Fast recommendation on latent collaborative relations

Chien-Liang Liu\textsuperscript{a,}\textsuperscript{*}, Xuan-Wei Wu\textsuperscript{b}

\textsuperscript{a} Department of Industrial Engineering and Management, National Chiao Tung University, 1001 University Road, Hsinchu 300, Taiwan (R.O.C.)
\textsuperscript{b} Department of Computer Science, National Chiao Tung University, 1001 University Road, Hsinchu 300, Taiwan (R.O.C.)

A R T I C L E   I N F O

Article history:
Received 28 January 2016
Revised 1 June 2016
Accepted 13 June 2016
Available online 18 June 2016

Keywords:
Recommender systems
Latent factor model
Locality-sensitive hashing
Nearest neighbors

A B S T R A C T

One important property of collaborative filtering recommender systems is that popular items are recommended disproportionately often because they provide extensive usage data and, thus, can be recommended to more users. Compared to popular products, the niches can be as economically attractive as mainstream fare for online retailers. The online retailers can stock virtually everything, and the number of available niche products exceeds the hits by several orders of magnitude. This work addresses accuracy, coverage and prediction time issues to propose a novel latent factor model called latent collaborative relations (LCR), which transforms the recommendation problem into a nearest neighbor search problem by using the proposed scoring function. We project users and items to the latent space, and calculate their similarities based on Euclidean metric. Additionally, the proposed model provides an elegant way to incorporate with locality sensitive hashing (LSH) to provide a fast recommendation while retaining recommendation accuracy and coverage. The experimental results indicate that the speedup is significant, especially when one is confronted with large-scale data sets. As for recommendation accuracy and coverage, the proposed method is competitive on three data sets.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

With the rise of the Internet, the last decade has witnessed the great success of recommender systems. The recommender systems serve two important functions. First, they can significantly help users find relevant and interesting items in the information era. Second, they help the businesses make more profits, since recommender systems provide more chances for the niche items to be bought by customers.

There has been much work done both in the industry and academia on developing new approaches to recommender systems. The recommender systems can significantly help users find relevant and interesting items in the information era, so they have been applied to many application domains, including, but not limited to, telecom businesses \cite{1}, e-government services \cite{2,3} and online shopping \cite{4}. The collaborative filtering (CF) recommender system is considered to be one of the most successful approaches, and many CF algorithms have been proposed over the last decades \cite{5-7}. It recommends to the active user the items that other users with similar tastes liked in the past. However, while majority of algorithms proposed in recommender systems literature have focused on improving recommendation accuracy, other important aspects of recommendation quality, such as the coverage of recommendations, have been overlooked \cite{8,9}.

One important property of CF recommender systems is that popular items are recommended disproportionately often because they provide extensive usage data and, thus, can be recommended to more users \cite{10-12}. Traditional retail tends to stock the popular products, since shelf space is expensive. Compared to popular products, the niches can be as economically attractive as mainstream fare for online retailers. The online retailers can stock virtually everything, and the number of available niche products exceeds the hits by several orders of magnitude.

Coverage concerns the degree to which recommendations cover the set of available items and the degree to which recommendations can be generated to all potential users. A recommender system with high coverage presents to the end user a more detailed investigation of the product space and benefits from long tail phenomenon, which has gained popularity in recent times as describing the retailing strategy of selling a large number of unique items with relatively small quantities \cite{13,14}. Therefore, coverage is another indicator of quality.

Recommendation accuracy alone is insufficient to satisfy user expectations, since other characteristics, such as coverage, novelty, serendipity and trust must be considered as well \cite{8,9}. Besides, most academic recommender systems are based on offline
methods that require costly recalculation over all the observed data [15], but critical problems concerning the computational cost emerge in practical situations. Building large-scale recommender systems need to consider other practical issues such as prediction time or scalability. For example, real world online recommender systems must adhere to strict response-time constraints. Consequently, the motivation of this work is to devise a fast recommendation algorithm, which considers recommendation accuracy and coverage in the model.

This work focuses on training to optimize retrieval for the top k items and addresses accuracy, coverage and prediction time issues to propose a novel algorithm called latent collaborative relations (LCR). Latent factor models have become popular in recently years [16–18], and most previous latent factor models use inner product as metric to calculate item similarities. Then, one can retrieve the most suitable items by scoring and sorting all items. However, when the number of items is enormous, scoring all items becomes intractable. Several research studies used approximate matches to speed up the retrieval. For example, Bachrach et al. [19] proposed an order preserving transformation to map the maximum inner product search problem to Euclidean space nearest neighbor search problem. Then, the item with the smallest Euclidean distance in the transformed space is retrieved based on the nearest neighbor search with PCA-Tree [20] data structure. Koenigstein et al. [21] proposed to use spherical clustering to index the users on the basis of their preferences, and pre-compute recommendations only for the representative user of each cluster to obtain extremely efficient approximate solutions. This work transforms the recommendation problem into a nearest neighbor search with $l_2$ norm problem, and integrates multi-probe locality sensitive hashing (LSH) [22] into the model to perform approximate search, which can speedup the retrieval by $5 \times -313 \times$ on three data sets used in the experiments.

The contributions and innovation of this work are listed as follows. First, this work combines latent factors and $l_2$ norm to formulate the recommendation problem as a $k$-nearest-neighbor ($k$NN) problem, and proposes a novel recommendation algorithm, which uses locality-sensitive hashing (LSH) [23] to reduce search time complexity. The prediction guarantees sub-linear time retrieval with a bounded approximation error. Second, this work uses Euclidean distance to measure the similarity of two vectors, and we empirically show that calculating the distance in Euclidean space achieves a higher coverage than those using inner products. Finally, the proposed algorithm can incorporate various features into the model to attain a higher recommendation accuracy and tackle cold-start problems. In summary, this work considers recommendation accuracy and practical deployment issues to devise a novel algorithm, which gives competitive performance while considering prediction time and long tail issues. In the experiments, we compare the proposed algorithm with several algorithms on three data sets. The experimental results indicate that the proposed algorithm is competitive on recommendation accuracy and coverage.

