HistoSketch: Fast Similarity-Preserving Sketching of Streaming Histograms with Concept Drift

Dingqi Yang*, Bin Li†, Laura Rettig*, Philippe Cudré-Mauroux*
*eXascale Infolab, University of Fribourg, Fribourg, Switzerland
Email: {firstname.lastname}@unifr.ch
†School of Computer Science, Fudan University, Shanghai, China
Email: libin@fudan.edu.cn

***Abstract*—Histogram-based similarity has been widely adopted in many machine learning tasks. However, measuring histogram similarity is a challenging task for streaming data, where the elements of a histogram are observed in a streaming manner. First, the ever-growing cardinality of histogram elements makes any similarity computation inefficient. Second, the concept-drift issue in the data streams also impairs the accurate assessment of the similarity. In this paper, we propose to overcome the above challenges with HistoSketch, a fast similarity-preserving sketching method for streaming histograms with concept drift. Specifically, HistoSketch is designed to incrementally maintain a set of compact and fixed-size sketches of streaming histograms to approximate similarity between the histograms, with the special consideration of gradually forgetting the outdated histogram elements. We evaluate HistoSketch on multiple classification tasks using both synthetic and real-world datasets. The results show that our method is able to efficiently approximate similarity for streaming histograms and quickly adapt to concept drift. Compared to full streaming histograms gradually forgetting the outdated histogram elements, HistoSketch is able to dramatically reduce the classification time (with a 7500x speedup) with only a modest loss in accuracy (about 3.5%).**

***Index Terms*—Similarity-Preserving Sketching, Histograms, Streaming Data, Concept Drift, Consistent Weighted Sampling**

I. 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 a 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], [2], document analysis [3], [4], and social network analysis [5].

Despite its importance in machine learning, computing histogram 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*. This is often the case 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, such as POI and activity recommendation [6], [7], semantic place labeling [8] or event detection [9]. 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 [10] have been intensively studied in stream data processing [11], [12]. Their key idea is to maintain a set of compact and fixed-size sketches for the original data while still preserving their similarity under a certain measure. In the current literature, most existing data sketching techniques [13], [14], [15], [16] 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).* Therefore, any sketching method for streaming histograms needs to be *incrementally updatable*, which departs from classical techniques that focus on sketching complete data instances. In other words, the new sketch of a streaming histogram should be incrementally computed from the former sketch and the newly arrived histogram element.

Moreover, as a common problem in data streams, concept drift also has to 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. Our collected POI dataset (see Section V-A) shows that 6.34% of POIs have changed their types (abrupt drift) in a period of two years. Considering concept drift issues can indeed improve the accuracy of histogram-based similarity techniques (see for example Section V-C1, where classification accuracy significantly increases when

2374-8486/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDM.2017.64

IEEE computer society

considering concept drift adaptation). 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]. In the case of streaming histograms, this means that a newer element of a histogram should have a higher weight than older ones when constructing the histogram. 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 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 HistoSketch, a similarity-preserving sketching method for streaming histograms with concept drift. HistoSketch is designed to efficiently and incrementally maintain a set of compact and fixed-sized sketches over streaming histograms to approximate similarity between the histograms with the special consideration of gradually forgetting outdated histogram elements. 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 [16]. To create a sketch from a histogram, we borrow the idea from consistent weighted sampling [20], which was originally proposed for approximating min-max similarity for complete data instances. To incrementally maintain the sketch when a new histogram element is observed, we first adjust the former sketch to seamlessly incorporate the weight decay factor for gradually forgetting old elements of the histogram, and then 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 piece of work considering the concept-drift issue for similarity-preserving sketching over streaming histograms. HistoSketch allows for fast maintenance of the sketches while gradually forgetting outdated data to ensure the robustness of our technique to concept drift.
- We formally prove both the correctness and scale-invariance of HistoSketch, and use those two properties to derive an incremental sketch updating technique that is efficient in both space and time.
- We empirically evaluate our method on multiple classification tasks using both simulated and real-world datasets. Our results show that HistoSketch is able to efficiently approximate similarity for streaming histograms and to quickly adapt to concept drift. Compared to full streaming histograms with gradual forgetting weights, HistoSketch is able to dramatically reduce the classification time (with a 7500x speedup) at the expense of a modest loss in accuracy (about 3.5%).

II. 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], [4], [5], [22]. 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 [23] and count-min sketch [24] and other online histogram building methods [25] 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 count-min sketch by addressing the problem of similarity-preserving sketching of data streams.

