Scalable and adaptive collaborative filtering by mining frequent item co-occurrences in a user feedback stream

A. Murat Yagci\textsuperscript{a,}\textsuperscript{*}, Tevfik Aytekin\textsuperscript{b}, Fikret S. Gurgen\textsuperscript{a}

\textsuperscript{a}Department of Computer Engineering, Bogazici University, 34342, Bebek, Istanbul
\textsuperscript{b}Department of Computer Engineering, Bahcesehir University, 34349,Besiktas, Istanbul

Abstract

Neighborhood-based methods are one of the mainstream approaches to collaborative filtering. A common problem with these methods is scalability to large number of users and items. Consequently, the adaptivity of a neighborhood-based model to system dynamics is often compromised due to model constraints and prolonged training intervals. These drawbacks can be important in designing demanding applications of today and the future. In this paper, we propose a novel real-time scalable and adaptive collaborative filtering algorithm, SASCF, suitable for personalized and item-to-item recommendations, in which the underlying neighborhood-based model is updated on-the-fly with the streaming user feedback. The algorithm does not perform an offline search for finding nearest neighbors in a full item similarity matrix. Instead, taking a landmark window over the user feedback stream, a space-efficient summary structure is maintained. This structure corresponds to the result of a standing iceberg query for finding every item’s top-$k$ frequently co-occurring items over a specified support threshold. Mining such frequent co-occurrences can facilitate approximate computation of several useful item similarity measures. The algorithm offers scalability thanks to the space-efficient summary structure which handles ever-changing users, items, and item similarities in a resource aware fashion. It also offers adaptivity in the sense that newly arriving user-item interactions are immediately integrated into the model. The model is always up-to-date and it can readily be used to recommend items to users with the most recent information.

Keywords: Collaborative filtering, Recommender systems, Data stream mining, Real-time intelligent systems, Nearest neighbor search

1. Introduction

Collaborative filtering is an information filtering technique based on past interactions between system users (or agents) and items rather than specific user or item features. It has been widely and successfully used in recommender systems (Ricci et al., 2015). Accuracy, recall, diversity, robustness, and privacy preservation are among commonly desired performance criteria of such systems. Moreover, in today’s demanding applications, scalability and adaptivity are two additionally important performance criteria. The former refers to system’s ability to deal with large collections of users and items together with their massive interactions. The latter may refer to a faster response to dynamic nature of these collections due to new users, new items, and recent user interactions with the system, as well as adaptation in the presence of concept drift. Fulfilling these criteria facilitates recommendation of large-scale durable items as well as fast-moving items, for example, in news and social networks.

Collaborative filtering can be used for personalized or item-to-item recommendations. Ways for achieving personalization include recommending to a user top-$N$ similar items to her item preference history, or recommending to a user items by looking at item preferences of similar users. Without the personalization step, collaborative filtering can still be used to define a similarity between items based on item preferences of many collaborating users, which enables item-to-item recommendations.

Neighborhood-based methods (Ricci et al., 2015, Chapter 2) are one of the mainstream approaches to collaborative filtering. A common problem of these methods is scalability (Deshpande and Karypis, 2004; Vinagre et al., 2015). Consequently, the adaptivity of a neighborhood-based model to system dynamics is often compromised due to model constraints and prolonged training intervals with the increasing numbers of users, items, and their interactions. These drawbacks can be important in many demanding applications and they are our first motivation for this study.

One way to approach problems in achieving scalability and adaptivity is to design collaborative filtering algorithms using the data stream model. Data stream mining (Leskovec et al., 2014; Gama, 2010; Liberty and Nelson, 2012) uses this model to maintain informative and space-efficient summaries of data by performing incremental updates as it streams. The summaries and the associated mining algorithms can facilitate resource aware computing, and they are useful for both server-side massive scale stream processing and in-device processing where computational resources are scarcer. Moreover, processing data gradually as it arrives can often make a more efficient

\textsuperscript{*}Corresponding author

Email address: murat.yagci@boun.edu.tr (A. Murat Yagci)

Preprint submitted to Engineering Applications of Artificial Intelligence
use of available computational resources compared to batch processing (Miner and Shook, 2012). Therefore, our second motivation for this study is that although data stream mining approaches can be used for scalability and adaptivity to temporal dynamics, such approaches are not common (Vinagre et al., 2015) in the case of neighborhood-based collaborative filtering.

In this work, we propose a novel real-time neighborhood-based collaborative filtering algorithm by extending ideas from data stream mining to improve scalability and adaptivity. The interactions between users and items are assumed to be binary, that is, a positive interaction or no interaction at all which is a very common real life scenario (Gantner et al., 2011; Vinagre et al., 2014). If there is extra assessment information in the stream, such as a rating, it can be used to define a binarization threshold. Instead of an item similarity matrix, the algorithm maintains very compact summaries of frequent co-occurrences of every item. These summaries are achieved by single pass frequent item finding algorithms for streams which can also be tailored to return approximate top-\(k\) frequent item co-occurrences readily on demand. To the best of our knowledge, ours is the first work applying this family of algorithms to neighborhood-based collaborative filtering. The item similarity matrix compression ratio achieved by the resulting summaries can be drastic as given in Section 4. The mined co-occurrences can facilitate approximate computation of several useful similarity measures to decide nearest neighbors (NNs) of an item. The proposed collaborative filtering approach is suitable for personalized as well as non-personalized item-to-item recommendations. The main contributions of this paper are as follows:

- Extending frequent item finding algorithms in streams to efficiently finding top-\(k\) frequent co-occurrences in streams. These co-occurrences are always up-to-date and they are shown to facilitate computation of several useful item similarity measures.

- A novel scalable and adaptive neighborhood-based collaborative filtering algorithm, SASCF, which can be used for personalized or item-to-item recommendations with the most up-to-date information in the system. The algorithm is resource aware in the sense that it can use available system resources through its control parameters to produce intelligent results in real time.

- We show theoretical and empirical results for the compression of a full item similarity matrix using the proposed summary structure. The structure additionally holds top-\(m\) item co-occurrences always in sorted order with respect to their approximate counts.

- We show empirically that the co-occurrences of items in sorted order demonstrate a power-law distribution. We also show that the support thresholds for catching top-\(k\) co-occurrences for every item are often distributed around a single mode for different datasets.

- We show comprehensive empirical results for the ranking ability of the proposed collaborative filtering algorithm, and benchmark the algorithm with various other approaches. The results are demonstrated for four large real life datasets from various recommendation domains.

The paper is organized as follows: We mention related work in Section 2. Then, we present the details of our proposed approach for scalable and adaptive collaborative filtering in Section 3. Experiments and experimental results on real life datasets are presented and discussed in Section 4. Finally, our conclusions and further research possibilities are given in Section 5.

2. Related work

We summarize the key literature related to our work under two different subsections.

