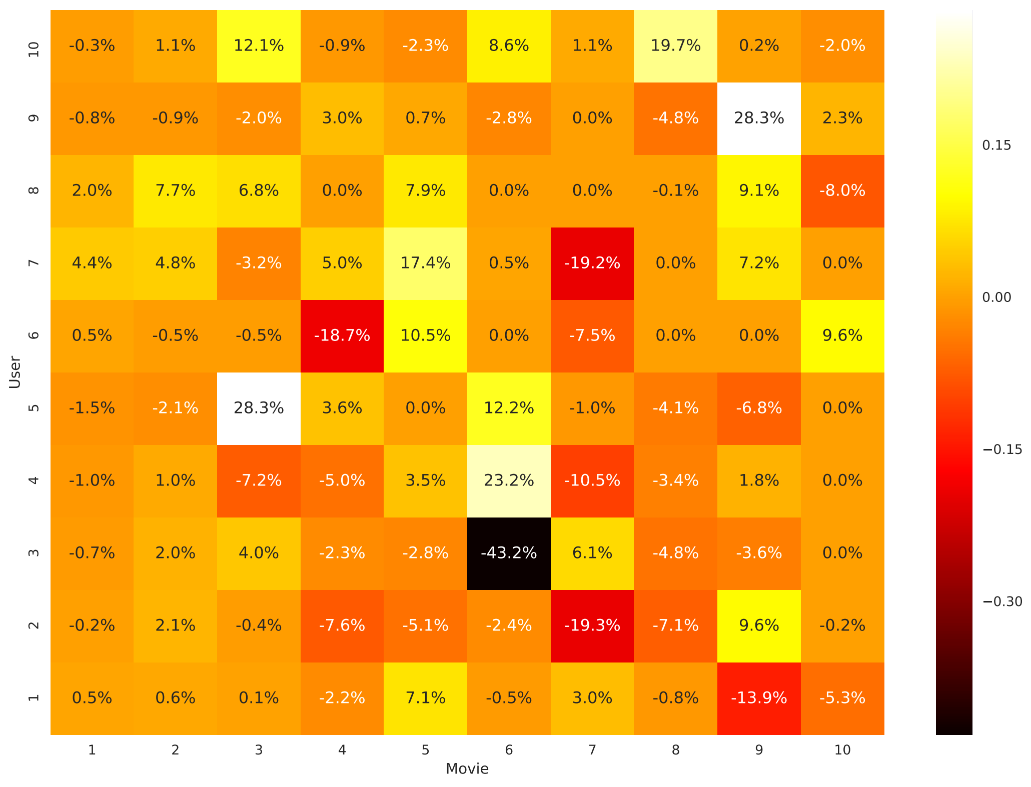


FIGURE 3.4: The heat map shows the performance of the WSVD model over the SVD model on Book-crossing dataset in percentage. The brighter the color, the better the WSVD model is than the SVD model. The result shows that WSVD model does not get better performance than SVD model on non-active user and long items.

3.8 Grouped weighted latent factor model

The non-active problem has been a big challenge in the recommendation system, several models are proposed in order to solve the non-active problem. Most of these models need additional information, an alternative approach is the local model. The local model create additional model for the non-active users and items and try to get the best result on them, this approach may prevent the non-active users and items being polluted by the active users and popular items, since the active users and popular items might dominate the value of MSE, in order to get the lowest value of MSE, the global model might sacrifice the prediction of non-active users and items.

In the previous section, we assume the WSVD model might get better performance on non-active users and items, but the result shows the WSVD model does not significantly outperform the SVD model, sometimes the WSVD model even gets worse on non-active users and items. From Figure 3.3 and Figure 3.4, we can observe that WSVD and WSVD-C model have significantly better performance on the active users and popular items, this means that the weight w in the WSVD model might not helping the non-active users or items but sacrifice them.

Inspired by the local model, we split the items into two groups, the non-active items and the others, and apply different weight w on them. We call this model the GWSVD model. We hope that by splitting the weight of the WSVD model, the non-active items won't be polluted by popular items. Equation 3.1 shows the GWSVD model, C denotes the non-active item set.

$$\hat{r}_{ij} = \begin{cases} p_i \cdot (w_1 \odot q_j^T) + \mu + b_i^{(U)} + b_j^{(I)} & : q_j \in C \\ p_i \cdot (w_2 \odot q_j^T) + \mu + b_i^{(U)} + b_j^{(I)} & : q_j \notin C \end{cases} \tag{3.1}$$

The learning algorithm is same as the WSVD model, if the dataset is not sparse enough, the clipping method is also applied.