Similarity-preserving sketching [10] has been extensively studied to efficiently approximate the similarity of high dimensional data, such as documents, images and graph [26], [27]. 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 [13], [14], cosine [15], and min-max [20], [16], [28], [8], [29] similarities. These sketches can then enable many applications, particularly for information retrieval systems like image or document search engines [26]. However, most of these 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 concept-drift issues in similarity-preserving data sketching, to which we give specific consideration in this paper.

Concept drift is a common problem in streaming data processing and 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 [30]. 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 [31] 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

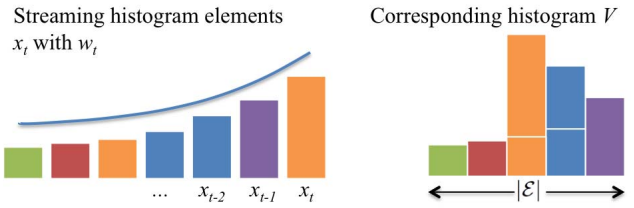


Fig. 1. Illustration of a streaming histogram with gradual forgetting (exponential decay weights). Left: The different histogram elements are assigned different colors, and their heights indicate the corresponding weights. Right: The same elements are accumulated to build the corresponding histogram.

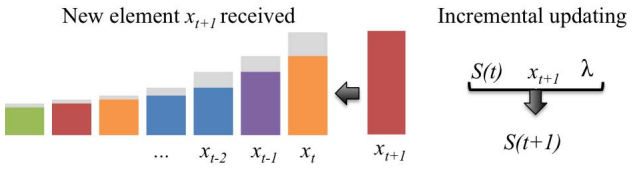


Fig. 2. Illustration of the incremental sketch update problem. Left: Weights decrease (exponentially) when a new histogram element x_{t+1} is received (former weights are represented in gray). Right: Incremental sketch update problem.

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 data sketching of streaming histograms.

III. Background and Problem Formulation

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. Due to its streaming nature, the cardinality $|\mathcal{E}|$ of a histogram 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.

To tackle the concept-drift issue in streaming histograms, we adopt the gradual forgetting approach when building V from x_t. Specifically, each streaming element x_t is associated with a weight w_t, which is inversely proportional to its age. To compute w_t, we adopt the exponential decay weight [19], which is computed as $w_t = e^{-\lambda(t_n - t)}$, 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., $V_i = \sum_t w_t \mathbb{1}_{x_t=i}$. Fig. 1 shows an example of streaming histograms with gradual forgetting (exponential decay weights).

To measure the similarity between such streaming histograms, we resort to normalized min-max similarity, which has been shown to be an effective measure for nonnegative data on many classification tasks over a sizable collection of public datasets [16]. Specifically, given two streaming histograms V^a and V^b, the min-max similarity is defined as follows:

$$Sim_{MM}(V^a, V^b) = \frac{\sum_{i \in \mathcal{E}} \min(V_i^a, V_i^b)}{\sum_{i \in \mathcal{E}} \max(V_i^a, V_i^b)} \tag{1}$$

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

$$\sum_{i \in \mathcal{E}} V_i^a = 1, \ \sum_{i \in \mathcal{E}} V_i^b = 1. \tag{2}$$

In that way, Eq. 1 becomes the normalized min-max similarity, denoted by Sim_{NMM}.

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 is exactly the normalized min-max similarity between V^a and V^b:

$$Pr[S_j^a = S_j^b] = Sim_{NMM}(V^a, V^b) \tag{3}$$

where $j = 1, 2, ..., K$. Then, the normalized min-max similarity between V^a and V^b can be approximated by the Hamming similarity [32] 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 similarity-preserving sketch S for the streaming histogram V with gradual forgetting weights, such that the new sketch $S(t+1)$ can be efficiently and incrementally computed based on the incoming histogram element x_{t+1}, the former sketch $S(t)$, and the weight decay factor λ. Fig. 2 illustrates the incremental sketch update problem.

IV. HistoSketch

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

A. 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) [20], [28], [16]. 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 $(i_j^*, f_j^*) = \text{argmin}_{i\in\mathcal{E}, f\in\{1,2,...,V_i\}} h_j(i,f)$ as one sample (i.e., one sketch element S_j). 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_j^{a*}, f_j^{a*}) and (i_j^{b*}, f_j^{b*}), which are generated from V^a and V^b, respectively, is proven to be exactly the min-max similarity of the two vectors [20]:

$$Pr[(i_j^{a*}, f_j^{a*}) = (i_j^{b*}, f_j^{b*})] = Sim_{MM}(V^a, V^b) \tag{4}$$

To improve the efficiency of the above method and allow real vectors as input, Ioffe [28] 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 $f \in \mathbb{N}$, $f \leq V_i$) 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 [16] further proposed to simplify the sketch by only keeping i_j^* rather than (i_j^*, f_j^*), and empirically proved the following property:

$$Pr[i_j^{a*} = i_j^{b*}] \approx Pr[(i_j^{a*}, f_j^{a*}) = (i_j^{b*}, f_j^{b*})] \tag{5}$$

