UNIVERSITY OF CALIFORNIA

Los Angeles

Pairwise Learning on Implicit Feedback based Recommendation System

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Zicong Feng

2022

ABSTRACT OF THE THESIS

Pairwise Learning on Implicit Feedback
based Recommendation System

by

Zicong Feng
Master of Applied Statistics
University of California, Los Angeles, 2022
Professor Ying Nian Wu, Chair

Recommendation engine is an integral part in digital business nowadays as abundant user interactions could be captured. This paper provides a comprehensive overview on existing approaches to solve implicit feedback recommendation systems. It introduces implicit feedback data and the challenge of solving recommendation problems uniquely for implicit feedback. Theoretical aspects sort out the necessary formulation, statistical learning, sampling and optimization approaches. To make theory into practice, several implementations using different combinations of statistical learning and optimization methods are applied on an real world e-Commerce data to illustrate the performance and drawbacks.

The thesis of Zicong Feng is approved.

Nicolas Christou

Hongquan Xu

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2022

TABLE OF CONTENTS

1 Introduction **1**

1.1 Objective 2

1.2 Challenges 2

2 Background **4**

2.1 Explicit and Implicit Feedback 5

2.2 Challenges 6

3 Methods **8**

3.1 Model Based Collaborative Filtering 8

3.2 Factorization Machine 9

3.3 Bayesian Personalized Ranking 11

3.4 BPR Learning 13

3.5 BPR Pairwise Learning on Factorization Models 13

3.5.1 Matrix Factorization 13

3.5.2 Factorization Machines 14

3.6 Weighted Approximate-Rank pairwise loss 16

3.7 WARP loss on Factorization Models 18

4 Application on Pairwise Learning **19**

4.1 Data 19

4.2 Model 21

4.2.1 BPR Algorithm 21

4.2.2 WARP Loss 23
4.3 Evaluation Metrics 23
4.3.1 AUC 23
4.3.2 Precision@k 25
4.4 Experiment Result 26

5 Discussion 28
5.1 Future Improvement 28
5.2 Summary 29

LIST OF FIGURES

3.1 Input data where each column represent an user (blue), item (orange) and item-user features. It indicates one when a row belongs to specific user and item. The column on the far right is the target. [15] 10

4.1 Transformation from actual feedback count to binary feedback 20

4.2 Distribution of user-item interactions 21

LIST OF TABLES

4.1 Retail Rocket data summary 20

4.2 AUC on testing data / Retail Rockets 26

4.3 Precision@5 on testing data / Retail Rockets 27

CHAPTER 1

Introduction

Recommendation system became popular recently due to the fact that it can help drive organization growth by recommending items that users would potentially like. Recommendation system is ubiquitous and it is embedded into our lives. From buying food, listening to music to online shopping, we would almost rely on some degree of technology from a hidden recommendation system given that there is a massive amount of information on the internet. Therefore, recommendation system plays an important role on driving user engagement. It also helps maintain platforms' ecosystems and a good recommendations could potentially increase web session time and reduce customer churn. As a result, customers would stick to the service if they always find something they like especially by recommendations.

The most common ways to recommend products to someone, naturally thinking, would be showing them the most popular items over a certain time window on the platform. However, this is problematic as each user will be recommended the same set of items. Recommendation system becomes more personalized as users' preferences are unique to each other. A recommendation system needs user and item input, interactions between them to be built. Complex recommendation system will take additional consideration on user features such as locations, age and gender and item features such as item categories and genres.

The goal of this thesis is to walk through and review some standard approaches used in building personalized recommendation systems using e-commerce consumer data. We will show the difference between explicit feedback and implicit feed, as well as the statistical learning techniques when dealing with implicit feedback.

The order of the topics are as follows: Chapter 1 will continue to introduce the objective and challenges on building a recommendation system. Chapter 2 will introduce the basic data structure for the recommendation system, discuss explicit feedback and implicit feedback. Chapter 3 will start introducing main approaches on dealing with implicit feedback. In chapter 4, we will implement the methods from chapter 3 in an e-Commerce data set, introduce evaluation metrics for implicit feedback recommendation and provide comparison and discussion on the statistical learning result. Chapter 5 will discuss some improvements that could be executed in the future and a summary of this thesis.

1.1 Objective

The main objective of a recommendation system varies by different industries. For subscription businesses such as Spotify and Youtube, it is to retain users and make them stay longer to consume more music or videos. Therefore, user engagement is the metrics to track. For an online shopping platform, the system is used to match user preference and drive sales. Sales could then be the final metrics to measure. Depending on the business domain, recommendation systems usually recommend top k items to a user based on a score calculated from some input information such as user and item interactions. Here k is an arbitrary integer less than 20.

1.2 Challenges

There are several challenges that need to be solved in order to build a recommendation system. First is scalability, which is a major challenge when facing a data set with millions of users. General approaches would then need to learn millions of model parameters using optimization methods like Stochastic Gradient Descent. A better computing system could be a solution to deal with scalability, while another approach [14] was proposed to make the

recommendation system more scalable by introducing a density based clustering algorithm.

Secondly, the cold start problem arises naturally when new users or items arrive. Under a cold start situation, most recommendation systems are not able to learn any parameters as historical data is lacking. Furthermore, cold start problems could cause model inaccuracy when it comes to ranking. Take item as an example, it has high probability to rank as irrelevant items due to the lack of interactions, which will then cause no user interactions as the system treats it as an unpopular item. This cycle will make items that have the potential to be a popular item end up being in a low ranking, which could cause revenue loss.

Thirdly, sparsity is considered as one of the challenges in building a recommendation system. Sparsity occurs when users only interact with a very small subset of items compared to the total number of items. This makes it difficult to apply methods such as nearest neighboring methods due to the difficulty of finding overlap items between users to compute similarity. Regularization is one way to control overfitting when dealing with sparse data.

Item distribution skewness could add challenges in the model fitting process. This means a certain number of items occur on almost every user's interaction, which causes a higher probability of being updated during the stochastic gradient descent algorithm. This is even more severe when working with pairwise learning, where a positive and negative sample are sampled simultaneously to learn the correct ranking. Eventually, it will not produce a personalized recommendation.

CHAPTER 2

Background

Recommendation systems generally can be separated into three categories, content-based [12], collaborative filtering and hybrid approach. Content-based approaches rely on user/item features such as age, location and sex and item categories to learn the probability a user would like a specific item or the rating of a user would rate a specific item. One drawback of content-based recommendation systems is the steps taken to form characteristic features on user and item, which requires domain knowledge and significant amounts of engineering. Content-based recommendation systems are less personalized compared to collaborative filters because it mainly relies on identifying what type of products/items you like, which might already be liked by many users.

