sensors

MDPI

Article

FlexSketch: Estimation of Probability Density for Stationary and Non-Stationary Data Streams

Namuk Park and Songkuk Kim *

School of Integrated Technology, Yonsei University, Incheon 21983, Korea; namuk.park@yonsei.ac.kr
* Correspondence: songkuk@yonsei.ac.kr

Abstract: Efficient and accurate estimation of the probability distribution of a data stream is an important problem in many sensor systems. It is especially challenging when the data stream is non-stationary, i.e., its probability distribution changes over time. Statistical models for non-stationary data streams demand agile adaptation for concept drift while tolerating temporal fluctuations. To this end, a statistical model needs to forget old data samples and to detect concept drift swiftly. In this paper, we propose FlexSketch, an online probability density estimation algorithm for data streams. Our algorithm uses an ensemble of histograms, each of which represents a different length of data history. FlexSketch updates each histogram for a new data sample and generates probability distribution by combining the ensemble of histograms while monitoring discrepancy between recent data and existing models periodically. When it detects concept drift, a new histogram is added to the ensemble and the oldest histogram is removed. This allows us to estimate the probability density function with high update speed and high accuracy using only limited memory. Experimental results demonstrate that our algorithm shows improved speed and accuracy compared to existing methods for both stationary and non-stationary data streams.

Keywords: probability density estimation; streaming data; sensor system

Citation: Park, N.; Kim, S. FlexSketch: Estimation of Probability Density for Stationary and Non-Stationary Data Streams. *Sensors* **2021**, *21*, 1080. https://doi.org/10.3390/s21041080

Academic Editor: Andrea Facchinetti
Received: 30 December 2020
Accepted: 28 January 2021
Published: 4 February 2021

1. Introduction

Estimating the probability density function (PDF) of a random variable based on a stream of data samples from sensors is a fundamental problem arising in a broad range of areas such as machine learning [1], data structures [2], and systems [3]. There are two recent challenges in this problem.

First, the characteristics of a data stream often change. This might be due to the accidental fluctuation caused by an insufficient number of samples. In this case, a sufficient amount of data could reduce the fluctuation gradually. In some other cases, the data stream itself is non-stationary. In other words, the probability distribution of a random variable over the data stream, called concept, changes over time, which is called concept drift [4]. Concept drift occurs in many types of data such as temporal sensor data [5], video [6], and spatiotemporal data [7]. A static model constructed with the assumption of stationarity of the data stream may lead to an erroneous conclusion under the presence of concept drift. Therefore, there is a need for a method that can estimate PDFs adaptively according to concept drift for practical applications.

Second, real-world applications need to be able to handle increasing amounts of data and high-speed data streams while keeping low latency. Therefore, the demand for an online algorithm to estimate PDFs with high speed and high accuracy using only a small amount of memory is ever-increasing. While there exist online PDF estimation algorithms in literature [8–11], they usually cannot update probability models at high speed. Furthermore, they cannot adapt well to various types of data streams including those with concept drift.

In order to deal with these challenges, we propose FlexSketch, which is an online probability density estimation algorithm that achieves high update speed and high accu-

racy with only a small amount of memory for both stationary and non-stationary data streams. As shown in Figure 1, FlexSketch estimates the PDF by using an ensemble structure composed of several statistical models. In particular, we exploit histogram for the statistical models, which allows fast and low-memory operations for data streams. Each histogram represents statistics of a different length of data history. FlexSketch updates each histogram for a new data sample and builds a new histogram when it detects the concept drift of input data. By decoupling updating each statistical model from amending the composition of the ensemble, FlexSketch achieves high accuracy both for stationary and non-stationary data streams.

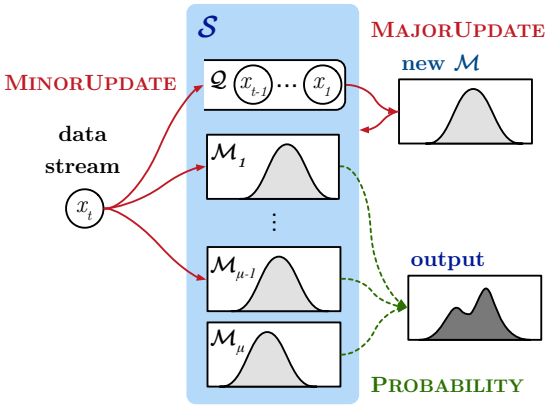


Figure 1. Overview of the proposed FlexSketch.

To adapt non-stationary data streams in an efficient way, a single operation updating the estimated PDF according to the data stream is divided into two elementary operations: "fast and minor update operation" (MinorUpdate) and "slow and major update operation" (MajorUpdate). The type of operation to be performed varies depending on the characteristics of the data stream. When minor changes occur in a data stream, FlexSketch updates the PDF for the data stream at high speed by using MinorUpdate. This operation simply updates each model in FlexSketch. In contrast, when major changes occur in the data stream, FlexSketch updates the PDF by using MajorUpdate, which builds a new model including the recent data stream and adds it to FlexSketch. Finally, FlexSketch has multiple versions of a model, ranging from a version representing only recent data to a version representing both recent and old data. FlexSketch constructs the PDF by linear combination of these coarse models.

FlexSketch dynamically decides when to forget old data and to build a new statistical model by measuring divergence between the current model and recently sampled data. This allows FlexSketch to stay stable when the concept of the data stream does not change, and to tolerate temporal out-linear data. In addition, FlexSketch achieves agile adaptation to sudden or incremental concept drift since MajorUpdate integrates a newly built histogram, which amends the statistical model. Though the histogram is a compact data structure and easy to maintain, it may provide coarse information about probability distribution. FlexSketch alleviates this problem by incorporating an ensemble of histograms.

The current implementation of FlexSketch only supports one-dimensional data. There are many applications relying on statistical modeling of one-dimensional data streams such as online anomaly detection [5,12], fault-detection [13] and DDoS detection [14], which are potential areas where FlexSketch is applicable. While many online sensor-based applications handle one-dimensional data, other applications need to deal with multi-dimensional data streams. Since the limited dimension of FlexSketch circumscribes the application of FlexSketch in some areas such as machine learning for high dimensional data [15] and anomaly detection based on multivariate data [16], FlexSketch needs to be expanded to handle multi-dimensional data for wider applications.

The experimental results demonstrate that FlexSketch updates PDFs for data streams up to 16× faster than the alternatives while using only a limited amount of memory. Moreover, FlexSketch adapts well to various types of concept drift, and it is more accurate than the alternatives.

The main contributions of FlexSketch are as follows:

- We propose a new method to estimate probability distribution for data streams with concept drift.
- FlexSketch decouples adapting to concept drift from adjusting the statistical model for stationary data by incorporating two separate operations.
- FlexSketch achieves low computational overhead and high throughput, which are critical for processing of stream data, using an ensemble of compact histograms.

The remainder of the paper is organized as follows. Section 2 briefly surveys the related work. In Section 3, the proposed FlexSketch is described in detail. Section 4 presents extensive experimental results. Finally, the conclusions are given in Section 5.

2. Related Work

There are multiple research topics to deal with data streams with concept drift [17]. Supervised classification of data streams with concept drift is also studied extensively, e.g., [18–21]. The core parts of these researches are how to detect concept drift, how to forget old data, and how to rebuild a new statistical model. Some methods [9,22] gradually update statistical model without explicit detection of concept drift. Other studies attempt to detect concept drift in batch-based methods [23–26] and online methods [27–29]. While [30,31] rely on process control, FlexSketch detects concept drift using a multiple-window-based method like [32]. Researches also focused on how to measure difference in distribution between recent data and old data. The common methods are based on entropy or KL-divergence [33–35]. We introduce an error based metric to detect concept drift in Section 3.1.

To deal with non-stationary data, statistical models should be able to forget old data or to depreciate their contribution. Some methods [9,36,37] decay the importance of old data linearly or exponentially. This approach is good for tolerating temporal fluctuations. Since gradual decaying is slow to adapt to sudden concept drift, [38,39] use sliding window mechanisms to keep some recent data and to discard old data. FlexSketch deploys both gradual decaying and abrupt forgetting. When the input data is stationary, FlexSketch depreciates old data exponentially. However, when FlexSketch detects concept drift, it discards the oldest histogram and incorporates a new histogram, which allows FlexSketch agile adaptation.

There exist many kinds of density estimation algorithms for data streams. Traditionally, kernel density estimation is performed as a batch-processing algorithm for density estimation of a dataset. There are some algorithms [9–11] generalized to online processing for adaptive density estimation. They usually constitute a Gaussian mixture model by assigning a Gaussian kernel to newly added data and thereafter merge kernels based on certain rules. Particularly, the online kernel density estimation (denoted by oKDE in this paper) [9] can adapt to concept drift by enabling to forget past data. However, this method is slow in updating the estimated probability density due to the requirement of relatively massive calculation. In contrast, we focus on developing an efficient mechanism that can update the probability density adaptively by exploiting updating operations having different levels of computational complexity.

There also exist methods to estimate the distribution of a data stream based on histograms, including the streaming parallel decision tree (denoted by SPDT in this paper) [8], variations of the V-optimal histogram algorithm [40,41] and quantile summarization algorithms [42–45]. However, they cannot forget the contribution of the past data and adapt to various types of concept drift. To solve this problem, model maintenance strategies using fixed and variable size sliding windows [46–48] for histograms can be used. [7] proposes a histogram-based sketch mechanism with gradual forgetting. However, it is unknown whether they guarantee satisfactory performance when different types of concept drift

occurs. On the other hand, FlexSketch efficiently updates the statistical model for both high accuracy and high efficiency by amending the composition of the ensemble.

