[THESIS] BSc Econometrics & Operational Research

Basket-Sensitive Random Walk & Factorization Machine Recommendations For Grocery Shopping

Thanh Huy Huynh [415846]

Erasmus School of Economics, Erasmus University Rotterdam, The Netherlands

7th July 2019

Supervisor: Luuk van Maasakkers

Second assessor: Ilker Birbil

Abstract

While recommendation systems have been a hot topic for a long time now due to its success in business applications, it is still facing substantial challenges. As grocery shopping is most often considered as a real drudgery, many online stores provide a shopping recommendation system for their customers to facilitate this purchase process. However, there is still a large majority of people who still hesitate from doing their groceries online even though this form of shopping provides consumers with distinct advantages. Therefore, the goal of this paper is to investigate whether traditional collaborative filtering techniques are applicable in the domain of grocery shopping, and further improve its recommendations using extensive models and machine learning techniques. Hence, various CF-based models have been constructed including your traditional similarity-based collaborative filtering models, a basket-sensitive random walk model, and a basket-sensitive factorization machine. Here, we found that our basket-sensitive factorization machine comes out on top when it comes to recommending less popular items. However, due to its computational time, it remains to be a question whether this model is applicable in practical use.

The views stated in this thesis are those of the author and not necessarily those of Erasmus school of Economics or Erasmus University Rotterdam.

Contents

1 Introduction

Over the last decades modeling choices of consumers have become of greater importance in the marketing field of econometrics. The chasm between online retail and its brick-and-mortar counterpart keeps expanding in numbers, and people's shopping preferences are evolving in turn, leaving retailers with little choice but to adapt. However, this has led to online grocery shopping becoming more and more prominent, and therefore, resulted in radical adjustments within the marketing decision framework of many retailers (Kurnia and Chien, 2003).

In a recent report of Deloitte (2015), 81% of the people have said that personal recommendations, including those from social-media circles, played a primary role in their purchase decision rather than promotional advertisements. In fact, retailer-sponsored content −advertisements, sales promotions, social media, etc.− are losing out to user-generated content as the predominant factor of customer's purchase decisions (Krumm et al., 2008). Similarly, Wolfinbarger and Gilly (2001) reports that consumers feel more comfortable searching online and reading others' opinions as a first step in gathering initial information. Therefore, online shopping results in giving consumers a substantially increased sense of freedom and control as compared to shopping in the local stores.

However, as grocery shopping is most often considered as a real drudgery, many online stores provide a shopping recommendation system for their customers to facilitate this purchase process. These recommendation systems are based on data filtering algorithms that make use of advanced machine learning techniques and historical transaction data to automatically identify items that are new to the individual consumer, but are likely of interest to them. Especially now with the growing amount of information on the internet and with a significant rise in the number of users, it is becoming more prevalent for retailers to gather a large amount of data and provide them with relevant chunk of information according to their preferences and tastes. Therefore, if set up and configured properly, it can significantly boost revenues, conversions, and other important metrics of retailers as well as having positive effects on the user experience in general.

For this paper, this research is an attempt to recreate and solidify a previous study of Li et al. (2009) by using their proposed Basket-Sensitive Random Walk model. Therefore, we investigate whether traditional collaborative filtering techniques are applicable in the domain of grocery shopping. Additionally, further refinement of the study is being attempted by adopting different hybrid and model-based methods. Here, we found that the extensive factorization machine performed better than the memory-based

approach. However, a clear conclusion cannot be drawn of which model is best for practical use.

The outline of this paper is as follows: We start by reviewing the literature of similar methods and the major challenges it has in section 2. This is followed by a data description in section 3, after which we discuss the methodology and results of our models in sections 4 and 5. Finally, in section 6 we discuss the results and draw a conclusion based on our findings which is then followed by section 7 where we discuss all the limitations we came across.

2 Literature

2.1 Related Work

In the past, studies have been mainly focused on recommendation systems of leisure products without repetitive purchases. Despite all the different approaches, however, the filtering method has always remained to be a key part of the algorithm.

2.1.1 Collaborative Filtering

The item-based Collaborative Filtering (CF) has been demonstrated to be an effective framework to generate recommendations and is not without reason one of the most popular recommendation algorithms nowadays (Breese et al., 1998). Due to its simplicity and relatively good performance it is able to identify the potential preferences of a consumer for a new product solely based on the information collected from other consumers. The item-based CF is a similarity algorithm that assumes that customers are likely to accept product recommendations that are similar to what they have bought in the past. As opposed to the content-based approach in Balabanović and Shoham (1997), there is no need for the item-based CF to apply more complicated −and sometimes less reliable− content analysis techniques due to its utilization of classical distance and similarity measures. In fact, Sarwar et al. (2001) shows that the item-based CF can achieve a prediction accuracy that is comparable to −or even better than− its user-based counterpart.

2.1.2 Random Walk Model

Until now, the vast majority of studies on the random walk model have only been applied on leisure product recommendations, e.g. movies, with non-repeated purchases (Wijaya and Bressan, 2008). For instance, Bogers (2010) used the random walk algorithm to rank movies in order of importance based on the structure of the network that best represents the data, i.e. movie genres, producers, tags, actors. This allowed the authors to create a system that is able to generate movie recommendations based on both the network structure and the rating of the movies. As

for online grocery shopping, however, the product preferences are not available as explicit ratings in the shopping basket data, and therefore, it is not immediately clear how this can be applied in the field of online grocery shopping (Li et al., 2009). More importantly, the algorithm of the authors depends on the user-based approach described in Huang et al. (2004) which makes it less favorable for large-scale applications in terms of heavy computation load.

2.1.3 Matrix Factorization

Another approach which has become more popular in recent years is the model-based matrix factorization method which combines good scalability with predictive accuracy. Previous studies have been adopting this model mainly for explicit feedback data such as Netflix recommendations in Koren et al. (2009). However, it has been shown by Le et al. (2017) that it is also able to give accurate personal recommendations using grocery shopping data. This is due to its ability to incorporate "hidden" features when generating recommendations. More importantly, this method offers much more flexibility for modeling various real-life situations (Bouguettaya et al., 2017). By using latent vectors this approach is able to effectively reduce and working with fewer dimensions which makes it less computational heavy whilst retaining its performances.

2.2 Major Challenges

Even though the CF techniques have shown relatively good results in previous studies, its performance, however, is still very limited.

2.2.1 Cold-Start

As for all pairwise distance or similarity metrics, the item-based CF is unable to explore similarities between the items that have never been co-purchased but share the same neighborhoods nevertheless (Deshpande, 2004). This problem is known as the *cold-start* problem and occurs when new users or new items appear in the transaction data. Classic recommender systems like the CF assumes that the rating of each user or item can be approximated based on the ratings of similar users, even if those ratings are unavailable. For new users or items, however, there is no transaction data available for them whatsoever. As a result, simple matrix multiplication to fill this gap will not work in this case.

2.2.2 Data Sparsity

On top of the *cold-start* problem discussed previously, another major issue challenging the usefulness of the CF-based techniques is the so-called *data sparsity* problem where a high level of sparsity is present. In general, similarity measure requires some sort of overlapping values to be able to compare two sparse vectors. As a result, how-

ever, the lower the amount of overlapping values the lower the reliability of these measures will be (Grčar et al., 2005). Therefore, the CF is unable to form any reliable neighborhoods which in turn affects its performances negatively. Additionally, there will potentially be many inactive users and unrated items (see *cold-start*), and therefore, have to be omitted from the CF process.

2.2.3 Popularity Bias

Many recommender systems in the past have suffered from the popularity bias problem: popular items are always recommended regardless of whether they are related to users' preferences while less popular, niche items are being neglected (Zhao et al., 2013). However, it is likely that users already made conscious decisions regarding these popular items. Therefore, this kind of recommendation rarely leads users to additional purchases which for retailers is an important matter for obvious reasons and cannot simply be ignored (Oh et al., 2011).

2.2.4 Grey Sheep

Another well-known challenge in the CF framework is its inability to give accurate recommendations to the so-called *grey sheep* users. This is a group of users who may neither agree or disagree with the majority of the users (Zheng et al., 2017). In the field of grocery shopping these users can be seen as outliers whose purchases consist of mainly less popular, niche items, and therefore, may introduce difficulties to produce accurate collaborative recommendations.

2.2.5 Implicit Feedback

It is often the case that explicit feedback in which a user explicitly states its preference over an item is unavailable. However, due to the abundant availability of implicit data over the years, it has become more important that recommender systems are designed to be able to work with implicit feedback datasets (Hu et al., 2008). Implicit feedback such as purchase history or time spent which indirectly reflects users' behavior is typically represented by a densely filled dataset. However, this makes it harder for a recommender system to deduce a user's exact preference level. Therefore, the system has to work differently, i.e. with cruder binary representations of the probability that a user likes an item or not. Fortunately, there are various ways in order to tackle down implicit feedback with CF techniques.

3 Data

The grocery shopping data that we used in this paper originates from TaFeng, a Chinese grocery store, which contains all transaction details from November 2000 to February 2001. It consists of

all kinds of information on *age, quantity, product, product subclass* as well as the *total billed price* for each customer represented at a categorical level. All the relevant information is subsequently defined as vectors of purchase frequencies sorted by its category index. However, before the historical transaction data could be used to build a model, further adjustments have to be made first in order to reproduce the research of Li et al. (2009).

Table 1: Data Summary

	Train	Test	Full
Data Characteristics			
No of baskets	85684	11437	108496
No of unique users	22812	6869	30581
No of unique items	1093	1093	2012
Most Popular Item			
No of item *ID_100205*	14474	2544	18077
Least Popular Item			
No of item *ID_560341*	19	5	27

First, as described in Li et al. (2009), we split the training and testing data sets longitudinally over time. This allows us to tackle down the temporal nature of grocery shopping, and let us always train on earlier baskets as well as testing later ones which, in addition, avoids possible artificial dependencies between earlier and later baskets. However, in Li et al. (2009) they do not explicitly state how they clean the data. Therefore, even though we are taking table 1 in Li et al. (2009) as our baseline (see appendix A), the choices we make in this process may very likely deviate from theirs. Thus, for consistency purposes, we take the same number of unique items by selecting only the baskets containing the 1093 most popular items. This allows us to simultaneously deal with the *data sparsity* problem (see section 2.2.2). This is followed by taking the same number of training baskets in Li et al. (2009) in chronological order, and selecting the remainder as our testing set. Furthermore, all test baskets with less than four items are removed from the testing set. This is required for the leave-three-out protocol which will be discussed later in the paper. A summary of the data is shown in table 1. Additionally, for each customer the training set will be aggregated into a single basket to maximize the information content, and as a result, reducing the *data sparsity* problem even further. The testing set, however, remains to be non-aggregated to maintain the simulation of a live shopping scenario. Furthermore, all baskets purchases by customers not in the training set are filtered out from the testing set in order to make personalized recommendations. By making personalized recommendations allows us to take care of the *cold-start* problem as well as the *grey sheep* problem to a lesser extent both discussed in section 2.2. Finally, the purchase frequency will be converted to $log(frequency + 1)$ which allows us to reduce the problem of overestimating the item relevance caused by extreme high values.

4 Methodology

While recommendation systems have become a popular research topic over recent years, not many of them are applicable for online grocery shopping because of its repetitive nature. What is more, most existing recommendation techniques are based on user-ratings, and therefore require preferences to be explicitly represented as ratings. As for online grocery shopping, however, only a mere reflection of consumer's preferences is observable, i.e. through purchase history. Hence, preferences can only be conveyed implicitly in the transaction data instead (see section 2.2.5). In general, implicit feedback is not a reliable source regarding which items a customer likes or not (Hu et al., 2008). Therefore, as proposed by Li et al. (2009), an improved algorithm is built under the CF framework, which is better suited to the characteristics of grocery shopping data. This is then followed by an extension in which we evaluate several model-based methods. Afterward, various evaluation metrics are being conducted in order to evaluate and compare the performances of all the different models.

4.1 Memory-Based Approach

The memory-based approach utilizes the entire user-item database to generate recommendations based on the similarity between users or items. These systems make use of CF techniques to find a set of items or users that have a history of agreeing with the target. In this paper, we will be using the item-based CF which allows us to evaluate and filter similar items based on others' preferences.

4.1.1 Item-Based Collaborative Filtering

As previously stated, the item-based CF, in particular, has been shown to be an effective framework to produce reliable recommendations and is in general preferred over the user-based approach. Unlike its user-based counterpart, the item-based approach looks into the set of items the target user has rated and computes how similar they are to the target item we are interested in. Therefore, the item-based CF resolves many problems −rapidly changing users' preferences, large number of users, etc.− the user-based approach has. Hence, an item-to-item based filtering process has been adopted by first building a model using various similarity matrices and find the affinity between all pairs of items.