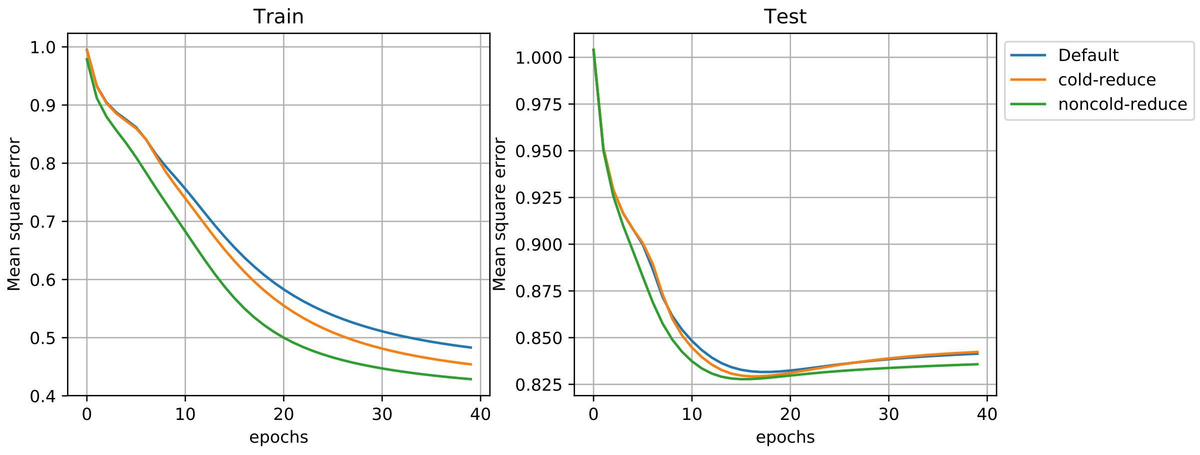


FIGURE 3.5: The MSE scores of the GWSVD model with different hyper-parameters, Default denote the hyper-parameters is same as WSVD, cold-reduce denote the hyper-parameters of non-active weight w is divide by 10, and noncold-reduce is the opposite.

We use the MovieLens-100K dataset to measure the performance, the items receive less than 10 ratings are consider as the non-active items. The hyper-parameters of GWSVD model are inherit from WSVD-C model for MovieLens-100K and hand tune them to find the best performance. In the process of tuning, we found that reduce the hyper-parameters of active-set weight w can reduce the MSE significantly. Figure 3.5 shows the MSE of the GWSVD model with different hyper-parameters, when the hyper-parameters of active-set weight w is divided by 10, we get the lowest MSE score.

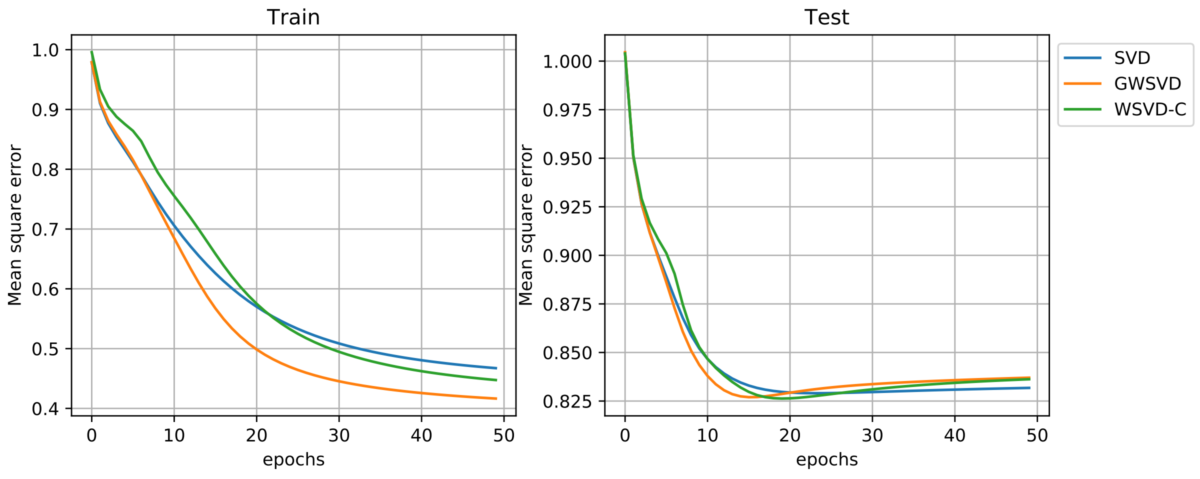


FIGURE 3.6: The MSE scores on the test data vs training epochs of SVD, WSVD, and GWSVD on the MovieLens-100K dataset