There are some studies using ensemble methods, e.g., [49–51]. While some methods [33,34] use hierarchical structure, FlexSketch uses flat combination of compact and simple data structure like [19,52]. Most of previous works focus on improving accuracy of supervised classification for data streams with concept drift. On the other hand, FlexSketch uses the ensemble technique to improve the update speed of density estimation.

3. Proposed Method

The goal of our method is to estimate the PDF of stationary (i.e., without concept drift) and non-stationary (i.e., with concept drift) data stream at high speed and high accuracy while using a small amount of memory. Here, the meaning of accurately estimating the PDF for a stationary data stream is straightforward. On the other hand, for a non-stationary case, it is not simple and has several aspects. We consider the accuracy of density estimation for a non-stationary data stream from three points of view. First, the estimated PDF should forget old concepts quickly after concept drift occurs. Second, the estimated PDF should adapt to the latest concept as soon as a concept drift occurs. Third, the estimated PDF should remain stable even if an accidental outlier occurs in the data stream.

To fulfil these requirements, the proposed method is built on the following ideas.

(a) We choose a histogram as the statistical model. When there are only minor changes in the data stream, a histogram is a suitable model since it can be updated at a high speed.
(b) Our method uses an ensemble data structure consisting of several histograms. The ensemble structure can compensate for inaccuracy of a histogram.
(c) We design two adaptation techniques. If the data stream is stationary or there are only minor changes in it, FlexSketch updates the models, i.e., histograms. On the other hand, if there are major changes in the data stream, updating the models may not guarantee sufficient accuracy. To address this issue, we generate a new model that represents the changed data stream and adds it to the data structure.

Let $\mathcal{S}$ denote the data structure of the FlexSketch framework. $\mathcal{S}$ consists of a recent data stream and multiple versions of a statistical model as follows:

$$\mathcal{S} = \{Q, \mathcal{M}_1, \ldots, \mathcal{M}_{N_M}, n_1, \ldots, n_{N_M}\} \tag{1}$$

where Q is the buffer for the recent dataset given through the input data stream, $\mathcal{M}_i$ is the ith histogram, $N_M \geqslant 2$ is the total number of histograms, and n_i is the number of data used to update $\mathcal{M}_i$. Since we build a new histogram when a major change in the data stream is detected, the histograms in $\mathcal{S}$ are created at different times. As a convention, $\mathcal{M}_1$ is the most recently added one and $\mathcal{M}_{N_M}$ is the oldest one. This means that $\mathcal{M}_{i+1}$ is older than $\mathcal{M}_i$ and thus undergoes more updates. Therefore, $n_{i+1} > n_i$. Each histogram is a set of disjoint intervals called bins (I_j) and the frequency count (m_j) for each bin, where $j = 1, \ldots, N_B$:

$$\mathcal{M} = \{I_1, \ldots, I_{N_B}, m_1, \ldots, m_{N_B}\}. \tag{2}$$

Two important operations of FlexSketch are (a) the update operation for the data stream online and (b) the query operation to obtain the probability for a certain data, which are explained below.

3.1. Update Operation

Algorithm 1 summarizes the operation to update $\mathcal{S}$ with a given data sub-stream X whose number of data is $|X|$. First, $\mathcal{S}$ is updated at a minor level using operation MINORUPDATE and X is appended in the buffer. If the size of the buffer Q exceeds a threshold N_Q, i.e., if only the minor update has been performed with a certain number of data, the adequacy of the most recently added model $\mathcal{M}_1$ is examined by operation

DIAGNOSE. If a large discrepancy is found from the operation between the recent data stream stored in Q and $\mathcal{M}_1$, the major update operation (MAJORUPDATE) is performed and the buffer is cleared.

Algorithm 1: Update operation.

Input: $\mathcal{S} = \{Q, \mathcal{M}_1, \ldots, \mathcal{M}_{N_M}, n_1, \ldots, n_{N_M}\}$, sub-stream: X, size limit of Q: N_Q, number of models: N_M, DIAGNOSE threshold: γ

Output: Updated $\mathcal{S}$

```
1  S ← MinorUpdate(S, X)
2  Q ← Q ∪ X
3  if |Q| > N_Q then
4      if Diagnose(M_1, Q) ⩾ γ then
5          S ← MajorUpdate(S, Q)
6      end
7      Q ← ∅
8  end
9  return S
10 function MinorUpdate(S, X):
11     for i ← 1 to N_M − 1 do
12         M_i ← SingleUpdate(M_i, X)
13         n_i ← n_i + |X|
14     end
15     S ← {Q, M_1, ..., M_{N_M}, n_1, ..., n_{N_M}}
16     return S
17 function MajorUpdate(S, X):
18     M' ← Build(S, X)
19     n' ← |X|
20     S ← {Q, M', M_1, ..., M_{N_M−1}, n', n_1, ..., n_{N_M−1}}
21     return S
```

The MINORUPDATE operation causes only minor changes of $\mathcal{S}$. The steps of the operation are shown in lines 10 to 16 of Algorithm 1. Each histogram in $\mathcal{S}$ except the oldest one ($\mathcal{M}_{N_M}$) is updated by operation SingleUpdate. This operation consists of two steps. First, it searches the bin I_j whose interval contains each data in X, x. For fast search, we use the red-black tree method. Second, it increases the count of I_j, m_j, by 1. If there is no bin for x, x is ignored. This causes a discrepancy between the histogram and X, which is resolved by the MAJORUPDATE operation.

The DIAGNOSE operation measures the amount of discrepancy between the most recent model $\mathcal{M}_1$ and the data stream in Q. If the output of the operation is larger than a threshold γ, we consider that concept drift occurs and the MAJORUPDATE operation needs to be performed.

There are some requirements for the DIAGNOSE operation. First, its result should be invariant under scaling transformation of the data, so that the threshold is independent of the scale of the data. Second, the result of DIAGNOSE should be stable even when only a small number of data are given; otherwise, MAJORUPDATE is performed too frequently, which results in increased computational complexity, and the PDF estimation becomes inaccurate. Third, the result of DIAGNOSE must be finite even if the input dataset is not included in the domain of the histogram; otherwise, the result will diverge whenever accidental outliers deviate from the domain.

To design a DIAGNOSE operation satisfying these requirements, we first define the error function $\Delta(x)$ between X and $\mathcal{M}$ as the absolute difference between the cumulative

distribution function (CDF) of $\mathcal{M}$, $\mathrm{CDF}_{\mathcal{M}}(x)$, and the empirical distribution function (EDF) for X, $\mathrm{EDF}_X(x)$ (Figure 2a):

$$\Delta(x) = |\mathrm{CDF}_{\mathcal{M}}(x) - \mathrm{EDF}_X(x)|. \tag{3}$$

Then, a representative value of the error function serves as the output of the DIAGNOSE operation. We consider two options, i.e., the maximum value given by

$$\check{\varepsilon} = \max_x \Delta(x) \tag{4}$$

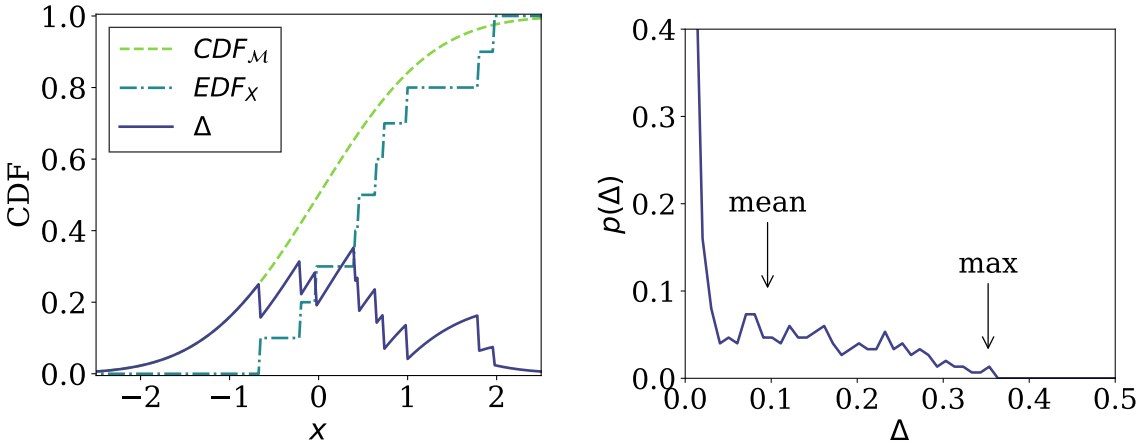
and the mean value given by

$$\bar{\varepsilon} = \int_0^1 \Delta \cdot p(\Delta) d\Delta \tag{5}$$

where $p(\Delta)$ is the PDF for Δ. $\check{\varepsilon}$ or $\bar{\varepsilon}$ may be used as the result of DIAGNOSE directly. Note that since $\Delta(x)$ is the difference between two probability distributions, its value can take only between 0 and 1, which is the range of the integration. However, we note that the range of $\check{\varepsilon}$ and $\bar{\varepsilon}$, which is between 0 and 1, is too narrow for practical use. Thus, for convenience, we scale them to obtain the final output of DIAGNOSE as follows:

$$\delta = \frac{\varepsilon}{1-\varepsilon} \tag{6}$$

Note that $\delta = 0$ for $\varepsilon = 0$, $\delta \simeq \varepsilon$ for small ε (i.e., $\varepsilon \ll 1$), and $\delta = \infty$ for $\varepsilon = 1$. Therefore, the output of DIAGNOSE ranges from 0 to ∞ through this scaling.



