D^{2}HistoSketch: Discriminative and Dynamic Similarity-Preserving Sketching of Streaming Histograms

Dingqi Yang, Bin Li, Laura Rettig, and Philippe Cudré-Mauroux

Abstract—Histogram-based similarity has been widely adopted in many machine learning tasks. However, measuring histogram similarity is a challenging task for streaming histograms, where the elements of a histogram are observed one after the other in an online manner. The ever-growing cardinality of histogram elements over the data streams makes any similarity computation inefficient in that case. To tackle this problem, we propose in this paper D^{2}HistoSketch, a similarity-preserving sketching method for streaming histograms to efficiently approximate their Discriminative and Dynamic similarity. D^{2}HistoSketch can *fast* and *memory-efficiently* maintain a set of compact and fixed-size sketches of streaming histograms to approximate the similarity between histograms. To provide high-quality similarity approximations, D^{2}HistoSketch considers both *discriminative* and *gradual forgetting* weights for similarity measurement, and seamlessly incorporates them in the sketches. Based on both synthetic and real-world datasets, our empirical evaluation shows that our method is able to efficiently and effectively approximate the similarity between streaming histograms while outperforming state-of-the-art sketching methods. Compared to full streaming histograms with both discriminative and gradual forgetting weights in particular, D^{2}HistoSketch is able to dramatically reduce the classification time (with a 7500x speedup) at the expense of a small loss in accuracy only (about 3.25%).

Index Terms—Similarity-Preserving Sketching, Histograms, Streaming Data, Concept Drift, Discriminative Weighting

1 Introduction

Histograms are an important statistic reflecting the empirical distribution of data. They have been widely used not only as a popular data analysis and visualization tool, but also as an important feature for measuring similarities between data instances, such as color histograms for images or word histograms for documents. As a result, histogram-based similarity measures have been extensively exploited in many classification and clustering tasks and for various application domains, including image processing [1], document analysis [2], social network analysis [3], and business intelligence [4].

Despite its importance in machine learning, computing histogram-based similarities is often difficult in practice, particularly for data streams. In this study, we consider *streaming histograms, where the elements of a histogram are observed over a data stream* as shown in Fig. 1(a) in Section 3. Streaming histograms can be used for a wide range of applications, such as solving range queries and similarity search in a streaming database, change detection and classification over data streams. In practice, streaming histograms are often seen when online or offline businesses observe their customers' activity data. For example, a Point of Interest (POI), such as a supermarket or a restaurant, may observe a continuous data stream of visits from its customers and consider to analyze the histogram of its customers' visits. By measuring the similarity between two POIs based on such histograms, one can build various high-quality applications. For example, semantic place labeling [4] infers a POI's type based on the assumption that two POIs sharing similar histograms of their customers probably belong to the same type. However, it is challenging to measure the similarity between such streaming histograms in practice, due to the ever-increasing cardinality of the histogram elements over time. In the above example, this corresponds to the case of an ever-growing number of customers. The monotonically increasing size of the streaming histograms makes any similarity computation inefficient, which further makes learning algorithms impractical.

To solve this problem, similarity-preserving data sketching (hashing) techniques [5] have been intensively studied in stream data processing [6], [7]. Their key idea is to maintain a set of compact and fixed-size sketches for the original data to approximate their similarity under a certain measure. In the current literature, most existing data sketching techniques [8], [9], [10], [11] consider the case of streaming data instances, where complete data instances are received one by one from a data stream (e.g., a stream of images whose color histogram can be easily derived). In contrast, a streaming histogram assumes that the elements of a histogram describing an individual data instance are continuously received in arbitrary order from a data stream (e.g., the histogram of customers' visits to a POI), which departs from classical techniques that focus on sketching complete data instances. Therefore, these methods cannot be efficiently applied for sketching streaming histograms.

In this paper, we tackle the similarity-preserving data

Dingqi Yang, Laura Rettig and Philippe Cudré-Mauroux are with the Department of Informatics at the University of Fribourg, Switzerland, E-mail: {dingqi.yang, laura.rettig, philippe.cudre-mauroux}@unifr.ch. Bin Li is with School of Computer Science, Fudan University, Shanghai, China, E-mail: libin@fudan.edu.cn.

Manuscript received xxx; revised xxx.

sketching problem for streaming histograms. Specifically, an efficient similarity-preserving sketching method for streaming histograms should allow for *fast* and *memory-efficient* maintenance of the sketches. *Fast maintenance* requires sketches of streaming histograms to be incrementally updatable. In other words, the new sketch of a streaming histogram should be incrementally computed from the former sketch and the newly arrived element. Moreover, *memory-efficient maintenance* requires the sketching method to create a small and bounded memory overhead when computing the sketches, which differs from existing sketching methods that require a large set of random variables as in-memory parameters [11], [12], [13], [14], where the size of these parameters is proportional to the cardinality of the histogram elements. In addition, to maintain high-quality similarity-preserving sketches, the following two issues should be considered when measuring similarities.

First, as histogram elements are not all equally important when measuring histogram-based similarity, discriminative similarity should be considered, which refers to the similarity that improves the discriminative capability of some classification/clustering methods [15]. Specifically, in the case of labeled histograms, a histogram element appearing only in the histograms of a specific label has more discriminative capability than one appearing uniformly in all histograms. Taking the example of semantic place labeling where we want to classify POIs according to their customers' visiting patterns, it means that visits from users having stronger preferences on visiting a specific type of POIs are more discriminative. For static datasets, such a discriminative similarity can be easily computed using various feature weighting methods [16] to give a higher weight to more discriminative histogram elements. However, it is not straightforward to incorporate such a discriminative similarity in sketching streaming histograms, where discriminative weights have to be updated over time, and more importantly, to be incorporated in the sketches.

Second, as a common problem in data streams, concept drift should also be taken into account for streaming histograms, where the underlying distribution of a streaming histogram changes over time in unforeseen ways. Taking the example of customers' visits to POIs, the customer population of a restaurant may change abruptly if the restaurant changes its type (e.g., from a Japanese restaurant to a pizzeria), or gradually if it updates its menu. It is therefore critical to consider this issue in sketching streaming histograms. The most common approach to handle concept drift is forgetting the outdated data [17]. A typical solution is *gradual forgetting*, where the streaming data are associated with weights inversely proportional to their age [18]. Taking exponential decay [19] as an example, the weight of a histogram element decreases by a weight decay factor every time when a new element is received from the data stream. Although such a weighting process is easy to implement when building a histogram from its streaming elements, it is not straightforward to incorporate such weights dynamically in sketching streaming histograms. This problem becomes even more challenging when further considering the requirement of incrementally updatable sketching.

To address the above challenges, we introduce D^{2}HistoSketch, a similarity-preserving sketching method for streaming histograms to approximate their *Discriminative* and *Dynamic* similarity. D^{2}HistoSketch is designed to efficiently maintain a set of compact and fixed-sized sketches over streaming histograms to approximate the similarity between the histograms. Specifically, to measure the similarity between histograms, our method focuses on normalized min-max similarity, which has been proven to be an effective similarity measure for nonnegative data in various application domains [11]. To create a sketch from a histogram, we borrow the idea from consistent weighted sampling [20] that was originally proposed for approximating min-max similarity for complete data instances. In addition, we formally derive a memory-efficient sketching method with few in-memory parameters. To efficiently maintain the sketch over the streaming histogram elements, we first adjust the original sketch to seamlessly incorporate both discriminative weights and gradual forgetting weights, and then incrementally compute the new sketch based on the adjusted sketch and the incoming histogram element. Our main contributions can be summarized as follows:

- To the best of our knowledge, this is the first work considering the discriminative and dynamic similarity-preserving sketching over streaming histograms.
- We design an efficient similarity-preserving sketching method for streaming histograms, D^{2}HistoSketch, which allows for *fast* and *memory-efficient* maintenance of the sketches, where the sketches can be incrementally updated with a small and bounded memory overhead.
- To provide high-quality similarity-preserving sketches for downstream tasks, D^{2}HistoSketch considers both *discriminative* similarity that improves the discriminative capability of the sketches for some classification/clustering methods, and *dynamic* similarity that adapts to concept drift by gradually forgetting outdated histogram elements.
- We empirically evaluate our method on multiple classification tasks using both synthetic and real-world datasets. Our results show that D^{2}HistoSketch is able to efficiently approximate discriminative and dynamic similarity for streaming histograms. Compared to full streaming histograms, D^{2}HistoSketch is able to dramatically reduce the classification time (with a 7500x speedup) at the expense of a modest loss in accuracy (about 3.25%).

Compared to our previous work HistoSketch [14], in this paper, we first formally derive a *memory-efficient* sketching method with fewer in-memory parameters, which also allows for faster sketch maintenance. We then add *discriminative* similarity for better similarity measurement. In particular, the new experiments we present show that compared to our previous work our newly proposed D^{2}HistoSketch achieves both a higher classification accuracy (with a 2.78% improvement on average) as well as faster sketch maintenance (with a 5.9% improvement on processing speed).

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 first presents the preliminary of our work. Afterwards, we present the proposed D^{2}HistoSketch method in Section 4. The experimental evaluation is shown in Section 5. We conclude our work in Section 6.

2 Related Work

As a key statistical tool in empirical data analysis, histograms have been widely used not only as a popular visualization of empirical data distribution [21], but also as a feature to measure data similarity that is further exploited in many machine learning tasks [1], [2], [3], [22], [23]. Although the histogram of a static dataset can often be easily computed, it is practically difficult to compute histograms for data streams with typically unknown cardinality and which thus require an unbounded amount of memory to maintain the histogram. In this context, count sketch [24] and count-min sketch [25] and other online histogram building methods [26] were proposed to approximate the frequency table of elements (i.e., histograms) from a data stream with a fixed-size data structure. However, the resulting sketches do *not* preserve the similarity between different data streams. This paper differs from the objective of the above sketching methods by addressing the problem of similarity-preserving sketching of data streams.

Similarity-preserving sketching [5] has been extensively studied to efficiently approximate the similarity of high dimensional data, such as graphs [27], images [28], [29] and videos [30], [31]. Its basic idea is to maintain a set of compact sketches of the original high dimensional data to efficiently approximate their similarities, such as Jaccard [8], [9], cosine [10], and min-max [4], [11], [12], [13], [20], [32] similarities. These sketches can then enable many applications, particularly for information retrieval systems like image or document search engines [28]. However, most of the existing methods are designed to sketch complete data instances, which are fundamentally different from streaming histograms, where histograms are incrementally built from the streams of its elements. More importantly, little attention has been given on studying high-quality similarity-preserving sketches, on which we put a particular focus in this paper by considering both discriminative and dynamic similarities.

