Weighted Similarity Estimation in Data Streams

Konstantin Kutzkov
NEC Laboratories Europe
Heidelberg, Germany
konstantin.kutzkov@neclab.eu

Mohamed Ahmed
NEC Laboratories Europe
Heidelberg, Germany
mohamed.ahmed@neclab.eu

Sofia Nikitaki
NEC Laboratories Europe
Heidelberg, Germany
sofia.nikitaki@neclab.eu

ABSTRACT

Similarity computation between pairs of objects is often a bottleneck in many applications that have to deal with massive volumes of data. Motivated by applications such as collaborative filtering in large-scale recommender systems, and influence probabilities learning in social networks, we present new randomized algorithms for the estimation of weighted similarity in data streams.

Previous works have addressed the problem of learning binary similarity measures in a streaming setting. To the best of our knowledge, the algorithms proposed here are the first that specifically address the estimation of weighted similarity in data streams. The algorithms need only one pass over the data, making them ideally suited to handling massive data streams in real time.

We obtain precise theoretical bounds on the approximation error and complexity of the algorithms. The results of evaluating our algorithms on two real-life datasets validate the theoretical findings and demonstrate the applicability of the proposed algorithms.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Theory; Algorithms; Experiments

Keywords

Recommender systems; Viral marketing; Collaborative filtering; Sketching; Streaming algorithms

1. INTRODUCTION

Similarity computation is a basic primitive in many data mining algorithms, ranging from association rule mining to clustering. However, for many Big data applications, similarity computation can be prohibitively expensive, hence necessitating the need for scalable methods for similarity estimation in massive datasets. As such the streaming model of computation has become extremely popular over the last decade, because it can handle massive data through sequentially processing the input. Further, streaming algorithms that need only one pass over the input open the door to real-time stream processing which considerably extends the applications domain. For example, Locality-sensitive hashing [17] techniques have found application in numerous research areas, see [11, 28, 30] for concrete examples.

Somewhat surprisingly however, similarity estimation in streams has received less attention for cases when the underlying similarity measures are weighted. Motivated by applications in recommender systems and viral marketing, in this work we consider how to estimate the *weighted* similarity between real vectors revealed in a streaming setting.

Related work.

Modern recommendation systems work with huge volumes of data which necessitates the scalable processing of the input. As such, traditional algorithms that work in an offline fashion are often not applicable since we cannot afford to load the full input in memory. Similarly, in online social networks, users activity results in huge amounts of data that needs to be processed efficiently. In the following we review previous work on stream processing for applications in recommender systems and viral marketing.

Collaborative filtering is widely applied in recommender systems. The basic idea is to recommend an item i to a user u if i has been highly ranked by users who have similar preferences to u. The similarity between users is defined based on their rating history, and is represented as a (sparse) vector whose dimensionality is the total number of items.

Several different similarity measures have been proposed in the literature, e.g. Jaccard and Cosine similarity, Euclidean distance and Pearson correlation. In a series of papers [3, 4, 5, 6], Bachrach et al. proposed sketching algorithms to estimate the Jaccard similarity between users using min-wise independent hashing [7]. Since Jaccard similarity only applies to unweighted sets, this is a simplication of the problem, whereby we are only interested which items users have rated, not how they have rated them. It is argued that rating itself is an indication of interest in a given product. However since user ratings indicate the level of interest,

CIKM'15, October 19–23, 2015, Melbourne, Australia.

DOI: http://dx.doi.org/10.1145/2806416.2806514.

i.e. two users may provide conflicting ratings on the same item, in practice, weighted similarity is commonly used [23]. The state-of-the-art result is [5], which details how to combine the compact *bit-sketches* from [24] with the method proposed in [16] that achieve an exponential speed-up of the evaluation time of a "random enough" hash function. Finally, in [4], an extension of Jaccard similarity to ranking estimation is proposed. (Note however that the extension does not achieve the improved time and space complexity of [5].)

Influence propogation is widely studied in the context of viral marketing in social networks. Based on the observation that influence propagates through the network, the goal is to detect the influential users that are most likely to determine the overall behavior of the network. Such users may then be targeted during advertisement campaigns or used to predict the success of a campaign before launch. In a seminal work Kempe et al. [21] introduced the *independent cascade* model and presented approximation algorithms for influence maximization in social networks. Under this model, a user u influences a neighbor of hers v with a certain probability p_{uv}, however, the authors assume that propagation probabilities p_{uv} are known in advance.

Goyal et al. [18] present methods for inferring influence probabilities. Informally, they define different measures of influence probability, whereby a user u is said to have influenced a user v if an action performed by u is later performed by v within τ time units, for a suitably defined τ. Note that this approach does not distinguish between how actions are performed. Building upon this work, Kutzkov et al. [22] have presented streaming algorithms capable of estimating the influence probability between users using only a small amount of memory per user. In [22], the influence probability is defined as an extension of Jaccard similarity with time constraints, and the algorithm extends min-wise independent hashing to handle time constraints.

Finally, another thread of works has analyzed user behavior in online social networks, in terms of social and correlational influence [2, 14, 19, 20]. In particular, Jamali et al. [19, 20] have studied the problem of influence propagation in Social Rating Networks (SRN). The correlational influence between users is computed as a Pearson correlation between the rating vectors.

Our contribution.

- Similarity estimation in data streams for collaborative filtering. A simple extension of AMS sketching for inner product estimation yields new sketching algorithms for cosine similarity and Pearson correlation between real vectors revealed in a streaming fashion.
- We extend cosine similarity and Pearson correlation to also consider time constraints in order to model influence propagation in viral marketing and predict users behavior. Building upon the STRIP algorithm from [22], we design new algorithms for the estimation of the considered similarity measures. The algorithm is particularly suitable for input vectors with entries from a small discrete domain, a common situation in rating networks.

We obtain precise theoretical bounds on the complexity of the presented algorithms. An experimental evaluation on real datasets confirms the theoretical findings.

Organization of the paper.

In Section 2 we present necessary notation and formal definitions. In Section 3 we present and analyze a weighted similarity estimation algorithm for collaborative filtering in data streams. We extend the considered similarity measures to model influence propagation and present streaming algorithms for their estimation in Section 4. Results from experimental evaluation on real data are reported in Section 5. The paper is concluded in Section 6.

2. PRELIMINARIES

Notation.

The k-norm of a vector $x \in \mathbb{R}^n$ is defined as $\|x\|_k = (\sum_{i=1}^n |x_i|^k)^{1/k}$. The 2-norm of x will be denoted as $\|x\|$, and the 1-norm as $|x|$. Let U be a set of m users and I a set of n items. A rating given by user $u \in U$ on an item $i \in I$ is denoted as r_{ui}. A user is described by an n-dimensional real vector. For the i-th entry in u it holds $u_i = r_{ui}$, thus we will also use u_i instead of r_{ui} to denote u's rating on i. The set of items rated by user u is denoted by $I_u \subseteq I$. We will denote by $[n]$ the set $\{0, 1, \ldots, n-1\}$.

Similarity measures.

We consider following measures for the similarity between two vectors.

1. Jaccard similarity computes the probability that an item $i \in I_u \cup I_v$ is also contained in $I_u \cap I_v$:
$$jaccard(I_u, I_v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|}.$$
2. Cosine similarity.
$$cos(u, v) = \frac{\sum_{i \in I_u \cap I_v} u_i v_i}{\|u\|\|v\|}$$
where $\|u\| = (\sum_{i \in I_u} u_i^2)^{\frac{1}{2}}$.
3. Pearson correlation.
$$\rho(u, v) = \frac{\sum_{i \in I_u \cap I_v} (u_i - \tilde{u})(v_i - \tilde{v})}{\|\hat{u}\|\|\hat{v}\|}$$
where $\tilde{u} = \frac{1}{|I_u|}\sum_{i \in I_u} u_i$, $\|\hat{u}\| = (\sum_{i \in I_u} (u_i - \tilde{u})^2)^{\frac{1}{2}}$.

Hashing, Probability and Approximation guarantees.