Collaborative filtering approaches are based solely on past user/item interactions. It can be categorized into two sub categories, memory-based and model-based. Memory-based methods find user or item similarities and rank them based on the scale of similarities. For user-based collaborative filtering, the system looks for the most N similar users relative to the target user and predicts the score of an non-interacted item by taking into account all selected user similarities, target item scores and all items average scores. However, memory-based methods computing are expensive when it comes to large user interaction data therefore, suffer from scaling issues. Model-based methods, on the other hand, are more efficient and scalable when facing large data sets. The main idea is to map a user-item interaction matrix into two lower dimension latent spaces and learn the weights on both matrices recursively to minimize errors between predicted and actual scores.

Hybrid approaches [2] combine both user/item interaction and user/item features if available. These methods could achieve state-of-art performance by combining collaborative filtering and content-based recommendation. There are various ways to build such as deep learning [3] and factorization machine [15].

A recommendation system starts with a set of users U, a set of items I and an interaction set $S \subseteq U \times I$. S can be ratings, number of songs played and indicators indicating whether or not a user interacted with an item. Under this condition, S is an $m \times n$ matrix with m rows where each row represents one particular user and n columns where each column represents an item. Given the fact that the majority of users have only interacted with a small subset of items, S is an extremely sparse matrix with a high percentage of missing values. A recommendation system can also accept additional inputs other than user-item interactions, such as item and user features if supplied. Item features are represented into a $n \times p$ matrix where p is the number of total item-feature weights. Similarly, user features can also be embedded as an $m \times q$, where q represents the number of user features, into the training.

Singular Value Decomposition [8], and its extended application has been utilized to solve the matrix completion problem with regularization [20] in order to prevent overfitting on the training data. Matrix Factorization has been a successful approach in recommendation system research because of its scalability and accuracy in extremely large data sets.

2.1 Explicit and Implicit Feedback

Input data in a recommendation system is separated into two categories, explicit and implicit feedback. Explicit feedback is the feedback directly input from users, such as movie rating and product rating at Amazon. Explicit feedback directly reflects users affinity or frustration on products or movies they have consumed, meaning that it is quantifiable.

Another type of feedback is called implicit feedback. This type of feedback is more

pervasive and passive as it is not generated by user direct actions. User behaviors like click, view, streaming, share, save and purchase are considered as implicit feedback because these feedback would come along with the user when they are browsing websites or using streaming services. While implicit feedback is easy to collect, it comes with noise. [6] Implicit feedback is non-negative feedback as they are not directly expressed by the user. A user did not interact with a certain item does not mean he/she did not like it. It could just mean he/she was not aware of it when browsing the system. Even if a user has an interaction such as purchase, it comes with uncertainty on whether the user were in real favor of the items or it could be a gift for someone else.

2.2 Challenges

While explicit feedback is less noisy and biased, it is expensive and hard to collect as it is input by users. In the real world, the majority of users are not someone who would make frequent ratings on movies. On the contrary, they would either rate a movie when they really like it or really hate it. Therefore, explicit feedback often comes with emotion. Additionally, user personality could affect their rating. Someone with a more generous personality might give movies a higher rating while someone with strict rules might give lower ratings no matter what movies they watched.

Implicit feedback is more noisy, and the steps it takes to learn and evaluate implicit feedback recommendation are different. In an explicit feedback system, mean square error (MSE), root mean squared error (RMSE) or mean absolute error (MAE) are commonly used to evaluate performance of the rating prediction. These metrics evaluate how good the predictions are to the actual user rating by taking the difference between the predicted and actual rating. For implicit feedback, prediction also generates a predicted score for an interaction. However, this predicted score does not mean rating but rather, user preference on the items. [6] In implicit feedback data, missing data is set to zero meaning no clicks or

no views for an item. However, a missing rating cannot be set as zero from explicit feedback data. When evaluating model performance, predicting a zero feedback would yield a good result using MSE/RMSE/MAE whereas it is meaningless for implicit feedback. Therefore, we need to consider ranking related metrics to evaluate model performance. [18]

CHAPTER 3

Methods

While there are popular methods [7, 13] have been developed to generate item prediction with implicit feedback data, many of them were not optimized for ranking.[1, 17] These methods such as SVD and matrix factorization, are built to optimize if an item is interacted by a user or not. The prediction itself quantifies the scale of the confidence of the item selection. However, it is more important to emphasize effort on how to rank a list of items because rankings are related to user preference on certain items. Additionally, we want a system to generate personalized ranking, which means every single user will be recommended a list of items that are unique to their preference.

A popular way to provide personalized ranking [17] items is to use a loss function called Bayesian Personalized Ranking by uniformly sampling positive samples and negative samples and optimizing for the correct ranking. Another pairwise optimization method, WARP loss [19], was developed to make the optimization convert faster by considering a violation situation in which the negative sample score is greater than positive sample utility score.

We will discuss the theoretical part of loss function BPR and WARP, along with two scoring methods, matrix factorization and factorization machines.

3.1 Model Based Collaborative Filtering

Matrix factorization [10] is a common model-based approach for collaborative filtering [9]. It starts with defining a user-item interaction matrix with entries r_{ui}, and aim to find a way

to approximate it using a user latent matrix X_u and a item latent matrix Y_i, where a row of X_u represent a user's hidden attributes toward an item and a row of Y_i represent an item's hidden attributes. X_u and Y_i are also called user and item embedding space. The utility score or interaction score r_{ui} can be represented as the dot product between x_u and y_i.

$$\hat{r}_{ui} = x_u^T y_i = \sum_{k=1}^{f} x_{u,k} y_{i,k} \tag{3.1}$$

where f is the number of hidden factors represented by the lower embedding.