4.1.1.1 Cosine Based Similarity

The effectiveness of item-based CF greatly depends on the quality of how the similarity matrices are estimated. One of the most commonly utilized similarity measures is the cosine based similarity matrix. The cosine based similarity between items i and j is given by:

$$sim_{cos}(i,j) = \frac{R_{*,i} \cdot R_{*,j}}{|R_{*,i}||R_{*,j}|}, \quad (1)$$

where R is the n x m user-item matrix. The underlying assumption is that two items are thought of as two vectors in the m dimensional user-space, and hence, the similarity is measured by computing the cosine of the angle between these two vectors. The cosine similarity is advantageous because −unlike the Euclidean distance− it allows us to efficiently compute similarities over high-dimensional positive spaces (Aggarwal et al., 2001). In fact, due to its low complexity in which only the non-zero dimensions need to be considered, it is an excellent choice when working with sparse data.

4.1.1.2 Conditional Probability Based Similarity

Despite the cosine based similarity having many advantages, it has one important drawback: the difference in implicit feedback between different users is not taken into account (Sarwar et al., 2001). Therefore, the conditional probability based similarity is being conducted as well, as proposed by Deshpande (2004). In this case, all similarities are already normalized by default. The conditional probability based similarity is calculated as follows:

$$sim_{cp}(i,j) = \frac{\sum_{\forall q:R_{q,j}>0} R_{q,i}}{Freq(i)Freq(j)^{\alpha}}, \quad (2)$$

where R is the normalized n x m user-item matrix, $Freq(\cdot)$ the number of users that have purchased items i and j and the control variable $\alpha \in [0,1]$ used to penalize popular items based on the number of users who purchased item j.

4.1.1.3 Bipartite Network

Even though the conditional probability based similarity alleviated the problem of cosine having non-normalized similarities, this method is being calculated from arbitrary measures, and possibly infeasible as it requires numerous estimates of chances (Blok et al., 2002). Therefore, a more rational approach would be to calculate the similarities directly from a graph that is able to represent the data. In this case, a bipartite network has been adopted to describe the shopping basket data in which there are two types of nodes: consumers and items. In addition, each edge in the network represents a consumer's purchase frequency of an item. As a result, the similarities can be defined as the transition probability between each item and is expressed as follows:

$$sim_{bn}(i,j) = \sum_{k=1}^{|C|} \Pr(p_j|c_k) \Pr(c_k|p_i), \quad (3)$$

with

$$\Pr(p_j|c_k) = \frac{f(c_k,p_j)}{(\sum f(c_k,\cdot))^{\alpha}}, \quad (4)$$

$$\Pr(c_k|p_i) = \frac{f(c_k,p_i)}{(\sum f(\cdot,p_i))^{\alpha}}, \quad (5)$$

where $C = c_1, c_2, ..., c_{|C|}$ is the set of customers and $P = p_1, p_2, ..., p_{|P|}$ the set of items, $F =$

$f(1,1), f(1,2), ..., f(|C|,|P|)$ the set of purchase frequencies and $\alpha \in [0,1]$ the control factor to penalize consumers that have a large number of purchases and items that are purchased often. Since the computation is made of a single transition from the original item node to the target item node, Eq.[3] can be considered as a first-order similarity between item i and j.

The key advantage of adopting the item-based CF is that −unlike customer's preferences− ratings on a given item will generally stay the same throughout the year (Sarwar et al., 2001). This allows us to calculate the item-item based similarity matrices offline, and therefore, the real-time predictions itself require little to no computational cost. However, a major flaw within the CF framework, described as the *cold-start* problem discussed in section 2.2.1, is its inability to explore transitive associations between the items that have never been co-purchased but share the same neighborhoods nevertheless. Hence, similarity measures such as the bipartite network based similarity tend to suffer when data is sparse.

4.1.2 Basket-Sensitive Random Walk

One way to tackle down the nature of CF and its limitations discussed previously is to adopt a Basket-Sensitive Random Walk model (BSRW), proposed by Li et al. (2009). In general, the similarity matrices are too sparse to capture actual dependencies between the items. For example, an item i that has not been rated by any user who has rated item j does not necessarily imply that there are no similarities between them. In fact, these items would be found as close to each other, if there was another item t similar to both of them. Therefore, a basket-sensitive random walk model has been constructed to further explore these transitive associations by incorporating the current shopping context into the model, which can be expressed as follows:

$$R_{basket} = d \cdot P \cdot R_{basket} + (1-d) \cdot U_{basket}, \quad (6)$$

where R_{basket} is the basket-based importance score used for ranking all the items in the basket, P the transition matrix, U_{basket} the personalization vector used to bias towards the items already in the basket, m the number of items currently in the basket and $d \in (0,1)$ the damping factor. This approach is a stochastic process in which the initial state is known and the next state depends on the transition probability matrix that dictates the likelihood of jumping from item i to item j. However, real-time predictions may require heavy computational cost due to the large number of item permutations, especially for grocery shopping where the content of the basket is likely to change very often. Therefore, an approximation of R_{basket} is proposed instead, which

is given by:

$$\hat{R}_{basket} = \sum_{p_i \in basket} R_{item_i}, \tag{7}$$

$$R_{item_i} = d \cdot P \cdot R_{item_i} + (1-d) \cdot U_{item_i}, \tag{8}$$

where R_{item_i} is the item-based importance score and U_{item_i} the personalization vector which sets the i^{th} element to one and the rest of the elements to zero. In addition, R_{item_i} can be further simplified as

$$R_{item_i} = (I - dP)^{-1} \cdot (1-d)\ U_{item_i}. \tag{9}$$

This approach allows us to calculate real-time predictions more efficiently. To be more precise, the item-based importance score R_{item} can be pre-computed offline for each item, and therefore, $\hat{R}_{basket}$ can easily be obtained by summing up all the R_{item_i} in the current basket. The underlying assumption is that R_{basket} in Eq.[6] can be rewritten as follows:

$$\begin{aligned} R_{basket} &= (I - dP)^{-1} \cdot (1-d)\ U_{basket}, \\ &= (I - dP)^{-1} \cdot (1-d) \sum_{p_i \in basket} \frac{U_{item_i}}{m}, \\ &= \frac{1}{m} \sum_{p_i \in basket} R_{item_i} = \frac{\hat{R}_{basket}}{m}, \end{aligned}$$

where m is the total number of items currently in the basket, and U_{basket} can now be interpreted as a weighted combination of the elements of U_{item}. Hence, it is straightforward to see from Eq.[3] that $\hat{R}_{basket}$ is proportional to R_{basket} as they both lead to the same rank-ordered list of recommendations.

4.1.3 Personalized Recommendations

So far, the recommendation system has been solely based on item similarities without taking the purchase history into account for every consumer. Therefore, in order to make the algorithm more personalized and to further enhance the performances, different weights will be assigned to the prediction computation in section 4.1.4. As proposed by Li et al. (2009), the weighting vector is calculated as follows:

$$w_{i,j} = P(p_j|c_k) \cdot R_{item}, \tag{10}$$

where $P(p_j|c_k)$ is the consumer preference calculated in Eq.[4] and R_{item} the item-based importance score calculated in Eq.[8] which allows us to smooth the weightings and avoid zero probabilities. This transformation is necessary in order to make the recommendation systems more sensitive to the current basket contents.

4.1.4 Prediction Computation

The most important step in the CF framework is to generate the output interface in terms of prediction (Sarwar et al., 2001). For the previously discussed models, a weighted sum has been applied in order to compute the predictions. This approach allows us to make a rank-ordered list

of recommendations based on the sum of the ratings weighted by its corresponding weight factor. This can be expressed as follows:

$$P_{u,j} = \frac{\sum_{\forall j:R_{u,j}>0} s_{i,j} w_{u,j} R_{u,j}}{\sum_{\forall j:R_{u,j}>0} |s_{i,j}|} \tag{11}$$

where $s_{i,j}$ is the similarity between item i and j, R the user-item matrix and $w_{i,j}$ the corresponding weights calculated in Eq.[10]. The underlying assumption behind the weighted sum is that it tries to capture how all the active users rate the similar items. This is then followed by a weighted sum scaled by the personalized sum of the similarity terms which varies for each consumer.

4.2 Model-Based Approach

While the memory-based approach computes the rank-ordered recommendations by accessing the database directly, the model-based approach, however, generate its recommendations by creating a model based on users' ratings (Bobadilla et al., 2013). As opposed to the arithmetic operations, e.g. cosine-based similarities, the modeling process is conducted by various machine learning techniques in order to get the optimal hyperparameters. Among the many different approaches to model-based methods, we will only be discussing a few Matrix Factorization models as well as combining both memory-based and model-based approaches into a hybrid model.

4.2.1 Alternating Least Square

The main advantage of the model-based approach is that it is capable of handling the problem of sparsity and scalability better than its memory-based counterpart (Aditya et al., 2016). In this case, the Alternating Least Square based Matrix Factorization model (ALS) allows us to efficiently compute recommendation by decomposing the large user-item matrix into smaller dimensional user and item features. The underlying assumption is that the user-item matrix R can be approximated by

$$R \approx U^T \cdot V, \tag{12}$$

where U and V are the corresponding weights of each hidden feature for every item and user. Instead of using arithmetic operations, the ALS method adopts the idea of turning the non-convex optimization problem into Eq.[12] by iteratively alternating between optimizing U and fixing V, and vice versa. In this paper, the ALS model is based on Hu et al. (2008) which has shown great results when dealing with implicit data sets. First, this approach makes use of preference $p(\cdot)$ of a user u or item i in order to compute its confidence $c_{u,i}$ by calculating the number of purchased items in each basket. This can be expressed as follows:

$$c_{u,i} = 1 + \alpha r_{u,i}, \tag{13}$$

where $r_{u,i}$ the number of purchased items i of user u, and $\alpha \in (0,1)$ its corresponding linear scaling factor. Hence, we can derive the corresponding quadratic loss function as

$$\begin{aligned} Loss = \min \sum_{u,i} c_{u,i}(p_{u,i} - x_u^T y_i)^2 \\ + \quad \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2), \end{aligned} \tag{14}$$

with the following minima x_u and y_i after derivation:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u), \tag{15}$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i), \tag{16}$$

where X and Y are the randomly initialized user and item matrices respectively, C^u and C^i its corresponding confidence values, $\lambda \in (0,1)$ used to penalize overfitting, and $p(u)$ and $p(i)$ the binary preference for an item which takes value one if an item has been purchased and 0 otherwise. Therefore, alternating and iterating the two equations above results in a user and item vector which we can then use to generate recommendations with its corresponding similarity score:

$$P_{u,*} = U_i \cdot V^T, \tag{17}$$

where U_i is the user vector and V^T the transpose of our item vector after convergence.

4.2.2 Basket-Sensitive Factorization Machine

On top of the ALS model described previously, we will also be discussing the Basket-Sensitive Factorization Machine (BSFM) proposed by Le et al. (2017). This approach can be seen as a more extensive model among the model-based factorization methods and has been proven to be an extremely powerful tool in combining both regression and factorization models (Rendle, 2010). Unlike the matrix factorization models −such as our ALS which can only predict the relation between two variables− the BFSM allows us to model feature-rich datasets by including multiple higher-order interactions between variables of larger domain. As a result, the BSFM is able to estimate reliable parameters even in highly sparse data, and therefore, is an excellent choice in overcoming the sparsity problem.

