The Journal of Supercomputing (2022) 78:17776–17796
https://doi.org/10.1007/s11227-022-04580-7

A TDF-WNSP-WLFM algorithm for product recommendation based on multiple types of implicit user behavior

Junchen Fu[1] · **Zhaohui Qi**[2]

Accepted: 1 May 2022 / Published online: 23 May 2022

Abstract

E-commerce platforms usually train their recommender system models to achieve personalized recommendations based on user behavior data. User behavior can be categorized into implicit and explicit feedback. Explicit feedback data have been well studied. However, the implicit feedback data still have many issues, such as the multiple types of behavior data, lack of negative feedback, and lack of the ability to express the real user preference. Targeting these problems of implicit feedback, we propose a TDF-WNSP-WLFM (time decay factor-weight of negative sample possibility-weighted latent factor model) based on the latent factor model for product recommendation. Our method mainly focuses on reconstructing the implicit rating matrix to enable the algorithm to perform better. The TDF-WNSP-WLFM algorithm is tested on two public user behavior datasets from Taobao and REES46, two big e-commerce platforms. Our algorithm compares favorably with other known collaborative filtering methods.

Keywords Collaborative filtering · Latent factor model · Implicit feedback · Recommender system · TDF · WNSP

✉ Zhaohui Qi
zhqi_wy2013@163.com

Junchen Fu
jcfu21@cse.cuhk.edu.hk

[1] Department of Computer Science and Engineering, The Chinese University of Hong Kong, Sha Tin, New Territories 999077, HKSAR, People's Republic of China

[2] College of Information Science and Engineering, Hunan Normal University, Changsha 410081, Hunan, People's Republic of China

1 Introduction

Recommender systems have become indispensable in our life recently since the overload problem online occurred [1]. It is efficient to check the recommendation list when shopping on Amazon or Taobao. During the COVID-19 pandemic, many people tend to purchase products online so that they can stay at home to avoid the risk of exposure to the virus [2]. Unlike the searching engine requiring users to type keywords to retrieve needed content from the databases, the recommender system shows more convenience via automatically generating a recommendation list for each customer based on their user profile [3].

The collaborative filtering (CF) algorithm is widely applied in the recommendation area, which usually depends on users' historical ratings or behavior (e.g., purchase or view). The main advantage of CF is that it does not rely on the specific information of the item, compared to a content-based algorithm which is, on the other hand, totally found on the similarity of different items' content. As a result, it would be a problem when the exact information of products is wrongly input by merchants.

User behavior big data usually plays a crucial role in product recommendation. Sufficient behavior data can be exploited to train the recommender system model. Different user behavior can be classified into two categories which are implicit and explicit feedback [4]. Explicit feedback is the ratings, and reviews that users should fill in intentionally, while implicit feedback is users' unintentional behaviors, e.g., views and purchases. Implicit feedback is easily collected, and the amount is much more than explicit feedback.

Since Koren et al. proposed the latent factor model (LFM) [5] for implicit feedback, matrix factorization algorithms have become a popular collaborative filtering algorithm for dealing with this type of data and have been applied in real industrial applications. In 2009, Rendle et al. proposed Bayesian Personalized Ranking (BPR) [6] centering on implicit feedback. The key idea of BPR is to implement the maximum a posteriori probability (MAP) estimation rule to ensure that observed items should be ranked higher than unobserved items. BPR optimization is a general method that is not limited to matrix factorization models but all machine learning algorithms. The experiments in the paper demonstrate its superiority over traditional matrix factorization algorithms on a dataset of user behavior of music recommendations. In 2014, Christopher et al. proposed a logistic matrix factorization (logistic FM) [7] method to cope with performing recommendations based on implicit feedback datasets, which is a probabilistic model based on the matrix factorization method. Zhao J et al. proposed MFMAP (maximize MAP with matrix factorization), a matrix factorization method focusing on Maximizing MAP [8]. He et al. proposed the state-of-the-art structure of matrix factorization in the format of the neural network [9]. Armielle Noulapeu Ngaffo and Zièd Choukair combined the matrix factorization techniques and neural networks with a twofold regularization to achieve decent prediction accuracy and recommendation quality [10].

The latent factor model's framework does not have a standard approach for processing the time dimension. Several studies are focusing on this issue. In 2005, Li

et al. proposed the use of an exponential time function for attenuation of explicit rating data [11], and in 2009 Koren proposed adding a bias term to the latent factor model [12]. This scholar argued that the user's latent factor is time-aware. In 2010, Xiong et al. proposed a Bayesian probability tensor decomposition model [13]. The two-dimensional rating matrix is transformed into three dimensions (user vector, item vector, and time vector). Chen et al. designed a dynamic decay collaborative filtering (DDCF) algorithm for CF recommendation, which emphasizes variation of preference of users [14]. In 2021, Yu et al. added personal preference fluctuations to the collaborative filtering structure [15]. Human preference is modeled mathematically. Recently, the dynamic changing strategy has been applied to deep neural networks. Wang et al. proposed a time-aware attention-based CF framework. [16].

The implicit feedback data are inherently lacking negative feedback data. Pan et al. proposed to treat all unobserved data as negative feedback data [17] to address this issue. This solution has the disadvantage of being computationally intensive and may mislead the optimization of the model [18]. Another, more common approach is to ignore unobserved data. This approach is also one of the most common approaches. Scholars such as Pan proposed a negative weighting approach based on sampling and weighting to neutralize the extremity [4]. In 2021, Lee et al. adopted two distinct encoder networks to avoid the negative samples problem [19]. To tackle the negative samples generation problem, Chen et. proposed UIWMF (user-activity and item-popularity weighted matrix factorization) to construct negative feedback based on the item popularity and user activity [20]. Researchers usually focus on one type of implicit feedback. However, there are multiple types of interactions between users and items. Taking different types of behavior (e.g., ***view, add-to-cart, favor***) into consideration will enable e-commerce platforms or merchants to emphasize the growth rate of users' specific behavior. For example, when double eleven comes, Chinese e-commerce platforms will regularly desire that the rate of their users' ***add-to-cart*** behavior goes up so that they could guarantee the total sale amount on double eleven to some extent.