We assume familiarity with basic probability theory notation. In the analysis of the algorithm we use Chebyshev inequality defined as follows. Let X be a random variable with expectation $\mathbf{E}[X]$ and variance $\mathbf{V}[X]$. Then
$$\Pr[|X - \mathbf{E}[X]| \geq \lambda] \leq \frac{\mathbf{V}[X]}{\lambda^2}.$$

A family $\mathcal{F}$ of functions from V to a finite set S is *k-wise independent* if for a function $f : V \to S$ chosen uniformly at random from $\mathcal{F}$ it holds
$$\Pr[f(v_1) = c_1 \wedge f(v_2) = c_2 \wedge \cdots \wedge f(v_k) = c_k] = \frac{1}{s^k}$$
for $s = |S|$, distinct $v_i \in V$ and any $c_i \in S$ and $k \in \mathbb{N}$.

A family $\mathcal{H}$ of functions from V to a finite totally ordered set S is called *(ε, k)-min-wise independent* if for any $X \subseteq V$

and $Y \subseteq X$, $|Y| = k$ for a function h chosen uniformly at random from $\mathcal{H}$ it holds

$$\Pr[\max_{y \in Y} h(y) < \min_{z \in X \setminus Y} h(z)] = (1 \pm \varepsilon) \frac{1}{\binom{|X|}{k}}$$

we will refer to a function chosen uniformly at random from a k-wise independent family as a k-wise independent function.

Finally, we will say that an algorithm computes an ε-approximation of a given quantity q if it returns a value $\tilde{q}$ such that $q - \varepsilon \leq \tilde{q} \leq q + \varepsilon$. In the theoretical analysis of the algorithms, we will show results stating that certain approximation guarantee is obtained with probability 2/3. Using the Chernoff inequality this probability can be amplified to $1 - \delta$ for any $\delta \in (0, 1)$ by running $O(\log \frac{1}{\delta})$ independent copies of the algorithm in parallel, c.f. [26]. Note that 2/3 can be replaced by any constant $c > 1/2$.

Problem statement.
Let $\mathcal{S}$ be a stream of triples (u, r_i, t_u) where u is the user identifier, r_i is the rating the user gives to item i, and t_u is the timestamp of the rating. We assume that each user rates an item at most once. Throughout the paper we will often represent the rating given by u on i as u_i instead of (u, r_{ui}).
We will consider the following two problems:

1. Collaborative filtering: Given users u and v, what is the similarity between u's and v's ratings for a given similarity measure? (Note that we disregard the timestamps of the ratings.)

2. Influence propagation: Given users $u, v \in U$, compute $sim_\Omega(u, v)$, i.e., the similarity between the ratings of u and v under a given similarity measure that satisfies given time constraint Ω. In particular, when a social graph $G = (V, E)$ is given, what is the probability that u influences a friend of hers v, i.e., $sim_\Omega(u, v)$ for $(u, v) \in E$. (Note that we present concrete examples for sim_Ω in Section 3.)

3. WEIGHTED SIMILARITY ESTIMATION IN DATA STREAMS

Before formally describing the proposed algorithms, we first provide an overview of our reasoning. We assume that we can store a sketch for each user's activity. Sketches are compact representations of a user's rating history. We observe that if we can estimate the inner product of uv, then we can also estimate $cos(u, v)$. This is achieved by computing $\|u\|$ and $\|v\|$ in a streaming setting by simply maintaining a counter to record the current value of the squared 2-norm of u and v. Utilising AMS sketches for the estimation of the inner product, as proposed in [12], we obtain an ε-approximation of $cos(u, v)$.

The Pearson correlation can be rewritten as $cos(u - \tilde{u}, v - \tilde{v})$, where $u - \tilde{u}$ is the vector obtained from u by replacing u_i with $u_i - \tilde{u}$, $\tilde{u}$ being the mean of u. Therefore, if two passes over the data are allowed, we can compute in the first pass the values $\tilde{u}$ and in a second pass we can run the cosine similarity estimation algorithm on the updated vectors $u - \tilde{u}$. We will show that the *linearity* of AMS sketches enables us to compute a sketch $u - \tilde{u}$ in a single pass without knowing $\tilde{u}$ in advance.

3.1 Algorithm

Before presenting our algorithm, we briefly present how AMS sketching works:

AMS sketching
AMS sketches were originally designed as an efficient technique for estimating the second moment of a data stream's frequency vectors [1]. For an (unweighted) stream $\mathcal{S} = i_1, i_2, i_3, \ldots$ over a set of items I, let f_i denote the frequency of item i, i.e., the number of occurrences of i in $\mathcal{S}$. Then the second moment of the stream $\mathcal{S}$ is defined as $F_2 = \sum_{i \in I} f_i^2$.

AMS sketching works as follows: Let $s : I \to \{-1, 1\}$ be a 4-wise independent function. We maintain a variable X, and for each incoming item i, we update X as $X \mathrel{+}= s(i)$. After processing the stream it holds that $X = \sum_{i \in I} s(i) f_i$, and we can show that $\mathbf{E}[X^2] = F_2$ since $\mathbf{E}[s(i)s(j)] = 0$ for $i \neq j$ and $s(i)^2 = 1$. Using that s is 4-wise independent, the variance of the estimator can be bounded by $\mathbf{V}[X^2] \leq \mathbf{E}[X^4] = F_2^2$. Since for a constant c, it holds that $\mathbf{V}(X/c) = \mathbf{V}(X)/c^2$, the average of $O(1/\varepsilon^2)$ random variables X, is an unbiased estimator of F_2 and has variance $\varepsilon^2 F_2^2$. A standard application of Chebyshev inequality yields that we obtain an ε-approximation from the average of $O(1/\varepsilon^2)$ AMS sketches. Finally, it was first observed in [12] that AMS sketching can be also used to estimate the inner product uv of vectors $u, v \in \mathbb{R}^m$. To do so, we maintain random variables $X = \sum_{i=1}^n s(i)u_i$ and $Y = \sum_{i=1}^n s(i)v_i$, then $\mathbf{E}[XY] = uv$ and, as we see next, the variance is bounded by $\mathbf{V}[XY] \leq (\|u\|\|v\|)^2$. From this, an estimate of $cos(u, v)$ easily follows.

A drawback of AMS sketching is that for each new item in the stream, we need to update all AMS sketches. The Count-Sketch algorithm helps to overcome this by improving the update time [10]. Instead of working with many different AMS sketches, Count-Sketch works with a single hash table. For a new entry u_i, one updates the hash table as $H_u[j] \mathrel{+}= s(i)u_i$ where $j = h(i)$ for a suitably defined hash function $h : [n] \to [k]$. After processing the vectors u, v, their inner product can be estimated as $\sum_{j \in [k]} H_u[j]H_v[j]$. Intuitively, the larger hash table we use, the better estimates we obtain because we have less collisions between items.

Pseudocode
A weighted similarity estimation algorithm is presented in Figure 1. We assume that the input is a stream $\mathcal{S}$ of pairs (u, r_i) denoting that user u rated item i with a rating r_i. For each user, we keep a hash table H_u and we run the Count-Sketch algorithm. We process an incoming pair (u, r_i) by updating H_u with r_i. In addition to the sketches H_u, we keep the following arrays: C – counting the number of items rated by each user u, N – for computing the squared 2-norm of u and S – for computing $\sum_{i \in I_u} u_i$. We also extend the sketch H_u as follows: in the j-th cell we store an array $Sign_u$ such that $Sign_u[j] = \sum_{i \in I_u : h(i) = j} s(i)$, we will need $Sign_u$ for the estimation of Pearson correlation.

After processing the stream, an inner product uv is estimated from the sketches H_u and H_v. The estimation of cosine similarity follows directly from the respective definition. As we show in the theoretical analysis we compute Pearson correlation $\rho(u, v)$ as if we computed $cos(u - \tilde{u}, v - \tilde{v})$. We show in the proof of Theorem 2 that after processing the stream, using $C[u], S[u], N[u], Sign_u$ we can update the sketches H_u such that we obtain an estimate of the Pearson correlation $\rho(u, v)$.

