An Item/User Representation for Recommender Systems based on Bloom Filters

Manuel Pozo, Raja Chiky, Farid Meziane, Elisabeth Métais

An Item/User Representation for Recommender Systems based on Bloom Filters

Manuel Pozo and Raja Chiky
LISITE-ISEP
28, rue Notre Dame des Champs
75006 Paris (France)
email: {name.surname}@isep.fr

Farid Meziane
University of Salford
The Crescent, Salford, Lancashire
M5 4WT, United Kingdom
email: f.meziane@salford.ac.uk

Elisabeth Métais
CNAM
292, rue Saint-Martin
75003 Paris (France)
email: elisabeth.metais@cnam.fr

***Abstract*—This paper focuses on the items/users representation in the domain of recommender systems. These systems compute similarities between items (and/or users) to recommend new items to users based on their previous preferences. It is often useful to consider the characteristics (a.k.a features or attributes) of the items and/or users. This represents items/users by vectors that can be very large, sparse and space-consuming. In this paper, we propose a new accurate method for representing items/users with low size data structures that relies on two concepts: (1) item/user representation is based on bloom filter vectors, and (2) the usage of these filters to compute bitwise AND similarities and bitwise XNOR similarities. This work is motivated by three ideas: (1) detailed vector representations are large and sparse, (2) comparing more features of items/users may achieve better accuracy for items similarities, and (3) similarities are not only in common existing aspects, but also in common missing aspects. We have experimented this approach on the publicly available MovieLens dataset. The results show a good performance in comparison with existing approaches such as standard vector representation and Singular Value Decomposition (SVD).**

I. Introduction

The large amount of information on the Internet makes it difficult for users to find exactly what they need. Recommender Systems (RS) analyse the interests of users and present first the information in which they might be more interested [1]. This dramatically reduces the amount of information presented to user and delivers the correct information to the correct users [2]. Hence, it attracted the interest from many fields, such as e-commerce, service providers and media services, with the aim of personalizing the users' experience.

In this paper we focus on improving items/users representation in the domain of recommender systems and within the context of very large datasets and multiple resources. As in other data based applications, such as Information Retrieval, these systems compute similarities among items/users based on the resources they have access to. However, these resources can be very large, thus making the representation and description of items/users difficult to deal with in terms of sparsity and space-consumption. Typically, data analysis systems reduce data representation into a lower space thus reducing both the sparsity and space complexity. However, reducing the data description may affect the accuracy of the similarity measure.

In this research, we use a compressed yet high quality data representation for items/users based on Bloom Filters. A Bloom Filter (BF) is a bit structure that allows to represent a set of elements in very low space [3]. To the best of our knowledge, these filters have not been used in the field of recommender systems and we believe that their properties can be very beneficial. In addition, we propose a similarity model based on these structures in order to address high quality representations, sparsity and space-consumption issues. The main advantage of this approach is that it hardly affect similarities matching. In fact, this idea may expand the representation of items/users by adding all possible resources and making very detailed representations at a very low space complexity. The motivation and the three following contributions are applied to a recommendation process:

1) Bloom filters highly reduce the size of item representations while having a great number of features. This fact involves a detailed items/users descriptions in a lower space complexity;
2) Bloom filters allow a fast bitwise AND operation to compare common features of items that we use to compute an accurate similarity measure;
3) Item and user similarities are not only in common features, but also in common missing features. To address this issue, a bitwise XNOR similarity operation is proposed and takes into account both common features, and common missing features.

In the recommender systems literature, recommendation techniques are usually classified into collaborative filtering, content-based and hybrid methods [1]. These techniques exploit the interest of users, a.k.a. feedback or preferences, in order to learn and adapt the recommendations. Moreover, they may also use the characteristics of items/users, a.k.a. features or attributes. Collaborative filtering techniques group users according to their feedback and recommend items that people from the same group of preferences have already liked in the past [4], [5]. Content-based techniques use the items features to compute items similarities and to recommend similar items to the ones that the user preferred in the past. Yet, it has to deal with big and heterogeneous data. Hybrid methods combine both techniques to alleviate their drawbacks and to enhance recommendations [6]. Thus, it increases the complexity of the system.

In content-based and hybrid systems specifically, there is a similarity computation process. For instance, in a movie application, the features of a movie (considered as an item) can be the genres, the actors, the directors, etc. This set of features may be represented as a vector of features or keywords, and thus, vector similarity methods can be computed [1]. The accuracy of the similarity will depend on the selected features for comparison and will affect the recommendation. For example, comparing two movies only by their genre, such as comedy and drama, is less accurate than involving also their actors in the similarity process.

Indeed, the number of features affect the similarity measure, and generally the more attributes are used the better the accuracy of the similarity is. However, new features may not appear in other items, for instance, a new actor that has played only in one movie, and increasing the number of features increases the size of vectors that will eventually become very large and sparse. Furthermore, increasing vector sizes makes similarity computations slower. New tendencies in the field of recommender systems favour adding heterogeneous data from external sources, such as open data, Twitter and Facebook, in order to better describe items/users [1], [7], [8]. However, this could create very large heterogeneous vector of features and render computations more difficult to handle.

Expert systems, modeling topic analysis, explanatory analysis or dimensional reduction models (such as Singular Value Decomposition or Principle Component Analysis) are often used to alleviate this space complexity. Indeed, these analysis identify the most relevant features of the items in order to reduce the vectors' size. However, reducing features may yield to a loss of quality in the item representation, and consequently, in items similarity. Besides, the enormous size of the datasets involves big memory resources and a very large time analysis while building these models.

In our experimentations we use the MovieLens dataset and IMDb database, which are publicly available [9]. The tests have been performed for: (1) analysing the pertinence of bloom filters in recommender systems, (2) enhancing the usage of a bitwise AND similarity instead of the common vector cosine/jaccard similarities and (3) using a bitwise XNOR similarity. We compare our bloom filter item representation against two models: a vector similarity model that uses all available features and a dimension reduction technique based on Singular Value Decomposition (SVD). The results show that our approach highly reduces the size of vector representations (97% per vector) while keeping a high fidelity in item similarity (accuracy of 98%). In addition, it outperforms the results of SVD and using the same space complexity the bloom model outperforms item similarity accuracy.

The structure of this paper is as follows: Section II gives the related work in similarity processes applied to recommender systems. Section III introduces the background and the context for bloom filters and its application in recommender systems. In Section IV, the proposed similarity model based on bloom filters is explained in details. In Section V, the experimentation phase and the achieved results are presented. We conclude and present the future work in Section VI.

II. Related Work

The most known recommendation techniques are collaborative filtering, content-based and hybrid methods [1]. As mentioned previously, hybrid methods combine both techniques by taking their advantages and alleviating their drawbacks [6]. In fact, collaborative filtering has already demonstrated simplicity and better accuracy than content-based, and thus, hybrid methods may enhance these techniques by incorporating heterogeneous data.

The representation of items/users as a vector is very used in content-based and hybrid method. This can be used also in collaborative filtering methods. Indeed, in memory-based collaborative filtering techniques, users are represented by a vector of preferences where each dimension corresponds to one item and each value is the degree of interest in this item [4], [5], [10], [11]. Then in order to find out similar users, correlations and similarities among these vectors are performed. On the contrary, model-based collaborative filtering is a new generation of techniques that makes vector representation obsolete [5], [12]–[15]. Particularly, the Matrix Factorization (MF) technique is widely extended due to its accuracy, scalability and simplicity. It creates a reduced and trained latent space model to explain future users' preferences. However, these techniques solely rely on the interest of users in items. In content-based techniques, items are represented by a vector of keywords/features [16]. Each dimension corresponds to one feature that the item may have or not. For example, in the domain of movies an actor only plays in a subset of movies. Hybrid methods may combine both techniques to get stronger performance. Consequently, they also exploit the feature of items/users and may use a vector to represent them.