We present a collaborative filtering algorithm, TDF-WNSP-WLFM (time decay factor-weight of negative sample possibility-weighted latent factor model), which is time-aware and takes unobserved behavior and different types of implicit feedback into account. Here are the main contributions of this work. First, we make full use of various types of implicit feedback and set the weight of each kind of implicit feedback behavior for different business personalization. Second, time variables are taken into account dynamically. We propose a time decay factor for implicit feedback ratings and apply it to the framework of the latent factor model. Third, we introduce a negative sample construction for unobserved data to solve the lack of negative feedback in the traditional latent factor model for implicit feedback. Fourth, this algorithm is not based on deep neural networks but on the basic matrix factorization method, which means that the computing complexity of this method is relatively low, and our algorithm is easily implemented. Last but not least, this algorithm is tested on two real datasets from the industry in China and abroad. Conclusions are drawn from the two different datasets, which is more rigorous than training only one dataset.

2 Proposed method: TDF-WNSP-WLFM

Our algorithm mainly focuses on the rating construction in the implicit feedback matrix to refine the recommendation model's plausibility and performance. Due to the awareness of time in this algorithm, the algorithm is dynamic and changeable. Only positive samples in the implicit feedback recommendations are solved by generating negative samples from unobserved data.

2.1 Notations

Table 1 includes important notations used in our study.

2.2 Latent factor model for implicit feedback

Koren et al. proposed a latent factor model for implicit feedback [5]. To differentiate from explicit feedback, the preference of user u for item i is defined as p_{ui}. It is a binary variable, i.e., it takes only two values, zero or one. This value is determined by the implicit feedback rating r_{ui}. The p_{ui} can be denoted by binarizing the r_{ui} values [5],

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases} \tag{1}$$

Table 1 Important notations

Notation	Meaning
u	User
i	Item
r_{ui}	Rating of item i by user u
p_{ui}	Preference of item i by user u
U	The set of all users
I	The set of all items
BT_{ui}	The set of all behavior types from user u to item i
B_{ui}	The set of all behaviors of user u on item i
bf_{uik}	Number of the k-th behavior of user u on item i
wr_{ui}	User u's weighted implicit rating on item i
w_i	Weight of the i-th type of implicit feedback
b_{ui}	The behavior of user u on item i
$t_{b_{ui}}$	The time when user u's behavior on item i occurs
t_n	The time to make recommendations
$tdf\text{-}wr_{ui}$	Positive ratings in the rating matrix of the final algorithm
$wnsp\text{-}wr_{ui}$	Negative ratings in the rating matrix of the final algorithm
pop_i	Popularity of item i
pop_u	Popularity of user u

In other words, if a user u has implicit feedback behavior on item i, i.e., if $r_{ui} > 0$, then we define that the user has a preference for the item. On the other hand, if there is no implicit behavior of user u toward item i, there is no preference. However, it is problematic to make such a simple judgment on whether the item has implicit feedback or not. A user might not have a preference for the item when the user has interactions with it. The user may just purchase the item as a gift for someone else. It is also possible that the user is not aware of the item's existence [5].

Therefore, the latent factor model introduces a new concept, "confidence." Define the confidence that user u likes item i as c_{ui}. This variable is applied to measure the degree of confidence in determining the preference between user u and item i. c_{ui} is calculated by Eq. (2) [5],

$$c_{ui} = 1 + \alpha r_{ui} \tag{2}$$

This gives minimum confidence for each user-item pair. The confidence in $p_{ui} = 1$ increases when there are more implicit feedback. The increasing rate is controlled by α. The goal of the latent factor model is to find a vector x_u for user u and a vector y_i for item i. In other words, the predicted preference is defined as the inner product of two vectors: $\hat{p}_{ui} = x_u^T y_i$. These two vectors are called user factors and item factors. It is important to note that the optimization procedure needs to account for all user-item pairs and not just the user-item pairs with rating data as in the explicit feedback latent factor model. The objective function is significant for the training process. In the latent factor model for implicit feedback, the objective function is given in Eq. (3) [5],

$$\min_{x_*, y_*} \sum c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \|y_i\|^2) \tag{3}$$

$\lambda(\sum_u \|x_u\|^2 + \|y_i\|^2)$ in Eq. (3) is a regularization parameter to prevent overfitting of the training set. The exact value of λ is data-dependent and determined by cross-validation.

2.3 Weighted implicit ratings

Nowadays, many studies on implicit feedback only focus on one type of implicit feedback data, such as purchase and browsing. However, in reality, implicit feedback behavior is diverse, and concentrating on only one kind of behavior will waste data. Merchants and e-commerce platforms may have different expectations for users. First, most merchants aim to recommend products to users who are more likely to buy, thus increasing their revenue directly. Second, some start-up e-commerce platforms might desire users to increase the length of time they spend on the platform. Therefore, the users are expected to view the products recommended to them more often instead of solely focusing on their revenue. Third, to increase their transaction volume on a particular day (Double 11 or 618 shopping carnival), large platforms like Tmall or Amazon may hope users increase the probability that the recommended products will be added to their shopping carts during the period that is close to the festival.

Overall, algorithms that could only deal with one type of user implicit feedback behavior cannot achieve customized recommendations for merchants and e-commerce platforms. We suggest that multiple types of implicit feedback should provide different weights to product recommendations according to particular business needs.

The set of all users is denoted as U, and the set of all items is denoted as I. We define that the set of all types of implicit feedback from user u to item i is BT_{ui}. The weight of the k-th type of implicit feedback is w_k, which needs to adjust according to different business requirements. This algorithm mainly relies on the frequency that the user performs implicit feedback behavior for an item. bf_{uik} denotes the number of times the user u performs the k-th type of implicit feedback behavior for item i. The weighted implicit rating wr_{ui} of user u for item i is calculated as,

$$\forall u \in U,\ i \in I : wr_{ui} = \sum_{k \in BT} w_k bf_{uik} \tag{4}$$

The advantage of constructing the rating matrix by this method is that the recommendation list would be generated to make users tend to perform the particular implicit behavior carrying a high value of weight. Our model has a scalable structure to incorporate multiple types of implicit feedback. For more types of implicit feedback, we could set different weights for them. A larger number of types will cause the weight constructing time complexity to increase linearly. However, if the number of types increases excessively, the performance will drop. For the dataset with many behavior types, the platform could only focus on the major types and ignore the minor types. The platform should pursue a balance between the number of behavior types and the model performance and efficiency.

The practical weight setting should be set manually according to the business requirement. The platform requires the frequency of a specific type of behavior to increase in the future. They can set a large value for this type of behavior. The platform should maintain the balance of the actual meaning behind each type of behavior and actual recommendation metrics and adjust the actual weight setting according to the actual performance.

2.4 Time decay factor

The ratings generated by weighting the individual frequencies as described above only solve multiple implicit feedback behaviors. However, it does not address the traditional algorithm's lack of time sensitivity.

