Expert Systems With Applications 137 (2019) 253–265

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

An effective distributed predictive model with Matrix factorization and random forest for Big Data recommendation systems

Badr Ait Hammou[a,*], Ayoub Ait Lahcen[a,b], Salma Mouline[a]

[a] LRIT, Associated Unit to CNRST (URAC 29), Rabat IT Center, Faculty of Sciences, Mohammed V University, Rabat, Morocco
[b] LGS, National School of Applied Sciences (ENSA), Ibn Tofail University, Kenitra, Morocco

ARTICLE INFO

Article history:
Received 5 December 2018
Revised 22 June 2019
Accepted 22 June 2019
Available online 22 June 2019

Keywords:
Big Data
Distributed computing
Random forest
Matrix factorization
Apache spark
Recommendation systems

ABSTRACT

Recommendation systems have been widely deployed to address the challenge of overwhelming information. They are used to enable users to find interesting information from a large volume of data. However, in the era of Big Data, as data become larger and more complicated, a recommendation algorithm that runs in a traditional environment cannot be fast and effective. It requires a high computational cost for performing the training task, which may limit its applicability in real-world Big Data applications.

In this paper, we propose a novel distributed recommendation solution for Big Data. It is designed based on Apache Spark to handle large-scale data, improve the prediction quality, and address the data sparsity problem. In particular, thanks to a novel learning process, the model is able to significantly speed up the distributed training, as well as improve the performance in the context of Big Data. Experimental results on three real-world data sets demonstrate that our proposal outperforms existing recommendation methods in terms of Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and computational time.

1. Introduction

In recent years, Big Data has emerged as the key to success in many companies, scientific disciplines, and government endeavors. The term Big Data generally refers to data sets whose volume is beyond the capacity of conventional tools to capture, manage, and process data within an acceptable computational time. The challenge is to explore and analyze massive data to extract relevant information needed for specific objectives (Hu, Dou, & Liu, 2014; Takaishi, Nishiyama, Kato, & Miura, 2014; Zhang, Zhou, Wang, Sun, & He, 2018).

Due to the explosive growth of information available on the Web, users confront the crucial challenge of overwhelming choices, which is known as the information overload problem (Ait Hammou, Ait Lahcen, & Mouline, 2019b; Sun, Wu, Wu, & Ye, 2019). It is difficult for users to find interesting information. This problem has increased the need for efficient information filtering technology to help users find the appropriate items like books, movies, music according to their needs (Ait Hammou & Ait Lahcen, 2017; Kaleli, 2014; Ma, Lu, Gan, & Zhao, 2016).

Recommendation systems have emerged as powerful tools to address the mentioned problem. They aim to provide users with personalized suggestions based on their past preferences and interests (Borràs, Moreno, & Valls, 2014; Lu, Wu, Mao, Wang, & Zhang, 2015; Mao, Lu, Zhang, & Zhang, 2017; Wang, Cheng, Jiang, & Lou, 2019). Examples of real-world applications include e-government, social network, e-commerce, e-learning, and so on (Ait Hammou et al., 2019b; Al-Shamri, 2014; Bobadilla, Ortega, Hernando, & Gutiérrez, 2013; Ortega, Hernando, Bobadilla, & Kang, 2016; Wu, Lu, & Zhang, 2015; Zhang et al., 2016).

Collaborative filtering (CF) is one of the most successful approaches used to build recommendation systems (Osadchiy, Poliakov, Olivier, Rowland, & Foster, 2019; Turk & Bilge, 2019; Zhang et al., 2016). In general, the CF methods are divided into two categories: model-based and memory-based CF. The model-based methods provide recommendations based on a mathematical model, while memory-based methods such as user-based and item-based CF predict the unknown ratings by aggregating the preferences of the most similar users, or items, respectively (Mazurowski, 2013; Turk & Bilge, 2019; Zhang et al., 2016; Zhang, Lu, Chen, Liu, & Ling, 2017a).

Currently, like many real-world Big data applications, the rapid increase in the number of users, items, and other information generated has created critical issues for traditional recommendation systems (Zhang et al., 2018). The need to analyze the preferences

* Corresponding author.
E-mail addresses: badr.aithamou@gmail.com (B. Ait Hammou), ayoub.aitlahcen@univ-ibntofail.ac.ma (A. Ait Lahcen), salma.mouline@um5.ac.ma (S. Mouline).

https://doi.org/10.1016/j.eswa.2019.06.046
0957-4174/

and interests of users has made it necessary to process an enormous amount of information (Hu et al., 2014; Zhang et al., 2018).

Although several recommendation techniques have proved good performance for small-scale data, they are difficult to be applied in the context of Big Data (Zhang et al., 2018). In particular, the computational cost for performing the training task can be computationally prohibitive on large-scale data, which may limit the applicability in real-world scenarios (Hu et al., 2014; Salah, Rogovschi, & Nadif, 2016; Zhang et al., 2018). Furthermore, taking into account the new preferences in the system involves performing the offline computation. This task costs much time with increasing data volume (Aggarwal, 2016; Ait Hammou & Ait Lahcen, 2017; Zhou, He, Huang, & Zhang, 2015). In addition to this, the data sparsity is another critical problem, which has a negative impact on the prediction quality (Ait Hammou et al., 2019b; Zhou et al., 2015).

Therefore, building large-scale recommendation systems requires the consideration of different issues such as coping with the data sparsity, shortening the computational time, improving the quality of predictions, and handling large-scale data efficiently (Ait Hammou, Ait Lahcen, & Mouline, 2018; Hu et al., 2014; Ma et al., 2016; Salah et al., 2016; Zhang et al., 2018).

This paper represents a continuation of our previous work (Ait Hammou et al., 2018). In particular, it presents an effective distributed predictive model for Big Data recommendation based on Apache Spark. The main objective is to address the aforementioned shortcomings.

The contributions of this work are summarized as follows.

- We propose a distributed recommendation model based on a data partitioning strategy and a novel learning process to speed up the training task, tackle data sparsity, improve the prediction quality, and handle large-scale data effectively.
- To further enhance the overall performance, we devise an approach based on distributed Matrix factorization and Random forest to provide high-quality results.
- We parallelize the proposed model using the Apache Spark platform. The data are stored in the Hadoop Distributed File System (HDFS). Every operation is performed based on the Resilient Distributed Dataset (RDD), which implies that the operations are efficient and fully scalable.
- We conduct extensive experiments on three real-world data sets. The experimental results demonstrate the effectiveness of our proposal in comparison with existing state-of-the-art methods.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 presents the preliminaries. Section 4 describes our proposal. Section 5 details the experiments. Section 6 presents the discussion. Finally, Section 7 concludes this paper.

2. Related work

With the advent of Big Data applications, the huge amount of data has brought several challenges for recommendation systems. These challenges have inspired many researchers to propose several approaches to improve the performance of recommender systems in the context of Big Data. For instance, Hsieh, Weng, and Li (2018) proposed a keyword-aware recommendation system, which is designed to handle large-scale data sets using Apache Hadoop. The main idea behind this system is to exploit the keywords extracted from the textual data of users and items, to improve the performance and alleviate the cold-start problem. Xu, Sun, Ma, and Du (2016) developed a personalized recommender system based on MapReduce. Its purpose is to help researchers and practitioners find the appropriate R&D project opportunities launched by governments and enterprises. Chen et al. (2018a) designed a Disease Diagnosis and Treatment Recommendation System (DDTRS) based on Big Data mining and Cloud Computing. Its main purpose is to recommend medical treatments based on the inspection reports of patients. Lee and Lin (2017) developed a restaurant recommender system based on lambda architecture. It is designed to handle a large amount of data by taking advantage of both batch and stream processing. While Ait Hammou et al. (2018) proposed an approximate parallel recommendation algorithm for Big Data called APRA, which is developed based on Apache Spark to process large-scale data efficiently.

However, due to the sparsity and scalability issues, Zhang et al. (2018) proposed a scalable approach called Covering Algorithm based on Quotient space Granularity analysis on Spark (CA-QGS), which is designed to suggest Web services in a Big Data environment. Hwang, Lee, Kim, Won, and Lee (2016) designed novel recommendation methods, which are based on the notion of category experts rather than neighbors' of neighborhood-based methods. The idea behind this work is that the users who have evaluated many items in a specific category are considered as knowledgeable about the category, thereby their opinions are useful for other users. Yin, Wang, and Park (2017) proposed two recommendation algorithms for Big Data. The first algorithm (CFRAT) is based on the trust in sociology. While the second one (HRAT) is based on the trust and similarity. Kupisz and Unold (2015) developed a solution for Big Data, which is tailored to speed up the parallel item-based collaborative filtering using Tanimoto coefficient. Zhao and Shang (2010) implemented User-based Collaborative Filtering algorithm to solve the scalability problem using Apache Hadoop. Jiang, Lu, Zhang, and Long (2011) developed a scaling-up item-based collaborative filtering algorithm based on MapReduce, which aims to split the costly computations into four Map-Reduce phases, in order to improve the parallel computation in each phase and minimize the communication cost. Hu et al. (2014) proposed a Clustering-based Collaborative Filtering approach for Big Data (ClubCF), which employs the clustering to divide data into manageable parts, and item-based collaborative filtering to perform the rating prediction.

Additionally, Winlaw, Hynes, Caterini, and De Sterck (2015) proposed a distributed method based on nonlinear conjugate gradient, which is devoted to speeding up the convergence of the parallel Alternating Least Squares (ALS) algorithm for Big Data Recommendation systems. Kim, Kim, and Min (2019) designed a parallel algorithm based on MapReduce called ConSimMR, which is tailored to construct the similarity matrix for improving collaborative filtering. Li and He (2017) developed an optimized item-based CF algorithm based on MapReduce for big data. Its main objective is improving the scalability and processing efficiency. Singh and Mehrotra (2018) studied the impact of Biclustering based Collaborative Filtering (BBCF) approach on the Performance of recommendation systems. Duma and Twala (2019) designed a collaborative filtering approach based on genetic algorithms and nearest neighbor artificial immune. Hernando, Bobadilla, and Ortega (2016) developed a non-negative matrix factorization based on a Bayesian probabilistic model, which aims to improve the performance of recommendation systems. Yuan, Han, Qian, Xu, and Yan (2019) proposed a recommendation approach called imputation-based SVD (ISVD), which aims to incorporate imputed data into the Singular value decomposition (SVD) model for alleviating data sparsity problem.