A short description of the method proposed in [16] is presented in the following. To generate one sketch element S_j (sample i_j^*), the method first draws three random variables offline as 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

$$y_{i,j} = \exp\left(r_{i,j}\left(\lfloor\frac{\log V_i}{r_{i,j}} + \beta_{i,j}\rfloor - \beta_{i,j}\right)\right) \tag{6}$$

$$a_{i,j} = \frac{c_{i,j}}{y_{i,j}\exp(r_{i,j})} \tag{7}$$

The sketch element is then returned as $S_j = \text{argmin}_{i\in\mathcal{E}} a_{i,j}$. Please refer to [16], [28] for more details and for the proof of Eq. 4 and 5.

In this paper, we design an incrementally updatable sketching process to handle the streaming histograms with gradual forgetting weights, where $V \in \mathbb{R}_{>0}^{|\mathcal{E}|}$. Specifically, we propose a new approach to compute $y_{i,j}$, which has the following two highly desirable properties: 1). The generated $y_{i,j}$ follows the exact same distribution as the one generated using Eq. 6, which ensures the correctness of our sketching method (i.e., Eq. 4 and Eq. 5 still hold), and 2). The created sketch S is invariant under uniform scaling of V, which serves as a basis for incremental sketch update. In the following, we first present our sketch creation method, and then the incremental sketch update process.

B. Sketch Creation

Our sketch creation method borrows the idea of consistent weighted sampling with real number inputs [16]. Different from the original method, we propose a new approach to compute $y_{i,j}$. 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. 6) 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 [28]). However, as proved in [16], f_j^* can be ignored from the sketch (i_j^*, f_j^*) (i.e., Eq. 5). In such a case, it is only necessary to sample $y_{i,j}$ satisfying its distribution requirement. Therefore, we propose to compute $y_{i,j}$ as follows:

$$y_{i,j} = \exp(\log V_i - r_{i,j}\beta_{i,j}) \tag{8}$$

for which the following proposition holds.

Proposition 1. *Eq. 8 generates $y_{i,j}$ following the same distribution as generated by Eq. 6, 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. 6 can be derived as (we ignore the subscript (i,j) of r and β in the following proof):

$$\begin{aligned}\log y_{i,j} &= r\left(\lfloor\frac{\log V_i}{r} + \beta\rfloor - \beta\right)\\ &= r\left(\lfloor\frac{\log V_i}{r} + \beta\rfloor - \left(\frac{\log V_i}{r} + \beta\right) + \frac{\log V_i}{r}\right)\\ &= \log V_i - r\left(\left(\frac{\log V_i}{r} + \beta\right) - \lfloor\frac{\log V_i}{r} + \beta\rfloor\right)\end{aligned} \tag{9}$$

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* [33]. Since both V_i and r are known, this frac function can be considered as $\text{frac}(\beta + C)$, 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 $\text{frac}(\frac{\log V_i}{r} + \beta) \sim Uniform(0,1)$, which can be replaced by β. Therefore, we obtain:

$$\log y_{i,j} = \log V_i - r\beta \tag{10}$$

which is the same as in Eq. 8. Therefore, $y_{i,j}$ generated by Eq. 8 follows the same distribution as generated by Eq. 6.

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

$$\begin{aligned}F_Z(z) = P(Z < z) &= P\left(\log V_i - r\beta < z\right)\\ &= P\left(\frac{\log V_i - z}{r} < \beta\right)\end{aligned} \tag{11}$$

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

$$F_Z(z) = 1 - \frac{\log V_i - z}{r} = \frac{z - (\log V_i - r)}{\log V_i - (\log V_i - r)} \tag{12}$$

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

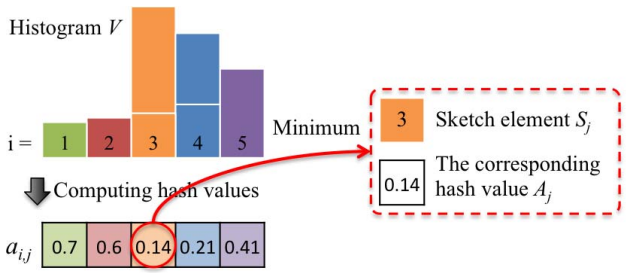


Fig. 3. Creating one sketch element from histogram V with cardinality $|\mathcal{E}| = 5$ ($i = 1, 2, ..., 5$). By computing the hash value $a_{i,j}$ for each i, we select the histogram element whose hash value is minimal as the sketch element and also keep its corresponding hash value, i.e., ($S_j =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