In this paper, we assume that items currently in a basket share some association based on an underlying latent need −ingredients for a specific recipe, spare parts for a device, etc.−, and therefore, it is important that a recommended item is relevant to both users and items currently in the basket. Hence, the BSFM model will be constructed by incorporating various types of basket-level associations within the grocery shopping data. But in order to do so, we first transform all baskets into binary features via one-hot

encoding which allows us to efficiently apply machine learning techniques later on (He and Chua, 2017). As a result, we are given a set of binary tuples T, where each tuple t can be expressed as

$$h_t = \langle u_i, B_i, v_j, \delta \rangle \in T,$$

where u_i connotes a user holding basket B_i, and δ takes value 1 if the user purchases target item v_j and -1 otherwise. This allows us to model recommendation as a function of four types of associations:

user - target item

$$\gamma_1 = \sum_{i=1}^{N} \sum_{j=N+1}^{N+M} h_i h_j (\phi_i^T \phi_j), \tag{18}$$

basket item - target item

$$\gamma_2 = \sum_{i=N+1}^{N+M} \sum_{j=N+M+1}^{p} h_i h_j (\phi_i^T \phi_j), \tag{19}$$

basket item - basket item

$$\gamma_3 = \sum_{i=N+M+1}^{p} \sum_{j=i+1}^{p} h_i h_j (\phi_i^T \phi_j), \tag{20}$$

user - basket item

$$\gamma_4 = \sum_{i=1}^{N} \sum_{j=N+M+1}^{p} h_i h_j (\phi_i^T \phi_j), \tag{21}$$

where $\phi_i \in \mathbb{R}^K$ is a K-dimensional latent vector associated with the i^{th} component. Therefore, adding a bias term μ_0 as well as a simple regression model we can derive the following model:

$$F(h;\Theta) = \mu_0 + \sum_{i=1}^{p} \mu_i h_i + \gamma_1 + \gamma_2 + \gamma_3 + \gamma_4, \tag{22}$$

where $\gamma_1, \gamma_2, \gamma_3$ and γ_4 are calculated in Eq.[18]-Eq.[21], and $\mu_i, \phi_i \in \Theta$ are parameters to be learned. Hence, it is straightforward to see that the previous model in Eq.[22] can be further simplified into a second-order factorization machine:

$$F(h) = \mu_0 + \sum_{i=1}^{p} \mu_i h_i + \sum_{i=1}^{p} \sum_{j=i+1}^{p} h_i h_j (\phi_i^T \phi_j),$$

in which we are able to estimate the parameters in Θ by minimizing the following logistic loss function:

$$\underset{\Theta}{\operatorname{argmin}} \Big[\sum_{t \in T} -\ln(\sigma(F(h;\Theta)\delta)) + \sum_{\theta \in \Theta} \lambda_\theta \theta^2 \Big] \tag{23}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and $\lambda_\theta \in \mathbb{R}^+$ the regularization factor for θ. The underlying assumption is that all parameters in Θ should converge to a value such that $F(h;\Theta)$ is high when $\delta = 1$, and $F(h;\Theta)$ low otherwise. For this purpose, we apply the Adaptive Gradient Descent (AdaGrad) which is a modified Stochastic Gradient Descent (SGD) proposed by Duchi et al. (2011) and allows us to adapt the learning rate for sparser parameters. As shown in Dean et al. (2012) the AdaGrad had been greatly improving the robustness of the SGD by using different learning rates for every θ_i, and therefore, is well-suited for dealing with sparse datasets.

Hence, in order to learn the optimal parameters, we induce positive tuples in the form of $t = \langle u_i, B_i, v_j, 1 \rangle$ for each item v_j not in the basket after removing the item we are trying to pre-

dict. This is then followed by the same amount of negative tuples $t' = \langle u_i, B'_i, v'_j, -1 \rangle$ with basket $B'_i \neq B_i$ and target item $v'_j \neq v_j$. Here, we randomly pick a target item v'_j that has never been purchased before by the user. The same goes for basket B'_i which contains items that have never been picked by the user. As a result, however, this method can be seen as a computational heavy model as the amount of input which has to be evaluated is multiplied by over a thousand. Nevertheless, it has been shown in Le et al. (2017) that it is able to generate accurate recommendations due to it being both sensitive to the basket as well as the user. At last, once the parameters have been optimized we are able to make a rank-ordered recommendation based on the value of $F(h)$ in which v_j is preferred over v'_j when $F(u_i, B_i, v_j; \Theta) > F(u_i, B_i, v'_j; \Theta)$.

4.2.3 Hybrid Model Using ALS & BSRW

The use of the hybrid model is heavily inspired by Hu et al. (2008) as well as code and concepts from Johnson (2016) where we transform the item vector obtained from ALS into an item-based similarity framework. In order to do so, we compute the dot-product between our item vector V obtained from ALS in section 4.2.1. This can be expressed as follows:

$$sim_{ALS} = V \cdot V^T, \tag{24}$$

which we can now apply your conventional item-based CF methods on. This, followed by incorporating a BSRW model discussed in section 4.1.2 results in the hybrid model based on ALS and the BSRW model of Li et al. (2009).

4.3 Performance Measures

Once the baseline of the recommendation systems has been constructed, we can start laying the foundation for measuring the performance of the models. Due to the complexity of evaluating the performance of recommendation systems in general, it is fundamental to determine an appropriate evaluation protocol to ensure that the evaluation results are representative of live, interactive behavior. Therefore, as proposed by Li et al. (2009), we will be using multiple performance measurements: binary hit rates based on the leave-three-out protocol as well as different weighted hit rates based on leave-one-out cross-validation protocol.

4.3.1 Binary Hit Rates

Based on the previous results of Sordo-Garcia et al. (2007) the popularity-based approach was the only one that ranked the recommendation algorithms consistently with their live performance. In this evaluation protocol, the items in each test basket are being split into two segments: targets and evidence. Therefore, based

on the leave-three-out protocol, the binary hit rate is calculated as the proportion of test baskets having at least one out of three target items predicted correctly. For the popularity based protocol, $bHR(pop)$, the three least popular items are selected as the targets, whereas the remainder is chosen for the evidence which will be used during the test procedure trying to predict the target items. This can be expressed as follows:

$$bHR(pop) = \frac{HIT(x_i)}{\#total\ test\ baskets}, \tag{25}$$

where x_i is the target item, and $hit(x_i) = 1$ if x_i is predicted correctly and 0 otherwise. Therefore, this allows us to simulate a real-time scenario of marketing promotion: increasing the visibility of less popular items. In addition, in order to check for robustness, a $bHR(rnd)$ has been adopted as well where the three target items are selected randomly instead.

4.3.2 Weighted Hit Rates

Despite the popularity based protocol having a consistently high performance based on the results in Sordo-Garcia et al. (2007), only a mere subset of the items in each basket has been evaluated. Therefore, an additional performance measure will be introduced which weighs each correct prediction inversely proportional to its popularity, and more importantly, it makes use of all the items available. Based on the leave-one-out cross-validation this approach involves iteratively using a single basket item as the target and the remaining basket items as evidence. This protocol continues until every item has been used once as the target item to be predicted, and therefore, the weighted hit rate can be calculated as follows:

$$wHR(loo) = \frac{\sum_i (1 - p(x_i)) * HIT(x_i)}{\sum_i (1 - p(x_i))}, \tag{26}$$

where $p(x_i)$ is the probability based on target item i's popularity. This is then followed by taking the average score over all the test baskets. This allows us to bias the results towards performance on less popular items. Additionally, a $macroHR(loo)$ weighted hit rate was also being conducted, which can be expressed as follows:

$$macroHR(loo) = \frac{\sum_i (1 - p(x_i)) * HIT(x_i)}{\#\ items\ in\ basket}, \tag{27}$$

which take the average across all the items in the current basket instead.

5 Results

In this section, we will evaluate the performances of all the models we discussed previously. First, a prior diagnostic has been conducted by evaluating the data and estimating the optimal parameters of our random walk model. This is followed by a comprehensive discussion of our memory-based models in which we match our findings with Li et al. (2009). Afterward, as an extension, the results will be compared with the per-

formances of our model-based methods and hybrid models.

5.1 Preliminary Evaluation

Before we can start evaluating the performances of our models, preliminary analysis has to be done first in order to make accurate judgments of our findings.

5.1.1 Data Evaluation

First, by looking at the way our target items have been organized it allows us to have a better grasp of how the baskets are structured and look for possible inconsistencies. A distribution of the target items *pop* and *rnd* obtained by the leave-three-out protocol discussed in section 4.3 can be seen in figure 1 and 2 respectively. Here, it shows the number of times an item appears as the target item, and subsequently weighted by its popularity. As we expected, figure 1 shows that the target items based on *pop* are heavily skewed towards less popular items. However, it still appears that more popular items remain as the vast majority of target items in *pop*. This, followed by a moderately right-skewed distribution in figure 2, indicates that there is little to no clustering presented in our data set. Hence, we can deduce to a lesser extent that the less popular items are often paired together with the more popular items in the baskets of our testing set. This, coupled with the fact that a vast majority of our evidence set consists of only one or two items (see figure 3) may heavily undermine the performances of similarity-based CF methods in general.

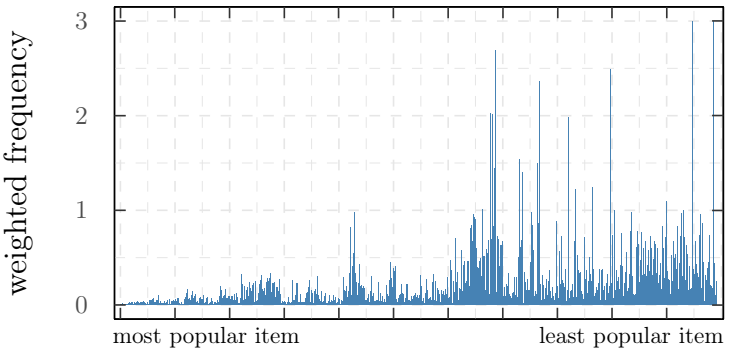


Figure 1: Distribution target items in *pop*

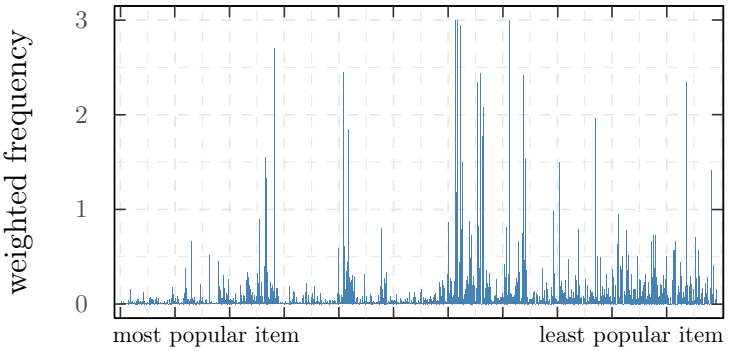


Figure 2: Distribution target items in *rnd*

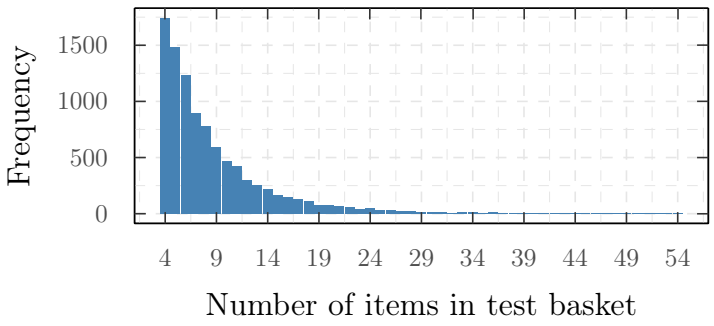


Figure 3: Distribution items in test basket

5.1.2 BSRW Parameters Evaluation

After we evaluated the data, the next step is to start tuning our hyperparameters in order to determine the optimal values of our bipartite network and BSRW model. Therefore, a grid search has been applied in which we try the find the two optimal hyperparameters: α and d both seen in Eq.[3]-Eq.[9]. Here, α is the penalty factor in order to punish larger number of purchases and items that are purchased often, and d is the damping factor corresponding to the BSRW model. Therefore, as proposed by Li et al. (2009), all possible combinations of $d \in (0, 1)$ and $\alpha \in \{0.5, 0.7, 0.9\}$ are evaluated in terms of $bHR(pop)$ and $macroHR(loo)$ shown in figure 4 and 5. Here, we can see that higher values of α and $(1-d)$ are generally preferred. Therefore, we can conclude to a lesser extent that a higher value of α tends to bias the model towards less popular items. Similarly, for higher values of $(1-d)$ in which it positively influences the importance scores of less popular items. However, unlike the results shown in Li et al. (2009) (see appendix B), the performance of $\alpha = 0.7$ and $\alpha = 0.9$ for both $bHR(pop)$ and $macroHR(loo)$ generally stays the same over all d. More importantly, for $\alpha = 0.5$ it initially has no prediction power for higher values of d whatsoever, and only tends to have better performances when d gets lower. This inconsistency can be explained

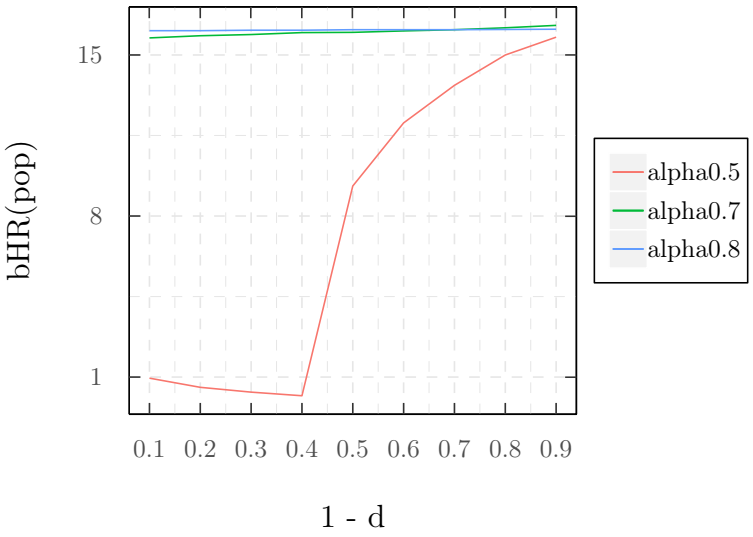


Figure 4: Impact of model parameters on bHR(pop)

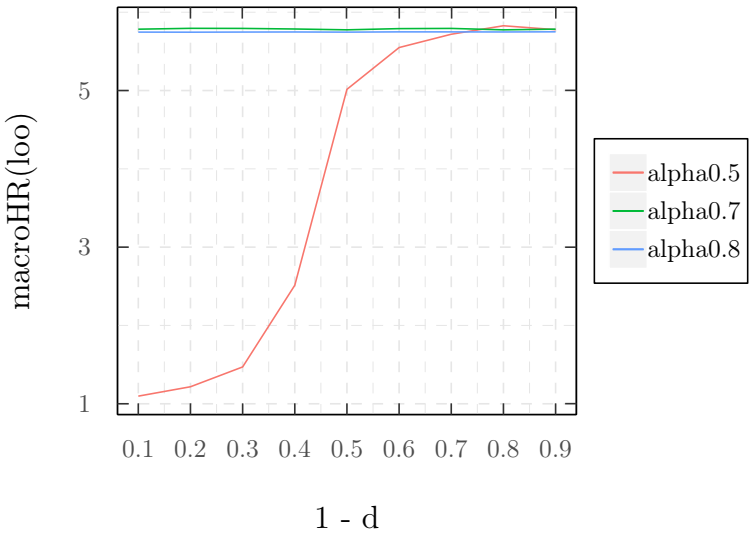


Figure 5: Impact of parameters on macroHR(loo)

due to our data not being clustered as well as our models being personalized as opposed to the non-personalized results of Li et al. (2009). Nevertheless, the outcome stays the same, and we select $\alpha = 0.7$ and $d = 0.1$ as our optimal hyperparameters.