On the other hand, da Costa, Manzato, and Campello (2019) presented an ensemble-based co-training approach called ECoRec, which employs two or more recommendation approaches to boost the performance of the system. Ait Hammou et al. (2019b) designed a recommendation approach called FRAIPA v2, which is based on self-adaptation and multi-thresholding. It is tailored to alleviate data sparsity, reduce

Table 1
Symbol denotation.

Notation	Description
U	The set of users
M	The number of users
R_u	The set of ratings expressed by the user u
I	The set of items
N	The number of items
R_i	The set of ratings expressed for the item i
$[r_{\min}, r_{\max}]$	The rating domain of the data set
$\hat{r}_{u,i}$	The user u's predicted rating on item i
$r_{u,i}$	The user u's rating on item i
$\|.\|$	The number of elements in the set

the computational time, and improve the prediction quality. Godoy-Lorite, Guimerà, Moore, and Sales-Pardo (2016) developed a recommendation model based on expectation maximization method. It aims to infer user preferences based on explicit probabilistic hypotheses about user behavior. Papadakis, Panagiotakis, and Fragopoulou (2017) designed a Synthetic Coordinate based Recommendation system (SCoR) for generating more accurate predictions. The key idea is to place the users and items in a multi-dimensional Euclidean space using the Vivaldi synthetic coordinates algorithm. Then, the Euclidean distance between an item and a user is adopted to predict the unknown preference. Meanwhile, Tran, Lee, Liao, and Lee (2018) proposed a Regularized Multi-Embedding recommendation model (RME), which employs weighted matrix factorization, with user embedding, co-disliked item embedding, and co-liked item embedding to achieve high prediction accuracy.

The various challenges of Big Data and recommendation systems have motivated us to propose an effective distributed predictive model, which is tailored to handle large-scale data, alleviate data sparsity, reduce the computational time, and enhance the prediction quality.

3. Preliminaries

This section presents the necessary background for understanding the remainder of this paper, including the problem definition, matrix factorization and the big data frameworks Apache Hadoop and Apache Spark used to implement our proposal.

3.1. Problem definition

In this paper, we dealt with the personalized recommendation problem in the context of Big Data. Let $U = \{u_1, u_2, \ldots, u_M\}$ and $I = \{i_1, i_2, \ldots, i_N\}$ be the sets of users and items, respectively. Let M and N be the number of users and items in the system. The preferences provided by each user u are represented as a rating vector $R_u = (r_{u,i_1}, r_{u,i_2}, \ldots, r_{u,i_N})$, where $r_{u,i}$ denotes the rating expressed by the user u for the item i. All users' preferences for items are represented using the user-item rating matrix $R \in \mathbb{R}^{M \times N}$.

As each user $u \in U$ rates only a very small number of items, $r_{u,i}$ is unknown for most pairs (u, i) in the rating matrix R, which is the principal cause of data sparsity. The main objective of our work is to analyze the past preferences of users to infer the preferences for the unseen items. Table 1 presents the descriptions of notations used in the rest of this paper.

3.2. Matrix factorization

Matrix factorization (MF) is a dimensionality reduction technique, which belongs to the class of latent factor models. It has gained great popularity due to superior performance in terms of scalability and recommendation quality. The main idea behind Matrix factorization is that the items and the users can be represented by a small number of latent factors inferred from the rating matrix (Fernández-Tobías, Cantador, Tomeo, Anelli, & Di Noia, 2019; Koren, Bell, & Volinsky, 2009; Liu, 2007). Specifically, given the user-item rating matrix R and the number of latent factors $C < \min(M, N)$, the MF model learns an approximation of the matrix $R \in \mathbb{R}^{M \times N}$ as the product of two low-rank matrices $E \in \mathbb{R}^{M \times C}$ and $W \in \mathbb{R}^{C \times N}$ as follows:

$$R \approx EW$$

where each row E_u of the user-factor matrix $E \in \mathbb{R}^{M \times C}$ is a C-dimensional vector associated to a user u. Each row W_i of the item-factor matrix $W \in \mathbb{R}^{C \times N}$ is associated to an item i.

As a result, predicting an unknown preference $\hat{r}_{u,i}$ of a user u on an item i can be computed as follows:

$$\hat{r}_{u,i} = E_u W_i$$

where E_u and W_i are the latent vectors of user u and item i.

For the training task, the common approach is to minimize the following objective function:

$$\mathcal{L} = \sum_{u,i} (r_{u,i} - E_u W_i)^2 + \lambda_m (\|E_u\|^2 + \|W_i\|^2)$$

where $r_{u,i}$ is a known preference in the user-item rating matrix R, λ_m is the regularization parameter.

3.3. Big Data frameworks

Big Data has attracted a lot of attention from both academia and industry. Several frameworks for distributed computing have been developed thanks to companies like Google and Yahoo. Specifically, Google designed the MapReduce programming model for processing large-scale data (Dean & Ghemawat, 2008). Then, Apache Hadoop (Hadoop, 2019) was developed by Yahoo. It is a popular open source implementation of the MapReduce programming paradigm. It provides a distributed file system called HDFS (Hadoop Distributed File System), to store massive data, and provide highly fault-tolerant storage (Glushkova, Jovanovic, & Abelló, 2019; White, 2012).

However, due to the inefficiency of MapReduce for applications that share data across multiple steps, Apache Spark (Spark, 2019) has recently emerged to overcome the weaknesses of Hadoop (Ait Hammou et al., 2018; Galicia, Torres, Martínez-Álvarez, & Troncoso, 2018; Maillo, Ramírez, Triguero, & Herrera, 2017). Indeed, it can run programs up to 100x faster than Hadoop MapReduce if data fits in memory, or 10x faster on disk. The key part in Spark is an abstraction called the resilient distributed data set (RDD), which is an immutable and partitioned collection of elements that can be processed in a distributed way. RDD is designed to be fault tolerant, i.e., in case of node failure, Spark is able to reconstruct the lost RDD partitions thanks to the lineage information (Zaharia et al., 2012; Zaharia, Chowdhury, Franklin, Shenker, & Stoica, 2010).

4. Our proposal

In this section, we present three distributed recommendation approaches for Big Data. We denote our proposed models as DPM, DPMI, DPMF. As stated in the introduction, several factors may impact the performance of recommendation systems such as sparsity, prediction quality, and the expensive computational time. Therefore, the objective of our solution is to address the data sparsity problem, improve the prediction quality, reduce the computational time, and handle large-scale data effectively.

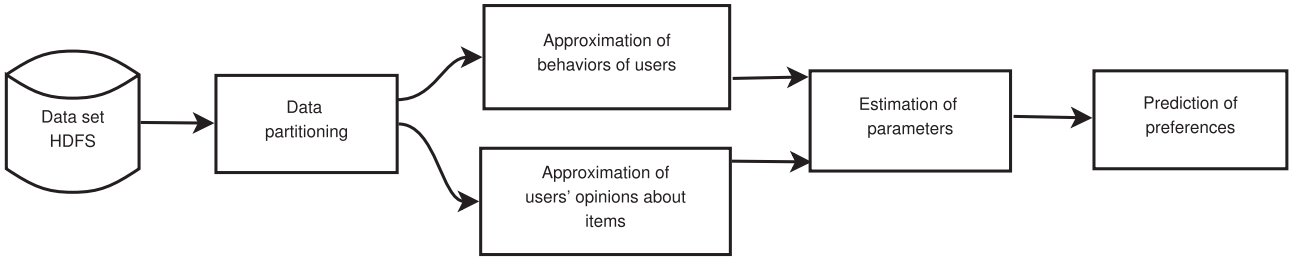

Fig. 1. Flowchart of the proposed model DPM.

4.1. DPM: A distributed predictive model for personalized recommendations

This section presents our proposed distributed predictive model for recommendation systems called DPM. It is designed based on Apache Spark. It is based on three steps. The first step is data partitioning, which aims to divide data into an optimal number of partitions. The second step is intended to train our model by employing a novel learning process for improving the recommendation quality while reducing the computational time. The third step is the prediction of the preferences. Fig. 1 depicts the flowchart of the model DPM.

4.1.1. Data partitioning

The distribution of data across nodes is of critical importance to the efficiency of the parallel and distributed computations. The principal objective of this step is to partition the training data RDD_{train} into an optimal number of partitions, which enables the proposed model to accelerate the parallel and distributed training task.

Let N_p be the set of the possible number of partitions, and $Time(RDD_{train}, n_p)$ be a function that represents the computational time required to perform the training task according to the parameter n_p. The problem can be defined as follows:

$$n_p^* = \underset{n_p}{\operatorname{argmin}}(Time(RDD_{train}, n_p)),\ \forall n_p \in N_p$$
$$s.t.\ RDD_{train} = \left(RDD_{train}^{(1)} \cup \ldots \cup RDD_{train}^{(n_p^*)}\right) \quad (1)$$

where the parameter n_p^* is the optimal number of partitions. $RDD_{train}^{(1)}$ refers to a partition. $Time(RDD_{\text{train}} n_p^*)$ represents the smallest computational time.

For example, given a training set which composed of 50 instances, and a set of two possible number of partitions $N_p = (2, 5)$. Suppose that the optimal number of partitions is $n_p^* = 5$, this means that the training set should be divided into 5 partitions, where each one contains 10 instances.

4.1.2. Training step

The proposed model can be represented as a directed acyclic graph. Each node represents a function that takes input data and produces output. The output of each node is used as input for the next node, and so on. To illustrate the sequence of steps, Fig. 2 represents the overall structure of the model DPM.

Let p_r be the probability of expressing the rating $r \in [r_{\min}, r_{\max}]$ by the users. The set of relevant probabilities can be defined as follows:

$$P = (p_{r^{(1)}}, p_{r^{(2)}}, \ldots, p_{r^{(k)}})$$
$$s.t.\ \sum_{j=1}^{k} p_{r^{(j)}} = 1 - \sum_{j=k+1}^{o} p_{r^{(j)}}$$
$$P \cap (p_{r^{(k+1)}}, \ldots, p_{r^{(o)}}) = \emptyset \quad (2)$$

where k denotes the number of probabilities adopted to reflect the opinions of users. Note that the probability of expressing the rating $r^{(1)}$ as a preference is higher than the probability of expressing $r^{(2)}$, and so on. The symbol o represents the number of possible ratings in the system. For example, $o = 5$ when the ratings are on a scale from 1 to 5.