Discriminative similarity refers to the similarity that improves the discriminative capability of some classification/clustering methods [15]. It has been widely used in many machine learning applications. For example, discriminative similarity has been shown to be able to significantly improve the performance of various computer vision and pattern recognition tasks [33]. In practice, discriminative similarity can be computed using feature weighting techniques, where a similarity function is parameterized with discriminative weights [16] (e.g., entropy weights [34]). Although it is easy to implement discriminative similarity for static datasets, it is not straightforward to apply it to similarity-preserving sketching of streaming histograms, as such discriminative weights have to be dynamically recomputed over time, and more importantly, to be incorporated in the sketches.

Dynamic adaptation to concept drift is a common problem in streaming data processing. It refers to the case where the underlying statistical properties of the streaming data change over time (often in unknown ways), which further degrades the performance of learning algorithms [35]. According to a recent survey on concept-drift, the most popular approach to handling data streams with unknown dynamics is forgetting outdated data [17]. Existing solutions can be classified into two categories. First, the abrupt forgetting approach selects a set of data for learning. A sliding window is often used to select recent data. Although this approach is effective against abrupt drifts (in terms of the data statistical properties), it is less applicable to gradual drifts [36] as it gives the same significance to all selected pieces of data while completely discarding all other data. Second, the gradual forgetting approach assigns weights that are inversely proportional to the age of the data [18], such as exponential decay weights [19]. In this study, we advocate the gradual forgetting approach to tackle the concept-drift problem in streaming histograms. The histogram is built with weighted elements from the streams, with weights decreasing over time. Different from existing methods using gradual forgetting against concept drift, we consider incorporating the gradual forgetting approach in similarity-preserving sketching of streaming histograms.

Compared to our previous work HistoSketch [14], this paper makes the following improvements. First, we formally derive a *memory-efficient* sketching method D^{2}HistoSketch with few in-memory parameters, while our previous work HistoSketch requires a large set of random variables as in-memory parameters for sketching, where the size of these parameters is proportional to the cardinality of the histogram elements. Second, we consider both *discriminative* and *dynamic* similarity in D^{2}HistoSketch, while HistoSketch only considers the dynamic adaptation to concept drift. Finally, we conduct new experiments to further validate D^{2}HistoSketch; *in particular, compared to our previous work HistoSketch, our newly proposed D^{2}HistoSketch achieves both a higher classification accuracy (with a 2.78% improvement on average) as well as faster sketch maintenance (with a 5.9% improvement on processing speed).*

3 Preliminary and Problem Formulation

In this section, we first formally introduce streaming histograms and then min-max similarity, followed by our problem definition of similarity-preserving sketching.

3.1 Streaming Histogram

We consider a streaming histogram computed over a data stream of its elements x_t where $t \in \mathbb{N}$ indicates the order of the observed element in the stream. Element $x_t \in \mathcal{E}$ are observed one by one, where $\mathcal{E}$ refers to the set of all histogram elements observed in any streaming histograms so far, which expands over time with new (unseen) histogram elements. In other words, $\mathcal{E}$ can be regarded as the union of histogram elements from all histograms observed so far. Due to the streaming nature, the cardinality $|\mathcal{E}|$ is unknown and continuously increases over time. A classical histogram can then be represented as a vector $V \in \mathbb{N}^{|\mathcal{E}|}$, where each value V_i encodes the cumulative count of the corresponding histogram elements $i \in \mathcal{E}$, i.e., $V_i = \sum_t \mathbb{1}_{x_t=i}$, where $\mathbb{1}_{cond}$ is an indicator function which is equal to 1 when $cond$ is true and 0 otherwise. Fig. 1(a) illustrates an example of a classical streaming histogram.

To measure the discriminative and dynamic similarity between streaming histograms, we build the histogram V

from its weighted elements. Specifically, we assign a discriminative weight and a gradual forgetting weight to each streaming element to measure discriminative and dynamic similarity between streaming histograms, respectively.

3.1.1 *Streaming Histogram with Discriminative Weights*

Each histogram element i is associated with a discriminative weight w_i^d. Specifically, we assume that each histogram is associated with a label l, where $l \in \mathcal{L}$ and $\mathcal{L}$ refers to a set of labels for histograms. Taking the histogram of customers' visits to a POI as an example, a POI can be associated with a category (label l) "fast food restaurant", and $\mathcal{L}$ refers to all possible POI categories here. To compute the discriminative weight w_i^d for each histogram element i, we leverage a widely used weighting function, i.e., entropy weighting [34]. (We note that our approach is not limited to any specific weighting function.) Specifically, we use entropy weighting to empirically measure the uncertainty of the labels when observing individual histogram elements. For each histogram elements i in $\mathcal{E}$, we compute its entropy weight w_i^d as follows:

$$w_i^d = 1 + \frac{\sum_{l \in \mathcal{L}} Pr[l,i] \log Pr[l,i]}{\log |\mathcal{L}|} \tag{1}$$

where $Pr[l,i]$ is the probability that a histogram with element i is labeled as l, $l \in \mathcal{L}$. Higher values of w_i^d imply higher degrees of discriminability for the corresponding histogram element i. To calculate $Pr[l,i]$, we maintain a label frequency vector F^l of size $|\mathcal{E}|$ for each label to record the cumulative frequency of each histogram element in $\mathcal{E}$. With each incoming histogram element i, we update F_i^l accordingly. Thus, we are able to empirically compute $Pr[l,i] = \frac{F_i^l}{\sum_{l' \in \mathcal{L}} F_i^{l'}}$ at any time. Particularly, for each incoming histogram element i, only the corresponding w_i^d needs to be updated. In practice, F^l is maintained in a count-min sketch data structure for memory efficiency (we will discuss it in Section 4.4).

Subsequently, we compute the histogram $V \in \mathbb{R}_{>0}^{|\mathcal{E}|}$ such that V_i is associated with its weighted w_i^d, i.e., $V_i = \sum_t w_i^d \mathbb{1}_{x_t=i}$. Fig. 1(b) shows an example of a streaming histogram with discriminative weights.

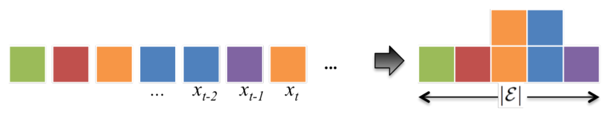
3.1.2 *Streaming Histogram with Gradual Forgetting Weights*

Each streaming element x_t is associated with a gradual forgetting weight w_t^g, which is inversely proportional to its age. To compute w_t^g, we adopt the exponential decay weight [19], which is computed as follows:

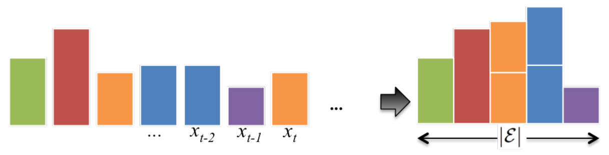
$$w_t^g = e^{-\lambda(t_n - t)} \tag{2}$$

where t_n is the order of the latest histogram element received from the stream and λ is the weight decay factor.

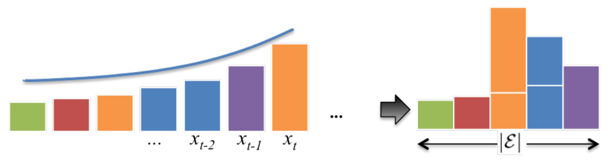
Subsequently, we compute the histogram $V \in \mathbb{R}_{>0}^{|\mathcal{E}|}$ such that V_i is the weighted cumulative count of the corresponding histogram elements i, i.e., $V_i = \sum_t w_t^g \mathbb{1}_{x_t=i}$. Fig. 1(c) shows an example of a streaming histogram with gradual forgetting weights.



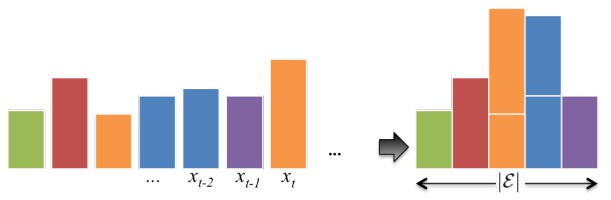
(a) Classical histogram (unweighted) $V_i = \sum_t \mathbb{1}_{x_t=i}$



(b) Histogram with discriminative weights $V_i = \sum_t w_i^d \mathbb{1}_{x_t=i}$



(c) Histogram with gradual forgetting weights $V_i = \sum_t w_t^g \mathbb{1}_{x_t=i}$



(d) Histogram with both weights $V_i = \sum_t w_i^d \cdot w_t^g \mathbb{1}_{x_t=i}$

Fig. 1. Illustration of the streaming histograms with different weights. Left: The different histogram elements are assigned different colors, and their heights indicate the corresponding weights. Right: The same elements are accumulated to build the corresponding histogram.

3.1.3 *Streaming Histogram with Both Weights*

We combine both the discriminative and the gradual forgetting weights to compute the histogram, $V_i = \sum_t w_i^d \cdot w_t^g \mathbb{1}_{x_t=i}$. Fig. 1(d) shows an example of a streaming histogram with both weights.

3.2 Normalized Min-Max Similarity

To measure the similarity between streaming histograms, we resort to normalized min-max similarity, which has been shown to be an effective similarity measure for nonnegative data. More precisely, Li [11] conducted an extensive study on min-max similarity (named as min-max kernel in [11]) by comparing four kernels including linear kernel, min-max kernel, normalized-min-max kernel and intersection kernel, on different classification tasks over a sizable collection of public datasets. The results illustrate the advantages of the (normalized) min-max kernels. Formally, given two streaming histograms V^a and V^b, the min-max similarity is defined as follows:

$$Sim_{MM}(V^a, V^b) = \frac{\sum_{i \in \mathcal{E}} \min(V_i^a, V_i^b)}{\sum_{i \in \mathcal{E}} \max(V_i^a, V_i^b)} \tag{3}$$

As a histogram is often used to characterize the empirical data distribution, we apply the sum-to-one normalization before computing the similarity:

$$\sum_{i \in \mathcal{E}} V_i^a = 1, \ \sum_{i \in \mathcal{E}} V_i^b = 1. \tag{4}$$

In that way, Eq. 3 becomes the normalized min-max similarity, denoted by Sim_{NMM}.