The latent factor model for implicit feedback suffers from time insensitivity. For example, two identical viewing behaviors in a dataset occurred in the last two days, and the other occurred a year ago. In the traditional LFM, both will be assigned the same weight. This may be unreasonable, as human interests are changing dynamically. Past preferences are not necessarily representative of their ongoing preferences. A recommender system based on users' real-time preferences mostly achieves

more accurate personalized recommendations, and e-commerce platforms tend to implement such recommender systems to gain more views and sales.

In summary, time sensitivity is vital for recommender systems. In this regard, Li et al. proposed that users' preferences decay exponentially [11]. This idea coincides with Ebbinghaus' forgetting curve, i.e., people forget things faster and then slower. Thus, we could assume that users' preferences conform to a similar rule and are presented in the form of an exponential function that is firstly very fast and then slow. Accordingly, here proposes a time decay factor.

The set of all implicit feedback behaviors of all users u for item i is denoted as B_{ui}, and each implicit behavior in B_{ui} is denoted as b_{ui}. As mentioned above, the weight of this algorithm, i.e., the weight of the k-th implicit feedback, can be set by the one employing this algorithm. Therefore, we denote by $w_{b_{ui}}$ the weight represented the behavior of user u for item i, and the weight is determined by the type of implicit feedback behavior of b_{ui}. Since this algorithm is time-sensitive, the time point at which the recommendation is made is denoted as t_n, and the time point at which b_{ui} occurs is marked as $t_{b_{ui}}$. Therefore, the implicit feedback rating $tdf\text{-}wr_{ui}$ (time decay factor-weighted rating) is calculated as follows:

$$\beta \in (0,1), \forall u \in U,\ i \in I : tdf\text{-}wr_{ui} = \sum_{b_{ui} \in B_{ui}} \beta^{t_n - t_{b_{ui}}} w_{b_{ui}} \tag{5}$$

β is a decimal number between zero and one. In the exponential function $y = a^x$, when a is less than one and greater than zero, the decay rate goes fast at the beginning and then slow. Figure 1 is an image of a function for the case that a is 0.5. For the value of t_n and $t_{b_{ui}}$, different time units can be applied according to the datasets. Generally speaking, the user's various implicit feedback behaviors are recorded by the system's backend logs and are mostly presented in the form of timestamps, i.e., Unix timestamps [21], which are the number of seconds accumulated from January

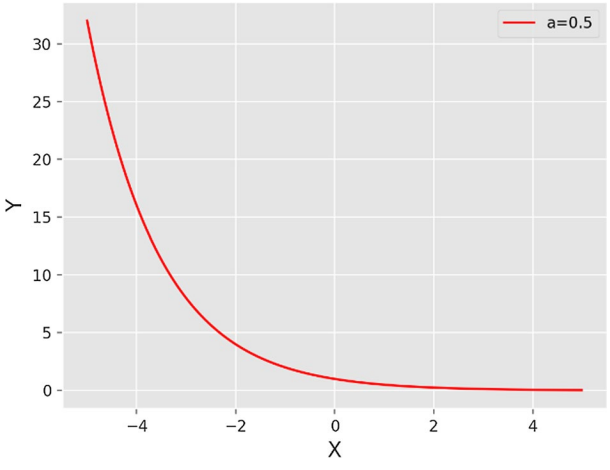

Fig. 1 Graph of $y = a^x$ where $a = 0.5$

1, 1970. Timestamps can be interpreted into the corresponding year, month, day, hour, minute, and second. In our method, the time unit, i.e., the unit of $t_n - t_{b_{ui}}$, is defined as one day.

2.5 Negative sample generating

The *tdf-wr*$_{ui}$ solves two problems that exist in traditional algorithms, namely, the waste of multiple types of implicit feedback and the time insensitivity problem. Thus, the introduction of this algorithm for positive rating has been completed. In this section, we introduce the construction of negative implicit feedback from the unobserved data to boost the performance further.

Lacking implicit feedback does not necessarily mean that the user is not interested in the item. It also could be that the user is not aware of the item's existence. For users, items with higher popularity, in reality, are often more likely to discover by users and thus interact. Therefore, here proposes the hypothesis that the possibility of a user learning about an item is related to the popularity of the item.

As mentioned above, we establish weights for each type of implicit feedback and *tdf* based on the time when the behavior happens. Here also assumes that the popularity of items is associated with the time decay factor of user behavior and the weights of each type of behavior.

B_i denotes the set of all behaviors for item i. Any behavior of item i in the set is denoted as b_i, and the time point of the behavior is denoted as t_{b_i}. w_{b_i} stands for the weight of the behavior. The current time point is t_n. Thus, the popularity of item i, pop_i, is calculated as,

$$\beta \in (0,1), \forall i \in I : pop_i = \sum_{b_i \in B_i} \beta^{t_n - t_{b_i}} w_{b_i} \tag{6}$$

Since that, a direct negative sample addition based solely on the above calculation may result in a significant scale difference of values between the negative and positive samples. Therefore, the popularity requires normalization to form a complete implicit feedback rating matrix. The negative sample weights generated after normalization are named weight of negative sample possibility (WNSP) in this paper and are calculated as,

$$wnsp_i = -\sigma\left(pop_i\right) * \epsilon \tag{7}$$

σ is a normalization function that maps the popularity to an interval from zero to one and ϵ is a constant factor that is supposed to be adjusted for different samples scales. Common normalization functions include Z-score standardization and linear function normalization (min–max scaling) [22]. The former usually requires the data to roughly present a Gaussian distribution, meaning the requirements are more stringent. Therefore, we take the latter as shown in Eq. (8),

$$\sigma(X) = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{8}$$

X_{max} and X_{min} are the maximum and minimum values of the original data. Each item's weight of negative sample possibility can be calculated based on Eq. (33). The set of negative sample ratings to be inserted is denoted as *NS* (Negative Sample Set). The set of users not having implicit feedback with the item i is represented as ϕ_i. *NS* is a set of triplets (user, item, and negative ratings). The mathematical expression of *NS* is shown in 9,

$$NS = \bigcup_{i \in I} \phi_i \times \left(i, wnsp_i\right) \tag{9}$$

In general, there will be too many items and users in the sample, causing *NS* to be immense. We only extract a part of the items with larger $wnsp_i$ for *NS* instead of adding negative samples of all items. The elements in *NS* are tuples with three values, which can be directly added to the rating matrix according to the corresponding user-item relationship, thus forming a complete rating matrix. The negative ratings put into the rating matrix are denoted as $wnsp\text{-}wr_{ui}$. In this work, we combine the positive ratings, $tdf\text{-}wr_{ui}$, and the negative ratings, $wnsp\text{-}wr_{ui}$ to form a final rating matrix. The rating in the final rating matrix is defined as $tdf\text{-}wnsp\text{-}wr_{ui}$.