As depicted in Fig. 2, the model maps each input instance $X \in \mathbb{R}^{|X|}$ to a relevant representation $A \in \mathbb{R}^{|A|}$ as follows:

$$A_u = T(X = R_u) = (r_{u,i_1}, \ldots, r_{u,i_{|I|}})$$
$$A_i = T(X = R_i) = (r_{u_1,i}, \ldots, r_{u_{|U|},i})$$
$$s.t.\ \forall r_{u,i} \in R_u,\quad r_{u,i} \in A_u \quad if\ \ p_{r_{u,i}} \in P$$
$$\forall r_{u,i} \in R_i,\quad r_{u,i} \in A_i \quad if\ \ p_{r_{u,i}} \in P \quad (3)$$

where X represents either the set of ratings R_u expressed by a user u, or the set of ratings R_i provided for an item i. The main idea behind the function $T(X)$ is to select only each rating $r_{u,i} \in X$ for which the condition $p_{r_{u,i}} \in P$ is true.

The resulting representations A_u and A_i are aggregated according to each user u, and item i as follows:

$$V_u = S(A_u) = \frac{\sum_{r_{u,i} \in A_u} r_{u,i}}{|R_u|}$$
$$V_i = S(A_i) = \frac{\sum_{r_{u,i} \in A_i} r_{u,i}}{|R_i|} \quad (4)$$

where $V_u \in \mathbb{R}$ generalizes the rating behavior of a user u. $V_i \in \mathbb{R}$ approximates the preferences of users about an item i. The principal goal of the Eq. (4) is to reflect optimism, neutrality, or pessimism of users. For instance, in the range [1,5], $V_{u_1} = 4$ means that the user u_1 can be considered an optimistic user because $V_{u_1} > 3$. Similarly, for an item i_3, $V_{i_3} = 2$ means that most users did not like the item i_3, which corresponds to $V_{i_3} < 3$.

Due to the different challenges of Big Data, and the sparse nature of the rating matrix. It is essential to adopt a more sophisticated mechanism to efficiently process a large amount of data and address the sparsity issue.

Therefore, for each R_u, the model enables the connections only from the unit of the user u (i.e., V_u), and the unit of each item rated by u (i.e., V_i), where $r_{u,i} \in R_u$ and $r_{u,i} \neq 0$. After that, the model decomposes the problem into a set of subproblems, where each subproblem is optimized based on a set of consecutive objective functions. Since the users in the system are independent. The model splits the problem into $|U|$ optimization subproblems as follows:

$$(\gamma_{u_1}, \beta_{u_1}, \omega_{u_1}^*) = subproblem_{u_1}(V_{u_1}, V_{i_1}, \ldots, V_{i_{|I|}}, R_{u_1})$$
$$(\gamma_{u_2}, \beta_{u_2}, \omega_{u_2}^*) = subproblem_{u_2}(V_{u_2}, V_{i_1}, \ldots, V_{i_{|I|}}, R_{u_2})$$
$$\vdots$$
$$(\gamma_{u_{|U|}}, \beta_{u_{|U|}}, \omega_{u_{|U|}}^*) = subproblem_{u_{|U|}}(V_{u_{|U|}}, V_{i_1}, \ldots, V_{i_{|I|}}, R_{u_{|U|}}) \quad (5)$$

where learning the parameters $(\gamma_u, \beta_u, \omega_u^*)$ is performed by solving a set of objective functions with respect to $subproblem_u$. Each

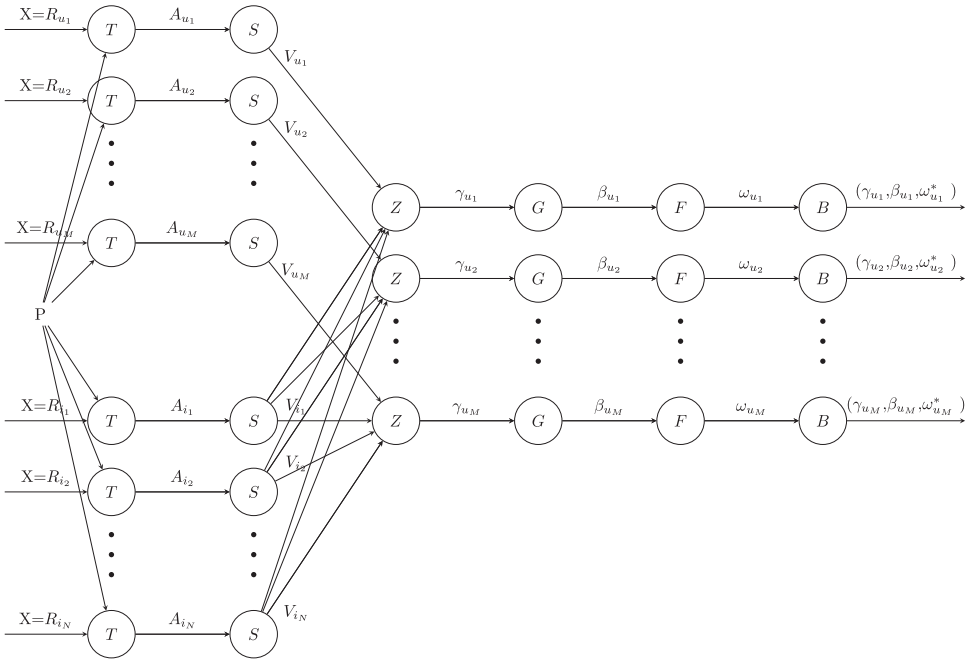

Fig. 2. DPM (Training step).

subproblem$_u$ is related to a user u. The main goal of defining independent subproblems is to carry out efficient parallel and distributed learning on large-scale data.

In general, the interaction between the units is determined based on minimizing the following objective function:

$$z = \sum_{u\in U}\sum_{i\in I}\left(r_{u,i} - \left(\frac{V_u + V_i}{\gamma_u}\right)\right)^2 \tag{6}$$

The partial derivative of z is written as follows:

$$\frac{\partial z}{\partial \gamma_u} = 2\sum_{r_{u,i}\in R_u}\left(r_{u,i} - \left(\frac{V_u + V_i}{\gamma_u}\right)\right)\left(\frac{V_u + V_i}{\gamma_u^2}\right) \tag{7}$$

According to $\frac{\partial z}{\partial \gamma_u} = 0$, the interaction between the enabled units for each user u is defined as follows:

$$\gamma_u = Z = \frac{\sum_{r_{u,i}\in R_u}(V_u + V_i)^2}{\sum_{r_{u,i}\in R_u}(r_{u,i}(V_u + V_i))} \tag{8}$$

where the parameter $\gamma_u \in \mathbb{R}$ represents the optimal value, which controls the interaction between units with respect to the user u.

In order to take into account the personalized behavior of each user u, with the impact of units. The parameter β_u is estimated directly by minimizing the following loss function:

$$g = \sum_{u\in U}\sum_{i\in I}\left(r_{u,i} - \left(\frac{2((1-\beta_u)V_u + \beta_u V_i)}{\gamma_u}\right)\right)^2 \tag{9}$$

The partial derivative of the function g with respect to the parameter β_u is determined as follows:

$$\frac{\partial g}{\partial \beta_u} = 2\sum_{r_{u,i}\in R_u}\left(r_{u,i} - \left(\frac{2((1-\beta_u)V_u + \beta_u V_i)}{\gamma_u}\right)\right)\left(\frac{-2(V_i - V_u)}{\gamma_u}\right) \tag{10}$$

At the minimizing point $\frac{\partial g}{\partial \beta_u} = 0$, the optimal value of $\beta_u \in \mathbb{R}$ is computed as follows:

$$\beta_u = G = \frac{\sum_{r_{u,i}\in R_u}\frac{2(V_i - V_u)}{\gamma_u}\left(r_{u,i} - \frac{2V_u}{\gamma_u}\right)}{\sum_{r_{u,i}\in R_u}\left(\frac{2(V_i - V_u)}{\gamma_u}\right)^2} \tag{11}$$

The main idea behind the parameter $\beta_u \in \mathbb{R}$ is to adjust the expected preferences according to the user's behavior.

On the other hand, to provide the more appropriate recommendation, it is essential to update the estimated personalized behavior of each user u (i.e., V_u), based on the optimal parameters γ_u, β_u and the past preferences. Thus, the new objective is to optimize the following loss function:

$$f = \sum_{u\in U}\sum_{i\in I}\left(r_{u,i} - \left(\frac{2((1-\beta_u)(V_u + \omega_u) + \beta_u V_i)}{\gamma_u}\right)\right)^2 \tag{12}$$

The partial derivative of the function f with respect to the parameter ω_u is defined as follows:

$$\frac{\partial f}{\partial \omega_u} = 2\sum_{r_{u,i}\in R_u}\left(r_{u,i} - \left(\frac{2((1-\beta_u)(V_u + \omega_u) + \beta_u V_i)}{\gamma_u}\right)\right)\left(\frac{-2(1-\beta_u)}{\gamma_u}\right) \tag{13}$$

According to $\frac{\partial f}{\partial \omega_u} = 0$, each parameter w_u is computed as follows:

$$\omega_u = F = \frac{\sum_{r_{u,i}\in R_u}\frac{2(1-\beta_u)}{\gamma_u}\left(r_{u,i} - \frac{2(\beta_u(V_i - V_u) + V_u)}{\gamma_u}\right)}{\sum_{r_{u,i}\in R_u}\left(\frac{2(1-\beta_u)}{\gamma_u}\right)^2} \tag{14}$$

Once the initial parameter $\omega_u \in \mathbb{R}$ is calculated according to the user u, the next goal is to learn the optimal value $\omega_u^* \in \mathbb{R}$, which is defined as follows:

$$\omega_u^* = \omega_u + B(x) \tag{15}$$

The key consideration of ω_u^* is to determine the optimal representation of the personalized behavior of the user u. The term $B(x) \in \mathbb{R}$ represents the relevant value to improve the estimated behavior.