Algorithm 1 Sketch creation

Input: Histogram V, Sketch length K, Parameters r, c and β

Output: Sketch S and the corresponding hash values A

1: **for** j=1,2,...,K **do**
2: Compute $y_{i,j} = \exp(\log V_i + r_{i,j}\beta_{i,j})$
3: Compute $a_{i,j} = c_{i,j}/(y_{i,j}\exp(r_{i,j}))$
4: Set sketch element $S_j = \text{argmin}_{i\in\mathcal{E}}\, a_{i,j}$
5: Set the corresponding hash value $A_j = \min_{i\in\mathcal{E}} a_{i,j}$
6: **end for**
7: **return** S and A

Proposition 1 ensures the correctness of our sketching method (i.e., Eq. 4 and 5). In summary, to create sketch S from V, we first sample the following independent random variables offline as input parameters: $r_{i,j} \sim Gamma(2,1)$, $c_{i,j} \sim Gamma(2,1)$ and $\beta_{i,j} \sim Uniform(0,1)$ for $i \in \mathcal{E}$ and $j = 1, 2, ..., K$. We then use Alg. 1 for sketch creation. Note that we keep both sketch S and its corresponding hash values A (the latter will be used for incremental sketch update). Fig. 3 illustrates the sketch creation process for one sketch element S_j.

C. Incremental Sketch Update

Incremental sketch update requires that a new sketch $S(t+1)$ can be computed based on the former sketch $S(t)$ (with its corresponding hash values $A(t)$), the incoming histogram element x_{t+1}, and the gradual forgetting weight decay factor λ. Specifically, when a new histogram element is received from the data stream, the weights of all existing elements of the histogram evolve by a factor of $e^{-\lambda}$ (as shown in Fig. 2), which results in a uniform scaling of V. One of the two key properties of our sketch created by Alg. 1 is that it is invariant under uniform scaling of V, which allows us to perform the scaling by quickly adjusting A only. Afterwards, 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}. In the following, we first present the uniform scaling invariance property of our sketch, and then the incremental update process.

Proposition 2. *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 (we ignore the subscript (i,j) of r, β and c in the following proof):

$$y'_{i,j} = \exp(\log(\gamma V_i) + r\beta) = \gamma \exp(\log V_i + r\beta) = \gamma y_{i,j} \tag{13}$$

$$a'_{i,j} = \frac{c}{\gamma y_{i,j}\exp(r)} = \frac{1}{\gamma}\cdot\frac{c}{y_{i,j}\exp(r)} = \frac{1}{\gamma}a_{i,j} \tag{14}$$

$$S'_j = \underset{i\in\mathcal{E}}{\text{argmin}}\left(\frac{1}{\gamma}a_{i,j}\right) = \underset{i\in\mathcal{E}}{\text{argmin}}\, a_{i,j} = S_j \tag{15}$$

$$A'_j = \min_{i\in\mathcal{E}}\frac{1}{\gamma}a_{i,j} = \frac{1}{\gamma}\min_{i\in\mathcal{E}} a_{i,j} = \frac{1}{\gamma}A_j \tag{16}$$

This completes the proof. □

Proposition 2 implies that our sketching method actually approximates the normalized min-max similarity, as the sum-to-one normalization is indeed a uniform scaling. More importantly, it serves as a basis for our incremental sketch update process, which works as follows (for one sketch element S_j):

Step 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 2, i.e., $S_j(t)$ and $A_j(t)\cdot e^{\lambda}$.

Step II. We add the incoming histogram element i' to the scaled histogram[1], i.e., $V_{i'}(t+1) = V_{i'}(t)\cdot e^{-\lambda} + 1$ if $i' \in \mathcal{E}$. In case of $i' \notin \mathcal{E}$, we add i' to $\mathcal{E}$ and expand V to include $V_{i'}(t+1) = 1$. Afterwards, we need to recompute only the hash value for i', i.e., $a_{i',j}$.