3.2 Factorization Machine

Factorization machine [15, 16] is a generic method to mimic matrix factorization while combining the ability to add feature interaction terms. It takes advantage of the lower embedding techniques on the second or higher order interaction between input variables including user, item and metadata. This attribute makes it capable of modeling data with high sparsity while learning interactions between users and items. Factorization machine requires a format of data input in which each row represents a user, its interacted item, user features and item features if available similar to the setup for support vector machine. To formulate a factorization machine, we recall a first-order linear model and then add a second order interaction term between each of the independent variables. In second order polynomial regression, interaction $w_{j,j'}$ between x_j and $x_{j'}$ form a $p \times p$ weight matrix, storing pairwise interaction between each variable.

$$\hat{f}^{PR}(x) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} w_{j,j'} x_j x_{j'} \tag{3.2}$$

There are three model parameters $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}$ and $\mathbf{W} \in \mathbb{R}^{p \times p}$. Given the number of interaction terms are high when categorical features present, learning the weight in $\mathbf{W}$ can be extremely expensive. However, factorization machine have the ability to learn $\mathbf{W}$ in a feasible way by introducing a low rank embedding on $\mathbf{W}$, where $w_{j,j'} \approx \langle v_j \, , v_{j'} \rangle$, i.e. inner

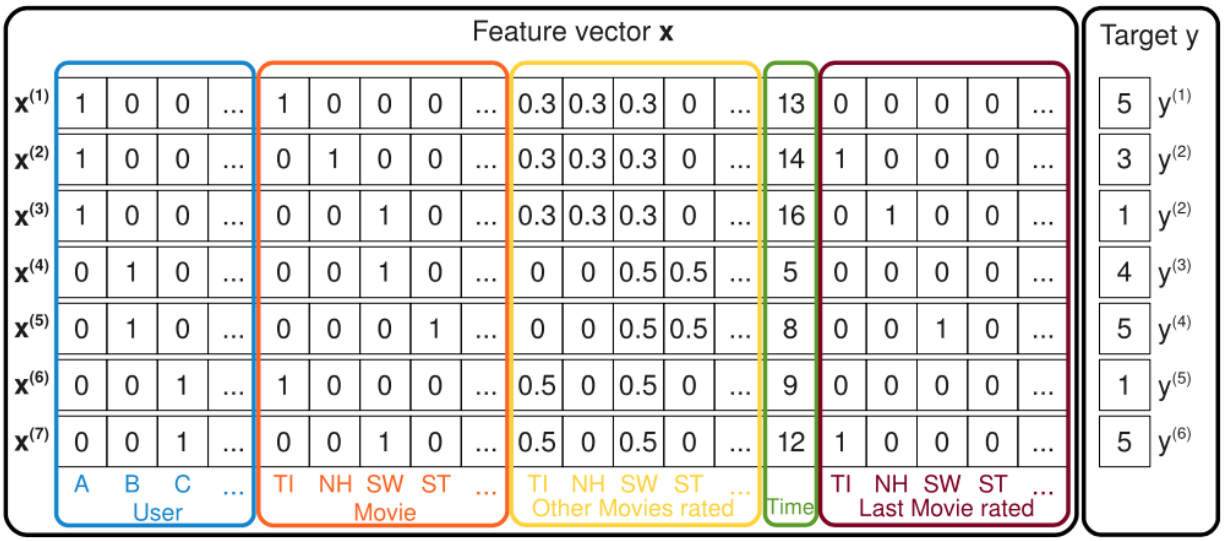

Figure 3.1: Input data where each column represent an user (blue), item (orange) and item-user features. It indicates one when a row belongs to specific user and item. The column on the far right is the target. [15]

product. In matrix form $\mathbf{W} \approx \mathbf{V}\mathbf{V}^T$ where $\mathbf{V} \in R^{p \times k}$ and $k \ll p$. Therefore, we can rewrite (3.2) in the following way:

$$\hat{f}^{PR}(x) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \left(\sum_{f=1}^{k} v_{j,f} v_{j',f} \right) \tag{3.3}$$

The computational cost for estimating the interaction terms using lower rank embedding is $\mathcal{O}\left(\frac{kp^2 - kp}{2}\right)$, which can further be expressed as $\mathcal{O}(kp^2)$. This will greatly increase the computation time of the learning process. However, it can be shown that the interaction terms can be computed in linear time $\mathcal{O}(kp)$ by reformulating the terms suggested by [15]. The main idea starts with subtracting all self-interaction terms and expanding the algebraic term as follow:

$$
\begin{aligned}
\sum_{j=1}^{p}\sum_{j'=j+1}^{p}\langle v_j, v_{j'}\rangle x_j x_{j'} &= \frac{1}{2}\sum_{j=1}^{p}\sum_{j'=1}^{p}\langle v_j, v_{j'}\rangle x_j x_{j'} - \frac{1}{2}\sum_{j=1}^{p}\langle v_j, v_j\rangle x_j x_j \\
&= \frac{1}{2}\left(\sum_{j=1}^{p}\sum_{j'=1}^{p}\sum_{f=1}^{k} v_{j,f} v_{j',f} x_j x_{j'} - \sum_{j=1}^{p}\sum_{f=1}^{k} v_{j,f} v_{j,f} x_j x_j\right) \\
&= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{j=1}^{p} v_{j,f} x_j\right)\left(\sum_{j'=1}^{p} v_{j',f} x_{j'}\right) - \sum_{j=1}^{p} v_{j,f}^2 x_j^2\right) \\
&= \frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{j=1}^{p} v_{j,f} x_j\right)^2 - \sum_{j=1}^{p} v_{j,f}^2 x_j^2\right)
\end{aligned}
\tag{3.4}
$$

Note here equation (3.4) only requires linear time complexity $O(kp)$.

A popular way to learn the model weights in FM models is by stochastic gradient descent for different loss functions. For SGD, the gradient is computed as follow:

$$
\frac{\partial}{\partial\theta}\hat{f}(x) = \begin{cases} 1, & \text{if } \theta = w_0 \\ x_j, & \text{if } \theta = w_j \\ x_j\left(\sum_{i=1}^{p} v_{i,f} x_i\right) - x_j^2 v_{j,f}, & \text{if } \theta = v_{j,f} \end{cases}
\tag{3.5}
$$

Later on, we will apply this gradient to account for pairwise loss.

3.3 Bayesian Personalized Ranking

Bayesian Personalized Ranking is an optimization approach to learn personalized ranking of items uniquely for every user. BPR helps achieve the optimal ranking by sampling positive item and negative item uniformly for every sampled user, and compute their scoring function difference. Once the difference has been computed, stochastic gradient descent is applied to update model parameters depending on which scoring functions are used.

To illustrate the theoretical aspect of BPR, we first define U as a user set, I as an item set and $S \subseteq U \times I$ as implicit feedback of the user-item interaction matrix. Let

$I_u^+ = \{i \in I : (u,i) \in S\}$ and $U_i^+ = \{u \in U : (u,i) \in S\}$ be all the positive items and users with positive items respectively. We also define $i >_u j$ as item i is preferred over item j by user u, which is a main assumption of BPR. In addition, the training set of BPR optimization is $D_s = \{(u,i,j) \mid i \in I_u^+ \wedge j \in I \backslash I_u^+\}$, displaying the negative item i and positive item u.