3.3 Problem Formulation

For streaming histograms, the ever-increasing cardinality $|\mathcal{E}|$ makes the computation of the normalized min-max similarity become inefficient. Therefore, we propose to maintain two sketches S^a and S^b of size K ($K \ll |\mathcal{E}|$) for V^a and V^b, respectively, with the property that their collision probability (i.e., $Pr[S^a_j = S^b_j]$, where $j = 1, 2, ..., K$) is exactly the normalized min-max similarity between V^a and V^b:

$$Pr[S^a_j = S^b_j] = Sim_{NMM}(V^a, V^b) \tag{5}$$

Then, the normalized min-max similarity between V^a and V^b can be approximated by the Hamming similarity between S^a and S^b. The computation over S, which is compact and of fixed size, is much more efficient than the one over the full histogram V, which is a large, ever-growing vector.

The problem tackled by this paper is how to create and maintain the above similarity-preserving sketch S for the streaming histogram V with both discriminative and gradual forgetting weights in a fast and memory-efficient manner.

4 D^{2}HistoSketch

Our D^{2}HistoSketch is designed to efficiently maintain a set of compact and fixed-size sketches for streaming histograms with both discriminative and gradual forgetting weights, in order to efficiently approximate their normalized min-max similarities. In this section, we first present consistent weighted sampling, which inspired our method. We then describe our method for sketch creation, followed by the proposed incremental sketch update process.

4.1 Consistent Weighted Sampling

Consistent weighted sampling was originally proposed to approximate min-max similarity for complete and high dimensional data (e.g., a vector of large size) [11], [12], [13], [20]. The basic idea is to generate data samples such that the probability of drawing identical samples for a pair of vectors is equal to their min-max similarity. A set of such samples can then be regarded as a sketch of the input vector.

The first consistent weighted sampling method [20] was designed to handle integer vectors. Specifically, taking a classical histogram $V \in \mathbb{N}^{|\mathcal{E}|}$ as an example, it first uses a random hash function h_j to generate independent and uniform distributed random hash values $h_j(i, f)$ for each (i, f), where $i \in \mathcal{E}$ and $f \in \{1, 2, ..., V_i\}$, and then returns $(i^*_j, f^*_j) = \mathrm{argmin}_{i \in \mathcal{E}, f \in \{1,2,...,V_i\}} h_j(i, f)$ as one sample (i.e., one sketch element S_j). Note that the random hash function h_j depends only on (i, f), and maps (i, f) uniquely to $h_j(i, f)$. By applying K independent random hash functions ($j = 1, 2, ..., K$), we generate sketch S (of size K) from V (of arbitrary size). Following this process, the collision probability between two sketch elements (i^{a*}_j, f^{a*}_j) and (i^{b*}_j, f^{b*}_j), which are generated from V^a and V^b, respectively, is proven to be exactly the min-max similarity of the two vectors [20], [32]:

$$Pr[(i^{a*}_j, f^{a*}_j) = (i^{b*}_j, f^{b*}_j)] = Sim_{MM}(V^a, V^b) \tag{6}$$

To improve the efficiency of the above method and allow real vectors as input, Ioffe [12] later proposed an improved method. Its key idea is that, rather than generating V_i different random hash values (where V_i has to be an integer), it directly generates one hash value $a_{i,j}$ (and its corresponding $f \in \mathbb{N}$, $f \le V_i$) for each i by taking V_i as the input of the random hash value generation process. In such a case, V_i can be any positive real number. Based on this method, Li [11] further proposed to simplify the sketch by only keeping i^*_j rather than (i^*_j, f^*_j), and empirically proved the following property:

$$Pr[i^{a*}_j = i^{b*}_j] \approx Pr[(i^{a*}_j, f^{a*}_j) = (i^{b*}_j, f^{b*}_j)] \tag{7}$$

A short description of the method proposed in [11] is presented in the following. To generate one sketch element S_j (sample i^*_j), the method first draws three random variables offline as in-memory parameters: $r_{i,j} \sim Gamma(2, 1)$, $c_{i,j} \sim Gamma(2, 1)$ and $\beta_{i,j} \sim Uniform(0, 1)$, and then computes

$$y_{i,j} = \exp\left(r_{i,j}\left(\lfloor\frac{\log V_i}{r_{i,j}} + \beta_{i,j}\rfloor - \beta_{i,j}\right)\right) \tag{8}$$

$$a_{i,j} = \frac{c_{i,j}}{y_{i,j}\exp(r_{i,j})} \tag{9}$$

The sketch element is then returned as $S_j = \mathrm{argmin}_{i \in \mathcal{E}}\, a_{i,j}$. Please refer to [11], [12] for more details and for the proof of Eq. 6 and 7.

In this paper, we design a fast and memory-efficient sketching process to handle streaming histograms with both discriminative and gradual forgetting weights, where $V \in \mathbb{R}^{|\mathcal{E}|}_{>0}$. Specifically, we propose a new approach to directly compute $a_{i,j}$, which has the following three highly desirable properties:

1) The generated $a_{i,j}$ follows the exact same distribution as the one generated using Eq. 9, which ensures the correctness of our sketching method (i.e., Eq. 6 and Eq. 7 still hold).
2) The $a_{i,j}$ sampling process does not require the large set of in-memory parameters $r_{i,j}, c_{i,j}, \beta_{i,j}$ where $i \in \mathcal{E}$ and $j = 1, 2, ..., K$; this makes our method memory-efficient.
3) The created sketch S is invariant under uniform scaling of V, which serves as a basis for the fast incremental sketch update.

In the following, we first present our sketch creation method, and then the incremental sketch update process.

4.2 Sketch Creation

Our sketch creation method borrows the idea of consistent weighted sampling with real number inputs [11]. Different from the original method, we propose a new approach to compute $a_{i,j}$. Specifically, the objective of the original method is to sample $y_{i,j}$ such that $\log y_{i,j}$ is uniformly distributed on $[\log V_i - r_{i,j}, \log V_i]$ conditioned on $r_{i,j}$. Among many possible formulations that can fulfill this distribution requirement, the original formulation (Eq. 8) is specifically designed to also sample the corresponding f_j to obtain the sketch element (i^*_j, f^*_j) (where $f = \lfloor\frac{\log V_i}{r_{i,j}} + \beta_{i,j}\rfloor$ in [12]). However, as proved in [11], f^*_j can be ignored from the sketch (i_j^*, f_j^*) (i.e., Eq. 7). In such a case, it is only necessary to sample $y_{i,j}$ satisfying its distribution requirement, and then compute the corresponding $a_{i,j}$. In the following, we first derive a simplified method to generate $y_{i,j}$, based on which we then derive a memory-efficient method to directly compute $a_{i,j}$.

First, we propose to compute $y_{i,j}$ as follows:

$$y_{i,j} = \exp(\log V_i - r_{i,j}\beta_{i,j}) \tag{10}$$

for which the following proposition holds.

Proposition 1. *Eq. 10 generates $y_{i,j}$ following the same distribution as generated by Eq. 8, i.e., $\log y_{i,j}$ follows a uniform distribution on $[\log V_i - r_{i,j}, \log V_i]$ conditioned on $r_{i,j}$.*

Proof. Considering the variable $\log y_{i,j}$, Eq. 8 can be derived as (we ignore the subscript (i,j) of r and β in the following proof):

$$\begin{aligned}\log y_{i,j} &= r\left(\lfloor\frac{\log V_i}{r} + \beta\rfloor - \beta\right)\\ &= \log V_i - r\left(\left(\frac{\log V_i}{r} + \beta\right) - \lfloor\frac{\log V_i}{r} + \beta\rfloor\right)\end{aligned} \tag{11}$$

where $(\frac{\log V_i}{r} + \beta) - \lfloor\frac{\log V_i}{r} + \beta\rfloor$ is the *frac function* of $\frac{\log V_i}{r} + \beta$, *which returns its fractional part* [37]. Since both V_i and r are known, this frac function can be considered as frac$(\beta + C)$, where $C = \frac{\log V_i}{r}$ is a constant. Considering $\beta \sim Uniform(0,1)$, this function actually returns the fractional part of a variable following $Uniform(C, C+1)$, which remains the same as $Uniform(0,1)$. In other words, it is a uniform mapping from $Uniform(0,1)$ to itself. Subsequently, we have frac$(\frac{\log V_i}{r} + \beta) \sim Uniform(0,1)$, which can be replaced by β. Therefore, we obtain:

$$\log y_{i,j} = \log V_i - r\beta \tag{12}$$

which is the same as in Eq. 10. Therefore, $y_{i,j}$ generated by Eq. 10 follows the same distribution as generated by Eq. 8.

We introduce $z = \log y_{i,j}$ and compute its Cumulative Distribution Function (CDF) as follows:

$$\begin{aligned}Pr(Z < z) &= Pr\left(\log V_i - r\beta < z\right)\\ &= Pr\left(\frac{\log V_i - z}{r} < \beta\right)\end{aligned} \tag{13}$$

Considering $\beta \sim Uniform(0,1)$, we obtain:

$$Pr(Z < z) = 1 - \frac{\log V_i - z}{r} = \frac{z - (\log V_i - r)}{\log V_i - (\log V_i - r)} \tag{14}$$

which is the CDF of $Uniform(\log V_i - r, \log V_i)$. This completes the proof. □

Second, based on Proposition 1, we derive a memory-efficient method to directly compute $a_{i,j}$ as follows:

$$a_{i,j} = \frac{-\log\beta}{V_i} \tag{15}$$

for which the following proposition holds.