Step 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 $S_j(t+1)$, $A_j(t+1)$ as follows:

$$S_j(t+1) = \begin{cases} i', \ if\ a_{i',j} < A_j(t)\cdot e^{\lambda} \\ S_j(t), \ otherwise \end{cases} \tag{17}$$

$$A_j(t+1) = \begin{cases} a_{i',j}, \ if\ a_{i',j} < A_j(t)\cdot e^{\lambda} \\ A_j(t)\cdot e^{\lambda}, \ otherwise \end{cases} \tag{18}$$

Fig. 4 illustrates the incremental sketch update process following the previous example shown in Fig. 3.

D. Implementation Details

Our incremental sketch update process requires to access the former histogram $V(t)$ in order 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. Specifically, the classical count-min sketch [24] is a fixed-sized probabilistic data structure Q (d rows and g columns) serving as a

[1]The newest histogram element has weight 1 as $w_t = e^{-\lambda(t_n - t)}$, where $t = t_n$ is the order of the latest histogram element received from the stream, and thus $w_t = e^0 = 1$.

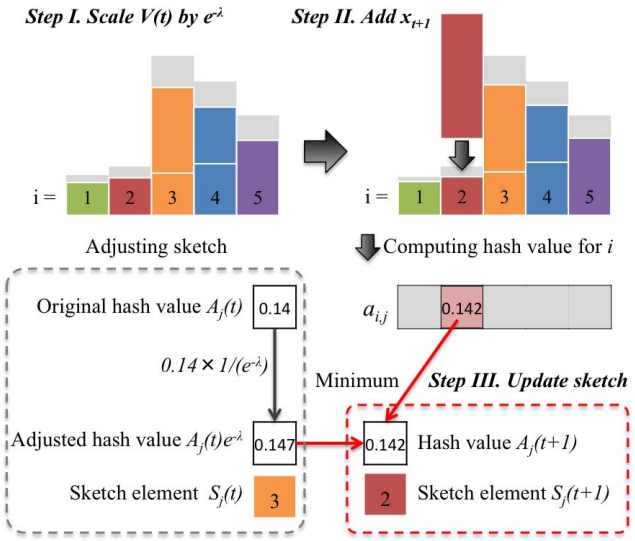


Fig. 4. An example of incrementally updating one sketch element. I). According to the scaling of the histogram, we keep the sketch invariant $S_j(t) = 3$, and adjust its hash value from $A_j(t) = 0.14$ to $A_j(t) \cdot e^{\lambda} = 0.147$ ($\lambda = 0.05$ in this example). II). By adding the incoming histogram element $x_{t+1} = 2$ ($2 \in \mathcal{E}$) to the scaled histogram, we recompute only the hash value for $i = 2$, i.e., $a_{2,j} = 0.142$. III). By selecting the minimum hash value between $A_j(t) \cdot e^{\lambda} = 0.147$ and $a_{2,j} = 0.142$, we update $S_j(t+1) = 2$ and $A_j(t+1) = 0.142$.

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 [34] to be at most $\frac{2}{g}$ with probability $1 - (\frac{1}{2})^d$.

In our case of streaming histogram with 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: before adding every new element from the data stream, we uniformly scale all counters across all rows by that factor, i.e., $Q(t) \cdot e^{-\lambda}$, and keep the following steps unchanged. In such a way, for any histogram element i, its estimated weighted cumulative count $V_i(t)$ is also scaled to $\min_l Q_{l,h_l(i)}(t) \cdot e^{-\lambda} = e^{-\lambda} \cdot V_i(t)$, which corresponds exactly to Step I in our sketch update process. In addition, it is easy to see that the estimated error does not change under this extension, as a uniform scaling does not affect the data structure itself. In this study, we set the parameters $d = 10$, $g = 50$ to guarantee an error of at most 4% with probability 0.999.

E. 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 sketch Q (with d rows), which takes $O(d)$ time. The total time complexity for updating one histogram element is hence $O(K + d)$.

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.

V. Experimental Evaluation

In this section, we evaluate 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.

A. Experimental Setup

To evaluate the performance of our similarity-preserving sketches, we perform classification tasks based on these sketches in difference scenarios. Specifically, based on labeled streaming histograms, we try to classify those histogram instances without labels. We use a KNN classifier [35] which can always take the most up-to-date training data (sketches) for classification. Such a property fits our case of classifying streaming histograms with continuously incoming histogram elements, where the sketches are continuously updated accordingly. We empirically set KNN to consider the five nearest neighbors. We consider the following evaluation scenarios:

Synthetic Dataset. Synthetic data is widely used in studying concept drift adaptation [19], [36]. 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 [36]. 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.