In general, finding the parameter ω_u^* based on $B(x)$ involves considering multiple objective functions, which are described as follows:

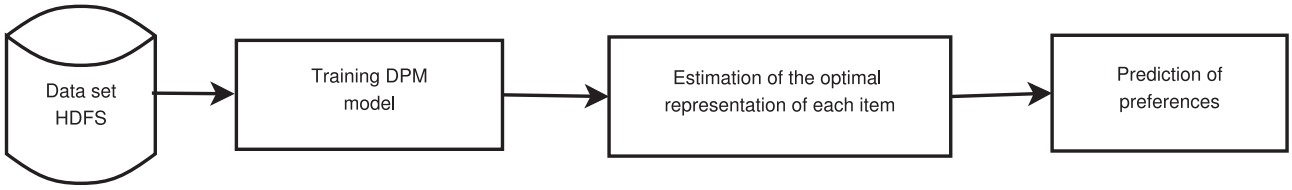

Fig. 3. Flowchart of the proposed model DPMI.

$$
\begin{aligned}
y(x) &= [y^{(1)}(x), y^{(2)}(x)] \\
y^{(1)}(x) &= \frac{\sum_{r_{u,i}\in R_u}\left| r_{u,i} - \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u V_i)}{\gamma_u}\right|}{|R_u|} \\
y^{(2)}(x) &= \sqrt{\frac{\sum_{r_{u,i}\in R_u}\left(r_{u,i} - \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u V_i)}{\gamma_u}\right)^2}{|R_u|}}
\end{aligned} \tag{16}
$$

where $(y^{(i)}(x),\ i = 1, 2)$ refer to the two objective functions.

Therefore, the estimation process is performed by considering the trade-off between the objectives. Mathematically, this task is formulated as follows:

$$
\begin{aligned}
B(x) &= \underset{x}{\operatorname{argmin}}\left(\frac{y^{(1)}(x) - min(Y^{(1)})}{max(Y^{(1)})} + \frac{y^{(2)}(x) - min(Y^{(2)})}{max(Y^{(2)})}\right), \forall x \in O \\
Y^{(1)} &= (y^{(1)}(x),\ x \in O) \\
Y^{(2)} &= (y^{(2)}(x),\ x \in O)
\end{aligned} \tag{17}
$$

where O denotes the set of possible values to estimate $B(x)$. $min(Y^{(1)})$ and $max(Y^{(1)})$ represent the minimum and maximum value of $Y^{(1)}$.

Based on the Eqs. (1)–(17), the training step of the model DPM is shown in Algorithm 1.

Algorithm 1 : DPM (Training step).

Input: RDD_{train}: Training data in HDFS, n_p^*: Optimal number of partitions, k: Dimension of the behavior

Output: RDD_{DPM}: RDD of the estimated parameters

1: Partition RDD_{train} into n_p^* partitions $RDD_{train} = (RDD_{train}^{(1)} \cup \ldots \cup RDD_{train}^{(n_p^*)})$;
2: Identify the set of relevant probabilities $P = (p_{r^{(1)}}, p_{r^{(2)}}, \ldots, p_{r^{(k)}})$ using Eq. (2);
3: Combine the set of preferences R_u provided by each user $u \in U$;
4: Combine the set of users' opinions R_i provided for each item $i \in I$;
5: **for** each user $u \in U$ **do**
6: Represent R_u as A_u based on P by Eq. (3);
7: Approximate the rating behavior V_u of the user u by Eq. (4);
8: **end for**
9: **for** each item $i \in I$ **do**
10: Represent R_i as A_i based on P by Eq. (3);
11: Approximate the users' opinions V_i about the item i by Eq. (4);
12: **end for**
13: **for** each $subproblem_u$ **do**
14: Compute γ_u based on $\frac{\partial z}{\partial \gamma_u} = 0$ using Eq. (8);
15: Compute β_u based on $\frac{\partial g}{\partial \beta_u} = 0$ using Eq. (11);
16: Compute ω_u based on $\frac{\partial f}{\partial \omega_u} = 0$ using Eq. (14);
17: Compute the optimal value ω_u^* using Eq. (15);
18: **end for**
19: $RDD_{DPM} < -((u_1, (\gamma_{u_1}, \beta_{u_1}, \omega_{u_1}^*)), \ldots, (u_M, (\gamma_{u_M}, \beta_{u_M}, \omega_{u_M}^*)))$;
20: **return** RDD_{DPM};

4.1.3. Prediction step

Given the estimated parameters the Model DPM, the predicted rating of a user u, for an item i is computed as follows:

$$
\hat{r}_{u,i} = \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u V_i)}{\gamma_u} \tag{18}
$$

Here β_u, γ_u and ω_u^* are the parameters of the trained model. In general, the main assumption behind the Eq. (18) is that, predicting an unknown preference $\hat{r}_{u,i}$ of a user u involves taking into account the optimal representation of the personalized behavior of the user u (i.e., $(V_u + \omega_u^*)$), and the opinions of most users on the item i denoted as V_i.

4.2. DPMI: A distributed predictive model based on adjusting the representation of items

This section presents our second model called DPMI, which is an improved variant of the model described in Section 4.1. It is tailored to take advantage of the optimal parameters of DPM with adjusting the estimated representation of users' opinions for each item. Fig. 3 illustrates the flowchart of the model DPMI.

4.2.1. Training step

To produce more accurate predictions, DPMI assumes that there exists a relevant representation that can reflect the opinions of users for each item. Therefore, it is necessary to take into account the optimal parameters of the model DPM and analyze the opinions of users to update the estimated representation (i.e., V_i) with respect to each item i.

As there are a large number of opinions expressed for each item, the model DPMI decomposes the problem into $|I|$ subproblems, where each $subproblem_i$ corresponds to an item $i \in I$. Then, it solves each subproblem independently based on the estimated parameters $((\gamma_u, \beta_u, \omega_u^*),\ u \in U)$.

In particular, the goal is to optimize the following objective function:

$$
h = \sum_{u\in U}\sum_{i\in I}\left(r_{u,i} - \left(\frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u(V_i+\lambda_i))}{\gamma_u}\right)\right)^2 \tag{19}
$$

The partial derivative of h with respect to the parameter λ_i is defined as follows:

$$
\frac{\partial h}{\partial \lambda_i} = 2\sum_{r_{u,i}\in R_i}\left(r_{u,i} - \left(\frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u(V_i+\lambda_i))}{\gamma_u}\right)\right)\left(\frac{-2\beta_u}{\gamma_u}\right) \tag{20}
$$

From Eq. (20), the parameter $\lambda_i \in \mathbb{R}$ is computed as follows:

$$
\lambda_i = H = \frac{\sum_{r_{u,i}\in R_u}\frac{2\beta_u}{\gamma_u}\left(r_{u,i} - \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u V_i)}{\gamma_u}\right)}{\sum_{r_{u,i}\in R_u}\left(\frac{2\beta_u}{\gamma_u}\right)^2} \tag{21}
$$

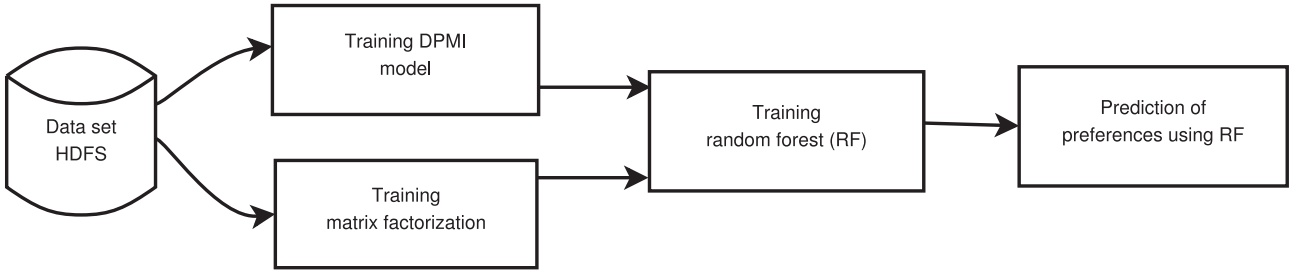

Fig. 4. Flowchart of the proposed model DPMF.

For each item i, the optimal parameter $\lambda_i^* \in \mathbb{R}$ is defined as follows:

$$\lambda_i^* = \lambda_i + D(x) \tag{22}$$

The term $D(x) \in \mathbb{R}$ denotes the estimated value, which is devoted to improving the estimation of users' opinions. The intuition behind λ_i^* is to determine the optimal representation of the opinions of users for an item i.

In order to find the most appropriate value $D(x)$, DPMI employs the following objective functions:

$$\begin{aligned} q(x) &= [q^{(1)}(x), q^{(2)}(x)] \\ q^{(1)}(x) &= \frac{\sum_{r_{u,i} \in R_i} \left| r_{u,i} - \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u(V_i+\lambda_i^*))}{\gamma_u} \right|}{|R_i|} \\ q^{(2)}(x) &= \sqrt{\frac{\sum_{r_{u,i} \in R_i} \left(r_{u,i} - \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u(V_i+\lambda_i^*))}{\gamma_u} \right)^2}{|R_i|}} \end{aligned} \tag{23}$$

where $(q^{(i)}(x),\ i = 1, 2)$ are the 2-objective functions.

By taking into account these two objectives, the solution that satisfies the best trade-off is formulated as follows:

$$\begin{aligned} D(x) &= \underset{x}{\operatorname{argmin}} \left(\frac{q^{(1)}(x) - min(Q^{(1)})}{max(Q^{(1)})} + \frac{q^{(2)}(x) - min(Q^{(2)})}{max(Q^{(2)})} \right), \forall x \in O \\ Q^{(1)} &= (q^{(1)}(x) : x \in O) \\ Q^{(2)} &= (q^{(2)}(x) : x \in O) \end{aligned} \tag{24}$$

where O represents the set of possible values to estimate $D(x)$. $min(Q^{(1)})$ and $max(Q^{(1)})$ are the minimum, maximum value of $Q^{(1)}$, respectively.

Based on the model DPM and Eqs. (19)–(24), the training step is shown in Algorithm 2.

Algorithm 2 : DPMI (Training step).