	Non-active items	Active items
WSVD	1.2238	**0.8225**
GWSVD	**1.1884**	0.8246

Table 3.5: MSE of non-active item and active items

Since the MovieLens-100K dataset is a dense dataset, the GWSVD model also applied the clipping method while updating the parameters. Figure 3.6 shows the MSE curve of the SVD model, WSVD-C model, and GWSVD model. The GWSVD model has similar performance with the WSVD model on testing data, splitting the weight w doesn't damage the WSVD model. Table 3.5 shows the MSE of the non-active items and the active items. The GWSVD model get about 0.035 improvement on the non-active item compare with the WSVD model, on the contrary, the GWSVD model lose about 0.002 on the active dataset. This suggests that split the weight of the WSVD model might help the non-active items by slightly damage the popular items. However, because we have not done more experiments to prove our concepts, we are hesitant to draw any strong conclusions.

4 Related work

In this chapter, we will introduce two main strategies in the recommendation system, content filtering and collaborative filtering. After introducing content filtering and collaborative filtering, we will introduce three other models related to the SVD model, namely SVD++, Factorization Machines and Neural Collaborative Filtering. Finally, we will introduce three ways to optimize hyper-parameters: grid search, random search, and Bayesian optimization.

4.1 Content filtering

Most of the time, when it comes to recommend items to a person, we pick the items based on his personality and appearance. For instance, as a clerk in video rental store, you might recommend *Toy Story* to a young family, not a mid-age man who came alone. If a young couple continues to rent romance movie, you might want to recommend *La La Land* as demonstrated in Figure 4.1, not *The Shawshank Redemption.* The strategy of using external information of users and items to recommend is known as *content filtering.*

In the content-based recommendation system, the system usually required some information extraction or information retrieval technique in order to profile the users and items properly. After we get enough information, we can recommend the item which is similar to the items previously rated by the specific user, or evaluate the correlation between user and item and recommend the most relative item to the user.

There are two main advantages of using the content-based recommendation system. First, the content filtering is usually explainable, we can explain the recommendation strategy by listing the content features of the item in the list of recommendations. Second, new items coming won't bring the cold-start problem. The content-based system compares the items with the extracted feature, this feature helps a lot when we need to promote new products to users.

Nevertheless, there are also some disadvantages. First, the information extraction needs a lot of domain knowledge. For instance, for movie recommendations, the systems need to know the genre, the actors and the director of the movies. Building a proper information extractor may cost more time than building recommendation model. Second, it's domain specific. As we mentioned, information extraction requires domain knowledge, therefore it's hard to build a domain-free content-based recommendation system since different domain extract different type of information. Third, the privacy issue. In modern days, privacy concern rises making it hard to collect the information from users, this brings out the cold-start problem of the new users since the only reliable information of the user is the past record of the user.

4.2 Collaborative filtering

In the previous section, we mentioned that the content filtering can't work without external information. To overcome this problem, an alternative approach which only relies on

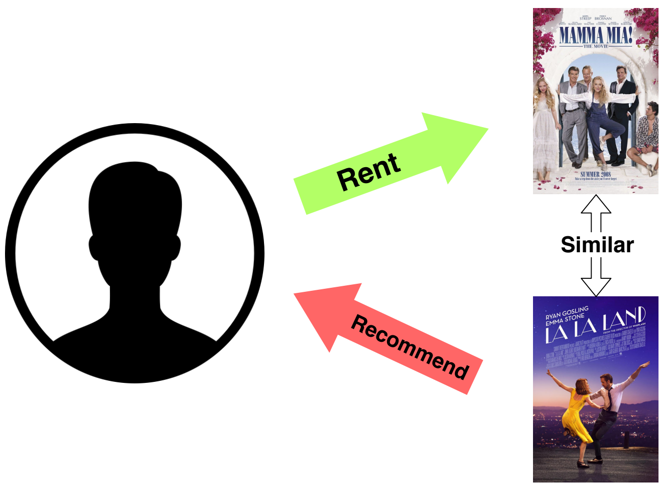


FIGURE 4.1: Content filtering

past user behavior is known as *collaborative filtering*. There are two main approaches to the collaborative filtering: the memory-based approach and the model-based approach.

4.2.1 Memory-based approach

The memory-based approach computes the similarity between users or items, and use it to make recommendations. There are the user-based and item-based approach in the memory-based approach. Figure 4.2 shows the spirit of the memory-based approach. User A and user B both rent the same movies, these two users may be seen as similar users in the system, as a result, the system would recommend the movies which were rented by user B to user A.