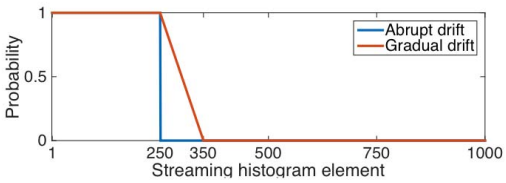


Fig. 5. Probability of streaming histogram elements generated from its initial distribution for the synthetic dataset.

- 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 [8], 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 patterns) yields much higher accuracy than considering only time (i.e., coarse-grained visiting patterns) [8]. 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 [22], [37]. 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[2]. Without loss of generality, we select three big cities, New York City, Tokyo and Istanbul, for our experiments. Table I summarizes the main characteristics of our dataset.

TABLE I
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

B. Performance on Synthetic Dataset

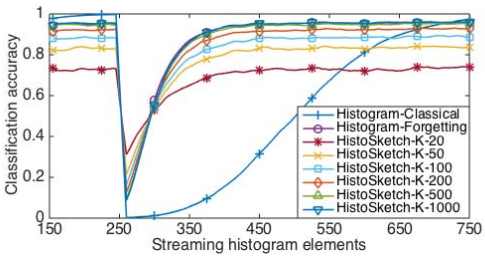
As the main purpose of HistoSketch is to efficiently approximate similarities for streaming histograms with concept drift, our experiments focus on how well it can approximate the similarity (considering the impact of the sketch length K), and how fast it can adapt to concept drift (considering the impact of the weight decay factor λ) in different scenarios.

1) 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 compare our method with the following two methods that keep the full histograms in memory:

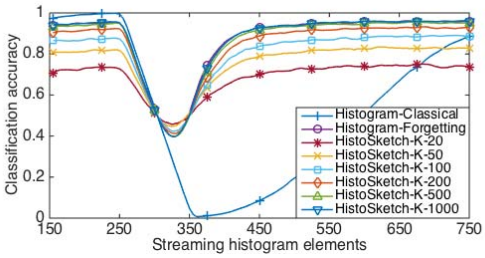
- ***Histogram-Classical*** where the histogram's elements are unweighted.
- ***Histogram-Forgetting*** where the elements are assigned gradual forgetting (exponential decay) weights.

Fig. 6 shows the classification accuracy over time (number of streaming elements per histogram) for both abrupt and gradual drift. First, compared to Histogram-Classical that slowly adapts to concept drift, Histogram-Forgetting shows fast adaptation for both cases. In the case of abrupt drift, for example, accuracy recovers after 700 and 400 elements for Histogram-Classical and Histogram-Forgetting, respectively. Second, HistoSketch can also quickly adapt to concept drift, showing similar adaptation speed as that of Histogram-Forgetting (i.e., accuracy recovers after 400 streaming histogram elements). Moreover, 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). Finally, 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

[2]https://developer.foursquare.com/categorytree

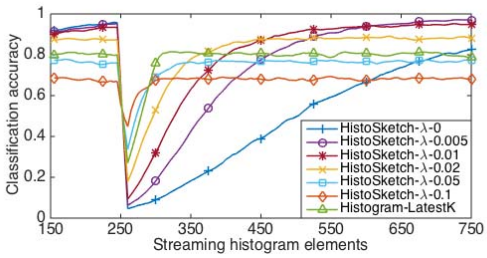


(a) Abrupt drift

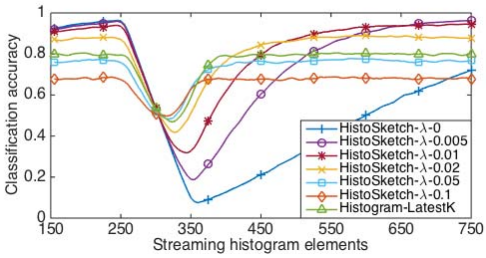


(b) Gradual drift

Fig. 6. Impact of sketch length K.



(a) Abrupt drift



(b) Gradual drift

Fig. 7. Impact of weight decay factor λ.

can preserve more information and thus better approximate the similarities of the Histogram-Forgetting. The accuracy flattens out after $K = 500$ (HistoSketch with $K \geq 500$ are highly overlapped with Histogram-Forgetting), indicating that a sketch of length 500 is sufficient for accurate similarity approximation.

2) 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 ***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. We set the same sketch length $K = 100$ for *Histogram-LatestK*.

Fig. 7 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 performance. Second, we find that Histogram-LatestK shows comparable results, i.e., its adaptation speed is faster than HistoSketch-λ-0.02 and slower than HistoSketch-λ-0.05 while its accuracy is lower than HistoSketch-λ-0.02 and higher than HistoSketch-λ-0.05. Despite such similar results, there are two obvious advantages of our methods: 1) *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, as the latter requires set operations while HistoSketch only relies on Hamming distance. For example, to classify one histogram using our testing PC[3], our method needs only 13ms while Histogram-LatestK takes 133ms, which shows a 10x speedup.