Input: RDD_{train}: Training data in HDFS, n_p^*: Optimal number of partitions, k: Dimension of the behavior
Output: RDD_{DPM}, RDD_{DPMI}: RDDs of the estimated parameters

```
RDD_DPM <- −DPM(RDD_train, n_p*, k)
for each subproblem_i do
    Compute λ_i based on ∂h/∂λ_i = 0 using Eq. (21);
    Compute the optimal value λ_i* using Eq. (22);
end for
RDD_DPMI <- −((i_1, λ*_{i_1}), ..., (i_N, λ*_{i_N}));
return RDD_DPM, RDD_DPMI;
```

4.2.2. Prediction step

Given the estimated parameters of the model DPMI, the predicted rating of user u, for an item i is defined as follows:

$$\hat{r}_{u,i} = \frac{2((1-\beta_u)(V_u+\omega_u^*)+\beta_u(V_i+\lambda_i^*))}{\gamma_u} \tag{25}$$

where β_u, γ_u, ω_u^* and λ_i^* are the parameters of the trained model. The underlying rationale behind Eq. (25) is the assumption that predicting a rating $\hat{r}_{u,i}$ can be performed by considering the optimal representation of rating behavior for a user u, with the relevant representation of users' opinions for an item i.

4.3. DPMF: A distributed predictive model with matrix factorization and random forest

This section presents our third approach called DPMF, which is designed to take advantage of the model DPMI presented in Section 4.2, with Matrix factorization and Random forest models for improving the recommendation quality. The basic idea is to represent each known rating $r_{u,i} > 0$ in the training set as features and label. Then, solving the rating prediction task as a regression problem. Fig. 4 shows the flowchart of the model DPMF.

Let $C = \{(x_j, y_j), j = 1, \ldots, |C|\}$ be the training data set, which is composed of $|C|$ instances. $|C|$ is the number of non-zero rating the user-item rating R. Each $x_j \in \mathbb{R}^{\beta+1}$ and $y_j \in \mathbb{R}$ denote the features of an instance j (i.e., the generated representation) and the label (i.e., the ground-truth rating), respectively.

Let β be the number of matrix factorization models, the main objective is to train β matrix factorization models with our second approach DPMI, then employ the learned models to generate a representation for each preference $r_{u,i}$ in the training set as follows:

$$\begin{aligned} L(r_{u,i}) &= (x_j, y_j) \\ x_j &= (E_u^{(1)} W_i^{(1)}, \ldots, E_u^{(\beta)} W_i^{(\beta)}, \hat{r}_{u,i}) \\ y_j &= r_{u,i} \end{aligned} \tag{26}$$

where the function $L(.)$ is devoted to represent the ratings using the learned models. $x_j \in \mathbb{R}^{\beta+1}$ denotes the features, y_j is the label, and $\hat{r}_{u,i}$ refers to the rating predicted using Eq. (25). $(E_u^{(1)}, W_i^{(1)})$ and $(E_u^{(2)}, W_i^{(2)})$ represent the latent factors estimated using the first and the second Matrix factorization models, respectively. The underlying assumption behind Eq. (26) is that generating a representation that characterizes each preference, and exploiting these representations by taking advantage of a machine learning algorithm may lead to better results.

Next, the rating prediction task is considered as a regression problem. It can be solved by training a machine learning model using the generated representations with the pre-defined labels.

Random forest (RF) is one of the most powerful machine learning (ML) algorithms. It is a supervised learning method, which is widely used for both regression and classification problems. Generally, the random forest is typically an ensemble of decision trees, where the training process is based on the bagging method (Breiman, 2001; Chen et al., 2016; Genuer, Poggi, Tuleau-Malot, & Villa-Vialaneix, 2017; Wyner, Olson, Bleich, & Mease, 2017). Because of its advantages over other existing machine learning approaches, we have adopted random forest to perform the

rating prediction task. Particularly, after training the random forest, the learned model is adopted to predict the unknown preferences.

5. Experiments

In this section, we will evaluate the effectiveness of our proposed methods using three real-world data sets. Section 5.1 presents the data sets. Section 5.2 describes the experimental setup. Section 5.3 presents the evaluation metrics. Section 5.4 summarizes the comparison methods. Section 5.5 details the parameter setting. Section 5.6 presents the implementation detail. Finally, Section 5.7 details the experimental results.

5.1. Data sets

In this paper, the experiments are conducted using three real-world data sets, namely Movielens 10M, Movielens 20M (Harper & Konstan, 2016), and Yelp (2019).

- MovieLens 10M[1] is a real-world data set, which was collected by GroupLens Research at the University of Minnesota. It consists of 10,000,054 ratings, given by 71,567 users for 10,681 movies of the online movie recommender service. Each user has rated at least 20 movies using a discrete score on the scale [0.5,5].
- MovieLens 20M[2] is another benchmark data set published by GroupLens research. It contains 20,000,263 ratings of approximately 27,278 movies made by 138,493 users between January 09, 1995 and March 31, 2015. Each user has rated at least 20 movies. The ratings are given on a 5-star scale with half-star increments.
- Yelp[3] is one of the most popular user review sites in the United States, a local directory service with social networking features. Customers rate the businesses, share their personal experiences and comments. Yelp data set is composed of 6,685,900 interactions expressed by 1,637,138 users for 192,609 businesses in 10 metropolitan areas. As the original data is highly sparse, we further process the data set by retaining the items and users with at least 30 preferences. The final data set contains approximately 15,139 users, 27,307 items and 1,210,783 preferences.

5.2. Experimental setup

For the experimental study, we split each data set according to the methodologies adopted in existing research works (Ait Hammou et al., 2018; Frémal & Lecron, 2017; Hernando et al., 2016; Zheng, Haihong, Song, & Song, 2016).

- Methodology 1: the data set is randomly partitioned into two parts, 80% of ratings are used as the training set, and the remaining 20% as the test set. This procedure is repeated 20 times. Then, the quality metrics of all the tests are averaged.
- Methodology 2: the data set is randomly divided into two parts, 80% of the data are used as the training set and the rest as the test set.
- Methodology 3: the data set is randomly split into 10 folds. In each run, 9 folds are utilized as the training set and the remainder as the test set. The results of the 10 runs are then averaged.

5.3. Evaluation metrics

To evaluate the performance of recommendation algorithms, we employed two well-known metrics, namely Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). They are widely used to measure the prediction accuracy (Ait Hammou, Ait Lahcen, & Mouline, 2019a; Al-Hassan, Lu, & Lu, 2015; Ar & Bostanci, 2016; Chen, Wang, Yan et al., 2018b; Da Costa, Manzato, & Campello, 2018; Hu et al., 2014; Li, Chen, Chen, & Tong, 2017; Ma et al., 2016; Mao et al., 2017; Parvin, Moradi, & Esmaeili, 2019; Wu, Chang, & Liu, 2014; Zhang, Wu, Lu, Liu, & Zhang, 2017b; Zhang et al., 2018; Zhao, Qian, & Xie, 2016). RMSE and MAE are defined as follows:

$$RMSE = \sqrt{\frac{\sum_{u\in U, i\in I}(r_{u,i} - \hat{r}_{u,i})^2}{n}} \tag{27}$$

$$MAE = \frac{\sum_{u\in U, i\in I}|r_{u,i} - \hat{r}_{u,i}|}{n} \tag{28}$$

where n is the number of ratings in the test set, $r_{u,i}$ refers to the ground-truth rating assigned by the user u to the item i, and $\hat{r}_{u,i}$ represents the predicted rating. Note that, a smaller error value indicates a better recommendation accuracy.

5.4. Comparison methods

We compared our proposal with the following state-of-the-art recommendation methods:

- **APRA** (Ait Hammou et al., 2018): it is a parallel approximate recommendation method, which adopts a random sampling technique and a special learning process to accelerate the training task and handle large-scale data.
- **ITAN** (Kupisz & Unold, 2015): Item-based collaborative filtering is a state-of-the-art recommendation method, which measures the similarities between items using the Tanimoto coefficient. Then, it estimates the preferences based on the identified neighbors.
- **UPCC** (Zhang et al., 2016): User-based collaborative filtering method calculates the similarities between users using the Pearson correlation coefficient. Then, the most similar users are employed to predict the preferences.
- **FRAIPA** (Ait Hammou & Ait Lahcen, 2017): It is a recommendation approach, which considers the rating behavior of each user and the opinions of users on each item as a set of probabilities. Then, it employs the estimated probabilities to predict the preferences.
- **CESP** (Hwang et al., 2016): This method computes the similarities between users, by considering the concept of category experts instead of all the users. It predicts the preferences by employing the similarity between each user and the category experts, with the category interest of the user.
- **CMPTF** (Zheng et al., 2016): This is a contextual modeling method based on probabilistic tensor factorization, which exploits the social relationships, ratings, item contents and contextual information to generate recommendations.
- **NMF** (Lee & Seung, 1999): It is a dimensionality reduction approach, which decomposes the rating matrix into two low-rank non-negative matrix factors. Then, it predicts the preferences using the latent factors associated with the users and items.
- **BMF** (Hernando et al., 2016): It is a non negative matrix factorization based on a Bayesian probabilistic model for personalized recommendations.
- **MLR, CM II, SW I** (Frémal & Lecron, 2017): These three strategies proposed by Frémal and Lecron (2017) are called Multiple Linear Regression, Confusion matrices II, Smart Weights I, respectively. The main idea is to combine content-based clustering with collaborative filtering. Specifically, after performing the clustering task according to item genre, a weighting strategy is used to predict the preferences.
- **PLSA-CF** (Hofmann, 2004): This is a model-based collaborative filtering approach, which is a generalization of the statistical approach called probabilistic Latent Semantic Analysis (pLSA).

[1] https://grouplens.org/datasets/movielens/10m/.
[2] https://grouplens.org/datasets/movielens/20m/.
[3] https://www.yelp.com/dataset.

- **KNN JMSD** (Bobadilla, Serradilla, & Bernal, 2010): This method identifies the neighbors using a similarity metric called JMSD. Then, it predicts the ratings by aggregating the preferences of the neighbors.
- **DPM**: This is our approach described in Section 4.1.
- **DPMI**: This is an enhanced version of our proposed approach DPM, which adjusts the estimated representation of users' opinions for each item.
- **DPMF**: This is our proposed approach presented in Section 4.3, which adopts the model DPMI with Matrix factorization and Random forest to perform the rating prediction task.

5.5. Parameter setting

For each method, there are a number of parameters to tune. In this work, several experiments were conducted to find the optimal parameters. Only the parameters that yield the best results are reported.