Some authors propose hybrid approaches to enhance recommender systems. In [17] authors propose a framework to control the similarity (versus diversity) factor in a top-K recommended items. They create "trees of interests" that allow creating a "zoom-in" technique to see more items of the same tree, which tend to be similar. Some other tendencies in recommendations use semantic technologies, like ontologies, to better describe users and items [7]. For instance, in [18] and [19] the authors propose a hybrid method to describe users and items through ontologies structures in order to compute an inference similarity. In [6], the author proposes to combine case-based reasoning recommendation and a collaborative filtering technique. It studies the past items and past user records to find, adapt or create new similar items for the recommendation.

Other authors propose to incorporate more heterogeneous data from external resources. In [20], authors use item-item similarity based on the context in Wikipedia pages to compute "artificial ratings" for an item. [21] characterizes economic articles using attributes as keywords. Then, they compute arti-

cles cosine vector similarity regarding their keywords. Finally, articles that match better with a user profile are recommended.

However, to deal with large number of features is computationally expensive. In [22], authors propose to add demographical and context information to the recommendation. They create users and/or items demographic vectors in order to represent information. If the dimension of these vectors is too high, they reduce it by applying the SVD technique. A cosine similarity method is later used to compute user and item similarities. In [23] authors propose a different hybrid method based on content-based and collaborative filtering. It adds an item-similarity process to an item-based collaborative filtering technique. This similarity uses semantic sources to bring the attributes of items to the recommender context. They create an item-attribute matrix where items express the attributes they have. However, this matrix may be too noisy and attributes may be correlated. Thus, the authors reduce it by using SVD. This reduced matrix allows the creation of an item-item matrix that represents items similarities. In [24] the authors proposed a framework for collaborative filtering that enhances the recommendation by analysing the implicit interest of users in the features of the items. This approach uses a Principle Component Analysis to reduce the features to the relevant ones.

However, most similarity processes do not take all possible or available attributes information into account. Larger number of features may improve similarity, but computation might become more expensive. In addition, approaches dealing with space reduction or latent space models may lose information. We propose to use a vector representation of items/users based on bloom filters. This reduces the size of the vectors and compute accurate and faster similarities. In addition, this reduction may allow taking more features into consideration, which helps to improve the similarity accuracy of the system. Similar bloom filter usages have been implemented in other applications. Recently, in [25] these filters have been applied to create a fast, accurate and private plagiarism system that compares the similarity among documents of different databases. In [26] these filters have been used for web search field. Typically web browsers return to users' request top websites which tend to be very similar. The authors propose to group similar websites results in order to allow the return of other diverse sites in the top results. In this paper we use these filters in a real life application within the recommender systems domain. In addition, we evaluate the accuracy of the bloom filters representation by comparing it against vector similarity methods and order reduction techniques.

III. Background and Context

The concept of bloom filter was first introduced by Burton H. Bloom in 1970 [27]. It has been used in many different applications, such as databases queries or computer networks, due to its memory-efficiency and fast-capabilities [3]. To the best of our knowledge, this structure has not been used yet in the field of recommender systems. In this section we explain bloom filters and we develop the problem of applying them to represent items/users in a recommender system context.

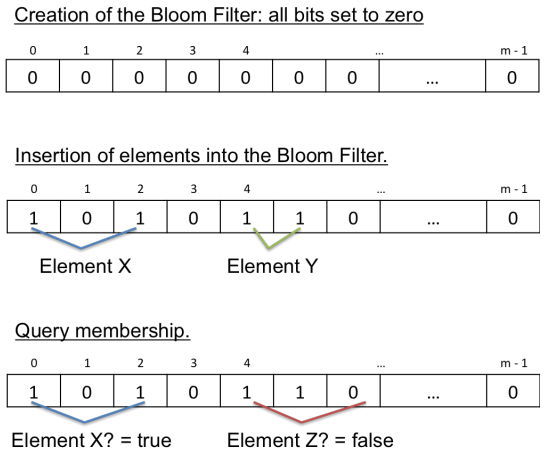

Fig. 1. Example of bloom filter. Initially the bloom filter is empty (bits set to "0"). Elements X, Y are inserted by hashing them and by setting adequate bits to "1". One membership query for element X returns true, while for a non-inserted element Z returns false.

A. Bloom Filter

A bloom filter is a bit-structure, which represents "n"-elements of the same set "S" in a lower space of "m"-bits [3]. Initially, the m-bits are set to "0" representing the absence of elements in the filter. Then, "k"-independent hash functions efficiently distribute the insertion of elements among the bit-structure. This modifies the status of "k" bits in the structure (one bit for each hash function used). In other words, to insert an element it is hashed to get the "k"-positions of the structure to set to "1".

Bloom filter structure allows fast membership queries. It works as follows: the requested element is hashed with the "k" hash functions and the "k" resulted indexes are checked in the bit-structure. If any of these indexes is set to "0", the bloom filter *assures that the element has not been inserted*. When all indexes are set to "1", one can assume that the element has been inserted with an error probability called false positive probability "fp". A false positive is the situation in which a membership query returns that the element belongs to the set, yet it actually does not. Nevertheless, the estimation of this false positive ratio is possible and near-optimal parameters may be computed [3]. A bloom filter example is shown in Figure 1.

In addition, the intersection and the union of two filters, A and B, are possible under two conditions: (1) the bits number of A is equal to the bits number of B, and (2) they both use the same hash functions. i.e m and k are equal in both bloom filters. These properties will allow us to perform filter comparisons and to develop our approach.

1) Near Optimal False Positive: As explained above, one may assure that an element has not been inserted in the filter if any of the associated indexes is set to "0". On the other hand, the probability of a false positive can be estimated.

Indeed, the false positive ratio depends on three parameters: (1) the quantity of expected insertions "n", (2) the size of the bit structure "m", and (3) the number of hash functions "k". After all the "n" elements have been inserted, some bits in the structure remain set to "0" but others are set to "1". The *probability that a specific bit is still zero* is given by Equation 1 [3].

$$p(bit = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx \left(e^{-kn/m}\right) \tag{1}$$

Moreover, the *probability that a specific bit is set to one* is given by $q(bit = 1) = 1 - p(bit = 0)$. Thus, the false positive probability is defined through this probability and the number of hash functions[1], as in Equation 2.

$$fp = q(bit = 1)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k \tag{2}$$

Further efforts to find out near-optimal values for the false positive ratio can demonstrate the existence of a global minimum [3] for "k", $k = \frac{m \cdot ln2}{n}$. In addition, "$m$" can be estimated by: $m \geq \frac{-n \cdot log_2(fp)}{ln2}$. Note that in a real application "k", "m" and "n" are integer numbers. These equations are used to build and initialize any bloom filter.

