Real-time recommendation with locality sensitive hashing

Article *in* Journal of Intelligent Information Systems · August 2019

DOI: 10.1007/s10844-019-00552-1

2 authors, including:

Maruf Aytekin
Bahçeşehir University
2 PUBLICATIONS **4** CITATIONS

SEE PROFILE

Real-time Recommendation with LSH manuscript No.
(will be inserted by the editor)

Real-time Recommendation with Locality Sensitive Hashing

Ahmet Maruf Aytekin · Tevfik Aytekin

Received: date / Accepted: date

Abstract Neighborhood-based collaborative filtering (CF) methods are widely used in recommender systems because they are easy-to-implement and highly effective. One of the significant challenges of these methods is the ability to scale with the increasing amount of data since finding nearest neighbors requires a search over all of the data. Approximate nearest neighbor (ANN) methods eliminate this exhaustive search by only looking at the data points that are likely to be similar. Locality sensitive hashing (LSH) is a well-known technique for ANN search in high dimensional spaces. It is also effective in solving the scalability problem of neighborhood-based CF. In this study, we provide novel improvements to the current LSH based recommender algorithms and make a systematic evaluation of LSH in neighborhood-based CF. Besides, we make extensive experiments on real-life datasets to investigate various parameters of LSH and their effects on multiple metrics used to evaluate recommender systems. Our proposed algorithms have better running time performance than the standard LSH-based applications while preserving the prediction accuracy in reasonable limits. Also, the proposed algorithms have a large positive impact on aggregate diversity which has recently become an important evaluation measure for recommender algorithms.

Ahmet Maruf Aytekin
Department of Computer Engineering, Bahçeşehir University, Çırağan Caddesi, 34353, Beşiktaş, İstanbul, Turkey
Tel.: +90-212-444-2864
Fax: +90-212-381-0020
E-mail: ahmetmaruf.aytekin@stu.bahcesehir.edu.tr

Tevfik Aytekin
Department of Computer Engineering, Bahçeşehir University, Çırağan Caddesi, 34353, Beşiktaş, İstanbul, Turkey
Tel.: +90-212-444-2864
Fax: +90-212-381-0020
E-mail: tevfik.aytekin@bahcesehir.edu.tr

Keywords Recommender systems · Locality sensitive hashing · Real-time recommendation · Scalability

1 Introduction

Recommender systems analyze users' past activities and help users to find the most relevant items or information that they are searching for [1]. Content-based filtering and CF are the two main approaches that are used by recommender systems. In content-based filtering, the content of items is used to build a predictive model. For example, in a movie recommender system the contents of movies (such as the genre, director, stars, and summary text) that a user likes are used to build a user preference model of movies [30]. On the other hand in CF, ratings of users on items are used to make predictions for unknown ratings (implicit feedback such as watching duration or number of clicks can also be used). CF is divided into two broad categories as neighborhood-based and model-based methods. In neighborhood-based CF, user and item ratings are directly used to predict the unknown ratings. This CF method will be explained in Section 2. On the other hand, model-based CF methods use the user and item ratings to learn a predictive model which is used to predict ratings [1, 2, 4]. For example in matrix factorization approaches, low dimensional latent vector representations of each user and item are learned and these vector representations are used to make a prediction for user i on item j by taking the dot product of user i's and item j's vectors [12, 13, 14]. Neighborhood-based approaches gained much popularity because they are easy to implement and give moderately accurate results [5]. However, they suffer from scalability problems, because they need to search all the users (or items) in the system to find the nearest neighbors to make a prediction [2].

Today's online systems produce massive amounts of data which potentially has millions of users and items. CF algorithms need to process all the data to find k-nearest-neighbors to make predictions and recommendations. ANN search methods provide a solution to this problem. ANN methods eliminate searching all of the data to find the nearest neighbors, but only search for the users (or items) that are likely to be similar. ANN methods such as clustering and LSH have been used in the recommendation domain to improve the real-time performance of recommender algorithms and overcome the scalability issues [6, 7, 8]. In general, ANN methods cluster users (or items) to narrow down the search space, but similarity computation is still required to find k-nearest neighbors in a cluster. In the worst case, the similarity calculations depend on the number of users or items which can be inefficient for on-line systems that require real-time processing of massive amounts of data. An LSH based algorithm, which is a type of ANN method, also provides a way to retrieve k-nearest-neighbor set without calculating similarity for the target user (or item). The algorithm retrieves a list of candidate users for a target user with LSH and counts the frequency of the users (co-occurrence with the target user) in buckets, then select the users who have a frequency higher than a pre-

configured threshold to build k-nearest-neighbor set [7, 8]. Therefore, LSH-based algorithms are employed to build highly scalable recommender systems. Moreover, LSH based algorithms are straightforward algorithms to implement and preserves the advantages of neighborhood-based CF such as simplicity and justifiability.

It has been recognized that accuracy is not the only dimension for evaluating recommender systems. One other dimension is the diversity of recommendation lists [3, 19]. Beyond accuracy, it is also important to recommend a diverse set of items to users since recommending accurate but very similar items will make it difficult for the users to discover new items. Another type of diversity is called aggregate diversity which refers to the diversity of items across all users lists [27]. This type of diversity is especially important from the business point of view since companies, in general, would be able to recommend all the items in their product catalog. In Section 6.1 we will define the metrics for measuring these two types of diversity and in Section 6.5 we will show how our methods have a positive effect on diversity.

Our contribution in this paper is twofold. First is an extensive evaluation of LSH parameters. Although LSH is used in the recommendation domain in the past, to the best of our knowledge, there has been no systematic evaluation of its configuration parameters. We investigate the boundaries of LSH configuration parameters and their effect on the recommendation performance and diversity metrics. This extensive evaluation is essential because tuning the LSH parameters by only considering accuracy might lead to abysmal results for other metrics (e.g., diversity) which will result in a poorly functioning recommender system. Our second contribution is to provide an improvement to the current LSH methods and experimentally show that the improved version has a better performance than the current methods with a tolerable decrease in accuracy and also improves aggregate diversity to a great extent.

This paper is organized as follows. Section 2 gives an introductory background. Section 3 gives the details of the LSH method. Section 4 explains how LSH can be used for CF. Section 5 describes our proposed algorithms and gives algorithmic details. Section 6 describes evaluation metrics, experimental techniques, and results. Finally, we provide concluding remarks in Section 7.

2 Background

CF can be defined as the problem of estimating a user's response to a new item that the user has not seen before by using historical user-item ratings stored in the system. In a general recommendation scenario, user ratings on various items are collected, and CF is used to generate rating predictions for unknown items. Users query the recommendation engine for an item, which is unknown to the user, to get a rating prediction. Another typical usage is to provide users with a list of highest predicted N items which is referred to as top-N recommendation [9]. Neighborhood-based CF can be done in two ways, user-based (UB) and item-based (IB). In UB-CF, the critical idea is that users

with similar tastes are likely to behave similarly. Likewise, in IB-CF the idea is that if two items are similar items, a user will likely rate these items in a similar way [4, 10].

In order to describe a CF problem, we introduce the following notation. R will denote user-item rating dataset in the recommendation system. The set of users and items in R will be denoted by U and I respectively. We assume that any user $u \in U$ gives only one rating to a specific item $i \in I$ and this rating is denoted by r_{ui}. In addition, the subset of users that have rated an item i is denoted by U_i; the subset of items that have been rated by a user u is denoted by I_u and the rest of the items, unknown items to u, is denoted by $I_u^{\complement}$. We also use I_{uv} for the item set that is rated by users u and v, where $I_{uv} = I_u \cap I_v$, and U_{ij} for the user set that have rated both items i and j, where $U_{ij} = U_i \cap U_j$.

In UB-CF, the basic methodology to predict user u's rating, r_{ui}, on an item i is as follows. The algorithm first finds the k most similar users (k-nearest-neighbors) to user u that have rated item i, then uses the ratings of these k-nearest-neighbors on item i to generate a prediction for user u on item i. The simplest approach is to take the average of these ratings for making a prediction. Other approaches are possible, such as taking a weighted average by similarity to the target user or by averaging only the ratings of the nearest neighbors for which the confidence level in similarity to target user is high, confidence in this context is measured by the number of common ratings used in measuring the similarity of two users [4]. Similar to UB methodology, in IB-CF, k-nearest-neighbors to item i which user u rated are found and their ratings are used to predict r_{ui}, such as by taking the average as in UB-CF [10]. We refer to these algorithms as UBP-CF and IBP-CF respectively and use them as baselines. The "P" in the names "UBP" and "IBP" denotes that these are rating *prediction* algorithms. In order to make top-N recommendation in CF, one approach is to predict u's ratings for the unknown items, $I_u^{\complement}$, then recommend the highest predicted N items to u. We use these algorithms as baselines for top-N recommendation and refer to them as UBR-CF and IBR-CF for user-based and item-based methods, respectively. The "R" in the names "IBR" and "UBR" denotes that these are top-N *recommendation* algorithms.

The main problem with CF methods is their scalability and real-time performance due to their computational complexity for k-nearest-neighbor search. The computational complexity of k-nearest-neighbor search increases linearly with the number of users in the system which can grow to be millions. In the case of pre-computing user-to-user similarity matrix in an off-line step (model building), computational complexity grows quadratically [4]. Another issue with pre-computing user-to-user similarity matrix is that it limits the constant addition of the new data to the model. Similar problems persist with IB recommendation technique as well [4, 10].

Researchers developed a variety of model-based recommendation techniques to solve the scalability issues of recommender systems. Some researchers used matrix factorization [12, 13, 14] and others treated the top-N recommendation problem as a classification problem. They developed a model-based rec-

ommender system to classify unrated items into two or more classes, such as like and dislike. Artificial neural networks are also used to learn a personalized model for each user and predict the missing values for the user [15]. Map-reduce, the distributed CF algorithms, and generic full-text search engines are also some other techniques that are used to build scalable, high-performance recommender systems [6, 16, 17, 18].