For MovieLens 10M and MovieLens 20M data sets, the number of nearest-neighbors of UPCC is set to 300. For FRAIPA, the parameter α used to predict the preferences with respect to each user lies in the interval [1.5, 2]. The parameter λ adopted to control the prediction error and the running time is set to 0.025. For CESP, the number of category experts is fixed at 100. For APRA, the values of the random sampling parameter ϕ used to approximate data are {1.0, 0.8, 0.6, 0.5, 0.4}. The dimension of the behavior is set to 6. The value of the parameter θ_u adopted to emphasize the predicted opinions regarding the personalized behavior of users lies in the interval $[0.1, 0.2, \ldots, 0.8, 0.9]$. While for our proposed methods DPM and DPMI, the number of possible partitions is set to $N_p = [8, 16, 24, 32]$. The number of relevant probabilities k is fixed at 6, and $O = [-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3]$. As the proposed approach named DPMF is based on DPMI with Matrix Factorization (MF) and Random Forest, the optimal parameters for DPMI are also used for DPMF. In addition, the parameter β which represents the number of MF models is fixed at 3. For each Matrix Factorization model, the rank is set to 10, the learning rate is fixed at 0.05, and the number of iterations for performing the training task is set to 15. For Random Forest, the number of trees is set to 5. Besides, it is important to note that, for more details about the optimal parameters of the other methods, readers could refer to the original papers.

On the other hand, the optimal parameters of the different methods adopted for MovieLens10M and MovieLens 20M data sets are also used for Yelp data set, except for APRA, the parameter of random sampling ϕ is set to 1, the dimension of the behavior is set to 3, and the value of the parameter θ_u lies in the interval $[0.1, 0.2, \ldots, 0.5]$. For the proposed methods DPM and DPMI, the parameters are fixed at $k = 3$, $O = [-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3]$ and $N_p = [8, 16, 24, 32]$.

5.6. Implementation detail

The experiments were conducted on a cluster consisting of two nodes: a master node and a slave node. Each node runs Ubuntu 16.04. It is equipped with a processor 2 cores (4 threads), a clock speed of 2.93 GHz, 4 GB of RAM.

The methods were implemented using the following frameworks: Apache Hadoop-2.7.4, Apache Spark 2.1.0, Apache Mahout 0.13.0 and Scala-2.11.8.

5.7. Experimental results

This section is intended to compare the performance of the proposed models described in Section 4 with the state-of-the-art recommendation methods.

For the three proposed approaches, data partitioning is an important step which aims to determine the number of partitions to accelerate the training step. Selecting the optimal number of partitions is essential for improving the performance of the proposed models. In particular, the first goal is to conduct experiments according to methodology 1, to study the impact of the number of partitions n_p on the performance in terms of computational time.

Fig. 5 and Fig. 6 show the variation of computational time for different values of n_p on MovieLens 10M and MovieLens 20M data sets. Particularly, the x-axis presents the number of partitions n_p tested $N_p = (8, 16, 24, 32)$, and the y-axis depicts the computational time in seconds.

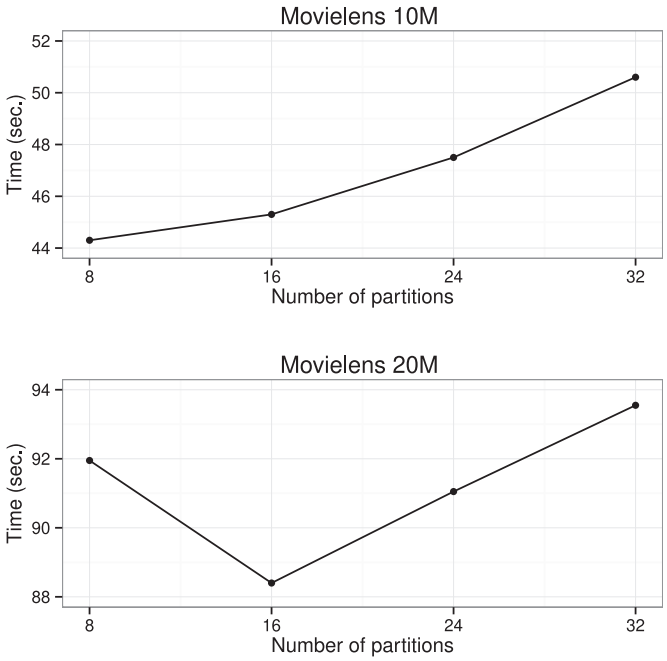

Fig. 5. The impact of the number of partitions n_p on the runtime (DPM).

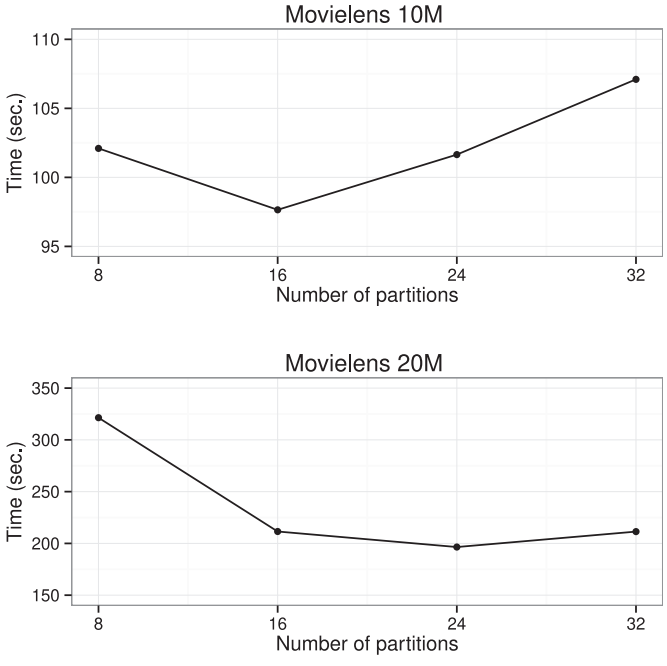

Fig. 6. The impact of the number of partitions n_p on the runtime (DPMI).

Table 2
The impact of the number of partitions n_p on the acceleration performance (methodology 1).

Method	Number of partitions	Movielens 10M	Movielens 20M
		Elapsed time (sec.)	Elapsed time (sec.)
DPM	8	**44.3**	91.95
	16	45.3	**88.40**
	24	47.5	91.05
	32	50.6	93.55
DPMI	8	102.10	321.4
	16	**97.65**	211.5
	24	101.65	**196.5**
	32	107.10	211.4

Table 3
Performance comparisons on Movielens 10M and Movielens 20M data sets (methodology 2).

Method	ϕ	Movielens 10M		Movielens 20M	
		MAE	RMSE	MAE	RMSE
UPCC (Zhang et al., 2016)	–	0.8218	1.0537	0.8261	1.0621
ITAN (Kupisz & Unold, 2015)	–	0.7308	0.9370	0.7131	0.9225
FRAIPA (Ait Hammou & Ait Lahcen, 2017)	–	0.6758	0.8892	0.6643	0.8822
CMPTF (Zheng et al., 2016)	–	–	–	0.6343	0.8182
CESP (Hwang et al., 2016)	–	0.6870	0.8988	–	–
	1.0	0.6623	0.8735	0.6519	0.8674
	0.8	0.6638	0.8754	0.6534	0.8694
APRA (Ait Hammou et al., 2018)	0.6	0.6662	0.8790	0.6557	0.8728
	0.5	0.6681	0.8818	0.6576	0.8755
	0.4	0.6711	0.8858	0.6604	0.8796
DPM	–	0.6591	0.8596	0.6490	0.8527
DPMI	–	0.6534	0.8570	0.6440	0.8502
DPMF	–	**0.6068**	**0.7928**	**0.5977**	**0.7854**

Table 4
Performance comparisons on Yelp data set (methodology 2).

Method	ϕ	Yelp	
		MAE	RMSE
UPCC (Zhang et al., 2016)	–	0.9748	1.2965
ITAN (Kupisz & Unold, 2015)	–	0.8410	1.0692
FRAIPA (Ait Hammou & Ait Lahcen, 2017)	–	0.7958	1.0442
APRA (Ait Hammou et al., 2018)	1.0	0.7922	1.0357
DPM	–	0.7949	**1.0335**
DPMI	–	**0.7887**	1.0383

From Fig. 5, it is clear that the number of partitions has a prominent impact on the performance of the model DPM. Specifically, for the MovieLens 10M data set, the elapsed time spent in the training task grows as the number of partitions increased. The model DPM achieves the optimal performance when the number of partitions is equal to 8. While, for the MovieLens 20M data set, it can be observed that, with the increase of the number of partitions n_p from 8 to 32, the computational time of DPM first decreases and reach minima when $n_p^* = 16$, and then grow when n_p continues to increase.

Fig. 6 presents how the number of partitions influences the performance of DPMI model. It is obvious that the computational time is also sensitive to the choice of the number of groups n_p. For MovieLens 10M, the model DPMI achieves the optimal performance when n_p is equal to 16. However, for MovieLens 20M, DPMI can considerably reduce the computational time when the number of partitions is approximately equal to 24.

Table 2 shows in detail, for the models DPM and DPMI, how the computational time changes according to the number of partitions. The best results are in bold.

In can be observed that DPM spends relatively less computational time than DPMI, which is consistent with the fact that DPMI is tailored to consider the optimal parameters of DPM to improve the quality of predictions.

To effectively evaluate the performance of the proposed models, it is necessary to compare them with other competitors. Table 3 reports the experimental results on MovieLens 10M and MovieLens 20M data sets, according to methodology 2. Table 4 shows the experimental results on Yelp data set using the methodology 2. The columns list the MAE and RMSE values. Comparative results show that the proposed models are better than other competitors in terms of prediction quality. Note that, the model DPMF adopts the same optimal parameters of DPMI.

Table 5 shows the experimental results on the MovieLens 10M and MovieLens 20M data sets, according to 10-fold cross-validation (i.e., methodology 3). It can be seen from the results, our proposal achieves the best results on both data sets.

On the other hand, Table 6 illustrates the comparison of MAE, RMSE, standard deviation (i.e., σ), and elapsed time according to methodology 1. The results reveal that the proposed models can

Table 5
Performance comparisons on Movielens 10M and Movielens 20M data sets (10-fold cross-validation).