The rest of this paper is organized as follows. Section 2 presents related surveys of latent factor models and nearest neighbor search. Section 3 introduces the proposed latent collaborative relations algorithm. Next, Section 4 summarizes the results of several experiments. Conclusions are finally drawn in Section 5.

2. Related work

The proposed method is based on latent factors and nearest neighbor search, so the literature survey focuses on these two research topics.

2.1. Latent factor models

The CF is perhaps one of the most well-studied recommendation approach, since it predicts the utility of items for a particular user based on the items previously rated by other users [24]. The two more successful approaches to CF are neighborhood models and latent factor models [5]. The neighborhood models comprise user-based [25] and item-based approaches [4,26,27], in which a rating is estimated based on the relationships between users and items, respectively. Koren [16] demonstrated that using low rank matrix factorization, which is a latent factor model, is superior to classic nearest-neighbor techniques for product recommendations.

The success of Netflix competition makes matrix factorization [16] one of the most popular collaborative filtering methods. The idea behind matrix factorization models is to map users and items to a latent space, and calculate the similarities with inner products. Latent factor models have become popular in recent years, since they can model interactions between variables and obtain good scalability and high predictive accuracy even in problems with huge sparsity [17]. The success of MF inspires many research studies to use latent factors to devise algorithms. Rendle combined the advantages of support vector machines (SVM) with factorization models to propose an algorithm called factorization machines (FM) [17,18]. Chen et al. [28] observed that the majority of matrix factorization models share common patterns, so they proposed a framework called feature-based matrix factorization, which can incorporate various features into the model. Root mean squared error (RMSE) is probably the most popular metric used in evaluating accuracy of predicted ratings. However, recent research studies have shown that ranking metric is a better evaluation metric [29–31], since most recommendation applications present to the user a list of recommendations, typically as vertical or horizontal list, imposing a certain natural browsing order. Therefore, recent work on recommendation tended to use ranking approach to devise algorithms. Rendle et al. [30] proposed a pairwise ranking algorithm called Bayesian Personalized Ranking (BPR) that is the maximum posterior estimator derived from a Bayesian analysis of the problem. The proposed method also uses maximum posterior estimator, but several differences exist between the two methods. First, the purposes of the two methods are totally different. The BPR focuses on presenting a generic optimization criterion for personalized ranking, while the proposed method addresses accuracy, coverage and prediction time issues to transform the recommendation problem into a nearest neighbor search problem by using the proposed scoring function. Second, the BPR considers to use the user-specific order of two items as the training data of pairwise ranking, while the proposed method uses the global ordering of two instances. Finally, the proposed method can incorporate LSH into the proposed model to provide a fast recommendation. Gantner et al. [32] proposed an adapted version of BPR to focus on a specific problem in which the candidate items are sampled from a given distribution. The adapted BPR takes the non-uniform sampling of negative test items into account rather than uniform sampling as used by the BPR. Liu et al. [33] proposed an algorithm called list-wise probabilistic matrix factorization (ListPMF), which maximizes the log-posterior over the predicted preference order with the observed preference orders.

Recommendation data can be represented as relational data, which comprises entities and relations between them. In movie recommendation problem, the entities involve users and movies, while relations encode users’ ratings of movies. Relational learning is concerned with predicting unknown values of a relation. Singh and Gordon [34] focused on relational learning and proposed a collective matrix factorization algorithm, which simultaneously factorizes several matrices and shares parameters among factors. The advent and popularity of social network inspires many research
studies to consider social factors into recommender systems [35,36]. For example, Qian et al. [35] considered to fuse three social factors, including personal interest, inter-personal interest similarity, and inter-personal influence, into a unified personalized recommendation model based on probabilistic matrix factorization. Meanwhile, Jiang et al. [36] fused individual preference and inter-personal influence in latent space.

2.2. Nearest neighbor search

Nearest neighbor search from massive data sets is a fundamental problem in data mining, and this problem arises in numerous application fields such as computer vision, plagiarism detection, and recommender systems. Given a set $S$ of points in a space $M$ and a query point $q \in M$, the goal is to find the closest point in $S$ to $q$. A generalization of this problem is to find the $k$ closest points, namely $k$NN search. A naive approach is to use linear search to compute the distance from the query point to every other point, resulting in a running time of $O(Nd)$, where $N$ is the cardinality of $S$ and $d$ is the dimensionality of $M$. This is very inefficient for massive data sets, so many research studies have been developed to reduce time complexity for nearest neighbor search over the past decades.

Several research studies tackle this problem by designing efficient data structures to find exact nearest neighbors [37–39]. The $k$-d tree [37] and its variants are probably the most popular data structures used for searching in multidimensional spaces. A review of nearest neighbor data structures from the database perspective can refer to [40]. However, these algorithms become inefficient as the dimension grows. Recently, approximate nearest neighbor searching algorithms have become increasingly important, especially in high dimensional data sets [23,41,42]. In this formulation, the algorithm is allowed to return a point whose distance from the query is at most $c$ times the distance from the query to its nearest point, in which $c > 1$ is called the approximation factor [43].

Locality sensitive hashing (LSH) [23] is one of the most promising approximate algorithms, and has attracted extensive research interest in the past years.