To be more specific, in the problem of ratings predicting, the value of ratings user u gives to item i is evaluated as an aggregation of some similar users' rating of the item i in the user-based approach. And for item-based approach, the value of ratings user u gives to item i is evaluated as an aggregation of some similar items' rating from the user u. The memory-based approach often doesn't need to train, we only need to define the aggregation function and the similarity measure function in advance.

Memory-based approach is used a lot in the E-commerce area. For example, the item-to-item collaborative filtering is used in Amazon.com [12]. They use the cosine similarity to measure the distance between the items and recommend the similar item to the customer in the item page of website.

4.2.2 Model-based approach

Model-based approach develop models using different data mining, machine learning algorithms to predict users' rating, our algorithms are based on latent factor model which is one of the model-based approaches. The two models we propose are variants of the SVD model in the Model-based approach. There are also many other models that are variants

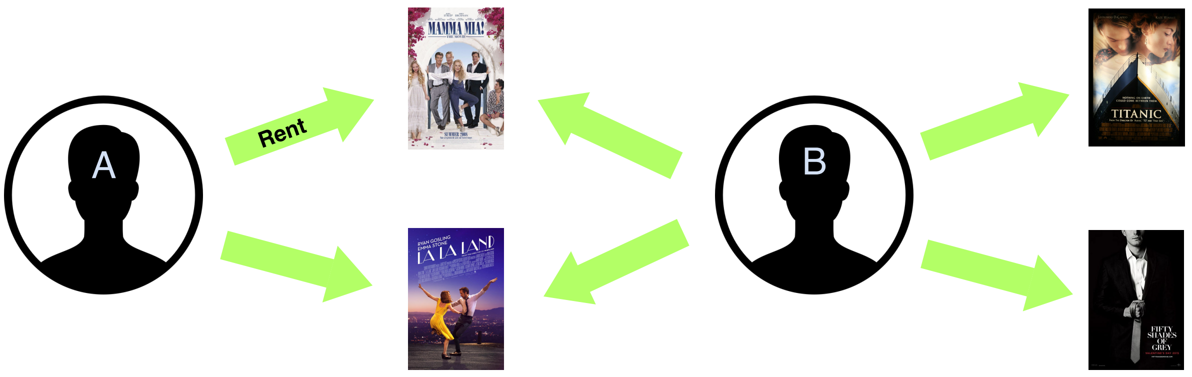


FIGURE 4.2: User-based collaborative filtering

of the SVD model, such as SVD++ models [11], NMF models [14]. The SVD++ model introduces implicit feedback on the basis of the SVD model, and uses the user's history browsing data, history rating data, historical browsing data of the movie, history rating data of the movie, and the like as new parameters. The NMF model is basically the same as the SVD model, but the NMF model assumes that the decomposed matrix does not contain negative values.

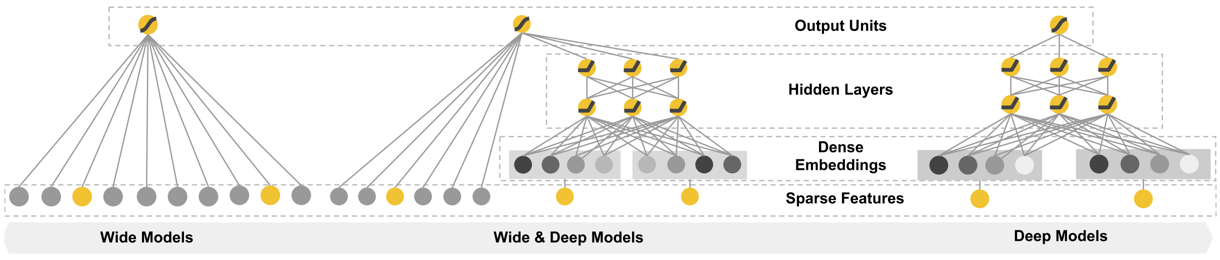


FIGURE 4.3: The wide & deep model

In recent years, deep neural networks have been highly successful in computer vision, natural language processing, and speech recognition, this has brought a huge wave of research of deep neural networks. Of course, deep neural networks have also applied to latent factor model. In [10], beside the weighted sum of latent factors, the neural network is also used to capture the interaction between latent factors. The PSVD model has very similar idea, we also keep the linear part in the model, this is inspired by [5], they show that jointly trained wide linear models and deep neural networks combine the benefits of memorization and generalization for recommender systems. Figure 4.3 shows a comparison of a wide model (logistic regression with sparse features and transformations), a deep model (feed-forward neural network with an embedding layer and several hidden layers), and a Wide and Deep model (joint training of both).