Clustering and LSH are used as ANN search methods to build scalable recommender systems [6, 7, 8]. In LSH, items are hashed to buckets such that similar items have a higher probability of being mapped to the same bucket than dissimilar items. LSH employs locality sensitive hash functions, which will be explained in the next section, to measure the similarity between the items. In this way, LSH maps similar items (or users) to the same buckets and narrows down the search space to a candidate set. Then, UB-CF or IB-CF is applied to the candidate set. However, since the candidate set is usually much smaller than the number of items, LSH is much faster than traditional UB-CF and IB-CF. In [7] LSH is used to generate personalized recommendations with CF methods. LSH with UB-CF generates recommendations for a target user as follows. It first clusters the users (places them in buckets) then selects the users that are in the same clusters with the target user. As the next step, it calculates the number of clicks given to the news story by the users in these clusters. As the last step, these click frequencies are combined to compute a recommendation score. In [8] LSH is used as an ANN method to provide real-time recommendations for a target user as follows. A list of candidate users for a target user with LSH is retrieved, and the frequency of the users, based on co-occurrence with the target user, is counted. As the following step, the users who have a higher frequency than a pre-configured threshold are selected to build a candidate user set. Then, the candidate user set is used to form a candidate item set for the target user by retrieving the items that candidate users rated. As the last step, the subset of candidate user set, who have rated on the items in the candidate item set, is used to predict ratings for the target user and top-N predicted items are recommended to the target user. We call this method frequency-based LSH since it selects users in the candidate set with respect to their frequencies. Even though frequency-based LSH reduces the amount of pairwise comparison with LSH, the frequencies of users still need to be computed, and a sorting operation should take place to find the users with the highest frequencies which can be expensive especially when the candidate set is large.

3 Locality Sensitive Hashing

In neighborhood-based CF, we need k-nearest-neighbors of users (or items) to make rating predictions. Classical neighborhood-based CF algorithms investigate every user (or item) in the dataset to find k-nearest-neighbors which does not scale well for large datasets. One approach to remedy this scalability problem is to use an approximate method, such as LSH, to find nearest neigh-

bors. The idea in LSH is to hash the items several times with hash functions (locality-sensitive functions) in a way that similar items have more chance to be hashed to the same bucket than different items [20, 21, 22, 29]. In this context, two items that are hashed to the same bucket are considered as a candidate pair.

In the LSH scheme, locality-sensitive functions take two items and decide whether or not they are a candidate pair. A locality sensitive function, h, hashes data points x and y to $h(x)$ and $h(y)$. A decision will be rendered based on whether the results are equal or not. It is convenient to use the notation $h(x) = h(y)$ to mean that "x and y are a candidate pair" and $h(x) \neq h(y)$ to mean that "x and y are not a candidate pair." A group of functions of this type constitutes a family of locality-sensitive functions which is denoted by $\mathcal{H}$ [21].

LSH family of functions can be defined as follows [21]. Let $d_1 < d_2$ be two distances for some distance measure d. A family of functions, $\mathcal{H}$, is said to be (d_1, d_2, p_1, p_2)-*sensitive* for d, if for every h in $\mathcal{H}$:

- If $d(x, y) \leq d_1$, then the probability that $h(x) = h(y)$ is at least p_1
- If $d(x, y) \geq d_2$, then the probability that $h(x) = h(y)$ is at most p_2

where p_1 and p_2 are probability values and $p_1 > p_2$ for the LSH method to be useful.

Previous researchers have proposed LSH family of functions (schemes) for a variety of similarity measures including Jaccard similarity, hamming distance, Euclidean distance, cosine similarity, and kernelized similarity [21]. We use cosine similarity to measure the distance between two data points because we represent the user and item data points as rating vectors. We use the proposal in [23] for cosine similarity to describe LSH scheme on R^z, i.e., a collection of vectors in rating dataset R. Let $\mathbf{u}$ and $\mathbf{v}$ be the rating vectors of user u and v respectively and $\mathbf{r}$ be a z-dimensional random vector whose components are -1 and +1. A random hyperplane based hash function $h_{\mathbf{r}}$ which uses $\mathbf{r}$ to hash input vectors is defined as follows.

$$h_{\mathbf{r}}(\mathbf{u}) = \begin{cases} 1 & \text{if } \mathbf{r} \cdot \mathbf{u} \geq 0 \\ 0 & \text{if } \mathbf{r} \cdot \mathbf{u} < 0 \end{cases} \tag{1}$$

Then for vectors $\mathbf{u}$ and $\mathbf{v}$,

$$Pr[h_{\mathbf{r}}(\mathbf{u}) = h_{\mathbf{r}}(\mathbf{v})] = 1 - \frac{\theta(\mathbf{u}, \mathbf{v})}{\pi}, \tag{2}$$

where $Pr[h_{\mathbf{r}}(\mathbf{u}) = h_{\mathbf{r}}(\mathbf{v})]$ is the probability of declaring users u and v as a candidate pair and $\theta(\mathbf{u}, \mathbf{v}) = \cos^{-1} \frac{u \cdot v}{\|u\| \|v\|}$. The dot '$\cdot$' in the equation represents dot product of two vectors.

Using this random hyperplane based hash function, we obtain a family of hash functions, $\mathcal{H}$, for cosine similarity. $\mathcal{H}$ also incorporates element weights in the similarity calculation since the values of the coordinates of the vectors are weighted rating values. In the next section, we describe how to construct a

new family of hash functions from $\mathcal{H}$ and apply LSH to CF methods. We will also define the process of applying LSH to neighborhood-based CF and give the details of the current LSH algorithms used in the recommendation domain and discuss their efficiency and drawbacks.

4 LSH for Collaborative Filtering

LSH provides a way to retrieve the approximate nearest neighbors for a target user or item which is referred as the candidate set C. LSH algorithms search only the set C to find k-nearest-neighbors. In order to use the LSH algorithm with CF, one needs to build an LSH model. LSH model building step requires configuration of the model and creation of hash functions from the LSH family. The LSH algorithm has two main parameters that need to be configured, namely, the number of hash tables, L, and the number of hash functions, κ. Once the parameters are configured, LSH algorithm constructs a new family of functions, $\mathcal{H}'$, from κ members of $\mathcal{H}$ by using *AND-construction* on the members of $\mathcal{H}$. In $\mathcal{H}'$ family, if h is a member of $\mathcal{H}'$ and constructed from the set $h_1, h_2, ..., h_\kappa$ of members of $\mathcal{H}$, u and v become candidate pair, $h(u) = h(v)$, if and only if $h_i(u) = h_i(v)$ for all $i = 1, 2, ..., \kappa$. After constructing $\mathcal{H}'$ family, the algorithm generates a hash function, h', from $\mathcal{H}'$ family for each hash table. Then, the algorithm hashes each data point (user or item vector) with h' for a hash table and creates a hash key (bucket id) for the hash table. Next, the algorithm adds the data point to the hash table based on the created hash key. The data points, which share the same hash key in the hash table, map to the same bucket and become candidate pairs. In this step, m and n rating vectors are hashed for UB-LSH and IB-LSH methods respectively and added to the buckets for each hash table, where $m = |U|$ and $n = |I|$. The algorithm repeats this procedure for L hash tables to build the LSH model [20, 21].

In order to avoid recalculation of hash keys during the recommendation computations, we create a hash key lookup table to store the user (or item) id and associated hash keys for each hash table. The computational complexity of the LSH model building is $O(mL\kappa t)$ and $O(nL\kappa t)$ for UB-LSH and IB-LSH respectively, where t is the time to evaluate a hash function. The space complexity of the model becomes $O(2mL)$ and $O(2nL)$ for UB-LSH and IB-LSH methods respectively, because a model and a hash key lookup table needs to be maintained that each requires $O(mL)$ or $O(nL)$ space for UB-LSH or IB-LSH methods, respectively.

In the LSH model, two users u and v become a candidate pair if they are hashed to the same bucket for at least one of the hash tables. The set of all users map to the same bucket in all hash tables with user u forms the candidate set, C, which is the set of approximate nearest neighbors of u. C is defined as $C = c_1 \cup c_2 \cup .... \cup c_L$, where c_j is the set of candidates that mapped to the same bucket with u in j_{th} hash table. $\mathcal{H}'$ family of functions generate 2^κ buckets at most. If we assume that the data points are uniformly distributed over the buckets, the average size of the candidate set c for one hash table

can be calculated as $|c| = m/2^\kappa$ and $|c| = n/2^\kappa$ for UB-LSH and IB-LSH methods, respectively. The candidate set size for UB-LSH is $|C| \leq Lm/2^\kappa$ and $|C| \leq Lm/2^\kappa$ for IB-LSH. In this model, most of the different pairs do not hash to the same bucket in any of the hash tables. Therefore, they are not checked during the similarity calculations. Most of the similar pairs are mapped to the same bucket under at least one of the hash tables, and they are checked during the similarity calculations. Those different pairs that are hashed to the same bucket in at least one of the hash tables are *false positives* and they are only a small fraction of all pairs. The similar pairs that do not map to the same bucket in any of hash tables are *false negatives* which are only a small fraction of truly similar pairs [21, 25].

UBP-LSH algorithm operates on the model to predict user u's rating, r_{ui}, on item i as follows. The algorithm first looks up the hash key for u from the hash key lookup table and uses the hash key to retrieve the candidate set, c, from each hash table, then builds candidate set, C. The algorithm then computes the similarity of u with all users in C to find the k-nearest-neighbors. Finally, it uses k-nearest-neighbors to predict r_{ui} with UB-CF algorithm. The computational complexity of UBP-LSH prediction becomes $O(L + |C|n + k)$, where $|C| \leq Lm/2^\kappa$, because it looks at L hash tables to build C and computes target users similarity with all users in C to find k-nearest-neighbors. Then it uses k nearest neighbors to compute the prediction for the target user. In this calculation, the cost of computing similarity or distance of two users is assumed as n. A similar approach is used to make a prediction with IBP-LSH as well. IBP-LSH algorithm constructs the candidate set C for item i and searches C for the k-nearest-neighbors of i, then predicts r_{ui} with IB-CF algorithm. The computational complexity of IBP-LSH prediction becomes $O(L+|C|m+k)$, we assume the cost of similarity or distance of two items as m and $|C| \leq Ln/2^\kappa$.