2.6 TDF-WNSP-WLFM

Based on the three steps mentioned above, we transform the rating matrix into one that can be implemented according to business requirements, is time-sensitive, and has negative rating data. Combining this rating matrix with the latent factor model for implicit feedback leads to the recommendation algorithm in this study, i.e., time decay factor-weight of negative sample possibility-weighted latent factor model (TDF-WNSP-WLFM). However, the rating matrix here has negative sampling, which is different from the latent factor model, which only contains nonnegative ratings. Thus, the mathematical definition of the latent factor model should be modified. In this algorithm, the preference p_{ui} is defined in Eq. (10),

$$p_{ui} = \begin{cases} 1 & tdf\text{-}wnsp\text{-}wr_{ui} > 0 \\ 0 & tdf\text{-}wnsp\text{-}wr_{ui} \leq 0 \end{cases} \tag{10}$$

The meaning of the above formula is that when the rating data are zero or negative, user u has no preference for item i. When the rating is positive, user u likes the item i. The function determining the confidence level c_{ui} is computed as,

$$c_{ui} = 1 + \alpha \mid tdf\text{-}wnsp\text{-}wr_{ui} \mid \tag{11}$$

The confidence of whether a user u likes item i is only related to the absolute value of the rating. The negative ratings among the rating matrix in this algorithm represent the confidence for the absence of preference of user u for item i, which is only related to the absolute value of this negative number. α serves as a constant factor, adjusted according to the sample. Since user preference and confidence have been redefined above, the objective function $f(x)$ does not need to change, as shown in Eq. (3).

The training and inference time complexity is the same as the traditional ALS model. The only difference we made in our paper is the construction of the rating matrix. The user set is denoted by U. I is the item set, and IM is the implicit behavior we have. The first step is to construct the weights and time decay factor based on timestamps. Since this step can be done by linear scan, the time complexity is $O(|\ IM\ |)$. After this step, the dataset is aggregated, and the dataset for this step is defined as R. To construct the negative samples, the aggregated dataset has to be linearly scanned to get the popularity of each user and item. Then, we have the time complexity of $O(|\ R\ |)$. To make the negative samples, we have to define the ratio of users and items we would like to add to the final ratings. The ratios of users and items are defined as r_u and r_i, respectively. The ratios of the user and item dataset are selected to conduct the Cartesian product. Overall we have the time complexity of $O(r_u\times |\ U\ | \times r_i\times |\ I\ |)$. Since the $|\ R\ |$ is smaller than $|\ IM\ |$, the overall time complexity of the construction is equal to $O(|\ IM\ | + r_u\times |\ U\ | \times r_i\times |\ I\ |)$.

3 Experiments and evaluation

The following section describes the evaluation of the algorithm on industrial data, and the flowchart is shown in Fig. 2. The whole experiment contains seven main procedures. First, we combine implicit feedback and their time point as the input data. After that, data preprocessing ensues to prepare the construction of rating matrices, which includes data cleaning, data selecting, and data scaling.

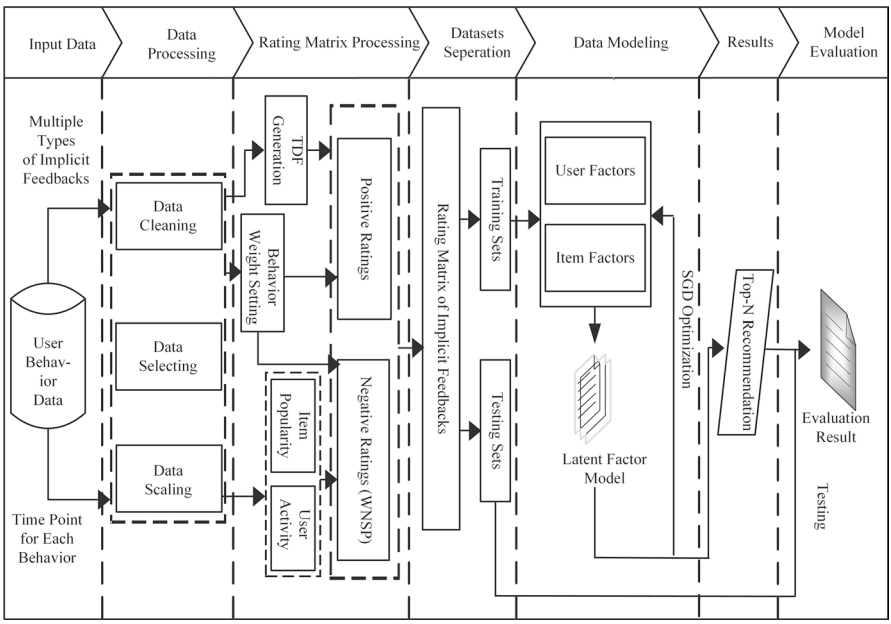

Fig. 2 Flowchart of the whole experiment procedure

Thereafter, positive ratings are generated from the weighted implicit feedback after time decaying. Established on the combination of item popularity, user activity, and weight setting, we could acquire the negative ratings from the unobserved data in the datasets. The later procedure is to associate these ratings to a complete rating matrix for our algorithm and separate it into training and testing sets. The final steps aim to train and test the Top-N recommendation model found on the LFM algorithm.

The Pandas [23] and Numpy [24] libraries among Python are applied for the data analysis and processing part, and the Matplotlib library [25] is used for the data visualization. The primary recommender system modeling in this paper adopts the "Implicit" library featuring implicit feedback recommendations developed by Benred et al. on GitHub in 2016 [26]. Nowadays, more recommender system researchers and developers keep maintaining and optimizing this library. It implements mainstream ML recommendation algorithms such as LFM and BPR.

3.1 Datasets description

This paper utilizes two sets of large public datasets in China and abroad. More tests make the performance of the algorithm better present. We adopt the real online shopping behavior data of about one million users from Taobao on the Tianchi big data platform [27]. The format of these data is shown in Table 2.

For the four behavior types, they represent the following meanings, *pv*, the abbreviation of a page view, which means that the user clicked on the item detail page, *buy*, a purchase, which means the user bought the item. *cart*, a user added the item to their shopping cart, *fav*, the user favorited the item.

Another set of data comes from the user behavior from the REES46 e-commerce website on the Kaggle big data platform [28]. This e-commerce website is a large overseas e-commerce platform. The users of this dataset are from a country in the Middle East region. The format of the Kaggle dataset is roughly the same as that of the aforementioned Tianchi Big Data platform but with some additional fields related to item content. The specific information is illustrated in Table 3.