4.3 SVD++

The SVD model was discovered in the Netflix Prize in 2006 to be effectively applied on the recommendation issue. After this many studies have come up with SVD variants. The SVD++ [11]model is one of them. The difference between the SVD++ model and the SVD model is that the SVD++ model also adds implicit feed back data into the model. For example, giving a score is a explicit feed back because we can clearly know if people like or dislike the items. Clicking is a implicit feed back because we have no way of knowing if people like or dislike the items.

The equation of the SVD++ model is as following:

$$\hat{r}_{ui} = b_{ui} + q_i^T(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j) \tag{4.1}$$

where b_{ui} denotes the bias term, q_i denotes the latent factor of item i, p_u denotes the latent factor of user u, $N(u)$ denotes the set of the items for which user u expressed an implicit preference. The vector $|N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$ represents the sum of latent factors of item which user u expressed an implicit preference, and the users is modeled as $p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$.

Implicit feedback can significantly increase the predictive power of the model. Although the importance of explicit feedback is still greater than the implicit feedback, implicit feedback is often more common than explicit feedback. Adding implicit feedback to the model will have Significant help.

4.4 Factorization Machines

Over the last few years, Factorization Machines [15] is a very popular method in the recommendation competition. Factorization Machines aims to deal with the task under very sparse data where SVMs fail. It use the matrix factorization method to reduce the number of parameters.

In the recommendation task, the features of data are often very sparse. The common function with pairwise training data is as follow:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j \tag{4.2}$$

Because the data is very sparse, it's hard to learn a proper w_{ij} as most of the $x_i x_j$ are zeros. To solve this problem, the model equation for a factorization machine of degree $d = 2$ is defined as:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{f=1}^{k} (v_{i,f} v_{j,f}) x_i x_j \tag{4.3}$$

A row v_i describes the i-th variable with k factors. The w_{ij} in Equation 4.2 is decompose into $v_i \cdot v_j$, which means that the weight of pairwise features is not independent. This feature solves the data sparsity problem in SVMs.

Although Factorization Machines appears to be a content filtering method, the SVD model is actually a subset of the Factorization Machines model. Assuming that we only have user and item number data, we can turn users and items into one hot encoding. When we fed the data into Factorization Machines, the w_i is equivalent to the bias of the SVD model, and the $(v_{i,f} v_{j,f})$ is equivalent to the latent factors.

4.5 Neural Collaborative Filtering

In recent years, deep learning has made great progress in computer vision, speech recognition, and natural language processing. This has caused an upsurge of research in deep learning. In the research of the recommendation system, deep learning is often applied to content filtering methods, and it is used to find the complex relationship of features. Compared with content filtering methods, deep learning is less applied to collaborative filtering.

Neural Collaborative Filtering replaces the inner-product in SVD with neural architecture. Although this framework is constructed from neural networks, in fact, this framework can also express SVD model. In order to add nonlinear interactions of latent factors to the model, Neural Collaborative Filtering also uses a multi-layer perceptron to learn user interaction with items. Finally, Neural Collaborative Filtering combines the linear SVD prediction results with nonlinear multi-layer perceptron predictions to obtain the final prediction. The architecture of the Neural Collaborative Filtering is shown in Figure 4.4.

4.6 Hyper-parameter optimization

In machine learning, hyper-parameter optimization or tuning is very important for the model selection. To fairly compare the model, we need to compare the best performance of each model on the same dataset. Since the hyper-parameter has to be set before the learning process of the model, hyper-parameter optimization is the key to find the best performance of the same model on the different dataset.