2) Intersection property: The intersection of two bloom filters, BF_A and BF_A, aims to find common elements of two different sets, S_A and S_B. One bit will be set to one if: (1) the element belongs to the intersection of BF_A and BF_B: $(BF_A \cap BF_B)$, or (2) this bit is set simultaneously to one in $BF_A - BF_A \cap BF_B$ and $BF_B - BF_A \cap BF_B$. Indeed, it represents the **AND operation** of the two filters A and B, and the result is a new bloom filter containing their common insertions. Thus, the *probability that a specific bit is set to one in both filters* is given by the simplified Equation 3 [3] and the false positive of this intersection is given by $fp = q(bit = 1)^k$.

$$q(bit = 1) = 1 - (1 - \frac{1}{m})^{k \cdot |S_A|} - (1 - \frac{1}{m})^{k \cdot |S_B|} + (1 - \frac{1}{m})^{k \cdot (|S_A| + |S_B| - |S_A \cap S_B|)} \tag{3}$$

3) Union property: The union of two bloom filters, BF_A and BF_A, aims to join two different sets, S_A and S_B. This is the **OR operation** of these filters and results in a new bloom filter joining both filters. Therefore, the resulted filter union represents a new set $S_A \cup S_B$. It is possible to approximate the size of the expected total insertion in a bloom filter [28]. Equation 4 shows this approximation, where $card(X)$ denotes the number of bits set to "1" in the bloom filter "X". By considering both Equations 4 and 2, it is possible to know the false positive ratio of the union of two bloom filters.

$$n^* = -\frac{m \cdot ln\left(1 - \frac{card(X)}{m}\right)}{k} \tag{4}$$

[1]Assuming perfectly random hash functions

B. Bloom Filters in Recommender Systems

For simplicity reasons and explanation purposes, we will consider items representations using a case study in the domain of movies. However, our approach remains applicable to any other representation and domain.

As explained in the introduction, the description of items can be very large due to the numerous resources, e.g. movies genre, actors, directors, writer, locations and social network tags. This is especially relevant in content-based and hybrid recommendations, and thus we will focus on such techniques. The increasing number of features has three important consequences: (1) the huge size of vectors, (2) the high sparsity in vectors, and (3) the time carried to perform operations among vectors. Indeed, the number of features changes this similarity, and generally the more attributes are used the better the quality of the similarity is. In this vector representation, adding features is equal to adding new dimensions to the vector. However, new features may not appear in other items, which increases sparsity in vectors. And finally, large dimensional vectors make similarity computations slower.

To deal with such quantity of features, some authors propose to perform a features selection which can be supported by domain experts or by using explanatory analysis, such as topic modelling, singular value decomposition or principle component analysis. These techniques look for the most relevant features of items in order to reduce the vectors' size, hence reducing vectors sparsity as well. This reduces the vector representation of items [23]. However, the item similarity measure might suffer from a loss of accuracy since less features have been taken into consideration for this comparison. For example, comparing two movies only by their genre is less accurate than involving actors in the similarity. It is possible to argue that there is an upper bound while adding features to describe items and to compare their similarity. Thus, the quantity and quality of features could be studied by analysing them semantically.

In very large datasets contexts, bloom filters can be extremely beneficial for both representing big sparse vectors and performing fast similarity computations. On the one hand, bloom filter highly reduces the size of item representations while not reducing their description, i.e. the number of features. Contrary to current tendencies, this fact invites to increase the features used for representations instead of selecting/reducing features. On the other hand, the union and intersection bloom filter properties make it easier to perform filter comparisons, and thus one can create a fast similarity measure based on these operations. We propose to use a fast bitwise AND operation to compare common features of items. The reduced size of filters and the fast bitwise operation implies a fast similarity comparison while keeping high fidelity in similarity comparisons. In addition a bitwise XNOR similarity

operation is proposed and takes into account not only common features, but also common missing-features.

In the next section the item representation model based on bloom filters is presented. The bloom filter model, a.k.a. bloom model, as well as the way we use it for computing items similarities are also detailed.

IV. Bloom Filter Similarity Model

In this section, we develop the idea of a similarity model based on bloom filters, the Bloom Filter Similarity Model (a.k.a. Bloom Model). As long as these filters behaviours are based on some parameters and probabilities, we analyse how they affect the similarity process. In addition, comparisons and references to the simple vector similarity model are also given.

One approach to represent items/users is to use a vector of keywords/values of their attributes. As explained before, vectors are large and sparse in big datasets. Some approaches explained in the state of the art deal with this fact by reducing or selecting features. Although, this may lead into a loss of accuracy. In large datasets contexts, we propose to use the bloom filter representations. Therefore, we propose the following context definition:

Definition 1. *Let S_N be the set of all possible items features in a database used in a similarity comparison, so that $|S_N| = N$ is the number of features in this set. Moreover, let S_i be the set of active features of an item "i", and $|S_i| = n$ the number of features in this set. As a consequence, $S_i \subset S_N$. In addition, within this context we assume that $n \ll N$.*

For instance, "N" can be the total number of actors, directors, editors and tags in a database. However, a single movie only has "n" of these features, a.k.a. active features. Normally increasing the features in the database has more impact over "N" than over "n". Indeed, incorporating external information increases the value of N, but only few items will increase their active entries "n" significantly. Thus, vector representations are larger and sparser.

Under these assumptions, we propose the usage of bloom filters for representing items/users as a set of features. Indeed, a bloom filter is a low-size hashed binary vector representing a huge set of elements. Therefore, one item is only represented by one bloom filter and this filter will contain all the "n" active features for this item. Figure 2 represents the difference between the vector similarity model and the bloom model.

Knowing this context, it is possible to build the bloom filters adequately. As explained in Section III-A, one has four parameters to take into account while building a bloom filter: the false positive ratio fp, the number of hash functions k, the size of the bit-structure m and the number of expected insertions n.

On the one hand, items and features are known a priori in a recommender application (the dynamic and incremental situations are later introduced in this section). In this case, as one item is represented by one filter, the number of expected insertions (active features) for this item is also known and represented as n . On the other hand, in a bloom model all

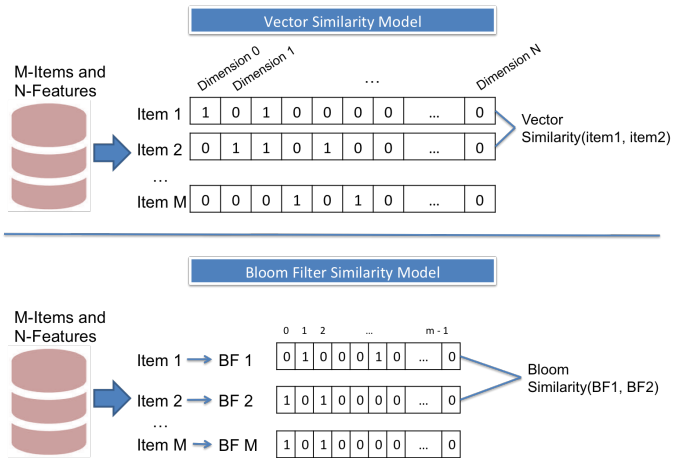

Fig. 2. Vector Similarity Model versus Bloom Filter Similarity Model

the built filters need the same hash functions k and the same size m. As explained in the last section, these conditions allow to perform operations between bloom filters (OR/AND operations). Thus, given n and fixed m and k: (1) the false positive fp increases in each insertion of an element, and (2) the maximum false positive fp_{max} is given by the maximum insertions n.

However, the items may not have the same expected number of active features. In order to keep the maximum desired false positive fp_{max} in the bloom model, it is necessary to look for the item which contains the maximum number of active features n_{max}. Then, one builds every filter according to n_{max}, m and k. This is the necessary condition to use a bloom filter similarity model. In the following sections, we discuss the operations to perform a similarity measure based on the item bloom filter representations.

A. Trade-Offs

The bloom model clearly depends on the parameters of bloom filters. In fact, the intersection and union of two filters are bit-to-bit operations over which the parameters m (bloom filter size) and k (number of hash functions) have relevant impacts. Our similarity model relies on these properties, and similarity matchings are affected by these parameters, and in general, by the resulted false positive of filters. In this section we explain the trade-offs of using different parameter configurations and the impact on the false positive similarity.

