PBloofI: An Enhanced Version of BloofI in Recommender Systems

Zahra Farahi*, Ali Moeini, Ali Kamandi, Mahmood Shabankhah, Seyed Mohsen Hosseini
School of Engineering Science
College of Engineering
University of Tehran
Tehran, Iran
{zhr.farahi, moeini, kamandi, shabankhah, mohsen.hosseini72}@ut.ac.ir

Received: 2019/04/15 Revised: 2019/06/05 Accepted: 2019/07/16

***Abstract*—In this paper, we focus on improving the performance of recommender systems. To do this, we propose a new algorithm named PBloofI which is a kind of hierarchical bloom filter. Actually, the Bloom filter is an array-based technique for showing the items' features. Since the feature vectors of items are sparse, the Bloom filter reduces the space usage by using the hashing technique. And also, to reduce the time complexity we used the hierarchical version of bloom filter which is based on B+ tree of order d. Since Bloom filters can make a tradeoff between space and time, proposing a new hierarchical Bloom filter causes a remarkable reduction in space and time complexity of recommender systems. To increase the accuracy of the recommender systems we use Probabilistic version of hierarchical Bloom filter. By measuring the accuracy of the algorithm we show that the proposed algorithms not only decrease the time complexity but also have no significant effect on accuracy**

Keywords—Recommender Systems, Bloom Filter, Hierarchical Bloom filter

1. INTRODUCTION

Due to the explosive growth of information available on the web, recommender systems have become widely utilized by businesses such as Amazon, Netflix, MovieLens or etc. Modeling the underlying mechanisms that predicts users' behavior on a particular item at a particular time could be useful for increasing users' engagement or selling items or advertisement.

Recommendation models advise users on which items they are more likely to be interested in. These models are usually classified into three categories: collaborative filtering, content based and hybrid recommender system.

Collaborative filtering [2, 31] recommends items to users based on their historical interactions with other items. Suppose someone has been rated some movies, then collaborative filtering will predict his ratings to other movies based on similarities between this user and other users. Based on the users interactions collaborative filtering could be categorized in to two fields: explicit (rating) or implicit feedback (browsing history).

The technique that have been used in Collaborative filtering can be classified into three categories:

A. Memory-based which uses user's rating data to compute the similarity between users or items.

B. Model-based which uses data mining techniques such as dimension reduction to predict users' rating of unrated items.

C. Hybrid based which combine the other two methods. In Table 1 an overview of these techniques presented.

Content based methods [20, 23] recommends items similar to those the user has liked in the past. These methods use user profile or item description. The idea behind this method is to use content of each item for recommending purposes. Content of an object is a powerful tool which can give us a lot of options. For instance, for a movie we could consider the genre, the director, the actors and the actress or other features.

Content based methods could handle cold start problem better than collaborative filtering. By combining these two methods hybrid methods [35] created which use both content and collaborative information seek to get the best of both worlds.

since Vectors would have a large length which led to sparsity and it could increase processing time, in most cases similarity process will not bring into account all available

TABLE 1. AN OVERVIEW ON COLLABORATIVE FILTERING TECHNIQUE

Collaborative Filtering Techniques	Advantages	Disadvantages
Memory-Based	1: Easy Implementation 2: New data can be added simply	1:Depend to human rating 2: Cold start 3: Sparsity
Model-Based	1: Prediction Improvement 2: Handle sparsity better	1: Information Loss in dimension reduction
Hybrid	1: Prediction Improvement	1:Implementation complexity

features. On the other hand, using dimension reduction techniques such as SVD, some information would be lost. The consequence of information loss is a reduction in accuracy.

In this paper, in order to reduce processing time and space in recommendation era, we use bloom filter (BF) [3, 5]. The concept of the bloom filter is introduced by Burton H.Bloom in 1970 [3]. Over the time, BF has become more interesting for researchers in computer science and today it has applications in fields like distributed system [23], database field, aggregate queries or ad-hoc iceberg queries [12, 25].

The first Idea for checking the membership of an item in a set is comparing the item with all members in that set. Bloom filter is a technique of checking the membership of an object in a set without any need to compare with all items. The fundamental features of any bloom filter are:

1) An array of n bits, needed for showing members and it is initially 0.

2) A collection of hash functions.

3) A set "S" of m key values.

The function of the bloom filter is such a way that for any elements in set "S" hashes it by k hash functions and the output of these hash functions are indexes of the array which have to set to1, it means that the element is a member of S [22].

There are different types of bloom filter like Traditional (Standard) Bloom Filter [3], Counting Bloom Filters [4], Multi-level Bloom Filters [16], Depth Bloom filters (DBF) which is one of the subsequences of multi-level bloom filters [36]. There are different applications for bloom filter like network applications in traffic measurement [19], data storage [7], [29] or in social networks fields like social tagging systems [6].

In order to decrease the time complexity, we have used BloofI, which is a kind of hierarchical bloom filter based on B+ tree. In BloofI all leaves are bloom filters of our objects and also bloom filters just appear in leaves.

In each level, the parent is bitwise OR of its children, that means each parent shows a set consists of just its children, also, each parent represents a set which is the union of all its children. When searching for a bloom filter among all bloom filters, time order is O(N) but by using bloofI it takes O(d logN) which d is a constant and shows the tree's order [10]. The order of the tree shows the maximum capacity of each node. Because in B+ tree each node can have d to 2d children.