BPR aims to maximize the posterior probability $p(\Theta \mid i >_u j)$. By Bayes theorem,

$$p(\Theta \mid i >_u j) \propto p(>_u \mid \Theta) p(\Theta) \tag{3.6}$$

where $p(>_u \mid \Theta)$ is the likelihood of a correct ranking given the model parameter and $p(\Theta)$ is the prior of the model parameter. $p(\Theta)$ is initialized by a normal distribution zero mean and variance-covariance matrix $\Sigma_\Theta = \lambda_\Theta I$. Since each user-item pair is independent, likelihood can be written as

$$p(>_u \mid \Theta) = \prod_{u \in U} p(>_u \mid \Theta) = \prod_{(u,i,j) \in S} p(i >_u j \mid \Theta) \tag{3.7}$$

Then the probability $p(i >_u j \mid \Theta)$ is defined by a sigmoid transformation

$$p(i >_u j \mid \Theta) = \sigma(\hat{x}_{uij}(\Theta)) \tag{3.8}$$

where

$$\sigma(\hat{x}_{uij}) = \frac{1}{1 + e^{-\hat{x}_{uij}}} \tag{3.9}$$

During optimization, a logarithmic function is applied to $P(\Theta \mid >_u)$ directly,which makes it easy to separate the computation of the likelihood and prior.

$$\begin{aligned}
\ln p(\Theta \mid >_u) &= \ln p(>_u \mid \Theta) p(\Theta) \\
&= \ln \prod_{(u,i,j) \in D_s} p(i >_u j) p(\Theta) \\
&= \sum_{(u,i,j) \in D_s} \ln p(i >_u j \mid \Theta) + \ln p(\Theta) \\
&= \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{x}_{uij}(\Theta)) + \ln p(\Theta) \\
&= \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{x}_{uij}(\Theta)) + \ln e^{-\frac{\lambda_\theta}{2} \theta^T \theta} \\
&= \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{x}_{uij}(\Theta)) - \frac{\lambda_\theta}{2} ||\theta||^2
\end{aligned} \tag{3.10}$$

3.4 BPR Learning

In the last section, we derive the loss function of BPR and we show how to use BPR to learn model parameters. Note that the loss function becomes differentiable when the sigmoid function is applied with the scoring function.

$$\begin{aligned}
\frac{\partial BPR-OPT}{\partial \Theta} &= \sum_{(u,i,j)\in D_s} \frac{\partial}{\partial \Theta} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \Theta \\
&= \sum_{(u,i,j)\in D_s} \frac{1}{\sigma(\hat{x}_{uij})} \frac{\partial}{\partial \Theta} \sigma(\hat{x}_{uij}) - \lambda_\Theta \Theta \\
&= \sum_{(u,i,j)\in D_s} \frac{1}{\sigma(\hat{x}_{uij})} \sigma^{'}(\hat{x}_{uij}) \frac{\partial}{\partial \Theta} (\hat{x}_{uij}) - \lambda_\Theta \Theta \\
&= \sum_{(u,i,j)\in D_s} \left(\frac{-e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \right) \frac{\partial}{\partial \Theta} (\hat{x}_{uij}) - \lambda_\Theta \Theta
\end{aligned} \tag{3.11}$$

Gradient descent requires summing all the gradients for all samples, which makes the learning algorithm not feasible. Therefore, stochastic gradient descent is instead used for the learning to make the optimization faster, meaning that only $(u, i, j) \in S$ will be used for an update:

$$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \frac{\partial}{\partial \Theta} (\hat{x}_{uij}) + \lambda_\Theta \Theta \right) \tag{3.12}$$

3.5 BPR Pairwise Learning on Factorization Models

3.5.1 Matrix Factorization

In BPR learning, a positive sample and a negative sample are uniformly sampled to learn the ranking. The utility score x_{uij} is calculated by subtracting the utility score of x_{ui} from x_{uj}, which is used for the gradient update. Here, $\hat{x}_{ui}$ can be viewed as the predicted score of user u preference over item i. We will use matrix factorization and factorization machines to represent the utility score, and derive the gradient for each model parameters.

Matrix factorization method has model parameter $\Theta = (X, Y)$. Therefore,

$$\hat{x}_{uij} = \hat{x}_{ui} - \hat{x}_{uj} = \sum_{f=1}^{k} x_{u,f} y_{i,f} - \sum_{f=1}^{k} x_{u,f} y_{j,f} \tag{3.13}$$

Here, the model parameters need to be learned from BPR are $x_{u,f}$, $y_{i,f}$ and $y_{j,f}$:

$$\frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} y_{i,f} - y_{j,f}, & \text{if } \theta = x_{u,f} \\ x_{u,f}, & \text{if } \theta = y_{i,f} \\ -x_{u,f}, & \text{if } \theta = y_{j,f} \end{cases} \tag{3.14}$$

The final gradient update rules for matrix factorization are:

$$x_{u,f} \leftarrow x_{u,f} + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} (y_{i,f} - y_{j,f}) + \lambda_x x_{u,f} \right) \tag{3.15}$$

$$y_{i,f} \leftarrow y_{i,f} + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} (x_{u,f}) + \lambda_{y^+} y_{i,f} \right) \tag{3.16}$$

$$y_{j,f} \leftarrow y_{j,f} + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} (-x_{u,f}) + \lambda_{y^-} y_{j,f} \right) \tag{3.17}$$

3.5.2 Factorization Machines

Compared to matrix factorization, factorization machines are a more generic way to model item and user features and their interactions. Here we present how to learn factorization machine in pairwise setting and the gradient update rules for each model parameters, $\{w_0, \mathbf{w}, \mathbf{V}\}$. $\mathbf{V}$ contain all the the hidden embedding weights for user, item, user features and item features when supplied.

Recall from (3.4), utility score for an item i and user u can be written as follow:

$$\hat{f}(u, i) = w_0 + \sum_{l=1}^{p} w_l x_l^{(i)} + \frac{1}{2} \sum_{f=1}^{d} \left(\left(\sum_{k=1}^{p} v_{k,f} x_k^{(i)} \right)^2 - \sum_{k=1}^{p} v_{k,f}^2 x_k^{(i)2} \right) \tag{3.18}$$

where p is the number of features including user and item and $v_{k,f}$ is the latent embedding weights for the corresponding feature. Similarly, we need to subtract $\hat{f}(u, i)$ with $\hat{f}(u, j)$ to

optimize the parameter in BPR, where $f(u, j)$ is the predicted score for a negatively sampled item during the learning process.