2.1. Collaborative filtering

Important references to the vast literature on collaborative filtering can be found in (Ricci et al., 2015; Bobadilla et al., 2013). We focus on a portion related to our work. A seminal memory-based collaborative filtering algorithm is described in (Deshpande and Karypis, 2004), where item-based neighborhood search is used to recommend top-\(N\) items to a user. Utility of various item similarity measures is also discussed in the same paper. An incremental version of such an algorithm to calculate Pearson-correlation-based similarities between users is given in (Papagelis et al., 2005). The algorithm assumes a data stream in the form of \((user, item, rating)\) tuples and stores bookkeeping variables for all possible user-user pairs to perform the necessary updates. Although improved running times are reported, each update takes linear time with the number of users and a full similarity matrix is maintained in the main memory. A related idea (Miranda and Jorge, 2009) is reported for binary ratings, \((user, item)\), with increments to two full matrices in memory holding co-occurrences and similarities, respectively. Yet another incremental algorithm is proposed in (Luo et al., 2013) where it is claimed that some space advantages are achieved by using lightweight similarity functions without loss of prediction accuracy. These incremental algorithms are exact and they attempt to do the required updates in a single pass. However, they do not attempt to maintain a space-efficient summary of useful information which can become problematic when working with large-scale item repositories. Furthermore, they require post-processing on the final similarity matrix to find nearest neighbors. Our work provides improvements regarding both aspects.

Personalized recommenders consider a user’s interaction history before recommendation. Many industrial applications, however, serve non-personalized item-to-item recommendation lists (Hidasi and Tikk, 2013; Koenigstein and Koren, 2013) such as those corresponding to “items frequently bought together” and “people who viewed this also viewed” by considering a single or very few item interactions. This may be due to the lack of personal information or response time constraints. Since our collaborative filtering approach offers up-to-date and
efficient top-$k$ queries, both personalized and item-to-item recommendations become cheaper for large-scale dynamic item repositories.

On the contrary to literature on online personalized recommendation algorithms which learn in an incremental fashion but also allow multiple passes over data, literature on recommendation algorithms using a strict data stream model is very limited (Vinaigre et al., 2015). Several preliminary ideas borrowing from this model have been investigated in (Nasraoui et al., 2007; Chandramouli et al., 2011). Matrix factorization (Koren et al., 2009) based on stream sampling has been investigated in (Diaz-Aviles et al., 2012; Chen et al., 2013). In (Luo et al., 2012; Recht and Re, 2013), authors show methods for parallel batch computation of matrix factorization models for scalable recommendations. Such latent factor models usually require multiple passes over data to converge. Furthermore, despite possible reductions in model computation times via parallelization, they are often more suitable for precomputed recommendations rather than adaptive. In (Vinaigre et al., 2014), it is argued that a single-pass incremental matrix factorization is useful for learning from positive-only feedback streams with some trade-off of recall in top-$N$ recommendations for efficiency. An evaluation protocol for the data stream model is also proposed in the same paper. We discuss this approach further in Section 4.3 and compare it to our proposal. A set of simple stream-based non-personalized recommenders in the news recommendation domain is compared in (Lommatsch and Albayrak, 2015) together with a basic collaborative filtering approach which uses only recent and a limited amount of user interactions that fit in the cache. It relies on these reduced amount of interactions to decrease the time for producing recommendation lists. Some of the most recent interesting approaches are a real-time item-based proprietary recommender system exploiting available frameworks for distributed computing (Huang et al., 2015), and a proof of concept collaborative filtering idea based on a real-time event processing system (Ludmann, 2015).

### 2.2. Frequent items and itemsets in a stream

Discovering frequent and interesting patterns in sequences of actions or events is a core research area in data mining and knowledge discovery (Han et al., 2011; Moens et al., 2013; Lin et al., 2015). Below, we focus on a portion of this research related to our work on data streams.

Maintaining a hot list of items (frequent items over some significant support threshold) in a stream is an interesting data stream mining problem when the number of items is large, the space is limited, and a single pass over data is allowed. Yet another level of complexity is added when itemsets instead of items are of concern.

FREQUENT algorithm (Karp et al., 2003) is devised to operate in the data stream model, and it provides a way for identifying in a set of items, the items with frequencies more than a support threshold, $\phi$. The space requirement of the algorithm is $O(1/\phi)$ and it guarantees to find all true positives, but some false positives may also be included in the resulting set. To get rid of false positives and obtain the exact frequency of true positives, a second pass over the stream is proposed. However, a second pass is not desired in the set of problems we are trying to address. SPACESAVING algorithm (Metwally et al., 2005) also offers guarantees on space complexity. Additionally, it claims to produce better approximations to exact frequencies of the top-$k$ frequent items in a single pass, especially if the number of item occurrences in ranked order follow a power-law relationship. Such a relationship is common in many problem domains. LOSSYCOUNTING algorithm (Manku and Motwani, 2002) is proposed for the same problem with controllable error bounds at the cost of increased space complexity. The algorithm is extended for mining frequent itemsets. Since they all conceptually use counters, these three algorithms are sometimes unified in a family called counter-based algorithms. Extensive comparisons of counter-based algorithms are covered in (Corrado and Hadjieleftheriou, 2008; Liu et al., 2011), which also include sketch-based algorithms, an alternative for finding frequent items by keeping a set of hash tables instead of counters. Yet another alternative is sampling the stream, but this does not always offer tight guarantees on space complexity.

### 3. Method

We first explain the foundations of our collaborative filtering approach in Section 3.1. Then, in Section 3.2, we carry on by extending the ideas for finding frequent items to finding frequent item co-occurrences in a stream. For every distinct item in the stream, a summary of its frequently co-occurring items is maintained. In other words, each summary contains frequent items conditioned on a distinct item in the stream. top-$k$ frequently co-occurring items in an item’s summary can be thought of as an approximation to its nearest neighborhood. The proposed collaborative filtering algorithm, SASCF, replaces a full item similarity matrix with a much more compressed summary structure which holds approximate top-$k$ co-occurrences for each item and in sorted order. In Section 3.3, personalized and item-to-item recommendations are elaborated. Similarity measures based on top-$k$ item co-occurrences are used for recommendation purposes without further necessitating an offline nearest neighbor search algorithm. We provide a comparative analysis of our approach together with computational complexities in Section 3.4.

#### 3.1. Preliminaries

##### 3.1.1. Neighborhood-based collaborative filtering

Neighborhood-based collaborative filtering algorithms (Ricci et al., 2015, Chapter 2) typically measure similarity among users or items in a user-item preference matrix, $R \in \mathbb{R}^{U \times I}$, with respect to a similarity function, where $U$ is the set of users in the system and $I$ is the set of items. In the user-based approach, preferences of most similar users to a user are considered when making recommendations whereas in the item-based approach, most similar items to a user’s item preference history are considered for recommendations. Our collaborative filtering approach is based on a widely used family of offline item-based collaborative
filtering algorithms (Deshpande and Karypis, 2004), which we name as OFFLINECF in this paper. Our primary aim is to achieve a more scalable and adaptive collaborative filtering algorithm while still preserving quality of recommendations in OFFLINECF.