[8] and [7] use LSH with frequency-based approach to compute prediction scores and generate recommendations. We generalize and study this approach for user-based (UB-LSH1) and item-based (IB-LSH1) methods. UBP-LSH1 algorithm predicts user u's rating, r_{ui}, on an item i as follows. The algorithm retrieves a list of candidate users, C_l, for a target user with LSH and counts the frequency of the users (co-occurrence with the target user), then selects the users who have a frequency higher than a pre-configured threshold to build a candidate user set. It then uses the subset of the candidate user set who have rated on item i and their frequency as a weight to calculate a prediction for u. The computational complexity of this algorithm becomes $O(L + |C_l| + |C_l|lg(|C_l|)+k)$, where $C_l = Lm/2^\kappa$. To predict user u's rating, r_{ui}, on an item i, IB version of this approach, IBP-LSH1, can be used as follows. The algorithm retrieves a list of candidate items, C_l, for i and counts the frequency of items in the candidate list. Then it takes the list of items that have a frequency higher than a pre-configured threshold and computes the prediction for u by using IB-CF algorithm. Similar to UBP-LSH1 algorithm, the computational complexity of IBP-LSH1 algorithm becomes $O(L + |C_l| + |C_l|lg(|C_l|) + k)$, where $|C_l| = Ln/2^\kappa$.

To recommend top-N items to user u, [7] uses LSH with frequency-based approach which is denoted by UBR-LSH1. The algorithm finds the set of candidate users, C, for u with LSH and retrieves the items that each user in C has rated and adds them to a running list, C_l. It then calculates the frequency of the items in C_l and recommends the most frequent top-N items which are not in I_u. LSH algorithm checks L hash tables to find the candidate set for u, and each hash table returns approximately $m/2^\kappa$ candidate users, therefore candidate set size, $|C|$, becomes $Lm/2^\kappa$. In addition, the size of $|C_l|$ becomes $pLm/2^\kappa$ in the worst case, where p is the number of items in I_u. The algorithm counts the frequency of the items in C_l to build a frequent items list, then sorts the frequent items list based on the frequency and takes the most frequent top-N items. The computational complexity of this algorithm becomes $O(L + |C| + |C_l| + |C_l|lg(|C_l|)$ in the worst case. In a similar fashion, the item-based version of this algorithm (IBR-LSH1) operates as follows. The algorithm first finds the candidate set for each item user u has rated and builds a list of candidate items, C_l. Then, it computes the frequency of the items in C_l and takes the most frequent top-N items to recommend. The size of C_l becomes $Lpn/2^\kappa$ in the worst case, where p is the number of items in I_u. The computational complexity of this algorithm consists of finding the candidates of each item, counting the frequency of items in C_l and sorting C_l which becomes $O(pL + |C_l| + |C_l|lg(|C_l|))$.

Recommender systems need to handle the constant addition of data which is typically observed in real-time applications. CF methods need to rebuild the model each time a new data is added to be able to use the new data in recommendations. The model building step can be inefficient if data is significant because the model building of CF methods requires quadratic time complexity (Fig. 1). Therefore model building step is usually executed as a nightly batch. Because of these limits, CF methods cannot handle constant additions of new data efficiently.

LSH model building needs to be executed once, and it has linear time complexity (Fig. 1). Once the LSH model is built, the new data can be added to the model by only computing the hash key for the user or item that corresponds to the new data. There is no need to rebuild the entire model. If a new user or item arrives at the system, it is hashed and added to L hash tables. If the ratings of an existing user or item are updated, corresponding user or item is re-hashed, and L hash tables are updated with the new hash value. The computational complexity of updating the model is $O(pL\kappa t)$, where p is the number of items a user rates and t is the time to evaluate a hash function.

5 Improved LSH Algorithms

Top-N recommendation algorithms, UBR-LSH1 and IBR-LSH1, described in the previous section have a significant drawback. UBR-LSH1 algorithm finds the candidate user set, C, for u with LSH and retrieves the item list that each user in C rated to build candidate item list C_l. Even though LSH limits the

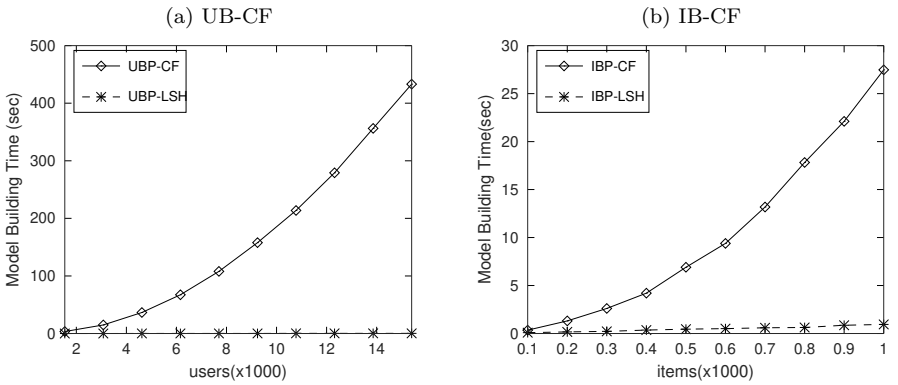

Fig. 1: Model building time for UB-CF, IB-CF, and LSH (Yahoo! Music)

number of users in C to a constant value, the number of items that users rated can be substantial depending on the sparsity level of the dataset[1]. The size of $|C_l|$ becomes $|C|n$ in the worst case. The algorithm counts the frequency of C_l and sorts it to take top-N most frequent items which has $O(L+|C|n+nlg(n))$ complexity, $|C|n$ comes from counting the frequency of items and $nlg(n)$ comes from sorting the items based on frequency. We can eliminate the count and sort steps of these algorithms by selecting N items from C_l randomly and recommending them to u. The idea of this approach is that more frequent items in C_l have more chance than less frequent items to be selected randomly. We denote this approach by UBR-LSH2. The complexity of this algorithm becomes $O(L+|C|+N)$, because it searches L hash tables for candidate users, retrieves $|C|$ users' items to build C_l, and selects N random items to recommend.

IBR-LSH1 algorithm finds the candidate set for each item user u has rated and builds a list of candidate items, C_l. The number of items in $|C_l|$ can be large, depending on the number of items u has rated. The size of $|C_l|$ becomes $Ln^2/2^\kappa$ in the worst case. IBR-LSH1 finds the candidate set of the items u rated, counts the frequency of items in C_l, and sorts them based on the frequency. Counting C_l takes $|C_l|$ time and sorting takes $|C_l|lg(|C_l|)$ time. Similar to UBR-LSH2, we can improve the performance of this algorithm by eliminating the count and sort steps by retrieving N items from $|C_l|$ randomly and recommending them to u. In this approach (denoted by IBR-LSH2), more frequent items in C_l have a higher chance than less frequent items to be selected for recommendation. The complexity of this algorithm becomes $O(nL+N)$, because it retrieves candidate set for all items in I_u, potentially n items, from L hash tables and randomly selects N items. However, the actual complexity is significantly smaller, because each user has a relatively small number of ratings, much less than n. Furthermore, the complexity of these algorithms in practice is $O(n)$ since L and N are small constants.

[1] sparsity of a dataset $= \#ratings/(\#items * \#users)$

Similar to top-N recommendation algorithms, prediction algorithms, UBP-LSH1 and IBP-LSH1, can be inefficient if the dataset is dense, because candidate set list, C_l, becomes large. The algorithms need to count the frequency of the items and sort the top k users (or items) based on the frequency. We eliminate frequency count and sorting steps of these algorithms in our proposed approaches which are referred as UBP-LSH2 and IBP-LSH2. Instead, we choose the candidates randomly as in UBR-LSH2 and IBR-LSH2. The computational complexity of the proposed UBP-LSH2 algorithm is $O(L + 2k)$, because the algorithm checks L hash tables for candidates and selects k random users then uses k selected users to compute the prediction. Similar to UBP-LSH2 algorithm, IBP-LSH2 algorithm finds candidate item list, C_l, which is rated by u, for the target item i and predicts the rating for u on i by using IB-CF. The complexity of this method is $O(L + 2k)$, because the algorithm checks L hash tables for candidates and selects k random items, then uses k selected items to compute the prediction. UBP-LSH2 and IBP-LSH2 are highly scalable methods because the complexity of these algorithms depends on L and the k which are both constants.

In the following section, we present a detailed experimental evaluation of LSH algorithms, investigate their properties, and show that they are highly scalable.

6 Empirical Analysis

In this section, we give the details of evaluation metrics, datasets used in the experiments, and experimental methods. We then present various configuration parameters and discuss how LSH solves the stability issue of CF methods. In the end, we present the experimental results on the datasets.

6.1 Evaluation Metrics

In this section, we give a brief description of the metrics we use to evaluate the algorithms. We use accuracy and running time performance to evaluate the prediction and top-N recommendation performance. We also use the metrics commonly used in the recommendation domain such as diversity and novelty for evaluating top-N recommendation performance.

6.1.1 Prediction Accuracy

Prediction accuracy metrics measure the deviation between the predicted ratings and actual ratings of a user on unknown items. We use Mean Absolute Error (MAE) metric to measure the prediction accuracy of the algorithms. We use Eq. 3 to compute MAE, where the algorithms predict r'_{ui} for user-item pairs (u, i) in the test set that the actual ratings, r_{ui}, are known.

$$MAE = \frac{1}{\mid R_{test} \mid} \sum_{r_{ui} \in R_{test}} \mid r'_{u,i} - r_{ui} \mid \tag{3}$$

6.1.2 Coverage

The coverage of a recommender system is the percentage of the total number of items that a recommender system can form a prediction for. If the candidate set, C, is empty then an LSH based CF cannot make a prediction or top-N recommendation. The size of C depends on κ and L. Therefore we run experiments to find out how κ and L effect the coverage.

6.1.3 Precision

Most recommendation systems recommend a list of items to the users as a top-N recommendation list from best to worst. To measure the success of a recommendation algorithm in terms of relevance to the users' preferences, we use mean average precision (MAP) which is a widely used metric for evaluating top-N recommendation performance. MAP at N is defined as in Eq. 4:

$$MAP@N = \frac{1}{|U_t|} \sum_{u \in U_t} \frac{1}{|I_u|} \sum_{i}^{N} P(i) * R(i) \tag{4}$$

where U_t is the set of users in the test set, $P(i)$ is the precision for user u at the ith position of the top-N list, $R(i)$ is a binary indicator which returns 1 if the ith item is relevant or 0 otherwise, and $|I_u|$ is the number of items of user u in the test set.