Let's define $g(u, i, j) = \hat{f}(u, i) - \hat{f}(u, j)$, and with some algebraic manipulations:

$$g(u,i,j) = \sum_{l=1}^{p} w_l \left(x_l^{(i)} - x_l^{(j)}\right) + \frac{1}{2}\sum_{f=1}^{d}\left(\left(\sum_{l=1}^{p} v_{l,f}x_l^{(i)}\right) - \left(\sum_{l=1}^{p} v_{l,f}x_l^{(j)}\right) - \sum_{l=1}^{p} v_{l,f}^2 (x_l^{(i)})^2 + \sum_{l=1}^{p} v_{l,f}^2 (x_l^{(j)})^2\right) \quad (3.19)$$

The generic gradients of (3.19) are:

$$\frac{\partial}{\partial\theta}\hat{g}_{u,i,j} = \begin{cases} x_l^{(i)} - x_l^{(j)}, & \text{if } \theta = w_l \\ \sum_{l=1}^{p} v_{l,f}\left(x_l^{(i)}x_k^{(i)} - x_l^{(j)}x_k^{(j)}\right) - v_{k,f}\left((x_k^{(i)})^2 - (x_k^{(j)})^2\right), & \text{if } \theta = v_{k,f} \end{cases} \quad (3.20)$$

where $x_k^{(i)}$ and $x_k^{(j)}$ are feature values k on under item i and j. Given there are $\mathbf{w}, \mathbf{V}$, we are representing these two parts in terms of item weights w_i, item features weights w_{if}, item latent factor weights $v_{i,f}$, user latent factor weights $v_{u,f}$, user features latent weights $v_{uf,f}$ and item features latent weights $v_{if,f}$. w_{if}, $w_{uf,f}$ and $v_{if,f}$ are not available when there are no user and item features. Under this case, the factorization machine will be reduce to general matrix factorization plus bias terms on item i. Let's assume that we have both user and item meta features available, then the corresponding gradients on each parameters based on (3.20) are:

$$\frac{\partial}{\partial\theta}\hat{g}_{u,i,j} = \begin{cases} x_i^{(i)} - x_i^{(j)}, & \text{if } \theta = w_i \\ x_{if}^{(i)} - x_{if}^{(j)}, & \text{if } \theta = w_{if} \\ v_{i,f} - v_{j,f} + \sum_{h=1}^{Q} v_{if_h,f}\left(x_{if_h}^{(i)} - x_{if_h}^{(j)}\right), & \text{if } \theta = v_{u,f} \\ v_{u,f} + \left(\sum_{h=1}^{N} v_{h,f}x_h\right), & \text{if } \theta = v_{i,f} \\ -v_{u,f} - \left(\sum_{h=1}^{N} v_{h,f}x_h\right), & \text{if } \theta = v_{j,f} \\ v_{u,f}\left(x_{if_n}^{(i)} - x_{if_n}^{(j)}\right), & \text{if } \theta = v_{if_n,f} \\ x_{uf_n}\left(v_{i,f} - v_{j,f}\right), & \text{if } \theta = v_{uf_n,f} \end{cases} \tag{3.21}$$

where Q is the number of item features and N is the number of user features.

3.6 Weighted Approximate-Rank pairwise loss

BPR is a powerful method to deal with ranking tasks to achieve personalized recommendation. However, the uniform sampling makes it converge slowly because a large percentage of the sampled pairs (u, i, j) has predicted score $f(u, i) > f(u, j)$, which is an obvious ranking where positive items rank higher than negative items. Since the process is uniform, the probability that it can sample a pair (i, j) where $f(u, i) < f(u, j)$ is small. In order to learn the correct ranking, violation pairs need to be identified so that the gradient can be updated.

We can use WARP [19] a.k.a Weighted Approximate Rank Pairwise Loss to make the learning process more accurate by identifying violating item pairs. Unlike BPR, where gradient update will be performed after (u, i, j) is sample, WARP will only make an update when the sampled when $f(u, j) > f(u, i)$, meaning the predicted score of a negative item is higher than the predicted score of a positive item. This is a violation on the current ranking order.

In addition, the scale of an update is also taken into account where there is an update.

If it takes a small number of steps to find a violated sample j, then the gradient will yield a large update. Similarly, the gradient will yield a small update when it takes a large number of steps to find a violated sample.

The loss function of WARP start with the definition:

$$\sum_{i \in I^+} L\left(rank_i(f_u)\right) \tag{3.22}$$

where I^+ is the set of positive feedback of a user u and

$$rank_i(f_u) = \sum_{i \neq j} I(f(u,j) > f(u,i)) \tag{3.23}$$

Then by applying a trick along with a differentiable log function:

$$\begin{aligned} L(rank_i(f_u)) &= \log(rank_i(f_u)) \frac{rank_i(f_u)}{rank_i(f_u)} \\ &= \log(rank_i(f_u)) \frac{\sum_{j \notin I^+} I\left(f_u(j) \geq f_u(i)\right)}{rank_i(f_u)} \\ &= \sum_{j \notin I^+} \log(rank_i(f_u)) \frac{I\left(f_u(j) \geq f_u(i)\right)}{rank_i(f_u)} \\ &= \sum_{j \notin I^+} \log(rank_i(f_u)) \frac{\mid 1 + f_u(j) - f_u(i) \mid_+}{rank_i(f_u)} \\ &= \sum_{j \notin I^+} \log(rank_i(f_u)) \mid 1 + f_u(j) - f_u(i) \mid_+ \end{aligned} \tag{3.24}$$

where Hinge Loss is applied to transform the indicator component $I(f_u(j) \geq f_u(i))$ into a differentiable function $\mid 1 + f_u(j) - f_u(i) \mid_+$. It is also noted that the probability to draw a negative sample j is $\frac{1}{rank_i(f_u)}$ which accounts for the denominator for the last step.

According to the WARP, computing $rank_i(f_u)$ is a computational expensive task because we have to compute all the predicted scores $f_i(x)$ for all items. An innovative way to estimate the term $rank_i(f_u)$ is using $\lfloor \frac{Y-1}{N} \rfloor$ borrowing from the concept of geometric distribution. It need to sample N trails before the sampling process find a negative item j, which can be treated as geometric distribution:

$$p(N_k = k) = (1-p)^{k-1} p \tag{3.25}$$

where $p = \frac{k}{Y-1}$. The mean of N_k is $\frac{1}{p}$, which is $\frac{Y-1}{K}$.