The size m of a bloom filter affects the false positive probability. Larger filter sizes reduce the probability of taking a specific bit $(1/m)$. Therefore, the false positive is reduced. The number of hash functions k has also a great impact on the false positive probability. Higher values may reduce rapidly the false positive, but they will quickly fill up the bloom filter. In addition, this parameter particularly affects a bit-to-bit similarity. In fact, the number of shared bits per insertion (m/kn) should be reduced. As long as insertions are associated to k-bits in the bloom filter, higher values in a fixed filter size

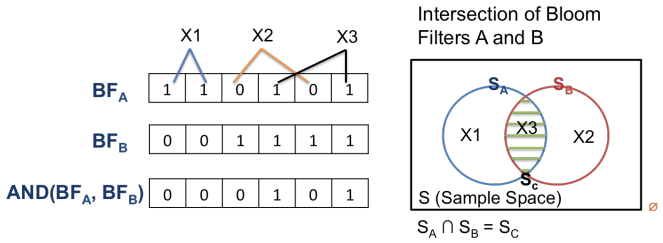

Fig. 3. AND intersection of two bloom filters. BF_A contains two inserted elements: X1 and X3.BF_B contains two inserted elements: X2 and X3. Thus, the intersected filter contains only one inserted element: X3.

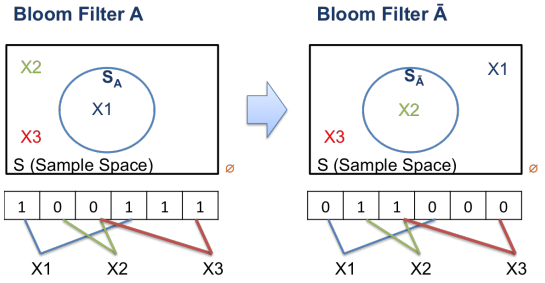

Fig. 4. Negation of a Bloom Filter (BF). Element X1 is inserted in BF_A. Elements X2 and X3 are not inserted in BF_A. The negation of BF_A, $BF_{\overline{A}}$, contains the element X2 and does not contain the element X1. Yet, X3 is not inserted because it shares a bit with an element which is inserted in BF_A.

makes it easier for elements to share some bits positions. As a result, a true similarity match is more complicated to find.

This m and k trade-off can be seen as a space-time trade-off. Space is represented by the size of the filter. On the contrary, k is the number of hash function to execute, and hence, higher values entail more time-consumption. Thus, correctly choosing these parameters is crucial in order to have a good performance in a bloom similarity model. However, the choices depend on the requirements of the application and the size of datasets. One suggested approach may be to use larger filters with fewer hash functions to improve similarity matchings in fast bit-to-bit comparisons.

B. Bitwise AND Similarity

The intersection of two bloom filters performs a bitwise AND operation between two filters (section III-A2) to find out **common insertions in both bloom filters**. Intuitively, one element which is inserted into two different bloom filters will activate the same bits positions in both structures. Figure 3 gives an example to represent this fact. Thus, we define a bitwise AND similarity as:

Definition 2. *The AND similarity between two items, A and B, represented by two bloom filters,* $BF_A = \vec{a}$ *and* $BF_B = \vec{b}$*, is given by:*

$$sim(A, B) = card(\vec{a} \cap \vec{b}) = \vec{a} \cdot \vec{b}$$

In our case, filters represent items and insertions are their active features. Due to the fewer size of filter vectors and the fast bitwise AND operation, one can rapidly compare the similarity of two very large item descriptions. This measure might replace the typical vector comparisons based on cosine or jaccard measures in recommender systems.

Equation 3 represents *the probability that a specific bit is set to one* in an intersected filter. Although, this can be reduced by considering the condition for the bloom model: every filter is built with same the parameters n_{max}, m and k. Filter A represents a set S_A, where $|S_A| = n_A$ is the number of inserted elements. Similarly, filter B represents a set S_B, where $|S_B| = n_B$. Therefore, one may apply that we know the maximum expected insertions: $n_{max} = n_A = n_B$. Moreover, this limits the maximum cardinality of the intersection, since $|S_A \cap S_B| \leq n_{max}$. As a consequence, the *probability that a specific bit is set to one in both filters* is given by :

$$q(bit = 1) = \left(1 - (1 - \frac{1}{m})^{k \cdot n_{max}}\right)$$

Thus, we define the maximum error in the similarity (a.k.a false positive similarity) in Equation 5. Note that this error highly depends on k, due to the bit-to-bit comparisons.

$$fp = q(bit = 1)^k = \left(1 - (1 - \frac{1}{m})^{k \cdot n_{max}}\right)^k \quad (5)$$

C. Negating a Bloom Filter

Negating a bloom filter will help us to perform and explain the XNOR similarity measure. Theoretically, the negation of a filter BF_A ($BF_{\overline{A}}$) results in another filter where each entry takes the complementary value of BF_A. Thus $BF_{\overline{A}}$ might only contain the non-inserted elements of BF_A. However, due to the k hash functions, insertions may share at most k-bits in the filter BF_A. Therefore, negating a bloom filter has two consequences: (1) inserted elements will appear as non-inserted elements, and (2) non-inserted elements may appear as inserted elements only if all their k indexes are set to 0. Otherwise, non-inserted elements also appear as non-inserted in the negated filter. This fact is represented in Figure 4. However, this problem might be minimized by reducing the number of shared bits per element (m/kn). In addition, since the number of bits set to 1 has changed, the false positive of $\overline{A}$ also changes. Yet, it is given by the *probability that a specific bit is zero in the original filter*: $fp = p(bit = 0)^k$ (Equation 1).

D. Bitwise XNOR Similarity

Typically, similarity methods measure the intersection of elements. The AND intersection takes common-inserted elements of two sets into account. That is, given two sets, $S_A = \{x1, x3\}$ and $S_B = \{x2, x3\}$, the intersection of both sets is $S_A \cap S_B = \{x3\}$. However, sets S_A and S_B have more in common than this intersection. Actually, the element $x4$ is not in any of these sets, and hence, $x4$ is a common non-inserted element.

As a consequence, we propose to go further in the similarity by considering also the common non-inserted elements. In fact, this similarity in **common-insertions and common**

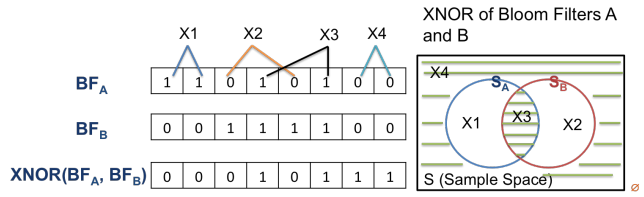

Fig. 5. XNOR intersection of two bloom filters BF_A and BF_B. BF_A contains two inserted elements: X1 and X3. BF_B contains other two inserted elements: X2 and X3. Thus, the XNOR operation will result in common inserted elements and common non-inserted elements: X3 and X4.

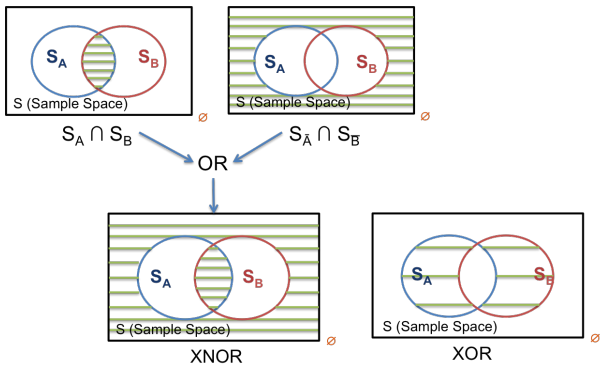

Fig. 6. Representation of how the XOR and XNOR operations can be extracted from AND and OR operations.

missing-insertions is the bitwise XNOR operation between two filters. This bit operation sets to one the bits which have similar bit-status. Notice that it is also necessary that every filter is built with the same parameters n_{max}, m and k. Figure 5 shows this operation.