6.1.4 Diversity

A recommender system should be able to recommend diverse items to help the users for knowledge discovery [3, 19, 24]. This kind of diversity is called individual diversity. We slightly modified [24]'s intra-list similarity formulation to compute intra-list dissimilarity. Let $L_N(u)$ be the list of top-N items recommended to user u and $CosSim(i,j)$ is the similarity between i and j, the individual diversity for $L_N(u)$ is calculated as in Eq. 5.

$$Diversity = \frac{1}{N * (N-1)} \sum_{i \in L_N(u)} \sum_{j \in L_N(u), i \neq j} (1 - CosSim(i,j)), \tag{5}$$

where we divide intra-list dissimilarity with number of dissimilarity computations, $\frac{N*(N-1)}{2}$, to make sure the average diversity is in interval [0,1].

There is another type of diversity called aggregate diversity which measures the number of unique items recommended to the users also known as the catalog coverage [2, 27]. This kind of diversity is important from a business

point of view. A low aggregate diversity means most of the items in the catalog are not recommended to users. We use the formula proposed by [27] and given in Eq. 6 to measure aggregate diversity which counts the unique number of items recommended across all recommendation lists.

$$Aggregate\text{-}diversity = \left| \bigcup_{u \in R_{test}} L_N(u) \right|. \tag{6}$$

6.1.5 Novelty

Novelty can be described as the ability of a recommender system to introduce users to items that they have not experienced before [24]. We measure novelty with the metric defined in Eq. 7 which is previously introduced by [28].

$$Novelty = \frac{1}{\mid L_N(u) \mid} \sum_{i \in L_N(u)} log_2(m/|U_i|), \tag{7}$$

where $L_N(u)$ is the set of top-N items recommended to user u, m is the number of users in the system, and U_i is the set of users who have rated item i.

6.2 Datasets

We use three different datasets to evaluate the algorithms and show the efficiency of the improved algorithms. The characteristics and rating distributions of the data sets are shown in Table 1.

The first dataset is Movielens (ML-1M) dataset referred to as ML-1M. The second dataset is Yahoo! Music[2] rating dataset. For the last dataset we used Amazon Movies and TV[3] [26]. In order to improve sparsity level, we removed the users that have less than 20 ratings and items that have less than 10 ratings from Amazon Movies and TV dataset.

dataset	# users	# items	# ratings	sparsity
ML-1M	6040	3706	1000209	0.047
Yahoo! Music	15400	1000	311704	0.0202
Amazon Movies and TV	16042	17454	698091	0.0025

Table 1: Characteristics of the datasets used in evaluating the algorithms.

[2] http://research.yahoo.com/Academic_Relations
[3] http://jmcauley.ucsd.edu/data/amazon/

6.3 Experimental Technique

In the experiments, we used nested 10-fold cross-validation. The optimum configuration parameters of CF and LSH are determined on the validation sets, and the algorithms are evaluated on the test sets. The running time we measure for the prediction task is the time required to make a prediction for a user on an item. In order to evaluate the algorithms regarding top-N recommendation performance, we select the users from the test set who have a minimum of 50 ratings and randomly hide ten items that each user has rated high, i.e., the user rated higher than the user's average rating. Then we make top-N recommendations to these users using the model built on the training set. We measure the performance of top-N recommendation algorithms in terms of commonly used metrics in the recommendation domain, namely, precision, diversity, and novelty. We also measure the performance of top-N recommendation algorithms which is the time required to make a top-N recommendation list to a single user.

6.4 Configuration Parameters

Configuration parameters have a significant effect on the performance of the algorithms. We run multiple tests on validation sets to determine the impact of the parameters and find the optimum parameters of the algorithms. We give the details of these parameters in the following subsections.

6.4.1 Collaborative Filtering Parameters

k is the number of nearest neighbors used in the prediction calculation in CF methods. k is data dependent and optimum k needs to be determined by cross-validation [9]. In our experiments, we run validation tests to detect the optimum k for each dataset. In the validation tests, prediction accuracy is stabilized around 20 and then did not change much (see Fig. 2) . Therefore, for all datasets, we set k to 20 for the rest of the experiments

In CF methods, if similarity weights are computed using only a few ratings, that may lead to poor predictions and recommendations. In this case, the magnitude of the similarity needs to be reduced by the factor of a given threshold, Y. We use [4]'s approach to reduce the user and item similarity weights. We run the validation experiments on the datasets and found out that CF algorithms perform best when Y is around 5 (see Fig. 3). Therefore, we set Y to 5 for the rest of the experiments.

6.4.2 Number of Hash Functions and Tables

The optimal value of L and κ should also be determined by cross-validation. We run cross-validation experiments to determine the optimal L and κ as

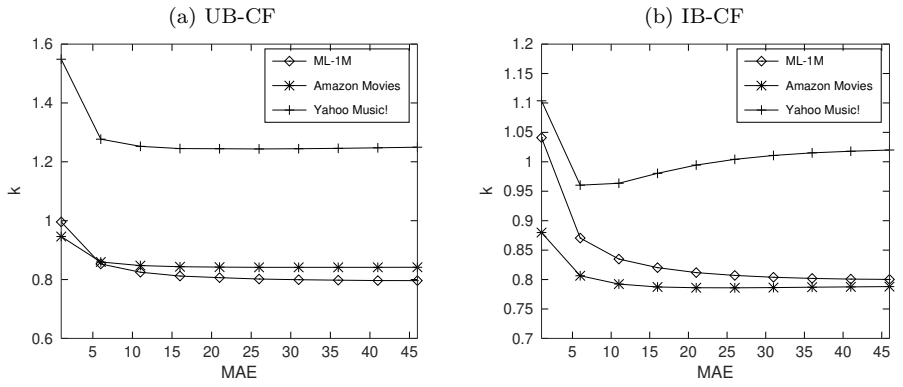

Fig. 2: MAE as a function of k.

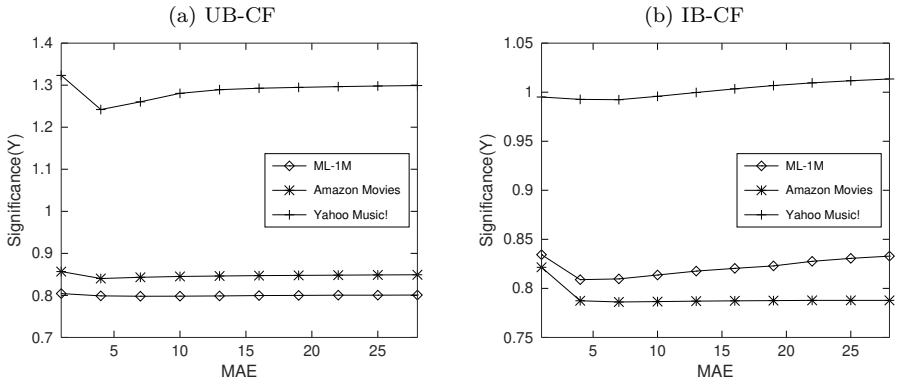

Fig. 3: MAE as a function of significance, Y.

follows. We vary L and κ from 1 to 10 and perform 10×10 experiments to find the optimum values and boundaries of these parameters.

In Fig. 4 we see that when we increase κ, prediction accuracy gets worse in LSH methods, but running time gets better. The accuracy gets worse because the candidate set size decreases with the decrease of κ and this results in the decrease o true positives. When we increase L, prediction accuracy gets better but running time gets worse. The reason is that the size of C increases; hence the number of true positives increases. To select the optimum configuration for L and κ, we look for the regions that maximize prediction accuracy and minimize the running time. Approximately all data sets provide reasonably good results with the configuration where $L = 5$ and $\kappa = 6$. Therefore, we use these same values for all datasets to simplify the experiments. Also, note that in Fig. 4 the decrease in prediction accuracy, as κ increases, is more pronounced for the Amazon Movies and TV dataset. The reason for this behavior is the sparsity level of the Amazon Movies and TV dataset (which is the most

sparse among the three datasets). This high sparsity causes a faster decrease in the candidate set size which can also be observed in Fig. 4. The decrease in candidate set size directly affects the accuracy results since when candidate set size decreases true positives also decrease.

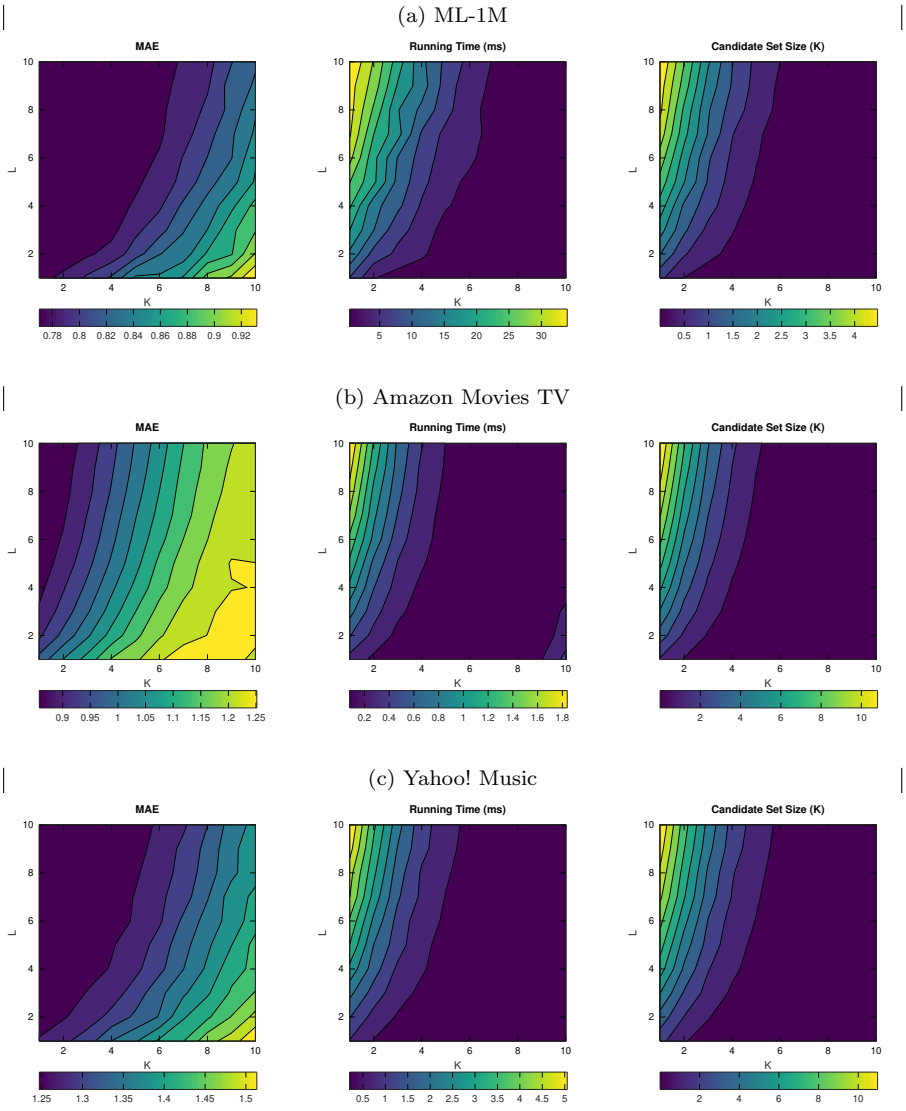

Fig. 4: MAE, running time, and candidate list size as a function of κ and L for LSH.