(**a**) (**b**)

Figure 2. Example of the error function. (**a**) The error function is given as the difference (solid line) between the cumulative distribution function (CDF) of a model (dashed line) and the empirical distribution function (EDF) of a sub-stream (dash-dot line). (**b**) The probability density function (PDF) for the value of the error function.

The MAJORUPDATE operation is shown in lines 18 to 21 in Algorithm 1. It first builds a new histogram $\mathcal{M}'$ with X through operation BUILD in order to accommodate a big change (i.e., concept drift) in the data stream. Then, the new histogram is enqueued to $\mathcal{S}$ as the first model of the ensemble and the oldest one ($\mathcal{M}_{N_M}$) is dequeued from $\mathcal{S}$.

The BUILD operation is shown in Algorithm 2. It basically creates a new histogram that covers the data ranges of both the existing histograms in $\mathcal{S}$ and the recent data, so that the new histogram can account for the characteristics of the recent data. First, the boundaries of the bins of the new histogram are obtained so as to uniformly split the range of the combined CDF (CDF of $\mathcal{S}$ and EDF of X) (lines 2 to 4 of Algorithm 2). Then, the EDF of X is used to obtain the count in each bin (lines 5 to 8 of Algorithm 2).

Algorithm 2: Build operation.

Input: $\mathcal{S} = \{Q, \mathcal{M}_1, \ldots, \mathcal{M}_{N_M}, n_1, \ldots, n_{N_M}\}$, sub-stream: X, number of bins: N_B
Output: new histogram: $\mathcal{M}'$

$CDF \leftarrow (\mathrm{CDF}_{\mathcal{S}} + \mathrm{EDF}_X)/2$
for $j \leftarrow 1$ ***to*** $N_B + 1$ **do**
  $z_j \leftarrow CDF^{-1}(\mathrm{CDF}_{\mathcal{U}(0,1)}(\frac{j}{N_B+2}))$
end
for $j \leftarrow 1$ ***to*** $N_B + 1$ **do**
  $I_j \leftarrow [z_j, z_{j+1})$
  $m_j \leftarrow |X| \cdot (\mathrm{EDF}_X(z_{j+1}) - \mathrm{EDF}_X(z_j))$
end
$\mathcal{M}' \leftarrow \{I_1, \ldots, I_{N_B}, m_1, \ldots, m_{N_B}\}$
return $\mathcal{M}'$

3.2. Query Operation

The query operation is to obtain the probability of a certain input data using $\mathcal{S}$ that has been established using the past data through the update operations explained above. As mentioned before, we employ an ensemble approach for this using the histograms contained in $\mathcal{S}$. In other words, the probability of a given data x is calculated from $\mathcal{S}$ as a linear combination of the PDFs represented by the histograms $\mathcal{M}_i$, i.e.,

$$p_{\mathcal{S}}(x) = \sum_{i=1}^{N_M} \alpha_i \cdot p_{\mathcal{M}_i}(x), \tag{7}$$

where $p_{\mathcal{M}_i}(x)$ is the probability of x from histogram $\mathcal{M}_i$ and α_i is the weight of $\mathcal{M}_i$. The former is given by the proportion of the data count for the bin to which x belongs, i.e.,

$$p_{\mathcal{M}}(x) = \begin{cases} \frac{m_j}{|I_j| \cdot \sum_{k=1}^{N_B} m_k}, & x \in I_j, \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

where $|I_j|$ is the size of an interval and we omit the subscript i for simplicity. The weight α_i is determined in a way that a newer histogram receives a higher weight. Then, the final probability (7) depends more on the histograms that have been built more recently. For this, we use that an older histogram has been updated with more data, i.e., $n_{i+1} > n_i$. Specifically, the weight α_i is set to be negatively proportional to n_i, where the exponential function is used to ensure the weight value remains positive and, at the same time, to forget an old concept exponentially:

$$\alpha_i = \frac{\exp(-n_{i-1}\lambda/N_Q)}{\sum_{k=1}^{N_M} \exp(-n_{k-1}\lambda/N_Q)}, \tag{9}$$

with $n_0 = 0$. Here, λ is a hyperparameter. Note that $\sum_{i=1}^{N_M} \alpha_i = 1$ due to the normalization and $1 > \alpha_i > \alpha_{i+1} > 0$ because $n_i < n_{i+1}$.

To see how this works, let us consider the situation where concept drift occurs continuously so that we can assume that $n = n_{i+1} - n_i$ and $\alpha = \alpha_{i+1}/\alpha_i = \exp(-n\lambda/N_Q)$ for all i. And, let a PDF of concept $\mathcal{C}_a$ changing over time be $p_{\mathcal{C}_a}$ for positive integer a with $\mathcal{C}_1$ being the latest one. Then, $p_{\mathcal{M}_1} = p_{\mathcal{C}_1}$, $p_{\mathcal{M}_2} = p_{\mathcal{C}_1} + p_{\mathcal{C}_2}$, $\ldots$, $p_{\mathcal{M}_{N_M}} = p_{\mathcal{C}_2} + \ldots$ since MINORUPDATE does not update the oldest model. Then, $p_{\mathcal{S}} \propto (1 + \alpha + \cdots + \alpha^{N_M - 1})p_{\mathcal{C}_1} + (\alpha + \cdots + \alpha^{N_M})p_{\mathcal{C}_2} + \cdots$ holds. In other words, the contribution of the concept decreases at a rate of α. This means that FlexSketch forgets an old concept exponentially.

4. Experiments

We evaluate the computation time, accuracy, and memory usage of the proposed method for various types of stationary and non-stationary data streams. In particular, we compare our method with the two representative existing density estimation algorithms, oKDE [9] and SPDT [8].

4.1. Datasets

Stationary datasets: We consider three distributions. The first is a standard normal distribution, $\mathcal{N}(0,1)$, which appears frequently. The second is a bimodal distribution, $\frac{1}{2}\mathcal{N}(-2,1)+\frac{1}{2}\mathcal{N}(2,1)$, which is used to test if a density estimation algorithm can recognize multiple modes. The third is a log-normal distribution, $\ln\mathcal{N}(0,1)$, which is used to test if an algorithm can estimate a long-tailed distribution. For all cases, one million data are randomly generated to follow the distributions.

Non-stationary datasets: For non-stationary datasets, we consider three types of concept drift as follows. For each case, we use one million data randomly sampled from the distribution.

(a) Sudden concept drift is defined as the case where the distribution of the data stream changes suddenly. It is to test how well a density estimation algorithm forgets old concepts after concept drift occurs. The underlying distribution is a normal distribution whose mean value changes abruptly, i.e., $\mathcal{N}\big(x(t),1\big)$, where $x(t)=0$ for $t<t_1$ and $x(t)=x_1$ for $t\geqslant t_1$. We consider $t_1=300$ and $x_1=5$ as shown in Figure 3a.

(b) Incremental concept drift is defined as the case where the distribution of the data stream changes incrementally. It is to test how well a density estimation algorithm adapts to the latest concept. The underlying distribution is a normal distribution whose mean value moves at a constant speed, i.e., $\mathcal{N}\big(x(t),1\big)$, where $x(t)=0$ for $t\leqslant t_1$ and $x(t)=v_1\cdot(t-t_1)$ for $t>t_1$. We set $t_1=300$ and $v_1=0.01$, as shown in Figure 3b.

(c) Blip concept drift is defined as the case where the distribution of data stream suddenly changes and returns to the original state in a short time. It is to test how well the estimated PDF remains stable even if an outlier occurs. The underlying distribution is a normal distribution whose mean value changes suddenly and returns, i.e., $\mathcal{N}\big(x(t),1\big)$, where $x(t)=x_1$ for $t_1<t\leqslant t_1+t_\epsilon$ and $x(t)=0$ otherwise. t_ϵ is the duration of blip concept drift, which is set to $t_\epsilon=3$. We also set $x_1=5$ and $t_1=300$, as shown in Figure 3c.

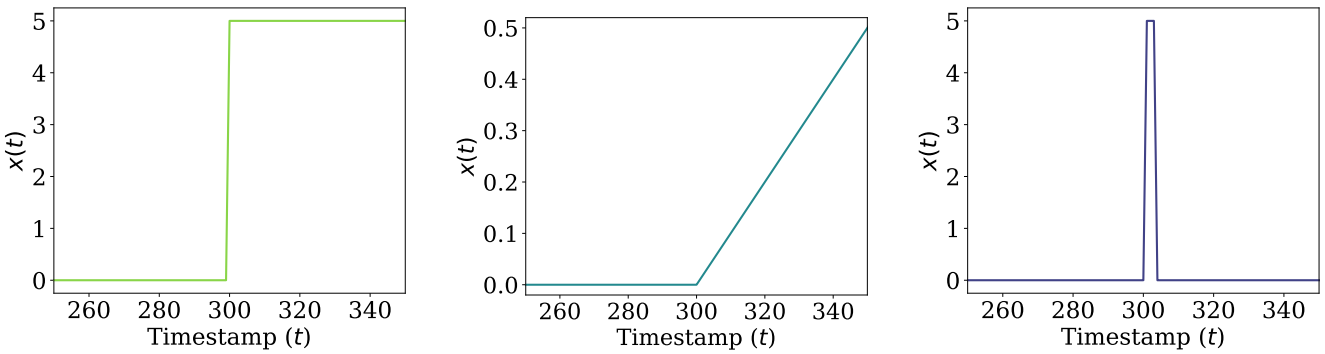


Figure 3. Changes in mean of distributions for data streams where concept drift occurs. (**a**) Sudden concept drift. (**b**) Incremental concept drift. (**c**) Blip concept drift.