Proposition 2. *Eq. 15 generates $a_{i,j}$ following the same distribution as generated by Eq. 9.*

Proof. As suggested by Proposition 1, we now use Eq. 10 to sample $y_{i,j}$. Combining Eq. 10 with Eq. 9 and knowing $(1-\beta)$ and β both follow $Uniform(0,1)$, we obtain:

$$a_{i,j} = \frac{c \cdot \exp(-r\beta)}{V_i} \tag{16}$$

Considering a new variable $m = r\beta$, we can compute its Probability Distribution Function (PDF) as:

$$f_M(m) = \int_{0^+}^{1} \frac{1}{\beta} f_B(\beta) f_R(\frac{m}{\beta}) d\beta \tag{17}$$

where $f_B(\cdot)$ and $f_R(\cdot)$ are the PDF of $\beta \sim Uniform(0,1)$ and $r \sim Gamma(2,1)$, respectively. We further derive $f_M(m)$ as follows:

$$f_M(m) = \int_{0^+}^{1} \frac{1}{\beta} \cdot 1 \cdot \frac{m}{\beta} \exp(-\frac{m}{\beta}) d\beta = \exp(-m) \tag{18}$$

We see that $m = r\beta$ follows the exponential distribution $Exp(1)$, and can then be sampled as $m = -\log\beta'$, $\beta' \sim Uniform(0,1)$. Subsequently, Eq. 16 can be simplified as:

$$a_{i,j} = \frac{c\beta'}{V_i} \tag{19}$$

It is worth noting that the numerator $c\beta'$ follows the same distribution as $r\beta \sim Exp(1)$, as they are both the multiplication of a variable from $Uniform(0,1)$ and a variable from $Gamma(2,1)$. Therefore, $c\beta'$ can be further simplified as $-\log\beta''$ where $\beta'' \sim Uniform(0,1)$. We note that as $\beta, \beta', \beta'' \sim Uniform(0,1)$, we keep using only $\beta \sim Uniform(0,1)$ for the sake of simplicity for notations, and obtain:

$$a_{i,j} = \frac{-\log\beta}{V_i} \tag{20}$$

which is the same as Eq. 15. This completes the proof. □

Proposition 2 ensures the correctness of our sketching method (i.e., Eqs. 6 and 7). More importantly, our method only requires the parameters $\beta_{i,j} \sim Uniform(0,1)$, which can be efficiently generated using random hash functions rather than being maintained as in-memory parameters.

4.2.1 Memory-Efficient Sketching Implementation

To efficiently generate $\beta_{i,j}$ in an online manner, we apply a random hash function h_j on i to obtain the corresponding hash value $h_j(i)$, which follows a uniform distribution over $(0,1)$. We then use $h_j(i)$ to replace $\beta_{i,j}$. With K independent random hash functions $\{h_j | j = 1, 2, ..., K\}$, we can compute $a_{i,j}$ in a memory-efficient manner as follows:

$$a_{i,j} = \frac{-\log h_j(i)}{V_i} \tag{21}$$

In summary, our memory-efficient method maintains only K independent random hash functions $\{h_j | j = 1, 2, ..., K\}$, rather than the large set of in-memory parameters $r_{i,j}, c_{i,j}, \beta_{i,j}$ whose size is proportional to the cardinality of the histogram elements $\mathcal{E}$. Alg. 1 shows the sketch creation process. Note that we keep both the sketch S and its hash values A (the latter will be used for incremental sketch update). Fig. 2 shows the sketch creation process for one sketch element S_j.

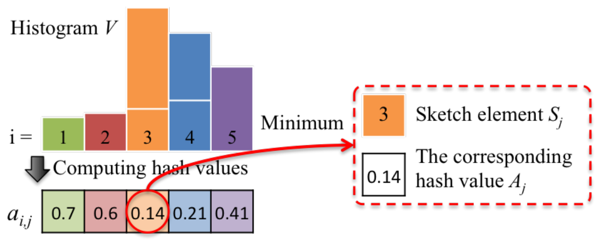


Fig. 2. Creating one sketch element from histogram V with cardinality $|\mathcal{E}| = 5$ ($i = 1, 2, ..., 5$). By computing the hash value $a_{i,j}$ for each i, we select the histogram element whose hash value is minimal as the sketch element and also keep its corresponding hash value, i.e., ($S_j = 3$, $A_j = 0.14$).

Algorithm 1 Sketch creation

```
Input: Histogram $V$, Sketch length $K$, Independent random hash functions $\{h_j | j = 1, 2, ..., K\}$
Output: Sketch $S$ and the corresponding hash values $A$
1: for j=1,2,...,K do
2:     Compute $a_{i,j} = \frac{-\log h_j(i)}{V_i}$
3:     Set sketch element $S_j = \text{argmin}_{i\in\mathcal{E}} a_{i,j}$
4:     Set the corresponding hash value $A_j = \min_{i\in\mathcal{E}} a_{i,j}$
5: end for
6: return $S$ and $A$
```

4.2.2 *Intrinsic Connection to Consistent Weighted Sampling with Integer Vectors*

We now discuss its intrinsic connection to the original consistent weighted sampling method with integer vector inputs [20], and show that our method is indeed a generalization of the original method to real vectors inputs.

The original consistent weighted sampling method [20] uses a random hash function h_j to generate the hash values $h_j(i, f)$ for each (i, f), where $i \in \mathcal{E}$ and $f \in \{1, 2, ..., V_i\}$, and then returns $(i^*_j, f^*_j) = \text{argmin}_{i\in\mathcal{E}, f\in\{1,2,...,V_i\}} h_j(i, f)$ as one sample (i.e., one sketch element S_j). As suggested by [11], the sketches can be simplified by keeping only i^*_j and discarding f^*_j. By defining $\hat{h}_j(i) = \min_{f=1,2,...,V_i} h_j(i, f)$, one sample is then returned as $i^*_j = \text{argmin}_{i\in\mathcal{E}} \hat{h}_j(i)$.

We now derive a new method to directly compute $\hat{h}_j(i)$ without involving f. The key idea is to directly compute $\hat{h}_j(i)$ from only one random hash value and V_i, rather than V_i random hash values. Specifically, since $h_j \sim Uniform(0, 1)$, we compute the CDF of $\hat{h}_j(i)$ as:

$$Pr[\hat{h}_j(i) \le p] = 1 - (1 - p)^{V_i} \tag{22}$$

for $p \in (0, 1)$. We assume a random uniformly distributed variable $\beta \sim Uniform(0, 1)$, and formulate its CDF $Pr[\beta \le q] = q$, $q \in (0, 1)$ as:

$$Pr[\beta \le q] = Pr[\beta \le 1 - (1 - p)^{V_i}] = 1 - (1 - p)^{V_i} \tag{23}$$

where $q = 1-(1-p)^{V_i}$ is an invertible continuous increasing function ($p = 1 - \sqrt[V_i]{(1 - q)}$) over $(0, 1)$. By applying the change-of-variable technique on Eq. 23, we obtain:

$$Pr[1 - \sqrt[V_i]{(1 - \beta)} \le p] = 1 - (1 - p)^{V_i} \tag{24}$$

Since Eqs. 24 and 22 show the same CDF, $\hat{h}_j(i)$ can be obtained by $\hat{h}_j(i) = 1 - \sqrt[V_i]{(1 - \beta)}$. As $(1 - \beta) \sim Uniform(0, 1)$, the computation can be simplified to $\hat{h}_j(i) = 1 - \sqrt[V_i]{\beta}$. As we only care about the ordering of those hash values, we can further simplify the computations via a monotonic transformation, and obtain $\hat{h}_j(i) = \frac{-\log \beta}{V_i}$. By replacing β with a random hash value $h_j(i)$ (h_j is a random hash function), we obtain:

$$\hat{h}_j(i) = \frac{-\log h_j(i)}{V_i} \tag{25}$$

which is the same as Eq. 21. In other words, $a_{i,j}$ is equal to $\hat{h}_j(i) = \min_{f=1,2,...,V_i} h_j(i, f)$ for integer $V \in \mathbb{N}^{|\mathcal{E}|}$. Therefore, our method can be regarded as a generalization of the original consistent weighted sampling method to measure the normalized min-max similarity with real $V \in \mathbb{R}^{|\mathcal{E}|}$.

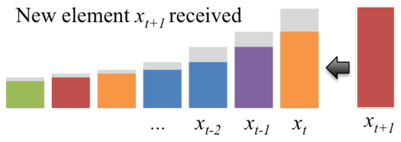


Fig. 3. Illustration of decreasing gradual forgetting weights (exponentially) when a new histogram element x_{t+1} is received (former weights are represented in gray)

4.3 Incremental Sketch Update

Incremental sketch update requires that a new sketch $S(t + 1)$ can be fast computed based on the former sketch $S(t)$ (with its corresponding hash values $A(t)$) and the incoming histogram element x_{t+1}. Specifically, when a new histogram element $x_{t+1} = i'$ is received from the data stream, the gradual forgetting weights of all existing elements of the histogram evolve by a factor of $e^{-\lambda}$ (as shown in Fig. 3), which results in a uniform scaling of V. One of the key properties of our sketch is that it is invariant under uniform scaling of V, which allows us to perform the scaling by quickly adjusting A only. Afterwards, as only the discriminative weight of histogram element i' is updated, we only need to recompute $V_{i'}$, and then its corresponding hash value $a_{i',j}$. Finally, the new sketch $S(t + 1)$ can be computed based on the adjusted sketch $S(t)$ (with $A(t)$) and the incoming histogram element $x_{t+1} = i'$ (with $a_{i',j}$). In the following, we first present the uniform scaling invariance property of our sketch, and then the incremental update process.

Proposition 3. *If sketch S (with its corresponding hash values A) is created for V using Alg. 1, then for any positive constant γ, S remains the sketch for γV with the corresponding hash values $\frac{1}{\gamma}A$.*

Proof. Alg. 1 computes a sketch element S'_j for γV as follows:

$$a'_{i,j} = \frac{-\log h_j(i)}{\gamma V_i} = \frac{1}{\gamma} a_{i,j} \tag{26}$$

$$S'_j = \underset{i\in\mathcal{E}}{\text{argmin}} \left(\frac{1}{\gamma} a_{i,j}\right) = \underset{i\in\mathcal{E}}{\text{argmin}}\, a_{i,j} = S_j \tag{27}$$

$$A'_j = \min_{i\in\mathcal{E}} \frac{1}{\gamma} a_{i,j} = \frac{1}{\gamma} \min_{i\in\mathcal{E}} a_{i,j} = \frac{1}{\gamma} A_j \tag{28}$$