OFFLINECF works as follows: In the model building phase, for each item $i \in I$, its $k$ most similar items are computed with their corresponding similarities. Exact computation of these similarities typically requires building, updating, and storing an item similarity matrix, $S \in \mathbb{R}^{|I| \times |I|}$. A rating prediction step is not necessarily performed, since the primary goal is top-$N$ recommendation. The top-$N$ recommendation problem corresponds to identifying an ordered set of items $I' \subset I$ such that $|I'| \leq N$, $I' \cap I_o = \emptyset$, and $I_o$ is a set of items a specific user $u \in U$ has already interacted with. To solve this problem for a queried user $u$, the set $C$ of candidate items are identified by taking the union of the $k$ most similar items for each $i \in I_u$ and by removing from the union any items that are already in $I_o$. Then, for each item $i, i_\ell \in C$, the similarity to the set $I_o$ is identified as the sum of similarities between all the items $i \in I_o$ and $i_\ell$, using only the $k$ most similar items to $i$. Finally, the items in $C$ are sorted in decreasing order with respect to this sum of similarities and the first $N$ items are selected as the top-$N$ list.

The top-$N$ prediction phase makes the above-mentioned algorithm a personalized one. Instead, if for a new interaction of a user $u$ with an item $i$, we just recommend the $N \leq k$ most similar items to $i$, then this can be considered as a non-personalized item-to-item recommendation. For example, in the case of binary interactions, and using cosine or dot product (Lee et al., 2007) similarity, the system can answer queries like “items frequently bought together” and “people who viewed this also viewed”.

The proposed approach in this paper extends this family of algorithms to a real-time setting by incorporating solutions for more scalability and adaptivity. We note that instead of item similarities, our approach can also be adapted to find and hold user similarities efficiently, but we do not exploit this idea further in this work.

### 3.1.2. Monitoring frequent items and frequent top-$k$ items in a stream

For the completeness of discussion, in Algorithm 1, we illustrate the single pass FREQUENT algorithm for finding frequent items in a stream (Karp et al., 2003; Liberty, 2013). Let $I$ be a set of $|I|$ distinct items and $S$ be a stream of $n$ item appearances. The frequency count $f_i$ of an item $i \in I$ is the number of times it appears in the stream $S$. It is trivial to maintain all exact item frequency counts if we are allowed to use $|I|$ counters. But, Algorithm 1 can assure that $O(l)$ space ($l = \lfloor |I|/\phi \rfloor$ and $l \ll |I|$) is used in the worst case and approximate frequency counts, $g_i$, such that $g_i > \phi n$ are guaranteed to be in $K$. The key operation in the algorithm is deleting one appearance of each item in the counters if the counters are fully occupied. The update operations assure that the stream summary is always stored in at most $l$ counters and $g_i$ is a useful approximation to the true frequency count $f_i$ even if $g_i = 0$. We show the usefulness of this idea for finding frequent item co-occurrences in Section 3.2.

**Algorithm 1: Single pass FREQUENT algorithm**

| Input: $S$: A stream of items, $\phi$: support threshold |
| Output: $K$: $L \subseteq K$, $L$ is the set of frequent items satisfying $\phi$ |
| (1) $K \leftarrow \emptyset$, $count \leftarrow |S|$; |
| (2) foreach item $s$ in $S$ do |
| (3) if $s \in K$ then |
| (4) $count[s] \leftarrow count[s] + 1$; |
| (5) else |
| (6) $K \leftarrow K \cup \{s\}$; |
| (7) $count[s] \leftarrow 1$; |
| (8) if $|K| > \frac{1}{\phi}$ then |
| (9) foreach item $a$ in $K$ do |
| (10) $count[a] \leftarrow count[a] - 1$; |
| (11) if $count[a] == 0$ then |
| (12) $K \leftarrow K \setminus \{a\}$; |

SPACESAVING (Metwally et al., 2005) requires the following modification to Algorithm 1: If the new item in the stream is not already monitored and $|K| = l$, instead of decrementing every counter by 1, it inserts the new item with a value $\min + 1$ into the counter having minimum value, $\min$. This provides a way for not missing frequent items by erring on the positive side, although the count of new stream element can actually be an integer in interval $[1, \min + 1]$. The algorithm guarantees to find all items with $f_i > \min$. However, since $\min \leq \lfloor n/l \rfloor$, items with $f_i > \phi n$ are not always guaranteed when $i$ is chosen to be the reciprocal of $\phi$. Nevertheless, the approach has still some practical implications such as preserving true counts of top-$k$ elements in skewed data distributions, because it always alters the element with the minimum count. In the rest of this paper, we continue to work with both FREQUENT and SPACESAVING since they allow strict bounds on space complexity and useful bounds on approximation errors.

We now show that both FREQUENT and SPACESAVING can be efficiently implemented using the generic data structure illustrated in Fig. 1. The data structure also has the useful property that it holds items always in sorted order with respect to their approximate frequency counts which enables efficient top-$k$ queries. These properties make the data structure convenient for the collaborative filtering approach proposed in Section 3.2. The data structure uses a hash table which, instead of directly holding counts, points to a doubly linked list of values. FREQUENT uses a linked list with a node value showing numeric difference from the left node value, where the leftmost node shows difference from 0. The item lists attached to nodes can be implemented as sets. When a set is empty, the node can be deleted. It can be seen that all update operations are performed at most on a single node and its two neighbors. The key operation, decrementing every counter, is achieved by updating the difference between the value of leftmost node and 0, which assures a few pointer updates. SPACESAVING uses a linked list with a node value directly showing a count. The $\min$ value
is always kept in the leftmost node which assures its constant time retrieval and update. Fig. 1 also illustrates the following example: Assume \( l = 4 \) and the 4 items have counts 2, 3, 2, 3, respectively. In this case, for FREQUENT, the values are \( v_1 = 2 \) and \( v_2 = 1 \). To decrement every counter by 1, \( v_1 \) is reduced to 1. For SPACESAVING, the values are \( v_1 = \min(2) = 2 \) and \( v_2 = 3 \). Two items with the \( \min \) value are contained in the leftmost node. If a new unmonitored item arrives, one of them can be randomly removed and the new item is inserted into the second node which has value \( \min + 1 \).

**Fig. 1:** Generic data structure for FREQUENT and SPACESAVING offering efficient updates and efficient response to top-\( k \) queries

### 3.2. \( k \)-Frequent co-occurrences and online collaborative filtering

Let \( U \) be a set of users and \( I \) be a set of items in a recommender system. Let \( S \) define a stream of tuples \((u, i, t)\), where \( u \in U, i \in I \), and \( t \) is a timestamp. We assume that a tuple represents a positive assessment of item \( i \) by user \( u \), such as a like, a click, a purchase, or a binned rating. The timestamp is used to keep track of temporal order while processing the stream.