The meaning of the three behavior types from REES46 is as follows, *view*, the user viewed the item's detail page, *cart*, the user added the item to their shopping

Table 2 Description of Taobao datasets

Categories	Explanation
User ID	Serialized user ID
Item ID	Serialized item ID
Category ID	Serialized category ID
Behavior Type	*pv*, *buy*, *cart*, *fav*
Timestamp	Timestamp for each user behavior

Table 3 Description of REES46 datasets

Categories	Explanation
Event_time	The time when the event happens
Event_type	*purchase*, *cart*, *view*
Product_id	Serialized Product ID
User_id	Serialized User ID
Category_id	Category ID of the product
Brand	Brand of the product
Price	Price of the product

cart, *purchase*, the user purchased the item. The above two datasets have different field names but similar data contents. The user ID, item ID, and behavior type are required in both datasets. The time decay factor is involved, meaning that this algorithm also demands the time point of the behavior.

3.2 Data scaling and cleaning

Data preprocessing is a crucial step in data modeling, as there is a high probability of missing data and errors in real datasets. The primary purpose of data preprocessing is to detect these problems. As mentioned in the data description, since this paper applies a collaborative filtering recommendation algorithm, we only require four fields in the large dataset: user ID, item ID, type of behavior, and time point. There are no null values in the two datasets. Therefore, we do not need to fill the dataset with values. In terms of the handling of outliers, this paper focuses on whether there is a problem in the time point. There are some deviations between the description of the dataset given by the authors and the data itself. In our experiment, we take the data as the focus and the description as a supplement to understanding.

The time format of the data from both the Tianchi and Kaggle platforms is transformed into a human-readable year-month-day format. We found that the Tianchi platform's data period is from 1902 to 2037. The authors mentioned in the dataset description that the period is from November 25, 2017, to December 3, 2017. We remove the data outside that range. The original data volume is 100150806, and the data without abnormal timestamps are 98914533. For the data of the REES46 platform on the Kaggle platform for October and November 2019, since the time format given is UTC, i.e., the corresponding year, month, and the day is given, there is no time misalignment problem after checking.

The time decay factor in Sect. 2.4 is expressed mathematically, and the time decay unit applied in this paper is one day. For the Taobao dataset, the recommendation time points t_n are defined as November 25, 2017, and November 31, 2019, for the dataset REES46. These time points are the maximum values among the whole dataset. For each record of the datasets, the time interval between the time when the behavior happens and t_n is defined as a gap, which could be computed by $t_n - t_{b_{ui}}$.

The collaborative filtering algorithm has specific requirements for the amount of data. We will only perform the recommendation for high-quality users and items

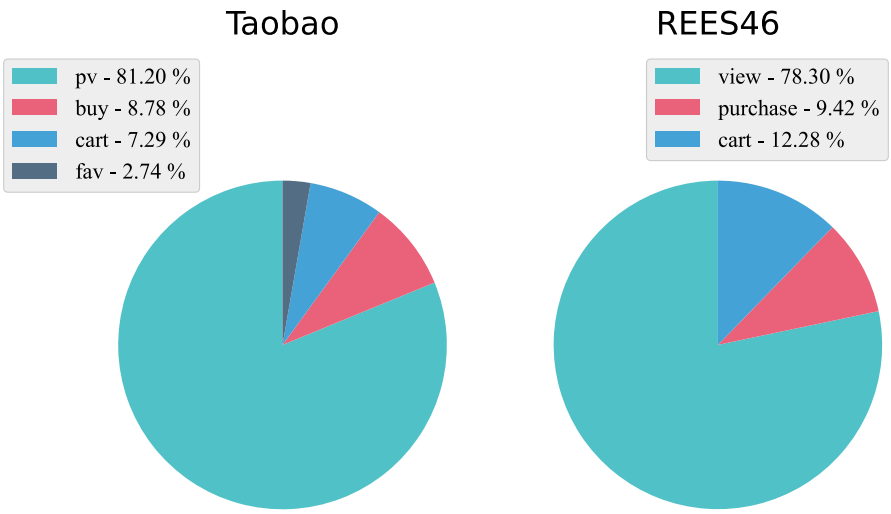

Fig. 3 The distribution of behavior types in two data platforms

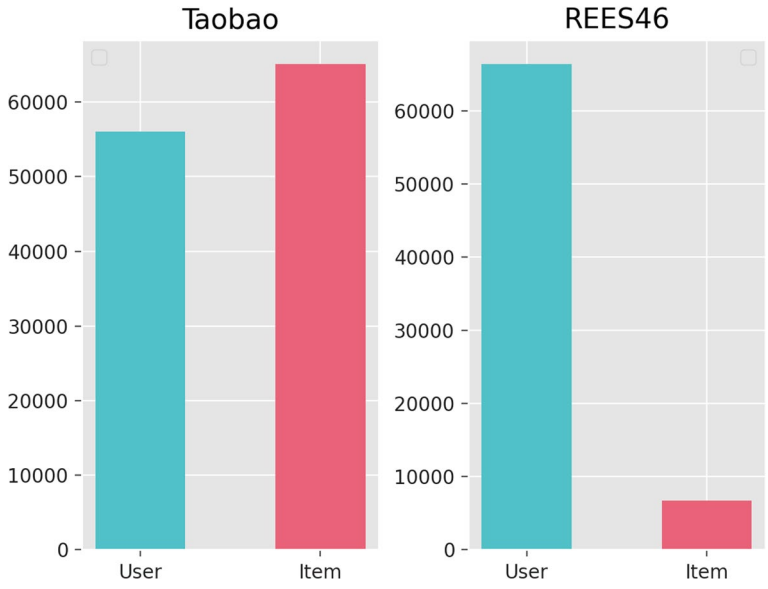

Fig. 4 Numbers of users and items on Taobao and REES46 platforms

in this experiment. Among various implicit feedback behaviors, the most reflective of quality customers and items is the purchase because this behavior involves a certain degree of monetary payment. We define the top ten percent of users and items as quality users and items. The interaction between quality users and quality items is modeled and evaluated in the following section. After filtering, there are 6249200 records in the Kaggle dataset and 3302114 records in the Tianchi dataset.

The distribution of different types in all records is as seen in Fig. 3, which shows that the proportion of behaviors is similar in these datasets.

The number of users and items also significantly impacts the recommendation results. The low number of items in the rating matrix leads to fewer candidates, resulting in more accurate recommendations. Figure 4 describes the number of users and items in the two datasets. The number of users is similar in both datasets. However, the number of items in the REES46 platform is significantly lower, about one-tenth of the number of users.