ProcessRatings

Input: stream of user-rating pairs $\mathcal{S}$, pairwise independent function $h : \mathbb{R} \to [k]$, 4-wise independent $s : \mathbb{N} \to \{0, 1\}$

1: **for** each $(u_i) \in \mathcal{S}$ **do**
2: Update(u_i, h, s)

Update

Input: rating u_i, k-wise independent function $h : \mathbb{N} \to [k]$, 4-wise independent $s : \mathbb{N} \to \{0, 1\}$

1: $H_u[h(i)]$ += $s(i)u_i$
2: $N[u]$ += ${u_i}^2$
3: $S[u]$ += u_i
4: $C[u]$ += 1
5: $Sign_u[h(i)]$ += $s(i)$

EstimateCosine

Input: users u, v, sketches H, array N

1: $inner\_product = 0$
2: **for** $i = 1$ to k **do**
3: $inner\_product$+ = $H_u[i]H_v[i]$
4: **return** $inner\_product/\sqrt{N[u]N[v]}$

EstimatePearson

Input: users u, v, sketches H, arrays N, S, C, $Sign$

1: $mean_u = S[u]/C[u]$
2: $mean_v = S[v]/C[v]$
3: $inner\_product = 0$
4: **for** $i = 1$ to k **do**
5: $H_u[i]$ −= $Sign_u[i]mean_u$
6: $H_v[i]$ −= $Sign_v[i]mean_v$
7: $inner\_product$+ = $H_u[i]H_v[i]$
8: $norm_u = N[u] - 2mean_u S[u] + C[u]mean_u^2$
9: $norm_v = N[v] - 2mean_v S[v] + C[v]mean_v^2$
10: **return** $inner\_product/\sqrt{norm_u norm_v}$

Figure 1: Weighted similarity estimation through sketching.

3.2 Theoretical analysis

Cosine similarity.

Theorem 1. *Let $u, v \in \mathbb{R}^n$ be revealed in a streaming fashion. There exists a one-pass algorithm that returns an ε-approximation of $cos(u, v)$ with probability 2/3 and needs $O(\frac{1}{\varepsilon^2})$ space. Each vector entry u_i and v_i can be processed in $O(1)$ time. The estimation can be computed in $O(\frac{1}{\varepsilon^2})$ time.*

Proof. Assume for u and v we keep hash tables H_u and H_v recording k values, for k to be specified later. We also keep a counter N that will record the squared 2-norm of each vector u. Let $h : \mathbb{R} \to [k]$ and $s : \mathbb{N} \to \{-1, 1\}$. For a new arrival u_i we then update $H_u[h(i)]$ += $s(i)u_i$. Simultaneously, we update $N[u]$ += u_i^2. After processing the stream, we return $\frac{\sum_{i=1}^k H_u[i]H_v[i]}{\sqrt{N[u]N[v]}}$ as an estimate of $cos(u, v)$.

Clearly, after processing the stream $\|u\| = \sqrt{N[u]}$ and $\|v\| = \sqrt{N[v]}$. Therefore, an $\varepsilon\|u\|\|v\|$-approximation of uv will yield an ε-approximation of $cos(u, v)$. Let I_{ij} be an indicator variable for the event that $h(i) = h(j)$. For a pairwise independent h it holds $\Pr[I_{ij} = 1] = 1$ if $i = j$ and $\Pr[I_{ij} = 1] = 1/k$ for $i \neq j$. Note that $\mathbf{E}(I_{ij}) = \mathbf{E}(I_{ij}^2)$. Let $Z = \sum_{i=1}^k H_u[i]H_v[i] = \sum_{i,j\in[n]} s(i)s(j)u_i v_j I_{ij}$. By linearity of expectation it holds that $\mathbf{E}(Z) = uv$. For a 4-wise independent s the variance of the estimator can be upper bounded as follows:

$$\mathbf{V}(Z) = \mathbf{E}((\sum_{i,j\in[n]} s(i)s(j)u_i v_j I_{ij})^2) - \mathbf{E}(Z)^2$$

$$= \sum_{i,j,k,l\in[n]} \mathbf{E}(s(i)s(j)s(k)s(l)I_{ij}I_{kl})u_i u_k v_j v_l - \sum_{i,j\in[n]} u_i v_i u_j v_j$$

$$= \sum_{i,j\in[n]} E(I_{ii}I_{jj}u_i v_i u_j v_j) + \sum_{i,j\in[n]} E(I_{ij}u_i^2 v_j^2) - \sum_{i,j\in[n]} u_i v_i u_j v_j$$

$$= \sum_{i,j\in[n]} u_i^2 v_j^2/k = (\|u\|\|v\|)^2/k.$$

The above follows from linearity of expectation and from observing that for 4-wise independent s, $\mathbf{E}(s(i)s(j)s(k)s(l)) = 1$ if and only if there are two pairs of identical indices. By Chebyshev's inequality we have

$$\Pr[|Z - \mathbf{E}(Z)| \geq \varepsilon\|u\|\|v\|] \leq \frac{\mathbf{V}(Z)}{\varepsilon^2(\|u\|\|v\|)^2} \leq \frac{1}{\varepsilon^2 k}.$$

Therefore, for $k = O(1/\varepsilon^2)$ we can show that Z is an ε-approximation of uv with probability 2/3. □

Pearson correlation.

For Pearson correlation it holds $\rho(u, v) = cos(u - \tilde{u}, v - \tilde{v})$. Therefore, if we are able to perform two passes over the data, in the first pass we can compute $\tilde{u}, \tilde{v}$, and in a second pass update each component as $u_i - \tilde{u}$ and estimate the cosine similarity between the vectors. Next, we adjust AMS sketching to estimate $cos(u - \tilde{u}, v - \tilde{v})$ in a single pass, without knowing $\tilde{u}, \tilde{v}$ in advance.

Theorem 2. *Let $u, v \in \mathbb{R}^m$ be revealed in a streaming fashion. There exists a one-pass algorithm that returns an ε-approximation of $\rho(u, v)$ with probability 2/3. The algorithm needs $O(\frac{1}{\varepsilon^2})$ space. Each vector entry u_i, v_i can be processed in $O(1)$ time.*

Proof. Assume we know $\tilde{u}$ in advance. Let H_u be the sketch for $u - \tilde{u}$. Then, $H_u[k] = \sum_{i\in I_u : h(i)=k} s(i)(u_i - \tilde{u}) = \sum_{i\in I_u : h(i)=k} s(i)u_i - \sum_{i:h(i)=k} s(i)\tilde{u}$. After processing the stream it holds that $Sign_u[j] = \sum_{i\in I_u : h(i)=j} s(i)$, $S[u] = \sum_{i\in I_u} u_i$ and $C[u] = |I_u|$. Let $mean_u = S[u]/C[u]$. We thus update $H_u[j]$ −= $Sign_u[j]mean_u$. The squared norm $\|u - \tilde{u}\|^2$ is computed as $\|u\|^2 - 2S[u]mean_u + C[u]mean_u^2$. Thus, in each H_u we have exactly the same values if we had sketched the vector $u - \tilde{u}$. The complexity and approximation guarantee of the algorithm follow directly from the proof of Theorem 1 and the pseudocode. □

3.3 Comparison to previous work

In a series of papers Bachrach et al. presented sketching algorithms for Jaccard similarity estimation [3, 4, 5, 6]. The state-of-the-art result is presented in [5]. The authors present a min-wise independent hashing algorithm that combines b-bit min-wise hashing [24], and yields optimal space complexity, with the min-wise independent sampling approach [16] that achieves the best known processing time per update.

Inner product estimation using AMS sketching was presented in [12] but, to the best of our knowledge, extensions to similarity estimation have not been considered in the literature. This appears to be surprising since LSH based approaches might not be suitable when applied to high-speed streams. Consider for example cosine similarity. The LSH algorithm proposed in [9] needs space and update time $O(\frac{1}{\Theta(u,v)\varepsilon^2})$ for a relative $(1\pm\varepsilon)$-approximation of the angle $\Theta(u,v)$, (the algorithm thus estimates $arccos(u,v)$)). From Theorem 1, we need space $O(\frac{1}{cos^2(u,v)\varepsilon^2})$, but the processing time per update is constant.