The lower bound of κ is 1 for $\mathcal{H}'$ family of functions because at least one hash function is needed to build an LSH model. In this case, there are only two buckets generated and the average number of data points in these buckets are $m/2$ for UB-LSH and $n/2$ for IB-LSH. When we increase κ, eventually, each user or item maps to one bucket, where $2^\kappa \leq m$ for UB-LSH and $2^\kappa \leq n$ for IB-LSH.

6.5 Experimental Results

Experiments are performed using three different datasets as we describe in Section 6.2. All experiments are performed on a T2 xlarge AWS EC2 instance with 4vCPU and 16GB of memory. The algorithms are executed in Java 1.7.0_171 OpenJDK 64-Bit Server VM in a single thread with no parallelism. In the following sections, we give the results of the experiments and discuss the evaluation results.

We have run multiple tests to measure the model building time for the algorithms. We also investigate the effects of different LSH parameters, κ and L, on the performance of LSH algorithms. We set L to the optimum value and increase only κ starting from 1 to 10 to measure the effect of κ for the prediction and top-N recommendation algorithms. We run the same tests to find out the effects of changes in L. We set κ to the optimum value and increase L from 1 to 10.

6.5.1 Prediction Accuracy and Time Efficiency

We perform a number of tests to find out the effect of LSH parameters L and κ and evaluate LSH based methods. We first run a test to detect the effect of increasing κ. We set L to the optimum value and start κ from 1 and increase by one at each iteration up to 10. Then we measure the prediction accuracy and running time for the algorithms. We perform these experiments using 10-fold cross-validation and average out the results over the rounds. We set κ to the optimum value and run similar tests to detect the effect of increasing L.

As can be seen in Fig. 5 when we increase L, prediction accuracy, indicated by MAE, is improved for all methods. The reason for this improvement is because as we increase L, $|C|$ increases and false negatives are reduced. In other words, all the nearest neighbors are retrieved along with some false positives, but these false positives are eliminated during the nearest neighbor search or random selection in C. In contrast to increasing L, increasing κ worsens prediction accuracy, because of the number of false negatives increases. In other words, some of the nearest neighbors are missed. It can be observed that IB methods for the Yahoo! Music dataset are drastically affected by the increase in κ. The reason for this behavior is that in Yahoo! Movies dataset the number of items is limited and for larger values of κ there will be very few items in each bucket. For example, when κ=8, assuming an even distribution, there will be $1000/2^8 \approx 4$ items in each bucket. So the prediction is forced to

be done with very few neighbors. Also, for all the LSH based algorithms, we can observe that there are no drastic changes in MAE due to a change in the parameters κ and L. This implies that parameters of LSH based algorithms can be tuned smoothly to adjust the trade-off between accuracy and performance.

When it comes to time efficiency, we see that (Fig. 6) our proposed methods have a clear advantage even compared to frequency-based LSH algorithms. These results show that the proposed methods are highly scalable. We do not include UBP-CF and IBP-CF in Fig. 6 since LSH-based algorithms are much faster than pure CF algorithms and showing the results of UBP-CF and IBP-CP would make it impossible to see the differences among LSH-based methods.

In general, there is a trade-off between running time and accuracy. LSH methods are approximation methods which are used to speed up the nearest neighbor search. They achieve this speed up by finding approximate results which in turn leads to a decrease in accuracy. The critical point here is that the resulting decrease in accuracy should be in a tolerable range. Our proposed LSH2 methods are no exception here. They further speed up the recommendation time but also lead to a decrease in accuracy in some instances. However, as the results show this decrease is tolerable. Furthermore, as we will discuss in Section 6.1.4 the randomization process in LSH2 models gives an essential boost to aggregate diversity.

6.5.2 Prediction Coverage

Fig. 7 shows the prediction coverage results. As expected L is direct whereas κ is inversely proportional to prediction coverage. This is because, as depicted in Fig. 4, while increasing L leads to an increase in the candidate set size, increasing κ leads to a decrease. It is not possible to make a prediction for users/items whose candidate set size is zero. In Fig. 7a and 7b we notice that prediction coverage in Amazon Movies and TV dataset is worse than the others. The reason for this result is the fact that Amazon Movies and TV dataset is much sparser than the other two datasets. If a dataset is sparse, the candidate set size will tend to be smaller since the neighbors of users/items will also be sparse. Hence it becomes difficult to find neighbors who rated the same items with the target user. We also notice that this situation is changed in Fig. 7c and 7d. In these figures, the worst prediction coverage appears in the Yahoo! Music dataset. Note that in these figures IB-LSH is used. The reason for this is that not only sparsity but also the number of users and the number of items effect prediction coverage. Compared to Amazon Movies and TV dataset, Yahoo! Music dataset has much less number of items. Therefore, this increases the probability of two users rating the same item which in turn increases the chance of finding similar users to the target user who rated the same item as the target user. So although Yahoo! Music dataset has high prediction coverage in UB-LSH, it has low prediction coverage in IB-LSH. Note that since the results we get from *-LSH1 and *-LSH2 methods are

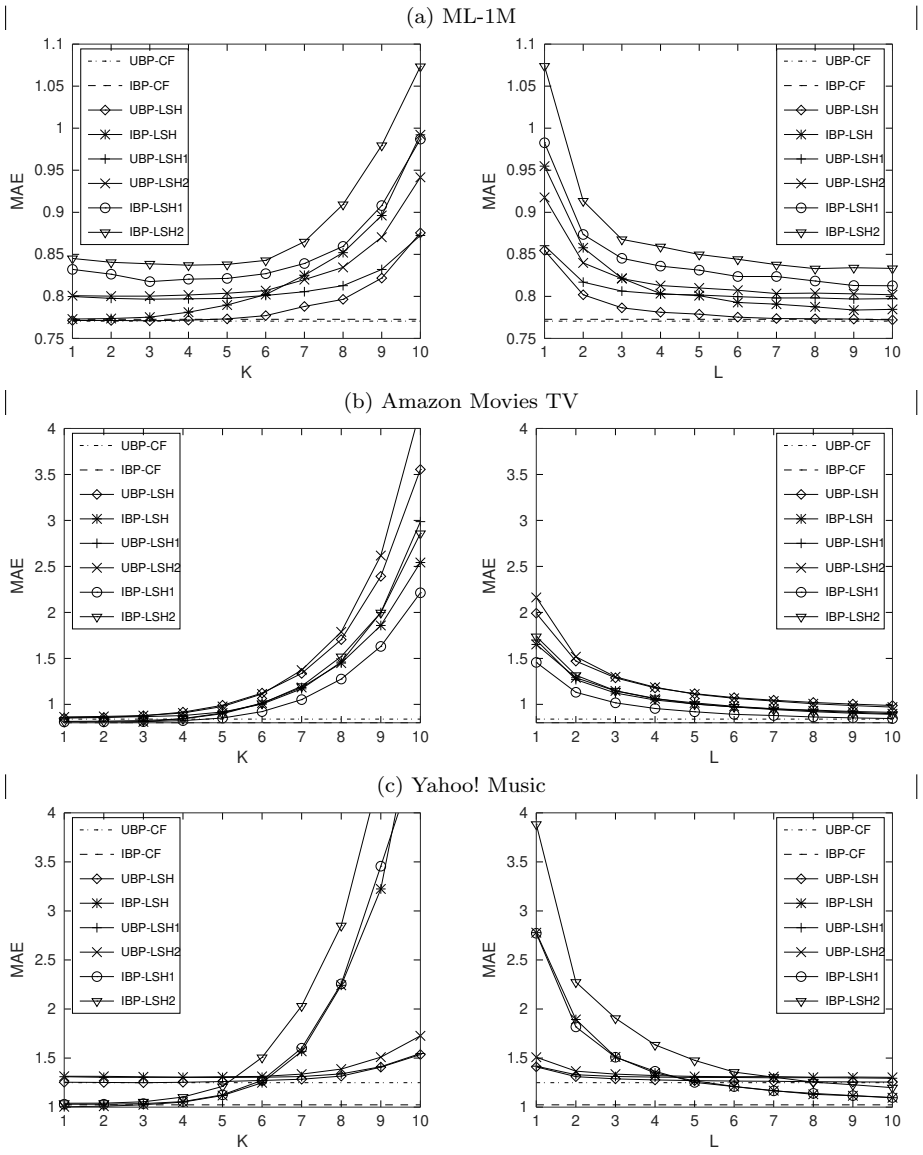

Fig. 5: Prediction algorithms - MAE as a function of κ and L.

almost exactly the same, in Fig. 7 we only show the results of UB-LSH and IB-LSH methods.