Let $H$ be a family of hash functions, each of which maps $\mathbb{R}^d$ to some universe $U$. The family $H$ is called locality sensitive (with proper parameters) if it satisfies the following condition [43].

**Definition 2.1** (Locality sensitive Hashing). A family $H$ is called $(R, cR, P_1, P_2)$-sensitive if for any two points $p, q \in \mathbb{R}^d$:

- if $\| p - q \| \leq R$, then $P[h(q) = h(p)] \geq P_1$
- if $\| p - q \| \geq cR$, then $P[h(q) = h(p)] \leq P_2$

To use LSH for approximate nearest search, we pick $c > 1$ and $P_1 > P_2$. The key idea of LSH is to hash the points using several hash functions to ensure that similar items are mapped to the same buckets with high probability. Given a query point $q$, one can determine near neighbors by hashing $q$ and retrieving the points stored in buckets containing $q$. It is noted that the hashing used in LSH is different from conventional hash functions, which avoid collisions for similar items.

Several LSH families for different distance functions have been discovered, including Hamming distance, Jaccard, $\ell_1$ and $\ell_2$ distance [23]. Datar et al. [44] further proposed LSH families for $l_p$ norms based on $p$-stable distribution [45]. The LSH requires enormous hash tables to achieve good search quality. Lv et al. [22] proposed a new indexing scheme called multi-probe LSH to overcome this problem by intelligently probing multiple buckets that are likely to contain query results in a hash table.

Many applications rely on measuring similarity between data points, and hashing technique has been widely adopted for approximate nearest neighbor search on large-scale data sets. For example, the similarity learning is concerned with binary relationship in many application settings, namely two data points are similar or dissimilar, and this problem could be addressed by similarity-sensitive hashing [46,47]. A typical application is content-based image retrieval, whose goal is to find the images in the database similar to the query image. In multimedia domain, an object often appears in several different views. Ding et al. [48] focused on multi-view problems and proposed a hashing algorithm called Collective Matrix Factorization Hashing (CMFH), which assumes that each view of one instance generates identical hash codes.

3. Latent collaborative relations

In collaborative filtering setting, there exist users $U = \{u^{(1)}, \ldots, u^{(|U|)}\}$, items $I = \{i^{(1)}, \ldots, i^{(|I|)}\}$, and data instances are represented as input features $X = \{x = (u, i) \mid u \in U, i \in I\}$ and their corresponding labels $Y = \{y_{u,i}\}$. This work focuses on implicit information, namely $y_{u,i}$ is one for the positive class (e.g., clicked, watched movies, played songs, purchased or assigned tags) or zero for negative class. The proposed method also provides the flexibility for the model to use various features, including collaborative filtering and content-based features.

3.1. Feature representation

As for content-based features, we introduce $\alpha \in A(u)$ and $\beta \in A(i)$ to represent the feature set for a user $u$ and an item $i$, respectively. For example, $A(u)$ can comprise a user profile such as gender and age, while $A(i)$ can consist of item features such as item categories and price. The standard collaborative filtering setting is a special case of the proposed model when $A(u)$ only contains the user id, and $A(i)$ only comprises the item id.

Moreover, this work introduces $x_{ui}$ to represent the value of the user (or item) feature $v$, and this work uses different schemes to present $x_{ui}$ for different cases. For the features whose values are real numbers, no change is applied to them. For a categorical feature with $m$ categories, we use $m$ variables to represent it, and each variable is a binary indicator. Therefore, the gender feature will be encoded as two variables, $v_1$ for “gender is male” and $v_2$ for “gender is female”, respectively. If $x_{ui} = 0$ and $x_{ui} = 1$, then the user is female. Additionally, categorical variables can belong to several categories at the same time, and we use normalization technique to encode them. For example, let $v_1$ and $v_2$ denote animation and comedy genres, respectively. If genres of this movie include animation and comedy, its $x_{v_1}$ and $x_{v_2}$ are both 0.5, and the other genre variables are zeros. Note that we normalize the values to let the sum be one.

3.2. Scoring function

The proposed method is a latent factor model, and all of the features can be mapped to a latent space with dimension $\omega$. This work associates the user feature $\alpha$ with a latent factor vector $w_\alpha \in \mathbb{R}^\omega$. The association is also applied to the item feature, namely $w_\beta \in \mathbb{R}^\omega$ is the latent factor vector for the item feature $\beta$. Hence, each user and item can be considered as a combination of latent vectors, and we use the center of mass of latent vectors to denote a user or item. Most previous research studies used inner product to calculate the similarity between user and item. However, the result of Euclidean embedding is more intuitively understandable for humans and the neighborhood structure of the unified Euclidean space allows very efficient recommendation queries [49], so this work uses Euclidean distance to measure the similarity of two vectors. We define a scoring function for a given item $i$ with respect
to a given user $u$ as presented in Eq. (1).

$$f_u(x) = f_d(u, l) = -(w_u - w_l)^T (w_u - w_l),$$

where $w_u = \sum_{v \in \Lambda(u)} X_v \sum_{a \in \Lambda(u)} x_{a}w_{x}$ and $w_l = \sum_{v \in \Lambda(l)} X_v \sum_{b \in \Lambda(l)} x_{b}w_{x}$.