4. WEIGHTED SIMILARITY ESTIMATION WITH TIME CONSTRAINTS

We consider the problem of similarity estimation in data streams where an additional time constraint is introduced. In particular, we consider

$$\Omega(u_i, v_i) = \begin{cases} 1 & \text{if } 0 \leq t(v_i) - t(u_i) \leq \tau \\ 0 & \text{otherwise,} \end{cases}$$

i.e., $\Omega(u_i, v_i)$ is the binary constraint that evaluates whether user v has rated items i within τ time units after user u has rated item i. When clear from the context, we will write Ω instead of $\Omega(u_i, v_i)$. This allows to model propagation and influence as discussed in the introduction. Note that the approach presented in the previous section does not seem to apply here. AMS sketching is a linear transformation of the stream, i.e., the order in which we sketch the items does not affect the final result. However, when sketching the stream of ratings we have to also record information about the time when items were rated. We will thus present a different solution building upon the STRIP algorithm [22] that estimates Jaccard similarity with time constraints. In the following we extend the considered similarity measures to also include time constraints, briefly present the STRIP approach and discuss how to generalize it to weighted similarity measures estimation.

Similarity measures with time constraint.

We extend the considered similarity measures as follows:

- **Cosine similarity.**

$$cos_\Omega(u,v) = \frac{\sum_{i \in I_u \cap I_v} u_i v_{i\Omega(u_i,v_i)}}{\|u\|\|v\|}$$

- **Pearson correlation.**

$$\rho_\Omega(u,v) = \frac{\sum_{i \in I_u \cap I_v} (u_i - \tilde{u})(v_i - \tilde{v})_{\Omega(u_i,v_i)}}{\|\hat{u}\|\|\hat{v}\|}$$

The STRIP algorithm.

Assume we are given a social graph and a stream of user-action pairs (u, a_i) denoting that user u performed action a_i. Actions can denote any activity like liking a photo, sharing a video or rating an item. For a possible set of n actions, user's activity is represented by a (sparse) binary vector where the i-th entry denotes whether a user has performed the action a_i. In our context an action corresponds to rating an item, whereby we are not interested in the exact rating but only in the fact that the item was rated. We want to detect users that appear to have high influence on their neighbors. Since we are only interested *which* items a user has rated, we can assume that the items rated by user u correspond to an n-dimensional binary vector r^u such that $r^u_i = 1$ iff user u has rated item i. Following the definition from [18], the influence probability is defined as

$$p_{uv} = \frac{|A^\tau_{u2v}|}{|A_{u|v}|},$$

where A^τ_{u2v} is the set of actions that have propagated from u to v within τ time units, i.e., u has done the action within τ time units before v. $A_{u|v}$ is the set of actions performed by either u or v, independent of time. In our setting actions correspond to item ratings without distinguishing how the item is rated.

The STRIP algorithm works by extending min-wise independent hashing [7]. Let $h : \mathcal{A} \to [0,1]$ be a random hash function that maps actions to values in the interval $[0,1]$. (As shown in [22], we can assume that h is injective with high probability.) A user-action-timestamp triple (u, a_i, t_i) is then processed as follows: For each user u we keep a sample H_u that records the k action-timestamp pairs with the smallest hash values. If $h(a_i)$ is smaller than the largest entry in H_u, or H_u contains less than k entries, we add (a_i, t_i) to H_u and remove the $(k+1)$-th largest entry, if any. Implementing H_u as a priority queue guarantees fast update. Once the stream has been processed, the influence probability p_{uv} of user u on user v is estimated as

$$\frac{\Omega_\tau(H_u, H_v)}{k}$$

where $\Omega_\tau(H_u, H_v)$ denotes the set of actions in *both* H_u and H_v which satisfy the time constraint Ω.

The new approach.

Assume that ratings are small integer numbers. (Clearly, such an assumption is justified in practice, users usually rate items on a 5- or 10-scale.) We extend the STRIP approach to handle weighted similarity measures by treating each rating as being composed by r_{max} binary ratings, r_{max} being the maximum rating. More precisely, a rating r_u can be expressed as $r^u = \sum_{i=1}^{r_{max}} c^u_i$, where $c^u_i \in \{0,1\}$. The product of two ratings r^u, r^v can thus be written as

$$r^u r^v = \sum_{k=1}^{r_{max}} c^u_k \sum_{k=1}^{r_{max}} c^v_k = \sum_{k=1}^{r_{max}} \sum_{\ell=1}^{r_{max}} c^u_k c^v_\ell.$$

For example, let $r^u = 1$, $r^v = 3$ and $r_{max} = 3$. Then, $r_u = 1+0+0$, $r_v = 1+1+1$ and $r^u r^v = (1\times1+1\times1+1\times1)+(0\times1+0\times1+0\times1)+(0\times1+0\times1+0\times1)$.

Let $c^u_k \in \{0,1\}^n$ be the binary vector that corresponds to the k-th position of u's ratings. For example, assume $n=5$, $r_{max}=5$ and a user u has given following ratings: $r_{u1}=3$, $r_{u4}=5, r_{u5}=1$, items 2 and 3 have not been rated by u. We have $c^u_1 = (1,0,0,1,1)$, $c^u_2 = (1,0,0,1,0)$, $c^u_3 = (1,0,0,1,0)$, $c^u_4 = (0,0,0,1,0)$ and $c^u_5 = (0,0,0,1,0)$.

We can rewrite an inner product uv as the sum of r^2_{max} inner products of binary vectors:

$$uv = \sum_{i=1}^{n} u_i v_i = \sum_{i=1}^{n} \sum_{k=1}^{r_{max}} c^{u_i}_k \sum_{\ell=1}^{r_{max}} c^{v_i}_\ell =$$

ProcessStream

Input: stream of user-rating-timestamp triples $\mathcal{S}$, pairwise independent function $h : \mathbb{N} \to [k]$

1: **for** each $(u, r_{ui}, t_u) \in \mathcal{S}$ **do**
2: Let $c = (c_1, \ldots, c_{r_{max}})$ such that $c_\ell = 1$ for $1 \le \ell \le r_{ui}$ and $c_\ell = 0$ for $r_{ui} + 1 \le \ell \le r_{max}$.
3: **for** $j = 1$ to r_{ui} **do**
4: StripUpdate(u, i, t_u, j, h)
5: $N[u]$ += r_{ui}^2
6: $S[u]$ += r_{ui}
7: $C[u]$ += 1

StripUpdate

Input: user u, item i, timestamp t_u, sample number j, hash function $h : I \to (0, 1]$.

1: **if** $h(i) < H_u^j.getMax()$ **then**
2: $H_u^j.pop()$
3: $H_u^j.add(h(i), t_u)$

MinWiseEstimate

Input: sketches H_u^k, H_v^ℓ, constraint Ω

1: Let Min_s be the s pairs $(h(i), t)$ with the smallest hash value in $H_u^k \cup H_v^\ell$
2: **return** $\frac{|\Omega(Min_s)|}{s}$

EstimateSum

Input: user u, user v, sketches H, constraint Ω

1: $sum_u = 0$
2: **for** $k = 1$ to r_{max} **do**
3: sum_u += $MinWiseEstimate(H_u^k, H_v^1, \Omega)$
4: **return** sum_u

EstimateInnerProduct

Input: users u, v, sketches H, constraint Ω

1: $ip = 0$
2: **for** $k = 1$ to r_{max} **do**
3: **for** $\ell = 1$ to r_{max} **do**
4: ip += $MinWiseEstimate(H_u^k, H_v^\ell, \Omega)$
5: **return** ip

EstimateCosine

Input: users u, v, sketches H constraint Ω

1: ip = EstimateInnerProduct(u, v, H, Ω)
2: **return** $ip/\sqrt{N[u]N[v]}$

EstimatePearson

Input: users u, v, sketches H, arrays N, S, C, constraint Ω

1: ip = EstimateInnerProduct(u, v, H, Ω)
2: $nnz_{\Omega(u,v)}$ = MinWiseEstimate(H_u^1, H_v^1, Ω)
3: sum_Ω^u =EstimateSum(H, u, v, Ω)
4: sum_Ω^v =EstimateSum(H, v, u, Ω)
5: $m_u = S[u]/C[u]$, $m_v = S[v]/C[v]$
6: $norms_u = N[u] - 2m_u S[u] + C[u]m_u^2$
7: $norms_v = N[v] - 2m_v S[v] + C[v]m_v^2$
8: ip −= $m_u sum_\Omega^v + m_v sum_\Omega^u - nnz_{\Omega(u,v)} m_u m_v$
9: **return** $ip/\sqrt{norms_u norms_v}$

Figure 2: Similarity estimation with time constraint.

$$= \sum_{k=1}^{r_{max}} \sum_{\ell=1}^{r_{max}} \sum_{i=1}^{n} c_k^{u_i} c_\ell^{v_i} = \sum_{k=1}^{r_{max}} \sum_{\ell=1}^{r_{max}} c_k^u c_\ell^v.$$

The above suggests the following algorithm. For each user we maintain r_{max} sketches. Thus, for each user we can consider r_{max} separate binary substreams. For each such stream we will run the STRIP algorithm and maintain a min-wise independent sample. Let (u_i, t_{ui}) be an incoming rating-timestamp tuple. We have to update u's k-th min-wise sample, $1 \le k \le r_{max}$, iff $r_{ui} \le k$. Once the stream has been processed, the inner product $uv_{\Omega(\tau)}$ can be estimated as $\sum_{k=1}^{r_{max}} \sum_{\ell=1}^{r_{max}} est(c_k^u c_{\ell\,\Omega(\tau)}^v)$ where $est(c_k^u c_{\ell\,\Omega(\tau)}^v)$ is the estimated constrained inner product of the binary vectors c_k^u and c_ℓ^v. A pseudocode based on the above discussion is presented in Figure 2.