Method	ϕ	Movielens 10M		Movielens 20M	
		MAE $\pm$ σ	RMSE $\pm$ σ	MAE $\pm$ σ	RMSE $\pm$ σ
MLR (Frémal & Lecron, 2017)	–	0.7106 $\pm$ 0.0006	0.8374 $\pm$ 0.0015	0.7027 $\pm$ 0.0004	0.8309 $\pm$ 0.0010
CM II (Frémal & Lecron, 2017)	–	0.7141 $\pm$ 0.0005	0.8296 $\pm$ 0.0012	0.7064 $\pm$ 0.0004	0.8216 $\pm$ 0.0010
SW I (Frémal & Lecron, 2017)	–	0.7136 $\pm$ 0.0005	0.8296 $\pm$ 0.0013	0.7054 $\pm$ 0.0004	0.8207 $\pm$ 0.0010
DPM	–	0.6592 $\pm$ 0.0004	0.8593 $\pm$ 0.0007	0.6485 $\pm$ 0.0003	0.8521 $\pm$ 0.0004
DPMI	–	0.6534 $\pm$ 0.0005	0.8565 $\pm$ 0.0007	0.6434 $\pm$ 0.0002	0.8497 $\pm$ 0.0003
DPMF	–	**0.6053** $\pm$ **0.0008**	**0.7903** $\pm$ **0.0010**	**0.5952** $\pm$ **0.0004**	**0.7819** $\pm$ **0.0005**

Table 6
Performance comparisons on Movielens 10M and Movielens 20M data sets (methodology 1).

Method	ϕ	Movielens 10M			Movielens 20M		
		MAE $\pm$ σ	RMSE $\pm$ σ	Elapsed time (sec.)	MAE $\pm$ σ	RMSE $\pm$ σ	Elapsed time (sec.)
BMF (Hernando et al., 2016)	–	0.676	–	–	–	–	–
PLSA-CF (Hofmann, 2004)	–	0.823	–	–	–	–	–
KNN JMSD (Bobadilla et al., 2010)	–	0.753	–	–	–	–	–
NMF (Lee & Seung, 1999)	–	1.356	–	–	–	–	-
FRAIPA (Ait Hammou & Ait Lahcen, 2017)	–	0.6764 $\pm$ 0.0003	0.8899 $\pm$ 0.0004	206.9	0.6644 $\pm$ 0.0002	0.8823 $\pm$ 0.0003	464.55
	1.0	0.6628 $\pm$ 0.0003	0.8740 $\pm$ 0.0003	239.2	0.6519 $\pm$ 0.0002	0.8676 $\pm$ 0.0003	489.2
	0.8	0.6643 $\pm$ 0.0003	0.8761 $\pm$ 0.0003	190.25	0.6534 $\pm$ 0.0002	0.8696 $\pm$ 0.0003	423.00
APRA (Ait Hammou et al., 2018)	0.6	0.6667 $\pm$ 0.0003	0.8794 $\pm$ 0.0004	130.65	0.6557 $\pm$ 0.0002	0.8729 $\pm$ 0.0003	237.75
	0.5	0.6687 $\pm$ 0.0003	0.8822 $\pm$ 0.0003	106.35	0.6577 $\pm$ 0.0002	0.8756 $\pm$ 0.0004	186.00
	0.4	0.6715 $\pm$ 0.0002	0.8861 $\pm$ 0.0003	84.05	0.6605 $\pm$ 0.0002	0.8795 $\pm$ 0.0003	142.15
DPM	–	0.6598 $\pm$ 0.0004	0.8602 $\pm$ 0.0005	**44.3**	0.6491 $\pm$ 0.0001	0.8530 $\pm$ 0.0002	**88.40**
DPMI	–	0.6541 $\pm$ 0.0004	0.8576 $\pm$ 0.0005	97.65	0.6440 $\pm$ 0.0001	0.8506 $\pm$ 0.0005	196.5
DPMF	–	**0.6072** $\pm$ **0.0004**	**0.7934** $\pm$ **0.0005**	499	**0.5972** $\pm$ **0.0004**	**0.7849** $\pm$ **0.0004**	1185.2

significantly outperform the state-of-the-art methods in terms of MAE, RMSE and computational time.

6. Discussion

The objective of this work is to propose a novel distributed recommendation model with matrix factorization and random forest, which adequately exploits the strengths Apache Spark and Hadoop frameworks, to improve the performance in Big Data environment.

From the results presented in Tables 2–6, we can highlight that our proposed methods are able to yield the best results in terms of prediction quality. Specifically, DPMI shows higher accuracy than DPM. This is due to the fact that adjusting the estimated opinions of users with respect to each item helps to improve the accuracy of recommendations. However, the third model DPMF based on Matrix factorization and random forest provides the best results.

In addition to the prediction quality, the computational time represents an important issue in building recommendation systems for Big Data. The obtained results show that DPM is faster than DPMI, DPMF and other competitors. The improvement in terms of the computational time is significant. In fact, DPM can save between 47.29% and 81.47% of the computational time for MovieLens 10M, and the improvement up to 81.92% for MovieLens 20M.

Furthermore, the algorithm APRA adopts the random sampling to reduce the dimensionality of data and speed up the training task. The results prove that it can significantly reduce the computational time with a slight increase in prediction error. However, thanks to the novel learning process, the proposed models are able to be efficient in terms of computational time without decreasing the quality of predictions.

The main advantages of this work are the ability to reduce the computational cost, the capacity to effectively improve the quality of predictions, handle large-scale data sets, and alleviate data sparsity. Furthermore, the proposed solution relies only on the user-item rating matrix to perform the training and prediction tasks, therefore it can be easily utilized as a recommendation system of real-world e-commerce companies. In addition, our models solve the recommendation problem as a set of independent subproblems, which allows estimating the parameters efficiently, and provides the flexibility to consider the new ratings in the system without offline computation. These facts represent a clear indication that our models can contribute to a more efficient design of recommendation systems in the context of Big Data.

7. Conclusions and future work

In this paper, a novel distributed predictive model is proposed for the personalized recommendation in the context of Big Data. It is designed based on Spark to address several challenges related to recommendation systems and Big Data.

The key point of this proposal lies in the adoption of data partitioning strategy to speed up Big Data processing. Secondly, the representation of the rating behavior for each user and the opinions of users for each item based on a more sophisticated procedure. Thirdly, the division of the optimization problem into a set of independent subproblems, where each subproblem can be resolved based on a series of consecutive objective functions. In particular, these cost functions are tailored to efficiently learn the parameters in a parallel and distributed way, and overcome the data sparsity problem. The experimental results show that the proposed solution significantly outperforms existing recommendation methods in terms of prediction error and computational time.

For future work, it would be interesting to design an efficient mechanism to further reduce the computational cost of the proposed approach DPMF. As a second future work, we plan to extend our approach by incorporating various information such as the social contexts of users, and the content-based features to improve the prediction quality. In addition, other large-scale data sets will be utilized to evaluate the effectiveness of our proposal in other recommendation scenarios like social recommendations, and group recommendations.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Credit authorship contribution statement

Badr Ait Hammou: Writing - original draft, Conceptualization, Methodology, Data curation, Formal analysis, Investigation, Writing - review & editing, Software. **Ayoub Ait Lahcen:** Supervision, Investigation, Validation. **Salma Mouline:** Supervision, Investigation, Validation.