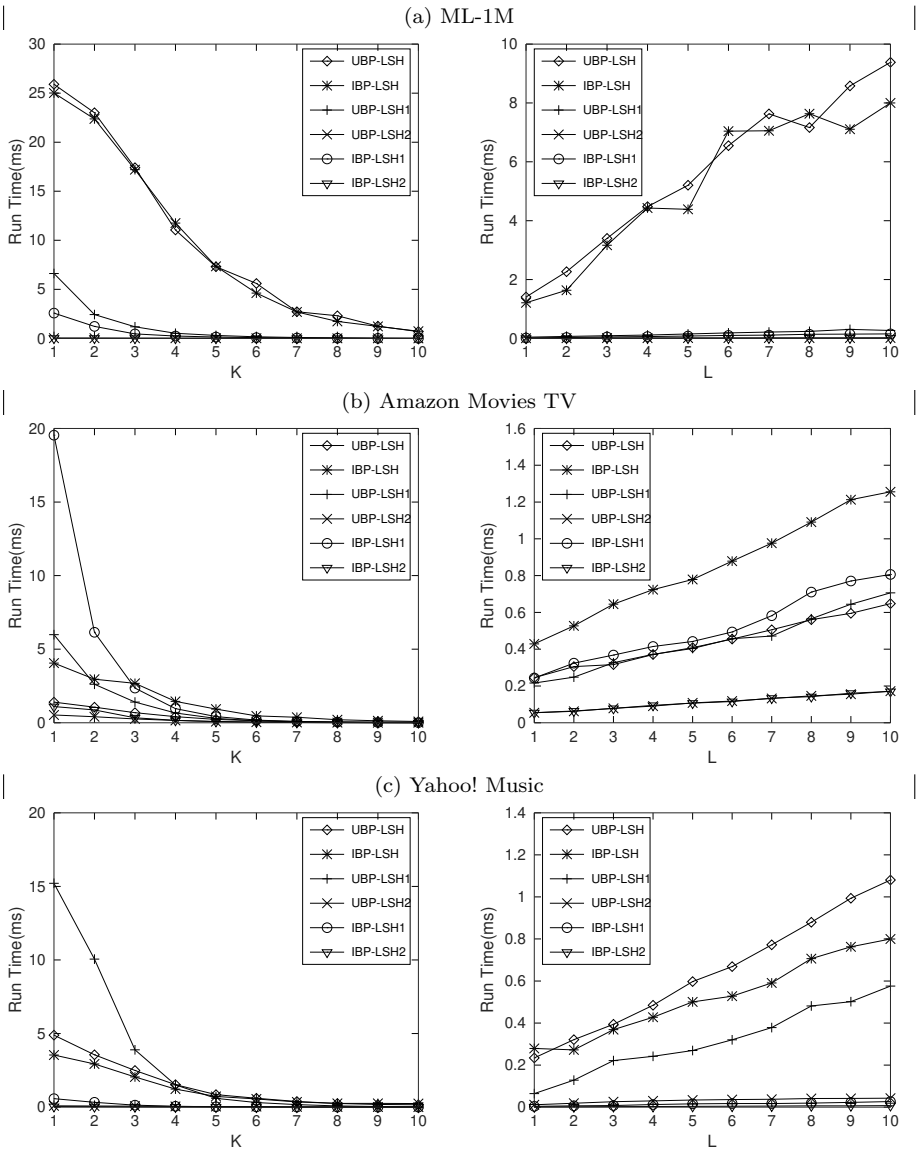

Fig. 6: Prediction algorithms - Running time as a function of κ and L.

6.5.3 Top-N Recommendations

During the evaluation tests, we found out that increasing κ, contrary to having a negative effect on MAE, has a positive effect on precision (Fig.8a). Elimination of false positives (which is larger in number than eliminated true positives) might be the cause of this positive effect. On the other hand, UBR-LSH2 and

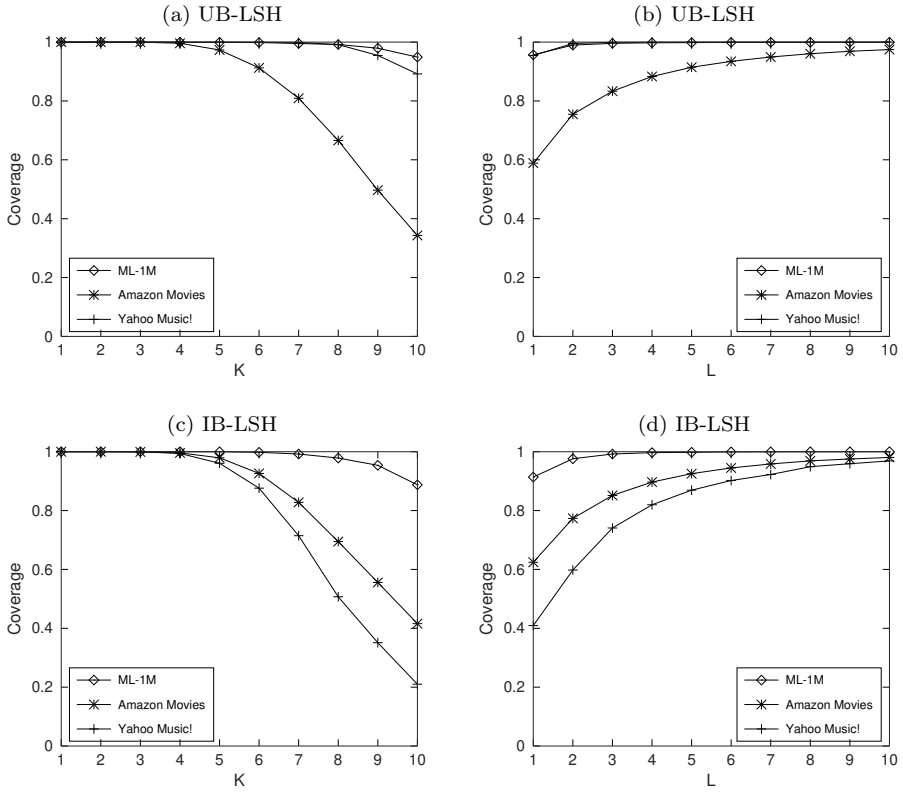

Fig. 7: Prediction Coverage for LSH methods ($L = 5$ and $\kappa = 6$ used in these experiments)

IBR-LSH2 methods have higher efficiency in terms of running time performance (Fig. 9).

For UBR-LSH1 and IBR-LSH1 methods precision slightly increases when we increase L (Fig. 8), because increasing L increases the number of true positives in the candidate list. False positives also increase, but these are eliminated by frequency sorting. Increasing L does not affect precision much for UBR-LSH2 and IBR-LSH2 methods because random selection is not enough to eliminate false positives.

6.5.4 Diversity

The evaluation tests for diversity and aggregate diversity are given in Fig. 10 and Fig. 11. Among the algorithms, we have found no significant differences in diversity. However, we can say that LSH-based methods keep the diversity levels of the recommendation lists. When we look at the aggregate diversity results, we have found that UBR-LSH2 and IBR-LSH2 methods manage to

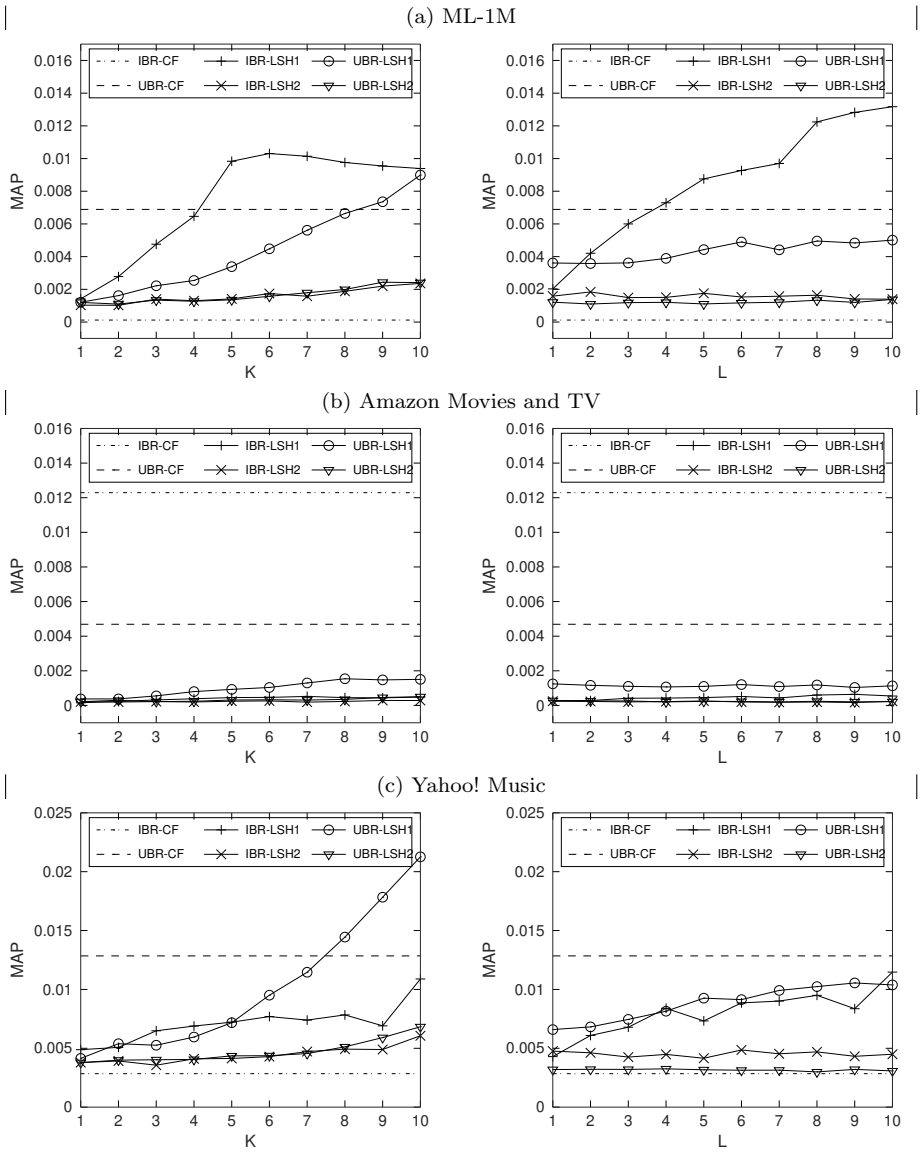

Fig. 8: Recommendation algorithms - MAP as a function of κ and L.

recommend almost all the items for all values of κ and L in all the three datasets. We interpret these results as follows: in order to increase time efficiency, the proposed methods randomly select items from the candidate set. This random selection process leads to the selection of different items and hence increases aggregate diversity. However, as we have discussed above since nearest items appear more often in the candidate list, the random selection