Pearson correlation. Clearly, an estimation of uv_Ω results in an estimation of $cos_{\Omega(u,v)}$. However, for Pearson correlation we need to estimate $(u - \tilde{u})(v - \tilde{v})_\Omega$. Let $sum_{\Omega(u,v)}^u = \sum_{i \in I:\Omega(u_i,v_i)} u_i$, i.e., we sum over the u_i for which $\Omega(u_i, v_i)$ is satisfied. Let $nnz_{\Omega(u,v)} = \sum_{i \in \mathcal{I}:\Omega(u_i,v_i)} 1$, i.e., the number of entries in uv_Ω. By rewriting the inner product we obtain $(u - \tilde{u})(v - \tilde{v})_\Omega = uv_\Omega - \tilde{u} sum_{\Omega(u,v)}^u - \tilde{v} sum_{\Omega(u,v)}^v + nnz_{\Omega(u,v)} \tilde{u}\tilde{v}$. Thus, we need to estimate $sum_{\Omega(u,v)}^u$, $sum_{\Omega(u,v)}^v$ and the number of nonzero entries in uv_Ω. We observe that we can rewrite $sum_{\Omega(u,v)}^u$ as $\sum_{i=1}^n \sum_{k=1}^{r_{max}} c_k^{u_i} c_{1\,\Omega(u,v)}^v$. This can be easily verified: we consider exactly those u_i for which $\Omega(u_i, v_i)$ and for each of them we add up exactly u_i 1's. The number of nonzero entries in uv_Ω is exactly the number of indices i for which $\Omega(u_i, v_i)$ is true, i.e., $c_1^u c_1^v$.

Lemma 1. *Let $z \ge 0$, $x \ge 1$ and $0 < \varepsilon < 1/2$. Then $\frac{z}{x+\varepsilon} \ge (1 - \varepsilon)\frac{z}{x}$ and $\frac{z}{x-\varepsilon} \le (1 + 2\varepsilon)\frac{z}{x}$.*

Proof. For $\varepsilon > 0$ it holds $\frac{z}{x+\varepsilon} \ge \frac{z}{(1+\varepsilon)x} = \frac{z}{x} - \frac{\varepsilon z}{(1+\varepsilon)x} \ge \frac{z}{x} - \frac{\varepsilon z}{x} = (1 - \varepsilon)\frac{z}{x}$.
Similarly, for $\varepsilon < 1/2$ we have $\frac{z}{x-\varepsilon} \le \frac{z}{(1-\varepsilon)x} = \frac{z}{x} + \frac{\varepsilon z}{(1-\varepsilon)x} \le \frac{z}{x} + \frac{2\varepsilon z}{x} = (1 + 2\varepsilon)\frac{z}{x}$ □

We first show that an inner product can be approximated using min-wise independent hashing.

Theorem 3. *Let $\mathcal{S}$ be a stream of vector entries u_i, $1 \le i \le m$ arriving in arbitrary order for different vectors $u \in \mathbb{N}^n$. Let $u_i \le r_{max}$ for all vector entries. There exists a one-pass algorithm that computes a sketch of the user activity using $O(\frac{r_{max}}{\varepsilon^2} \log r_{max})$ space per user and $O(r_{max} \log r_{max} \log^2(\frac{1}{\varepsilon}))$ processing time per pair. For a vector pair $u, v \in \mathbb{R}^n$, after preprocessing the sketches of u and v in time $O(\frac{r_{max}}{\varepsilon^2} \log(\frac{r_{max}}{\varepsilon}))$ we obtain an $\varepsilon r_{max}(|u| + |v|)$-approximation of the inner product uv with probability 2/3 in time $O(\frac{r_{max}^2}{\varepsilon^2})$.*

Proof. Consider an incoming entry u_i. We update all H_u^k for which it holds $u_i \ge k$. In H_u^k we keep the s pairs $(h(i), t_u)$ with smallest hash values. (Under the assumption that h is injective, the s pairs are well-defined.) We assume h is implemented as $(1/c, s)$-min-wise independent function [16], thus $h(i)$ can be stored in space $O(s)$ and evaluated in time $O(\log^2 s)$. Implementing H_u^k as a priority queue, the total processing time is $O(r_{max} \log^2 s)$.

We next show the quality of the estimate in terms of the sample size s. Let $A, B \subseteq [n]$ be two subsets. Let $\alpha = J(A, B)$ denote the Jaccard similarity between A and B. We first show how to obtain an ε-approximation of α. Let

$\min_s^h(A \cup B)$ denote the s smallest elements in $A \cup B$ under h. Let X be a random variable counting the number of attributes from $A \cap B$ in $\min_s^h(A \cup B)$. Clearly, it holds $E[X] = \alpha s$. The number of attributes from $A \cap B$ in size-s subsets follows hypergeometric distribution with s samples from a set of size n and αn successes. Thus, for $s > 1$ we have

$$V[X] = \alpha(1-\alpha)s\frac{n-s}{n-1} < \alpha s.$$

By Chebyshev's inequality we thus have

$$\Pr[|X - E[X]| \geq \varepsilon s] \leq \frac{V[X]}{\varepsilon^2 s^2} < \frac{\alpha}{\varepsilon^2 s}.$$

For $s = O(1/\varepsilon^2)$ we thus can bound the probability to $1/c$ for arbitrary fixed c. For h being $(1/c, k)$-wise independent we will thus obtain ε-approximation of α for $s = O(1/\varepsilon^2)$ with probability $\frac{2}{3}$.

We obtain an approximation of $|A \cup B|$ as follows. It holds $\alpha = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$. Thus, $|A \cap B| = \frac{\alpha(|A|+|B|)}{1+\alpha}$.

Consider an ε-approximation of α. For the approximation error of $|A \cap B|$ we obtain

$$\frac{(\alpha \pm \varepsilon)(|A|+|B|)}{1+\alpha \pm \varepsilon} = \frac{\alpha(|A|+|B|)}{1+\alpha \pm \varepsilon} \pm \frac{\varepsilon(|A|+|B|)}{1+\alpha \pm \varepsilon}.$$

By Lemma 1 and using $\alpha \in [0,1]$ we can thus bound the approximation error by $O(\varepsilon(|A|+|B|))$.

The total approximation error for estimating an inner product uv is then bounded by

$$\sum_{k=1}^{r_{max}} \sum_{\ell=1}^{r_{max}} c_k^u c_\ell^v \pm \varepsilon(|c_k^u| + |c_\ell^v|) =$$

$$uv \pm \varepsilon(\sum_{\ell=1}^{r_{max}} \sum_{k=1}^{r_{max}} |c_k^u| + \sum_{k=1}^{r_{max}} \sum_{\ell=1}^{r_{max}} |c_\ell^v|) = uv \pm \varepsilon r_{max}(|u|+|v|).$$

In order to guarantee the ε-approximation error with probability 2/3, we work with $t = O(\log r_{max})$ independent hash functions and sketches. By taking the median of the estimates from the t sketches, by Chernoff bounds each binary inner product $c_l^u c_\ell^v$ is approximated with probability $\frac{2}{3r_{max}^2}$. By the union bound we obtain an ε-approximation of uv with probability 2/3.