Acknowledgments

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

Aggarwal, C. C. (2016). *Recommender systems: the textbook*. Springer.
Ait Hammou, B., & Ait Lahcen, A. (2017). Fraipa: A fast recommendation approach with improved prediction accuracy. *Expert Systems with Applications, 87*, 90–97.
Ait Hammou, B., Ait Lahcen, A., & Mouline, S. (2018). Apra: An approximate parallel recommendation algorithm for big data. *Knowledge-Based Systems, 157*, 10–19.
Ait Hammou, B., Ait Lahcen, A., & Mouline, S. (2019a). A distributed group recommendation system based on extreme gradient boosting and big data technologies. *Applied Intelligence*. doi:10.1007/s10489-019-01482-9.
Ait Hammou, B., Ait Lahcen, A., & Mouline, S. (2019b). Fraipa version 2: A fast recommendation approach based on self-adaptation and multi-thresholding. *Expert Systems with Applications, 118*, 209–216.
Al-Hassan, M., Lu, H., & Lu, J. (2015). A semantic enhanced hybrid recommendation approach: A case study of e-government tourism service recommendation system. *Decision Support Systems, 72*, 97–109.
Al-Shamri, M. Y. H. (2014). Power coefficient as a similarity measure for memory-based collaborative recommender systems. *Expert Systems with Applications, 41*(13), 5680–5688.
Ar, Y., & Bostanci, E. (2016). A genetic algorithm solution to the collaborative filtering problem. *Expert Systems with Applications, 61*, 122–128.
Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems, 46*, 109–132.
Bobadilla, J., Serradilla, F., & Bernal, J. (2010). A new collaborative filtering metric that improves the behavior of recommender systems. *Knowledge-Based Systems, 23*(6), 520–528.
Borràs, J., Moreno, A., & Valls, A. (2014). Intelligent tourism recommender systems: A survey. *Expert Systems with Applications, 41*(16), 7370–7389.
Breiman, L. (2001). Random forests. *Machine learning, 45*(1), 5–32.
Chen, J., Li, K., Rong, H., Bilal, K., Yang, N., & Li, K. (2018a). A disease diagnosis and treatment recommendation system based on big data mining and cloud computing. *Information Sciences, 435*, 124–149.
Chen, J., Li, K., Tang, Z., Bilal, K., Yu, S., Weng, C., & Li, K. (2016). A parallel random forest algorithm for big data in a spark cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems, 28*(4), 919–933.
Chen, J., Wang, H., Yan, Z., et al. (2018b). Evolutionary heterogeneous clustering for rating prediction based on user collaborative filtering. *Swarm and Evolutionary Computation, 38*, 35–41.
da Costa, A. F., Manzato, M. G., & Campello, R. J. (2019). Boosting collaborative filtering with an ensemble of co-trained recommenders. *Expert Systems with Applications, 115*, 427–441.
Da Costa, A. F., Manzato, M. G., & Campello, R. J. (2018). Corec: a co-training approach for recommender systems. In *Proceedings of the 33rd annual acm symposium on applied computing* (pp. 696–703). ACM.
Dean, J., & Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113.
Duma, M., & Twala, B. (2019). Sparseness reduction in collaborative filtering using a nearest neighbour artificial immune system with genetic algorithms. *Expert Systems with Applications, 132*, 110–125.
Fernández-Tobías, I., Cantador, I., Tomeo, P., Anelli, V. W., & Di Noia, T. (2019). Addressing the user cold start with cross-domain collaborative filtering: Exploiting item metadata in Matrix factorization. *User Modeling and User-Adapted Interaction*, 1–44.
Frémal, S., & Lecron, F. (2017). Weighting strategies for a recommender system using item clustering based on genres. *Expert Systems with Applications, 77*, 105–113.
Galicia, A., Torres, J. F., Martínez-Álvarez, F., & Troncoso, A. (2018). A novel spark-based multi-step forecasting algorithm for big data time series. *Information Sciences, 467*, 800–818.
Genuer, R., Poggi, J.-M., Tuleau-Malot, C., & Villa-Vialaneix, N. (2017). Random forests for big data. *Big Data Research, 9*, 28–46.
Glushkova, D., Jovanovic, P., & Abelló, A. (2019). Mapreduce performance model for Hadoop 2. x. *Information Systems, 79*, 32–43.
Godoy-Lorite, A., Guimerà, R., Moore, C., & Sales-Pardo, M. (2016). Accurate and scalable social recommendation using mixed-membership stochastic block models. *Proceedings of the National Academy of Sciences, 113*(50), 14207–14212.
Hadoop (2019). Apache Hadoop. https://hadoop.apache.org/. Accessed 02 june 2019.
Harper, F. M., & Konstan, J. A. (2016). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS), 5*(4), 19.
Hernando, A., Bobadilla, J., & Ortega, F. (2016). A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowledge-Based Systems, 97*, 188–202.
Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS), 22*(1), 89–115.
Hsieh, M.-Y., Weng, T.-H., & Li, K.-C. (2018). A keyword-aware recommender system using implicit feedback on Hadoop. *Journal of Parallel and Distributed Computing, 116*, 63–73.
Hu, R., Dou, W., & Liu, J. (2014). Clubcf: A clustering-based collaborative filtering approach for big data application. *IEEE Transactions on Emerging Topics in Computing, 2*(3), 302–313.
Hwang, W.-S., Lee, H.-J., Kim, S.-W., Won, Y., & Lee, M.-s. (2016). Efficient recommendation methods using category experts for a large dataset. *Information Fusion, 28*, 75–82.
Jiang, J., Lu, J., Zhang, G., & Long, G. (2011). Scaling-up item-based collaborative filtering recommendation algorithm based on Hadoop. In *Proceedings of the IEEE world congress on services (services)* (pp. 490–497). IEEE.
Kaleli, C. (2014). An entropy-based neighbor selection approach for collaborative filtering. *Knowledge-Based Systems, 56*, 273–280.
Kim, S., Kim, H., & Min, J.-K. (2019). An efficient parallel similarity matrix construction on mapreduce for collaborative filtering. *The Journal of Supercomputing, 75*(1), 123–141.
Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, (8), 30–37.
Kupisz, B., & Unold, O. (2015). Collaborative filtering recommendation algorithm based on Hadoop and spark. In *Proceedings of the IEEE international conference on industrial technology (ICIT)* (pp. 1510–1514). IEEE.
Lee, C.-H., & Lin, C.-Y. (2017). Implementation of lambda architecture: A restaurant recommender system over apache mesos. In *Proceedings of the IEEE 31st international conference on advanced information networking and applications (AINA)* (pp. 979–985). IEEE.
Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature, 401*(6755), 788.
Li, C., & He, K. (2017). CBMR: An optimized mapreduce for item-based collaborative filtering recommendation algorithm with empirical analysis. *Concurrency and Computation: Practice and Experience, 29*(10), e4092.
Li, J., Chen, C., Chen, H., & Tong, C. (2017). Towards context-aware social recommendation via individual trust. *Knowledge-Based Systems, 127*, 58–66.
Liu, B. (2007). *Web data mining: Exploring hyperlinks, contents, and usage data*. Springer Science & Business Media.
Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015). Recommender system application developments: A survey. *Decision Support Systems, 74*, 12–32.
Ma, X., Lu, H., Gan, Z., & Zhao, Q. (2016). An exploration of improving prediction accuracy by constructing a multi-type clustering based recommendation framework. *Neurocomputing, 191*, 388–397.
Maillo, J., Ramírez, S., Triguero, I., & Herrera, F. (2017). Knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data. *Knowledge-Based Systems, 117*, 3–15.
Mao, M., Lu, J., Zhang, G., & Zhang, J. (2017). Multirelational social recommendations via multigraph ranking. *IEEE Transactions on Cybernetics, 47*(12), 4049–4061. doi:10.1109/TCYB.2016.2595620.
Mazurowski, M. A. (2013). Estimating confidence of individual rating predictions in collaborative filtering recommender systems. *Expert Systems with Applications, 40*(10), 3847–3857.
Ortega, F., Hernando, A., Bobadilla, J., & Kang, J. H. (2016). Recommending items to group of users using matrix factorization based collaborative filtering. *Information Sciences, 345*, 313–324.
Osadchiy, T., Poliakov, I., Olivier, P., Rowland, M., & Foster, E. (2019). Recommender system based on pairwise association rules. *Expert Systems with Applications, 115*, 535–542.
Papadakis, H., Panagiotakis, C., & Fragopoulou, P. (2017). SCOR: A synthetic coordinate based recommender system. *Expert Systems with Applications, 79*, 8–19.
Parvin, H., Moradi, P., & Esmaeili, S. (2019). TCFACO: Trust-aware collaborative filtering method based on ant colony optimization. *Expert Systems with Applications, 118*, 152–168.
Salah, A., Rogovschi, N., & Nadif, M. (2016). A dynamic collaborative filtering system via a weighted clustering approach. *Neurocomputing, 175*, 206–215.
Singh, M., & Mehrotra, M. (2018). Impact of biclustering on the performance of biclustering based collaborative filtering. *Expert Systems with Applications, 113*, 443–456.
Spark (2019). Apache spark. https://spark.apache.org. Accessed 02 june 2019.
Sun, Z., Wu, B., Wu, Y., & Ye, Y. (2019). APL: Adversarial pairwise learning for recommender systems. *Expert Systems with Applications, 118*, 573–584.
Takaishi, D., Nishiyama, H., Kato, N., & Miura, R. (2014). Toward energy efficient big data gathering in densely distributed sensor networks. *IEEE Transactions on Emerging Topics in Computing, 2*(3), 388–397.

Tran, T., Lee, K., Liao, Y., & Lee, D. (2018). Regularizing matrix factorization with user and item embeddings for recommendation. In *Proceedings of the 27th acm international conference on information and knowledge management* (pp. 687–696). ACM.

Turk, A. M., & Bilge, A. (2019). Robustness analysis of multi-criteria collaborative filtering algorithms against shilling attacks. *Expert Systems with Applications, 115*, 386–402.

Wang, R., Cheng, H. K., Jiang, Y., & Lou, J. (2019). A novel matrix factorization model for recommendation with LOD-based semantic similarity measure. *Expert Systems with Applications, 123*, 70–81.

White, T. (2012). *Hadoop: The definitive guide*. O'Reilly Media, Inc.

Winlaw, M., Hynes, M. B., Caterini, A., & De Sterck, H. (2015). Algorithmic acceleration of parallel ALS for collaborative filtering: Speeding up distributed big data recommendation in spark. In *Proceedings of the IEEE 21st international conference on parallel and distributed systems (icpads)* (pp. 682–691). IEEE.

Wu, D., Lu, J., & Zhang, G. (2015). A fuzzy tree matching-based personalized e-learning recommender system. *IEEE Transactions on Fuzzy Systems, 23*(6), 2412–2426. doi:10.1109/TFUZZ.2015.2426201.

Wu, M.-L., Chang, C.-H., & Liu, R.-Z. (2014). Integrating content-based filtering with collaborative filtering using co-clustering with augmented matrices. *Expert Systems with Applications, 41*(6), 2754–2761.

Wyner, A. J., Olson, M., Bleich, J., & Mease, D. (2017). Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research, 18*(1), 1558–1590.

Xu, W., Sun, J., Ma, J., & Du, W. (2016). A personalized information recommendation system for r&d project opportunity finding in big data contexts. *Journal of Network and Computer Applications, 59*, 362–369.

Yelp (2019). Yelp dataset. https://www.yelp.com/dataset. Accessed 02 june 2019.

Yin, C., Wang, J., & Park, J. H. (2017). An improved recommendation algorithm for big data cloud service based on the trust in sociology. *Neurocomputing, 256*, 49–55.

Yuan, X., Han, L., Qian, S., Xu, G., & Yan, H. (2019). Singular value decomposition based recommendation using imputed data. *Knowledge-Based Systems, 163*, 485–494.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., … Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th usenix conference on networked systems design and implementation*. USENIX Association. (pp. 2–2)

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets.. *HotCloud, 10*(10-10), 95.

Zhang, F., Gong, T., Lee, V. E., Zhao, G., Rong, C., & Qu, G. (2016). Fast algorithms to evaluate collaborative filtering recommender systems. *Knowledge-Based Systems, 96*, 96–103.

Zhang, F., Lu, Y., Chen, J., Liu, S., & Ling, Z. (2017a). Robust collaborative filtering based on non-negative matrix factorization and r1-norm. *Knowledge-based systems, 118*, 177–190.

Zhang, Q., Wu, D., Lu, J., Liu, F., & Zhang, G. (2017b). A cross-domain recommender system with consistent information transfer. *Decision Support Systems, 104*, 49–63.

Zhang, Y.-w., Zhou, Y.-y., Wang, F.-t., Sun, Z., & He, Q. (2018). Service recommendation based on quotient space granularity analysis and covering algorithm on spark. *Knowledge-Based Systems, 147*, 25–35.

Zhao, G., Qian, X., & Xie, X. (2016). User-service rating prediction by exploring social users' rating behaviors. *IEEE Transactions on Multimedia, 18*(3), 496–506. doi:10.1109/TMM.2016.2515362.

Zhao, Z.-D., & Shang, M.-S. (2010). User-based collaborative-filtering recommendation algorithms on Hadoop. In *Proceedings of the third international conference on Knowledge discovery and data mining WKDD'10*. (pp. 478–481). IEEE.

Zheng, C., Haihong, E., Song, M., & Song, J. (2016). CMPTF: Contextual modeling probabilistic tensor factorization for recommender systems. *Neurocomputing, 205*, 141–151.

Zhou, X., He, J., Huang, G., & Zhang, Y. (2015). Svd-based incremental approaches for recommender systems. *Journal of Computer and System Sciences, 81*(4), 717–733.