4.2. Implementation

The parameters of FlexSketch are selected such that it has similar accuracy to oKDE and SPDT for the stationary data streams as follows: $N_M = 3$, $N_Q = 30$, $\lambda = 2.5$, and $\gamma = 0.4$. FlexSketch is implemented in Scala, which is publicly available at GitHub [53]. For oKDE, we use the JAVA implementation available in [54]. For SPDT, we use the Scala implementation in [55]. Note that the accuracy of SPDT decreases when concept drift occurs since SPDT stores the entire frequencies of the data stream. To address this issue, we modify SPDT by using a sliding window, which is referred to as SPDTw. The window size is set to 100 (this value was tuned such that SPDTw would exhibit similar accuracy to SPDT for stationary data. Increases in window size favor accuracy of stationary (or slowly changing) data streams to sudden concept drift, which reductions cause the inverse. Therefore, we calibrated these comparative methods for equitable results), for which SPDTw shows similar accuracy to FlexSketch for non-stationary data streams.

We perform all experiments on a machine with 4-core Intel CPU i7-7700K @ 4.2 GHz and 16 GB memory. The experiments run on a single thread. The version of Scala is 2.12.5 and the version of Java is 1.8.0.

4.3. Performance Metrics

Throughput We evaluate the computation times of the update and query operations of FlexSketch in million operations per second (Mops), which indicates the number of times per second our benchmark operation can be executed. There is a performance degradation in JVM in the first few iterations. Thus, we start to record the throughput after 20 iterations to warm up. Then, we record the mean value of the throughputs for the subsequent 30 iterations to minimize accidental deviations.

Error We measure the discrepancy between the estimated PDF and the ground truth distribution. We adopt the scaled mean average error (scaled MAE) of CDF, which is defined in (5) with scaling in (6), i.e., $\delta = \overline{\varepsilon}/(1-\overline{\varepsilon})$.

Adaptability When concept drift occurs, the PDF estimated by a density estimation algorithm changes over time, so does the error. Thus, the mean of the error for a given time interval is not a sufficient metric for the accuracy of the algorithm for non-stationary data streams. Instead, we measure the adaptability of the algorithm using how the error changes over time. For this, we introduce a damped harmonic oscillator model in classical mechanics (e.g., a vibrating mass connected to a spring under damping) to represent the change in the error of the density estimation. In other words, the stability against outliers is equivalent to the resistance force (or frictional force) and the UPDATE operation is equivalent to the restoring force. The density estimation algorithm tries to make the error smaller as the error increases and to keep the error unchanged as the error suddenly increases. Then, the governing equation for the time-dependent error $\delta(t)$ can be written as:

$$\left(\ddot{\delta}(t) + \ddot{\delta}_0(t)\right) + \underbrace{c \cdot \left(\dot{\delta}(t) + \dot{\delta}_0(t)\right)}_{\text{resistance force}} + \underbrace{k \cdot \delta(t)}_{\text{restoring force}} = 0 \tag{10}$$

where $\delta_0(t)$ is the error between before and after the data distribution changes, $\dot{\delta}$ and $\ddot{\delta}$ are the first- and second-order time derivatives of δ, respectively, and k and c are model coefficients. k and c are determined by fitting the observed values of $\delta(t)$ and $\delta_0(t)$ to the model (10) under the assumption of over-damped oscillation (i.e., $(c/2)^2 > k$). The solution is given by

$$\delta(t) = \underbrace{A_1 e^{-(c/2+\sqrt{c^2/4-k})t}}_{\text{short-lived term}} + \underbrace{A_2 e^{-(c/2-\sqrt{c^2/4-k})t}}_{\text{long-lived term}} \tag{11}$$

where A_1 and A_2 are constants. Based on the fitted model, the following four performance metrics are derived.

(a) Half-life: In order to measure the adaptability of an algorithm under sudden concept drift, we measure the time taken until the error at the time of concept drift is reduced by a half, which is denoted as half-life:

$$t_{1/2} = \delta^{-1}\left(1/2 \cdot \delta(t_1)\right). \tag{12}$$

This metric basically measures how quickly an old concept is forgotten in the short term.

(b) Lifetime: Similarly, we also quantify how long the contribution of the past data stays, or equivalently, how quickly the old concept is forgotten in the long term. The lifetime is defined as the time required for a long-lived term in (11) to reduce to $1/e$ times its initial value, which is given by

$$\tau = \left(c/2 - \sqrt{c^2/4 - k}\right)^{-1}. \tag{13}$$

(c) Lag: The lag measures how well the estimated PDF adapts to the data stream under incremental concept drift. It is defined as the absolute ratio of δ and the derivative of δ_0 at $t \to \infty$, which can be obtained by

$$\left|\frac{\delta(\infty)}{\dot{\delta}_0(\infty)}\right| = \frac{c}{k}. \tag{14}$$

If an algorithm does not adapt well to the concept drift, the accumulated error makes the algorithm lag behind more and more.

(d) Instability The instability measures how fast the estimated PDF moves for a short duration when blip concept drift occurs. It is defined as the velocity of the error, which can be approximated as

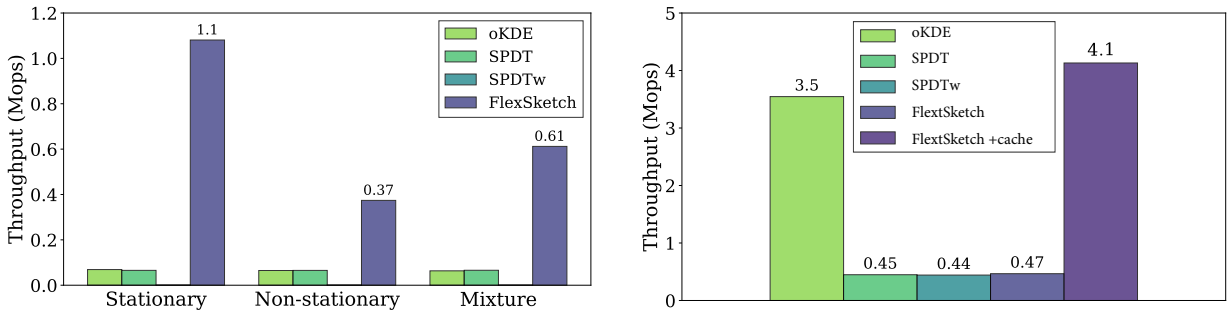
$$\sigma = \dot{\delta}(t_1) \simeq \frac{\delta(t_1 + t_\epsilon) - \delta(t_1)}{t_\epsilon}. \tag{15}$$

Memory Usage The PDF estimated using the density estimation algorithm continues to use memory. After the estimation, this result or its changing history is recorded in the disk if necessary. Therefore, we record only the memory usage of the estimated PDF, but not the whole memory usage consumed by the UPDATE or QUERY operation.

4.4. Throughput

We compare the throughput performance of the existing and our methods for the two key operations, i.e., updating the estimated density and producing the probability for a given data, which correspond to the UPDATE and QUERY operations in our method, respectively.

UPDATE Figure 4a shows the throughputs of the update operation of different density estimation algorithms for different types of data streams. The throughput of FlexSketch is 1.1 Mops, which is 16×, 16×, and 1800× higher than that of oKDE, SPDT, and SPDTw, respectively, for the stationary data streams. For the non-stationary data streams, the throughput of FlexSketch is 0.37 Mops, which is 5.8×, 5.7×, and 570× higher than that of oKDE, SPDT, and SPDTw, respectively. For the mixture data streams, the throughput of FlexSketch is 0.61 Mops, which is 9.7×, 9.3×, and 1000× higher than that of oKDE, SPDT, and SPDTw, respectively. We also perform the one-sample Wilcoxon signed-rank test under the hypothesis that the median of the throughput differences between the proposed method and the existing methods is zero, which confirms the significance of the differences ($p < 0.005$). This result demonstrates that the additional computation time to manage multiple models is significantly smaller than the computation time to represent the data stream elaborately. This effect becomes more prominent when major concept drift does not occur. However, it shows a noticeable improvement even for a data stream with frequent major concept drift.



(**a**) Update operation (**b**) Query operation

Figure 4. Throughput performance of the update and query operation (a higher value is better).

QUERY Figure 4b shows the throughputs of different density estimation algorithms for the query operation. The throughput of FlexSketch is 0.47 Mops, which is similar to that of SPDT and SPDTw and smaller than that of oKDE. As shown in (7), FlexSketch linearly combines multiple models for each query. In order to improve the querying speed, we can add a caching algorithm, although it consumes 20–30% more memory. The throughput of FlexSketch is significantly improved with cache up to 4.1 Mops, which is 1.2×, 9.2×, and 9.4× higher than that of oKDE, SPDT, and SPDTw, respectively.

4.5. Accuracy (Error and Adaptability)

We compare the accuracy of FlexSketch with that of the alternatives by measuring errors for the stationary and non-stationary data streams.