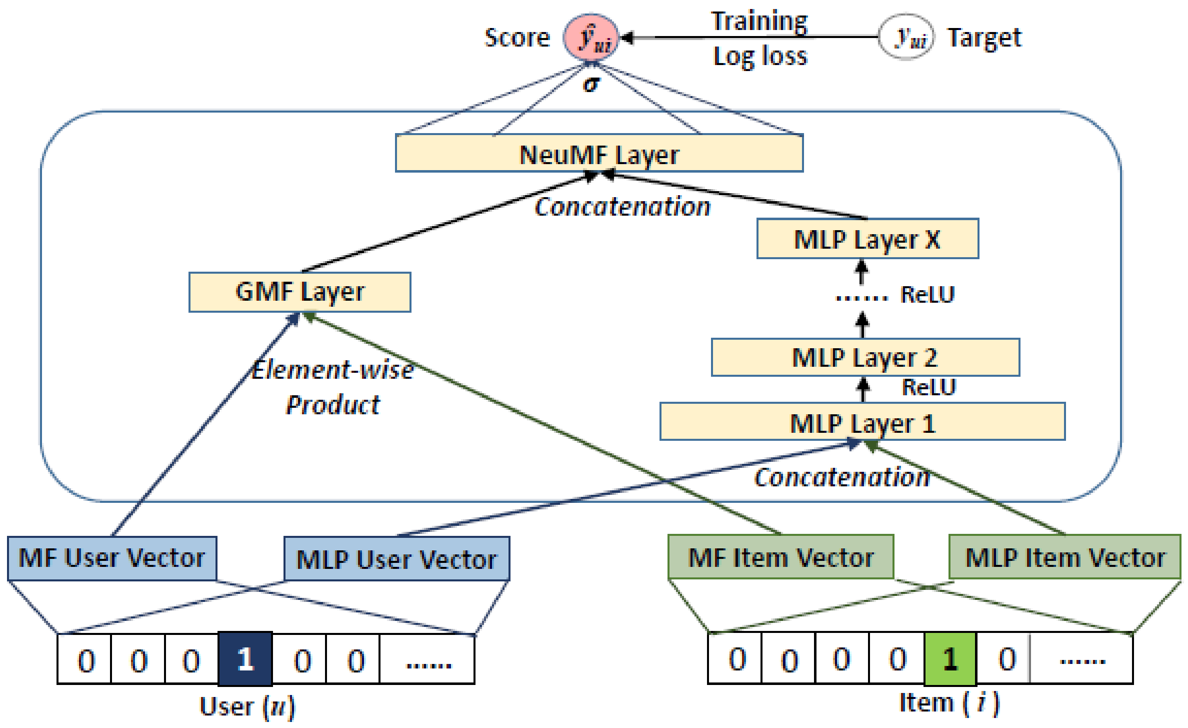


FIGURE 4.4: Neural Collaborative Filtering

4.6.1 Grid search

The most traditional way of tuning hyper-parameter is the grid search. The grid search is an exhaustive searching, we simply define a hyper-parameter space and the grid search traverse over a manually specified subset of the space to find the best hyper-parameter. The grid search needs a metric to measure the performance, it's typically measured by cross-validation on the training set or evaluation on a held-out validation set. Several machine learning software packages have include the grid search such as libsvm [4] and scikits.learn[1]. There are many benefits to use the grid search: grid search is easy to implement and apply parallelization; grid search usually finds better hyper-parameter than manual search in the same amount of time; grid search is reliable in low dimensional spaces.

[1]scikit-learn: http://scikit-learn.org/stable/

4.6.2 Random search

There is another exhaustive searching approach called random search. The random search is very similar to the grid search, the only difference between them is instead of traverse over a manually specified subset, the random search select values from the space randomly. In [3], they shows that random search is more efficient than grid search. Figure 4.5 shows that although the grid search covers the hyper-parameter space evenly, the projections onto either the x-axis or y-axis subspace produces an inefficient coverage of the subspace. In contrast, the random search is less evenly cover the original space, but more evenly cover the sub-spaces. The random search is also easy to implement and apply parallelization since the random search and the grid search is very similar.