The s smallest elements in the intersection of two sets with $O(s)$ elements each can be found in $O(s)$ time after presorting the sets in time $O(s \log s)$. Thus, from the sketches H_u and H_v the inner product uv can be estimated in time $O(\frac{r_{max}^2}{\varepsilon^2})$. The time and space bounds follow directly from the above discussion. □

We next extend the above result to estimate time constrained inner product uv_Ω.

THEOREM 4. *Let $\mathcal{S}$ be a stream of entry-timestamp pairs (u_i, t_{ui}) arriving in arbitrary order for different vectors $u \in \mathbb{N}^n$ such that $u_i \leq r_{max}$. There exists a one-pass algorithm that computes a sketch of the user activity using $O(\frac{r_{max}}{\varepsilon^2})$ space and $O(r_{max} \log^2(\frac{1}{\varepsilon}) \log r_{max})$ processing time per pair. Let Ω be an appropriately defined time constraint. For any two users u, v, from the sketches of u and v we obtain an $\varepsilon(|u|+|v|)$-approximation of the inner product uv_Ω with probability 2/3 in time $O(\frac{r_{max}^2}{\varepsilon^2})$.*

Source	# users	# items	# ratings
MovieLens[1]	71,567	10,681	10M
Flixster[2]	1M	49000	8.2M

Table 1: Evaluation datasets. Both ratings sets are for movies, and in 5-scale with half-star increments.

PROOF. Consider the estimation of the time constrained inner product of two binary vectors $c_k c_{\ell\Omega}$. As in the proof of Theorem 3, we consider two sets A, B and apply minwise independent hashing in order to estimate $\alpha_\Omega = \frac{|A \cap B_\Omega|}{|A \cup B|}$ ($A \cap B_\Omega$ are the elements in $A \cap B$ that satisfy Ω). Let $\alpha = \frac{|A \cap B|}{|A \cup B|}$. By the very same reasoning as in the proof of Theorem 3 we obtain an ε-approximation of α_Ω using $O(1/\varepsilon^2)$ space and $O(\log^2(\frac{1}{\varepsilon}))$ processing time.

Assume we have computed an $\varepsilon(|A|+|B|)$-approximation of $|A \cap B|$. (By Theorem 3 we need again $O(1/\varepsilon^2)$ space and $O(\log^2(\frac{1}{\varepsilon}))$ processing time.) It holds

$$|A \cap B|_\Omega = \alpha_\Omega |A \cup B| = \alpha_\Omega(|A| + |B| - |A \cap B|).$$

With some simple algebra we can bound the approximation error to $O(\varepsilon(|A|+|B|))$. The claimed time and space bounds follow directly from the above and the discussion in the proof of Theorem 3. □

The above two theorems yield following approximation guarantees for the considered similarity measures.

COROLLARY 1. *Let $u, v \in \mathbb{N}^n$ be revealed in a streaming fashion. Let $u_i, v_i \leq r_{max}$. There exists a one-pass streaming algorithm processing each entry in time $O(r_{max} \log r_{max} \log^2(\frac{1}{\varepsilon}))$ and returning a sketch for each vector using $O(\frac{r_{max}}{\varepsilon^2})$ space. After preprocessing each sketches in time $O(\frac{r_{max}}{\varepsilon^2} \log(\frac{r_{max}}{\varepsilon}))$ we compute $\varepsilon r_{max} cos(u,v)$-approximation of $cos_\Omega(u,v)$ and $\rho_\Omega(u,v)$ in time $O(\frac{r_{max}^2}{\varepsilon^2})$.*

PROOF. Observe that for $u, v \in \mathbb{N}^n$ it holds

$$|u|+|v| = \sum_{i=1}^n (u_i + v_i) \leq 2\sum_{i=1}^n \max(u_i, v_i) \leq 2\sum_{i=1}^n u_i v_i = 2uv.$$

(The last inequality follows from $u_i, v_i \geq 1$.) Thus, for $q \geq uv - \varepsilon r_{max}(|u|+|v|)$ we have

$$\frac{q}{\|u\|\|v\|} \geq \frac{uv - \varepsilon r_{max}(|u|+|v|)}{\|u\|\|v\|} \geq \frac{(1-2\varepsilon r_{max})uv}{\|u\|\|v\|} =$$

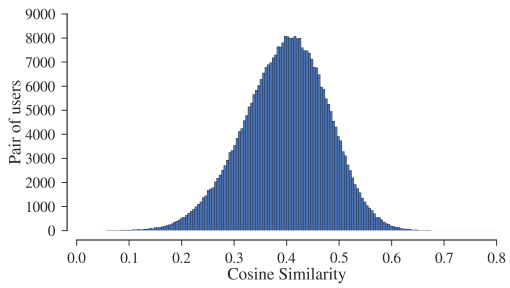
$$(1 - 2\varepsilon r_{max}) cos(u,v).$$

Similarly, for $q \leq uv + \varepsilon r_{max}(|u|+|v|)$ we obtain

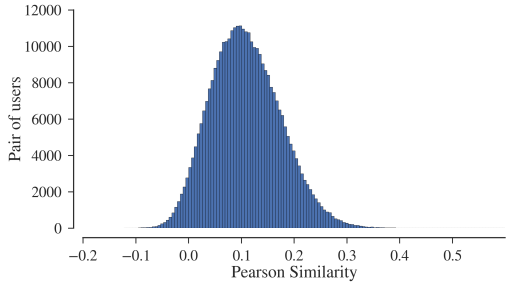
$$\frac{q}{\|u\|\|v\|} \leq (1 + 2\varepsilon r_{max}) cos(u,v).$$

Rescaling ε, we obtain the claimed bound for cosine similarity.

Consider now Pearson correlation. As shown in the proof of Theorem 3, we obtain an $\varepsilon(|c_1^u + c_1^v|)$-approximation of the inner product $c_1^u c_{1\Omega}^v \leq uv_\Omega$, and an $\varepsilon r_{max}(c_1^u + |v|)$-approximation of $c_1^u \sum_{k=1}^{r_{max}} c_k^v \leq uv$ and $nnz_{\Omega(u,v)}$, within the claimed time and space bounds. Since $|c_1^u| \leq |u|$ for any $u \in \mathbb{N}^n$, we obtain an $O(\varepsilon r_{max}(|u|+|v|))$-approximation of $(u - \tilde{u})(v - \tilde{v})_\Omega$. Dividing by $\|u\|\|v\|$ yields the claimed approximation bounds. □



(a) Cosine similarity.



(b) Pearson correlation

Figure 3: Similarity distribution for the MovieLens dataset.

5. EXPERIMENTAL EVALUATION

In this section, we present experimental evaluation of the algorithms presented in Sections 3 and 4. The main purpose of the evaluation is to validate the theoretical findings on real data. Note that the approximation quality does not depend on the dataset size. Therefore, for larger dataset the space saving becomes more pronounced. We use two publicly available datasets, MovieLens and Flixtser detailed in Table 1. The two datasets consist of movie ratings on a 5-star scale, with half-star increments. In Figure 3 we plot the distribution of cosine similarity and Pearson correlation for the MovieLens dataset. In addition, the Flixster dataset contains a who-trusts-whom social network containing links between users. For more details on the properties of the two datasets we refer to [20, 29].

Implementation: The algorithms are implemented in the Python programming language using its standard data structures for the hash tables and priority queues. All experiments were performed on commodity hardware. We worked with a single hash function implemented using tabulation hashing [8]. Even if tabulation hashing is only 3-wise independent, it is known to simulate the behavior of a truly random hash function, see Pătraşcu and Thorup [27] for Chernoff-like concentration bounds for the randomness of tabulation hashing. (Note that working with several hash functions in parallel and returning the median of the estimates results in more accurate approximation. However, this comes at the price of slower processing time and increased space usage.)