4.5.1. Stationary Case

Figure 5 compares the estimation error of each algorithm after performing the update operation for three different stationary data streams. The error of FlexSketch for the normal distribution is 0.012, which is 0.75×, 1.2×, and 3.2× less than those of oKDE, SPDT, and SPDTw, respectively. It is intuitive that oKDE records the lowest error because it estimates the distribution by using a mixture of Gaussian distributions. SPDTw is less accurate than FlexSketch since the number of data used for update by SPDTw is limited to a fixed size within its window (note that the parameters of SPDTw are deliberately selected so as to have similar accuracy as FlexSketch when concept drift occurs, as mentioned in Section 4.2).

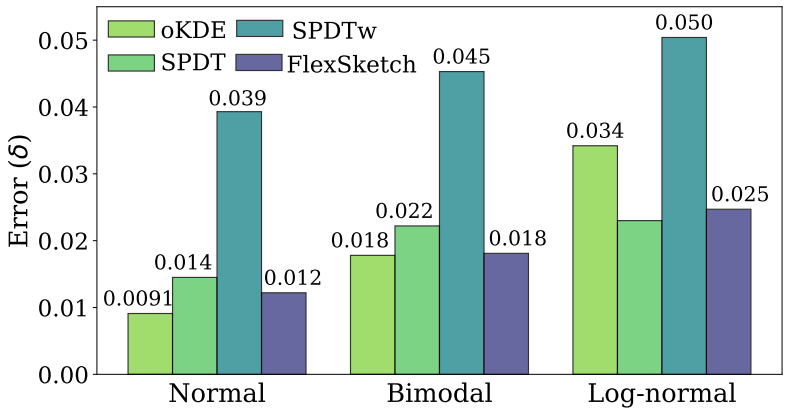


Figure 5. Estimation error (δ) for stationary data streams (a smaller value is better).

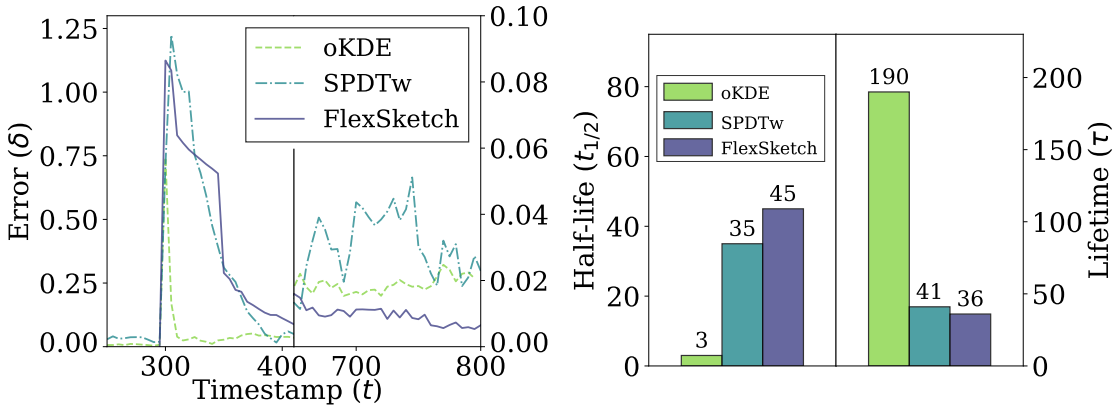
The error of FlexSketch for the bimodal distribution is 0.018, which is 0.98×, 1.2×, and 2.5× smaller than those of oKDE, SPDT, and SPDTw, respectively. oKDE using a

Gaussian kernel shows the best result, as in the case of the normal distribution. And, the performance of FlexSketch is equal to that of oKDE within a margin of error. This indicates that the BUILD operation successfully constructs a new model that recognizes different modes well. Again, SPDTw is less accurate than FlexSketch for the aforementioned reason.

The error of FlexSketch for the log-normal distribution is 0.025, which is 1.4×, 0.93×, and 2.0× smaller than those of oKDE, SPDT, and SPDTw, respectively. Contrary to the results for the normal and bimodal distributions, FlexSketch and SPDT, which have high degrees of freedom, show smaller errors than oKDE.

4.5.2. Non-Stationary Case

Figure 6 shows the error and adaptability performance of different methods under sudden concept drift, i.e., the errors over time between the PDFs estimated using different algorithms and the underlying distribution of the data stream in Figure 6a, and the half-life and lifetime in Figure 6b. When the concept drift occurs at $t = 300$, the errors jump to 1.0 or higher for all methods. As soon as the PDFs adapts to the new concept, the PDF forgets the old concept and the errors slowly fall to zero. In the short term, the error of oKDE decreases more quickly compared to SPDTw and FlexSketch, resulting in the shortest half-life by oKDE. However, oKDE shows the longest lifetime, indicating that it is affected by the old concept for a long time. In the long term, FlexSketch shows the smallest error in Figure 6a and also the shortest lifetime in Figure 6b.

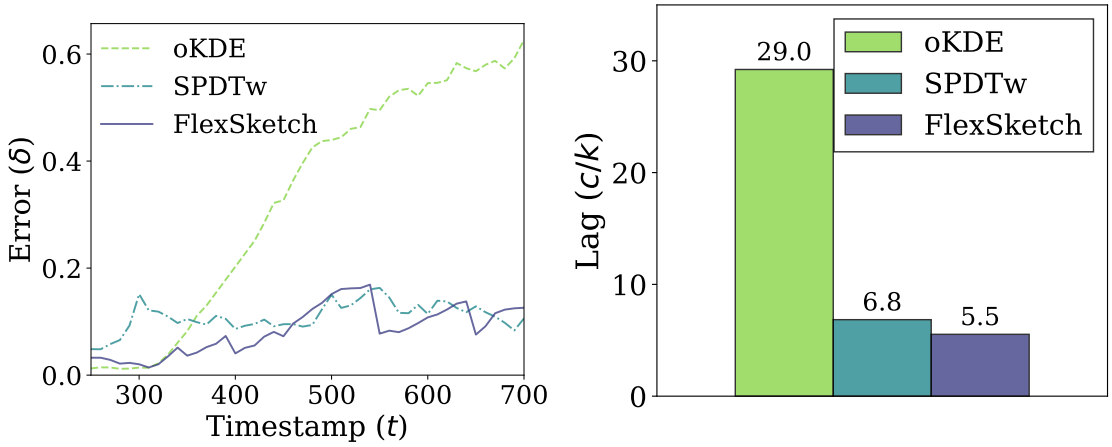


(**a**) (**b**)

Figure 6. Error and adaptability performance under sudden concept drift. (**a**) Change of the error over time. The period between about 400 and 650 is omitted for better visualization, and the error after $t = 650$ is magnified with the scale at the right side. (**b**) Half-life (left side) and lifetime (right side). A smaller value is better.

We also measure the accuracy of the three methods over data generated by the MOA framework [56] for sudden concept change. Function 2 and 3 of the SEA generator [57] is used with a narrow drift-window (100 samples) to produce data streams and the error is measured against the mean of each cluster at the 2000 sample point after the drift. FlexSketch, SPDTw and oKDE exhibit errors of 0.27, 0.21 and 0.57, respectively. Though the overall trend is similar, FlexSketch shows a slightly higher error than SPDTw since the window size of SPDTw is small enough to evade from the effect of old data.

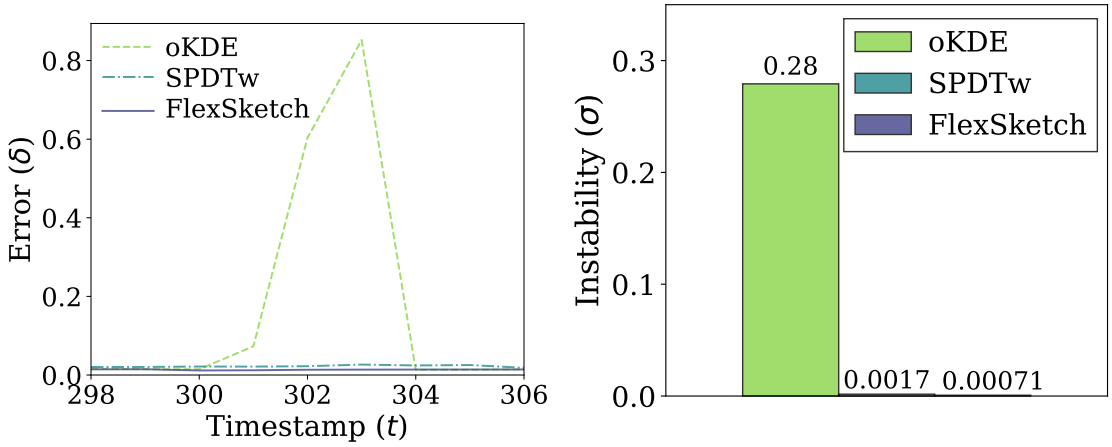
In Figure 7, the performance of different methods under incremental drift is shown. The error of FlexSketch is saturated at 0.11, while those of SPDTw and oKDE are saturated at 0.14 and 0.58, respectively, as shown in Figure 7a. In addition, Figure 7b shows that FlexSketch has the smallest lag (5.3× and 1.2× smaller than those of oKDE and SPDTw, respectively). This demonstrates that FlexSketch can not only speed up computation, but also adapt more accurately to the changes in the data stream. This is also consistent with the observation for sudden concept drift that FlexSketch forgets the past concept faster than oKDE in the long term.



(**a**) (**b**)

Figure 7. Error and adaptability performance under incremental concept drift. (**a**) Change of the error over time. (**b**) Lag (a smaller value is better).