5.2 Performance Evaluation

Once we have obtained the optimal parameters α and d for our bipartite network and $BSRW$ model, we can start to benchmark the $BSRW$ model against other item-based CF models also discussed in Li et al. (2009). The results are shown in table 2. Here, we include three types of models: the standard CF models, the BSRW based CF models, and the model-based methods which will be in the extension.

5.2.1 Comparing Memory-Based Methods

First, a simple *pop* model is being conducted whose recommendation is the most popular item not in the basket. This will be used as a guideline to benchmark our models. We can then compare the performances of the memory-based models: the $CF(\cdot)$ models and its corresponding $CF(\cdot)$ + $BSRW$ extension. First of all, table 2 shows that there is little difference in performance between the memory-based models. More importantly, all the results have significantly lower performance compared to the personalized results in Li et al. (2009) seen in table 3. As already mentioned in section 5.1.1, this can be due to our baskets not being clustered which heavily undermine the performances of our similarity-based CF models. This is further illustrated by comparing the performances of *pop* in tables 2 and 3. We can clearly see that our *pop* has little to no predictive power for less popular items as opposed to the result of *pop* in table 3

Table 2: Performance comparison

Methods	L-3-O		L-1-O
	bHR(pop)	bHR(rnd)	wHR(loo)
pop	0.43	16.80	2.65
CF(cos)	16.72	31.62	5.65
CF(cp)	16.46	30.84	5.67
CF(bn)	16.75	31.88	5.79
CF(cos) + BSRW	16.63	31.70	5.67
CF(cp) + BSRW	16.46	30.80	5.71
CF(bn) + BSRW	16.75	31.84	5.78
ALS	15.28	26.28	4.34
BSFM	20.17	19.21	2.25*
Hybrid	15.28	26.36	4.32

where the cosine, the conditional probability and the bipartite network based similarity CF models are referred to as $CF(cos), CF(cp)$ and $CF(bn)$ respectively. The BSRW based CF models are noted as BSRW for their respective CF models. As for the model-based methods, the same notations follow as discussed in the methodology section 4.2.

Table 3: Performances of Li et al. (2009)

Methods	L-3-O		L-1-O
	bHR(pop)	bHR(rnd)	wHR(loo)
pop	7.99	16.57	2.28
CF(cos)	25.66	28.09	5.27
CF(cp)	25.49	29.21	5.09
CF(bn)	26.42	28.66	4.61
CF(cos) + BSRW	25.30	28.75	5.09
CF(cp) + BSRW	25.45	28.64	5.04
CF(bn) + BSRW	26.63	30.01	5.28

where $CF(cos)$, $CF(cp)$, $CF(bn)$ and their respective BSRW model are taken from table 3 and *pop* taken from table 2 in Li et al. (2009).

based on $bHR(pop)$. Therefore, we can assume that the data set presented in Li et al. (2009) have been clustered to a lesser extent, and moderately more biased towards popular items.

Nevertheless, we can still see that the $CF(bn)$ as well as its $CF(bn) + BSRW$ extension comes out on top in all three evaluation metrics, and therefore, is consistent with the findings of Li et al. (2009). In fact, given the results of $bHR(rnd)$ and $wHR(loo)$ all results are consistently very close to −or even better than− the personalized results seen in table 3. Additionally, there is little to no difference between the traditional $CF(\cdot)$ methods and the $CF(\cdot) + BSRW$ methods. Therefore, we can say to a lesser extent that the target items are insensitive to the items currently in the basket, just as in Li et al. (2009).

5.2.2 Comparing Model-Based Methods

While the memory-based methods have shown reasonably good performance, its performance is still not on par with the more extensive model-based methods. In this section, we will be evaluating the standard ALS model, a more extensive BSFM model, and a hybrid model combining the ALS model with BSRW approach.

5.2.2.1 Evaluation ALS & Hybrid Model

Using the latent factor $K = 5$ as proposed by Hu et al. (2008), we can clearly see in table 2 that its performance is rather weak compared to the other models. The ALS model is designed for dealing with sparse data. However, due to the training set being aggregated, and therefore, losing out on information, makes the structure of our research less favorable for the ALS model. This is also the case for our hybrid model which is based on the ALS model but more biased towards the current basket items. Therefore, the result is not far different from our ALS model.

5.2.2.2 Evaluation BSFM Model

As opposed to the ALS method, the BSFM model shows outstanding performances using the same number of latent factors as the ALS model. However, a subset of 10% of the testing had to be taken instead due to its computational heavy nature. As a result, $wHR(loo)$ is not representative due to its low number of test sets, and very limited time available. This is much less the case for $bHR(pop)$ and $bHR(rnd)$ as it requires less computational resources. Despite its outstanding performances in $bHR(pop)$, its $bHR(rnd)$ fell short compared to the other models. This can be explained due to the nature of BSFM using one-hot-encoding which disregards the frequency or popularity of the items in the basket. Therefore, there is little to no difference between $bHR(pop)$ and $bHR(rnd)$, and as a result, this model has no additional prediction power for popular items.

6 Conclusion

While recommendation systems have been a hot topic for a long time now due to its success in business applications, it is still facing substantial challenges such as the *cold-start* problem and alleviating the *data sparsity*. Therefore, this paper was an attempt to replicate and further develop the research done by Li et al. (2009) in which we tried to answer the research question of whether different models under the CF framework can be applied in the field of grocery shopping. In order to do so, additional data preparation had to be done to overcome the *cold-start*, *data sparsity*, *grey sheep* as well as the *implicit feedback* problem discussed in section 2. As mentioned in section 3, we tried to filter and clean the data in a similar fashion as explained in Li et al. (2009). However, the authors did not specify exactly how they prepared the data; some assumptions had to be made. Therefore, a subset of baskets which contains only the 1093 most popular items is taken instead. This, followed by personalized recommendations, allowed us to deal with the majority of challenges described in section 2.

In addition to that, various CF models have been constructed using both memory-based and model-based methods. First, using the transaction data of TaFeng for November 2000 to February 2001 we replicate the experiment of Li et al. (2009) by evaluating the traditional CF models as well as the proposed random walk model. Here, we can see that the performances of our models differ from Li et al. (2009). As described in section 5.1.1, we found that there is little to no clustering presented within our data as opposed to Li et al. (2009). Nevertheless, based on the various leave-three-out and leave-one-out cross-validation protocols the same conclusion can be drawn. Explicitly, we concluded that the proposed basket sensitive random walk model based on a bipartite network has consistently outperformed the traditional CF models. In fact, it is shown here that incorporating a random walk model allows us to consistently improve the performances of our CF models.

Once we evaluated the memory-based methods of Li et al. (2009), we explored other approaches as well to answer the same research question. These models will be referred to as 'extensions' henceforth. The first extension of Li et al. (2009) was to adopt a traditional model-based ALS model as opposed to our memory-based methods. Unfortunately, its performances fell short compared to the other methods due to the training baskets being aggregated, and therefore, losing out on relevant information in order to obtain the optimal latent factors.

However, as for the BSFM model which can be seen as a more advanced ALS model, its performances have exceeded our expectations and outshines other models in terms of accurately

predicting less popular items. However, as seen in the results, its computational heavy load can not be overlooked, and therefore, remains to be a question whether this model can be used in practice.

As our third extension, we adopted a hybrid model by incorporating the BSRW model into our ALS model. However, this had little to no additional predictive power to our ALS model, and therefore, we were not able to improve our results with the hybrid model.

Hence, based on our results we can conclude that traditional CF models have a reasonable good performance in accurately recommending less popular items, and therefore, are applicable in the field of grocery shopping. More importantly, incorporating a BSRW model allows us to further enhance their performances. This is in line with the results as written in Li et al. (2009). Moreover, despite the great performances of our BSFM model, its computational heavy nature makes it debatable whether this advanced model can be used in practice.

7 Limitations & Further Research

Given the scope of this study and the limited amount of time available, the choice was made to use a subset for the BSFM model. This computational heavy method would have had to run for many days in order to get the results of all three evaluation metrics. This is due to the fact that the algorithm uses an extensive second-order factorization machine which requires many algebraic operations in order to calculate a single value. More importantly, AdaGrad has to consider all its positive and negative tuples for each iteration in order to learn the optimal parameters which puts a heavy workload on the CPU. This, coupled with the fact that this process has to be repeated for every individual, makes it a very time-consuming algorithm. Furthermore, we were not able to get a desirable result for the $wHR(loo)$ due to the long computational time. Therefore, further research would be to find a way to reduce the computational time of the BSFM, i.e. using the LIBfm algorithm of Rendle (2012). Another aspect in which our research fell short was not taking data clustering into account as opposed to Li et al. (2009). Therefore, further research would be to investigate various clustering techniques and evaluate to what extent this affects our results.

8 Acknowledgments

I would like to thank Luuk van Maasakkers for his aid and counsel during the writing process of this paper. The discussions with him helped me get a better grasp of the methods used.

References

Aditya, P., Budi, I., and Munajat, Q. (2016). A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for e-commerce in indonesia: A case study pt x. In *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 303–308. IEEE.

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer.

Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72.

Blok, S., Medin, D., and Osherson, D. (2002). Probability from similarity. In *AAAI Conference on Commonsense Reasoning. AAAI Press*, pages 36–42. AAAI Press.

Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, 46:109–132.

Bogers, T. (2010). Movie recommendation using random walks over the contextual graph. In *Proc. of the 2nd Intl. Workshop on Context-Aware Recommender Systems*.

Bouguettaya, A., Gao, Y., Klimenko, A., Chen, L., Zhang, X., Dzerzhinskiy, F., Jia, W., Klimenko, S. V., and Li, Q. (2017). Web information systems engineering–wise 2017.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231.

Deloitte (2015). Digital democracy survey deloitte development llc 2015.

Deshpande, Mukund, K. G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Grčar, M., Mladenič, D., Fortuna, B., and Grobelnik, M. (2005). Data sparsity issues in the collaborative filtering framework. In *International Workshop on Knowledge Discovery on the Web*, pages 58–76. Springer.

He, X. and Chua, T.-S. (2017). Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 355–364. ACM.

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *ICDM*, volume 8, pages 263–272. Citeseer.

Huang, Z., Zeng, D., and Chen, H. (2004). A link analysis approach to recommendation under sparse data. *AMCIS 2004 Proceedings*, page 239.

Johnson, C. (2016). implicitmf. https://github.com/MrChrisJohnson/implicit-mf.git.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, (8):30–37.

Krumm, J., Davies, N., and Narayanaswami, C. (2008). User-generated content. *IEEE Pervasive Computing*, 7(4):10–11.

Kurnia, S. and Chien, J. (2003). The acceptance of the online grocery shopping. In *The 16th Bled Electronic Commerce Conference, Bled, Slovenia*. Citeseer.

Le, D. T., Lauw, H. W., and Fang, Y. (2017). Basket-sensitive personalized item recommendation. IJCAI.

Li, M., Dias, M., Jarman, I., El-Deredy, W., and Lisboa, P. (2009). Grocery shopping recommendations based on basket-sensitive random walk.

Oh, J., Park, S., Yu, H., Song, M., and Park, S.-T. (2011). Novel recommendation based on personal popularity tendency. In *2011 IEEE 11th International Conference on Data Mining*, pages 507–516. IEEE.

Rendle, S. (2010). Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE.

Rendle, S. (2012). Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57.