Assume all keys come from a universe $\mathcal{U}$ of size n. With tabulation hashing, we view each key $r \in \mathcal{U}$ as a vector consisting of c characters, $r = (r_1, r_2, \ldots, r_c)$, where the i-th character is from a universe $\mathcal{U}_i$ of size $n^{1/c}$. (W.l.o.g. we assume that $n^{1/c}$ is an integer). For each universe $\mathcal{U}_i$, we initialize a table T_i and for each character $r_i \in \mathcal{U}_i$ we store a random value v_{r_i} obtained from the Marsaglia Random Number CDROM [3]. Then the hash value is computed as:

$$h_0(r) = T_1[r_1] \oplus T_2[r_2] \oplus \ldots \oplus T_c[r_c]$$

where $\oplus$ denotes the bit-wise XOR operation. Thus, for a small constant c, the space needed is $O(n^{1/c} \log n)$ bits and the evaluation time is $O(1)$. In our setting keys are 32-bit integers (the item ids), and we set $c = 4$. Clearly, this yields a very compact representation of the hash function.

[3]http://www.stat.fsu.edu/pub/diehard/cdrom/

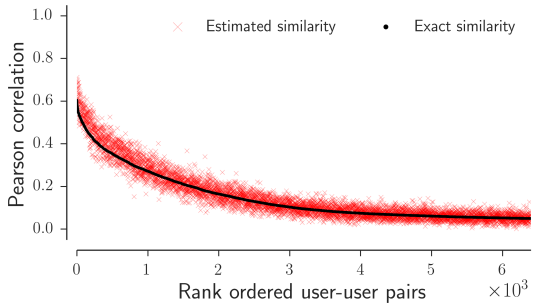
	Cosine			Pearson		
s	aae	1-dev	2-dev	aae	1-dev	2-dev
100	0.0727	0.7282	0.9706	0.0818	0.6694	0.9482
200	0.0522	0.7184	0.9696	0.0568	0.679	0.9539
300	0.0426	0.7184	0.9694	0.0459	0.6818	0.9552
400	0.0421	0.6528	0.9442	0.0404	0.6761	0.9512
500	0.0338	0.7079	0.9646	0.0362	0.6747	0.9525
600	0.031	0.7059	0.9651	0.0322	0.686	0.9572
700	0.0382	0.5557	0.8941	0.0308	0.6694	0.9505
800	0.0345	0.5636	0.9035	0.0305	0.6417	0.9377

(a) MovieLens dataset.

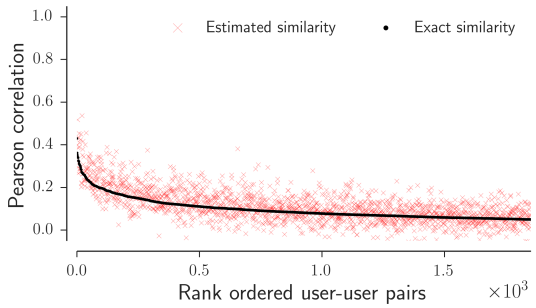
	Cosine			Pearson		
s	aae	1-dev	2-dev	aae	1-dev	2-dev
100	0.0795	0.687	0.9514	0.081	0.6801	0.9492
200	0.0518	0.7382	0.9734	0.0577	0.6683	0.9480
300	0.0451	0.6906	0.9582	0.0463	0.6779	0.9559
400	0.0376	0.7088	0.9691	0.0394	0.6861	0.9573
500	0.0346	0.7067	0.9652	0.0347	0.6946	0.9559
600	0.0342	0.6534	0.9431	0.0325	0.6795	0.9542
700	0.0265	0.7425	0.9769	0.0297	0.6874	0.9572
800	0.0281	0.6614	0.958	0.0277	0.6908	0.9588

(b) Flixster dataset.

Table 2: Quality of the similarity approximation for varying sketch sizes.



(a) Cosine similarity.



(b) Pearson correlation.

Figure 4: MovieLens approximation for sketch size $s = 200$, for pairs with similarity at least 0.1.

Evaluation: After sketching the activity for each user, we consider only users that have rated at least 1,000 movies. In MovieLens there are 840 such users and 1,204,445 ratings, and in Flixster there are 1,231 such users and 2,050,059 ratings. For the quality of approximation, we report i) the average approximation error (*aae*): $\sum_{i=1}^{n} \frac{|r_i - \tilde{r}_i|}{n}$, where $\tilde{r}$ is the approximated value of a rating r. ii) Given an approximation parameter ε, we report the quality of approximation in terms of the number of estimates $\tilde{r}_i$ that are within $[r_i - \varepsilon, r_i + \varepsilon]$ (denoted as *1-dev*), and within $[r_i - 2\varepsilon, r_i + 2\varepsilon]$ (*2-dev*). Finally, for all experiments we compute 1-dev and 2-dev w.r.t. $\varepsilon = 1/\sqrt{s}$[4], and not the more complicated form of the approximation guarantee we showed in Theorem 3. Note that the approximation guarantees in Section 4 also depend on r_{max} and the cosine similarity between users. Since the approximation quality scales with s, we present the approximation guarantees in terms of the sketch size per position, i.e., the total space usage is sr_{max}.

For the scalability of the algorithms, we report the memory requirements in terms of the sketch size (s) used, i.e., the number of entries stored in the sketch. Finally, we briefly report the run times of the algorithms on the datasets. The precise run times and actual space used highly depend on low-level technical details that are beyond the scope of the paper; we refer to [13] for a discussion on the evaluation of different implementations of streaming algorithms.

5.1 Weighted similarity estimation using AMS sketches

Having run the algorithms described in Section 3 on the datasets, we evaluated the impact of varying the sketch size (s) from 200 to 800 in increments of 100. The processing time varied between 90 - 110 seconds for MovieLens and 80-100 seconds for Flixster when varying sketch sizes.

To evaluate the approximation error (aae) and the percentage of good estimates (1-dev and 2-dev), we randomly sampled two separate groups of 300 users for each dataset and computed the similarity for every pair of users. In both MovieLens and Flixtser there are almost 90,000 pairs with cosine at least 0.1, while in MovieLens there are about 48,000 pairs with Pearson at least 0.1, and in Flixster – less than 10,000 pairs. Table 2 summarizes the results for MovieLens (Table 2a) and Flixster (Table 2b) respectively.

We observe as expected, the estimation error falls in response to increasing the sketch size (s). From the values 1-dev and 2-dev, we observe that the quality approximation is very high; the 2-dev ranges between 0.89 - 0.97 across both measures and data sets. Furthermore, the average error is always smaller than the ε-approximation given in Theorem 1 and Theorem 2. Figure 4 plots the exact alongside the approximated cosine similarity (Figure 4a), and Pearson correlation (Figure 4b) for MovieLens, with $s = 500$. As we see, despite working with a single hash function there are no outliers in the estimates.

5.2 Influence propagation probability estimation

We evaluated the algorithm listed in Section 4 on the two data sets, and tracked the influence probability between users for a period of 6 months. In order to speed up the processing time, we discretized the rating to be in a 5-star scale using ceiling r_{ui} to $\lceil r_{ui} \rceil$. However, we compute the exact similarity using the original 10-scale ratings. Because the social graph of the Flixster network is very sparse, to demonstrate statistical significance, we have increased the density of links. This is achieved by adding a new link between a pair of users u and v if $d(u) \cdot d(v) \geq 1/r$ for a random number in $(0, 1]$, where $d(u)$ is the number of neighbors of u in the network. Note that there is no social graph in the

[4]For example, for a sketch size $s = 400$ we have $\varepsilon = 0.05$

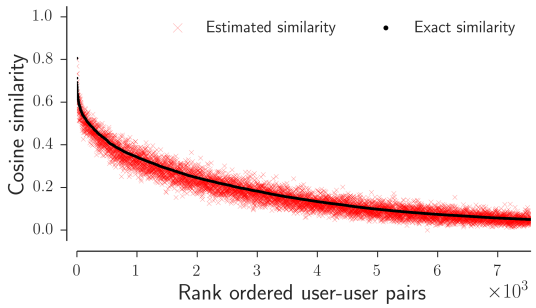
	Cosine			Pearson		
s	aae	1-dev	2-dev	aae	1-dev	2-dev
50	0.0562	0.9347	0.999	0.0842	0.8207	0.9767
100	0.0394	0.9341	0.9981	0.0711	0.748	0.9563
150	0.0271	0.967	0.9995	0.0489	0.8206	0.991
200	0.0243	0.9565	0.9987	0.0439	0.8036	0.9735
250	0.0256	0.9159	0.9924	0.0431	0.7698	0.9595
300	0.0201	0.9508	0.9975	0.0428	0.7264	0.9536
350	0.0227	0.8938	0.9826	0.0369	0.7815	0.9658
400	0.0183	0.9367	0.994	0.0331	0.779	0.9742

(a) MovieLens dataset.