### 3.3. Pairwise loss function

This work models the problem by using maximum a posteriori (MAP) estimator. Let $\Theta \in \mathbb{R}$ be a component of a certain feature's latent factor vector, i.e., if $\Theta$ is the $f$-component of $w$, then $\Theta$ is $w_{f}$, and $\Theta$ denotes the collection of all the components of features' latent factor vectors, i.e., $\Theta = [w_{f} | f \in \Lambda(u) \cup \Lambda(l) \text{ and } f = 1, 2, \ldots, n]$. Assume each parameter $\Theta$ in $\Theta$ is independent to each other, and follows a Gaussian distribution $N(0, \lambda^{-1})$. Then, the prior of $\Theta$ is

$$p(\Theta) = \prod_{\Theta \in \Theta} \frac{1}{\sqrt{2\pi \lambda}} e^{-\frac{1}{2\lambda} \Theta^2}.$$  

Explicit feedback such as ratings is unavailable in most application settings, but it is much easier to collect implicit feedback such as clicks, view times, and purchases from websites. Therefore, this work focuses on implicit feedback. Interesting about implicit feedback systems is that only positive observations are available, and non-observed data instances are a mixture of negative feedback and missing values. It is difficult to differentiate real negative feedback from missing values, so this work simplifies the task by categorizing the data into two classes, positive and negative. For those data instances that do not belong to positive class, they are viewed as negative instances.

Let $X^{(+)}$ denote the set of positive instances and $X^{(-)}$ the set of negative instances. Additionally, assume the pairs of data are independent and identically distributed (i.i.d.), then the log likelihood of $\Theta$ conditional on observed data is

$$\ell(\Theta) = -\sum_{(x^{(+)}, x^{(-)}) \subset X^{(+)}} \ln p(x^{(+)} > x^{(-)} | \Theta),$$

where $X^{(+) \times X^{(-)}}$ and $p(x^{(+) > x^{(-)}} | \Theta)$ is the probability of the data instance $x^{(+)}$ ranked before $x^{(-)}$ given the model parameters $\Theta$.

In recommendation setting, the goal is to let positive examples be ranked before negative ones. Thus, further use the scoring function as listed in Eq. (1) and a sigmoid function $\sigma$ to transform the pairwise relationship between positive and negative examples into a classification problem. Eq. (4) shows the formula in which the output is a probability value between zero and one. It approaches one if $f_d(x^{(+)})$ is much larger than $f_d(x^{(-)})$, and zero if $f_d(x^{(-)})$ is much less than $f_d(x^{(+)})$.

$$p(x^{(+) > x^{(-)}} | \Theta) = \sigma(f_d(x^{(+)}) - f_d(x^{(-)})) = \frac{1}{1 + e^{-(f_d(x^{(+)}) - f_d(x^{(-)})}}$$

Combine the prior as listed in Eq. (2) and likelihood listed in Eq. (3), one can obtain the loss function as presented in Eq. (5), which is the negative of the MAP estimator.

$$\sum_{(x^{(+)}, x^{(-)}) \subset X^{(+)}} -\ln p(x^{(+) > x^{(-)}} | \Theta) + \frac{\lambda\Theta}{2} \sum_{\Theta \in \Theta} \Theta^2$$

Each parameter $\lambda$ can be treated as a hyperparameter, which can be searched by a holdout method. Practically, one can assume not every parameter has its own hyperparameter. The proposed model groups parameters and let regularization parameter $\lambda \Theta \in \mathbb{R}^+$ be shared among parameters in the group.

### 3.4. Learning algorithm

Using batch learning algorithms such as gradient descent to minimize Eq. (5) is infeasible when one is confronted with large-scale data sets. The main reason is that Eq. (5) involves summing over all examples. Moreover, it is unreasonable to assume that the data set is fixed, since the data set is growing all the time. Besides computational complexity, using batch learning approaches to design recommendation algorithms generally suffers from model update problem, explaining why this work proposes to minimize the loss function by using stochastic gradient descent, which performs gradient updates for just one example once. Thus, instead of optimizing Eq. (5), the objective function of this work is listed in Eq. (6), where we technically sample a positive instance $x^{(+)}$ and a negative instance $x^{(-)}$ from the data set at each iteration.

$$J(\Theta) = -\ln \sigma(f_d(x^{(+)}) - f_d(x^{(-)})) + \frac{\lambda\Theta}{2} \sum_{\Theta \in \Theta} \Theta^2$$

With the loss function listed in Eq. (6) along with the proposed scoring function, we can derive the update rules in the form of the latent factor vector $w$ for the model as listed in Eq. (7), where $\eta$ is the learning rate. The partial derivative of the loss function can be further expressed as Eq. (8), in which the partial derivatives of the scoring function with respect to parameters are listed in Eq. (9). Finally, learning model parameters with LCR is done by looping over Eq. (7).

$$w = w - \eta \frac{\partial}{\partial w} J(\Theta)$$

$$\frac{\partial}{\partial w} J(\Theta) = -\left(\frac{\partial f_d(x^{(+)})}{\partial w} - \frac{\partial f_d(x^{(-)})}{\partial w}\right) + \lambda\Theta$$

$$\frac{\partial}{\partial w} f_d(x) = \begin{cases} -2x_v (w_u - w_l) & \text{if } w \text{ is a user latent factor vector} \\
\sum_{v \in \Lambda(u)} x_v & \sum_{v \in \Lambda(l)} x_v & \text{if } w \text{ is an item latent factor vector} \end{cases}$$

The complexity for updating Eq. (7) depends on the partial derivative of cost function $J(\Theta)$ as listed in Eq. (8), which involves the scoring function $f_d(x)$ as listed in Eq. (1) and partial derivatives of the scoring function $f_d(x)$ as listed in Eq. (9). The time complexities for Eq. (1) and Eq. (9) both depend on $w_u$. $w_l$, and $w_u - w_l$. So the time complexity for updating Eq. (7) is $O(|\Lambda(u)| + |\Lambda(l)|)$.