Figure 8a shows the errors of the three algorithms for the case of blip concept drift. The errors of FlexSketch and SPDTw increase only by 0.0021 and 0.0050, respectively, due to the blip concept drift, whereas oKDE shows a much larger increase of the error (up to 0.84). The comparison of the instability metric in Figure 8b also confirms that FlexSketch shows significantly improved performance, especially compared to oKDE.



(**a**) (**b**)

Figure 8. Error and adaptability performance under blip concept drift. (**a**) Change of the error over time. (**b**) Instability (a smaller value is better).

4.6. Memory Usage

Figure 9 compares the amount of memory used in the data structures of different density estimation algorithms for the stationary data stream. Our FlexSketch consumes 6.2 kbytes of memory, which is 1.3×, 1.4× and 0.68× more than that of oKDE, SPDT, and SPDTw, respectively. Since we set $N_M = 3$, one could expect that FlexSketch requires 3× more memory usage than the others. However, the increased amount of memory consumption is much less than such an expectation by using efficient histogram computation.

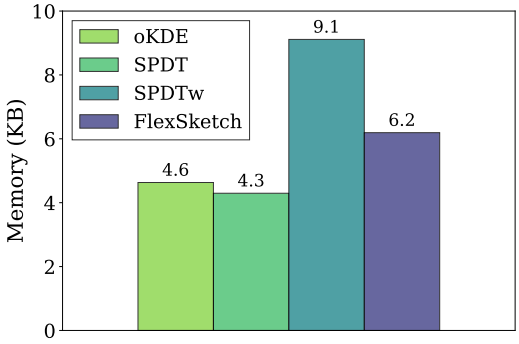


Figure 9. Comparison of memory usage.

4.7. Effects of Parameters

We investigate the effects of the algorithm parameters, i.e., N_M, N_Q, λ, and γ, on the performance in terms of throughput of the UPDATE operation, error, and memory usage for the stationary data stream and the non-stationary data stream with incremental concept drift. The ranges of the parameters are as follows: 2 to 10 for N_M, 10 to 150 for N_Q, 0.2 to 3.0 for λ, and 0.01 to 2.3 for γ. Experimental results with the combinations of these parameter values are analyzed below.

4.7.1. Stationary Case

Figure 10a is a three-dimensional representation of the throughput, error, and memory usage for the stationary data stream as the parameters are changed. Many of the data points are located in the upper left side, indicating that FlexSketch achieves high throughputs and low errors over various combinations of the parameter values.

Figure 11 presents the effect of each parameter separately by increasing one of the four parameters while the others remain fixed. The following observations can be made. First, as N_M increases, the throughput tends to decrease because of increased computational complexity for more histograms, while the error does not change (Figure 11a). Second, increasing N_Q and γ results in decreased errors and increased throughputs in Figure 11b,d, respectively. The improved throughput is because using a larger Q, or increasing the threshold γ allows the computationally intensive MAJORUPDATE operation to be performed less frequently. Since the MAJORUPDATE operation adds a model representing the latest data to the data structure, performing less MAJORUPDATE operations reduces the dependence on the latest data, which improves the accuracy for the stationary data stream. Third, the value of λ does not affect much on the performance (Figure 11c).

4.7.2. Non-Stationary Case

Figure 10b shows the throughput, error, and memory usage for the non-stationary data stream with incremental concept drift as the parameters are changed. Depending on the values of the parameters, the performance of FlexSketch may become degraded (i.e., larger errors, lower throughput, or larger memory consumption).

Figure 12 shows how each parameter affects to the performance, from which we draw the following observations. First, as shown in Figure 12a, increasing N_M results in lowering the throughput without changing the error much, which is due to the increased number of histograms as in the case of the stationary data stream. Second, when N_Q or γ increases, a trade-off relationship is observed, i.e., the throughput increases but the error also increases (Figure 12b,d). Suppressing the MAJORUPDATE operation with increased N_Q or γ improves the throughput, but prevents FlexSketch from accurately adapting to the concept drift. Third, by increasing λ, the contribution of past data is reduced and thus the error can be reduced.

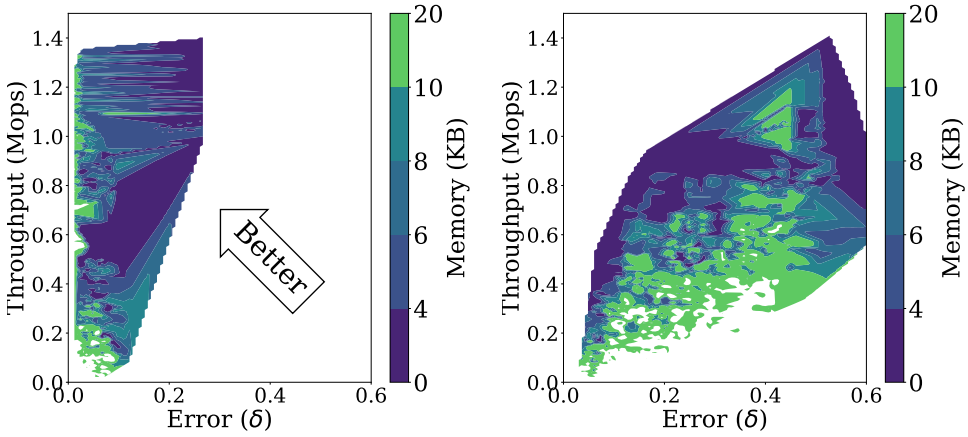


Figure 10. Performance of FlexSketch for various combinations of parameters' values. (**a**) Stationary data stream. (**b**) Non-stationary data stream.

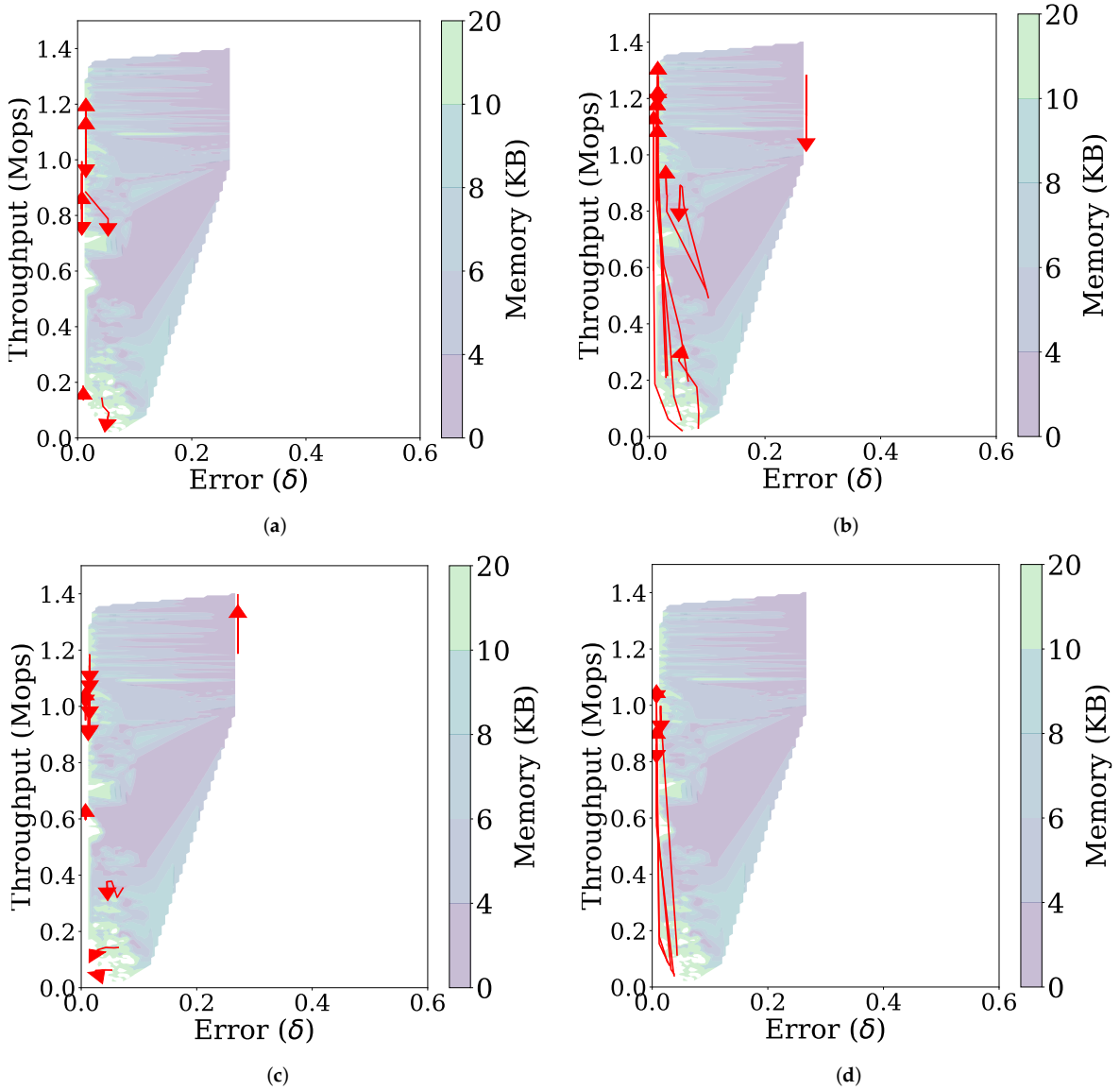


Figure 11. Trajectories (represented by red arrows) of the performance of FlexSketch for the stationary data stream as the value of each parameter increases (**a**) N_M (**b**) N_Q (**c**) λ (**d**) γ.