In this paper, we introduce an enhanced version of the bloom filter and suggest to use bloom filters in recommendation systems. Actually, we will use the bloom filter in order to show the features of items. In this way, it causes a reduction in time and space. The structure of this paper is as follow: in section II some basic definitions. In section III related works in recommender system and techniques used in them and also bloom filters- their type and application- are mentioned we have explained a short review of bloom filter and bloofI as background knowledge. Section IV explains the proposed models are explained, the BloofI and PBloofI. And finally, section VI contains experiments results shown by charts.

2. BASIC DEFINITION

This section surveys related works in both fields, recommender systems and bloom filters, their applications and technologies. But in former some basic knowledge of bloom filters are explained. Finally, the application of bloom filter in recommender systems is illustrated in details.

Definition 2.1. Bloom Filter: Bloom filter B consists of

1)$S = \{s_1, s_2, \ldots, s_n\}$ a set of n items

2)$h_i(x)$ is a hash function i = 1…k

3) $\acute{S}$: An array of m bits, initially all 0's ;m << n

Bloom filters let us do fast query the membership of an element. For illustration, consider checking the membership of an item in a set. The basic solution is comparing that item with all items in the set. Since sets are mostly huge so it is needed to cache them to the disk. This process causes some costs not only in the case of time but also in the case of space. In this point of view, bloom filters are far more advantageous since there is no need for comparing with all items. actually, each element in the set will be hashed by k hash functions, the output of each hash function is an index in the array will set to 1 [5].

Also for query the membership of an item, the Bloom filter hashes that item with the same hash functions, and then If even an index in the array had been set to 0 it confirms that for sure the element is not in the set otherwise if all indexes have been set to 1 it will contend that the element is in the set, but with an error probability that is named "false positive error" [3, 22].

Example 2.1. consider the set S ={x, y, z} the goal is answering the Query(w). Figure 1 shows the vector contents.

Definition 2.2. Collison:

$\exists i, j : h_i(x) \neq h_j(y)$ [3]

So if we have k hash functions, by adding an item to the set at most k bit of our array would set to 1. There are different classes of hash function [9] but In our work, it has no matter for using anyone. our concentration is on bloom filter.

Definition 2.3. False positive error: The false positive is a situation when the output indexes regarded an item is a combination of output indexes of some other items [26]

Example 2.2. False positive error: When adding element x to a set, it will set some bits to 1 and then by adding element y to the set some other bits to will be set to 1. Now we want to query the membership of the item z in the set. when we hash z, part of output indexes will set to 1, by insertion of x and some others are set to 1 by insertion of y so bloom filter acclaims that z is in the set while an error has happened, this is what we say false positive error. The example of the false positive error is illustrated in Figure 2.

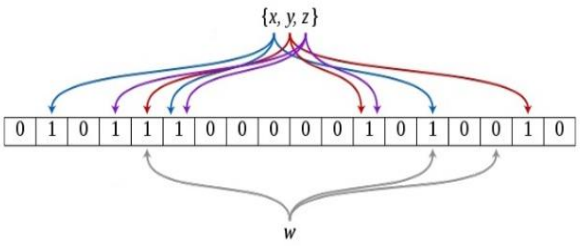

Fig. 1. An example of bloom filter for representing a set S = {x,y,z} which each item is hashed by three hash functions. K=3, m=18, N=3 and item w is not involved in the set. It is obvious that one of w's bit is 0 so can be inferred that w is not in the set.

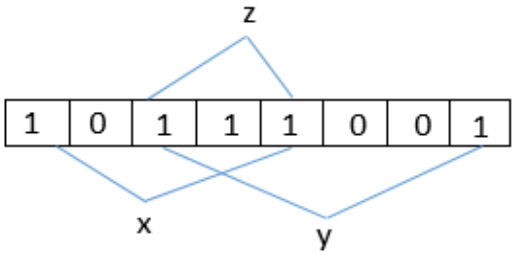

Fig. 2. Items x and y are inserted but z is not inserted. When checking the membership of z the answer is "Yes". The false positive error has occurred.

3. RELATED WORK

Recently, RS implementation in the Internet has increased, which has facilitated its use in diverse areas such as book, article, music, eLearning and etc. The most common research papers are focused on movie recommendation studies.

One of the first paper on this topic was NewsWeeder [20]. In this method each user read an article and then rate it (the ratings are 1 to 5). NewsWeeder learn the user's behavior based on their rating and suggest articles which consider user will find it more interesting.

Through best of our knowledge [17] was the first paper that used top-n recommendation task as an evaluation metrics. In their paper they improved memory-based and model-based methods to achieve better result and then they merged those two methods. Based on their reported result merging those methods could improve the predictions.

Using restricted Boltzmann machine in recommender system was first suggested by Salakhutdinov [32]. In that time Most of the existed approaches to collaborative filtering could not handle very large data sets. They claimed that their methods outperforms SVD models and it can work on a data set which has more than 100 million user/movie ratings.

Collaborative filtering methods find similar users or items using user-item rating matrix so that the system can show recommendations for users. However, most approaches related to this approach are based on similarity algorithms, such as cosine, Pearson correlation coefficient, and mean squared difference. In [24] authors presents a new user similarity model to improve the recommendation performance on cold start.

User-based and item-based collaborative filtering methods are two of the most widely used techniques in recommender systems. While these algorithms are widely used in industry they require a considerable amount of time to find top-k similar neighbors (items or users) to predict user preferences of unrated items. In [28] a Reversed collaborative filtering presented, a rapid CF algorithm which utilizes a k-nearest neighbor (k-NN) graph. This method outperforms traditional user-based/item-based CF algorithms in terms of both preprocessing time and query processing.