C. Performance on POI Dataset

To evaluate our method in the semantic place labeling task, we first compare it with state-of-the-art approaches, and then show its classification accuracy over time, followed by its runtime performance. We focus on the tradeoff between classification time and accuracy, and show that our sketches can dramatically reduce the classification time with only a small loss in accuracy.

1) Comparison with other methods: We compare HistoSketch with the following methods:

- ***Histogram-Coarse***: Discretized time slots (168 hours in a week, i.e., coarse-grained visiting patterns) are considered as histogram elements [7];
- ***Histogram-Fine-Classical***: User-time pairs (i.e., fine-grained visiting patterns) are regarded as histogram elements (unweighted) [8];
- ***Histogram-Fine-LatestK***: Histogram-Fine with only latest K histogram elements;

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

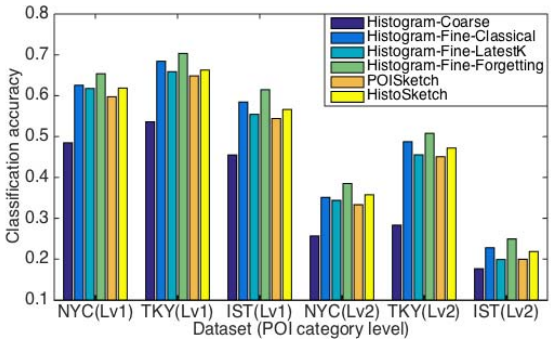


Fig. 8. Comparison with other methods

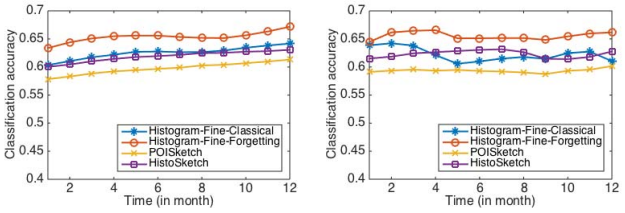
- ***Histogram-Fine-Forgetting***: Histogram-Fine with gradual forgetting weights (λ is empirically set to 0.01);
- ***POISketch***: POISketch is a state-of-the-art sketching method to approximates Histogram-Fine-Classical [8], which is equivalent to our HistoSketch with λ=0 (giving equal weights to all histogram elements);
- ***HistoSketch***: Approximation of Histogram-Fine-Forgetting (λ=0.01).

The sketch length K is empirically set to 100 for all related methods.

Fig. 8 plots the average classification accuracy over 12 months for all datasets on two-level POI categories. First, we observe that Histogram-Coarse has the worst accuracy, as coarse-grained visiting patterns can only capture the temporal dynamics of POIs. Second, based on fine-grained visiting patterns, Histogram-Fine-Forgetting yields the highest accuracy, showing the effectiveness of considering gradual forgetting weights on streaming histogram elements. Third, our HistoSketch also outperforms POISketch by considering gradual forgetting weights in the sketching process. In particular, HistoSketch can efficiently approximate the similarity with only a small loss of classification accuracy (e.g., about 3.5% for root POI categories on the NYC dataset) compared to Histogram-Fine-Forgetting. Finally, HistoSketch also outperforms Histogram-Fine-LatestK, showing the effectiveness of gradual (rather than abrupt) forgetting of historical data in the semantic place labeling task.

2) Classification accuracy over time: In this experiment, we study the classification accuracy of our method over time. 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. 9(a) shows the classification accuracy for each of the 12 testing months on the NYC dataset with the 9 root levels of POI categories (Experiments on the other datasets and on the 291 sub-categories show similar results). We observe that the accuracy slightly increases over time with the accumulated histogram elements, as observing more histogram elements leads to more accurate similarity measurement of streaming histograms. Compared to POISketch that approximates Histogram-Fine-Classical, our HistoSketch achieves consistently higher accuracy by efficiently approximating Histogram-Fine-Forgetting.



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

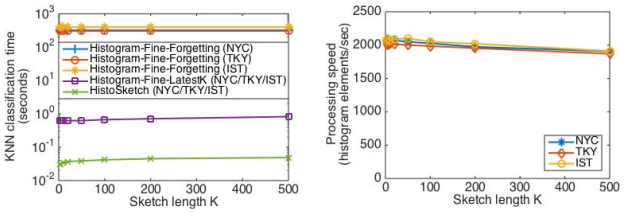
Fig. 9. Classification performance over time

Fig. 9(b) shows the same results on the testing POIs with category changes only. We observe a larger improvement of our method over POISketch than that in Fig. 9(a), which 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.