Sarwar, B. M., Karypis, G., Konstan, J. A., Riedl, J., et al. (2001). Item-based collaborative filtering recommendation algorithms. *Www*, 1:285–295.

Sordo-Garcia, C. M., Dias, M. B., Li, M., El-Deredy, W., and Lisboa, P. J. (2007). Evaluating retail recommender systems via retrospective data: Lessons learnt from a live-intervention study. In *DMIN*, pages 197–206.

Wijaya, D. T. and Bressan, S. (2008). A random walk on the red carpet: rating movies with user reviews and pagerank. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 951–960. ACM.

Wolfinbarger, M. and Gilly, M. C. (2001). Shopping online for freedom, control, and fun. *California management review*, 43(2):34–55.

Zhao, X., Niu, Z., and Chen, W. (2013). Opinion-based collaborative filtering to solve popularity bias in recommender systems. In *International Conference on Database and Expert Systems Applications*, pages 426–433. Springer.

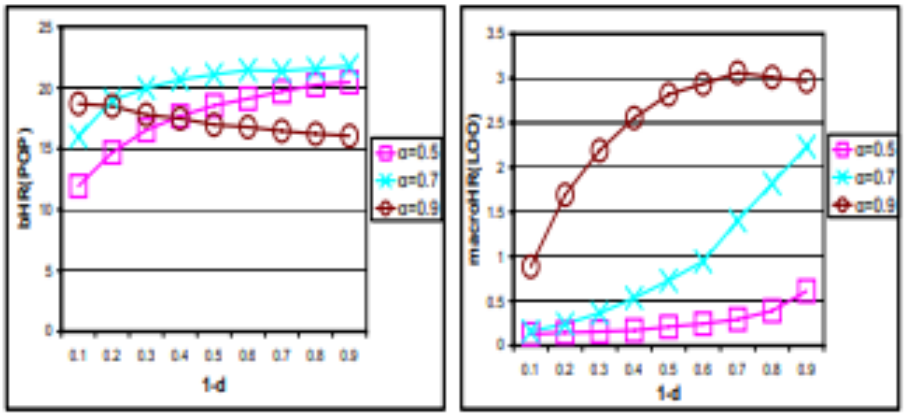
Zheng, Y., Agnani, M., and Singh, M. (2017). Identification of grey sheep users by histogram intersection in recommender systems. In *International Conference on Advanced Data Mining and Applications*, pages 148–161. Springer.

9 Appendix A

Data	*#item*	data size (*train*)	data size (*test*)	sparsity
Leshop	611	37772	28828	0.002
TaFeng	1093	85684	16756	0.57
Belgium	4951	56592	24149	0.86

Figure 6: Summary data of Li et al. (2009)

10 Appendix B



(b) TaFeng data

Figure 7: Grid search of BN and BSRW of Li et al. (2009)

11 Appendix C

[INITIALIZE] Load All Packages

```
packages <- c("dplyr", "data.table", "recommenderlab", "lsa", "Matrix", "rlist", "devtools", "reticulate", "ggplot2", "
    plotly")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}

library(dplyr)
library(data.table)
library(recommenderlab)
library(lsa)
library(Matrix)
library(rlist)
library(reticulate)
library(ggplot2)
library(plotly)

source_python("fm_gd.py")
```

[DATA] Load data & transform into user-item matrix

```
data <- read.csv(file = "ta_feng_all_months_merged.csv")

df <- data.frame(customer_ID = as.character(data$CUSTOMER_ID),
                 transaction_date = as.factor(data$TRANSACTION_DT),
                 product_subclass = as.factor(data$PRODUCT_SUBCLASS),
                 product_ID = as.factor(data$PRODUCT_ID),
                 frequencies = as.numeric(data$AMOUNT))

setDT(df)
df_matrix <- as.data.frame(dcast(df, transaction_date + customer_ID ~ product_subclass, value.var = "frequencies"))

popularity <- names(colSums(df_matrix[, c(3:ncol(df_matrix))])[order(colSums(df_matrix[, c(3:ncol(df_matrix))]),
    decreasing = TRUE)]) #get all names sorted by popularity
df_matrix <- cbind(df_matrix[, c(1:2)], df_matrix[, popularity]) #sort by popularity
df_matrix <- df_matrix[rowSums(df_matrix[, c(1096:ncol(df_matrix))]) == 0, c(1:1095)] #select the 1093 most popular items
df_matrix <- df_matrix[order(as.Date(df_matrix$transaction_date, format="%d/%m/%Y")), c(1:1095)] #chronological order

head(df_matrix)
```

[DATA] Split data into training & testing set

```
# split baskets longitudinally
df_train <- df_matrix[c(1:85684), ]
df_test <- df_matrix[c(85685:nrow(df_matrix)), ]
```

[DATA] Aggregate training set

```
# Aggregate training baskets
df_train2 <- df_train %>% group_by(customer_ID) %>% summarize_at(2:(ncol(df_train)-1), sum)
df_train2 <- df_train2 %>% mutate(count = rowSums(df_train2[, c(2:ncol(df_train2))]!=0))
df_train2 <- df_train2[!(df_train2$count < 2), c(1:(ncol(df_train2)-1))]

for (i in 2:ncol(df_train2)){
```

```
  df_train2[[i]] <- as.numeric(log(df_train2[[i]] + 1))
}

head(df_train2)
```

[DATA] Transform testing set & filter

```
# Filter out test baskets with less than 4 items
df_test2 <- df_test %>% mutate(count = rowSums(df_test[, c(3:ncol(df_test))]!=0))
df_test2 <- df_test2[!(df_test2$count < 4), c(1:(ncol(df_test2)-1))]
df_test2 <- df_test2[df_test2$customer_ID %in% df_train2$customer_ID, ]

for (i in 3:ncol(df_test2)){
  df_test2[[i]] <- as.numeric(log(df_test2[[i]] + 1))
}

head(df_test2)
```

[DATA] Prepare evidence and target items POP & RND

```
set.seed(1234)

df_test2_pop_evidence <- list()
df_test2_pop_target <- list()
df_test2_rnd_evidence <- list()
df_test2_rnd_target <- list()
pb <- txtProgressBar(min = 0, max = nrow(df_test2), style = 3)
for (i in 1:nrow(df_test2)){
  basket <- df_test2[i, ]
  basket_products <- colnames(basket[, -c(1:2)][, basket[, -c(1:2)] > 0])

  #split the already sorted baskets by popularity
  df_test2_pop_target[[i]] <- basket_products[(length(basket_products)-2):length(basket_products)]
  df_test2_pop_evidence[[i]] <- basket
  df_test2_pop_evidence[[i]][, df_test2_pop_target[[i]]] <- 0

  #create random sample
  rnd <- sample(length(basket_products), 3)
  df_test2_rnd_target[[i]] <- basket_products[rnd]
  df_test2_rnd_evidence[[i]] <- basket
  df_test2_rnd_evidence[[i]][, df_test2_rnd_target[[i]]] <- 0

  setTxtProgressBar(pb, i)
}
df_test2_pop_evidence <- as.data.frame(rbindlist(df_test2_pop_evidence))
df_test2_rnd_evidence <- as.data.frame(rbindlist(df_test2_rnd_evidence))
```

[DATA] Prepare evidence and target items in leave-one-out cross-validation protocol

```
df_test2_weighted <- list()
df_test2_weighted_customer_ID <- list()
pb <- txtProgressBar(min = 0, max = nrow(df_test2), style = 3)
for (i in 1:nrow(df_test2)){
  basket <- df_test2[i, ]
  basket_products <- colnames(basket[, -c(1:2)][, basket[, -c(1:2)] > 0])

  # loop through all basket items in which we take out a single target item for each iteration
```

```
  df_test2_weighted_temp <- list()
  for(j in 1:length(basket_products)){
    df_test2_weighted_temp$target[[j]] <- basket_products[j]
    df_test2_weighted_temp$evidence[[j]] <- basket
    df_test2_weighted_temp$evidence[[j]][, df_test2_weighted_temp$target[[j]]] <- 0
  }
  # store all evidence and target items into a list
  df_test2_weighted[[i]] <- df_test2_weighted_temp
  df_test2_weighted_customer_ID[[i]] <- as.data.frame(basket$customer_ID)

  setTxtProgressBar(pb, i)
}
df_test2_weighted_customer_ID <- cbind(index = c(1:length(df_test2_weighted_customer_ID)), customer_ID = rbindlist(df_
    test2_weighted_customer_ID))
df_test2_weighted[[i]]
```

[METHODOLOGY] SIMILARITY MATRICES

[SIM] Cosine similarity

```
R_item <- lsa::cosine(as.matrix(df_train2[, c(2:ncol(df_train2))]))
head(R_item)
```

[SIM] Conditional Probability similarity

```
R_item2 <- matrix(NA, nrow = nrow(R_item), ncol = ncol(R_item), dimnames = list(colnames(R_item), colnames(R_item)))

df_train3 <- df_train2[, c(2:ncol(df_train2))]
df_train3 <- df_train3 / rowSums(df_train3)

freq <- colSums(df_train3 != 0)
alpha <- 1

pb <- txtProgressBar(min = 0, max = nrow(R_item2), style = 3)
for (i in 1:nrow(R_item2)){
  R_item2[i, ] <- colSums(df_train3[which(df_train3[, i] > 0), ]) / (freq[i] * freq^alpha)
  setTxtProgressBar(pb, i)
}

head(R_item2)
```

[SIM] Bipartite Network Similarity

```
alpha_full <- c(0.5, 0.7, 0.9)

df_train3 <- df_train2[, c(2:ncol(df_train2))]

P_transition_list <- list()
for (n in 1:length(alpha_full)){
  alpha <- alpha_full[n]

  P_pc_top <- df_train3
  P_pc_bot <- rowSums(df_train3)^alpha
  P_pc <- P_pc_top / P_pc_bot

  P_cp_top <- df_train3
  P_cp_bot <- colSums(df_train3)^alpha
  P_cp <- matrix(NA, nrow = nrow(df_train3), ncol = ncol(df_train3), dimnames = list(row.names(df_train3), colnames(df_
      train3)))
  for (i in 1:ncol(df_train3)){
    P_cp[, i] <- as.matrix(P_cp_top[, i] / P_cp_bot[i])
  }

  pb <- txtProgressBar(min = 0, max = nrow(R_item), style = 3)
  P_transition <- matrix(NA, nrow = nrow(R_item), ncol = ncol(R_item), dimnames = list(colnames(R_item), colnames(R_item))
      )
  for (i in 1:nrow(R_item)){
    P_transition[i, ] <- colSums(P_pc * P_cp[, i])
    setTxtProgressBar(pb, i)
  }
  P_transition_list[[n]] <- t(P_transition)
}
head(t(P_transition))
```

[METHODOLOGY] Basket Sensitive Random Walk

[FUNCTIONS] Used formulas in methodology (self explanatory)

```r
personalized_weight <- function(a, n){
  df_train3 <-df_train2[, -1]
  P_pc_top <- df_train3
  P_pc_bot <- rowSums(df_train3)^a
  P_pc <- P_pc_top / P_pc_bot
  row.names(P_pc) <- df_train2$customer_ID

  if (a == 0.5){
    w <- as.matrix(P_pc) %*% as.matrix(R_bsrw_bn_0.5[[n]])
  } else if (a == 0.7){
    w <- as.matrix(P_pc) %*% as.matrix(R_bsrw_bn_0.7[[n]])
  } else {
    w <- as.matrix(P_pc) %*% as.matrix(R_bsrw_bn_0.9[[n]])
  }
  row.names(w) <- df_train2$customer_ID

  return(w)
}

prediction_weighted_sum <- function(sim, basket, weight, customer_ID, evidence){
  output <- sim %*% c(basket) / rowSums(sim) * c(weight[row.names(weight) == customer_ID, ])
  output <- output[order(output, decreasing = TRUE), ]
  output <- output[!(names(output) %in% evidence)]

  return (names(output))
}

bHR <- function(type, n, input_list, name, boolean=TRUE){
  if (type == "pop"){
    target_list <- df_test2_pop_target #pop
  } else{
    target_list <- df_test2_rnd_target #rnd
  }

  counter <- 0
  for (i in 1:n){
    if (any(input_list[[i]][1:3] %in% target_list[[i]]) == TRUE){
      counter <- counter + 1
    }
  }

  output <- counter / n
  if (boolean == TRUE){
    output <- paste(name, counter / n, sep = ": ")
  }

  return (output)
}

wHR <- function(recommendation, target, pop){
  output <- data.frame(target = recommendation, value = as.numeric(recommendation == target))
  pop <- data.frame(target = names(pop), boolean = (1 - pop))
  output <- merge(x = output, y = pop, all = TRUE)

  return ((output$value %*% output$boolean) / sum(output$boolean))
```

```
}

macroHR <- function(recommendation, target){
  output <- data.frame(target = recommendation, value = as.numeric(recommendation == target))

  return(sum(output$value) / length(output$value))
}
```