For a certain user, user-based k-nearest-neighbor (userkNN) collaborative filtering methods first identify a set of similar users, and then recommend top-N items based on what items those similar users have purchased. Similarly, itembased k-nearest-neighbor (itemkNN) collaborative filtering methods first identify a set of similar items for each of the items that the user has purchased, and then recommend top-N items based on those similar items [27].

The main idea of latent factor models is to factorize the user-item matrix into (low-rank) user factors and item factors that represent user tastes and item characteristics in a common latent space. Pure Singular-Value-Decomposition-based (PureSVD) matrix factorization method was introduced by Cremonesi [14].

A weighted regularized matrix factorization (WRMF) method applies a weighting matrix to differentiate the contributions from observed purchase/rating activities and unobserved ones.

"Tapestry" in [13] is one of the first recommender systems which has been introduced and the word collaborative filtering has been used for the first time. New versions for hierarchical bloom filter have been introduced in [10] and some application for hierarchical bloom filter in networks are mentioned in [34] in case of payload attributes in networks.

Also [36] have used hierarchical bloom filter in VANET to combat the challenge of the large population, rich content and low latency in VANET. A way to represent item/users in recommender system are matrices, [18] has used matrix factorization in recommender systems. In [37] the Bloom filter has been used for ensuring the security of data integration in wireless sensor networks when it is deployed in an enemy environment.

[11] has used a protocol called "summary cache" for cache sharing. The structure of this summaries is bloom filters. Moreover, Google Bigtable is an example of distributed systems and [7] have used bloom filters to improve searching in SSTable. Since the counter in counting bloom filter are fixed to eliminate this defect. Furthermore, in [8] the author has introduced an extension of bloom filter named "spectral bloom filter "(SBF) which tries to predict the multiplicity of items in a spectrum. SBF uses a static counter to solve this drawback. The author in [1] has introduced the Dynamic Count Filter which is made if two fundamental vectors: a fixed size vector and a dynamic vector named Overflow Vector that counts the number of times that the elements in the first vector have overflowed.

A. BloofI

A hierarchical form of bloom filter named bloofI is introduced in [10]. The basic construction of bloofI is a B+ tree. The leaves of the tree are the bloom filters and each parent is bitwise OR of its children. So each parent is the union of all its children and root shows a set which is consist of all elements.

[10] has used bloofI to find all bloom filters which match with a special bloom filter. when adding an item, bloofI considers the distance between new item and all other items. By this strategy, similar items will be siblings.

In B+ tree the number of each node's children must be between d and 2d, d is the order of the tree, except root that can have at least 2 children. BloofI will be constructed by sequential insertions. Figure 3 illustrates the structure of bloofI.

Recently some researchers have used bloom filter in recommender systems. [30] proposes to use bloom filter in recommender systems. In bloofI there are some bloom filter vectors constructing leaves, likewise, in recommender systems, there are some items and users, and each item has some features.

Moreover, any user has some preferences that we count as user's features.

Example 3.1. Constructing a feature vector: To construct feature vectors, we have to assign each bit of a vector to any feature. For instance, suppose our dataset are movies. As shown in Table 2, each movie has features like writer, director, actors, and genre. So we have to allocate a bit to each possible feature.

Presume that features of a movie like X, are as follow: In item X's vector, related bits to d_1, w_1, c_1, c_{10}, c_{20}, *comedy*, *drama* will set to 1 and other bits will be 0. For any feature that the related bit is set 1 is called an active feature and it means that the item has that feature.

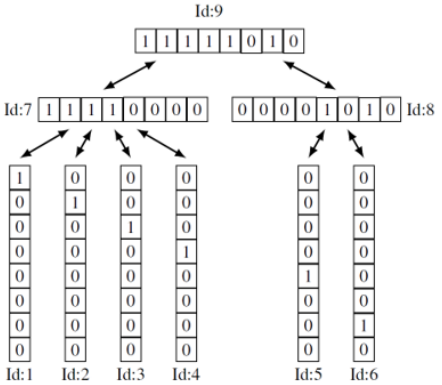

Fig. 3. A bloofI of order d=2. Internal nodes are bitwise OR of their children.

TABLE 2. NOMENCLATURES

d_i	Director i
w_i	Writer i
c_i	Actor i

Now it is time to convert vectors to bloom filters. In this case, we treat each movie as a set and the features are members of the set. The universal set is a set containing features of all movies such as all writers, all directors. Since vectors of features are sparse, this technique reduces space complexity because of its compression in features representation. To make bloom filters, we hash each active feature by k hash functions.

To find similar items to a special item, the item should be compared with all other items. For measuring distances, Jaccard distance, Cosine distance or Hamming distance can be used.

AND operation between two bloom filters shows the intersection of them[30]. In our case, it shows that which features are in common between two movies. Consider X is a movie and we want to find similar movies, outcome vector of AND operation between X and other movies should be computed. After that, among resulted AND vectors, anyone which is more similar will be suggested.

It is worth noticing that, two items are similar not only in common features - common inserted features - but also they are similar in features which both don't have those features - common missing features. To compute this similarity, we can apply XNOR operation on bloom filters. Actually, this technique not only counts common inserted feature but also counts common missing features [30].

Both techniques, using AND or XNOR will reduce the operation time. But their bottleneck is the comparison of an item with all other items. We claim that the time can be reduced more effectively. To do this we used a hierarchical bloom filter (BloofI) instead of the standard bloom filter. Since now presume that we want to suggest a list of k similar items.