3) Runtime performance: In this experiment, we investigate the runtime performance of both sketch-based classification and HistoSketch maintenance. More precisely, we evaluate both the classification time and the streaming histogram processing speed of our method w.r.t. sketch length K (as the time complexity of maintaining HistoSketch mainly depends on the sketch length K).

Fig. 10(a) plots the KNN classification time (log scale) on our test PC[3]. We observe that compared to the full histograms (Histogram-Fine-Forgetting), using 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., $|\mathcal{E}|$=2M for the NYC dataset). This also indicates that our sketches take significant less memory to maintain. Compared to Histogram-Fine-LatestK, our method also shows a 15x speedup, as Histogram-Fine-LatestK requires set operations for similarity computation.

Fig. 10(b) shows the processing speed of streaming histogram elements. We observe that our method is able to process histogram elements at high velocity, i.e., about 2000 per second for all three datasets. We also find that the processing speed slightly decreases with increasing sketch lengths, as longer sketches required a little more time to update. We believe that such a processing speed can 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.



(a) Classification time (b) Processing speed

Fig. 10. Runtime performance

VI. Conclusion and Future Work

This paper introduces HistoSketch, an efficient similarity-preserving sketching method for streaming histograms with concept drift. HistoSketch maintains a set of compact and fixed-sized sketches of streaming histograms to approximate their normalized min-max similarity. By incrementally updating the sketches with incoming histogram elements, our method can gradually forget outdated elements and thus gracefully adapt to concept drift. Based on both synthetic and real-world datasets, our empirical evaluation showed both the efficiency and the effectiveness of our method for similarity approximation and concept drift adaptation. Compared to full streaming histograms with gradual forgetting weights in particular, HistoSketch is able to dramatically reduce the classification time (with a 7500x speedup) at the expense of a small loss in accuracy only (about 3.5%).

As future work, we plan to further explore the problem of sketching two-dimensional (bivariate) streaming histograms, and apply our method to other application domains, such as for recommendation or community detection.

Acknowledgment

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).

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] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[3] F. Pereira, N. Tishby, and L. Lee, "Distributional clustering of english words," in *Proc. of ACL*. Association for Computational Linguistics, 1993, pp. 183–190.

[4] L. D. Baker and A. K. McCallum, "Distributional clustering of words for text classification," in *Proc. of SIGIR*, 1998, pp. 96–103.

[5] R. Zafarani and H. Liu, "Connecting users across social media sites: a behavioral-modeling approach," in *Proc. of KDD*, 2013, pp. 41–49.

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

[7] 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.

[8] D. Yang, B. Li, and P. Cudré-Mauroux, "Poisketch: Semantic place labeling over user activity streams," in *Proc. of IJCAI*, 2016, pp. 2697–2703.

[9] L. Chen, J. Jakubowicz, D. Yang, D. Zhang, and G. Pan, "Fine-grained urban event detection and characterization based on tensor cofactorization," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 3, pp. 380–391, 2017.

[10] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *arXiv preprint arXiv:1408.2927*, 2014.

[11] C. C. Aggarwal and P. S. Yu, "On classification of high-cardinality data streams," in *Proc. of SDM*, 2010, pp. 802–813.

[12] Y. Bachrach, E. Porat, and J. S. Rosenschein, "Sketching techniques for collaborative filtering," in *Proc. of IJCAI*, 2009, pp. 2016–2021.

[13] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," in *Proc. of STOC*, 1998, pp. 327–336.

[14] M. Mitzenmacher, R. Pagh, and N. Pham, "Efficient estimation for high similarities using odd sketches," in *Proc. of WWW*, 2014, pp. 109–118.

[15] K. Kutzkov, M. Ahmed, and S. Nikitaki, "Weighted similarity estimation in data streams," in *Proc. of CIKM*, 2015, pp. 1051–1060.

[16] P. Li, "0-bit consistent weighted sampling," in *Proc. of KDD*, 2015, pp. 665–674.

[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, 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[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] S. Ioffe, "Improved consistent sampling, weighted minhash and l1 sketching," in *Proc. of ICDM*, 2010, pp. 246–255.

[29] W. Wu, B. Li, L. Chen, and C. Zhang, "Consistent weighted sampling made more practical," in *Proc. of WWW*, 2017, pp. 1035–1043.

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

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

[32] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

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

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

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

[36] 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.

[37] 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.