Besides the update rules listed above, this work further discusses implementation issues, which can make the learning algorithm more efficient. We can further analyze the partial derivative listed in Eq. (8), which has a multiplicative scalar, namely $\frac{1}{1 + e^{-(f_d(x^{(+)}) - f_d(x^{(-)})}}$. Here, we introduce a variable $\delta(x^{(+)}, x^{(-)})$ listed...
below to represent the scalar, and it comprises two properties.

\[
\delta(x^+, x^-) = \frac{1}{1 + e^{\phi(x^+) - \phi(x^-)}} = 1 - \sigma(f_d(x^+) - f_d(x^-))
\]

First, the value of \(\delta(x^+, x^-)\) can be further interpreted as the influence of a specific data pair instance \((x^+, x^-)\) on the update of the parameter \(\theta\), and approaches zero if the ranking is correct and the positive instance is assigned a larger score than that of the negative instance. Conversely, it approaches one in the opposite case. Second, it is apparent that the output width \(\delta(x^+, x^-)\) is also a probability value. We design a sampling process which helps the learning process obtain the data pairs with good quality based on the above two properties. Instead of using a straightforward approach to draw data pairs uniformly and accepting all drawn data pairs to update model parameters, the implementation of this work samples data instances according to the value of \(\delta(x^+, x^-)\), namely, to draw \((x^+, x^-) \in X^2 \propto \delta(x^+, x^-)\). Note that \(\delta(x^+, x^-)\) is not a fixed scale value, but depends on the model parameters \(\theta\).

The proposed algorithm is presented in Algorithm 1, which comprises two parts. First, sample a pair of instances. The sampling process is listed in Line 3–4, in which we draw \(x^+\) and \(x^-\) from \(X^+\) and \(X^-\) respectively. The initialization of the latent factor vectors are completed in Line 5–9. Then, we use the sampling technique mentioned above to obtain data pairs. Here, we introduce a random number \(r\) which is sampled from an uniform distribution on the interval \([0, 1]\) to determine whether accepting the drawn data pair \((x^+, x^-)\). The above processes are mentioned in Line 10–12. The second part focuses on updating model parameters as listed in Line 13–15, in which we make a gradient step to minimize the pairwise loss.

**Algorithm 1**: Latent collaborative relations algorithm.

**Input**: \(X^+, X^-, \eta, \lambda_{\phi\theta}, \Theta\)

**Output**: \(\Theta\)

1. **repeat**
   2. **repeat**
   3. Draw \(x^+ \in X^+\) uniformly
   4. Draw \(x^- \in X^-\) uniformly
   5. **foreach** latent factor \(w\) of \(x^+\) and \(x^-\) **do**
   6. **if** \(w\) is not initial **then**
   7. Initialize \(w\)
   8. **end**
   9. \(\delta(x^+, x^-) = 1 - \sigma(f_d(x^+) - f_d(x^-))\)
   10. Draw \(r \sim \text{unif}(0, 1)\)
   11. **until** \(r < \delta(x^+, x^-)\);
   12. **foreach** latent factor \(w\) of \(x^+\) and \(x^-\) **do**
   13. \(w = w + \eta \left(\delta(x^+, x^-) \lambda_{\phi\theta} (f_d(x^+) - f_d(x^-))\right)\)
   14. **end**
   15. **until** convergence;

3.5. Fast approximate recommendation generation by multi-probe LSH

In a ranking-based recommendation setting, traditional latent factor models require two steps to obtain the top \(k\) relevant items with respect to an active user. First, project users and items to a latent space, and then calculate their similarities in the latent space. The most commonly used similarity metric between a user vector and an item vector is inner product. Second, sort the scores to obtain the top \(k\) items. The above approach suffers from computation problem when one is confronted with a large-scale item set. This work transforms the recommendation problem into a nearest neighbor search problem. Given a user latent vector as a query point, and all the item latent factors are data points. The goal is to search for the data points that are close to the query point. Straightforward approach is to compute all the pairwise distances between the query point and data points, and sort the distances to obtain the nearest ones. It is apparent that this approach is infeasible for massive data sets. The \(k\)-d tree provides an efficient alternative to speed up the retrieval, but it is not suitable for efficiently finding the nearest neighbor in high-dimensional spaces.

To provide an efficient recommendation retrieval, this work provides the flexibility to incorporate an efficient approximate nearest neighbor search algorithm called LSH into the proposed latent collaborative relations (LCR) model. Although the proposed LCR is based on latent factors, several differences exist between the proposed LCR and previous latent factor models. First, instead of using inner product, the LCR searches for the top \(k\) relevant items based on \(\ell_2\) norm. The LCR searches for items near the active user, and the search space is bounded by a circle. Second, the LCR can transform the ranking problem into an approximate nearest neighbor problem, in which we use LSH to efficiently locate the candidates. Third, the LCR provides the recommendations with coverage owing to the characteristics of Euclidean space, and the experiments present the results.

The goal of the hashing used in LSH is to maximize probability of collision of similar items, and the performance of LSH is associated with the parameters, including the number of hash functions \(M\), the number of hash tables \(l\) and the window size \(W\). The LSH relies on hashing functions to reduce the candidate sizes. Given an item latent factor vector \(w\), we insert it into hash tables, each of which is defined as follows [44]:

\[
h_{a,b}(w) = \left[ \frac{aw + b}{W} \right]
\]

where \(a\) is an \(\omega\)-dimensional random vector with entries chosen independently from a standard Gaussian distribution, which is \(2\)-stable and works for the Euclidean distance, and \(b\) is a real number chosen uniformly from the range 0 to \(W\). The \(h_{a,b}\) projects the latent factor vector \(w\) onto \(a\) with an offset \(b\) and indexed by the window size \(W\).