3.7 WARP loss on Factorization Models

In the previous section, we walked through the definition of WARP Loss in detail. To formulate the learning step in WARP, stochastic gradient descent is again applied. We can rearrange the last step of (3.24),

$$\begin{aligned} \sum_{j \notin I^+} \log(rank_i(f_u)) \mid 1 + f_u(j) - f_u(i) \mid_+ &= \sum_{j \notin I^+} \log(rank_i(f_u))(1 - \hat{x}_{uij}) \\ &= \sum_{j \notin I^+} \log(\lfloor \frac{Y-1}{k} \rfloor)(1 - \hat{x}_{uij}) \end{aligned} \tag{3.26}$$

and then the gradient update will be:

$$\theta \leftarrow \theta + \gamma \left(\log(\lfloor \frac{Y-1}{k} \rfloor) \right) \left(\frac{\partial \hat{x}_{uij}}{\partial \theta} \right) \tag{3.27}$$

where $\frac{\partial \hat{x}_{uij}}{\partial \theta}$ would be (3.14) for matrix factorization and (3.21) for factorization machine.

CHAPTER 4

Application on Pairwise Learning

In the experiment section, we aim to implement the statistical methods and the learning algorithms introduced in Chapter 3. One of the natural challenges that arise during the learning step is the cold-start problem [4], and we will alleviate the impact of cold-start by performing a filter on item interactions on a user base. We will also show the experimental results using BPR, WARP loss with two scoring models, matrix factorization and factorization machine with item features. In order to evaluate how the loss function can help learn the ranking of items, we will also introduce common evaluation metrics to measure model fit on implicit feedback.

4.1 Data

RetailRocket is an e-Commerce online platform bringing personalized shopping experience to web shoppers. The data used in the analysis comes from Kaggle.com. It describes three types of user events, view, add-to-cart and purchase ranging from 2015-05-02 to 2015-09-17. There are a total 1,407,580 unique visitors and 235,061 unique items available for interactions. The following table (Figure 4.1) demonstrates the structure of the event data, where a user could view an item multiple times at different times. In this analysis, we turn the item interacted count into a binary feedback in order to apply the pairwise learning-to-rank methods, which also focus on ranking task rather than predicted preference. For example, if a user views an item 10 times, the binary format will indicate 1 and 0 otherwise. The potential drawbacks is to treat all the interaction items equally where some items might have more views than

the other for a user. To model the actual number of implicit feedback, an alternative least square has been proposed [6]. Graded pairwise learning [11] can also deal with numeric type of feedback by applying ranking methods with extension on BPR.

Users	Items	Interactions	Sparsity
1407580	235061	2145179	99.99%

Table 4.1: Retail Rocket data summary

	item 1	item 2	item 3	item 4
user 1	5	0	12	13
user 2	0	0	7	1
user 3	5	4	0	0
user 4	2	0	0	5

(a) Actual feedback demo

	item 1	item 2	item 3	item 4
user 1	1	0	1	1
user 2	0	0	1	1
user 3	1	1	0	0
user 4	1	0	0	1

(b) Binary feedback

Figure 4.1: Transformation from actual feedback count to binary feedback

In addition to the event log data, additional metadata on items is provided such as item category, item availability and item parent category. Majority of the context on item properties has been hashed, which creates difficulties in translating to meaningful text.

In this experiment, we would want to isolate the cold start problem and focus on visitors with enough feedback for our study. The data set has severe skew on interaction distribution, where 79.5% visitors have only one interaction. (Figure 4.2) As training and testing sets are needed during the training and evaluation process, we filter the data and include only visitors that have 6 or more item interactions during 4.5 months.

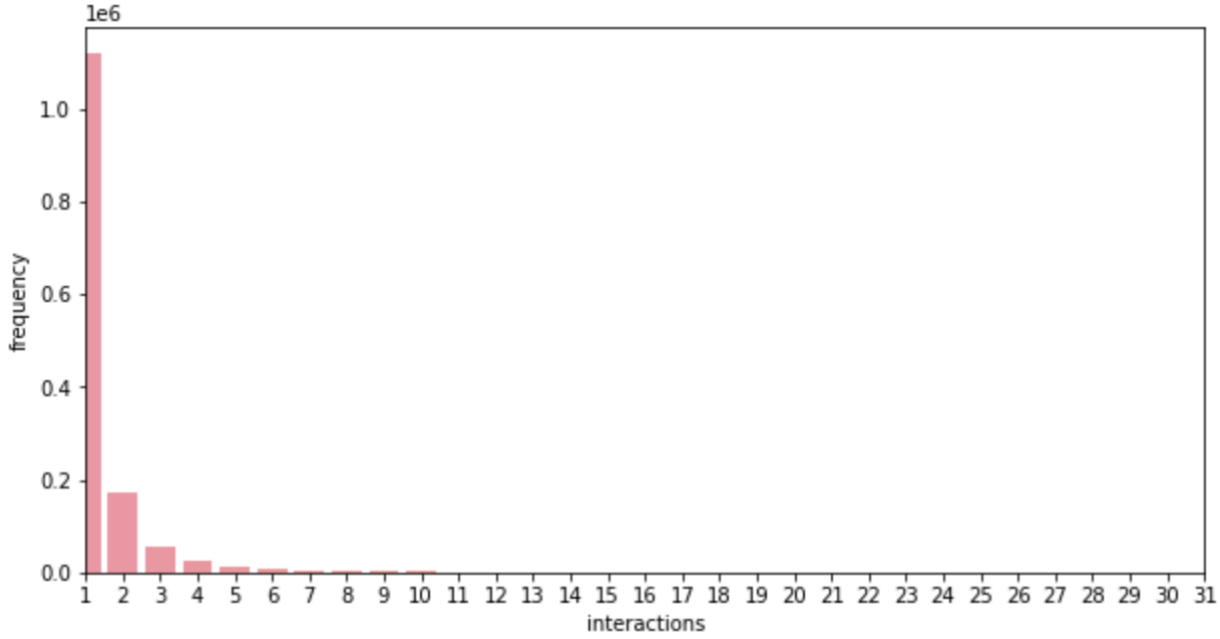

Figure 4.2: Distribution of user-item interactions

4.2 Model

In Chapter 3, we introduce two main optimization algorithms, BPR loss and WARP loss on factorization models. In this chapter, we put together the pseudo code for these two approaches.

4.2.1 BPR Algorithm

In BPR optimization, a user u is sampled from user pools, a positive item i belongs to user u and a negative item (non-interacted) j are sampled. Then the predicted scores for i and j are computed, and the difference between two scores are taken. The way to compute the predicted score depends on the modeling approach. We used matrix factorization and a factorization machine to compute the predicted value. Once the difference in predicted value between i and j has been computed, the model parameters are learned by batch stochastic gradient descent.