Instead of a user-item preference matrix, \( R \in \{0, 1\}^{|U| \times |I|} \), of interactions, and an item similarity matrix, \( S \in \mathbb{R}^{|I| \times |I|} \), the proposed collaborative filtering algorithm maintains two structures: First, exploiting the typical sparsity of \( R \), a list of size \(|U|\) is maintained which holds a separate set of items for each user’s interaction history. We call this structure the user list, \( UL \), where \( UL_u \) denotes the set of items, user \( u \) has already interacted with. Second, a list of size \(|I|\) is maintained which holds an item for every item to hold its \( k \)-frequent co-occurrences with other items. This is a list of stream summary structures implemented using the generic data structure in Fig. 1, where the counts now refer to the approximate number of co-occurrences. We call this second structure the item list, \( IL \), where \( IL_i \) denotes the summary structure of an item \( i \).

It can be shown that due to its characteristics outlined in Section 3.1.2 and by making use of the generic data structure in Fig. 1, FREQUENT-based implementation of \( IL_i \) provides the following guarantees for finding frequent co-occurrences of an item \( i \):

**Lemma 1.** Let \( L = \{j \in IL_i : f_{ji} > \phi n_i \} \) where \( f_{ji} \) is the co-occurrence frequency count of item \( j \) in \( IL_i \) and \( n_i \) is the number of all co-occurrences of item \( i \) or equivalently the size of its co-occurrence stream, \( S_i \). Then, \(|L| < \frac{|I|}{\phi} = l = |IL_i|\).

**Proof.** Otherwise, there would be more than \( 1/\phi \times \phi n_i \) co-occurrences of items from \( L \) in \( S_i \), which is impossible.

**Lemma 2.** Consider an item \( j \notin IL_i \). Then, \( f_{ji} < \phi n_i \).

**Proof.** Each co-occurrence of \( j \) was deleted with \( l - 1 \) other items. Therefore, \( f_{ji}l < n_i \) or \( f_{ji} < \phi n_i \).

**Lemma 3.** The upper bound for approximation error \( f_{ji} - g_{ji} \leq n_i/l \) where \( g_{ji} \) is the approximated co-occurrence of item \( j \) in \( IL_i \).

**Proof.** Assume that \( f_{ji} - g_{ji} \leq d \), where \( d \) is the total number of times deletion condition occurs. Each deletion decrements the count of a distinct item by at most 1 and deletes \( l \) distinct items. Therefore, \( dl \leq n_i \) or \( d \leq n_i/l \).

**Lemma 4.** By using the data structure in Fig. 1, an update operation is constant time and the top-\( k \) retrieval operation is \( O(k) \) since the linked list holds items in sorted order.

**Theorem 1.** FREQUENT-based implementation of \( IL_i \) assures the following: At any time \( O(l) \) space \((l = 1/\phi)\) is used and all frequent co-occurrences are maintained. For all items \( j \in I \), the approximation error of co-occurrence frequency is bounded by \( n_i/l \). The retrieval of top-\( k \) co-occurrences from \( IL_i \), is \( O(k) \).

**Proof.** Follows from Lemmas 1, 2, 3, and 4.

Lemma 4 also holds for SPACESAVING-based implementation of \( IL_i \). As mentioned in Section 3.1.2, when we choose \( l = 1/\phi \), SPACESAVING cannot always guarantee all frequent co-occurrences, though it is still possible to test if all frequent co-occurrences are retrieved by using an auxiliary bookkeeping variable (Metwally et al., 2005). On the other hand, since SPACESAVING always alters the item with the minimum count, it achieves good approximations in practice (Cormode and Hadjieleftheriou, 2008) for top-\( k \) co-occurrences. This situation will also be reflected later in Fig. 2. Therefore, we refer to the proofs provided in (Metwally et al., 2005) for tighter guarantees, but also note that SPACESAVING-based implementation of \( IL_i \) is still of practical importance.

By making use of \( UL \) and \( IL \), the stream is processed as follows: With the arrival of each \((u, i, t)\), stream summary of each item previously assessed by user \( u \) is updated by inserting item \( i \). Furthermore, stream summary of item \( i \) is created or updated by inserting each item previously assessed by user \( u \). These operations take \( O(|UL_u|) \) time, where often \(|UL_u| << |I| \) in practice. Finally, item \( i \) is inserted into user \( u \)'s set of items, \( UL_u \). Constructing and performing the required updates in \( UL \) and \( IL \) correspond to training a scalable and adaptive stream collaborative filtering algorithm, which we name as SASCF: We summarize the procedure in Algorithm 2.

The hash function \( h(.) \) and the parameter \( \alpha/\beta \) in Algorithm 2 constitute an optional scheme to achieve a representative sampling of item co-occurrences. This scheme can be instrumental in controlling the efficiency of the algorithm if the stream is too massive. Assume we wish to process 1/10th of the stream. A naive approach would be to generate a random integer between 0 and 9 for each tuple and store the tuple if the outcome is 0. If we have a very large stream, the law of large numbers will assure a fraction quite close to 1/10th of the frequency of each
item. However, this scheme is not very useful for sampling co-occurrences. Assume all users have interacted exactly with two items each of which constitutes a single co-occurrence. Then, the expectation is that 1/100th of co-occurrences will be sampled. Therefore, we must strive to pick 1/10th of users rather than tuples. To achieve this goal efficiently, we can select a hash function \( h(u, 10) \) which maps users in the system randomly to 10 buckets (Leskovec et al., 2014). We sample all tuples from users mapping to a certain single bucket and ignore all other users. Therefore, the hash function can be thought of as a random number generator. More generally, we can obtain a representative sample consisting of any rational fraction \( \alpha/\beta \) of the users by hashing user IDs to \( \beta \) buckets, \( 0 \) through \( \beta - 1 \), and sampling a tuple \( (u, i, t) \) if the hash value \( h(u, \beta) < \alpha \). Specifically, if \( \alpha = \beta \), there will be no sampling. We note that \( \alpha \) and \( \beta \) are not two different control parameters, but together they set a single control parameter, that is, the sampling ratio, \( \alpha/\beta \), for the stream.

By using \( IL \), instead of constructing a full item similarity matrix, \( S \in \mathbb{R}^{I \times I} \), we maintain \( l \) counters for each item, where the expectation is \( l \ll |I| \). There are important potential benefits of this approach: First, the full similarity matrix can be very large and dense, which complicates the nearest neighbor search process. This way, we can fix and compress the size in one dimension keeping interesting co-occurrences only. Second, many uninteresting co-occurrences are automatically filtered which eases finding top-\( k \) frequent co-occurrences (akin to finding nearest neighbors). When using the generic data structure in Fig. 1, querying for top-\( k \) frequent co-occurrences is cheap because the co-occurring items are already kept in sorted order with respect to their approximate frequency counts. Third, counters are always up-to-date with recent user interactions and no offline training is necessary.