Intuitively, each hashing function can be viewed as a partition over the space, so a larger value of \(M\) leads to fewer latent factor vectors colliding in the same bucket, and result in lower recall but shorter query time. To solve the problem, multiple hash tables (parameterized by \(L\)) are used, that is, increasing \(L\) will enlarge the number of candidate neighbors since more buckets are included across these tables. Fig. 1 illustrates the idea of LSH, in which \(M = 2\) and \(L = 2\). The hashing function in the left hand side hashes the query data point to bucket (1,1), and the points that are hashed to bucket (1,1) are candidate neighbors. Similarly, we can use the hashing function in the right hand side to find candidate neighbors. Using the two hashing functions, the points with orange circle, purple circle, and dark blue circle are candidates. The performance of LSH relies on the parameters, and the implementation of this work is based on the offline parameter tuning technique mentioned in [50] to find the optimal values of \(M\), \(W\) and \(L\).

Given a user latent factor vector \(w_u\), the LSH can quickly locate its buckets, and obtain the candidates by retrieving the items whose latent factor vectors are in these buckets. This work further uses multi-probe LSH [22] to intelligently probe multiple buckets that are likely to contain query results in a hash table, yielding a more efficient retrieval. The probing sequence for neighboring buckets is not randomly chosen but considers the distance to the original hashed bucket. Take Fig. 1 as an example, the star point
is hashed to bucket (0, 1) in the right side hash table, and we can also probe bucket (0, 2) since its margin is the most close to the query. Theoretically, the speedup of the retrieval is achieved since the number of items in these buckets is much smaller than the number of all items.

4. Experiments

We conduct experiments on three data sets, and compare the proposed algorithm with several algorithms. This work repeats each experiment ten times and uses the average and standard deviation of the results to present the experimental results, in which mean plus or minus two standard deviations is presented in the tables. Each evaluation randomly selects 20% of data for testing, and the remaining data for training. For Some performance metrics, such as coverage, recall@k and selectivity, they are computation intensive tasks, so the experiments randomly select 50 users in each testing set, denoted by \( \mathcal{U}_{\text{test}} \), to evaluate performance.

4.1. Data sets

This work conducts experiments on three data sets, and focuses on two-class problems, in which positive class implicitly indicates a user likes a specific item. Note that negative class does not mean that user dislikes it. In most cases, the number of positive classes is overwhelmed by negative ones, so we use sampling technique, of which the sampling probability for an item is proportional to its global popularity [51], to select partial negative samples to balance the two classes.

- Last.fm Data Set:
  This data is collected from Last.fm online music system, which comprises the listening habits for nearly 2000 users and 18,000 artists. The number of artist listening records is 93,000, and they are viewed as positive data. For the artists that are not the above case, we sample about 93,000 records as negative data. Besides user id and item id, this data set comprises 11,946 tag assignments of artists, which are also used in the experiments.

- Gowalla data set:
  Gowalla is a location-based social networking website where users share their locations by checking-in. The total number of check-ins for Gowalla data set [52] is about 6.3 million, which comprises 107,000 users and 1.3 million locations. We regard check-ins logs as positive data and the locations where visitors did not check in are viewed as negative data. As mentioned above, we sample approximately 6.3 million negative data for model training. Besides user id and item id, content features for each location, such as latitude and longitude, are used as item features in the experiments.

- Amazon data set:
  Amazon data set was collected by crawling Amazon website [53]. It is based on “Customers Who Bought This Item Also Bought” feature of the Amazon website. If a product \( i \) is frequently co-purchased with a product \( j \), the graph contains a directed edge from \( i \) to \( j \). It is apparent that this data set focuses on item-item relationship, so we view product \( i \) as a user and product \( j \) as an item. Note that user id is unavailable in this data set. This data set comprises around 335,000 products and 1 million co-purchasing records. We view the product pairs in co-purchasing records as positive data. Conversely, the product pairs that do not appear in the co-purchasing records are viewed as negative data. Similarly, we sample 1 million negative data from the negative data collection.

4.2. Evaluation metric

- Area under the curve (AUC):
  A receiver operating characteristics (ROC) curve is a plot used to illustrate the performance of a binary classifier system as its discrimination threshold is varied. In a ROC curve the true positive rate is plotted in function of the false positive rate for different cut-off points. To compare classifiers one may want to reduce ROC performance to a single scalar value representing expected performance. The AUC provides an alternative in evaluating classifier performance, in which its value is between 0 and 1. Larger AUC values indicate better classifier performance across the full range of possible thresholds. Many approaches have been proposed to calculate AUC, and we use the trapezoidal rule here. The AUC has an important statistical property, namely the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance as shown in Eq. (10), where \( i \) is an indicator function. More detailed introduction about ROC and AUC can refer to [54].

\[
\frac{1}{|\mathcal{X}_{\text{test}}|} \sum_{x^{(+), \mathcal{X}_{\text{test}}}} \sum_{x^{(-), \mathcal{X}_{\text{test}}}} [f_{\mathcal{D}}(x^{(+)}) > f_{\mathcal{D}}(x^{(-)})]
\]  

- Coverage:
  If we use \( Z \) to denote the set of available items and \( \text{Rec}@k(u) \) to denote the set of top-k ranked items for the given user \( u \). Given a test set \( \mathcal{U}_{\text{test}} \), a measurement for prediction coverage [9] can be given by:

\[
\frac{\left| \bigcup_{u \in \mathcal{U}_{\text{test}}} \text{Rec}@k(u) \right|}{|Z|}
\]