This completes the proof. □

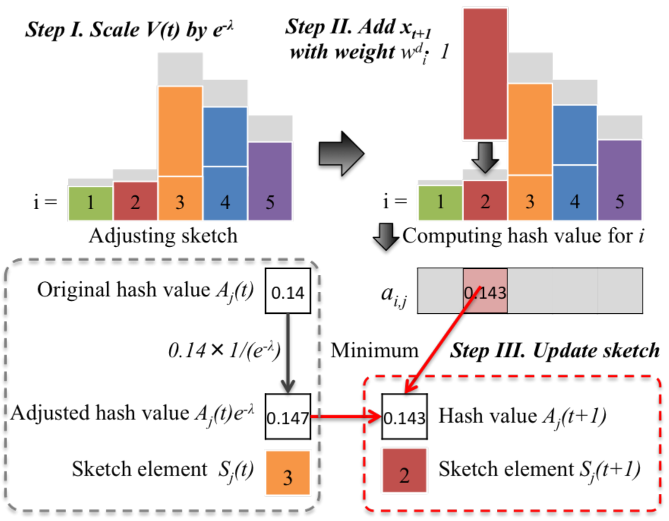

Step I. Scale V(t) by e^{λ}
Step II. Add x_{t+1} with weight $w_i^d \cdot 1$
i = 1 2 3 4 5
Adjusting sketch
Computing hash value for i
Original hash value $A_j(t)$
0.14
0.14 × 1/($e^{-\lambda}$)
Adjusted hash value $A_j(t)e^{\lambda}$
0.147
Sketch element $S_j(t)$
3
$a_{i,j}$
0.143
Minimum
Step III. Update sketch
Hash value $A_j(t+1)$
Sketch element $S_j(t+1)$
2

Fig. 4. An example of incrementally updating one sketch element. I). According to the scaling of the histogram, we keep the sketch invariant $S_j(t) = 3$, and adjust its hash value from $A_j(t) = 0.14$ to $A_j(t) \cdot e^{\lambda} = 0.147$ ($\lambda = 0.05$ in this example). II). By adding the incoming histogram element $x_{t+1} = 2$ ($2 \in \mathcal{E}$) with weight $w_i^d \cdot 1$ to the scaled histogram, we recompute only the hash value for $i = 2$, i.e., $a_{2,j} = 0.143$. III). By selecting the minimum hash value between $A_j(t) \cdot e^{\lambda} = 0.147$ and $a_{2,j} = 0.143$, we update $S_j(t+1) = 2$ and $A_j(t+1) = 0.143$.

Proposition 3 serves as a basis for our incremental sketch update process, which works as follows (for one sketch element S_j):

I. When a new histogram element $x_{t+1} = i'$ is received, we scale $V(t)$ by a factor of $e^{-\lambda}$, and adjust the sketch according to Proposition 3, i.e., $S_j(t)$ and $A_j(t) \cdot e^{\lambda}$.

II. We add the incoming histogram element i' to the scaled histogram. First, we update the discriminative weight w_i^d according to the method described in Section 3.1.1. Then, as the newest histogram element always has gradual forgetting weight 1 ($w_t^g = e^{-\lambda(t_n - t_n)} = 1$), we add the incoming histogram element i' with the overall weight $w_i^d \cdot 1$ to the scaled histogram: $V_{i'}(t+1) = V_{i'}(t) \cdot e^{-\lambda} + w_i^d \cdot 1$ if $i' \in \mathcal{E}$. In case $i' \notin \mathcal{E}$, we add i' to $\mathcal{E}$ and expand V to include $V_{i'}(t+1) = w_i^d \cdot 1$. Afterwards, we recompute only the hash value for i', i.e., $a_{i',j}$.

III. By comparing the new hash value $a_{i',j}$ with the adjusted hash value $A_j(t) \cdot e^{\lambda}$, we update the sketch $S_j(t+1) = i'$ and $A_j(t+1) = a_{i',j}$ if $a_{i',j} < A_j(t) \cdot e^{\lambda}$, $S_j(t+1) = S_j(t)$ and $A_j(t+1) = A_j(t) \cdot e^{\lambda}$ otherwise.

Fig. 4 illustrates the incremental sketch update process following the previous example shown in Fig. 2.

4.4 Implementation Details

Our incremental sketch update process requires to access the former histogram $V(t)$ to compute $V(t+1)$. To maintain such a streaming histogram V of an ever-increasing size, we propose an extended count-min sketch model.

The classical count-min sketch [25] is a fixed-sized probabilistic data structure Q (d rows and g columns) serving as a frequency table of streaming elements. It uses d independent random hash functions h_l ($l = 1, 2, ..., d$) to map streaming elements onto a range of $1, 2, ..., g$ (counters). Every time a new element i is received, for each row l, its hash function h_l is applied to i to determine a corresponding column $h_l(i)$, and then the counter $Q_{l,h_l(i)}$ is increased by 1. To get the estimated frequency at time t, the corresponding hash function is applied to i to look up the corresponding counter for each row. The estimate is then returned as the minimum of all the probed counters across all rows, i.e., $V_i(t) = \min_l Q_{l,h_l(i)}$. The estimated frequency error is guaranteed [38] to be at most $\frac{2}{g}$ with probability $1 - (\frac{1}{2})^d$.

In our case of streaming histogram with both discriminative and gradual forgetting weights, the cumulative frequency (weights) of all historical histogram elements is scaled with a factor $e^{-\lambda}$ (i.e., $V(t) \cdot e^{-\lambda}$) every time a new element is received from the data streams. Therefore, we extend the above count-min sketch method to consider such decay weights as follows. First, before adding a new element i from the data stream, we uniformly scale all counters across all rows by that factor, i.e., $Q_{l,h_l(i)}(t) \cdot e^{-\lambda}$. In such a way, for any histogram element i, its estimated weighted cumulative count $V_i(t)$ is also scaled to $\min_l Q_{l,h_l(i)}(t) \cdot e^{-\lambda} = V_i(t) \cdot e^{-\lambda}$, which corresponds exactly to Step I in our sketch update process. Afterwards, we add the new element with its corresponding weight $w_i^d \cdot 1$, i.e., $Q_{l,h_l(i)}(t+1) = V_i(t) \cdot e^{-\lambda} + w_i^d \cdot 1$, which corresponds to Step II in our sketch update process. As the aforementioned estimated frequency error of count-min sketch depends only on its structural parameter g and d, it is easy to see that the estimated error does not change under this straightforward extension, as a uniform scaling does not affect the data structure itself. For the detailed proof of the estimated error, please refer to [38]. In this study, we empirically set the default parameters $d = 10$, $g = 50$ to guarantee an error of at most 4% with probability 0.999 (see Section 5.2.4 for more details).

In addition, we also use a classical count-min sketch to store the label frequency vector F^l, which is used to compute the discriminative weights as described in Section 3.1.1. For the sake of simplicity, we keep the same parameters as the one we used for V.

4.5 Analysis of D²HistoSketch

4.5.1 Discussion on Error Bound

The original consistent weighed sampling technique [32] have proven the correctness of Eq. 6, with its associated error bound. However, this method takes only integer vectors as input, and requires a sketch element (i_j^*, f_j^*) which prohibits its application to streaming histograms.

To handle streaming histograms, our proposed method is based on the 0-bit consistent weighted sampling technique proposed by [11]. This technique is able to take real vectors as input, and more importantly, the author empirically proved Eq. 7, which serves as a fundamental element of our method. However, as mentioned by the author in [11], a rigorous proof of Eq. 7 turns out to be a difficult probability problem, which is not given in the original paper. In this study, we focus on designing a sketching method for streaming histograms with both discriminative and gradual forgetting weights, and we have rigorous proven the correctness of our method by showing its equivalence to the 0-bit consistent weighed sampling technique (using Proposition 1 and 2). Then, our method has the same error bound as the 0-bit consistent weighed sampling technique [11]. However, the further proof about the 0-bit consistent weighed sampling technique and its associated error bound remains a difficult problem, and is outside the scope of our paper.

4.5.2 *Time and Space Complexity Analysis*

Time. Since our sketches are incrementally maintained, we discuss the time complexity for each incoming histogram element from the data streams. Specifically, to update a sketch of length K, we perform our incremental sketch update process for all K sketch elements, which takes $O(K)$ time. In addition, we also need to retrieve/update the corresponding count-min sketches for both V and F^l (with d rows), which takes $O(d)$ time. The total time complexity for updating one histogram element is hence $O(K + d)$. Compared to our previous work HistoSketch [14], our proposed method with few in-memory parameters can compute sketches in a more efficient way, showing a higher processing speed with an improvement of 12.2% (see Section 5.3.4 for more detail).

Space. Each streaming histogram is represented by a sketch of length K taking $O(K)$ space. For the incremental sketch update purpose, we store the raw streaming histogram V in a count-min sketch data structure (d rows and g columns) taking $O(dg)$ space. Subsequently, the total space complexity is $O(K+dg)$ for one streaming histogram. Considering a set of n histograms with a set of L labels (the label frequency vector F^l stored in a count-min sketch data structure with d rows and g columns), the total space complexity is $O(n \cdot (K + dg) + L \cdot (dg))$. Note that our space complexity is linear w.r.t. the number of histograms n, and more importantly, that it is independent of the ever-growing cardinality of streaming histogram elements $|\mathcal{E}|$. Compared to our previous work HistoSketch [14] that requires in-memory parameters $r_{i,j}, c_{i,j}, \beta_{i,j}$ and thus has a space complexity $O(n \cdot (K + dg) + L \cdot (dg) + K \cdot |\mathcal{E}|)$, our proposed memory-efficient method can dramatically reduce the memory consumption in practice, as $|\mathcal{E}|$ is usually large and growing over time. For example, in our experiments on the NYC dataset ($n = 3173, K = 100, L = 9, d = 10, g = 50$ and $|\mathcal{E}|$ is about 2 million), the memory consumption of our proposed method is only 1% of the memory required by our previous work HistoSketch.

5 Experimental Evaluation

In this section, we evaluate D^{2}HistoSketch on multiple classification tasks using both synthetic and real-world datasets. In the following, we first present our experimental setup, followed by the results on both types of datasets.

5.1 Experimental Setup

To evaluate the performance of our similarity-preserving sketches, we perform classification tasks based on these sketches in difference scenarios, which is a common evaluation scheme for similarity preserving sketching methods [4], [11], [13], [14]. Specifically, based on labeled streaming histograms, we try to classify those histogram instances without labels. We use a KNN classifier [39] which can always take the most up-to-date training data (sketches) for classification without maintaining a built classification model. Such a property fits our case of classifying streaming histograms with continuously incoming histogram elements, where the sketches are continuously updated accordingly. In addition, using a simple KNN classifier, we can put more focus on the distance measures rather than classifiers. We empirically set KNN to consider the five nearest neighbors. We consider the following evaluation scenarios:

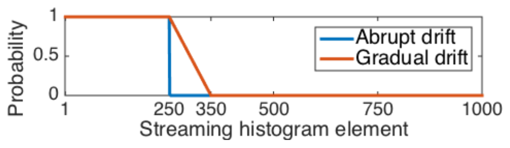


Fig. 5. Probability of streaming histogram elements generated from its initial distribution for the synthetic dataset.