4. PROPOSED APPROACHES

In this section, we want to explain our two attempts to aim at improving the performance of recommender systems. Our first attempt is using BloofI in the recommender system andthe second attempt is probabilistic Bloom filter.

A. BloofI in recommender systems

As our first attempt, we used BloofI in recommender systems. BloofI will be built by sequential insertions in a B+ tree with an order of d. To make a BloofI tree, firstly two bloom filters will be inserted and we define their parent as the union of them, Then, other bloom filters will be added. Consider that after each insertion parent has to be updated.

When a node's degree reaches to 2d the node will be divided into two nodes. Consequently, half of the children will resist as their own parent's children and the other half have to be added as a new node's children. Now both parents have to be updated and this updating will continue up to reach the root. In this way, at each time we can admit that till now the root is the union of all inserted bloom filters. Figure. 3 is an example of BloofI in recommender systems.

A novel accomplishment of this technique is using this insertion method, we not only can suggest new items similar to an item, but also classify all other leaves such that more similar bloom filters will be as siblings.

In this technique, to construct a parent, we apply OR on its children so when a bit of parent is one it means that at least one of the children has that feature so it doesn't matter how many of children have that feature. Since it declines the effect of the number of 1's in a special bit of children, somewhat it effects on our computations.

Example 4.1. To illustrate more, suppose that, we want to add a new bloom filter Id14: [0110000001] to Figure. 4. The Hamming similarity between Id14 and Id3 is equal to 2. On the other hand, the Hamming similarity between Id14 and Id2 is equal to 3. So the bloofI algorithm recognizes that Id14 is more similar to Id2 while according to Figure. 4, Id14 is more similar to Id2. So we tried another attempt which we are going to explain below.

- **BloofI Algorithm**

In proposed the algorithm we have four basic methods, *findIndex*, *insert*, *lookup* and *update* in the following paragraphs we describe these four methods.

The *Algorithm 1* finds the best place for an item. The inputs are a pointer to the tree's root and item X. The best palace for insertion is the index of a parent which its children are more similar to the X. In this method, we compute the Hamming distance between X and all root's children. Among all resulted vectors, anyone that has greater cardinality, we choose that node to flow a path to leaves. Repeatedly for newly chosen node we compare the resulted vectors of Hamming distance and choose another new node dawn in the path.

To make a tree we use consequential insertion. In the *Algorithm 2*, again the inputs are the pointer to the root and item X. The first two bloom filters will be inserted and their parent is their bitwise OR. Since now, to add any other node we first run findIndex, to discover the best place for inserting the node. when we insert a new item to the tree the parent's vector has to be updated by the *Algorithm 3* and also other parents till we reach the root. For updating the parent, we use this method too.

In the update method, the input is the returned index by the *findIndex* method and its output is an updated tree.

Finally, the most important method is *Algorithm 4*. This is the main method which its inputs are item X, the item which we want to find similar objects to it, an integer K and a pointer to the tree's root. The lookup method tries to make a suggestion list of k most similar items to X. According to the value of k and the number of leaves in sub-tree, two different cases may happen (index.leaves is the number of leaves).

- K ≤ index.leaves: in this case, we follow the path down to the parent of leaves, now just among children of this parent we choose k most similar items for suggesting.
- K > index.leaves: in this case, as the difference between k and d increases we have to stop the process of following the path, in an upper level and choose members of our suggestion list among siblings and cousins.

In Figure. 5, the different relation between d and K are exemplified.

B. PBloofI: a probabilistic version of bloofI

In BloofI, each node is bitwise OR of its children and when a bit sets to 1 it doesn't matter how many of the children have that bit as an active feature. Anyway, by binary vectors, we cannot find out this issue. Hence in order to bring into account the number of children with a special active feature, we propose to use probabilities.

Example 4.2. As shown in Figure 6, the leaves in PBloofI are binary bloom filters but internal nodes (parents) are different. The content of any bit is a fraction of children which have one in that bit. So the maximum value of parent's bits can be 1.

Consider an internal node (a parent) X of size m < x1, x2, … , xm >, and suppose that node has i children. To set a value to x1, we have to count the number of children that the first bit of them is 1. suppose in k number of children the

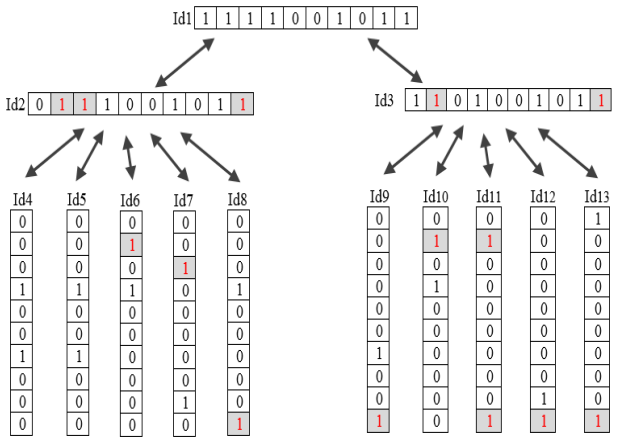

Fig. 4. When using bloofI, Id14:[0110000001] has to be inserted as a child of Id2 while it is recognizable that Id14 is more similar to children of Id3. Filled bits shows common bits between Id14 and any vector.