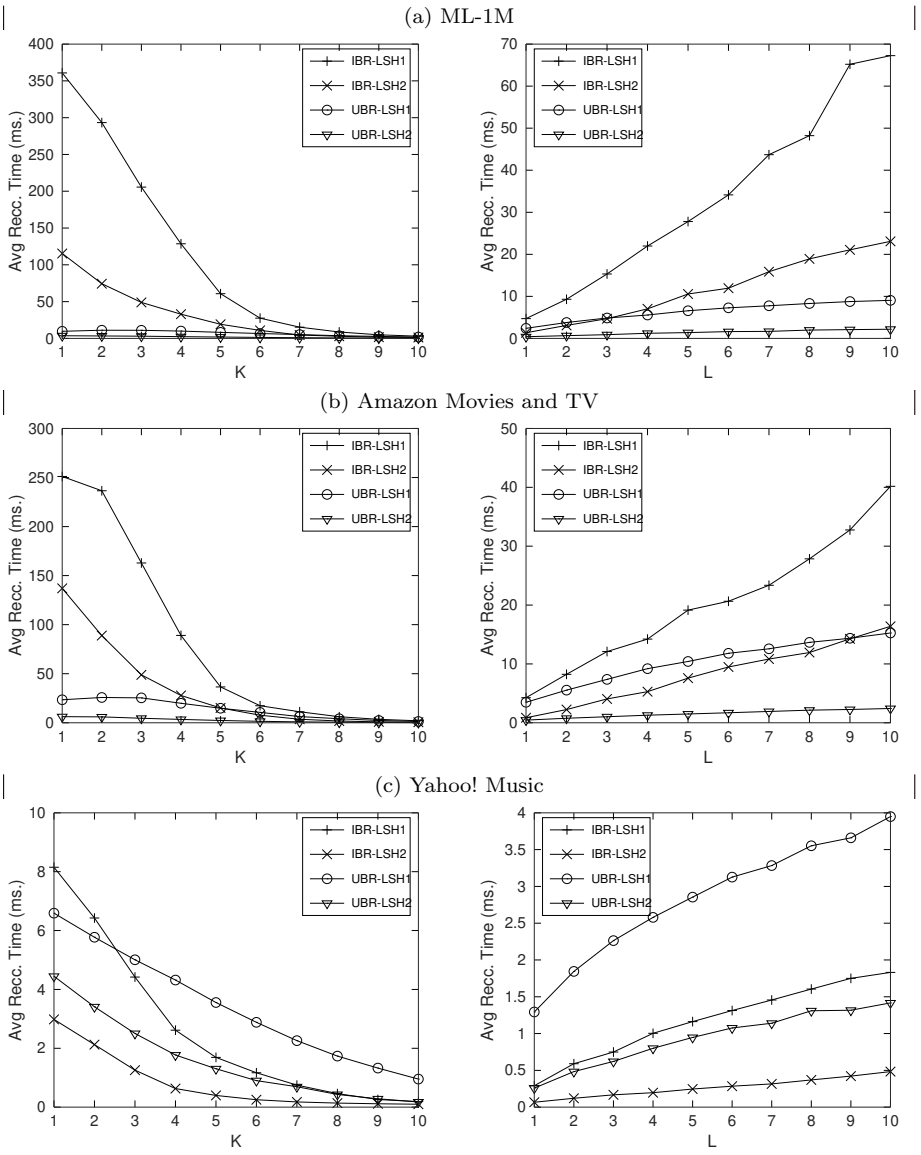

Fig. 9: Recommendation algorithms - Running time as a function of κ and L..

process also keeps to a certain extent the accuracy of the recommendations. Hence we can say that our proposed algorithms can also be used as an effective method for increasing the aggregate diversity of recommendation lists which is an important issue in recommender systems.

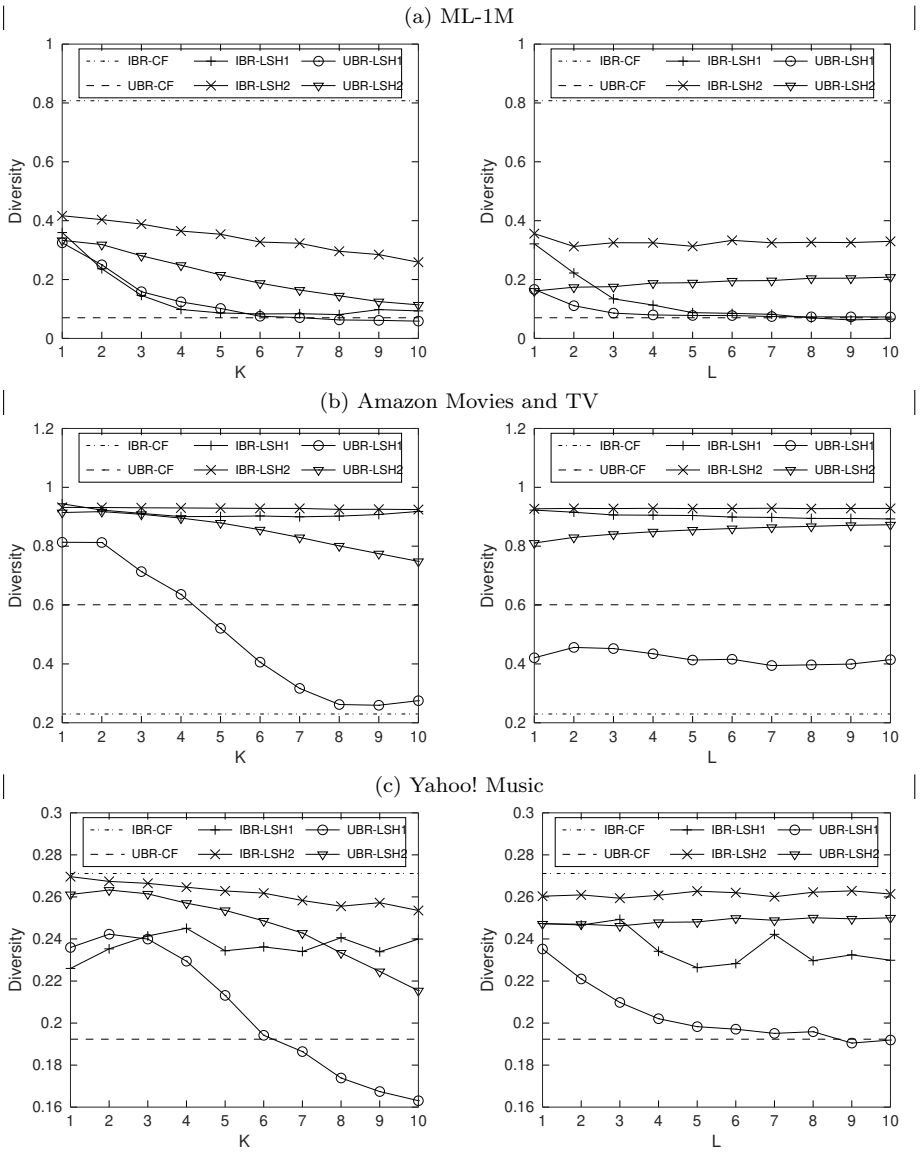

Fig. 10: Recommendation algorithms - Diversity as a function of κ and L.

6.5.5 Novelty

The novelty metric defined in Eq. 7 assumes that novelty increases when unpopular items (i.e., items in the long tail) are recommended to users. We see in Fig. 12 that LSH-based methods do not have a significant effect on novelty. In some datasets, they increase novelty, but in some others, they either do not

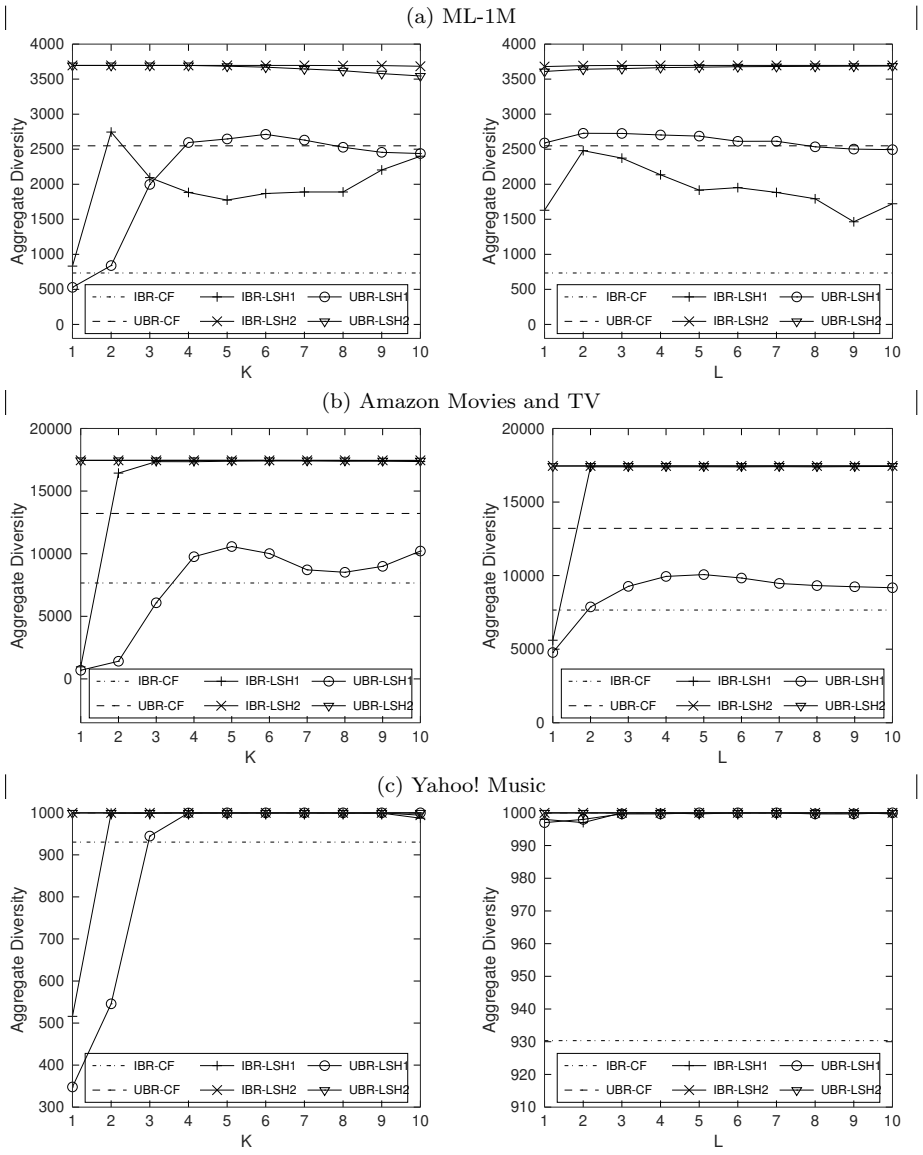

Fig. 11: Recommendation algorithms - Aggregate Diversity as a function of κ and L.

increase or slightly decrease novelty. This result is because the items in the candidate set do not differ much with respect to popularity.