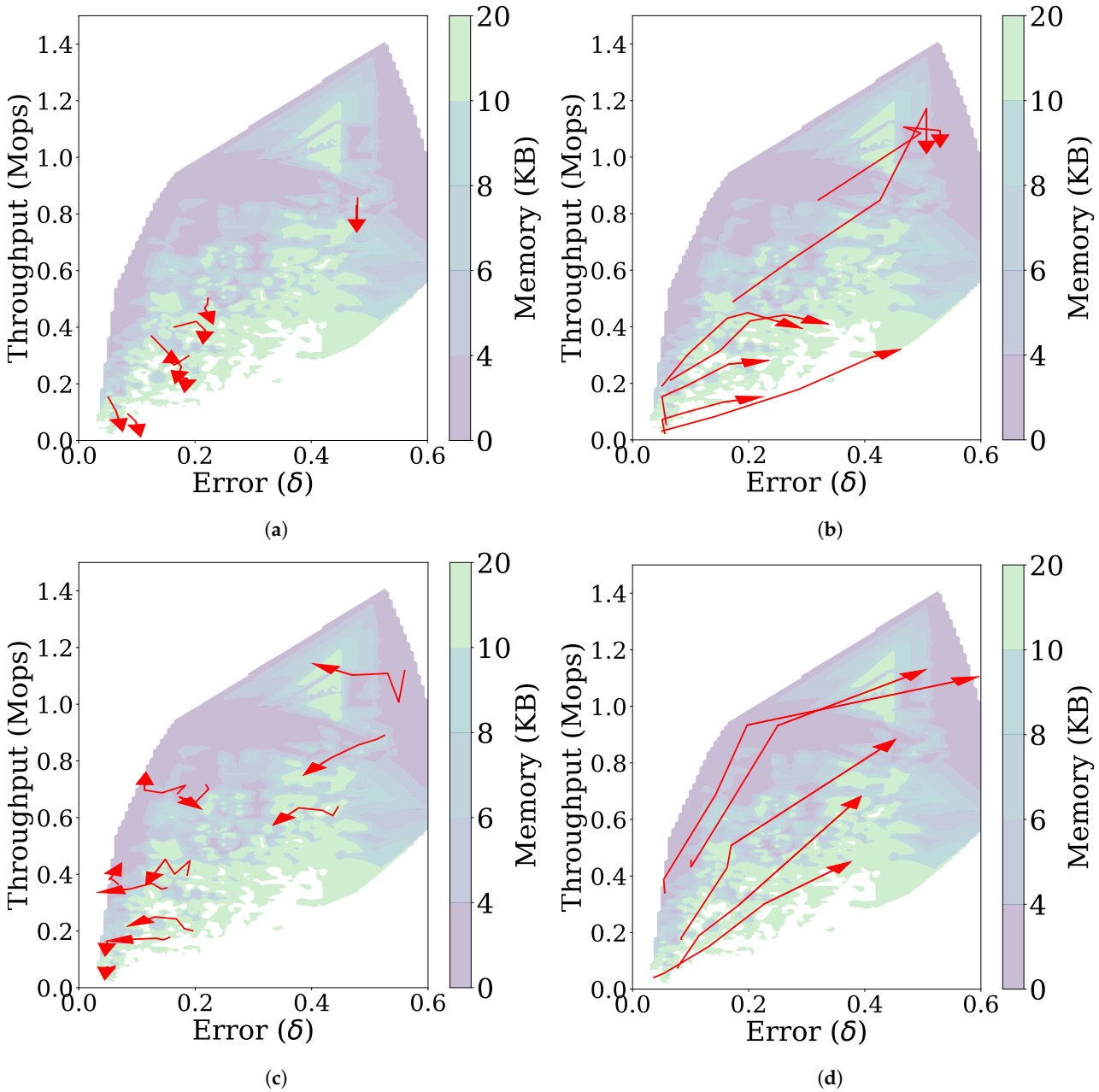


Figure 12. Trajectories (represented by red arrows) of the performance of FlexSketch for the non-stationary data stream with incremental concept drift as the value of each parameter increases. (**a**) N_M (**b**) N_Q (**c**) λ (**d**) γ.

5. Conclusions

In this paper, we have proposed the FlexSketch framework, which is an online algorithm based on an ensemble of histograms and consists of three operations: MINORUPDATE, MAJORUPDATE, and DIAGNOSE. Since it dynamically determines when to forget old data by observing divergence, it estimates probability distributions stably for stationary data streams without invoking the MAJORUPDATE operation. FlexSketch adapts to concept drift swiftly for non-stationary data streams by updating underlying model rapidly using MAJORUPDATE. As shown in Section 4.5.2, FlexSketch estimates probability distribution with high accuracy for data streams with sudden and incremental concept drift. Because FlexSketch utilizse simple histogram as the elemental data structure, it achieves high throughput update and query operations using only limited memory. The experimental results demonstrated the advantages of the method we propose in this paper. FlexSketch exhibits significantly improved speed compared to its alternatives. Moreover, FlexSketch adapts

well to various non-stationary data streams while maintaining stability over temporal fluctuations. Nevertheless, FlexSketch has a disadvantage since it has multiple parameters. As discussed in Section 4.7, FlexSketch exposes some changes in throughput and accuracy according to parameters, which could be a burden of design choice in domain specific applications. While FlexSketch exhibits preferable characteristics, it needs to be extended to overcome the current limitation of supporting one-dimensional data only, which could be a drawback for some applications. In our future work, we plan to extend our method for multi-dimensional data streams. Because histogram is a simple and efficient underlying data structure for ensemble methods as shown in this paper, we will try to incorporate multi-dimensional histogram [58,59] to accommodate multi-dimensional data. In addition, we will explore applications that utilize probability estimation as a core building block. Drifting data stream classification [35] and anomaly detection in non-stationary data stream [60] would be good candidates to deploy FlexSketch for practical applications.

Author Contributions: Data curation, N.P.; Funding acquisition, S.K.; Investigation, S.K.; Project administration, S.K.; Software, N.P.; Writing—original draft, N.P.; Writing—review and editing, S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Samsung Science and Technology Foundation under Project Number SSTF-BA1501-52 and Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1801-10.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kraska, T.; Beutel, A.; Chi, E.H.; Dean, J.; Polyzotis, N. The case for learned index structures. In Proceedings of the International Conference on Management of Data, Houston, TX, USA, 10–15 June 2018; pp. 489–504.
2. Ustinova, E.; Lempitsky, V. Learning deep embeddings with histogram loss. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 4170–4178.
3. Geng, Y.; Liu, S.; Yin, Z.; Naik, A.; Prabhakar, B.; Rosenblum, M.; Vahdat, A. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, Renton, WA, USA, 9–11 April 2018; pp. 81–94.
4. Webb, G.I.; Hyde, R.; Cao, H.; Nguyen, H.L.; Petitjean, F. Characterizing concept drift. *Data Min. Knowl. Discov.* **2016**, *30*, 964–994. [CrossRef]
5. Ahmad, S.; Lavin, A.; Purdy, S.; Agha, Z. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* **2017**, *262*, 134–147. [CrossRef]
6. Cheng, K.W.; Chen, Y.T.; Fang, W.H. Video anomaly detection and localization using hierarchical feature representation and Gaussian process regression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 2909–2917.
7. Yang, D.; Li, B.; Rettig, L.; Cudré-Mauroux, P. HistoSketch: Fast similarity-preserving sketching of streaming histograms with concept drift. In Proceedings of the IEEE International Conference on Data Mining, New Orleans, LA, USA, 18–21 November 2017; pp. 545–554.
8. Ben-Haim, Y.; Tom-Tov, E. A streaming parallel decision tree algorithm. *J. Mach. Learn. Res.* **2010**, *11*, 849–872.
9. Kristan, M.; Leonardis, A.; Skočaj, D. Multivariate online kernel density estimation with Gaussian kernels. *Pattern Recognit.* **2011**, *44*, 2630–2642. [CrossRef]
10. Heinz, C.; Seeger, B. Towards kernel density estimation over streaming data. In Proceedings of the International Conference on Management of Data, Chicago, IL, USA, 27–29 June 2006; pp. 1–12.
11. Qahtan, A.A.; Wang, S.; Zhang, X. KDE-Track: An efficient dynamic density estimator for data streams. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 642–655. [CrossRef]
12. Hill, D.J.; Minsker, B.S. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environ. Model. Softw.* **2010**, *25*, 1014–1022. [CrossRef]
13. Wu, C.; Jiang, P.; Ding, C.; Feng, F.; Chen, T. Intelligent fault diagnosis of rotating machinery based on one-dimensional convolutional neural network. *Comput. Ind.* **2019**, *108*, 53–61. [CrossRef]
14. Wang, J.; Yang, X.; Long, K. A new relative entropy based app-DDoS detection method. In Proceedings of the IEEE Symposium on Computers and Communications, Riccione, Italy, 22–25 June 2010; pp. 966–968.
15. Wilson, A.G.; Gilboa, E.; Nehorai, A.; Cunningham, J.P. Fast kernel learning for multidimensional pattern extrapolation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)—Volume 2*; MIT Press: Cambridge, MA, USA, 2014; pp. 3626–3634.

