Clustering and sketching techniques for uncertain data streams

COMP5112 Fall 2020 Project Paper

Patrick Killeen, pkill013@uottawa.ca , November 30 2020

Abstract

Data mining and stream processing have become important fields of research in recent years. Many applications can make use of the knowledge discovered from mining data streams, and such applications include stock market analysis, weather predictions, social media analysis, sensor networks, and intrusion detection systems. Applying classical data mining techniques (clustering, for example) to data streams in an efficient manner is difficult, since a) distributed data sources are being deployed in increasing numbers and size, b) the dimension of time must be considered, and c) the data sources may be noisy and inaccurate. Data stream processing techniques include sketching, sliding windows, and uncertain data stream processing, where the latter is used to process fuzzy/noisy data streams by addressing the uncertainty of the data in an efficient manner. Various types of data stream processing (count-min sketch, for example), clustering (k-means, for example), and uncertain data stream clustering algorithms (FDBSCAN, for example) are explored in the present work, along with their complexity and other important details.

1 INTRODUCTION

Data mining and stream processing has become an important field of research in recent years. Distributed data sources are being deployed in increasing numbers (for example, sensor networks), which increases the global volume of data being created and the number of streams to be processed, and the speed at which the data is being produced by the data sources is also increasing (higher frequency sensor readings, for example). This makes it difficult to apply classical data mining techniques efficiently [23]. Furthermore, value can also be found from processing these data sources, enabling knowledge discovery that may help analysts make educated business intelligence decisions that were not previously possible. Stream processing applications include stock market analysis, weather information aggregation, sensor network analysis, and network intrusion detection [26].

1.1 Problem Statement

Novel challenges arise when applying classical data mining techniques to data streams. These techniques (for example, clustering) are struggling to efficiently process the large number of distributed data streams, since these classical data mining techniques were designed for static datasets, while data streams are dynamic in nature. Static datasets have a fixed number of data instances, while the number of elements in data streams is unbounded. A new dimension must also be consider in data stream processing algorithms; that is, the dimension of time, where older elements should be weighted with less importance compared to newer instances [26][12]. Furthermore, there are usually time and resource constraints in the domain of stream processing, which means one-pass algorithms and efficient techniques will be required to summarize statistics over a stream [23][18].

Consider an example stream summary problem that requires storing all the elements of the stream in memory. Solving such a problem naively may not be possible, since the stream could be unbounded or there may be resource constraints that make it impossible to store the entire stream in memory.

1.2 Motivation

To address the challenges of stream processing, a variety of **data mining techniques** are used, which include sketch techniques, stream clustering, and uncertain data stream clustering.

- Sketch techniques are used in practice to approximate summaries of streams in a quick and efficient manner [8].
- Data stream clustering, an important subset of the clustering problem, attempts to group similar objects that evolve over time in a stream.
- Uncertain data stream clustering combines notions from both clustering and stream processing. In uncertain data streams, data instances are fuzzy (have noise and inaccuracies) and have a probabilistic nature. Clustering these objects together requires probability distribution comparison techniques instead of using standard distance metrics (such as the Euclidean distance, for example) [2][14].

Detailed **application** examples of uncertain data stream clustering follow:

- Electing a cluster leader of nearby wireless devices [24]. For example, a mobile phone may gather sensor data from its nearby peers via short-range wireless protocols. To elect a leader, location data is required, which may take the form of uncertain data streams, since the sensor accuracy and GPS noise add an element of uncertainty to the GPS data streams. By electing a leader using uncertain data stream clustering, the peers in a cluster can save bandwidth by avoiding unnecessary cellular network communicate, which is achieved by only communicating with a nearby leader over a cheaper wireless medium (such as Bluetooth or Wi-Fi, for example).
- Clustering moving objects, where the underlying state of the object changes (for example, ambient temperatures or sensor reading databases), where the data is uncertain and the distribution is changing over time [20][24][7].
- At times, the data stored in the stream data structure is not 100% accurate. This could be the case for the following reasons: a) in a security domain, for privacy reasons, noise is added to the data to keep the data sources anonymous, or b) for bandwidth limitation reasons [20]. Clustering these types of noisy data structures could be done using uncertain data stream clustering algorithms.
- An application of querying uncertain data is obtaining the actual temperature reported by various sensors. Since the temperature may change, querying the data from the last sensor reading may not represent the actual temperature at the time of the query [2][7]. Probability density functions can also be used to represent sensor data to enable querying uncertain data from temperature sensors [7].