3.3 Modeling

The core of our algorithm lies in the processing and modification of the implicit feedback rating matrix, and therefore the transformation of the rating matrix is of paramount importance. The rating matrix in the TDF-WNSP-WLFM algorithm is composed of $tdf\text{-}wnsp\text{-}wr_{ui}$. The construction of positive ratings in the matrix requires the determination of two factors, i.e., the weight setting for each implicit feedback behavior in the customized recommender system and the time decay factor of each behavior.

Our recommender system model is designed for merchants, meaning that the main focus is on users' purchase behavior. Since the rating allocation is based on the actual scenario, the exact number depends on both mission and performance [29]. Therefore, we assign the purchase behavior with the most significant value. Based on the performance and the mission of providing recommendations to the merchants, we initial the *purchase* and *buy* behavior in both datasets to ten and *view* and *pv* to one. The *cart* and *fav* behavior from the Taobao platform and *cart* behavior from REES46 are assigned to two.

For the time decay factor, according to the decaying method mentioned in Sect. 2.4, the gap defined in Sect. 3.2, the $tdf\text{-}wr_{ui}$ should be calculated as,

$$\beta \in (0,1), \forall u \in U,\ i \in I\ :\ tdf\text{-}\ wr_{ui} = \sum_{b_{ui} \in B_{ui}} \beta^{gap} w_{b_{ui}} \tag{12}$$

After debugging, the model performs better when setting β to about 0.8 for the Tianchi platform and 0.5 for the Kaggle platform. We have to consider the data scale for the construction of negative samples. If all unobserved user-item pairs are exploited to build negative samples, which will lead to a large amount of the occupation of storage and computation resources.

For most products, because the product's popularity is presented too unevenly, negative feedback is mostly with lower popularity. Figure 5 shows the distribution of item popularity on the Tianchi platform. Most items are at the bottom, as shown in the picture. Since the final negative weight has to be normalized, the role of putting in this trivial data is not much different from filling in zero directly. Therefore, we should filter most of them.

Most of the users are inactive, which is similar to the popularity distribution of the products. Due to our limited computation resources and storage, sampling is

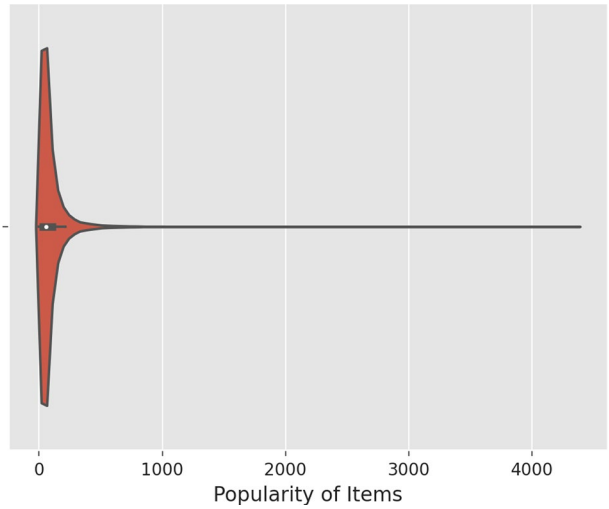

Fig. 5 Distribution of item popularity on the Tianchi platform

performed for users. The users' activity is defined similarly to the items' popularity. Define the set of all behavior of user u as B_u, and each behavior in this set is denoted as b_u. We define the weight of the behavior as w_{b_u}, and the occurrence time point of the behavior is t_{b_u}. The popularity pop_u of each user u is calculated as,

$$pop_u = \sum_{b_i \in B_u} \beta^{t_n - t_{b_u}} w_{b_u} \tag{13}$$

The ratings of the top one percent of items and users with high popularity are selected, and $wnsp_i$ is calculated according to Eq. (7). The last step for data preprocessing is adding negative ratings to the rating matrix.

In this experiment, two sets of data are trained and tested separately. First, 80% of the data in the rating matrix is randomly selected as the training set, and the remaining 20% is used as the test set, which is emptied from the rating matrix. The number of training epochs is set to 100, and the number of latent factors is 100. The stochastic gradient descent learning rate is 0.2, and the regularization parameter λ is 0.01. This paper determines the λ by multiple trials of threefold cross-validation. In our experiments, we tested the λ of different values and selected λ according to the recommendation performance. Among all trials, we choose the one with better performance.

3.4 Evaluation of models

Several algorithms are applied as the baseline: the BPR algorithm based on matrix factorization, logistic matrix factorization algorithm [7], the K-nearest neighbor

algorithm with cosine similarity [30], TF-IDF [31], and BM25 [32]. For the ablation study, the LFM after only weighting, LFM with weighting and time decay factors, and the LFM only after the adding of the WNSP are implemented along with the main algorithm of this paper to train models on two datasets. The models are evaluated using three metrics, i.e., AUC, Precision, and NDCG [33]. The following methods are all assessed according to the Top-20 recommendation.

Define the positive results in the recommendation list as *positiveClass*. $rank_i$ is the correct rank for item i from the recommendation result. M denotes the number of positive results. N is the number of negative results. The AUC (area under curve) calculation method in this paper is defined as,

$$AUC = \frac{\sum_{i \in positiveClass} rank_i - \frac{M(1+M)}{2}}{M \times N} \tag{14}$$

Precision is a simple evaluation metric. Define tp_u as the correct results in the recommendation list. fp_u is the incorrect results in the recommendation list, the calculation of precision is below,

$$Precision = \frac{tp_u}{tp_u + fp_u} \tag{15}$$

NDCG (normalized discounted cumulative gain) can evaluate the ranking performance more precisely compared to the AUC. NDCG is the normalized format of DCG (discounted cumulative gain). The NDCG is calculated as,

$$NDCG_N = \frac{DCG_N}{IDCG_N}, DCG_N = \sum_{i=1}^{N} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{16}$$

Define the number of items in the recommendation list as N. rel_i represents that relevant score of the item at the i-th position from the recommendation list. In our experiment, if the result is positive, then $rel_i = 1$; otherwise, $rel_i = 0$. IDCG (ideal discounted cumulative gain) is the maximum value we can get for the DCG.