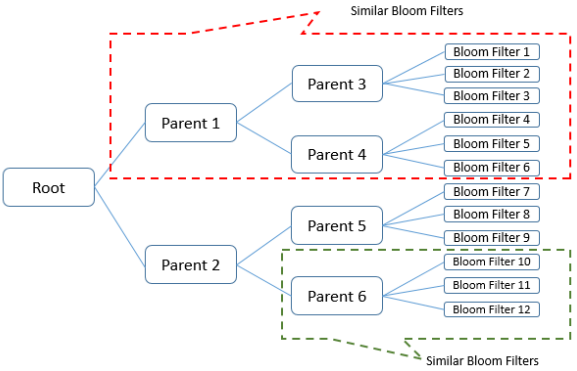

Fig. 5. A bloofI with d=2. Leaves are divided into two groups by dash lines.The upper group is 3 < K ≤6 for and the lower group is for K ≤3.

first bit is 1 so the value of the first bit of X will be $\frac{k}{l}$. By this technique, we can bring into notice that how many children have a special feature. If all children have 1 in the same bit, the value of that bit in their parent would be 1. On the other side, if none of them is 1 in a special bit, the value of that parent's bit will be 0.

To suggest a list of similar items to an item X, at first, we check root's children. because bits are not binary so we cannot use AND operation to calculate distances so we propose to use cosine distance. We compare all distances and choose the node with the most appropriate value. Now on, we continue a path to leaves by repeating these operations for the new node. Figure. 6 is a simple but illustrative example of PBloofI.

In this model, there is no change in insert and lookup methods. Since in this model we use Cosine distance, we will have a slight change in findIndex method but the inputs and outputs are the same as previous algorithms.

Algorithm 1 FindIndex Procedure

```
1: procedure FINDINDEX
2:     Input: node: A Boolean array of length m
3:     Input: root: The tree's root. Before making tree can be null
4:     Output: index of an internal node. Return value is the index of a parent where "X" is more analogous to its children
5:     create a root node R
6:     while R.child is not leaf do
7:         z ← 0
8:         create a new node T
9:         T ← NULL
10:        while R.child ≠ NULL do
11:            T ← R.child
12:            y = hammingDistance (X, T)
13:            if y > z then
14:                z ← y
15:                index ← T
16:                return index
17:            else
18:                z = y
19:            z = y
```

Algorithm 2 Insert Procedure

```
1: procedure INDEX
2:     Input: node X
3:     Output: Pointer root
4:     if root = Null then
5:         root ← x
6:     index ← findIndex(nodeX)
7:     if index.childNumber > 2d then
8:         create a new node newParent
9:         newParent ← null
10:        transport half of the children to the new parent
11:    index.add(X)
12:    update(index)
13:    z ← y
14:    return root
```

C. *Hash function*

The hash functions that are used in implementing a recommender system with bloom filters, must have special characteristics. Conventional hash functions that are used in cryptography try to minimize the probability of collision for different items, but in recommender systems, the definition of different items is something else. Two items may have

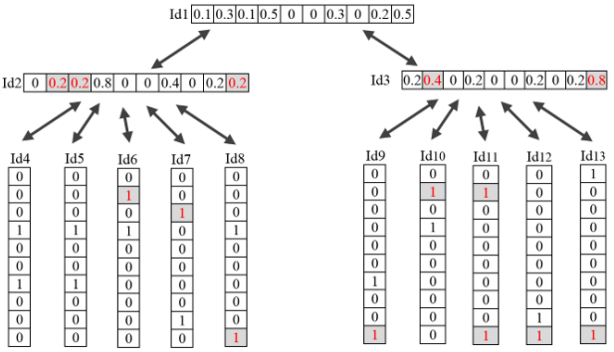

Fig. 6. Inserting bloom filter Id14:[0110000001] to the tree. Filled bits shows the in internal node shows bits that will be affected by this insertion and in leaves shows common inserted features

Algorithm 3 Update Procedure

```
1: procedure INDEX
2:     Input: index: returned index by findIndex algorithm
3:     while index ≠NULL do
4:         while index.child ≠ NULL do
5:             index = OR(index:child)
6:         index = index.parent
```

Algorithm 4 Lookup Procedure

```
1: procedure LOOKUP
2:     Input: node: is a Boolean array of length m
3:     Input: root: is the tree's root. Before making tree can be null
4:     Input: K: is the number of items in suggested list
5:     Output: suggested list: Return value is a list of k most similar items to X
6:     Index ← findindex(X)
7:     If index.Leaves ≥ k then
8:     while index.leaves < k do
9:         index = index.parent
10:    Return Suggested list
```

some different active feature, but since their similar active features are more, we call them similar items. On the other hand, in common case, these are different items in the hash functions' point of view. So, we need to map similar items in recommender systems to the same buckets. So, the family of Locality-sensitive hashing (LSH) is more suitable in these systems.

Definition 4.1: **LSH family**

LSH family f is a set of hash functions h: U → M where each hash function maps elements from universe U to a bucket $m \in M$, and satisfy the following two conditions, where R is a positive threshold and $c > 1$ is an approximation factor:

Condition I: If $d(p,q) \leq R$, then $h(p) = h(q)$ with the probability at least P_1. In this relation, p and q are two

different items, and d is the distance function (e.g. Jaccard or Cosine distance). In other word, the hash function should map the similar objects (points) into the same bucket with the probability at least P_1.

Condition II: If $d(p,q) \geq cR$, then $h(p) = h(q)$ with the probability at most P_2. In this relation, p and q are two different items, and d is the distance function. In other word, it is expected that the hash function does not map the different items into the same bucket.