Unfortunately, uncertainty complicates data stream clustering [6]. Traditional clustering typically involves unsupervised machine learning and grouping similar objects together in a heuristic fashion by using distance metric to compare data objects. Uncertain data clustering uses a metric to compare probability density/mass functions to compare distances between objects. As such, a naive approach for clustering uncertain data will require many more distance computations compared to traditional clustering. Therefore, further research in the area of uncertain data stream clustering is required for designing clever optimizations and efficient approximations [24].

1.3 Paper Overview

This paper is structured as follows: section 2 contains some background, related work, and key mathematical definitions to help a reader understand the data mining algorithms discussed in next sections; section 3 presents a detailed review of algorithms related to clustering, stream processing, and uncertain data stream clustering, which may include pseudo-code, complexity analysis, and algorithm correctness; section 4 provides question and answer pairs related to the algorithms discussed in the present work; and section 5 concludes the present work with a summary of the literature discussed in the present work and some future work that could be done to improve the present work.

2 BACKGROUND AND RELATED WORK

This section discusses some necessary background required to present algorithms for sketching, clustering, stream processing, uncertain data stream clustering, and other related data mining algorithms.

2.1 Definitions

Some general mathematical concepts related to data mining are defined in this section. Mathematical definitions that are strongly related to topics presented later in the present work are not presented in this section, but rather presented in the appropriate later section.

Definition 1. *2-universal hash functions*

According to [19], let $H = \{h_i\}_{i=1}^{k}$ define a hash family of k hash functions, where D is the input domain of each hash function and $x \in D$, $h_i(x) \rightarrow \{0, 1, ..., n-1\}\ \forall\ i \in \{1, 2, ..., k\}$; that is, hashing an element x will produce an integer between 0 and $n-1$. H is said to be 2-universal if $\forall x, y \in D$,

$$\Pr(h(x) = h(y)) \leq \frac{1}{n}$$

, where $x \neq y$ and h is sampled from H uniformly at random.

Definition 2. *Metric*

According to [3], given a set X, a metric is a function d: $X \times X \rightarrow \mathbb{R}$ such that, for any $x, y, z \in X$, we have:

$$\begin{aligned} Non-negativity:\ & d(x,y) \geq 0 && (a)\\ Identity\ of\ indiscernibles:\ & d(x,y) = 0 \Leftrightarrow x = y && (b)\\ Symmetry:\ & d(x,y) = d(y,x) && (c)\\ Triangle\ inequality:\ & d(x,y) \leq d(x,y) + d(z,y) && (d) \end{aligned}$$

Definition 3. *Sum of squares error - SSE*

According to [11], the SSE is defined as:

$$\text{SSE} = \frac{1}{2}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

, where n is the number of instances, x_i is the value of the ith data instance, and $\bar{x}$ is the mean of the data instances.

2.2 Distance and Divergence Measures

This sections presents a few important distance and divergence measures related to comparing two probability distributions. According to [3], in stream processing, it is important to compare two different data streams together using a distance or divergence measure. The Kullback-Leibler-divergence and Hellinger distance are popular examples of such measures. Furthermore, there are two classes of measures, namely, the f-divergences and the Bregman divergences.