Table 4 Performance on Taobao user dataset

	AUC	Precision	NDCG
TDF-WNSP-WLFM	0.5319	0.0603	0.0530
TDF-WLFM	0.5314	0.0600	0.0509
WNSP-WLFM	0.5316	0.0602	0.0506
WLFM	0.5316	0.0602	0.0508
KNN(cosine)	0.5095	0.0145	0.0150
KNN(TF-IDF)	0.5186	0.0296	0.0296
KNN(BM25)	0.5309	0.0686	0.0520
LMF	0.5028	0.0460	0.0045
BPR	0.5212	0.0376	0.0312

Table 5 Performance on REES46 user dataset

	AUC	Precision	NDCG
TDF-WNSP-WLFM	0.6631	0.2642	0.2252
TDF-WLFM	0.6591	0.2613	0.2251
WNSP-WLFM	0.6571	0.2577	0.2215
WLFM	0.6570	0.2574	0.2216
KNN(cosine)	0.6106	0.1694	0.1523
KNN(TF-IDF)	0.6525	0.2500	0.2205
KNN(BM25)	0.6365	0.2100	0.1912
LMF	0.5222	0.0460	0.1048
BPR	0.5821	0.1286	0.1048

The latent factor model after weighting is denoted as WLFM. The rating matrix in this algorithm is obtained by summing the weights. The algorithm that ignores negative samples and combines a time decay factor with WLFM is denoted as TDF-WLFM. The WLFM that only contains negative samples is defined as WNSP-WLFM. Each algorithm tested on the Taobao and REES46 datasets is in Table 4 and Table 5, respectively. The performance of the latent factor model implemented in our study is better than the common methods: K-nearest neighbor algorithm using cosine similarity and the BPR algorithm. The TDF-WNSP-WLFM proposed in this paper has the best overall performance among all the algorithms on the two datasets. Although the AUC in the Kaggle dataset is close to K-nearest neighbor (cosine) and BPR algorithms, it is over 400% higher in precision and NDCG.

The item-based K-nearest neighbor cosine similarity algorithm and the BPR algorithm perform differently in the two datasets. However, neither can outperform the basic weighted latent factor model. The KNN (BM25) shows remarkable performance on the Taobao user dataset, which has an even higher score in precision compared to the TDF-WNSP-WLFM algorithm. However, the performance on the Kaggle dataset is not very decent compared to the KNN (TF-IDF) method, which shows lower performance in the Tianchi dataset.

A comparison between the two datasets shows that each algorithm has better performance on the Kaggle data than the Tianchi data. The sparsity of the data and the significant difference in the number of items may dramatically impact the quality of recommendations.

We extend our experiment on the Kaggle dataset by giving twice tolerance to the data selecting procedure. Since the total size of the dataset is doubled, the performance drops due to the average quality of users and items becoming low. Although the overall performance for all algorithms decreases, our TDF-WNSP- WLFM still have the best performance among all algorithm. The results are shown in Table 6.

One significant feature in these tables is that TDF-WNSP-WLFM and TDF-WLFM are similar to the WLFM algorithm in the Tianchi platform dataset. The performance of TDF-WLFM is even lower than WLFM in evaluation in AUC and precision, while the performance of both algorithms exceeds the WLFM in Kaggle dataset. The time decay factor plays a vital role in the Kaggle dataset. This dataset

Table 6 Performance on REES46 extended user dataset

	AUC	Precision	NDCG
TDF-WNSP-WLFM	0.6516	0.2349	0.2094
TDF-WLFM	0.6502	0.2348	0.2089
WNSP-WLFM	0.6491	0.2348	0.2079
WLFM	0.6482	0.2345	0.2080
KNN(cosine)	0.6027	0.1508	0.1370
KNN(TF-IDF)	0.6434	0.2297	0.2011
KNN(BM25)	0.6296	0.1876	0.1761
LMF	0.5215	0.0421	0.0302
BPR	0.5756	0.1092	0.0930

has a more extended period, which may be why it could successfully boost the performance with the factor. The WNSP-WLFM shows comparative performance with WLFM on both datasets. This tells us that simply adding the negative samples to the dataset might not affect the overall result. However, if we combine the WNSP with the TDF, the overall performance could be strengthened. Another conclusion can be drawn that the K-nearest neighbor method using a different metric to calculate the similarity on different datasets may vary. When researchers would like to test the overall performance of the KNN method, the ideal way is to test all metrics and find the one with the best performance to use. In our experiments, the most common KNN with cosine similarity has the worst performance among all KNN-based methods.

In order to check the effectiveness of the time decay factor, we conducted an experiment on the Kaggle dataset since it has a relatively long-time span, and the TDF-WNSP-WLFM shows more significant improvement compared to the WLFM method. We change the time unit to an hour. The performance of the model does not have a significant difference. On the other hand, if we increase the time unit to one week, the performance drops to the performance similar to WNSP-WLFM and WLFM. This means that the time decay factor is effective only if the time unit is short enough for the model to examine the preference decay. On the other hand, setting one day as the time unit is proper for the Kaggle dataset.

The evaluation is based on offline tests in academia, which may not represent its actual performance in the industrial field. It requires to be further tested in real-time on e-commerce websites.

4 Conclusion

This paper improves the latent factor model framework by applying multiple types of implicit feedback. We propose a method to adjust the weight of each type of behavior according to business scenarios to customize the algorithm for merchants or e-commerce platforms.

At the same time, we propose a time decay factor (TDF), which enables the implicit feedback rating data to dynamically decay according to the time point when the recommendation happens. Due to the similarity of users' preferences and ratings after time decaying, this algorithm improves overall performance. For products with implicit feedback that happened a long time ago, users are more likely to forget about them, or the products only matched users' preferences at that particular period.

Negative samples are introduced in this study to increase the diversity and plausibility of the rating matrix. The implicit feedback itself is inherently positive. In this algorithm, the negative samples are generated based on the user's possibility of not favorite the product.

To evaluate this algorithm, we compare the performance with several commonly used recommendation algorithms in modeling recommender systems on two datasets of real e-commerce platforms. We find that the latent factor model has better performance in resolving multiple implicit feedback problems. In the algorithm evaluation, it can be inferred that the presence of a time decay factor is more necessary for data with a more extended period. In contrast, the LFM algorithm could be applied directly after weighting the data for a short period to obtain a similar recommendation performance as the TDF-WNSP-WLFM. The WNSP should be added along with the TDF, simply adding the WNSP may not improve the overall recommendation performance.

Overall, the TDF-WNSP-WLFM achieves performance improvement without a complex structure. The rating construction time complexity is relatively low. The general performance can be improved compared with the original method since our method has the same training structure as the LFM algorithm. It can be applied in the paralleled computing platform as the LFM does. It can be applied for a product recommendation or any other platform that collects multiple types of user behavior. Future work could improve the ways of weighting and exact weights for a customized recommendation. A more sophisticated and effective approach might be applied.