Indeed, the XNOR operation between items A and B, represented by two sets, S_A and S_B, is the union of two AND operations: $XNOR(A,B) = S_A \cap S_B \cup S_{\overline{A}} \cap S_{\overline{B}}$. Figure 6 graphically shows a set oriented XNOR and its complement XOR operation.

We can observe that XNOR similarity is composed of two AND similarities: common-inserted elements similarity and common-non-inserted elements similarity. Thus, we can compute separately these two AND similarities (explained in the bitwise AND similarity) and merge them. In addition, we propose to weight such similarities by using the weight α.

Definition 3. *The XNOR similarity between two items, A and B, represented by two bloom filters, $BF_A = \vec{a}$ and $BF_B = \vec{b}$, and with a weight of α is given by:*

$$sim(A,B) = \alpha \cdot card(\vec{a} \cap \vec{b}) + (1-\alpha) \cdot card(\overline{\vec{a}} \cap \overline{\vec{b}})$$
$$= \alpha \cdot \vec{a} \cdot \vec{b} + (1-\alpha) \cdot \overline{\vec{a}} \cdot \overline{\vec{b}}$$

By varying α, one might avoid the effect of too many common non-insertions in the second part of the similarity. Note that if $\alpha = 1$, this measure becomes the AND bitwise similarity measure explained above.

It is possible to argue that two items may have infinite common non-inserted elements. For instance, a rock and a paper do not have a priori common properties, hence the similarity could be potentially infinite. However, it does not mean they are as similar as an orange and a mandarin can be, and indeed they also have infinitely uncommon features. Thus, a dataset containing rock, paper, orange and mandarin may represent a problem for this measure. Yet, this fact does not happen when the domain of recommendation is fixed and the features to use are limited. In case of multiple recommendation domains, a semantic analysis can be performed in order to limit the features.

The false positive similarity can be also computed. For simplicity, let's consider the complementary probability $P(XOR) = 1 - P(XNOR)$ (fact shown in Figure 6):

$$P(XOR) = P(BF_A) + P(BF_B) - P(BF_A \cap BF_B)$$

Where $P(BF_A)$ and $P(BF_B)$ are the already known *probabilities that a specific bit is still one*, $q(bit=1) = 1 - p(bit=0)$ in filters BF_A and BF_B, as explained in section III-A1. Moreover, $P(BF_A \cap BF_B)$ is the *probability that a specific bit is set to one* in both filters, explained in section III-A2. Taking into account the bloom model condition $n = n_{max}$, one may apply that $q(bit=1) = P(A) = P(B) = P(A \cap B)$. Hence, the *probability that a specific bit is set to one* in the final resulted vector in the XOR operation is $P(XOR) = q(bit=1)$. As a result, the *probability that a specific bit is set to one* in the XNOR operation is:

$$P(XNOR) = 1 - q(bit=1) = \left(1 - \frac{1}{m}\right)^{k \cdot n_{max}} \quad (6)$$

Thus, the false positive similarity is given by $P(XNOR)^k$. Operations between identical filters are a singular case. Since all possible elements are represented, this operation results in a totally full vector of 1 being the highest similarity. Hence, membership queries return always true.

E. Shortcomings and solutions

In this section we have explained a very simple case of representing items based on the standard definition of bloom filters, where we have supposed a static dataset. In fact, when new features need to be taken into consideration the number of insertions raises. However, the current bloom filter parameters (size "m" and hash functions "k") can not face such raise and yield to a false positive raise as well. Indeed, these parameters cannot be modified, hence when new features are added to the system all bloom filters need to be re-built according to new parameters. In addition, standard bloom filters do not count the number of insertions of the same element, if necessary. Thus, these filters can not be compared to frequency vectors.

In fact, these situations correspond to static versus dynamic datasets and binary versus frequency vector representations. In our model we use standard bloom filters for simplicity

and explanation, yet these drawbacks can be addressed by other bloom filter approaches. On the one hand, dynamic datasets are possible to model by using dynamic/scalable bloom filters [29], [30]. The main idea is that a scalable bloom filter is formed by one or more bloom filters. Thus, these filters are built by blocks, and hence by adding new blocks to the filter one may add a new set of features. As a consequence, this avoid to re-build the bloom model when new features are added. In addition, similar probability inductions given in this section can be applied to scalable and dynamic probabilities, and thus our bloom model remains possible in these cases. On the other hand, counting insertions is possible as well by using counting bloom filters [3]. These filters contains new bit-sets to count the insertions performed. Thus, if one insertion has been made several times, filters may approximate the counting. As a result, one may compare counting filters to frequency vectors, with more complexity. Other variances of bloom filters deal with extra compression [3].

In this paper we consider using standard bloom filters due to the nature of our dataset and experimentations, in order to have an easy comparable dataset and reproducible experimentations. This may be complicated in other dynamic circumstances, e.g. bloom filters in very large datasets using multiple resources, where non public datasets are accessible.

V. Experimentation

In our experimentations, we use the publicly availables MovieLens dataset and IMDb database, both used in the recommendation field. An already merged dataset is given by GroupLens [9]. The dataset is composed of 2113 users, 10197 items, 855598 ratings, 6 features and 104957 possible different values, which are all the available features used to describe our items. In this experimentation we only focus on similarity tasks. We aim at comparing the performance of the bloom model against two other models namely: a vector model[2] and an order reduction model. The former uses a binary/boolean representation of an item. It takes into account all available features. The latter tries to reduce the size of features by performing a SVD analysis. These three models are compared taking four factors into consideration: (1) size, compression or space complexity, (2) operation time and model time, (3) the similarity fidelity of the AND measure, and (4) the similarity accuracy of the XNOR measure. The results and the bloom model may be used by recommenders to improve item representations by reducing the size of vector descriptions and by adding new features not taken yet into account.

This section goes through two kinds of tests. On the one hand, we aim to prove that the filter similarities in bloom model are close to the ones computed by vector similarities (using cosine or jaccard measures) in the vector model. On the other hand, we aim to prove the pertinence of bloom filters in recommender systems. Hence, we compare the accuracy in the response of the system to several top-K similarities.

[2]We use typical vector definition. Highlight that sparse map implementation of vectors may reduce space complexity, yet it increases programming and time complexity in terms of vector similarities.

TABLE I
Optimal Bloom Filters.

Bloom Filter	n	fp	k	m
bloom 0.2	237	0.2	3	794
bloom 0.1	237	0.1	4	1136
bloom 0.01	237	0.01	7	2272
bloom 0.001	237	0.001	10	3408

For example, the system retrieves the top 5, 10, 20, 50, 100, 150, 200, 300 and 500 most similar elements to a given one. Notice that a simple top 20 or top 50 is enough in most of recommendation contexts. Then, we compare if the similarity models return similar top-K items.

The results show that the bloom model reduces the size of vectors up to 97%, keeping a similarity accuracy of 98%. In addition, bloom model outperforms feature reduction methods in terms of similarity fidelity and time performance.

A. Bloom Filter Representation

As explained before, items/users might be represented by vectors of attributes. Typically, these vectors may become too large and sparse depending on the available features. Some authors deal with this problem by reducing the quantity of features in the representation of items. Hence, a feature selection or a dimensional reduction analysis are performed. These choices might cause a loss of item description, and thus, a loss of accuracy in item similarities. Bloom filters have the capacity of representing large sets in low space. This fact allows to fully describe items/users through all available features by using a low space structure.

1) Trade-Off: Compression and Time Analysis: The experimentation dataset has 104957 available features, hence item vector size is $N = 104957$. However, the number of active entries in these big vectors is very reduced. In fact, the item 3246 has the maximum number of active entries over all items, $n = n_{max} = 237$ (notice that $n_{max} << N$). This shows the minimum sparsity of a vector and the worst bloom filter case (due to the maximum inserted entries).