Notice that we interested in hash functions where $P_1 > P_2$.

Proposition 4.1: Assume that two items i_1, i_2 are mapped into two Bloom filters b_1, b_2. Also assume that k different LSH hash function is used in this mapping. Equation (1) shows the probability of matching exactly i hash function, shown by $m(i)$ is:

$$m(i) = \binom{k}{i} . {P_1}^i . (1 - P_1)^{k-i} \qquad (1)$$

Proposition 4.2: Assume that two items i_1, i_2 are mapped into two Bloom filters b_1, b_2. Also assume that k different LSH hash function is used in this mapping. If we use Jaccard similarity measure and $d(x,y)$ indicates Jaccard distance between x and y, (2) shows the expected value of similarity between b_1, b_2:

$$similarity(b_1, b_2) = \sum_{i=1}^{k} m(i) . \frac{i}{k+(k-i)} \qquad (2)$$

Figure 7 shows the accuracy (according to similarity equation derived in proposition 4.2., for different values of k, and also the number of items that mapped into the same bloom filter. As we expect, the accuracy will decrease when we increase the number of hash functions, but the number of items that mapped to a bloom filter will decrease. In this example, we supposed that the size of bloom filter is m bit, and the total number of items is 1000. In this case, the value of $k = 3$ is an ideal choice. For larger values of k, the accuracy will decrease, but the average number of items in each bloom filter does not decrease.

5. EXPERIMENT

In this paper, we have used dataset " hetrec2011-movielens 2k" available on "https://grouplens.org/datasets/hetrec-2011/". In Table 3 details about Data set information can be reviewed.

To construct the feature vectors, we defined a Boolean array. Then we allocated the first 20 bits to genres such that each bit belongs to a genre, and then allocate the next bits to screenwriters and kept doing this allocation for other features. In this way, the length of the feature vector will be 112537.

Since we use bloom filters there is no need for feature selection. Consequently, the proposed methods are supposed to be trustworthy. In these test, we took the list produced by vector model as the standard list. In the vector model, the vector of features will not convert to bloom filter and by computing the Hamming distances the list will be created.

To check the loyalty of methods, we had compared the suggested list for l= 100 with the output list of the vector model. It is supposed that items in both lists have to be in common. To compute the accuracy of our models, we have used the (3). Items which exist in both the standard list and our list, we define as True Positive(TP). Also, (3) is the number of items in the suggestion list.

$$\text{accuracy} = \frac{TP}{l} \qquad (3)$$

Now consider fp as the probability of false positive error, k as the number of hash functions, m as the length of the array and finally n as the number of elements in the set. Suppose all n elements have been inserted. Equation (4) indicates the probability for a zero bit after all insertions.

$$p(bit = 0) = (1 - \frac{1}{m})^{kn} \approx e^{\frac{-kn}{m}} \qquad (4)$$

Obviously, (5), which is complementary of (4), shows the probability of a bit 1 after all insertions.

TABLE 3. DATASET OVERVIEW

Dataset	#users	#movies	#screenwriters	#tags	#genres
ML2K	2113	10195	4060	13222	20

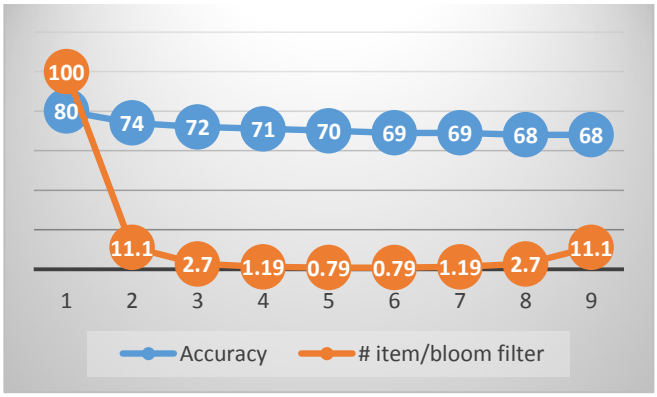

Fig. 7. The effect of k on the accuracy and number of items that mapped into the same bloom filter.

$$q = p(bit = 1) = 1 - p(bit = 0) \quad (5)$$

Finally, for computing false positive error fp , consider (6)

$$f_p = q = p(bit = 1)^k = (1 - (1 - \frac{1}{m})^{kn})^k \approx e^{\frac{-kn}{m}} \quad (6)$$[28]

As bloom filters are showing sets, we can use the set's operation on bloom filters. assume we have two bloom filters BFa and BFb which are the representations of sets Sa and Sb.

To show the union of these two sets we have to apply bitwise OR on their bloom filters. Bitwise OR on bloom filters can be applied under two conditions:

1) both bloom filters should be in the same size

2) the number of hashed functions applied on both of them should be equal and also same hash functions have to be used [17].

Suppose n is the number of all features. hence, it is also the length of the feature vectors. And suppose n_{max} is the number of active features of a vector with maximum active features. Also, suppose X is an Item with n_{max}. we claim that X is our worst case since it causes the maximum false positive error. Again consider the (6), the replication of n with n_{max} results in (8), Obviously, it proves our claim [26].

$$fp = q(bit = 1)^k = (1 - (1 - \frac{1}{m})^{kn_{max}})^k \quad (8)$$

To best describe the performance of each method, we chose 20 random items and prepared a suggestion list for each of them. In the end, we used a box plot to compare the performance of all suggested algorithms.