On choosing \( l \) (or \( \phi \)), our key observation is that the true numbers of item co-occurrences in ranked order often follow a power-law relationship (Adamic, 2000) with respect to the rank. Both FREQUENT and SPACESAVING are expected to yield good approximate counts with a low number of counters when true counts demonstrate this behavior. A reflection of this situation is illustrated in Fig. 2 for two different datasets, where we are interested in topmost frequently co-occurring items. We see that the true ranks are captured by both FREQUENT- and SPACESAVING-based \( IL \). The former captures a more transient relationship at the expense of increased approximation errors for toomost frequent co-occurrences. The latter captures better approximations for topmost frequent co-occurrences, but constantly increasing \( \min \) count results in a thicker tail. More implications of these observations will be discussed in Section 4.

3.3. Item recommendations with SASCF

SASCF given in Algorithm 2 can be used for both personalized and item-to-item recommendations. Since top-\( k \) query to \( IL \) is cheap, the most up-to-date \( N \leq k \) items frequently co-associating with item \( i \) can be readily recommended upon arrival of a new tuple \( (u, i, t) \) in a non-personalized fashion. As explained in Sections 2 and 3.1.1, such item-to-item recommendations are common in practice and our method additionally offers adaptivity to recent interactions.

On the other hand, personalized top-\( N \) recommendations detailed in Section 3.1.1 require some more effort to process a queried user’s history, \( UL_u \), before an updated recommendation can be made. We note that for various practical reasons, \( UL_u \) can be restricted, for example, to a few most recent item interactions, or to a few interactions available in a particular session. In the general case, personalized recommendation to a queried user, \( u \), can be performed with the up-to-date \( UL \) and \( IL \). We apply a procedure similar to OFFLINECF, but using top-\( k \) frequent co-occurrences in \( IL \), instead of nearest neighbor search in an offline fashion. For each item \( i \in UL_u \), its top-\( k \) frequently co-occurring items are returned from \( IL_u \). If a returned item is not already in \( UL_u \), then it is assumed to be a candidate item, \( i_c \), for recommendation. Each time a candidate item appears, its similarity, \( similarity(i, i_c) \), to its associated item is recorded. This similarity can be measured in several ways: First, we can directly use the approximate count of co-occurrences between the two items, \( \text{count}[i, i_c] \), as similarity. Here, \( \text{count}[i, i_c] \) denotes count of \( i_c \in IL_u \) and it is assumed to be zero if \( i_c \) is not in \( IL_u \). If we think of the preference matrix, \( R \), this approximates the dot product between two binary vectors holding assessments of all users for the two items,

\[
similarity(i, i_c) = r_i r_{i_c}^\top \approx \text{count}[i, i_c],
\]

where \( r_i \) is a column vector in \( R \). Alternatively, similarity can be based on conditional probability (or confidence) (Deshpande and Karypis, 2004). This similarity is easily obtained because the denominator term can be stored in a simple accumulator during course of the stream, and it corresponds to the number of times an item is seen in the stream,

\[
similarity(i, i_c) = \frac{p(X_1 = i_c, X_2 = i)}{p(X_2 = i)} \approx \frac{\text{count}[i, i_c]}{\text{count}[i]}. \tag{2}
\]
Fig. 2: (Top) True co-occurrences in ranked order for a representative item. This power-law behavior is similar throughout the stream and facilitates choice of $l$. (Middle-Bottom) Approximate frequency counts capture a similar pattern preserving the true ranks of the topmost items.

Yet another similarity measure can be obtained by putting the frequency count of $i_c$ in the denominator to remedy for increased similarity to frequently assessed candidate items. One such measure is cosine similarity,

$$similarity(i, i_c) = \frac{r_i r_i^T}{\|r_i\| \|r_i^c\|} \approx \frac{\text{count}[i, i_c]}{\sqrt{\text{count}[i] \times \text{count}[i_c]}}$$  \hspace{1cm} (3)

However, this last similarity cannot fully be captured by only maintaining an item's frequent co-occurrences. The denominator term should also be considered during counter update phase.

We provide a comparison of SASCF with the offline benchmark algorithm OFFLINECF in Table 1. The comparison includes time and space complexities of the algorithms as well as differences in parameterization.

4. Experiments

We perform comprehensive testing of our collaborative filtering approach on four datasets from domains with durable and fast-moving items. We begin by describing the datasets and analyzing their statistics in the context of our data stream model. Then, we define two different sets of experiments in offline and online fashion and perform them on the proposed algorithm. We discuss quality of our results with varying control parameters and in comparison to other algorithms.

4.1. Datasets

We refer to the four datasets summarized in Table 2 throughout our discussion. ML10M\footnote{http://grouplens.org/datasets/movielens/10m/} (Harper and Konstan, 2015),
Table 1: A comparison of SASCF with OFFLINECF

<table>
<thead>
<tr>
<th></th>
<th>SASCF</th>
<th>OFFLINECF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training scheme</td>
<td>Online incremental updates to U_L and I_L</td>
<td>Offline training for finding item similarities,</td>
</tr>
<tr>
<td></td>
<td>with streaming tuples. For every item, top-</td>
<td>S ∈ ℝ^</td>
</tr>
<tr>
<td></td>
<td>k co-occurrences are always up-to-date.</td>
<td>k, R, S, and a matrix of ⌊</td>
</tr>
<tr>
<td>Space complexity</td>
<td>U_L holding a representation of R in sparse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>format, and I_L of worst case size O(</td>
<td>I</td>
</tr>
<tr>
<td>Time complexity</td>
<td>Online: for each arriving tuple (u, i, t),</td>
<td>Offline: O(</td>
</tr>
<tr>
<td>(model building)</td>
<td>O(U/L_u) updates are done in I_L. Each update</td>
<td>S from R. Then, O(</td>
</tr>
<tr>
<td></td>
<td>is constant time.</td>
<td>for each item.</td>
</tr>
<tr>
<td>Time complexity</td>
<td>Online or offline: O(k ×</td>
<td>U/L_u</td>
</tr>
<tr>
<td>(personalized top-N query)</td>
<td></td>
<td>k NNs, where q =</td>
</tr>
<tr>
<td>Time complexity</td>
<td>Online: Constant time for the most up-to-</td>
<td>Online: Constant time for pre-stored N items</td>
</tr>
<tr>
<td>(item-to-item top-N query)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td>i (or φ), k, N</td>
<td>Exact k NNs</td>
</tr>
<tr>
<td>Similarity measures</td>
<td>dot product, conditional probability, cosine</td>
<td>dot product, conditional probability, cosine</td>
</tr>
</tbody>
</table>

Table 2: Basic dataset properties

| Dataset     | | | # of tuples | Explanation |
|-------------|| |          |             |
| ML10M       | 71,567 | 10,681 | 10,000,054 | MovieLens 10M dataset containing movie ratings |
| MTWT        | 35,894 | 20,419 | 368,490   | Current snapshot of MovieTweeterings dataset containing movie ratings obtained through Twitter stream |
| KOSARAK     | 990,002| 41,270 | 8,019,015 | Anonymized click stream data from an online news portal |
| AMAZON      | 2,146,057 | 1,230,915 | 5,838,041 | Product ratings from Amazon online shopping website |

MTWT² (Dooms et al., 2013), and AMAZON³ contain tuples ⟨u, i, r, t⟩ showing a user, an assessed item, a rating, and a timestamp, respectively. The datasets are binarized such that if a user has rated an item, irrespective of the rating, an association is assumed between the user and that item. In other words, we assume streaming tuples in the form of ⟨u, i, t⟩ unless otherwise stated. KOSARAK⁴ is from the online news domain and represents a common real life scenario where anonymized users produce a sequence of clicks to their items of interest. Therefore, each click session is treated as a separate user and the tuples are already in the form of ⟨u, i, t⟩. This time t represents order of clicks.