In order to reduce the size of the vectors and still keep items representation and items similarities, we use bloom filters. The goal is to get bloom filter sizes fewer than N, thus $m < N$. Two different cases have been tested:

(1) Optimal bloom filters in Table I are built by fixing both maximum insertions n_{max} and the desired false positive fp. Notice that in the case of using a false positive of 0.001 we obtained 3408 bits, which represents almost 3% of N, thus around 97% of size reduction. This tiny false positive and reduced space is given by the 10 hash functions.

(2) Non-Optimal bloom filters in Table II are built by fixing insertions (n), filter size (m) and hash functions (k). In this case, the m and k values have been chosen by varying the last optimal computed bloom filter ($m = 3408$ and $k = 10$). Notice that for the same false positive 0.001, we obtained a size of 7000 bits, which represents almost 6% of N by only using 3 hash functions.

TABLE II
NON OPTIMAL BLOOM FILTERS

Bloom Filter	n	fp	k	m
bloom M3000K3	237	0.01	3	3000
bloom M3000K4	237	0.005	4	3000
bloom M7000K3	237	0.001	3	7000
bloom M7000K4	237	0.00026	4	7000

TABLE III
VECTOR AND BLOOM MODELS TRADE-OFFS.

Item Representation	VM	OPM	NOPM
Size (bits)	104957	3408	7000
Hash Functions	-	10	3
Building model (sec)	6.58	8.94	8.32
AND Similarity (ms)	-	0.001	0.003
XNOR Similarity (ms)	-	0.004	0.006
Reconstruction (ms)	-	127.28	141.27
Jaccard Similarity (ms)	0.4	2.6	2.3

Actually, the more hash functions are used, the less false positive error one achieves. However, the filter fills more quickly as well, and thus, cardinal similarities (such as AND or XNOR similarities) might fall into similarity-mistakes. In addition, the more hash functions to perform, the slower is the system. On the contrary, increasing the size of filters better distribute hashed insertions but more space is needed.

Table III shows interesting comparisons in terms of space and time-consumption. The machine used is a MacOS 4Go RAM with 2 cores (2.53GHz). Three models are compared: Vector Model (VM), Optimal Bloom Model (OPM) and Non-Optimal Bloom Model (NOPM). To build vectors and filters, a database access was required. Query time was around 200 ms (not included in these results). One may observe that building bloom model takes extra time due to the hash functions, however, this is acceptable as there is a high space reduction. In addition, operations among bloom models can perform faster due to two facts: (1) bitwise operation are faster than jaccard/cosine similarities, and (2) bitsets are smaller.

Moreover, it might be interesting to reconstruct a vector from a bloom filter to compare vector and bloom models under the same conditions and similarities. Reconstruction time is given by the features loading (140 ms) and the features hashing (145 ms), which are performed only once. Then, a single vector reconstruction is created by querying filters. This is also shown in Table III.

2) Singular Value Decomposition (SVD) model: SVD is a factorization model that decomposes the big matrix in three smaller matrices (left-eigenvectors, eigenvalues and right eigenvectors) in such a way that the multiplication of the three is an approximation of the original big matrix. By using this, the rank of the big matrix may be reduced by decreasing the matrix sparsity as well.

We applied SVD to the set of items-features to reduce sparsity and to find an accurate reduced matrix representation. We compared this model against the vector model and bloom model in terms of data compression and fidelity in item similarities. To have a fair comparison between SVD and bloom models, the rank of the SVD matrix should not be above the 300 non-zero eigenvalues due to the data precision of the matrix: each eigenvalue has double precision by using a 32 bits representation. Thus the total rank representation may use 9600 bits, which is already larger than the 7000 bits of our biggest bloom filter.

The comparisons of our bloom model were stopped due to two facts: (1) the large dataset made the SVD model very timely expensive, and (2) the item similarities that one may achieve by using the reduced SVD model were not accurate. Indeed, the items-features matrices are so sparse that the SVD can not find a very low rank model. This may be solved by increasing the rank of the matrix, yet the bloom model have already demonstrated very good similarity fidelity in a much more reduced item space representation.

B. Fidelity of the AND similarity

In this section we evaluate the accuracy of the AND Similarity measure. This computes the similarity between two bloom filters, and thus two items, by performing the bitwise AND operation. First, the AND bitwise operations among filters (two by two) is performed. There are two ways to exploit such results: the *cardinality representation* and the *set representation* of the bloom filter. The former compares two filters by using their bit-set vectors. It is fast but it highly depends on low values of k. The latter compares two filters by their set representation. In fact, it reconstructs the AND intersected bloom filter into a N dimensional vector by querying it. Hence, it is slower, yet it allows to compare bloom model and vector model similarities under the same conditions, since items in both models are represented as vectors of dimension N. In this section we will use both set and cardinality representations of the bloom filter.

On the one hand, we reconstruct the set representation of the resulted AND operation. The first test aims to check whether the bloom model is loyal to the similarity degree of items. Hence, we compare item similarities in both models: vector similarity model (using jaccard similarity) and bloom similarity model (using AND bitwise similarity). We focus on item 3426 which has more active features, and thus the bloom filter with more insertions. This is our worst case comparison, since it is more likely to have bits conflicts in a filter comparison. Figure 7 shows the degree of similarity of both approaches. It demonstrates the high fidelity in the similarity of the bloom model (case bloom 0.001 in Table I) while reproducing the vector model jaccard similarity.

On the other hand, one can use the cardinality of the resulted bloom filter intersection. Thus, the hash functions k highly affect this similarity due to the shared bits of insertions. Figure 8 shows an optimal bloom filter case where the high value of k causes the points sparsity. In fact, high values of k associate inserted elements to more number of bits. Thus, it makes difficult it to do comparisons based on the cardinality of the intersection. Increasing m and decreasing k the bloom filter fills up slowly with insertions, and thus better cardinality

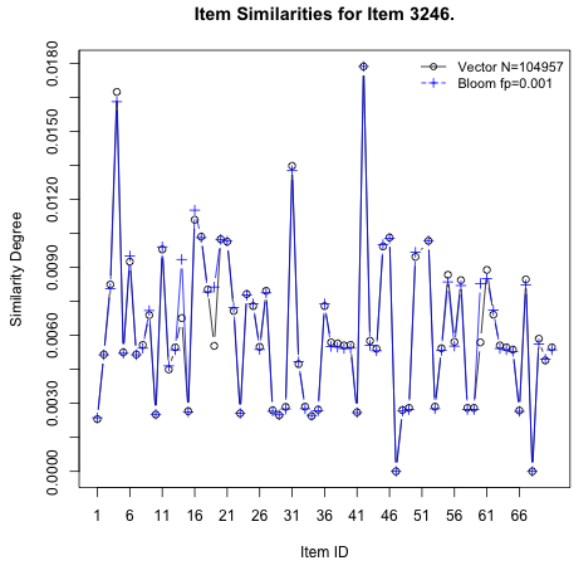

Fig. 7. Degree of Similarity of the item 3246 with several items. Bloom Filter (n=237, fp=0.001) has been reconstructed into a vector of size N.