Declarations

Funding This work is supported by Humanities and Social Sciences Research of Ministry of Education of China (Grant No. 19YJAZH069) and Hunan Provincial Science and Technology Project Foundation (Grant No. 2018TP1018).

References

1. Jacoby J (1984) Perspectives on information overload. J Consum Res 10(4):432–435
2. Ali B (2020) Impact of COVID-19 on consumer buying behavior toward online shopping in Iraq. Econ Stud J 18(42):267–280
3. Belkin NJ, Croft WB (1992) Information filtering and information retrieval: two sides of the same coin? Commun ACM 35(12):29–38. https://doi.org/10.1145/138859.138861
4. Yuan F (2018) Learning implicit recommenders from massive unobserved feedback. Thesis

5. Hu Y, Koren Y, Volinsky C (2008) Collaborative filtering for implicit feedback datasets. In: 2008 Eighth IEEE international conference on data mining, pp 263–272 . https://doi.org/10.1109/ICDM.2008.22
6. Rendle S, Freudenthaler C, Gantner Z, Schmidt-Thieme L BPR (2012) bayesian personalized ranking from implicit feedback. CoRR arxiv:1205.26181
7. Johnson CC (2014) Logistic matrix factorization for implicit feedback data. Adv Neural Inf Process Syst 27(78):1–9
8. Zhao J, Fu Z, Sun Q, Fang S, Wu W, Zhang Y, Wang W (2019) Mfmap: Learning to maximize map with matrix factorization for implicit feedback in recommender system. KSII Trans Int Inf Syst (TIIS) 13(5):2381–2399. https://doi.org/10.3837/tiis.2019.05.008
9. He X, Liao L, Zhang H, Nie L, Hu X, Chua T-S (2017) Neural collaborative filtering. In: Proceedings of the 26th international conference on world wide web. WWW '17, pp 173–182. International world wide web conferences steering committee, Republic and Canton of Geneva, CHE. https://doi.org/10.1145/3038912.3052569
10. Noulapeu Ngaffo A, Choukair Z (2022) A deep neural network-based collaborative filtering using a matrix factorization with a twofold regularization. Neural Comput Appl 34(9):6991–7003. https://doi.org/10.1007/s00521-021-06831-9
11. Ding Y, Li X (2005) Time weight collaborative filtering. In: Proceedings of the 14th ACM International conference on information and knowledge management. CIKM '05, pp 485–492. Association for Computing Machinery, New York https://doi.org/10.1145/1099554.1099689
12. Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. Computer 42(8):30–37. https://doi.org/10.1109/MC.2009.263
13. Xiong L, Chen X, Huang T.-K, Schneider J, Carbonell JG (2010) Temporal collaborative filtering with bayesian probabilistic tensor factorization. In: Proceedings of the 2010 SIAM international conference on data mining, pp 211–222 . https://doi.org/10.1137/1.9781611972801.19
14. Chen Y-C, Hui L, Thaipisutikul T (2021) A collaborative filtering recommendation system with dynamic time decay. J Supercomput 77(1):244–262. https://doi.org/10.1007/s11227-020-03266-2
15. Yu J, Shi J, Chen Y, Ji D, Liu W, Xie Z, Liu K, Feng X (2021) Collaborative filtering recommendation with fluctuations of user' preference. In: 2021 IEEE International conference on information communication and software engineering (ICICSE), pp 222–226 . https://doi.org/10.1109/ICICSE52190.2021.9404120
16. Wang R, Wu Z, Lou J, Jiang Y (2022) Attention-based dynamic user modeling and deep collaborative filtering recommendation. Expert Syst Appl 188:116036. https://doi.org/10.1016/j.eswa.2021.116036
17. Pan R, Zhou Y, Cao B, Liu NN, Lukose R, Scholz M, Yang Q (2008) One-class collaborative filtering. In: 2008 Eighth IEEE international conference on data mining, pp 502–511. https://doi.org/10.1109/ICDM.2008.16
18. Aggarwal CC (2016) Content-based recommender systems. Recommender systems. Springer, Cham, pp 139–166
19. Lee D, Kang S, Ju H, Park C, Yu H (2021) Bootstrapping user and item representations for one-class collaborative filtering, pp 317–326. Association for computing machinery, New York https://doi.org/10.1145/3404835.3462935
20. Chen L, Yang W, Li K, Li K (2021) Distributed matrix factorization based on fast optimization for implicit feedback recommendation. J Intell Inf Syst 56(1):49–72. https://doi.org/10.1007/s10844-020-00601-0
21. Cristian F, Jahanian F (1991) A timestamp-based checkpointing protocol for long-lived distributed computations. In: [1991] Proceedings tenth symposium on reliable distributed systems, pp 12–20. https://doi.org/10.1109/RELDIS.1991.145399
22. Zill DG (2020) Advanced engineering mathematics. Jones and Bartlett Publishers, New York
23. McKinney W (2011) pandas: a foundational python library for data analysis and statistics. Python High Perform Sci Comput 14(9):1–9
24. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, van Kerkwijk MH, Brett M, Haldane A, del Río JF, Wiebe M, Peterson P, Gérard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE (2020) Array programming with NumPy. Nature 585(7825):357–362. https://doi.org/10.1038/s41586-020-2649-2
25. Hunter JD (2007) Matplotlib: A 2d graphics environment. Comput Sci Eng 9(03):90–95. https://doi.org/10.1109/MCSE.2007.55

26. Benfred HLEA Implicit (2016) https://github.com/benfred/implicit
27. Tianchi: User Behavior Data from Taobao for Recommendation (2018). https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
28. Kechinov M (2019) eCommerce behavior data from multi category store. https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store
29. Falk K (2019) Practical recommender systems. Simon and Schuster, Shelter Island
30. Jaiswal S, Kharade T, Kotambe N, Shinde S (2020) Collaborative recommendation system for agriculture sector. In: ITM web of conferences, vol 32, p 03034. EDP Sciences
31. Wang B, Liao Q, Zhang C (2013) Weight based knn recommender system. In: 2013 5th international conference on intelligent human-machine systems and cybernetics, vol 2, pp 449–452 . https://doi.org/10.1109/IHMSC.2013.254
32. Suchal J, Návrat P (2010) Full text search engine as scalable k-nearest neighbor recommendation system. In: Bramer M (ed) Artificial intelligence in theory and practice III. Springer, Berlin, Heidelberg, pp 165–173
33. Chen M, Liu P (2017) Performance evaluation of recommender systems. Int J Perform Eng 13(8):1246. https://doi.org/10.23940/ijpe.17.08.p7.12461256