### 4.2 Exploratory data analysis

We now provide an exploratory analysis of the datasets in the context of our data stream model.

The first column of Fig. 3 illustrates the number of co-occurrences in ranked order for representative items in different datasets. We observe that the co-occurrences follow a power-law relationship with respect to rank. Since we are interested in top-k co-occurrences of every item, the figure also shows for common choices of k, the top-k-th item. The second column of Fig. 3 shows the log-log plots of the first column and characterizes the power-law behavior of the item co-occurrences (Adamic, 2000). In (Metwally et al., 2005), authors provide a theoretical analysis for the choice of l for SPACE-SAVING regarding two extreme cases: Assuming no specific item distribution and Zipf distribution. Accordingly, we observe that as the power-law behavior of item co-occurrences is characterized by a steeper peak, φ is greatly increased and thus l is greatly reduced. This situation is also reflected in Fig. 3. Although some datasets have a much larger |I|, since their number of item co-occurrences in ranked order demonstrate a more transient figure, a relatively much smaller number of counters is enough to assure all frequent co-occurrences.

The choice of φ among different datasets can be affected by the distribution of co-occurrences, |I|, and the stream sizes. However, we observe in the third column of Fig. 3 that φk, which we define as the maximum value of support to guarantee mining up to the top-k-th item, is often distributed around a single mode for very different items of the same dataset. This observation is important because it allows usage of a fixed φ in the proposed algorithm. For example, when we choose \( φ = \text{MoI}(\text{pmf}(φ_k)) \) and FREQUENT, items on the right hand side of the mode are guaranteed to capture all of their true positive co-occurrences. Furthermore, items on the left hand side of the mode are often not very interesting since their co-occurrences demonstrate a more flat distribution rather than a power-law relationship mainly due to sparsity. In such cases, finding frequent co-occurrences is not meaningful either. This

---

²https://github.com/sidooms/MovieTweeterings
³http://konect.uni-koblenz.de/networks/amazon-ratings
⁴http://fimi.ua.ac.be/data/
Fig. 3: (Column 1) Basic statistics of representative items in different datasets. Tails are trimmed for visualization purposes. Top-20-th and -50-th frequent co-occurrences are explicitly marked with dashed lines. Horizontal axis is rank, vertical axis is the number of co-occurrences. (Column 2) log-log plots of the first column. (Column 3) $\phi_{20}$ distribution of items in an entire dataset. Amazon is sampled by 20%.

Fig. 4 illustrates the distribution of number of co-occurrences in all datasets. If, instead of $IL$, we maintain a full item similarity matrix, then the space requirement is $O(|I|^2)$ and the figure gives us information about the sparsity of this matrix. Alternatively, if we maintain a full list of nonzero co-occurrences for every distinct item, then the figure informs us about its space requirement. We observe that ML10M has a relatively dense similarity matrix. Regarding the third column of Fig. 3 and observation is also justified in Section 4.3, where we observe that choosing $\phi$ around the mode achieves often the best evaluation results and a smaller $\phi$, hence a larger number of counters, does not bring much improvement, if any.

We note that the parameter $\phi$ can be estimated practically when letting the stream flow for a while or if we have assumptions about the particular power-law behavior of the number of item co-occurrences in ranked order. The former is possible by either monitoring a random subset of items with $|I|$ counters or by trial-and-error with online evaluation. Furthermore, $\phi$ can be chosen according to the available system resources in a resource aware fashion as shown experimentally in Section 4.3.
through experiments we see that optimally, its \( IL \) has about 900 counters for every item. For \( \text{MTWT}, \text{KOSARAK}, \) and \( \text{AMAZON} \), the skewed distributions indicate sparsity. On the other hand, optimally, their \( I_L \)s only keep about 300, 500, and 1000 counters, respectively which mean a huge compression of the full item similarity matrix, besides the additional benefit of keeping the topmost similar items always sorted with respect to their co-occurrences.

4.3. Experimental evaluations

We define two different sets of experiments to evaluate the proposed algorithm:

- The first set of experiments uses a holdout set and performs testing in an offline fashion without considering natural time order. This first experimental setup has two goals: One is to reveal the effects of different control parameters on a wide range. The tested control parameters are the number of counters (\( l \) or the corresponding \( \phi \)), the frequent co-occurrence finding method (\textsc{Frequent} or \textsc{Spacesaving}), the similarity measure (dot product, conditional probability, and cosine), and the sampling ratio (\( \alpha/\beta \)). The other goal is to benchmark \textsc{SASCF} with \textsc{OfflineCF} and test whether we obtain similar results using our approximate but online and scalable approach.

- The second set of experiments is done using a first-test-then-train sequential scheme suitable for online testing and with the best parameters from the former set of experiments. Here, the primary goal is to test temporal predictive ability and adaptivity of the proposed approach. One comparative evaluation is carried out with periodically performed \textsc{OfflineCF} over the stream, and yet another one with an incremental matrix factorization algorithm.

In the following two subsections we give the details of these experiments and discuss their results.

4.3.1. Experiments using a holdout test set

For the experiments in this section, we employ the following evaluation scheme: If there is explicit rating information, prior to binarization of a dataset, we randomly remove one tuple with the user’s maximum rating for each user who has rated more than 2 items. We call such users eligible users. The removed tuples form a holdout test set, while the remaining tuples form a training set of tuples \( \langle u, t, i \rangle \), each showing an assessment of a user \( u \) for an item \( i \). In case of \textsc{KOSARAK} data, there is no rating information in the stream. Therefore, we randomly remove one item from the click stream of each eligible user.