- Recall@k:
The proposed method recommends the items approximately, so we use recall to evaluate the quality of the fast recommendation result. Let \( u \) be the given user, \( \text{Rec}@k(u) \) be the set generated by the original recommendation, and \( \text{ApproxRec}@k(u) \) be the set generated by approximate recommendation. Mean Recall@k over the entire test set \( \mathcal{U}_{\text{test}} \) is defined as:

\[
\frac{1}{|\mathcal{U}_{\text{test}}|} \sum_{u \in \mathcal{U}_{\text{test}}} \frac{|\text{Rec}@k(u) \cap \text{ApproxRec}@k(u)|}{|\text{Rec}@k(u)|}
\]

- **Selectivity:**
  The recall of the proposed method requires to scan through the items in the candidate set, and compute their predicted scores to obtain the top \( k \) items. It is apparent that the prediction time is thus proportional to the size of the candidate set. Dong et al. [50] define selectivity as the size ratio between the candidate set \( C(u) \) and the set of whole items \( I \), and report its mean over the entire test set \( \mathcal{U}_{\text{test}} \). The formula is:

\[
\frac{1}{|\mathcal{U}_{\text{test}}|} \sum_{u \in \mathcal{U}_{\text{test}}} \frac{|C(u)|}{|I|}
\]

### 4.3. Evaluation results

This work uses six algorithms to compare with the proposed algorithm. The first algorithm is called “Random”, indicating that it randomly recommend items to the user. The second algorithm is called “Most Popular”, which recommends popular items to user. The 3rd–5th algorithms are the variants of FM [55], which combines the generality of feature engineering with the superiority of factorization models in estimating interactions between categorical variables of large domain. The libFM library\(^2\) is a software implementation for FM that features stochastic gradient descent (SGD) and alternating least-squares (ALS) optimization. We further refer to the implementation used by Wu et al. [56], who modified the objective function of FM and won the first prize of KDD Cup 2012 Track 2, and abbreviate the modified FM as “FM (NTU)”. The final one is SVDFeature [28], which has won KDD Cup for two consecutive years. This work sets the number of factors \( \omega \) as ten in the experiments.

The proposed method belongs to latent factor models, and it can incorporate LSH into the model to speed up item retrieval. For the experiments in this section, we only focus on the evaluation of the proposed LCR without using LSH. The experimental results are presented in Table 1, in which we focus on recommendation accuracy. The proposed method works well and outperforms the other alternatives. It is apparent that the “Random” algorithm fails to capture user’s behaviors. Recommending popular items is a straightforward approach, but the experimental results indicate that this approach fails to perform well on the three data sets. The other algorithms are latent factor models. Compared to these latent factor models, the proposed method can benefit from the proposed scoring function, optimization over ranking loss with \( \ell_p \) regularization term, and yield a better performance. Note that SVDFeature fails to perform well on Amazon data set, since this data set allows the same product to play different roles, including user and item. However, SVDFeature uses conventional user-item relationship to design model, and regards the same product in different roles as two kinds of objects to map them to two different latent factor vectors.

The second experiment focuses on the recommendation coverage. Figs. 2–4 show the experimental results, in which horizontal bar charts represent the coverage percentages on top-10 and top-20 recommendation items. Intuitively, the “Random” can cover the largest item sets, while the item coverage of “Most Popular” should be small. The experimental results conform to the intuition. The

\(^2\) libFM: http://www.libfm.org/.
recommendation of the proposed LCR can cover more items than the other latent factor models. The experimental results indicate that calculating the distance in Euclidean space achieve a higher coverage than those using inner products.

An important property of Euclidean embedding is that users and items are embedded in a unified Euclidean space, and the interpretation of user points and item points are the same, since the maximum rating can be reached when they are at the same point [49]. Therefore, the recommendation can be viewed as a nearest neighbors search problem. In a latent space, all the dimensions should comprise latent information. For example, in movie recommendation problems, the latent information could be interpreted as movie genre. Given a user data point, the item data points that are close to the user data point should share some latent information with the user data point in common, giving a base to obtain high coverage on recommendation results. In contrast, the inner product between two vectors q and x involves the length of the two vectors and the angle between q and x vectors. Thus, given a query data point q, one has to consider the length of x vector and the angle θ to determine the top k data points when using inner product in recommendation problems. The item points whose lengths are large have a good chance to become recommendation candidates, so using inner product as similarity metric tends to recommend some specific items.

### 4.4. The effect of fast recommendation

Besides recommendation quality, prediction time is always the most important issue in building a recommender systems. If the recommender systems fails to make recommendations in near real-time, the other quality issues will fail as well. This work considers the strength of latent factors and transforms the recommendation problem into a k-nearest neighbors search problem in the Euclidean space. We further use multi-probe LSH technique to accelerate nearest neighbor search. We conduct experiments to investigate the effects by using the proposed model with two schemes. The first scheme is multi-probe LSH, while the second one is brute-force search.

A high value for recall@k implies that the approximate recommendations are very similar to the original recommendations. The recall and retrieval time is a trade-off in approximate search. Moreover, a significant drawback of LSH is that its performance is very sensitive to the parameters, and tuning parameters for a given data set remains a tedious process. Dong et al. [50] proposed a performance model for multi-probe LSH. Given a small sample data set, their proposed model can accurately predict the average search quality and latency. Note that the performance model requires to give a required value for recall. The experiments set the recalls as 0.7 and 0.8, respectively. The experimental results are presented in Tables 2–4, which summarize the result of top-10 and top-20 recommendation. In the experiments, this work carries out each experiment ten times, but all experiments share the same parameter, which is obtained using the performance model on the first experiment. Thus, each Recall@k in Tables 2–4 approaches the required value instead of not always being larger than required one. Additionally, the experimental results indicate that a higher recall leads to a lower speedup, conforming to the design of approximate search algorithm. The experimental results indicate that the proposed model incorporating with multi-probe LSH can speed up the retrieval performance, giving 5–313 times faster than the baseline. Furthermore, selectivity indicator explains the reason why the speedup can be significant, since only a small portion of the data is searched to obtain the top k items. As the data set becomes larger, the selectivity is much more lower since LSH filters out most of the items. As a consequence, the speed-up on a large-scale data set, such as Gowalla, can be significant. Note that the coverage of the fast recommendation is almost the same as that of the brute force approach listed in Figs. 2–4. Furthermore, we conduct experiments on Last.fm and Amazon data sets with |Utest| = 100, and the experimental results are presented in Tables 5 and 6. The experimental results resemble those presented in the previous experiments, in which the proposed model incorporating with multi-probe LSH can speed up the retrieval performance.