Synthetic Dataset. Synthetic data is widely used in studying concept drift adaptation [19], [40]. The advantage is that we can simulate different cases of concept drift in streaming histograms with controllable parameters. Typical methods of simulating data streams with concept drift often use a moving hyperplane to generate a stream of complete data instances [40]. However, it cannot be directly adopted for streaming histograms, as the elements of a histogram are observed in a streaming manner. Therefore, we design our own data simulation method. Specifically, we consider two Gaussian distributions $\mathcal{N}(100, 20)$ and $\mathcal{N}(110, 20)$ representing two classes of histograms, respectively. The streaming histogram elements are then generated as the nearest integers of the random numbers sampled from those distributions. For each class, we simulate 500 histograms with 1000 elements each. We then split the 500 histograms to 50%-50% for training and the testing, respectively. The histogram elements are generated in a random order. To simulate concept-drift issues, we consider both abrupt and gradual drift cases [19] in testing data.

- For abrupt drift, starting from 25% of streaming histogram elements, the testing data of one distribution abruptly starts to receive the histogram elements generated from the other distribution, and also changes their labels immediately.
- For gradual drift, from 25% to 35% of streaming histogram elements, the testing data of one distribution gradually starts to receive the histogram elements generated from the other distribution with an increasing probability (from 0 to 1). The labels of the testing data also change gradually from one class to the other, i.e., from 0% to 100%.

Fig. 5 shows the probability of streaming histogram elements generated from its initial distributions. Note that it is complementary to the probability of histogram elements generated from the other distribution.

POI Dataset. Our sketching method can be applied to solve the problem of semantic place labeling [4], where we want to infer a place's category (e.g., supermarket or bar) based on its customers' visiting patterns (i.e., the streaming histogram of its customers' visits). The basic intuition is that POIs of the different type usually have different temporal visiting patterns, e.g., bars are mostly visited during the night while museums are often visited during the daytime. Previous studies have shown that considering user-time pairs as histogram elements (i.e., fine-grained visiting

TABLE 1
POI dataset statistics

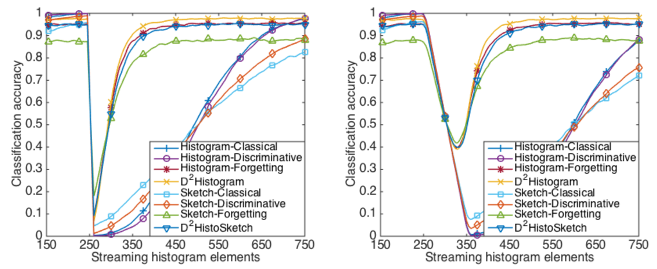
Dataset	New York City (NYC)	Tokyo (TKY)	Istanbul (IST)
Number of check-ins	142,495	494,702	292,771
Number of POIs	3,174	2,993	3,120
Number of users	12,798	9,160	15,479

patterns) yields much higher accuracy than considering only time (i.e., coarse-grained visiting patterns) [4]. We thus consider fine-grained patterns in the following. Specifically, for one user's visit to a POI, we first map the visiting time onto one of the 168 hours in a week period (discretization of time), and then consider the user-time pair as a histogram element. With a large and continuously increasing number of users over time, the cardinality of the streaming histogram rapidly increases. More importantly, the visiting pattern of a POI may change both abruptly (e.g., caused by the change of POI type) and gradually (e.g., caused by the introduction of new menu items in a restaurant).

To evaluate our method using this task, we use a dataset from Foursquare provided by [3], [23], which has been widely used to study the problems of semantic place labeling [4], [14] and POI recommendation [41], [42], [43]. The dataset contains user check-in data on POIs for about two years (from April 2012 to March 2014). Each check-in records one visit of a user to a POI (with the associated category) at a certain time. We randomly select 20% of the POIs as unlabeled testing data and regard the rest as training data. The classification is performed at the end of each month on the second year (in order to avoid too few check-ins for some POIs during the first year). The categories (labels) of POIs in the dataset are classified by Foursquare into 9 root categories (i.e., Arts & Entertainment, College & University, Food, Great Outdoors, Nightlife Spot, Professional & Other Places, Residence, Shop & Service, Travel & Transport), which are further classified into 291 sub-categories. Without loss of generality, we select three big cities, New York City, Tokyo and Istanbul, for our experiments. Table 1 summarizes the main characteristics of our dataset.

Movielens Dataset. Our sketching method can also be applied to predict movie genre based on users' tagging activity streams. Specifically, on a movie recommendation platform, a user can add tags to a movie. Subsequently, we characterize each movie using the histogram of user tagging activities, where each user is a histogram element. Similar to the case of POI dataset, we are then able to classify a movie in to different genres based on its streaming histogram.

We use the MovieLens 20M Dataset provided by [44]. This dataset contains movie tagging activity about 10 years. We select top 8 primary genres (i.e., "action", "adventure", "animation", "children", "comedy", "Crime", "documentary", "Drama"), and keep only the tagging data on the related movies. Finally, we obtain 24,394 movies, 7,622 users, 434,054 tagging activities. We randomly select 20% of the movies as unlabeled testing data and regard the rest as training data. The classification is performed at the end of 6th to 10th years.



(a) Abrupt drift (b) Gradual drift

Fig. 6. Performance on classification task.

5.2 Performance on Synthetic Dataset

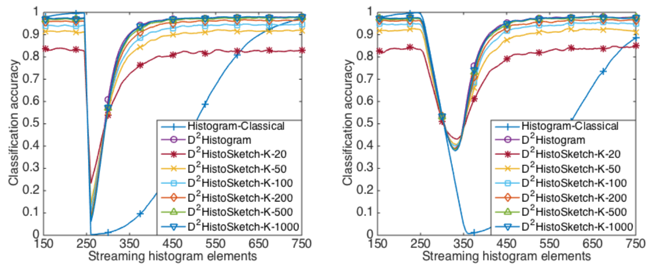
In this section, we first show the effectiveness of D^{2}HistoSketch by comparing the classification performance using different streaming histograms and their corresponding sketches. Afterwards, we study the impact of the sketch length K, the weight decay factor λ, and the count-min sketch parameters in different scenarios.

5.2.1 Performance on classification task

To demonstrate the advantages of considering discriminative and gradual forgetting weights in streaming histograms and the similarity-preserving sketches, we compare the following four methods that keep the full histograms in memory, and also their corresponding sketching methods. We set sketch length $K = 100$ for all the sketching methods, use entropy weighting for discriminative weights, and set $\lambda = 0.02$ for gradual forgetting weights when applicable.

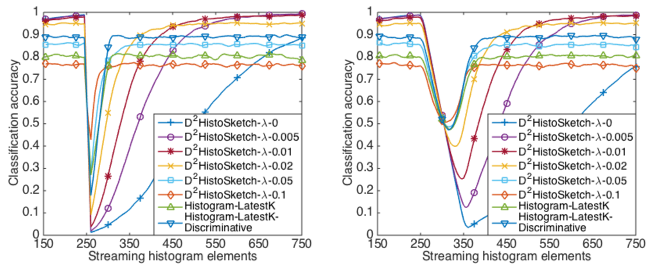
- ***Histogram-Classical***: full histograms where the histogram's elements are unweighted.
- ***Histogram-Discriminative***: full histograms where the histogram's elements are assigned with discriminative weights.
- ***Histogram-Forgetting***: full histograms where the histogram's elements are assigned with gradual forgetting weights.
- ***D^{2}Histogram***: full histograms where the histogram's elements are assigned with both discriminative and gradual forgetting weights.
- ***Sketch-Classical***: our sketching method configured with unweighted histogram elements (approximation of *Histogram-Classical*).
- ***Sketch-Discriminative***: the sketching method with only discriminative weights (approximation of *Histogram-Discriminative*), which is equivalent to POISketch [4].
- ***Sketch-Forgetting***: the sketching method with only gradual forgetting weights (approximation of *Histogram-Forgetting*), which is equivalent to HistoSketch [14].
- ***D^{2}HistoSketch***: our proposed sketching method with both discriminative and gradual forgetting weights (approximation of *D^{2}Histogram*).

Fig. 6 shows the classification accuracy over time (number of streaming elements per histogram) for both abrupt and gradual drifts. First, comparing the accuracy of using full histograms, we observe that our D^{2}Histogram achieves the best results overall. On one hand, considering gradual forgetting weights can significantly reduce the recovery time from concept drift. For example, in the case of abrupt



(a) Abrupt drift (b) Gradual drift

Fig. 7. Impact of sketch length K.



(a) Abrupt drift (b) Gradual drift

Fig. 8. Impact of weight decay factor λ.

drift (starting from 250th histogram elements), accuracy recovers after receiving 450 and 150 elements for Histogram-Discriminative and D^{2}Histogram, respectively. On the other hand, discriminative weights can effectively improve accuracy outside of the period for concept drift adaptation (i.e., before concept drift and after recovery from concept drift). For example, for both cases of abrupt and gradual drifts, D^{2}Histogram shows a 2.6% improvement of accuracy from Histogram-Forgetting. Second, among all the sketching methods, the proposed D^{2}HistoSketch achieves the best results, as it can efficiently approximate the similarity between D^{2}Histogram with both discriminative and gradual forgetting weights. Specifically, D^{2}HistoSketch quickly adapts to concept drift (showing similar adaptation speed as that of D^{2}Histogram), and shows a significant accuracy improvement of 7.9% compared to Sketch-Forgetting.

5.2.2 *Impact of sketch length K*

The sketch length K influences how well the sketch can approximate the similarity with the original data. In this experiment, by fixing the gradual forgetting weight decay factor $\lambda = 0.02$, we vary the sketch length K within $[20, 50, 100, 200, 500, 1000]$ to investigate the performance of our method. We also plot the results from ***Histogram-Classical*** and ***D^{2}Histogram*** as references.

Fig. 7 shows the classification accuracy over time in both cases of abrupt and gradual drift. First, we observe a positive impact of sketch length K on the classification accuracy, i.e., larger values of K imply a higher accuracy, as longer sketches can preserve more information and thus better approximate the similarities of the D^{2}Histogram. The accuracy flattens out after $K = 500$ (D^{2}HistoSketch with $K \geq 500$ are highly overlapped with D^{2}Histogram), indicating that a sketch of length 500 is sufficient for accurate similarity approximation. Second, we find that the sketch length K has no obvious impact on the adaptation speed, which is actually controlled by the gradual forgetting weight decay factor λ (we will discuss this in the next experiment).