After a training phase with \textsc{SASCF}, we use hit rate (\( \text{HR} \)) (Deshpande and Karypis, 2004), akin to \textit{Recall}(\( N \)) (Cremonesi et al., 2010) to evaluate the number of hits from the test set in top-\( N \) recommendations of each eligible user,

\[
\text{HR} = \frac{\text{# of hits in top-}N\text{ recommendations}}{|U'|},
\]

where \( U' \subseteq U \) is the set of eligible users. Furthermore, we measure the mean reciprocal rank (\( \text{MRR} \)) to see the positions of the removed items in the recommendation lists of eligible users,

\[
\text{MRR} = \frac{1}{|U'|} \sum_{j=1}^{U'} \frac{1}{\text{rank}_j},
\]

where \( \text{rank}_j \) is the rank of hit in the top-\( N \) list of an eligible user. We assume zero reciprocal rank, if \( \text{rank}_j > N \). We repeat the test procedures three times and report the average \( \text{HR} \) and \( \text{MRR} \).
In the first two columns of Fig. 5, we observe that OFFLINECF and SPACESAVERS-based SASCF, respectively. Here since both algorithms operate on a holdout set without considering time order. We refer to SASCF-F and SASCF-S as the FREQUENT- and SPACESAVERS-based SASCF, respectively. In the first two columns of Fig. 5, we observe that OFFLINECF results are matched for a wide range of \( \phi \) values (\( l = \lfloor 1/\phi \rfloor \)). Further increasing the \( \phi \) value enables usage of a smaller number of counters, but \( HR \) and \( MRR \) also start to decline. In the third column, we observe that our naive approach to cosine similarity is effective but in a narrower range of \( \phi \) values with a steeper descent in \( HR \) values especially when SASCF-S is used. On the other hand, it should be noted that cosine similarity is not always the best choice. We observe in all figures except those of ML10M that SASCF-F is often better than SASCF-S in terms both \( HR \) and \( MRR \). Although SPACESAVERS offers to maintain better approximations to frequency counts of the top-\( k \) items, FREQUENT has stricter guarantees on frequent co-occurrences with respect to \( \phi \). Moreover, FREQUENT’s steeper descent (as shown in Fig. 2) seems to achieve a useful weighting when calculating similarities. Finally, AMAZON results are provided for proof of resource awareness concept where we vary the number of counters in \( IL \) (hence \( \phi \)) proportional to the available main memory in our testing machine and still observe acceptable evaluation results. This observation is also valid for the other datasets over a wide range of \( \phi \) values.

In Fig. 6, we show change of \( HR \) with respect to varying sampling ratios (\( \alpha/\beta \)) and similarity measures. Here, regarding Fig. 5, the best configurations of control parameter \( l \) (or \( \phi \)) are fixed and SASCF-F is used. \( MRR \) results are not shown for clarity since they are highly correlated with the \( HR \) results as in Fig. 5. These results are quite interesting since the \( HR \) trade-off is often small even when the data streams are heavily sampled.
This brings further scalability to the proposed method.

![Graphs showing Hit Ratio (HR) for different datasets with respect to sampling.](image)

**Fig. 6:** Change of HR with respect to sampling. The results are shown for a broad range of sampling ratios \((\alpha/\beta)\) and different similarity measures. \(k = 20, N = 10\). Number of counters, \(l\), are 900, 250, 500, and 1,000 for ML10M, MTWT, KOSARAK, and AMAZON, respectively.

4.3.2. Sequential evaluation

There are several ways to carry out such an experiment. For example, it is possible to perform it with or without incorporating an explicit forgetting mechanism. In the former, one can do the training in a certain number of past windows while doing the prediction in the current window. If the windows are taken in an overlapping fashion, it is possible to start training before prediction starts in the same window. Furthermore, it is also possible to follow a decaying time window approach (Cormode et al., 2008). All of these scenarios can be useful in different problem domains. We do not employ an explicit forgetting mechanism and use the implicit mechanism available in the proposed method for adapting to the recently arriving user interactions.

We stick to a sequential evaluation scheme similar to (Vina-gre et al., 2014; Gama et al., 2013) and suitable for testing algorithms in the data stream model. The idea is to evaluate the number of hits in sliding time windows to observe temporal behavior of the proposed algorithm. The training is performed incrementally using a landmark window starting from beginning of the stream. The procedure is described in Algorithm 3. This time there are no separate training and test sets. All streaming tuples are used for incremental updates in a first-test-then-train fashion (Bifet et al., 2011). We also evaluate the hits over all streaming tuples whose user has previously interacted with another item. In case the stream has rating information, only tuples with a rating above some threshold \(\rho\) are used for evaluations. For example, \(\rho\) can be the median of each dataset’s rating scale.

**Algorithm 3:** First-test-then-train procedure for sequential evaluation

1. Define a sliding window of size \(W\)
2. \(\text{hits} \leftarrow 0, \text{recommended} \leftarrow 0, w \leftarrow 0\)
3. \(\text{foreach tuple } \langle u, i, t, r \rangle \text{ in } S \text{ do}
   4. \hspace{1em} w \leftarrow w + 1
5. \hspace{1em} \text{if } u \text{ has already assessed an item and } r > \rho \text{ then}
   6. \hspace{2em} \text{Recommend } u \text{ a list of top-}\(N\) items
   7. \hspace{2em} \text{recommended} \leftarrow \text{recommended} + 1
8. \hspace{1em} \text{if } i \text{ is one of the top-}N\text{ items then}
9. \hspace{2em} \text{hits} \leftarrow \text{hits} + 1
10. \hspace{1em} \text{if } w = W \text{ then}
11. \hspace{2em} \text{hit ratio} \leftarrow \frac{\text{hits}}{\text{recommended}} \quad \text{// Report for current time window}
12. \hspace{1em} \text{hits} \leftarrow 0, \text{recommended} \leftarrow 0, w \leftarrow 0
13. \text{Update SASCF model with } \langle u, i, t \rangle \text{ (as per Algorithm 2)}

Sequential evaluation results in comparison to OFFLINECF are illustrated in Fig. 7. For each dataset, we report two results. The first one is obtained by directly using Algorithm 3 with the proposed method. The second result is obtained using an adaptation of OFFLINECF and by slightly diverging from Algorithm 3: Instead of updating the stream summary structures of SASCF, a full item similarity matrix is sequentially updated with the streaming tuples. However, in this case, nearest neighbors of an item are not immediately available for recommendation purposes. Therefore, offline nearest neighbor search is performed on the similarity matrix at regular intervals, and \(k\) nearest neighbors of every item are recorded and used for evaluations in a time window until the next search. In other words, offline periodic batch training is performed. The following settings are used for this experiment: For all streams, \(k = 20\), and \(N = 10\). Conditional probability is used as the similarity measure. For ML10M, \(\phi = 0.001 \text{ (} l = 900\text{)}\) is fixed for SASCF. OFFLINECF is performed every 60,000 tuples. \(W = 10,000\). For MTWT, \(\phi = 0.004 \text{ (} l = 250\text{)}\) is fixed for SASCF. OFFLINECF is performed every 2,000 tuples. \(W = 1,000\). For KOSARAK, \(\phi = 0.002 \text{ (} l = 500\text{)}\) is fixed for the proposed method. OFFLINECF is performed every 20,000 tuples. \(W = 10,000\). For AMAZON, \(\phi = 0.001 \text{ (} l = 1000\text{)}\) is fixed for the proposed method. OFFLINECF is performed every 30,000 tuples. \(W = 10,000\).