[BSRW] Calculate R-item matrix offline

```
d <- seq(0.1, 0.9, by = 0.1)

test <- length(d)

R_bsrw_bn_0.5 <- list()
R_bsrw_bn_0.7 <- list()
R_bsrw_bn_0.9 <- list()
pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){

  R_bsrw_bn_0.5[[i]] <- solve(diag(1, nrow = nrow(R_item)) - d[i]*P_transition_list[[1]]) %*% t((1-d[i])*diag(1, nrow =
      nrow(R_item)))
  R_bsrw_bn_0.7[[i]] <- solve(diag(1, nrow = nrow(R_item)) - d[i]*P_transition_list[[2]]) %*% t((1-d[i])*diag(1, nrow =
      nrow(R_item)))
  R_bsrw_bn_0.9[[i]] <- solve(diag(1, nrow = nrow(R_item)) - d[i]*P_transition_list[[3]]) %*% t((1-d[i])*diag(1, nrow =
      nrow(R_item)))

  colnames(R_bsrw_bn_0.5[[i]]) <- colnames(R_item)
  colnames(R_bsrw_bn_0.7[[i]]) <- colnames(R_item)
  colnames(R_bsrw_bn_0.9[[i]]) <- colnames(R_item)

  setTxtProgressBar(pb, i)
}
```

[BSRW] Grid search $bHR(pop)$ for optimal α and d

```
test <- nrow(df_test2)

d <- seq(0.1, 0.9, by = 0.1)

trace_0.5 <- list()
trace_0.7 <- list()
trace_0.9 <- list()

for (n in 1:length(d)){

  recommendations_bsrw_bn_0.5 <- list()
  recommendations_bsrw_bn_0.7 <- list()
  recommendations_bsrw_bn_0.9 <- list()

  w_0.5 <- personalized_weight(0.5, n)
  w_0.7 <- personalized_weight(0.7, n)
  w_0.9 <- personalized_weight(0.9, n)

  pb <- txtProgressBar(min = 0, max = test, style = 3)
  for (i in 1:test){
    basket_test <- df_test2_pop_evidence[i, -c(1:2)]
    basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]
```

```
23
24      # use the offline R_item matrix, and select the corresponding columns and sum it up
25      R_bsrw_basket_0.5 <- rowSums(cbind(R_bsrw_bn_0.5[[n]][, basket_test_products], 0))
26      R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))
27      R_bsrw_basket_0.9 <- rowSums(cbind(R_bsrw_bn_0.9[[n]][, basket_test_products], 0))
28
29      # calculate the recommendations of BSRW for the appropriate parameters
30      recommendations_bsrw_bn_0.5[[i]] <- prediction_weighted_sum(P_transition_list[[1]], R_bsrw_basket_0.5, w_0.5, df_test2
            _pop_evidence[i, ]$customer_ID, basket_test_products)
31      recommendations_bsrw_bn_0.7[[i]] <- prediction_weighted_sum(P_transition_list[[2]], R_bsrw_basket_0.7, w_0.7, df_test2
            _pop_evidence[i, ]$customer_ID, basket_test_products)
32      recommendations_bsrw_bn_0.9[[i]] <- prediction_weighted_sum(P_transition_list[[3]], R_bsrw_basket_0.9, w_0.9, df_test2
            _pop_evidence[i, ]$customer_ID, basket_test_products)
33
34      setTxtProgressBar(pb, i)
35    }
36
37    # store the results
38    trace_0.5[[n]] <- bHR("pop", test, recommendations_bsrw_bn_0.5, "", FALSE)
39    trace_0.7[[n]] <- bHR("pop", test, recommendations_bsrw_bn_0.7, "", FALSE)
40    trace_0.9[[n]] <- bHR("pop", test, recommendations_bsrw_bn_0.9, "", FALSE)
41
42  }
43  cbind(trace_0.5, trace_0.7, trace_0.9)
```

[BSRW] Grid search $macroHR(loo)$ for optimal α and d

```
1   test <- length(df_test2_weighted)
2
3   d <- seq(0.1, 0.9, by = 0.1)
4
5   trace_0.5 <- list()
6   trace_0.7 <- list()
7   trace_0.9 <- list()
8
9   for (n in 1:length(d)){
10
11    recommendations_bsrw_bn_0.5 <- list()
12    recommendations_bsrw_bn_0.7 <- list()
13    recommendations_bsrw_bn_0.9 <- list()
14
15    w_0.5 <- personalized_weight(0.5, n)
16    w_0.7 <- personalized_weight(0.7, n)
17    w_0.9 <- personalized_weight(0.9, n)
18
19    pb <- txtProgressBar(min = 0, max = test, style = 3)
20    for (i in 1:test){
21
22      hit <- list()
23      for (j in 1:length(df_test2_weighted[[i]]$target)){
24        basket_test <- df_test2_weighted[[i]]$evidence[[j]][, -c(1:2)]
25        basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]
26
27        # use the offline R_item matrix, and select the corresponding columns and sum it up
28        R_bsrw_basket_0.5 <- rowSums(cbind(R_bsrw_bn_0.5[[n]][, basket_test_products], 0))
29        R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))
30        R_bsrw_basket_0.9 <- rowSums(cbind(R_bsrw_bn_0.9[[n]][, basket_test_products], 0))
31
```

```
      # calculate the recommendations of BSRW for the appropriate parameters
      hit$bsrw_bn_0.5[j] <- prediction_weighted_sum(P_transition_list[[1]], R_bsrw_basket_0.5, w_0.5, df_test2_weighted[[i
          ]]$evidence[[j]]$customer_ID, basket_test_products)[1]
      hit$bsrw_bn_0.7[j] <- prediction_weighted_sum(P_transition_list[[2]], R_bsrw_basket_0.7, w_0.7, df_test2_weighted[[i
          ]]$evidence[[j]]$customer_ID, basket_test_products)[1]
      hit$bsrw_bn_0.9[j] <- prediction_weighted_sum(P_transition_list[[3]], R_bsrw_basket_0.9, w_0.9, df_test2_weighted[[i
          ]]$evidence[[j]]$customer_ID, basket_test_products)[1]
    }

    # get the corresponding macroHR evaluation metrics
    recommendations_bsrw_bn_0.5[i] <- macroHR(hit$bsrw_bn_0.5, df_test2_weighted[[i]]$target)
    recommendations_bsrw_bn_0.7[i] <- macroHR(hit$bsrw_bn_0.7, df_test2_weighted[[i]]$target)
    recommendations_bsrw_bn_0.9[i] <- macroHR(hit$bsrw_bn_0.9, df_test2_weighted[[i]]$target)

    setTxtProgressBar(pb, i)
  }

  # store the results
  trace_0.5[[n]] <- sum(unlist(recommendations_bsrw_bn_0.5)) / length(recommendations_bsrw_bn_0.5)
  trace_0.7[[n]] <- sum(unlist(recommendations_bsrw_bn_0.7)) / length(recommendations_bsrw_bn_0.7)
  trace_0.9[[n]] <- sum(unlist(recommendations_bsrw_bn_0.9)) / length(recommendations_bsrw_bn_0.9)
}
cbind(trace_0.5, trace_0.7, trace_0.9)
```

[RESULTS] Results of replication part

[OUTPUT] Popularity based hit rates - $bHR(pop)$

```
d <- seq(0.1, 0.9, by = 0.1)
test <- nrow(df_test2)
n <- 1

w <- personalized_weight(0.7, n)

recommendations_pop <- list()
recommendations_cos <- list()
recommendations_cp <- list()
recommendations_bn_0.7 <- list()
recommendations_bsrw_cos <- list()
recommendations_bsrw_cp <- list()
recommendations_bsrw_bn_0.7 <- list()

pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  basket_test <- df_test2_rnd_evidence[i, -c(1:2)]
  basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]

  # calculate recommendations of pop
  recommendations_pop[[i]] <- colnames(basket_test[, !(colnames(basket_test) %in% basket_test_products)])

  # calculate recommendations of traditional CF models
  recommendations_cos[[i]] <- prediction_weighted_sum(t(R_item), t(basket_test), w, df_test2_pop_evidence[i, ]$customer_ID
      , basket_test_products)
  recommendations_cp[[i]] <- prediction_weighted_sum(t(R_item2), t(basket_test), w, df_test2_pop_evidence[i, ]$customer_ID
      , basket_test_products)
  recommendations_bn_0.7[[i]] <- prediction_weighted_sum(P_transition_list[[2]], t(basket_test), w, df_test2_pop_evidence[
      i, ]$customer_ID, basket_test_products)

  # calculate recommendations of respective BSRW models
  R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))

  recommendations_bsrw_cos[[i]] <- prediction_weighted_sum(t(R_item), R_bsrw_basket_0.7, w, df_test2_pop_evidence[i, ]$
      customer_ID, basket_test_products)
  recommendations_bsrw_cp[[i]] <- prediction_weighted_sum(t(R_item2), R_bsrw_basket_0.7, w, df_test2_pop_evidence[i, ]$
      customer_ID, basket_test_products)
  recommendations_bsrw_bn_0.7[[i]] <- prediction_weighted_sum(P_transition_list[[2]], R_bsrw_basket_0.7, w, df_test2_pop_
      evidence[i, ]$customer_ID, basket_test_products)

  setTxtProgressBar(pb, i)
}

# print results
bHR("pop", test, recommendations_pop, "pop")
bHR("pop", test, recommendations_cos, "cos")
bHR("pop", test, recommendations_cp, "cp")
bHR("pop", test, recommendations_bn_0.7, "bn")
bHR("pop", test, recommendations_bsrw_cos, "cos_bsrw")
bHR("pop", test, recommendations_bsrw_cp, "cp_bsrw")
bHR("pop", test, recommendations_bsrw_bn_0.7, "bn_bsrw")
```

[OUTPUT] Random based hit rates - $bHR(rnd)$

```
d <- seq(0.1, 0.9, by = 0.1)
```

```
test <- nrow(df_test2)
n <- 1

w <- personalized_weight(0.7, n)

recommendations_rnd <- list()
recommendations_cos <- list()
recommendations_cp <- list()
recommendations_bn_0.7 <- list()
recommendations_bsrw_cos <- list()
recommendations_bsrw_cp <- list()
recommendations_bsrw_bn_0.7 <- list()

pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  basket_test <- df_test2_rnd_evidence[i, -c(1:2)]
  basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]

  # calculate recommendations of pop
  recommendations_rnd[[i]] <- colnames(basket_test[, !(colnames(basket_test) %in% basket_test_products)])

  # calculate recommendations of traditional CF models
  recommendations_cos[[i]] <- prediction_weighted_sum(t(R_item), t(basket_test), w, df_test2_rnd_evidence[i, ]$customer_ID
      , basket_test_products)
  recommendations_cp[[i]] <- prediction_weighted_sum(t(R_item2), t(basket_test), w, df_test2_rnd_evidence[i, ]$customer_ID
      , basket_test_products)
  recommendations_bn_0.7[[i]] <- prediction_weighted_sum(P_transition_list[[2]], t(basket_test), w, df_test2_rnd_evidence[
      i, ]$customer_ID, basket_test_products)

  # calculate recommendations of respective BSRW models
  R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))

  recommendations_bsrw_cos[[i]] <- prediction_weighted_sum(t(R_item), R_bsrw_basket_0.7, w, df_test2_rnd_evidence[i, ]$
      customer_ID, basket_test_products)
  recommendations_bsrw_cp[[i]] <- prediction_weighted_sum(t(R_item2), R_bsrw_basket_0.7, w, df_test2_rnd_evidence[i, ]$
      customer_ID, basket_test_products)
  recommendations_bsrw_bn_0.7[[i]] <- prediction_weighted_sum(P_transition_list[[2]], R_bsrw_basket_0.7, w, df_test2_rnd_
      evidence[i, ]$customer_ID, basket_test_products)

  setTxtProgressBar(pb, i)
}

# print results
bHR("rnd", test, recommendations_rnd, "pop")
bHR("rnd", test, recommendations_cos, "cos")
bHR("rnd", test, recommendations_cp, "cp")
bHR("rnd", test, recommendations_bn_0.7, "bn")
bHR("rnd", test, recommendations_bsrw_cos, "cos_bsrw")
bHR("rnd", test, recommendations_bsrw_cp, "cp_bsrw")
bHR("rnd", test, recommendations_bsrw_bn_0.7, "bn_bsrw")
```