Algorithm 1 BPR Optimization

Input: $m \times n$ user item interaction matrix, $D_s = \{(u, i, j) \mid i \in I_u^+ \wedge j \in I \setminus I_u^+\}$

initialize $\hat{\Theta}$

repeat

Draw $(u, i, j) \in D_s$

Compute $\hat{x}_{ui}$, $\hat{x}_{uj}$

Compute $\hat{x}_{uij} = x_{ui} - x_{uj}$

$\Theta \leftarrow \Theta + \gamma \left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \frac{\partial}{\partial \Theta}(\hat{x}_{uij}) + \lambda_\Theta \Theta \right)$

until convergence

Return $\hat{\Theta}$

Depending on the modeling approach, the input data is usually formatted as a $m \times n$ matrix with m users and n items. This is the default input data structure for matrix factorization. When dealing with factorization machines, there are three data structures required. The first data structure is the same as matrix factorization $m \times n$. The second data structure, user metadata, is a structure where each row represents a user along with its attributes such as age, days since last visit, total purchases etc. The third data structure is item metadata where each row represents an item along with its features including item category, item tags etc.

Parameters $\hat{\Theta}$ need to be initialized. Regardless of the scoring function, we will initialize the embedding matrix V_u, V_i, and V_{if}, to normal distribution with mean 0 and standard deviation 1. For the first-order weights, w_i and w_{if}, we initialize them as zero vectors with corresponding length equivalent to the number of items and the number of item features. All embedding matrices are set to 15 hidden factors for this whole analysis.

4.2.2 WARP Loss

WARP loss differs from BPR by selecting a so-called violated negative item where its predicted value is higher than the predicted value of the targeted positive items. In terms of running time, WARP is slower than BPR as it is more difficult to find an violated item when iteration increases since most of the item pairs have been corrected on prior iterations. In practice, instead of going through all the items, a threshold could be applied on deciding when to stop finding the negative sample with violation if a large set of items presented. It would significantly reduce the training time especially on the last several iterations.

Similar to BPR, WARP also optimized ranking by drawing (u, i, j) from the user item interactions. The computing step on x_{ui} and x_{uj} is again dependent on the modeling approaches.

4.3 Evaluation Metrics

4.3.1 AUC

The evaluation metrics for implicit feedback is different from the one used to evaluate a explicit feedback recommendation engine. Rather using RMSE, common metrics on ranking task are AUC, precision@k, recall@k, NDCG(Normalized Discounted Cumulative Gain) AND MRR (Mean Reciprocal Rank) [18]. In this experiment, we are choosing to use AUC and precision@k as our core evaluation on our implementation.

AUC was introduced as an evaluation in [17] to measure performance of a ranking task. AUC is a measurement for overall ranking quality as it involves comparison on all positive and negative item pairs. AUC can be related to the BPR optimization by expressing, for a specific user u, by

Algorithm 2 WARP Optimization

Input: $m \times n$ user-item matrix, $D_s = \{(u, i, j) \mid i \in I_u^+ \wedge j \in I \setminus I_u^+\}$

initialize $\hat{\Theta}$

repeat

  Draw $(u, i, j) \in D_s$

  Compute $\hat{x}_{ui}$

  Set $N = 0$

  repeat

    Pick a negative item $j \in I_u^-$

    Compute $\hat{x}_{uj}$ $N = N + 1$

  until $\hat{x}_{uj} > \hat{x}_{ui} - 1$ or $N > Y - 1$

  if $\hat{x}_{uj} > \hat{x}_{ui} - 1$ **then**

    Make a gradient update:

    $\theta \leftarrow \theta + \gamma \left(\log(\lfloor \frac{Y-1}{k} \rfloor\right) \left(-\frac{\partial \hat{x}_{uij}}{\partial \theta}\right)$

    Enforce $\|V_u\|_2 \leq C$

    Enforce $\|V_i\|_2 \leq C$

    until convergence

  end if

until validation error does not improve

Return $\hat{\Theta}$

$$AUC(u) = \frac{1}{\mid I_u^+ \mid\mid I \setminus I_u^+ \mid} \sum_{i \in \mid I_u^+ \mid} \sum_{j \in \mid I \setminus I_u^+ \mid} I(\hat{x}_{uij} > 0) \tag{4.1}$$

where I is an indicator function:

$$I(\hat{x}_{uij} > 0) = \begin{cases} 1, & \text{if } \hat{x}_{uij} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Intuitively, AUC is averaging on all the combinations of positive item and negative item pairs by counting pairs where positive item predicted score is higher than negative item predicted score given that a positive item is preferred over a negative item if the predicted score is higher. AUC usually ranges between 0.5 and 1, where 0.5 means a poor fit and 1 means perfect fit. An example of using AUC is where a recommended system does not always show the top ranked items because there might be items in the top list that are out of stock. In this situation, we want to move further down to the recommended list where the overall ranking comes into play.

4.3.2 Precision@k

Different from AUC, the concept of precision@k focuses on the k recommended items, where k is usually less than 20. The top k recommended items are decided by the predicted score using the underlying scoring function e.g. matrix factorization. Then precision@k is defined as the number of items that are relevant among the top k recommended items, where relevant items means interacted items. When we evaluate the testing data, relevant items are the interacted items on the testing data. Mathematically, it can be expressed as, for each user:

$$precision@k = \frac{\text{\# of relevant items from top k recommended}}{k} \tag{4.3}$$

Note that the maximum value of precision@k could be less than 1 as it depends on the number of items in the testing data. For example, if a user only has 3 positive items in the

testing data, then the maximum precision is 3/5, i.e. 0.6.

4.4 Experiment Result

Due to limited computing power, computing factorization machines (3.21) with WARP loss (3.27) using gradient descent is expensive and slow on local machines. There are $N = 269$ item features after transforming the categories into binary coding where the iteration needs to go through all of the item features.

We are also imposing a limit on searching for a violation in WARP loss $x_{ui} < x_{uj} + 1$. Gradient is updated until it finds such a violation, which could potentially take up the number of times equivalent to the total negative item numbers for a user. Given that the training data contains 66,117 items, the time to find out a violation pair will be prolonged especially for a higher number of iterations.

Epoches	**BPR-MF**	**BPR-FM**	**WARP-MF**	**WARP-FM**
2	0.502	0.501	0.502	**0.552**
5	0.502	0.505	0.509	0.598
10	0.498	0.509	0.534	0.638
15	0.497	0.512	0.564	0.664
25	0.499	0.506	**0.596**	**0.808**

Table 4.2: AUC on testing data / Retail Rockets

We run the experiment by iterating the learning process 2, 5, 10, 15 and 25 times for each of the scoring and loss function combinations. In only two iterations, WARP-FM has already outperformed the remaining learning methods by AUC. An interesting result is that BPR does not improve the AUC and precision@k even if the iteration increases. This is expected as uniform sampling usually takes a substantial amount of training time to convert without some additional steps to pick the pair of items for training. However, we see some

prominent progress using the WARP loss implementation, by adding additional item features in particular. As the result shown, AUC reaches 0.81 after 25 iterations using item features embedding on item categories.