In Fig. 7, the experimental results suggest that the proposed algorithm is at least as good as OFFLINECF in terms of hit ratio and often significantly better. This result is expected since the algorithm is more adaptive to recent interactions. OFFLINECF results show various degrees of degradation with MTWT being the strongest. We note that these results are sensitive to the length of training intervals. For example, the degradations tend to increase even more when the batch training is done less regularly. Although characteristics of time series vary from dataset to dataset, the results suggest usefulness of the proposed approach where the latest information is readily used for the recommendation task in comparison to offline periodic batch training. The results are also interesting due to low space require-
ments and usage of incremental training instead of retraining with the entire past data in each batch training episode.

The statistical significance of these results is tested using a Wilcoxon signed rank test (Lowry, 2015) as follows: At each sliding window, the error of an algorithm is defined as the miss ratio \(1 - HR\). The null hypothesis is that paired miss ratios from the two algorithms come from the same distribution. For a significance level of 1\%, we fail to reject the null hypothesis for ML10M, but we reject it for NTWT, KOSARAK, and AMAZON where rejection means there is statistical significance between the errors of two algorithms. The results are in accordance with the visualizations in Fig. 7. We also observe that the statistical significance of results between the two algorithms is even more emphasized when we decrease the batch training frequency of OFFLINECF.

We finally compare our approach to a recent online matrix factorization approach called Incremental stochastic gradient descent (ISGD) (Vinagre et al., 2014). Similar to our work, ISGD works on binary ratings and the top-\(N\) recommendation problem. In the training phase, single pass SGD updates are performed by each streaming tuple with a complexity proportional to the number of latent factors, \(\kappa\). In the top-\(N\) item pre-
prediction phase, each recommendation list requires $O(\kappa \times |I|)$ operations for predicting item scores and $O(|I| \times \log(|I|))$ operations for sorting items with respect to their score. The space complexity is the same as that of typical matrix factorization approaches (Koren et al., 2009) with two component matrices of size $O(\kappa \times |I|)$ and $O(\kappa \times |U|)$. Authors report comparative hit ratios of their incremental method to several well-known algorithms. In Fig. 8, we illustrate the results of our experiments. Again, we carry out experiments using Algorithm 3, except this time line 13 refers to ISGD updates. We implement the original ISGD algorithm exactly and refer to learning rate, $\eta$, regularization parameter, $\lambda$, besides $\kappa$. We also note that stream sampling is not applied in case of ISGD since it affects its convergence and quality of results. We first observe that generally ISGD takes some time at the beginning of the stream to converge before producing more comparable hit ratios. For ML10M, we observe that ISGD hit ratios have an improving trend with $\eta = 0.005$, $\lambda = 0.05$, and $\kappa = 10$. For MTWT, $\eta = 0.04$, $\lambda = 0.01$, and $\kappa = 10$ produce competitive hit ratios after initial convergence, but then the results show a degradation trend beginning towards the middle of the stream as newer users and items are introduced. For KOSARAK $\eta = 0.008$, $\lambda = 0.002$, and $\kappa = 10$ produce comparatively low hit ratios on session-based user data. The parameters are found by cross-validation in a limited search space but in general, such parameters for SGD-based approaches are not trivial to fine-tune and a slight change in one of the parameters may affect convergence. For example, in case of AMAZON, we could not find the convergence parameters for ISGD with a limited grid search (SASCF results are shown in Fig. 7). In all tests, we observe that ISGD and SASCF results are significantly different both visually and statistically. Overall, we note that the updates of ISGD are quite fast when single pass updates and a low number of latent factors are used. As in other SGD-based approaches, its space complexity is attractive when a low number of latent factors are sufficient. On the other hand, the effect of growing number of items on the prediction step of ISGD needs further investigation since it both affects the comparative hit ratios and the computational cost of prediction which may necessitate offline precomputation of recommendation lists although the learning stage of the algorithm is online. Since convergence of parameters takes time, adaptivity of ISGD also needs further investigation for streams with dynamic users and items, and in the cases where users or items have a few interactions. In these sit-
ations, neighborhood-based methods like SASCF can be more suitable.

5. Conclusions

We propose a novel real-time collaborative filtering algorithm, SASCF, suitable for personalized and item-to-item recommendations. SASCF makes use of the data stream model by extending two frequent item finding algorithms. The core idea is continuously maintaining a compact summary of frequently co-occurring items instead of an offline nearest neighbor search on the item similarity matrix. Moreover, the frequent co-occurrences are maintained in sorted order with respect to their approximate frequency counts which enables efficient top-k queries. The usefulness of these ideas is due to the observation that the number of item co-occurrences in ranked order demonstrate a power-law behavior.

The proposed algorithm with its stream summary structures offers scalability and adaptivity. The former is because the structures can be very space-efficient and easy to query scaling to large collections of users, items, and their interactions. The latter is because the structures are always up-to-date with the latest data from the stream adapting to the dynamic nature of user and items. The algorithm is also resource aware in the sense that the available system resources can still be used for intelligent recommendations.

Finding nearest neighbors in the data stream model with respect to an arbitrary similarity measure is an open problem. If the item similarity matrix can be computed efficiently and fit in memory, and adaptivity is not a major concern, then offline nearest-neighbor search can be used at the cost of sorting similarity vectors, but with the luxury of using many different similarity measures without approximation. Otherwise, the proposed algorithm for the data stream model can be very useful for approximating the results with several useful similarity measures.

We perform comprehensive experiments on four real-life and non-trivial datasets. First, experimental results in an offline setup are reported. Given the selection of control parameters in a broad range, the results comply with offline item-based collaborative filtering in terms of prediction quality. Second, we perform experiments in an online fashion using a first-test-then-train sequential scheme. The sequential evaluation results indicate adaptivity of the proposed approach in comparison to offline item-based collaborative filtering. In all comparative experiments, the proposed method has hit ratios at least as good with a statistical significance in miss ratios for three out of four datasets. We observe that further extending the batch training intervals of the offline method increases the miss ratios more in comparison to SASCF. We also show sequential evaluation results in comparison to a recent incremental matrix factorization approach and compare hit ratios and general properties of the algorithms.

We think that our approach is useful for many collaborative filtering applications with desired properties like scalability, adaptivity, and resource awareness. We note some possible future research directions as follows: Other strategies for finding frequent items in streams (Cormode and Hadjieletheriou, 2008; Liu et al., 2011; Thanh Lam and Calders, 2010; Woodruff, 2016) can be plugged in the SASCF framework. Apart from the inherent strategies for adaptivity, different strategies (Gama et al., 2014) like time weighting (Cormode et al., 2008) item co-occurrences can also be considered. A parallel version of the algorithm improving the benefits of the data stream model is another improvement to be considered. Finally, other incremental algorithms can be compared to SASCF regarding different performance criteria of collaborative filtering systems.

Acknowledgements

We would like thank the anonymous reviewers for their detailed comments to further improve this work.

References