### 4.5. The effect of content-features

As in collaborative filtering setting, the introduction of new users and new items leads to cold-start problem, since insufficient data can be used to work accurately. Conversely, content-based algorithms are based on item information and user profile, so they

---

**Table 2**

<table>
<thead>
<tr>
<th>Recall</th>
<th>Top k</th>
<th>Recall@k</th>
<th>Coverage (%)</th>
<th>Speedup</th>
<th>Selectivity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>10</td>
<td>0.7073 ± 0.0201</td>
<td>2.3915 ± 0.1982</td>
<td>9.9140 ± 2.5314</td>
<td>6.5263 ± 3.2646</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.6677 ± 0.0553</td>
<td>4.4427 ± 0.0365</td>
<td>8.8870 ± 2.4596</td>
<td>7.1786 ± 3.3835</td>
</tr>
<tr>
<td>0.8</td>
<td>10</td>
<td>0.8351 ± 0.0416</td>
<td>2.3064 ± 0.1573</td>
<td>6.4879 ± 2.4148</td>
<td>14.0341 ± 5.4349</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.8147 ± 0.0359</td>
<td>4.3463 ± 0.1929</td>
<td>5.4196 ± 1.9357</td>
<td>15.9811 ± 6.4921</td>
</tr>
</tbody>
</table>

**Table 3**

<table>
<thead>
<tr>
<th>Recall</th>
<th>Top k</th>
<th>Recall@k</th>
<th>Coverage (%)</th>
<th>Speedup</th>
<th>Selectivity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>10</td>
<td>0.7420 ± 0.1010</td>
<td>0.0362 ± 0.0017</td>
<td>313.8905 ± 56.3864</td>
<td>0.0602 ± 0.0123</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.7007 ± 0.1201</td>
<td>0.0705 ± 0.0019</td>
<td>286.1822 ± 15.7925</td>
<td>0.0654 ± 0.0119</td>
</tr>
<tr>
<td>0.8</td>
<td>10</td>
<td>0.8787 ± 0.0726</td>
<td>0.0390 ± 0.0001</td>
<td>280.9170 ± 165.6483</td>
<td>0.2624 ± 0.2213</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.8480 ± 0.0681</td>
<td>0.0768 ± 0.0017</td>
<td>265.6217 ± 145.3929</td>
<td>0.2732 ± 0.2205</td>
</tr>
</tbody>
</table>

**Table 4**

<table>
<thead>
<tr>
<th>Recall</th>
<th>Top k</th>
<th>Recall@k</th>
<th>Coverage (%)</th>
<th>Speedup</th>
<th>Selectivity (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>10</td>
<td>0.7280 ± 0.0418</td>
<td>0.1492 ± 0.0003</td>
<td>52.0737 ± 6.1795</td>
<td>0.4999 ± 0.3121</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.6760 ± 0.0727</td>
<td>0.2983 ± 0.0010</td>
<td>47.0362 ± 6.0529</td>
<td>0.5253 ± 0.2981</td>
</tr>
<tr>
<td>0.8</td>
<td>10</td>
<td>0.8313 ± 0.0569</td>
<td>0.1575 ± 0.0000</td>
<td>26.1614 ± 13.2057</td>
<td>3.6175 ± 1.6453</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.7931 ± 0.0479</td>
<td>0.3153 ± 0.0004</td>
<td>25.5252 ± 13.8326</td>
<td>3.6034 ± 1.6813</td>
</tr>
</tbody>
</table>
can use these available content to tackle cold-start problems. In most e-commerce websites, content information, such as user basic profile and item description, is always available. As mentioned above, the proposed method provides the flexibility to incorporate various features into the model to enhance prediction quality, and prevent cold-start problems.

This work conducts two experiments on Last.fm and Gowalla data sets to evaluate the effect of content-features. Note that Amazon data set does not provide additional content features, so this data set is absent from the experiments. Figs. 5 and 6 list the experimental results, which indicate that the proposed method can benefit from additional information. Besides, incorporating additional content features into the proposed model does not affect the results of coverage significantly.

5. Conclusion and further study

This work considers recommendation accuracy, coverage and prediction time issues to devise a latent factor model algorithm. The main ideas behind the proposed algorithm are to transform the recommendation problems into a nearest neighbor search problem and provide the flexibility to incorporate LSH into the model to speed up the search for the top k items. The proposed model provides an elegant combination with LSH to provide a fast recommendation while retaining recommendation accuracy and coverage. Additionally, the proposed method considers the practical system deployment issue to use stochastic gradient descent algorithm along with sampling technique to optimize the ranking loss. The future work is to incorporate fast recommendation into the model, and tune the LSH parameters systematically using a deeper theoretical analysis.

Acknowledgment

This work was supported in part by Ministry of Science and Technology, Taiwan, under Grant No. MOST 104-2218-E-009-028.

References