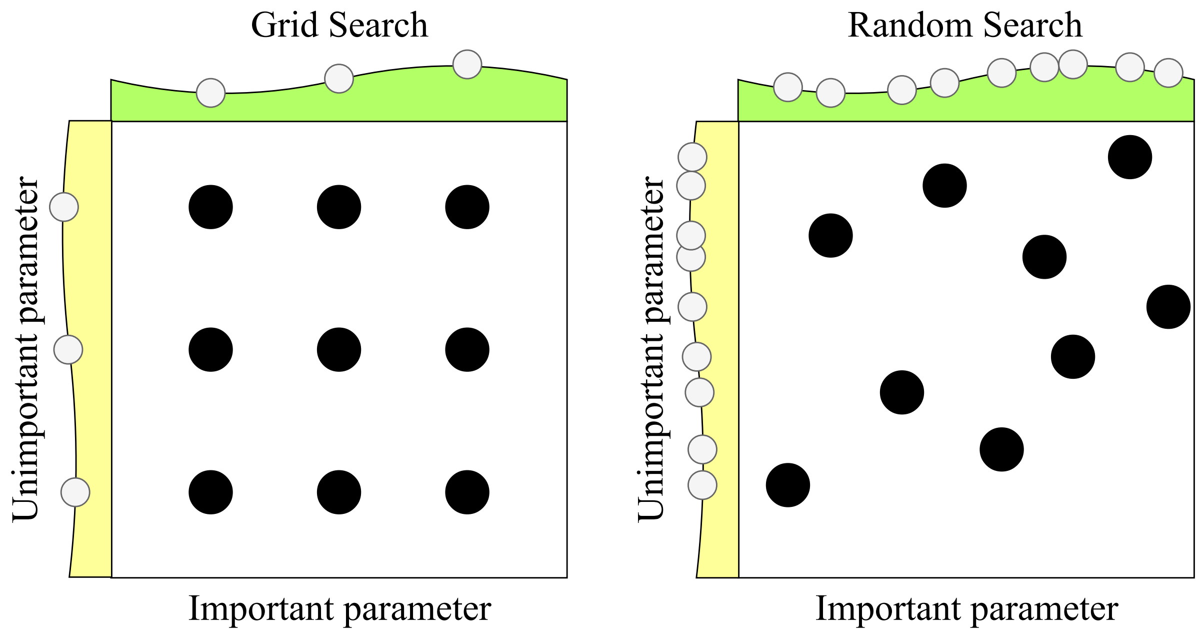


FIGURE 4.5: Grid search v.s. Random search

4.6.3 Bayesian optimization

In addition to the exhaustive searching approach, there is also Bayesian optimization [16], which is a sequential design strategy for global optimization of black-box functions that doesn't require derivatives. Bayesian optimization is not only used for hyper-parameter optimization but also model learning algorithm. There are two major choice to be made when performing Bayesian optimization. First, we need to select a prior over function that express assumptions about the function. Second, we must choose *anacquisition* function to determine the next point to evaluate. Figure 4.6 shows the flow chart of Bayesian optimization.

Bayesian optimization can find the minimum of difficult non-convex functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try. If the function we want to optimize is very hard to evaluate, Bayesian optimization would be a good choice to try. For example, in today's deep learning model, the depth is often more than hundred layers, Bayesian optimization will be a good way to find hyper-parameters. On the contrary, because of the small amount of computation, there are no significant benefits to use Bayesian optimization for our WSVD model and PSVD models.

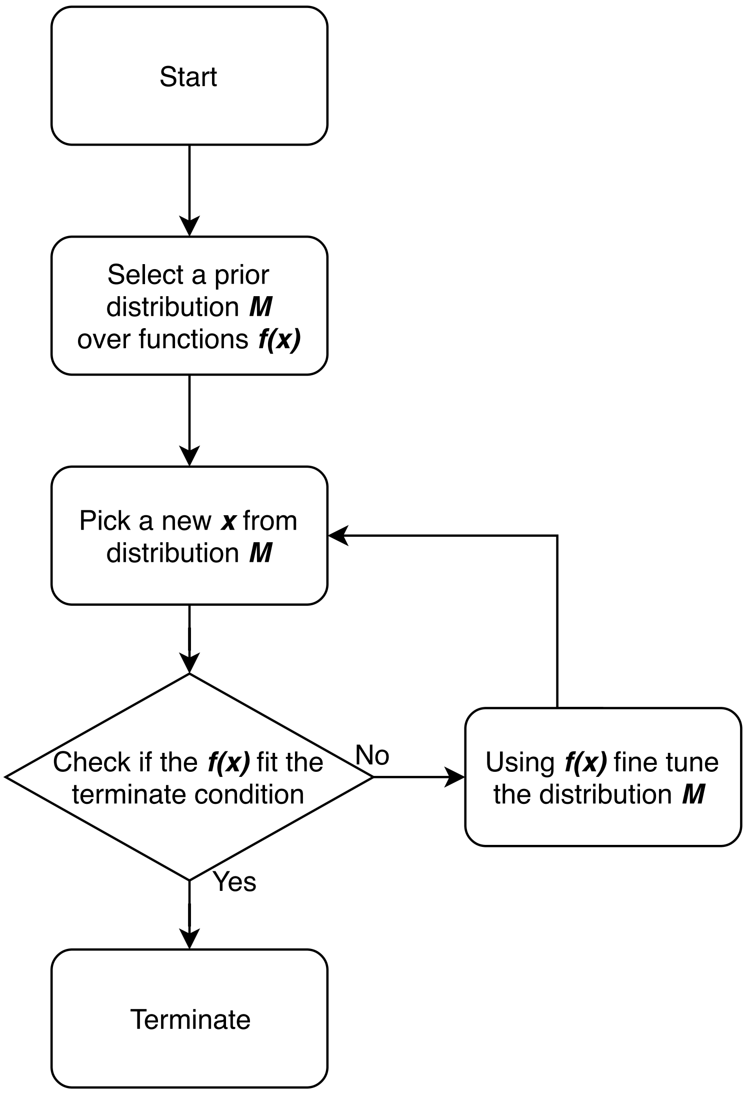

Start
Select a prior distribution ***M*** over functions ***f(x)***
Pick a new ***x*** from distribution ***M***
Check if the ***f(x)*** fit the terminate condition
No
Using ***f(x)*** fine tune the distribution ***M***
Yes
Terminate