[OUTPUT] Weighted hit rate - $wHR(loo)$

```
d <- seq(0.1, 0.9, by = 0.1)
test <- nrow(df_test2)
n <- 1

w <- personalized_weight(0.7, n)
```

```
6
7  recommendations_pop <- list()
8  recommendations_cos <- list()
9  recommendations_cp <- list()
10 recommendations_bn_0.7 <- list()
11 recommendations_bsrw_cos <- list()
12 recommendations_bsrw_cp <- list()
13 recommendations_bsrw_bn_0.7 <- list()
14
15 popularity <- t(colSums(exp(df_train2[, -1])-1) / sum(colSums(exp(df_train2[, -1])-1)))
16
17 pb <- txtProgressBar(min = 0, max = test, style = 3)
18 for (i in 1:test){
19   hit <- list()
20   for (j in 1:length(df_test2_weighted[[i]]$target)){
21     basket_test <- df_test2_weighted[[i]]$evidence[[j]][, -c(1:2)]
22     basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]
23
24     # calculate recommendation of pop based model
25     hit$pop[j] <- colnames(basket_test[, !(colnames(basket_test) %in% basket_test_products)])[1]
26
27     # calculate recommendations of traditional models
28     hit$cos[j] <- prediction_weighted_sum(t(R_item), t(basket_test), w, df_test2_weighted[[i]]$evidence[[j]]$customer_ID,
           basket_test_products)[1]
29     hit$cp[j] <- prediction_weighted_sum(t(R_item2), t(basket_test), w, df_test2_weighted[[i]]$evidence[[j]]$customer_ID,
           basket_test_products)[1]
30     hit$bn_0.7[j] <- prediction_weighted_sum(P_transition_list[[2]], t(basket_test), w, df_test2_weighted[[i]]$evidence[[j
           ]]$customer_ID, basket_test_products)[1]
31
32     # calculate recommendations of the respective BSRW based models
33     R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))
34
35     hit$bsrw_cos[j] <- prediction_weighted_sum(t(R_item), R_bsrw_basket_0.7, w, df_test2_weighted[[i]]$evidence[[j]]$
           customer_ID, basket_test_products)[1]
36     hit$bsrw_cp[j] <- prediction_weighted_sum(t(R_item2), R_bsrw_basket_0.7, w, df_test2_weighted[[i]]$evidence[[j]]$
           customer_ID, basket_test_products)[1]
37     hit$bsrw_bn_0.7[j] <- prediction_weighted_sum(P_transition_list[[2]], R_bsrw_basket_0.7, w, df_test2_weighted[[i]]$
           evidence[[j]]$customer_ID, basket_test_products)[1]
38   }
39
40   # store the wHR evaluation of pop based models
41   recommendations_pop[i] <- wHR(hit$pop, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in% hit$pop])
42
43   # store the wHR evaluation of the traditional based models
44   recommendations_cos[i] <- wHR(hit$cos, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in% hit$cos])
45   recommendations_cp[i] <- wHR(hit$cp, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in% hit$cp])
46   recommendations_bn_0.7[i] <- wHR(hit$bn_0.7, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in% hit$
         bn_0.7])
47
48   # store the wHR evaluation of the respective BSRW based models
49   recommendations_bsrw_cos[i] <- wHR(hit$bsrw_cos, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in%
         hit$bsrw_cos])
50   recommendations_bsrw_cp[i] <- wHR(hit$bsrw_cp, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in% hit
         $bsrw_cp])
51   recommendations_bsrw_bn_0.7[i] <- wHR(hit$bsrw_bn_0.7, df_test2_weighted[[i]]$target, popularity[, colnames(popularity)
         %in% hit$bsrw_bn_0.7])
52
53   setTxtProgressBar(pb, i)
54 }
```

```
# print results
print(paste("pop", sum(unlist(recommendations_pop)) / length(recommendations_pop), sep = ": "))
print(paste("cos", sum(unlist(recommendations_cos)) / length(recommendations_cos), sep = ": "))
print(paste("cp", sum(unlist(recommendations_cp)) / length(recommendations_cp), sep = ": "))
print(paste("bn_0.7", sum(unlist(recommendations_bn_0.7)) / length(recommendations_bn_0.7), sep = ": "))

print(paste("bsrw_cos", sum(unlist(recommendations_bsrw_cos)) / length(recommendations_bsrw_cos), sep = ": "))
print(paste("bsrw_cp", sum(unlist(recommendations_bsrw_cp)) / length(recommendations_bsrw_cp), sep = ": "))
print(paste("bsrw_bn_0.7", sum(unlist(recommendations_bsrw_bn_0.7)) / length(recommendations_bsrw_bn_0.7), sep = ": "))
```

[EXTENSION] Model-based approach

[FUNCTIONS] Used formulas and built-in packages - PYTHON CODE

```
import tensorflow as tf
import numpy as np
import random
import pandas as pd

import scipy.sparse as sparse
from scipy.sparse.linalg import spsolve
from sklearn.preprocessing import MinMaxScaler
from scipy.sparse import csr_matrix


def FM_GD(data):

  df = data
  x_data = np.matrix(df.drop('target', 1))
  y_data = np.array(df[['target']])

  n, p = x_data.shape

  # number of latent factors
  k = 5

  # design matrix
  X = tf.placeholder('float', shape=[n, p])
  # target vector
  y = tf.placeholder('float', shape=[n, 1])

  # bias and weights
  w0 = tf.Variable(tf.zeros([1]))
  W = tf.Variable(tf.zeros([p]))

  # interaction factors, randomly initialized
  V = tf.Variable(tf.random_normal([k, p], stddev=0.01))

  # estimate of y, initialized to 0.
  y_hat = tf.Variable(tf.zeros([n, 1]))

  linear_terms = tf.add(w0, tf.reduce_sum(tf.multiply(W, X), 1, keep_dims=True))
  interactions = (tf.multiply(0.5, tf.reduce_sum(tf.subtract(tf.pow( tf.matmul(X, tf.transpose(V)), 2),tf.matmul(tf.pow(X,
        2), tf.transpose(tf.pow(V, 2)))), 1, keep_dims=True)))
  y_hat = tf.add(linear_terms, interactions)

  # L2 regularized sum of squares loss function over W and V
  lambda_w = tf.constant(0.001, name='lambda_w')
  lambda_v = tf.constant(0.001, name='lambda_v')

  l2_norm = (tf.reduce_sum(
              tf.add(
                  tf.multiply(lambda_w, tf.pow(W, 2)),
                  tf.multiply(lambda_v, tf.pow(V, 2)))))

  #error = tf.reduce_mean(tf.square(tf.subtract(y, y_hat)))
  error = tf.reduce_sum(tf.math.negative(tf.math.log(tf.math.sigmoid(tf.math.multiply(y_hat, y)))))
  loss = tf.add(error, l2_norm)

```

```
  eta = tf.constant(0.1)
  optimizer = tf.train.AdagradOptimizer(eta).minimize(loss)

  N_EPOCHS = 1000
  # Launch the graph.
  init = tf.global_variables_initializer()
  with tf.Session() as sess:
      sess.run(init)

      for epoch in range(N_EPOCHS):
          indices = np.arange(n)
          np.random.shuffle(indices)
          x_data, y_data = x_data[indices], y_data[indices]
          sess.run(optimizer, feed_dict={X: x_data, y: y_data})

      weight_opt = sess.run(W, feed_dict={X: x_data, y: y_data})
      latent_opt = sess.run(V, feed_dict={X: x_data, y: y_data})
      predictions = sess.run(y_hat, feed_dict={X: x_data, y: y_data})
      loss_value = sess.run(loss, feed_dict={X: x_data, y: y_data})

  output = [w0, weight_opt, latent_opt]

  return(output)


def FM_value(data, phi, mu, M, p):

  data = np.matrix(data)
  phi = np.matrix(phi)

  output = []
  for i in range(len(data)):
    F_2 = np.sum(np.multiply(np.dot(np.transpose(data[i, 0:M]), data[i, M:p]),
                      np.dot(np.transpose(phi[:, 0:M]), phi[:, M:p])))
    F_3 = 0
    for j in range(M, p-1):
      F_3 = np.add(F_3, np.sum(np.multiply(np.dot(np.transpose(data[i, j]), data[i, (j+1):p]),
                                    np.dot(np.transpose(phi[:, j]), phi[:, (j+1):p]))))
    F = np.sum(np.multiply(mu, np.transpose(data[i, :]))) + F_2 + F_3
    output.append(F)

  return(output)


def implicit_als(sparse_data, alpha_val=40, iterations=10, lambda_val=0.1, features=10):

  sparse_data = sparse.csr_matrix(np.matrix(sparse_data))

  # Calculate the foncidence for each value in our data
  confidence = sparse_data * alpha_val

  # Get the size of user rows and item columns
  user_size, item_size = sparse_data.shape

  # We create the user vectors X of size users-by-features, the item vectors
  # Y of size items-by-features and randomly assign the values.
  X = sparse.csr_matrix(np.random.normal(size = (user_size, features)))
  Y = sparse.csr_matrix(np.random.normal(size = (item_size, features)))

```

```
    #Precompute I and lambda * I
    X_I = sparse.eye(user_size)
    Y_I = sparse.eye(item_size)

    I = sparse.eye(features)
    lI = lambda_val * I

    # Start main loop. For each iteration we first compute X and then Y
    for i in range(iterations):
        print ('iteration %d of %d' % (i+1, iterations))

        # Precompute Y-transpose-Y and X-transpose-X
        yTy = Y.T.dot(Y)
        xTx = X.T.dot(X)

        # Loop through all users
        for u in range(user_size):

            # Get the user row.
            u_row = confidence[u,:].toarray()

            # Calculate the binary preference p(u)
            p_u = u_row.copy()
            p_u[p_u != 0] = 1.0

            # Calculate Cu and Cu - I
            CuI = sparse.diags(u_row, [0])
            Cu = CuI + Y_I

            # Put it all together and compute the final formula
            yT_CuI_y = Y.T.dot(CuI).dot(Y)
            yT_Cu_pu = Y.T.dot(Cu).dot(p_u.T)
            X[u] = spsolve(yTy + yT_CuI_y + lI, yT_Cu_pu)


        for i in range(item_size):

            # Get the item column and transpose it.
            i_row = confidence[:,i].T.toarray()

            # Calculate the binary preference p(i)
            p_i = i_row.copy()
            p_i[p_i != 0] = 1.0

            # Calculate Ci and Ci - I
            CiI = sparse.diags(i_row, [0])
            Ci = CiI + X_I

            # Put it all together and compute the final formula
            xT_CiI_x = X.T.dot(CiI).dot(X)
            xT_Ci_pi = X.T.dot(Ci).dot(p_i.T)
            Y[i] = spsolve(xTx + xT_CiI_x + lI, xT_Ci_pi)

    return Y.todense()
```

[SIM] Calculate optimal latent factor - ALS

```
item_factors <- implicit_als(as.matrix(df_train2[, -1]))

R_als <- item_factors %*% t(item_factors)
colnames(R_als) <- colnames(R_item)
row.names(R_als) <- colnames(R_item)
head(R_als)
```

[OUTPUT] Popularity based hit rates - $bHR(pop)$

```
d <- seq(0.1, 0.9, by = 0.1)
test <- nrow(df_test2)
n <- 1

w <- personalized_weight(0.7, n)

recommendations_als <- list()
recommendations_bsrw_als <- list()

pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  basket_test <- df_test2_pop_evidence[i, -c(1:2)]
  basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]

  # calculate recommendation of ALS based model
  recommendations_als[[i]] <- prediction_weighted_sum(R_als, t(basket_test), w, df_test2_pop_evidence[i, ]$customer_ID,
       basket_test_products)

  # calculate recommendations of hybrid model
  R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))

  recommendations_bsrw_als[[i]] <- prediction_weighted_sum(R_als, R_bsrw_basket_0.7, w, df_test2_pop_evidence[i, ]$
       customer_ID, basket_test_products)

  setTxtProgressBar(pb, i)
}
# print results
bHR("pop", test, recommendations_als, "als")
bHR("pop", test, recommendations_bsrw_als, "als_bsrw")
```

[OUTPUT] Random based hit rates - $bHR(rnd)$

```
d <- seq(0.1, 0.9, by = 0.1)
test <- nrow(df_test2)
n <- 1

w <- personalized_weight(0.7, n)

recommendations_als <- list()
recommendations_bsrw_als <- list()

pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  basket_test <- df_test2_rnd_evidence[i, -c(1:2)]
  basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]

  # calculate recommendation of ALS based model
```

```
  recommendations_als[[i]] <- prediction_weighted_sum(R_als, t(basket_test), w, df_test2_rnd_evidence[i, ]$customer_ID,
      basket_test_products)

  # calculate recommendations of hybrid model
  R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))

  recommendations_bsrw_als[[i]] <- prediction_weighted_sum(R_als, R_bsrw_basket_0.7, w, df_test2_rnd_evidence[i, ]$
      customer_ID, basket_test_products)

  setTxtProgressBar(pb, i)
}
# print results
bHR("rnd", test, recommendations_als, "als")
bHR("rnd", test, recommendations_bsrw_als, "als_bsrw")
```