Figure 8. shows the accuracy comparison between all methods, the bloofI, PBloofI and finally the AND method which does bitwise AND on a random item and all other items, and suggests items with grater AND vector cardinality(AND method is introduced in [30]). This figure shows that all three algorithms have somehow the same accuracy level.

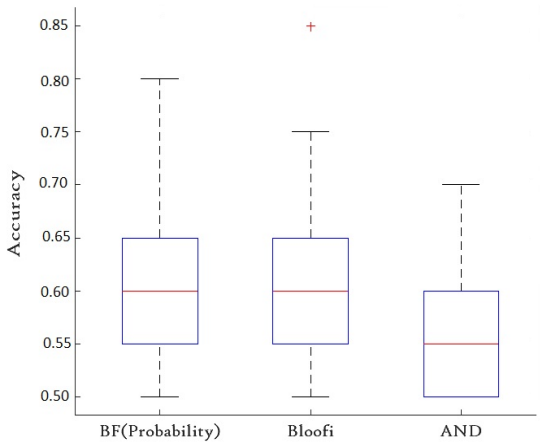

Fig. 8. The comparison of three algorithms in case of suggesting a list of 100 items.

Also, the comparison between the time complexity of methods is shown in Figure 9. This figure shows a remarkable difference in the order of time complexity between newly proposed algorithms and the former one. So, we can say that the algorithms will not decrease the accuracy while they decrease the time complexity.

Another way to measure the quality of a recommendation system is Hit Rate (HR) which is defined as follows (9):

$$\text{Hit Rate} = \frac{\#hits}{\#users} \quad (9)$$

where #users is the total number of users, and #hits is the number of users whose item in the testing set is recommended (i.e., hit) in the size-N recommendation list. In the Table 4 the comparison of some of these algorithms on ML10M could be seen. Note that in this table the N is equal to 10.

Columns corresponding to parametrs a and b present the parameters for the corresponding method. For methods itemkNN and userkNN, the paramesters are number of neighbors. For method PureSVD, the parameters are the number of singular values and the number of iterations during SVD. For method WRMF, the parameters are the dimension of the latent space and the weight on purchases [27].

6. CONCLUSION

Recommender systems represent items by a vector of features. Mostly, features are in an extended range, while

TABLE 4. COMPARISON OF TOP-N RECOMMENDATION ALGORITHMS

method	Parameter a	Parameter b	HR
itemkNN	20	-	0.238
userkNN	50	-	0.303
pureSVD	170	10	0.274
WRMF	100	2	0.306

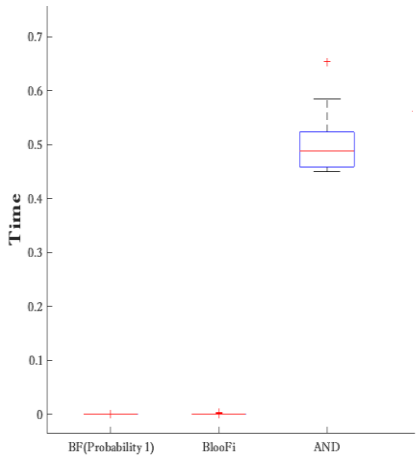

Fig. 9. The comparison of time complexity between three different algorithms. (the unit of time is second)

each item has a small number of those features. As a result, showing items by vectors, the result vectors are sparse. This sparsity causes an increase in time and space cost. There are some methods that by selecting more effective features try to decrease the sparseness. Since some features are ignored, the accuracy will decrease.

In this paper, we have focused on two important issue:

1) reducing time complexity.

2) increasing the accuracy.

Bloom filters can be used for representing items in lower space. Since bloom filter uses hashing functions, the accuracy will have just a slight change. So bloom filters can be a good choice for item representation in order of reduction in time and space costs. Only representing items by bloom filters is not efficient and we have to choose a methodology for comparing bloom filters in order of finding similar items. In an earlier method, a special bloom filter had to be compared with all other bloom filters. which is not time-consuming.

Making some changes in bloofI can cause a significant improvement in accuracy. A way of these changes is using the probability of happening 1 in each index of children. Since using bloom filter in recommender systems is a new idea, it seems that be a good idea for future researches. We have just examined bloofI, but there are other types of bloom filters. so experimenting other types of bloom filters is future work. Also, we have used static data, while in the network there are dynamic data too. Other future experiments can be defined as experimenting with different distance methods, analyzing the effect of different hash functions on results.

REFERENCES

[1] Aguilar-Saborit, J., Trancoso, P., Muntes-Mulero, V., & Larriba-Pey, J. (2006). Dynamic count filters. *Acm Sigmod Record*, 35(1), 26-32.

[2] Billsus, D., & Pazzani, M. J. (1998). Learning collaborative information filters. In *Icml*, 98, 46–54.

[3] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 422–426.

[4] Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., & Varghese, G. (2006). An improved construction for counting bloom filters. In *European Symposium on Algorithms*, pp. 684–695. Springer.

[5] Broder, A., Mitzenmacher, M., & Broder, A. (2002). Network applications of bloom filters: A survey. *Internet Mathematics*. 1(4), 485-509.

[6] Cantador, I., Bellogın, A., & Vallet, D. (2010). Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems* (pp. 237–240). ACM.

[7] Chang, F., Dean, J,. Ghemawat, S,. Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T,. Fikes, A., & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2), 4.

[8] Cohen, S., & Matias, Y. (2003). Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data* (pp. 241–252). ACM.