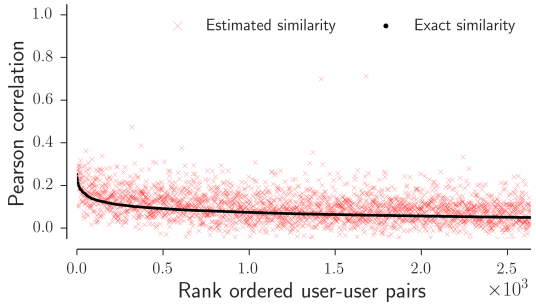
	Cosine			Pearson		
s	aae	1-dev	2-dev	aae	1-dev	2-dev
50	0.0453	0.9794	0.9992	0.109	0.7194	0.9387
100	0.0394	0.9398	0.9991	0.0867	0.6644	0.9187
150	0.0365	0.9432	0.9868	0.0586	0.7551	0.9633
200	0.033	0.8911	0.9964	0.0508	0.7592	0.9529
250	0.0291	0.9154	0.9957	0.0448	0.7439	0.9727
300	0.0254	0.9091	0.9967	0.0419	0.7139	0.9384
350	0.0245	0.8861	0.9952	0.0408	0.6926	0.92658
400	0.0241	0.8824	0.994	0.0395	0.6319	0.8995

(b) Flixster dataset.

Table 3: Quality of approximation of the influence probability for varying sketch sizes.



(a) Cosine similarity.



(b) Pearson correlation.

Figure 5: Approximation of the influence probability for Flixster dataset for sketch size $s = 200$.

MovieLens dataset. As discussed in the introduction, we are interested in users whose behaviour is a good predictor for the behaviour of other users and we consider all user pairs.

Table 3 again reports approximation error when varying the sketch size (s). We observe very precise estimates for cosine similarity for both datasets; Table 3a for MovieLens and Table 3b for Flixster. In fact these numbers are considerably better than what the theoretical analysis suggests. This is not very surprising, in [3] the authors also report a gap between theoretical and empirical approximation.

Furthermore, from Table 3 we observe that unsurprisingly, the approximation error for the Pearson correlation is higher than the corresponding Cosine similarity error. This is due to the fact that we need to estimate four different quantities which makes inaccurate estimates more likely. However, again the results show a lower error than the bounds we would expect from Theorem 3.

In Figure 5 we plot the approximation error for the Flixster dataset for sketch size 200. As evident, the approximation of the Pearson correlation is not so precise and there are a few significant outliers.

With respect to the datasets considered, space savings are made for smaller sketch sizes. For example, for a sketch size of 200 and ratings on a 5-scale, in the MovieLens dataset we need to store 840,000 samples, while for Flixster we need more than 1,2 million samples. (The pre-processed MovieLens and Flixster datasets contain 1,204,445 and 2,050,059 ratings respectively.) Finally, we observe higher running time compared to the AMS-sketch based algorithm, for MovieLens it varies between 170 and 190 seconds and for Flixster between 160 and 180. (However, this might be due to different time formats in the two datasets.)

6. CONCLUSIONS

We presented the first streaming algorithms for handling weighted similarity measures in data streams. The algorithms extend state-of-the-art techniques for scalable stream processing and are shown to be applicable to real-world domains. A few remarks are in place here. In [22] the algorithms were extended to min-wise independent hashing from sliding windows [15] where we are interested only in more recent user activity and to sublinear space usage where we are only interested in users whose activity is above certain threshold, i.e., have rated a certain amount of items. It is straightforward to extend the influence probability learning algorithm from Section 4 to also consider these extensions. However, it does not appear easy to extend the AMS sketch based algorithm from Section 3 to similarity estimation over sliding windows and active user mining. Finally, we note that it is easy to extend the here presented algorithms to the estimation of Euclidean distance, but due to lack of space we omit this result.

Acknowledgements. The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627, “mPlane”.

7. REFERENCES

[1] N. Alon, Y. Matias, M. Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.* 58(1): 137–147 (1999)

[2] A. Anagnostopoulos, R. Kumar, M. Mahdian. Influence and correlation in social networks. *KDD 2008*: 7–15

[3] Y. Bachrach, R. Herbrich. Fingerprinting Ratings for Collaborative Filtering - Theoretical and Empirical Analysis. *SPIRE 2010*: 25–36

[4] Y. Bachrach, R. Herbrich, E. Porat. Sketching Algorithms for Approximating Rank Correlations in Collaborative Filtering Systems. *SPIRE 2009*: 344–352

[5] Y. Bachrach, E. Porat. Sketching for Big Data Recommender Systems Using Fast Pseudo-random Fingerprints. *ICALP (2) 2013*: 459–471

[6] Y. Bachrach, E. Porat, J. S. Rosenschein. Sketching Techniques for Collaborative Filtering. *IJCAI 2009*: 2016–2021

[7] A. Z. Broder, M. Charikar, A. M. Frieze, M. Mitzenmacher. Min-Wise Independent Permutations. *STOC 1998*: 327–336

[8] L. Carter, M. N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci. 18(2)*: 143–154 (1979)

[9] M. Charikar. Similarity estimation techniques from rounding algorithms. *STOC 2002*: 380–388

[10] M. Charikar, K. Chen, M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci. 312(1)*: 3–15 (2004)

[11] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, C. Yang. Finding Interesting Associations without Support Pruning. *IEEE Trans. Knowl. Data Eng. 13(1)*: 64–78 (2001)

[12] G. Cormode, M. N. Garofalakis. Sketching Streams Through the Net: Distributed Approximate Query Tracking. *VLDB 2005*: 13–24

[13] G. Cormode, M. Hadjieleftheriou. Finding the frequent items in streams of data. *Commun. ACM 52(10)*: 97–105 (2009)

[14] D. J. Crandall, D. Cosley, D. P. Huttenlocher, J. M. Kleinberg, S. Suri. Feedback effects between similarity and social influence in online communities. *KDD 2008*: 160–168

[15] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput., 31(6)*:1794 – 1813, 2002.

[16] G. Feigenblat, E. Porat, A. Shiftan. Exponential time improvement for min-wise based algorithms. *Inf. Comput. 209(4)*: 737–747 (2011)

[17] A. Gionis, P. Indyk, R. Motwani. Similarity Search in High Dimensions via Hashing. *VLDB 1999*: 518–529

[18] A. Goyal, F. Bonchi, L. V. S. Lakshmanan. Learning influence probabilities in social networks. *WSDM 2010*: 241–250

[19] M. Jamali, G. Haffari, M. Ester. Modeling the temporal dynamics of social rating networks using bidirectional effects of social relations and rating patterns. *WWW 2011*: 527–536

[20] M. Jamali, M. Ester. TrustWalker: a random walk model for combining trust-based and item-based recommendation. *KDD 2009*: 397–406

[21] D. Kempe, J. M. Kleinberg, E. Tardos. Maximizing the spread of influence through a social network. *KDD 2003*: 137–146

[22] K. Kutzkov, A. Bifet, F. Bonchi, A. Gionis. STRIP: stream learning of influence probabilities. *KDD 2013*: 275–283

[23] Y. Kwon. Improving top-n recommendation techniques using rating variance. *RecSys 2008*: 307–310

[24] P. Li, A. C. König. b-Bit minwise hashing. *WWW 2010*: 671–680

[25] P. Massa, P. Avesani. Trust-aware recommender systems. *RecSys 2007*: 17–24

[26] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[27] M. Pătraşcu, M. Thorup. The Power of Simple Tabulation Hashing. *J. ACM 59(3)*: 14 (2012)

[28] D. Ravichandran, P. Pantel, E.H. Hovy. Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions for High Speed Noun Clustering. *ACL 2005*

[29] A. Said, B. J. Jain, S. Albayrak. Analyzing weighting schemes in collaborative filtering: cold start, post cold start and power users. *SAC 2012*: 2035–2040

[30] A. Shrivastava, P. Li. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). *NIPS 2014*: 2321–2329