[OUTPUT] Weighted hit rates - $wHR(loo)$

```
d <- seq(0.1, 0.9, by = 0.1)
test <- nrow(df_test2)
n <- 1

w <- personalized_weight(0.7, n)

recommendations_als <- list()
recommendations_bsrw_als <- list()

popularity <- t(colSums(exp(df_train2[, -1])-1) / sum(colSums(exp(df_train2[, -1])-1)))

pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  hit <- list()
  for (j in 1:length(df_test2_weighted[[i]]$target)){
    basket_test <- df_test2_weighted[[i]]$evidence[[j]][, -c(1:2)]
    basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]

    # calculate recommendation of ALS based model
    hit$als[j] <- prediction_weighted_sum(R_als, t(basket_test), w, df_test2_weighted[[i]]$evidence[[j]]$customer_ID,
        basket_test_products)[1]

    R_bsrw_basket_0.7 <- rowSums(cbind(R_bsrw_bn_0.7[[n]][, basket_test_products], 0))

    # calculate recommendations of hybrid model
    hit$bsrw_als[j] <- prediction_weighted_sum(R_als, R_bsrw_basket_0.7, w, df_test2_weighted[[i]]$evidence[[j]]$customer_
        ID, basket_test_products)[1]
  }
  # store the wHR evaluation of the ALS based model
  recommendations_als[i] <- wHR(hit$als, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in% hit$als])

  # store the wHR evaluation of the hybrid model
  recommendations_bsrw_als[i] <- wHR(hit$bsrw_als, df_test2_weighted[[i]]$target, popularity[, colnames(popularity) %in%
      hit$bsrw_als])

  setTxtProgressBar(pb, i)
}
# print results
print(paste("als", sum(unlist(recommendations_als)) / length(recommendations_als), sep = ": "))
print(paste("bsrw_als", sum(unlist(recommendations_bsrw_als)) / length(recommendations_bsrw_als), sep = ": "))
```

[OUTPUT] BSFM popularity & random based hit rates - $bHR(pop)$ & $bHR(rnd)$

```
# initialize one-hot encoding tuples

N <- 1
M <- nrow(R_item)
p <- 2*M

h_user <- matrix(0, nrow = N, ncol = 1, dimnames = list("customer_ID"))
h_target <- matrix(0, nrow = M, ncol = 1, dimnames = list(paste("target", row.names(R_item), sep = "_")))
h_basket <- matrix(0, nrow = M, ncol = 1, dimnames = list(paste("basket", row.names(R_item), sep = "_")))
h <- t(rbind(rbind(h_user, h_target), h_basket))

set.seed(1234)

test <- nrow(df_test2)*0.1
target_list <- list()
predictions_list <- list()
target_list_rnd <- list()
predictions_list_rnd <- list()
counter <- 0
pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  customer_unique <- unique(df_test2[, "customer_ID"])[i]
  basket_train <- df_train2[df_train2$customer_ID == customer_unique, ]
  basket_train_products <- colnames(basket_train[, 2:ncol(basket_train)][, c(basket_train[, 2:ncol(basket_train)]) > 0])


  # generate tuples for training for each customer
  tuple_pos <- list()
  tuple_neg <- list()
  for (t in 1:length(basket_train_products)){
    target_train <- basket_train_products[t]
    evidence_train <- basket_train_products[-t]

    input <- h
    input[, 1] <- customer_unique
    input[, colnames(input) %in% paste("basket", evidence_train, sep = "_")] <- 1
    input[, colnames(input) %in% paste("target", target_train, sep = "_")] <- 1
    tuple_pos[[t]] <- as.data.frame(input)
    tuple_pos[[t]]$target <- 1

    sample <- sample(colnames(R_item)[!(colnames(R_item) %in% basket_train_products)], length(basket_train_products) + 1)
    input2 <- h
    input2[, 1] <- customer_unique
    input2[, colnames(input2) %in% paste("basket", sample[1:length(basket_train_products)], sep = "_")] <- 1
    input2[, colnames(input2) %in% paste("target", sample[length(basket_train_products) + 1], sep = "_")] <- 1
    tuple_neg[[t]] <- as.data.frame(input2)
    tuple_neg[[t]]$target <- -1
  }
  tuple_pos_matrix <- as.data.frame(rbindlist(tuple_pos))
  tuple_neg_matrix <- as.data.frame(rbindlist(tuple_neg))
  training_set <- rbind(tuple_pos_matrix, tuple_neg_matrix)

  # perform AdaGrad called from Python file in order to learn the parameters
  params <- FM_GD(training_set[-1])

  # obtain values of the factorization machine for every tuple
  basket_test_all <- df_test2[df_test2$customer_ID %in% customer_unique, ]
```

```
  for (j in 1:nrow(basket_test_all)){
    counter <- counter + 1
    basket_test <- basket_test_all[j, ]
    basket_test_products <- colnames(basket_test[, 3:ncol(basket_test)][, c(basket_test[, 3:ncol(basket_test)]) > 0])

    # calculate all test tuple values corresponding for bHR(pop)
    evidence_test <- basket_test_products[1:(length(basket_test_products) - 3)]
    target_list[[counter]] <- as.data.frame(basket_test_products[(length(basket_test_products)-2):length(basket_test_
        products)])
    tuple_test <- matrix(0, nrow = ncol(R_item) - length(evidence_test), ncol = 2*M, dimnames = list(NULL, colnames(h)
        [-1]))
    tuple_test[, colnames(tuple_test) %in% paste("basket", evidence_test, sep = "_")] <- 1
    tuple_test[, !(colnames(tuple_test) %in% paste("target", evidence_test, sep = "_") | colnames(tuple_test) %like% "
        basket")] <- diag(1, nrow = nrow(tuple_test))
    testing_set <- as.data.frame(tuple_test)
    testing_set$target <- 1
    testing_set <- cbind(customer_ID = customer_unique, testing_set)

    # convert to python code in order to speeden things up as R is very slow in matrix multiplications
    output <- FM_value(as.matrix(testing_set[, -c(1, ncol(h))]), as.matrix(params[[3]]), as.matrix(params[[2]]), as.
        integer(M), as.integer(p))
    output <- as.data.frame(t(unlist(output, use.names=FALSE)))
    colnames(output) <- colnames(R_item)[!(colnames(R_item) %in% evidence_test)]

    predictions <- colnames(output[, order(output, decreasing = TRUE)])
    predictions_list[[counter]] <- as.data.frame(predictions[1:3])

    # calculate all test tuple values corresponding for bHR(rnd)
    rnd <- sample(length(basket_test_products), 3)
    evidence_test_rnd <- basket_test_products[-c(rnd)]
    target_list_rnd[[counter]] <- as.data.frame(basket_test_products[c(rnd)])
    tuple_test_rnd <- matrix(0, nrow = ncol(R_item) - length(evidence_test_rnd), ncol = 2*M, dimnames = list(NULL,
        colnames(h)[-1]))
    tuple_test_rnd[, colnames(tuple_test_rnd) %in% paste("basket", evidence_test_rnd, sep = "_")] <- 1
    tuple_test_rnd[, !(colnames(tuple_test_rnd) %in% paste("target", evidence_test_rnd, sep = "_") | colnames(tuple_test_
        rnd) %like% "basket")] <- diag(1, nrow = nrow(tuple_test_rnd))
    testing_set_rnd <- as.data.frame(tuple_test_rnd)
    testing_set_rnd$target <- 1
    testing_set_rnd <- cbind(customer_ID = customer_unique, testing_set_rnd)

    # convert to python code in order to speeden things up as R is very slow in matrix multiplications
    output_rnd <- FM_value(as.matrix(testing_set_rnd[, -c(1, ncol(h))]), as.matrix(params[[3]]), as.matrix(params[[2]]),
        as.integer(M), as.integer(p))
    output_rnd <- as.data.frame(t(unlist(output_rnd, use.names=FALSE)))
    colnames(output_rnd) <- colnames(R_item)[!(colnames(R_item) %in% evidence_test_rnd)]

    predictions_rnd <- colnames(output_rnd[, order(output_rnd, decreasing = TRUE)])
    predictions_list_rnd[[counter]] <- as.data.frame(predictions_rnd[1:3])
  }
  setTxtProgressBar(pb, i)
}
```

[OUTPUT] BSFM Weighted hitrate - $wHR(loo)$

```
# initialize one-hot encoding tuples

N <- 1
M <- nrow(R_item)
```

```
p <- 2*M

h_user <- matrix(0, nrow = N, ncol = 1, dimnames = list("customer_ID"))
h_target <- matrix(0, nrow = M, ncol = 1, dimnames = list(paste("target", row.names(R_item), sep = "_")))
h_basket <- matrix(0, nrow = M, ncol = 1, dimnames = list(paste("basket", row.names(R_item), sep = "_")))
h <- t(rbind(rbind(h_user, h_target), h_basket))

test <- nrow(df_test2)*0.01
target_list <- list()
predictions_list <- list()
target_list_rnd <- list()
predictions_list_rnd <- list()

# initialize popularity based chance p(x)
popularity <- t(colSums(exp(df_train2[, -1])-1) / sum(colSums(exp(df_train2[, -1])-1)))

counter <- 0
pb <- txtProgressBar(min = 0, max = test, style = 3)
for (i in 1:test){
  customer_unique <- unique(df_test2[, "customer_ID"])[i]
  basket_train <- df_train2[df_train2$customer_ID == customer_unique, ]
  basket_train_products <- colnames(basket_train[, 2:ncol(basket_train)][, c(basket_train[, 2:ncol(basket_train)]) > 0])

  # generate tuples for training for each customer
  tuple_pos <- list()
  tuple_neg <- list()
  for (t in 1:length(basket_train_products)){
    target_train <- basket_train_products[t]
    evidence_train <- basket_train_products[-t]

    input <- h
    input[, 1] <- customer_unique
    input[, colnames(input) %in% paste("basket", evidence_train, sep = "_")] <- 1
    input[, colnames(input) %in% paste("target", target_train, sep = "_")] <- 1
    tuple_pos[[t]] <- as.data.frame(input)
    tuple_pos[[t]]$target <- 1

    sample <- sample(colnames(R_item)[!(colnames(R_item) %in% basket_train_products)], length(basket_train_products) + 1)
    input2 <- h
    input2[, 1] <- customer_unique
    input2[, colnames(input2) %in% paste("basket", sample[1:length(basket_train_products)], sep = "_")] <- 1
    input2[, colnames(input2) %in% paste("target", sample[length(basket_train_products) + 1], sep = "_")] <- 1
    tuple_neg[[t]] <- as.data.frame(input2)
    tuple_neg[[t]]$target <- -1
  }
  tuple_pos_matrix <- as.data.frame(rbindlist(tuple_pos))
  tuple_neg_matrix <- as.data.frame(rbindlist(tuple_neg))
  training_set <- rbind(tuple_pos_matrix, tuple_neg_matrix)

  # call Python code to learn the optimal parameters
  params <- FM_GD(training_set[-1])

  # loop all factorzation machine value of all basket items in order to calculate wHR(loo)
  user_index <- df_test2_weighted_customer_ID[df_test2_weighted_customer_ID$customer_ID %in% customer_unique, ]$index
  for (q in 1:length(user_index)){
    index <- user_index[q]
    hit <- list()
    counter <- counter + 1
    for (j in 1:length(df_test2_weighted[[index]]$target)){
```

```
      basket_test <- df_test2_weighted[[index]]$evidence[[j]][, -c(1:2)]
      basket_test_products <- names(basket_test)[which(basket_test > 0, arr.ind = TRUE)[, "col"]]

      # generate test tuples
      evidence_test <- basket_test_products
      tuple_test <- matrix(0, nrow = ncol(R_item) - length(evidence_test), ncol = 2*M, dimnames = list(NULL, colnames(h)
          [-1]))
      tuple_test[, colnames(tuple_test) %in% paste("basket", evidence_test, sep = "_")] <- 1
      tuple_test[, !(colnames(tuple_test) %in% paste("target", evidence_test, sep = "_") | colnames(tuple_test) %like% "
          basket")] <- diag(1, nrow = nrow(tuple_test))
      testing_set <- as.data.frame(tuple_test)
      testing_set$target <- 1
      testing_set <- cbind(customer_ID = customer_unique, testing_set)

      # convert to python code in order to speeden things up as R is very slow in matrix multiplications
      output <- FM_value(as.matrix(testing_set[, -c(1, ncol(h))]), as.matrix(params[[3]]), as.matrix(params[[2]]), as.
          integer(M), as.integer(p))
      output <- as.data.frame(t(unlist(output, use.names=FALSE)))
      colnames(output) <- colnames(R_item)[!(colnames(R_item) %in% evidence_test)]

      predictions <- colnames(output[, order(output, decreasing = TRUE)])
      hit$predictions[j] <- predictions[1]
    }
    # calculate wHR(loo)
    predictions_list[[counter]] <- wHR(hit$predictions, df_test2_weighted[[index]]$target, popularity[, colnames(
        popularity) %in% hit$predictions])
  }

  setTxtProgressBar(pb, i)
}
```