FIGURE 4.6: Bayesian optimization flow chart

5 Conclusion and future work

In this thesis, we propose two variants of the SVD model: the WSVD model, and the PSVD model. We compared our models with the conventional SVD model on five different public datasets. The results show that our proposed models defeat the SVD model in the different scenario. The PSVD model outperform the SVD model in 4 out of 5 datasets, and the WSVD model outperforms all datasets using different optimization strategy on the different dataset.

From the perspective of model structure, it's easy to believe that the superiority of the WSVD and the PSVD model is because of the number of parameters is larger than the SVD model. However, the expressiveness of the WSVD model and the SVD model is actually identical. We also compare our model with the alternative SVD model which set the number of parameters equal to the PSVD model since the PSVD model has the largest number of parameters in our compared models. The results show that our models still get better performance. Therefore, the PSVD model and the WSVD model performs better is not because of the better expressiveness of the model, but probably because of the difference between the interaction of latent factors and better and faster optimization.

However, although our model wins the SVD model in each dataset, the difference is not significant. In future research, we may want to use other hyper-parameters optimization or other machine learning methods to find better models.

Since our proposed models are all based on the traditional SVD model, for the future work we are interested in applying the weighting technique and non-linearity relationship to other latent factor-based approaches. Currently, we are also designing another model using recurrent neural networks based on the latent factor model since there might exist some sequential relationship between different ratings of the users, this relationship is especially important in the recommender system of e-commerce system since there exists a strong relationship between items.

In addition to our model applied to the recommendation system, our models can also be applied to any problem that needs to fill the sparse matrix. For example, in a social network, social connections of people can be used to calculate latent factors of people, and using latent factors of people, we can find out potential friends in social networks. We can also use our models to find out the correlation between diseases. We can record the number of people who have both diseases at the same time in the matrix. Using our model, we can estimate the position of the disease in the latent factors space and use the distance of the disease in space to infer the relevance of the disease.

References

[1] Arora, S., Cohen, N., and Hazan, E. On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. *ArXiv e-prints* (Feb. 2018).

[2] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D., and Smith, K. Cython: The best of both worlds. *Computing in Science Engineering 13*, 2 (2011), 31 –39.

[3] Bergstra, J., and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res. 13*, 1 (Feb. 2012), 281–305.

[4] Chang, C.-C., and Lin, C.-J. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 27:1–27:27.

[5] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (New York, NY, USA, 2016), DLRS 2016, ACM, pp. 7–10.

[6] Dooms, S., De Pessemier, T., and Martens, L. Movietweetings: a movie rating dataset collected from twitter. In *Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec at RecSys 2013* (2013).

[7] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res. 12* (July 2011), 2121–2159.

[8] Guo, G., Zhang, J., and Yorke-Smith, N. A novel bayesian similarity measure for recommender systems. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)* (2013), pp. 2619–2625.

[9] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems 5*, 4 (2016), 19.

[10] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web* (Republic and Canton of Geneva, Switzerland, 2017), WWW '17, International World Wide Web Conferences Steering Committee, pp. 173–182.

[11] Koren, Y. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2008), KDD '08, ACM, pp. 426–434.

[12] Linden, G., Smith, B., and York, J. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing 7*, 1 (Jan 2003), 76–80.

[13] Lops, P., de Gemmis, M., and Semeraro, G. *Content-based Recommender Systems: State of the Art and Trends.* Springer US, Boston, MA, 2011, pp. 73–105.

[14] Luo, X., Zhou, M., Xia, Y., and Zhu, Q. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics 10*, 2 (May 2014), 1273–1284.

[15] Rendle, S. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (2010), IEEE, pp. 995–1000.

[16] Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (2012), pp. 2951–2959.

[17] Takács, G., Pilászy, I., Németh, B., and Tikk, D. Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM Conference on Recommender Systems* (New York, NY, USA, 2008), RecSys '08, ACM, pp. 267–274.

[18] Wang, H., Wang, N., and Yeung, D.-Y. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2015), KDD '15, ACM, pp. 1235–1244.

[19] Zeiler, M. D. ADADELTA: An Adaptive Learning Rate Method. *ArXiv e-prints* (Dec. 2012).

[20] Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web* (New York, NY, USA, 2005), WWW '05, ACM, pp. 22–32.