16. Pham, D.S.; Venkatesh, S.; Lazarescu, M.; Budhaditya, S. Anomaly detection in large-scale data stream networks. *Data Min. Knowl. Discov.* **2014**, *28*, 145–189. [CrossRef]
17. Gama, J.A.; Žliobaitundefined, I.; Bifet, A.; Pechenizkiy, M.; Bouchachia, A. A survey on concept drift adaptation. *ACM Comput. Surv.* **2014**, *46*, 1–37. [CrossRef]
18. Bifet, A.; Holmes, G.; Pfahringer, B.; Kirkby, R.; Gavaldà, R. New ensemble methods for evolving data streams. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 139–148.
19. Bifet, A.; Holmes, G.; Pfahringer, B. Leveraging bagging for evolving data streams. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Barcelona, Spain, 19–23 September 2010; pp. 135–150.
20. Gomes, H.M.; Bifet, A.; Read, J.; Barddal, J.P.; Enembreck, F.; Pfharinger, B.; Holmes, G.; Abdessalem, T. Adaptive random forests for evolving data stream classification. *Mach. Learn.* **2017**, *106*, 1469–1495. [CrossRef]
21. Cano, A.; Krawczyk, B. Kappa updated ensemble for drifting data stream mining. *Mach. Learn.* **2020**, *109*, 175–218. [CrossRef]
22. Klinkenberg, R.; Joachims, T. Detecting concept drift with support vector machines. In Proceedings of the International Conference on Machine Learning, Stanford, CA, USA, 29 June–2 July 2000; pp. 487–494.
23. Li, B.; Wang, Y.J.; Yang, D.S.; Li, Y.M.; Ma, X.K. FAAD: An unsupervised fast and accurate anomaly detection method for a multi-dimensional sequence over data stream. *Front. Inf. Technol. Electron. Eng.* **2019**, *20*, 388–404. [CrossRef]
24. Bashir, S.; Petrovski, A.; Doolan, D. A framework for unsupervised change detection in activity recognition. *Int. J. Pervasive Comput. Commun.* **2017**, *13*, 157–175. [CrossRef]
25. Sethi, T.; Kantardzic, M. Handling adversarial concept drift in streaming data. *Expert Syst. Appl.* **2018**, *97*, 18–40. [CrossRef]
26. Costa, A.F.J.; Albuquerque, R.A.S.; dos Santos, E.M. A drift detection method based on active learning. In Proceedings of the International Joint Conference on Neural Networks, Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.
27. Koh, Y.S. CD-TDS: Change detection in transactional data streams for frequent pattern mining. In Proceedings of the International Joint Conference on Neural Networks, Vancouver, BC, Canada, 24–29 July 2016; pp. 1554–1561.
28. De Mello, R.; Vaz, Y.; Grossi, C.; Bifet, A. On learning guarantees to unsupervised concept drift detection on data streams. *Expert Syst. Appl.* **2019**, *117*, 90–102. [CrossRef]
29. Pinagé, F.; dos Santos, E.M.; Gama, J. A drift detection method based on dynamic classifier selection. *Data Min. Knowl. Discov.* **2019**, *34*, 50–74. [CrossRef]
30. Bouchachia, A. Fuzzy classification in dynamic environments. *Soft Comput.* **2011**, *15*, 1009–1022. [CrossRef]
31. Gomes, J.A.B.; Menasalvas, E.; Sousa, P.A.C. Learning recurring concepts from data streams with a context-aware ensemble. In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC'11), Taichung, Taiwan , 21–24 March 2011; Association for Computing Machinery: New York, NY, USA, 2011; pp. 994–999.
32. Adä, I.; Berthold, M.R. EVE: A framework for event detection. *Evol. Syst.* **2013**, *4*, 61–70. [CrossRef]
33. Vorburger, P.; Bernstein, A. Entropy-based concept shift detection. In Proceedings of the International Conference on Data Mining (ICDM'06), Hong Kong, China, 18–22 December 2006; pp. 1113–1118.
34. Gözüaçık, O.; Büyükçakır, A.; Bonab, H.; Can, F. Unsupervised concept drift detection with a discriminative classifier. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM'19), Beijing, China, 3–7 November 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 2365–2368.
35. Wang, X.; Kang, Q.; An, J.; Zhou, M. Drifted Twitter spam classification using multiscale detection test on K-L divergence. *IEEE Access* **2019**, *7*, 108384–108394. [CrossRef]
36. Prabhu, S.S.; Runger, G.C. Designing a multivariate EWMA control chart. *J. Qual. Technol.* **1997**, *29*, 8–15. [CrossRef]
37. Koren, Y. Collaborative filtering with temporal dynamics. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 447–456.
38. Pechenizkiy, M.; Bakker, J.; Žliobaitė, I.; Ivannikov, A.; Kärkkäinen, T. Online mass flow prediction in CFB boilers with explicit detection of sudden concept drift. *ACM SIGKDD Explor. Newsl.* **2010**, *11*, 109–116. [CrossRef]
39. Forman, G. Tackling concept drift by temporal inductive transfer. In Proceedings of the 29th ACM Conference on Research and Development in Information Retrieval, Seattle, WA, USA, 6–11 August 2006, pp. 252–259.
40. Gilbert, A.C.; Guha, S.; Indyk, P.; Kotidis, Y.; Muthukrishnan, S.; Strauss, M.J. Fast, small-space algorithms for approximate histogram maintenance. In Proceedings of the Annual ACM Symposium on Theory of Computing, Montreal, QC, Canada, 19–21 May 2002; pp. 389–398.
41. Guha, S.; Koudas, N.; Shim, K. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.* **2006**, *31*, 396–438. [CrossRef]
42. Greenwald, M.; Khanna, S.; Greenwald, M.; Khanna, S. Space-efficient online computation of quantile summaries. *ACM SIGMOD Rec.* **2001**, *30*, 58–66. [CrossRef]
43. Shrivastava, N.; Buragohain, C.; Agrawal, D.; Suri, S. Medians and beyond: New aggregation techniques for sensor networks. In Proceedings of the International Conference on Embedded Network Sensor Systems, Baltimore, MD, USA, 3–5 November 2004; pp. 239–249.
44. Cormode, G.; Korn, F.; Muthukrishnan, S.; Srivastava, D. Effective computation of biased quantiles over data streams. In Proceedings of the International Conference on Data Engineering, Tokoyo, Japan, 5–8 April 2005; pp. 1–12.

45. Singh, S.A.; Srivastava, D.; Tirthapura, S. Estimating quantiles from the union of historical and streaming data. In Proceedings of the VLDB Endowment, New Delhi, India, 9 May 2016; Volume 10, pp. 433–444.
46. Datar, M.; Gionis, A.; Indyk, P.; Motwani, R. Maintaining stream statistics over sliding windows. *SIAM J. Comput.* **2002**, *31*, 1794–1813. [CrossRef]
47. Kuncheva, L.I.; Žliobaitė, I. On the window size for classification in changing environments. *Intell. Data Anal.* **2009**, *13*, 861–872. [CrossRef]
48. Deypir, M.; Sadreddini, M.H.; Hashemi, S. Towards a variable size sliding window model for frequent itemset mining over data streams. *Comput. Ind. Eng.* **2012**, *63*, 161–172. [CrossRef]
49. Kolter, J.Z.; Maloof, M.A. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.* **2007**, *8*, 2755–2790.
50. Elwell, R.; Polikar, R. Incremental learning of concept drift in nonstationary environments. *IEEE Trans. Neural Netw.* **2011**, *22*, 1517–1531. [CrossRef] [PubMed]
51. Gomes, H.M.; Barddal, J.P.; Enembreck, F.; Bifet, A. A survey on ensemble learning for data stream classification. *ACM Comput. Surv.* **2017**, *50*, 1–36. [CrossRef]
52. Oza, N.C. Online bagging and boosting. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, HI, USA, 12 October 2005; Volume 3, pp. 2340–2345.
53. Source Codes of FlexSketch, October 2020. Available online: https://xxxnell.github.io/flex/docs/core/sketch.html (accessed on 4 February 2021).
54. Source Codes of Online Kernel Density Estimation. June 2017. Available online: https://github.com/joluet/okde-java (accessed on 4 February 2021).
55. Source Codes of Streaming Parallel Decision Tree. June 2017. Available online: https://github.com/soundcloud/spdt (accessed on 4 February 2021).
56. Bifet, A.; Holmes, G.; Kirkby, R.; Pfahringer, B. MOA: Massive online analysis. *J. Mach. Learn. Res.* **2010**, *11*, 1601–1604.
57. Street, N.; Kim, Y. A streaming ensemble algorithm (SEA) for large-scale classification. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 26–29 August 2001; pp. 377–382.
58. Thaper, N.; Guha, S.; Indyk, P.; Koudas, N. Dynamic multidimensional histograms. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD '02), Madison, WI, USA, 4–6 June 2002; Association for Computing Machinery: New York, NY, USA, 2002; pp. 428–439.
59. Diakonikolas, I.; Kane, D.M.; Peebles, J. Testing identity of multidimensional histograms. In Proceedings of the Conference on Learning Theory (PMLR), Phoenix, AZ, USA, 25–28 June 2019; pp. 1107–1131.
60. Jordaney, R.; Sharad, K.; Dash, S.K.; Wang, Z.; Papini, D.; Nouretdinov, I.; Cavallaro, L. Transcend: Detecting concept drift in malware classification models. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; USENIX Association: Vancouver, BC, Canada, 2017; pp. 625–642.