5.2.3 *Impact of weight decay factor λ*

The weight decay factor λ balances the trade-off between the concept drift adaptation speed and the similarity approximation performance. In this experiment, by fixing the sketch length $K = 100$, we vary the weight decay factor λ within $[0, 0.005, 0.01, 0.02, 0.05, 0.1]$ to investigate the performance of our method. We also compare our method with the following two methods:

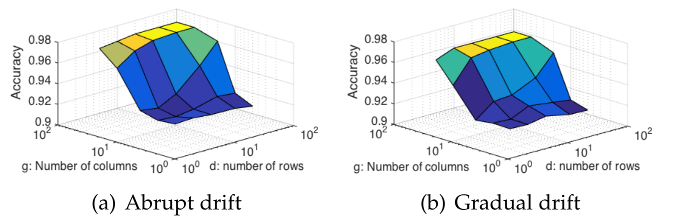
- ***Histogram-LatestK*** where the histogram is built with the latest K histogram elements from the stream, which is a typical method for abrupt forgetting (i.e., sliding window based concept-drift adaptation) [17]. It can also be regarded as a sketching method in the sense that the latest K histogram elements (unweighted) are the sketches to represent the histogram.
- ***Histogram-LatestK-Discriminative*** where we further incorporate discriminative weights (i.e., entropy weights as for our method) into *Histogram-LatestK*.

Fig. 8 shows the classification accuracy over time in both cases of abrupt and gradual drift. First, by comparing the results of different weight decay factors, we observe clearly the trade-off between the concept drift adaptation speed and classification accuracy. On one hand, larger λ values imply faster adaptation to concept drift, as the algorithm quickly forgets outdated data (i.e., it puts lower weights on the outdated data). On the other hand, larger values of λ lead to lower classification accuracy, as the algorithm uses less information from former histogram elements for sketching, which leads to worse similarity approximation. Second, compared to the two additional baselines, we find that our D^{2}HistoSketch-λ-0.05 outperforms Histogram-LatestK by achieving higher accuracy and faster adaptation speed at the same time. However, we found that Histogram-LatestK-Discriminative shows comparable results to our method, i.e., its adaptation speed is faster than D^{2}HistoSketch-λ-0.02 and slower than D^{2}HistoSketch-λ-0.05 while its accuracy is lower than D^{2}HistoSketch-λ-0.02 and higher than D^{2}HistoSketch-λ-0.05. Despite such similar results, there are two obvious advantages of our methods: 1) D^{2}HistoSketch is able to balance the adaptation speed and the accuracy under fixed-size sketches while Histogram-LatestK needs to vary sketch length K to tune such trade-off; 2) our method is much faster in similarity computation than Histogram-LatestK-Discriminative, as the latter requires set operations while D^{2}HistoSketch only relies on Hamming distance. For example, to classify one histogram using our testing PC[1], our method needs only 13ms while Histogram-LatestK takes 152ms, which shows a 12x speedup.

5.2.4 *Impact of count-min sketch parameters*

In this experiment, we study the impact of the count-min sketch parameters on the classification performance. Specifically, we study the classification accuracy of our method by varying the number of rows d and the number of column g within $[5, 10, 20, 50, 100]$.

1. Intel Core i7-4770HQ@2.20GHz, 16GB RAM, Mac OS X, implementation using MATLAB v2014b.



(a) Abrupt drift (b) Gradual drift

Fig. 9. Impact of count-min sketch parameters.

Fig. 9 shows the average classification accuracy outside of the period for concept drift adaptation. We observe that larger d and g can both lead to a higher accuracy, as they can better approximate the frequency table of histogram V and the label frequency vector F^l. However, larger d and g will also lead to higher time and space complexity as discussed in Section 4.5.2. Therefore, as the accuracy flattens out after $d = 10$, $g = 50$, we empirically set the count-min sketch parameters $d = 10$, $g = 50$ for all the experiments.

5.3 Performance on Real-World Datasets

To evaluate our method using real-world datasets, we first compare it with state-of-the-art approaches, and then show its classification accuracy over time, followed by its runtime performance.

5.3.1 *Comparison with other methods*

We compare D^{2}HistoSketch with all aforementioned baselines. The sketch length K is empirically set to 100 for all related methods. Fig. 10 plots the average classification accuracy. First, we observe that D^{2}Histogram yields the highest accuracy, showing the effectiveness of considering both discriminative and gradual forgetting weights on streaming histogram elements. Third, our D^{2}HistoSketch also outperforms other sketching methods by efficiently approximating similarity between D^{2}Histogram. In particular, D^{2}HistoSketch can efficiently approximate the similarity with only a small loss of classification accuracy (e.g., about 3.25% for root POI categories on the NYC dataset) compared to D^{2}Histogram. Finally, D^{2}HistoSketch also outperforms Histogram-LatestK/Histogram-LatestK-Discriminative, showing the effectiveness of gradual (rather than abrupt) forgetting of historical data.

In addition, comparing the results between the POI dataset and the Movielens dataset, we find that the improvement by considering gradual forgetting (either in histograms and sketches) on Movielens dataset is smaller than on POI dataset. It is due to the fact that concept-drift is rarely observed in the Movielens dataset. In other words, compared to POIs, a movie rarely changes its genre.

5.3.2 *Classification accuracy over time*

In this experiment, we study the classification accuracy of our sketching method over time. We focus on the NYC dataset with the 9 root levels of POI categories (Experiments on the other datasets show similar results). In addition to the randomly selection strategies of testing POIs, we further consider only the POIs with category changes as testing data, as the change of POI categories will likely lead to concept drift (particularly of the abrupt kind). We keep the same parameter setting as in the previous experiment.

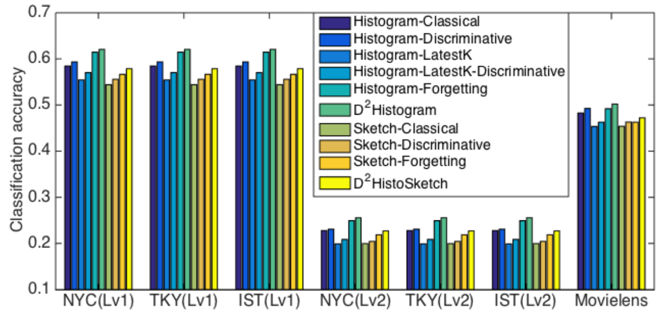
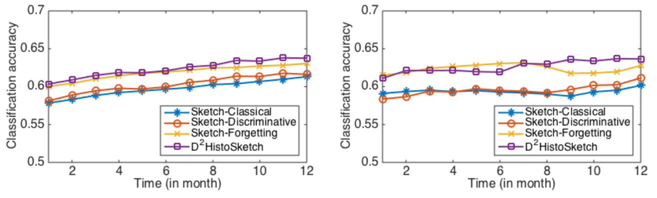


Fig. 10. Comparison with other methods



(a) Random testing POIs (b) Category-changing POIs

Fig. 11. Classification performance over time

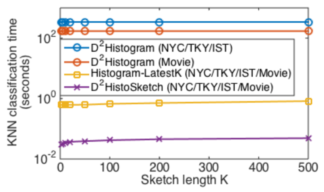
Fig. 11(a) shows the classification accuracy for each of the 12 testing months. We observe that the accuracy of all sketching methods slightly increases over time with the accumulated histogram elements, as observing more histogram elements leads to more accurate similarity measurement of streaming histograms. In particular, compared to other sketching methods, our D^{2}HistoSketch achieves consistently higher accuracy by considering both discriminative and gradual forgetting weights.

Fig. 11(b) shows the same results on the testing POIs with category changes only. We observe a larger improvement of considering gradual forgetting weights than that in Fig. 11(a). For example, the accuracy gain of our method over Sketch-Discriminative is 3.03% when testing on category-changing POIs, while it is 2.13% when testing on random POIs. This observation further shows the effectiveness of our method at handling concept drift. Note that as opposed to the synthetic dataset, we do not observe any sudden drop of accuracy, as sets of POIs rarely change their types simultaneously.

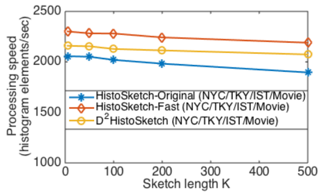
5.3.3 *Runtime performance for classification*

In this experiment, we investigate the runtime performance of sketch-based classification. More precisely, we evaluate the classification time w.r.t. sketch length K. We also compare with Histogram-LatestK which can be regarded as a sketching method as it keeps only the latest K histogram elements.

Fig. 12(a) plots the KNN classification time (log scale) on our test PC[3]. We observe that compared to the full histograms (D^{2}Histogram), using D^{2}HistoSketch dramatically reduces the classification time (with a 7500x speedup), since the Hamming distance between sketches of size K (e.g., K=100 in previous experiments) can be much more efficiently computed than the normalized min-max similarity between the full histograms of much larger size $|\mathcal{E}|$ (e.g., $|\mathcal{E}|$ is about 2 million for the NYC dataset). We also note that all the sketching methods have similar classification time, as they all compute the Hamming distance between vectors



(a) Classification time



(b) Processing speed

Fig. 12. Runtime performance

of size K. Compared to Histogram-LatestK, our method also shows a 15x speedup, as Histogram-LatestK requires set operations for similarity computation.

5.3.4 Runtime performance for sketch maintenance

In this experiment, we investigate the runtime performance of sketch maintenance. We evaluate the streaming histogram processing speed w.r.t. sketch length K (as the time complexity of maintaining D^{2}HistoSketch mainly depends on the sketch length K). Here we compare our method with the following two methods: 1) ***HistoSketch-Original*** [14] (considering gradual forgetting weights only) whose sampling process is implemented with Eq 10 and 9 and requires the large set of in-memory parameters $r_{i,j}, c_{i,j}, \beta_{i,j}$; 2) ***HistoSketch-Fast*** which is implemented with our efficient sampling method as show in Eq. 21 and considering gradual forgetting weights only (for a fair comparison with HistoSketch-Original).