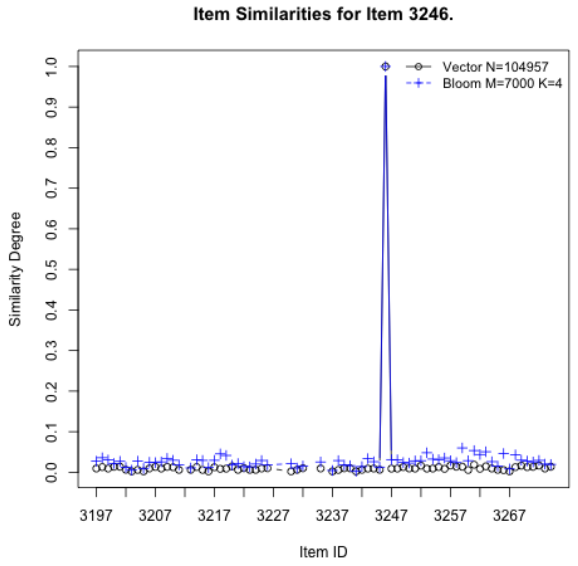

Fig. 9. Bloom Filter (m=7000, k=4). Trade-Offs for a Bloom Similarity Model.

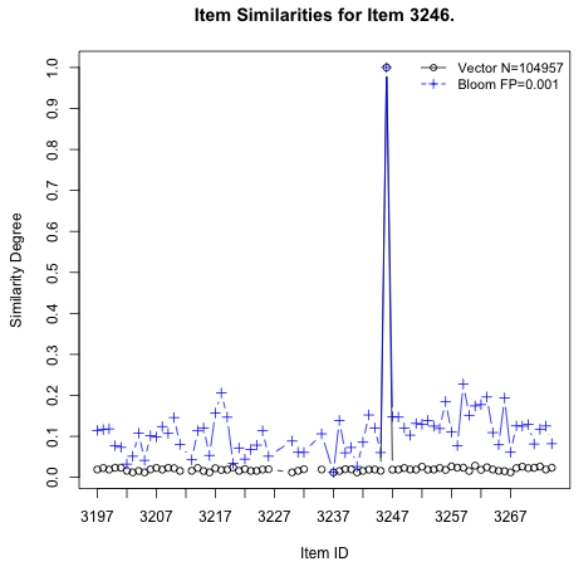

Fig. 8. Bloom Filter (m=3408, k=10). Trade-Off for a Bloom Similarity Model.

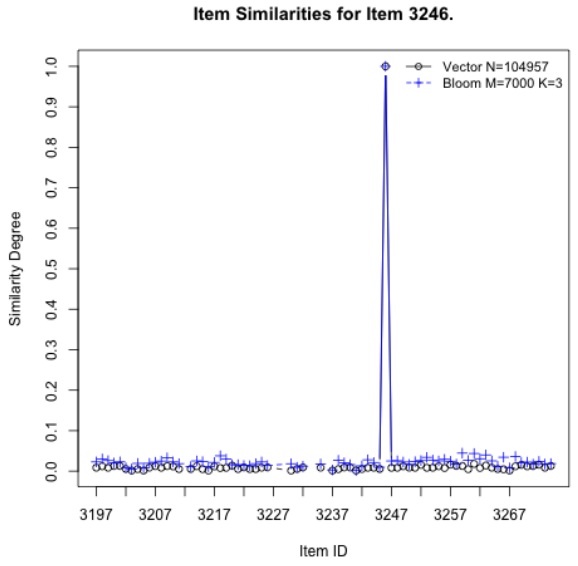

Fig. 10. Bloom Filter (m=7000, k=3). Trade-Off for a Bloom Similarity Model.

comparisons are possible as shown in Figure 10. Finally, in Figure 11 we made a zoom over few values in the last configuration to better appreciate these values.

Similarity models can be loyal while reproducing such item similarities, yet tiny differences may have big differences in top-K comparisons. We aim to compare the capacity to find out the K-most similar items. Thus, we request the vector similarity model to return several top-5, 10, 20, 50, 100, 150, 200, 300 and 500 most similar items by using the jaccard similarity. The goal of the bloom filter similarity model is to reproduce such tops: the same items should appear in both tops (in this test the order of items is not taken into account). As a result, by comparing tops, one knows the correct presence of items (True Positive (TP)), the correct absence of items (True Negative (TN)), and the errors (False Negative (FN) and False Positive (FP)). As a consequence, the accuracy of the system is defined by Equation 7:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

The values for this accuracy vary between 0 and 1, where

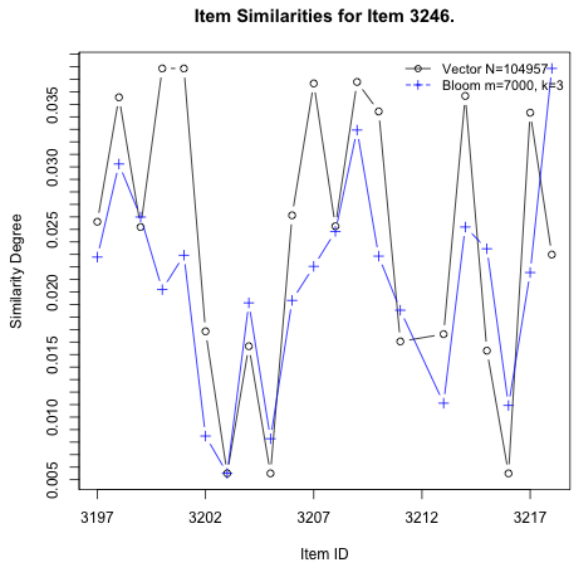

Fig. 11. Zoom Bloom Filter (m=7000, k=3). Trade-Off for a Bloom Similarity Model.

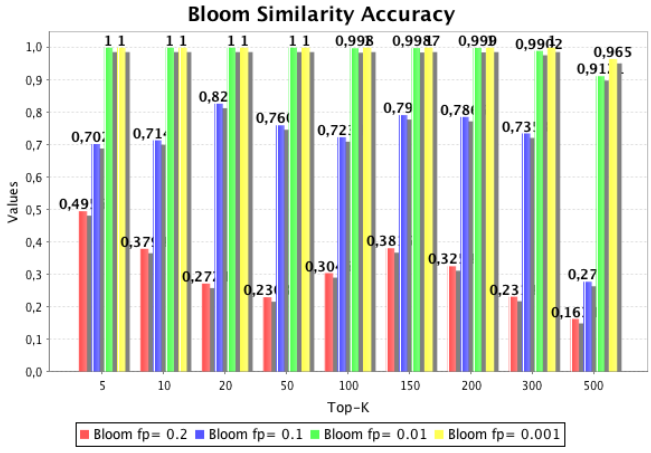

Fig. 12. Accuracy of the Bloom Filter Similarity Model against the Vector Similarity Model. Bloom Filters with fewer false positive values achieve better accuracy in tops comparisons.

the value of 1 is the highest possible accuracy.

In order to compare both similarity models under the same conditions, we reconstruct the bloom filter representations into vector representations of size N. As the bloom filter model depends on the false positive of bloom filters, the comparisons are done over the cases in Table I. Figure 12 shows the results for these tops comparisons. Notice that bloom filter representations may achieve almost a perfect accuracy fidelity in very reduced sizes. In fact, the bloom filter with a false positive of 0.001 and 3408 bits achieves an almost perfect score in these presented top similarities.

C. The XNOR similarity

In this section we use a XNOR Similarity measure in order to compare items. This measure shows common insertions in

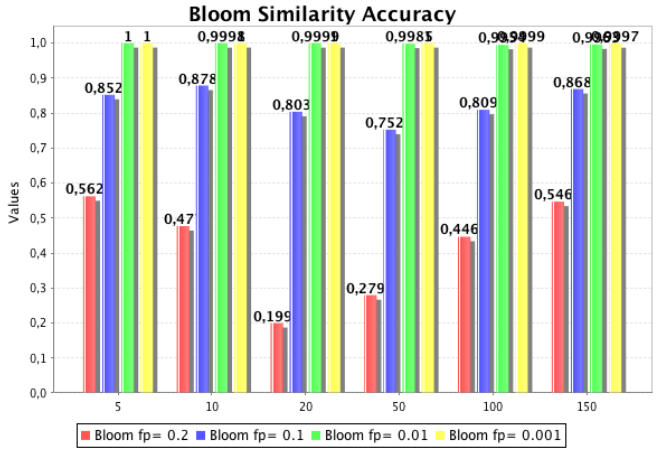

Fig. 13. Accuracy of the Bloom Filter Similarity Model against the Vector Similarity Model. This similarity uses the XNOR operation.

two bloom filters, but it also shows the features that likely have not been inserted in any of the bloom filters. That is, common insertions and common non-insertions. In this test we compare whether the bloom model correctly find similar items by using this XNOR operation. For this purpose, we first compute the XNOR operation in vector models. Then, we compare this to the XNOR operation in the bloom filter similarity model.