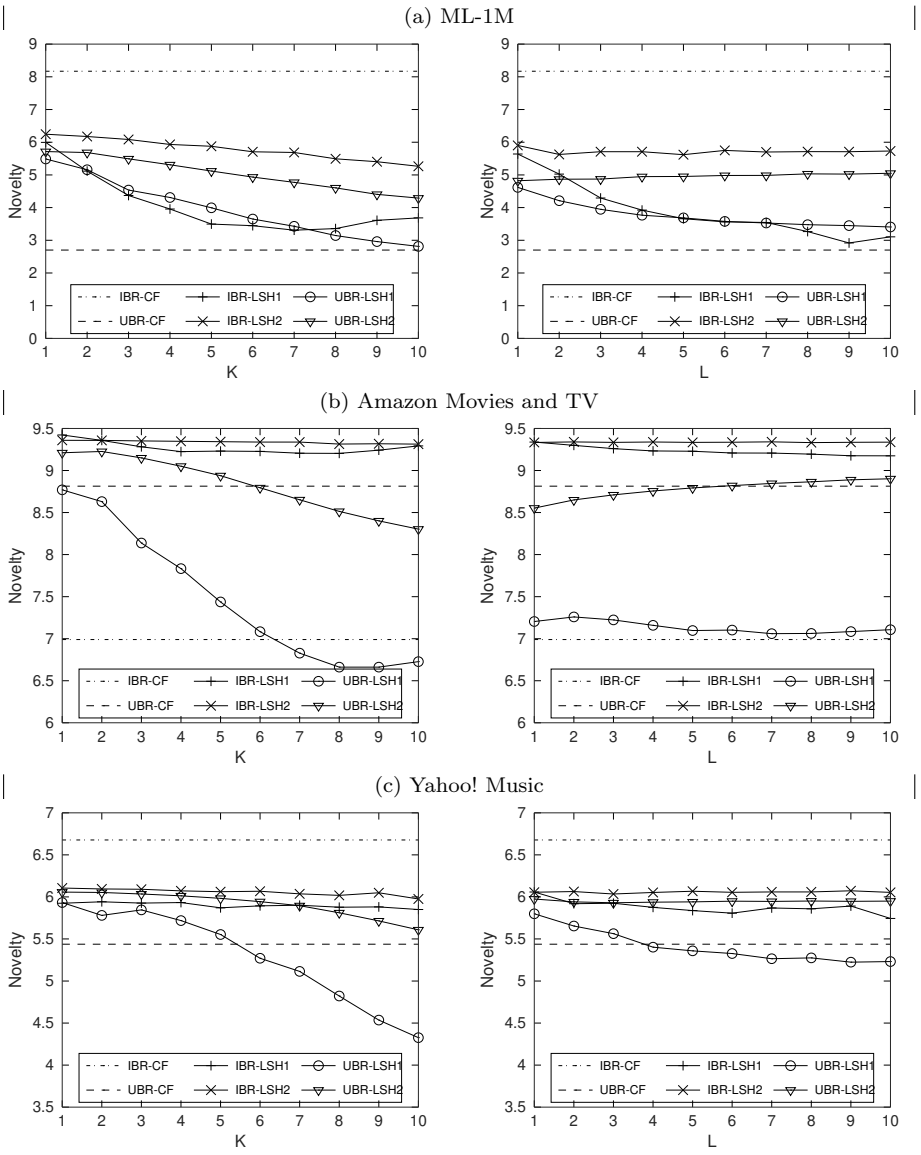

Fig. 12: Recommendation algorithms - Novelty as a function of κ and L.

7 Conclusion and Future Work

In this paper, we introduce a highly scalable neighborhood-based CF algorithm based on LSH which can handle massive amounts of data while preserving accuracy. Furthermore, it preserves the advantages of neighborhood-based CF such as simplicity and justifiability.

During the evaluation tests, our results show that the parameters of LSH are highly decisive on the outcome of LSH algorithms. They can be tuned for the model to trade off speed and accuracy and configured to provide an optimum outcome that balances accuracy and performance according to expectations from the system.

Aggregate diversity is an essential measure for evaluating the performance of recommendations, especially for business needs. Experiments revealed that our proposed algorithms have a positive effect on the aggregate diversity of recommendations. This feature is an additional benefit to these algorithms.

We observed that LSH solves the issue of handling the constant addition of new data to the system very efficiently. In LSH, when an update (new data) arrives at the system, it can be added to the model by only computing the hash key for the user or item that corresponds to update. There is no need to build the entire model. Hashing the new data occurs in real-time, and new data becomes available for use in recommendations.

In the future, we are planning to evaluate LSH for streaming algorithms for recommendations. We believe that LSH can be used in streaming algorithms very efficiently because LSH model permits the incremental updates to the model and does not require rebuilding of the entire model to make the updates available for use.

To the best of our knowledge, there is no examination of the differences between clustering techniques (such as k-means) and LSH for increasing scalability in recommender systems. Therefore, this is another area we plan to work in the future.

Acknowledgements This research was supported by Central Securities Depository Institution (MKK) of Turkish Capital Markets.

References

1. Ekstrand MD, Riedl J, Konstan JA, Collaborative Filtering Recommender Systems, Foundations and Trends in Human-Computer Interaction, 4, 2, 175–243, 2011
2. Herlocker JL, Konstan JA, Terveen LG, Riedl J, Evaluating collaborative filtering recommender systems, ACM Trans. Inf. Syst., 22,1,5–53, 2004
3. Aytekin T, Karakaya MO, Clustering-based diversity improvement in recommendation, J. Intell. Inf. Syst.,42,1,1–18, 2014
4. Desrosiers C, Karypis G, A Comprehensive Survey of Neighborhood-based Recommendation Methods, Recommender Systems Handbook, 107–144, Springer, 2011
5. Cacheda F, Carneiro V, Fernández D, Formoso V, Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems, TWEB, 5, 1, 2, 2011
6. Rashid AM, Lam SK, LaPitz A, Karypis G, Riedl J, Towards a Scalable k NN CF Algorithm: Exploring Effective Applications of Clustering, Advances

in Web Mining and Web Usage Analysis, WebKDD 2006, Philadelphia, PA, USA, pp 147–166
7. Das A, Datar M, Garg A, Rajaram SS, Google news personalization: scalable online collaborative filtering, Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, pp 271–280
8. Liang, Huizhi, Du, Haoran, Wang, Qing, Real-time Collaborative Filtering Recommender Systems, Proceedings of the 12nd Australasian Data Mining Conference (AusDM), 2014
9. Herlocker JL, Konstan JA, Riedl J, An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms, Inf. Retr., 5, 4, pp 287–310, 2002
10. Deshpande M, Karypis G, Item-based top-N recommendation algorithms, ACM Trans. Inf. Syst., 22, 1, pp 143–177, 2004
11. Karypis G, Evaluation of Item-Based Top-N Recommendation Algorithms, Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 247–254, 2001
12. Yu HF, Hsieh CJ, Si S, Dhillon IS, Scalable Coordinate Descent Approaches to Parallel Matrix Factorization for Recommender Systems, 12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, pp 765–774
13. R. Kannan, M. Ishteva, H. Park, Bounded matrix factorization for recommender system, Knowledge and information systems, vol. 39, no. 3, pp. 491-511, 2014
14. Zhao X, Niu Z, Chen W, Shi C, Niu K, Liu D, A hybrid approach of topic model and matrix factorization based on two-step recommendation framework, J. Intell. Inf. Syst., 44, 3, 335–353, 2015
15. Billsus D, Pazzani MG, Learning Collaborative Information Filters, Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, pp 46–54
16. Gong S, A Collaborative Filtering Recommendation Algorithm Based on User Clustering and Item Clustering, JSW, 5, 7, 745–752, 2010
17. Jiang J, Lu J, Zhang G, Long G, Scaling-Up Item-Based Collaborative Filtering Recommendation Algorithm Based on Hadoop, World Congress on Services, SERVICES 2011, Washington, DC, USA, pp 490–497
18. Suchal J, Návrat P, Full Text Search Engine as Scalable k-Nearest Neighbor Recommendation System, Artificial Intelligence in Theory and Practice III - Third IFIP TC 12 International Conference on Artificial Intelligence, IFIP AI 2010, Brisbane, Australia, pp 165–173
19. Shani G, Gunawardana A, Evaluating Recommendation Systems, Recommender Systems Handbook, 257–297, Springer, 2011
20. Gionis A, Indyk P, Motwani R, Similarity Search in High Dimensions via Hashing, VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases (1999), Edinburgh, Scotland, UK, pp 518–529

21. Rajaraman, Anand, Ullman, Jeffrey David, Mining of Massive Datasets, 73–126, Cambridge University Press, New York, NY, USA, 2011
22. Bahmani B, Goel A, Shinde R, Efficient distributed locality sensitive hashing, 21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, 2174–2178, ACM, 2012
23. Charikar M, Similarity estimation techniques from rounding algorithms, Proceedings on 34th Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 380–388, ACM, 2002
24. Zhang YC, Ó Séaghdha D, Quercia D, Jambor T, Auralist: introducing serendipity into music recommendation, Proceedings of the Fifth International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, WA, USA, 13–22, ACM, 2012
25. Andoni A, Indyk P, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, Commun. ACM, 51, 1, 117–122, 2008
26. McAuley JJ, Targett C, Shi Q, Hengel AVD, Image-based Recommendations on Styles and Substitutes, CoRR, abs/1506.04757, 2015
27. Adomavicius G, Kwon Y, Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques, IEEE Trans. Knowl. Data Eng., 24, 5, 896–911, 2012
28. Zhou T, Kuscsik Z, Liu JG, Medo M, Wakeling JR, Zhang YC, Solving the apparent diversity-accuracy dilemma of recommender systems, Proceedings of the National Academy of Sciences, 107, 10, 4511–4515, 2010
29. Koga H, Ishibashi T, Watanabe T, Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing, Knowl Inf Syst 12(1): 25–53, 2007
30. Pazzani, M. J., Billsus, D, Content-based recommendation systems, In The adaptive web, Springer, 325–341, 2007