Fig. 12(b) shows the processing speed of streaming histogram elements. First, compared to HistoSketch-Original, HistoSketch-Fast shows a higher processing speed (with a 12.2% improvement over HistoSketch-Original). Note that these two methods are equivalent in preserving similarities of streaming histograms with gradual forgetting weight only, and thus show exactly the same classification performance. Therefore, our proposed sampling method is more runtime-efficient than the sampling method in HistoSketch-Original. Second, by adding the discriminative weights, D^{2}HistoSketch yields a slightly slower processing speed than HistoSketch-Fast, but with higher classification performance. Finally, compared to HistoSketch-Original, D^{2}HistoSketch can achieve a faster processing speed (with a 5.9% improvement on average) and a higher classification accuracy (with a 2.78% improvement on average) at the same time.

We also find that the processing speed slightly decreases with increasing sketch lengths for all the methods, as longer sketches required a little more time to update. Moreover, we believe that the processing speed of our method (about 2200 per second) is able to handle most real-world use cases. For example, Foursquare check-in streams achieved a peak-day record of 7 million check-ins/day in 2015 (about 81 check-ins/sec on average). In addition, our method can be easily parallelized w.r.t. the number of histograms (i.e., number of POIs), as sketches of streaming histograms are independently maintained from each other.

6 Conclusion and Future Work

This paper introduces D^{2}HistoSketch, a similarity-preserving sketching method for streaming histograms to efficiently approximate their *Discriminative* and *Dynamic* similarity. D^{2}HistoSketch is designed to fast and memory-efficiently maintain a set of compact and fixed-sized sketches of streaming histograms to approximate their normalized min-max similarity. To provide high-quality similarity approximations, D^{2}HistoSketch considers both discriminative and gradual forgetting weights for similarity measurement, and seamlessly incorporates them in the sketches. Based on both synthetic and real-world datasets, our empirical evaluation shows that our method is able to efficiently and effectively approximate the similarity between streaming histograms while outperforming state-of-the-art sketching methods. Compared to full streaming histograms with both discriminative and gradual forgetting weights in particular, D^{2}HistoSketch is able to dramatically reduce the classification time (with a 7500x speedup) at the expense of a small loss in accuracy only (about 3.25%).

As future work, we will explore the problem of sketching two-dimensional (bivariate) streaming histograms, and apply our method to other application domains.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 683253/GraphInt), and Fudan University Startup Research Grant (SXH2301005).

References

[1] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support vector machines for histogram-based image classification," *IEEE Trans. on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.
[2] L. D. Baker and A. K. McCallum, "Distributional clustering of words for text classification," in *Proc. of SIGIR*, 1998, pp. 96–103.
[3] D. Yang, D. Zhang, L. Chen, and B. Qu, "Nationtelescope: Monitoring and visualizing large-scale collective behavior in lbsns," *Journal of Network and Computer Applications*, vol. 55, pp. 170–180, 2015.
[4] D. Yang, B. Li, and P. Cudré-Mauroux, "Poisketch: Semantic place labeling over user activity streams," in *Proc. of IJCAI*, 2016, pp. 2697–2703.
[5] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 769–790, 2018.
[6] C. C. Aggarwal and P. S. Yu, "On classification of high-cardinality data streams," in *Proc. of SDM*, 2010, pp. 802–813.
[7] Y. Bachrach, E. Porat, and J. S. Rosenschein, "Sketching techniques for collaborative filtering," in *Proc. of IJCAI*, 2009, pp. 2016–2021.
[8] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," in *Proc. of STOC*, 1998, pp. 327–336.
[9] M. Mitzenmacher, R. Pagh, and N. Pham, "Efficient estimation for high similarities using odd sketches," in *Proc. of WWW*, 2014, pp. 109–118.
[10] K. Kutzkov, M. Ahmed, and S. Nikitaki, "Weighted similarity estimation in data streams," in *Proc. of CIKM*, 2015, pp. 1051–1060.
[11] P. Li, "0-bit consistent weighted sampling," in *Proc. of KDD*, 2015, pp. 665–674.
[12] S. Ioffe, "Improved consistent sampling, weighted minhash and l1 sketching," in *Proc. of ICDM*, 2010, pp. 246–255.
[13] W. Wu, B. Li, L. Chen, and C. Zhang, "Consistent weighted sampling made more practical," in *Proc. of WWW*, 2017, pp. 1035–1043.
[14] D. Yang, B. Li, L. Rettig, and P. Cudré-Mauroux, "Histosketch: Fast similarity-preserving sketching of streaming histograms with concept drift," in *Proc. of ICDM*, New Orleans, USA, 2017.

[15] Y. Yang, F. Liang, N. Jojic, S. Yan, J. Feng, and T. S. Huang, "Discriminative similarity for clustering and semi-supervised learning," *arXiv:1709.01231 [stat.ML]*, 2017. [Online]. Available: https://arxiv.org/abs/1709.01231

[16] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artif. Intell. Rev.*, vol. 11, no. 1-5, pp. 273–314, 1997.

[17] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, p. 44, 2014.

[18] I. Koychev, "Gradual forgetting for adaptation to concept drift." Proc. of ECAI Workshop, 2000, pp. 101–107.

[19] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis*, vol. 8, no. 3, pp. 281–300, 2004.

[20] M. Manasse, F. McSherry, and K. Talwar, "Consistent weighted sampling," *Technical Report MSR-TR-2010-73*, 2010.

[21] M. O. Ward, G. Grinstein, and D. Keim, *Interactive data visualization: foundations, techniques, and applications*. CRC Press, 2010.

[22] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 129–142, 2015.

[23] D. Yang, D. Zhang, and B. Qu, "Participatory cultural mapping based on collective behavior data in location-based social networks," *ACM Trans. on Intelligent Systems and Technology*, vol. 7, no. 3, p. 30, 2016.

[24] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[25] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[26] Y. Ben-Haim and E. Tom-Tov, "A streaming parallel decision tree algorithm," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 849–872, 2010.

[27] B. Li, X. Zhu, L. Chi, and C. Zhang, "Nested subtree hash kernels for large-scale graph classification over streams," in *Proc. of ICDM*, 2012, pp. 399–408.

[28] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Proc. of VLDB*, vol. 99, no. 6, 1999, pp. 518–529.

[29] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. Shen, "Binary generative adversarial networks for image retrieval," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 394–401.

[30] J. Song, H. Zhang, X. Li, L. Gao, M. Wang, and R. Hong, "Self-supervised video hashing with hierarchical binary auto-encoder," *IEEE Transactions on Image Processing*, vol. 27, no. 7, pp. 3210–3221, 2018.

[31] J. Song, L. Gao, L. Liu, X. Zhu, and N. Sebe, "Quantization-based hashing: a general framework for scalable image and video retrieval," *Pattern Recognition*, vol. 75, pp. 175–187, 2018.

[32] B. Haeupler, M. Manasse, and K. Talwar, "Consistent weighted sampling made fast, small, and easy," *arXiv:1410.4266 [cs.DS]*, 2014. [Online]. Available: https://arxiv.org/abs/1410.4266

[33] C. Yang, R. Duraiswami, and L. Davis, "Efficient mean-shift tracking via a new similarity measure," in *Proc. of CVPR*, vol. 1. IEEE, 2005, pp. 176–183.

[34] P. Nakov, A. Popova, and P. Mateev, "Weight functions impact on lsa performance," *EuroConference RANLP*, pp. 187–193, 2001.

[35] A. Tsymbal, "The problem of concept drift: definitions and related work," *Technical Report TCD-CS-2004-15 Computer Science Department*, 2004.

[36] Y. Koren, "Collaborative filtering with temporal dynamics," *Communications of the ACM*, vol. 53, no. 4, pp. 89–97, 2010.

[37] R. L. Graham, *Concrete mathematics: a foundation for computer science*. Pearson Education India, 1994.

[38] G. Cormode and S. Muthukrishnan, "Approximating data with the count-min data structure," *IEEE Software*, 2012.

[39] T. M. Mitchell, "Machine learning." pp. I–XVII, 1997.

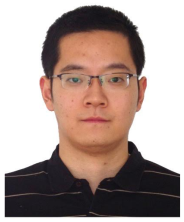
[40] H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining concept-drifting data streams using ensemble classifiers," in *Proc. of KDD*, 2003, pp. 226–235.

[41] D. Yang, D. Zhang, Z. Yu, and Z. Wang, "A sentiment-enhanced personalized location recommendation system," in *Proce. of HT*, 2013, pp. 119–128.

[42] D. Yang, D. Zhang, Z. Yu, and Z. Yu, "Fine-grained preference-aware location search leveraging crowdsourced digital footprints from lbsns," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*. ACM, 2013, pp. 479–488.

[43] Z. Yu, H. Xu, Z. Yang, and B. Guo, "Personalized travel package with multi-point-of-interest recommendation based on crowdsourced user footprints," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 151–158, 2016.

[44] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, p. 19, 2016.



Dingqi Yang is currently a senior researcher at the University of Fribourg in Switzerland. He received the Ph.D. degree in computer science from Pierre and Marie Curie University and Institut Mines-TELECOM/TELECOM SudParis, where he won both the CNRS SAMOVAR Doctorate Award and the Institut Mines-TELECOM Press Mention in 2015. His research interests include big social media data analytics, ubiquitous computing, and smart city applications.

Bin Li received the Ph.D. degree in computer science from Fudan University, Shanghai, China. He was a Senior Research Scientist with Data61 (formerly NICTA), CSIRO, Eveleigh, NSW, Australia and a Lecturer with the University of Technology Sydney, Broadway, NSW, Australia. He is currently an Associate Professor with the School of Computer Science, Fudan University, Shanghai, China. His current research interests include machine learning and data analytics, particularly in complex data representation, modeling, and prediction.

Laura Rettig is a Ph.D. student at the University of Fribourg, Switzerland under the supervision of Philippe Cudré-Mauroux. Her areas of interest are big data infrastructures, semantic data, entity linking, data streams, and deep learning. She received her Master's degree from the University of Fribourg, with her thesis written on big data streaming using real-world telecommunication data during an internship at Swisscom, for whch she won two best thesis awards at the University of Fribourg.

Philippe Cudre-Mauroux is a Full Professor and the director of the eXascale Infolab at the University of Fribourg in Switzerland. He received his Ph.D. from the Swiss Federal Institute of Technology EPFL, where he won both the Doctorate Award and the EPFL Press Mention. Before joining the University of Fribourg he worked on information management infrastructures for IBM Watson Research, Microsoft Research Asia, and MIT. His research interests are in next-generation, Big Data management infrastructures for non-relational data. Webpage: http://exascale.info/phil