Epoches	**BPR-MF**	**BPR-FM**	**WARP-MF**	**WARP-FM**
2	0.0006	0.0004	0.000 19	0.000 22
5	0.000 89	0.0005	0.000 23	0.000 35
10	0.000 38	0.0006	0.000 36	0.000 513
15	0.000 38	0.000 89	0.000 57	0.000 72
25	0.0004	0.0012	0.0009	**0.003**

Table 4.3: Precision@5 on testing data / Retail Rockets

Recommendation system is typically asked for delivery of the top ranking of items. The precision results show that we still need to fine tune our model and continue to optimize the top K recommendations.

CHAPTER 5

Discussion

5.1 Future Improvement

We have showcased how pairwise learning could solve an implicit recommendation task by introducing item pairs, a.k.a a positive and negative item schema. Our result shows that additional features could potentially help improve the model accuracy. However, our model suffers from some bottleneck. The first one is speed. When data becomes large, computing the interaction terms within factorization machines depends largely on the resources given a linear run-time provided from the formula. Indeed, the training data we used only contain less than 5% of users and the majority of users have only interacted with one item. It is likely that the majority of these users are new users while we also need to use more data to justify this hypothesis. That means cold-start can be an issue when new users/new items arrive. [5] can provide various choices to deal with cold-start problems in recommendation.

The next improvement is item features. Given that we do not have user features available, we can exploit more item features in addition to item categories. However, detailed information about items was hashed except for the item category. This makes it challenging to generate meaningful features to interpret its relationship with the item attributes. One of the improvements can be to pick the top ten text features appearing across items and start generating binary features to enrich the item feature space. Otherwise, a wide range of Natural Language Processing techniques could be applied to uncover insights on the text data and then incorporate it into the training data.

In this analysis, we are modeling binary feedback in pairwise learning. However, there are common scenarios where implicit feedback is being represented as numeric value such as the number of views for the same items. If there are two positive interactions for a user, converting the numeric feedback into binary value could downgrade the importance of one positive on another given that one item had more views than the other. Lerche and Jannach [11] derived a potential approach to tackle this concern and it extended the existing BPR framework by changing the scoring function into a weighted function that can take the scale of feedback and meta information into account.

5.2 Summary

In this analysis, we implemented BPR and WARP loss using two of the most successful modeling approaches, matrix factorization and factorization machines using implicit feedback data from an e-Commerce business. With the technique of pairwise loss learning, the recommending system was able to learn correct item order preference to optimize overall or top ranking with implicit feedback. We specifically tested four models with maximum 25 iterations and discovered that the best performance so far occurs on the maximum iteration using WARP loss with factorization machines by adding additional item information. It turned out that adding this additional feature could improve the overall AUC score, equivalent to overall ranking quality. In summary, pairwise learning is a good candidate for personalized recommendation in implicit feedback.

Bibliography

[1] Chris Burges et al. “Learning to rank using gradient descent”. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 89–96.

[2] Robin Burke. “Hybrid recommender systems: Survey and experiments”. In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.

[3] Paul Covington, Jay Adams, and Emre Sargin. “Deep neural networks for youtube recommendations”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 191–198.

[4] Reginaldo Aparecido Gotardo et al. “Approach to cold-start problem in recommender systems in the context of web-based education”. In: *2013 12th International Conference on Machine Learning and Applications*. Vol. 2. IEEE. 2013, pp. 543–548.

[5] Parth Gupta et al. “Treating Cold Start in Product Search by Priors”. In: *Companion Proceedings of the Web Conference 2020*. 2020, pp. 77–78.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative filtering for implicit feedback datasets”. In: *2008 Eighth IEEE international conference on data mining*. Ieee. 2008, pp. 263–272.

[7] Christopher C Johnson. “Logistic matrix factorization for implicit feedback data”. In: *Advances in Neural Information Processing Systems* 27.78 (2014), pp. 1–9.

[8] Virginia Klema and Alan Laub. “The singular value decomposition: Its computation and some applications”. In: *IEEE Transactions on automatic control* 25.2 (1980), pp. 164–176.

[9] Yehuda Koren. “Collaborative filtering with temporal dynamics”. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009, pp. 447–456.

[10] Yehuda Koren, Robert Bell, and Chris Volinsky. “Matrix factorization techniques for recommender systems”. In: *Computer* 42.8 (2009), pp. 30–37.

[11] Lukas Lerche and Dietmar Jannach. “Using graded implicit feedback for bayesian personalized ranking”. In: *Proceedings of the 8th ACM Conference on Recommender systems*. 2014, pp. 353–356.

[12] Greg Linden, Brent Smith, and Jeremy York. “Amazon. com recommendations: Item-to-item collaborative filtering”. In: *IEEE Internet computing* 7.1 (2003), pp. 76–80.

[13] Andriy Mnih and Russ R Salakhutdinov. “Probabilistic matrix factorization”. In: *Advances in neural information processing systems* 20 (2007).

[14] Siavash Ghodsi Moghaddam and Ali Selamat. “A scalable collaborative recommender algorithm based on user density-based clustering”. In: *The 3rd International Conference on Data Mining and Intelligent Information Technology Applications*. IEEE. 2011, pp. 246–249.

[15] Steffen Rendle. “Factorization machines”. In: *2010 IEEE International conference on data mining*. IEEE. 2010, pp. 995–1000.

[16] Steffen Rendle. “Factorization machines with libfm”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 3.3 (2012), pp. 1–22.

[17] Steffen Rendle et al. “BPR: Bayesian personalized ranking from implicit feedback”. In: *arXiv preprint arXiv:1205.2618* (2012).

[18] Gunnar Schröder, Maik Thiele, and Wolfgang Lehner. “Setting goals and choosing metrics for recommender system evaluations”. In: *UCERSTI2 workshop at the 5th ACM conference on recommender systems, Chicago, USA*. Vol. 23. 2011, p. 53.

[19] Jason Weston, Samy Bengio, and Nicolas Usunier. “Wsabie: Scaling up to large vocabulary image annotation”. In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[20] Shuai Zheng, Chris Ding, and Feiping Nie. “Regularized singular value decomposition and application to recommender system”. In: *arXiv preprint arXiv:1804.05090* (2018).