[9] Coron, J., Dodis, Y., Malinaud, C., & Puniya, P. (2005). Merkle-damgard revisited: How to construct a hash function. In *Annual International Cryptology Conference* (pp. 430–448). Springer.

[10] Crainiceanu, A., & Lemire, D. (2015). Bloofi: Multidimensional bloom filters. *Information Systems*, 54, 311–324.

[11] Fan, L., Cao, P., Almeida, J., & Broder, A. Z. (2000). Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, 8(3), 281–293.

[12] Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., & Ullman, J. D. (1999). Computing iceberg queries efficiently. In *Internaational Conference on Very Large Databases (VLDB'98), New York, August 1998*. Stanford InfoLab.

[13] Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12), 61–70.

[14] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining* (pp. 263-272). IEEE.

[15] Jain, N., Dahlin, M., & Tewar, R. (2017). Using bloom filters to refine web search results.

[16] Koloniari, G., Petrakis, Y., & Pitoura, E. (2003). Contentbased overlay networks for xml peers based on multi-level bloom filters. In *International Workshop on Databases, Information Systems, and Peerto-Peer Computing* (pp. 232–247). Springer.

[17] Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 426–434). ACM.

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

[19] Kumar, A., Xu, J., & Wang, J. (2006). Space-code bloom filter for efficient per-flow traffic measurement. *IEEE Journal on Selected Areas in Communications*, 24(12), 2327–2339.

[20] Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995* (pp. 331–339). Elsevier.

[21] Ledlie, J., Serban, L., & Toncheva, D. (2002). Scaling filename queries in a large-scale distributed file system.

[22] Leskovec, J., Rajaraman, A., & David Ullman, J. (2014). *Mining of Massive Datasets*. Cambridge University Press.

[23] Li, X., Cheung, M., She, J. (2016). Connection discovery using shared images by gaussian relational topic model. In *2016 IEEE International Conference on Big Data (Big Data)*, pp. 931–936. IEEE.

[24] Liu, H., Hu, Z., Mian, A., Tian H., & Zhu, X. (2014). A new user similarity model to improve the accuracy of collaborative filtering. *Knowledge-Based Systems*, 56, 156–166.

[25] Singh Manku, G., & Motwani, R. (2002). Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pp. 346–357. Elsevier.

[26] Mitzenmacher, M., & Upfal, E. (2005). *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press.

[27] Ning, X. & Karypis, G. (2011). Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pp. 497–506. IEEE.

[28] Park, Y. Park, S., Jung, W., & Lee, S. (2015). Reversed cf: A fast collaborative filtering algorithm using a k-nearest neighbor graph. *Expert Systems with Applications*, 42(8), 4022–4028.

[29] Podder, S., & Mukherjee, S. (2018). A bloom filter-based data deduplication for big data. In *Advances in Data and Information Sciences*, pp. 161–168. Springer.

[30] Pozo, M., Chiky, R., Meziane, F., & Metais, E. (2016). An item/user representation for recommender systems based on bloom filters. In *IEEE Tenth International Conference on Research Challenges in Information Science (RCIS 2016)*. IEEE.

[31] Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pp. 452–461. AUAI Press.

[32] Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pp. 791–798. ACM.

[33] Salman, A. (2011). Bloom's filters: their types and analysis. *Dogus Universitesi Dergisi* , 6(2), 268–278.

[34] Shanmugasundaram, K., Bronnimann, H., & Memon, N. (2004). Payload attribution via hierarchical bloom filters. In *Proceedings of*

the 11th ACM conference on Computer and communications security, pp. 31–41. ACM.

[35] Wang, C., & Blei, D. M. (2011). Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 448–456. ACM.

[36] Yu, Y., Gerla, M., & Sanadidi, M. (2015). Scalable vanet content routing using hierarchical bloom filters. *Wireless Communications and Mobile Computing*, 15(6), 1001–1014.

Zahra Farahi received her master degree in algorithms and computation from the University of Tehran in 2018. Her research field was about recommender systems. Currently, she is a PhD student at Tehran University in algorithms and computation. Her main research interest includes: network science, data science, data mining

Ali Moeini received his Ph.D. in Nonlinear Systems at the University of Sussex, UK, in 1997. He is a full Professor of Department of Algorithms and Computation, School of Engineering Science, College of Engineering, University of Tehran, His research interests include Network Science, Mining of massive Datasets, Randomized Algorithms, Online Algorithms and Competitive Computations, and Bioinformatics Algorithms

Ali Kamandi received his Ph.D. in Software Engineering from Sharif University of Technology in 2010. Since 2015 he has been a faculty member in the Algorithms and computation group at School of Engineering Science at the University of Tehran. He has several publications in the areas such as software engineering, e-Commerce, e-Marketing, and data sciences. His main research interests are data sciences, data mining, and distributed systems.

Mahmood Sabankhah holds a BSc in Electrical Engineeringfrom Amirkabir University if Technology(2002) an MSc (2004) and a PhD (2008) in ;Pure Mathematics both both from Universite Lava, canada. Although his main research intrest lie in field of Complex Function Theory and Operators acting on such space, he is aldo intrested in the field of Optimization and Machine Learning.

Seyed Mohsen Hosseini was born in Iran in 1993. He completed his B.S.'s degree in computer engineering with focus on Hardware design at IRAN UNIVERSITY OF SCIENCE AND TECHNOLOGY in 2017 and he is currently a master's student in field of Algorithms and computation in university of Tehran. His main areas of research interests are recommender systems, machine learning, data mining, and frequent patterns recognition.