The comparisons are performed by reconstructing bloom filters as vectors of N dimensions. As the bloom model depends on the false positive of bloom filters, the comparisons are done over the cases shown in Table I. The XNOR similarity equally balances the similarity of common insertions and common non-inserted elements. That is, $\alpha = 0.5$. However, one may change the weight in this similarity by giving more relevancy to one aspect or another, e.g. giving more relevancy to common inserted elements versus common non-inserted elements. By doing this, results tend to approach the AND similarity measure.

The evaluation is again made in terms of accuracy. We compare both XNOR top-K looking for the presence and the absence of items among the top-K. The results of this similarity is shown in Figure 13. One may see how the bloom model correctly finds the same similarities given by the vector model in much fewer sizes.

VI. Conclusion

The items/users in recommender systems techniques, such as content-based and hybrid methods, are usually represented by a large set of features [1]. Hence, it is possible to compute similarities among items/users that will help to achieve pertinent recommendations. However, the accuracy of the similarity depends on the selected features and their number. Generally, the more features are used the better the similarity might be. In addition, items' features are typically represented by vectors, which are large and sparse. Hence, increasing vectors' sizes makes similarity computations slower. We propose in this paper a new method for representing the items/users based on

bloom filters that allows to: considerably improve the space complexity, to perform faster bitwise operations and to keep the fidelity in the similarity among items.

This paper is focused on three aspects: (1) the reduction capacity of bloom filters in a recommender system context, (2) the usage of AND operations as a similarity measure in bloom filters to consider common insertions, and (3) the usage of XNOR operations as a similarity measure that takes into account not only common inserted items, but also common non-inserted items.

The experimentations performed on a public dataset [9] show that the bloom filter representation highly reduces the size of vector representations (94-97% per vector) while keeping a high fidelity in the item similarity (accuracy of 98%) in comparison with standard approaches.

Finally, new experimentations to evaluate the impact of the AND and the XNOR bloom similarities in a real recommender systems are on-going. The current experimentations are limited by the dataset which contains 6 features and 104957 possible different values to describe items. However, we aim to test it on extremely large datasets. Therefore, the future work focus on: (1) to study the effect of these similarities in a complete recommendation context, (2) to analyse the trade-off between similarity accuracy and features bounds, (3) to test on larger and heterogeneous datasets, and (4) to study the scalability of the similarity based on bloom filters. In addition, other bloom filters, such as counting bloom filter and dynamic bloom filters, are part of future experiments. Those are all important tasks in recommender systems in order to improve recommendations and to make these systems more scalable.

ACKNOWLEDGMENTS

This work has been supported by FIORA project, and funded by "DGCIS" and "Conseil Regional de l'Île de France".

REFERENCES

[1] P. B. Kantor, F. Ricci, L. Rokach, and B. Shapira, "Recommender systems handbook," in *Recommender systems handbook*, Springer. Springer US, 2011, p. 848.

[2] K. Zhou, S.-H. Yang, and H. Zha, "Functional matrix factorizations for cold-start recommendation," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 2011, pp. 315–324.

[3] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

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

[5] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[6] J. Karim, "Hybrid system for personalized recommendations," in *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*. IEEE, 2014, pp. 1–6.

[7] E. Peis, J. M. del Castillo, and J. Delgado-López, "Semantic recommender systems. analysis of the state of the topic," *Hipertext. net*, vol. 6, pp. 1–5, 2008.

[8] R. Dahimene, C. Constantin, and C. du Mouza, "Recland: A recommender system for social networks," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 2014, pp. 2063–2065.

[9] I. Cantador, P. Brusilovsky, and T. Kuflik, "2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011)," in *Proceedings of the 5th ACM conference on Recommender systems*, ser. RecSys 2011. New York, NY, USA: ACM, 2011.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[11] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering." in *SDM*, vol. 5. SIAM, 2005, pp. 1–5.

[12] T. Hofmann, "Collaborative filtering via gaussian probabilistic latent semantic analysis," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. ACM, 2003, pp. 259–266.

[13] Y. Koren, "The bellkor solution to the netflix grand prize," *Netflix prize documentation*, 2009.

[14] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[15] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender Systems Handbook*. Springer, 2011, pp. 145–186.

[16] A. Tiroshi, T. Kuflik, J. Kay, and B. Kummerfeld, "Recommender systems and the social web," *Advances in User Modeling*, pp. 60–70, 2012.

[17] R. Boim, T. Milo, and S. Novgorodov, "Direc: Diversified recommendations for semantic-less collaborative filtering," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 1312–1315.

[18] Y. B. Fernández, J. J. P. Arias, M. L. Nores, A. G. Solla, and M. R. Cabrer, "Avatar: An improved solution for personalized tv based on semantic inference," *Consumer Electronics, IEEE Transactions on*, vol. 52, no. 1, pp. 223–231, 2006.

[19] P.-Y. Pan, C.-H. Wang, G.-J. Horng, and S.-T. Cheng, "The development of an ontology-based adaptive personalized recommender system," in *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, vol. 1. IEEE, 2010, pp. 1–76.

[20] G. Katz, N. Ofek, B. Shapira, L. Rokach, and G. Shani, "Using wikipedia to boost collaborative filtering techniques," in *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 2011, pp. 285–288.

[21] D. Werner, C. Cruz, and C. Nicolle, "Ontology-based recommender system of economic articles," *arXiv preprint arXiv:1301.4781*, 2013.

[22] M. G. Vozalis and K. G. Margaritis, "Using svd and demographic data for the enhancement of generalized collaborative filtering," *Information Sciences*, vol. 177, no. 15, pp. 3017–3037, 2007.

[23] B. Mobasher, X. Jin, and Y. Zhou, "Semantically enhanced collaborative filtering on the web," in *Web Mining: From Web to Semantic Web*. Springer, 2004, pp. 57–76.

[24] M. Pozo, R. Chiky, and E. Métais, "Extraction de l'intérêt implicite des utilisateurs dans les attributs des items pour améliorer les systèmes de recommandations," in *15ème conférence internationale sur l'extraction et la gestion des connaissances*. IEEE, January 2015, pp. 1–6.

[25] S. Geravand and M. Ahmadi, "An efficient and scalable plagiarism checking system using bloom filters," *Computers & Electrical Engineering*, vol. 40, no. 6, pp. 1789 – 1800, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790614001712

[26] N. Jain, M. Dahlin, and R. Tewari, "Using bloom filters to refine web search results." in *WebDB*, 2005, pp. 25–30.

[27] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[28] S. J. Swamidass and P. Baldi, "Mathematical correction for fingerprint similarity measures to improve chemical retrieval," *Journal of chemical information and modeling*, vol. 47, no. 3, pp. 952–964, 2007.

[29] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 1, pp. 120–133, 2010.

[30] P. S. Almeida, C. Baquero, N. Preguiça, and D. Hutchison, "Scalable bloom filters," *Information Processing Letters*, vol. 101, no. 6, pp. 255–261, 2007.