2.2.1 f-Divergence. According to [3], the f-divergence class of measure is defined as follows: let p and q be two distributions with the same number of points and let f:$(0, \infty) \rightarrow \mathbb{R}$ denote a complex function, such that $f(1) = 0$. The f-divergence of q from p is:

$$\mathcal{D}_f(p) = \sum_{i \in \Omega} q_i f\left(\frac{p_i}{q_i}\right)$$

, where Ω is the domain of points, and q_i and p_i are the ith point of q and p, respectively. Also, by convention $0f\left(\frac{0}{0}\right) = 0$, $\alpha f\left(\frac{0}{\alpha}\right) = \alpha \lim_{u \rightarrow 0} f(u)$, and $0f\left(\frac{a}{0}\right) = \alpha \lim_{u \rightarrow \infty} f(u)/u$ if these limits exist.

2.2.2 Kullback-Leibler Divergence. According to [14], the Kullback-Leibler Divergence is a measure to compare two probability distributions and is defined in equation 1:

$$D(f \parallel g) = \sum_{x \in Y} f(x) \log \frac{f(x)}{g(x)} \tag{1}$$

, where f and g are probability mass functions in a discrete domain Y. In the case where f and g are probability mass functions and Y is a continuous domain, the KL-Divergence can be defined by equation 2:

$$D(f \parallel g) = \int_{Y} f(x) \log \frac{f(x)}{g(x)} \mathrm{d}x. \tag{2}$$

It is also the case that $D(P \parallel Q) \geq 0$, such that $D(P \parallel Q) = 0$ when $P = Q$, where P and Q are the objects with probability distribution functions f and g, respectively. As P and Q become more similar (as their probability distributions become more similar), $D(P \parallel Q)$ approaches 0.

In the case of a discrete domain Y, we can define the probability mass function of an object P as shown in equation 3:

$$P(x) = \frac{|p \in P | p = x|}{|P|} \tag{3}$$

, and by applying equation 3 with equation 1, we can compute the KL-Divergence of some object P and Q from domain Y as shown in equation 4:

$$D(P \parallel Q) = \sum_{x \in Y} P(x) \log \frac{P(x)}{Q(x)}. \tag{4}$$

In the case where Y is a continuous domain, we estimate $D(P \parallel Q)$ as shown in equation 5, assuming $P = \{p_1, p_2, ..., p_s\}$,

$$\hat{D}(P \parallel Q) = \frac{1}{s} \sum_{i=1}^{s} \log \frac{P(p_i)}{Q(p_i)}. \tag{5}$$

, where we assume that $x \in Y$ only when $P(x) > 0$, which also means $Q(x) > 0$. To make sure this assumption holds, smoothing can be used. We can smooth the probability distribution of P by applying equation 6:

$$\hat{P}(x) = \frac{P(x) + \delta}{1 + \delta |Y|} \tag{6}$$

, where $0 < \delta < 1$ is the smoothing parameter that can be used to adjust the desired accuracy of $\hat{P(x)}$, and $|Y|$ is the number of possible values in the domain Y (use the integral if Y is continuous). The error can be defined by equation 7:

$$|\hat{P}(x) - P(x)| = \left| \frac{1 - P(x)|Y|}{1/\delta + |Y|} \right| \in \left[0, \frac{\max\{1, |1 - |Y||\}}{1/\delta + |Y|} \right]. \tag{7}$$

2.2.3 Important distance measures. Some important and popular distance measures are defined in this section.

Definition 4. *Manhattan distance*

According to [16], the Manhattan distance can be defined as:

$$\text{Manhattan}\,(a,b) = \sum_{i=1}^{k} |a_i - b_i|$$

, where a and b are k-dimensional points. Note that Manhattan(a,b) may also be denoted as $\| a - b \|_1$, which is also known as the L_1 distance/norm. To help explain this distance measure, to illustrate the radius of a point p defined by the Manhattan distance, one could imagine a rectangle around p.

Definition 5. *L1 Norm*

According to [13], the L1 norm is defined as:

$$\text{L1 norm of the } k\text{-dimensional vector } a = \| a \|_1 = \sum_{i=1}^{k} |a_i|$$

, where a is a point in k-dimensional space. Note that $\| a \|_1$ = Manhattan$(a, 0)$, where 0 in this case is the point (0,0...,0).

Definition 6. *Euclidean distance*

According to [16], the Euclidean distance is defined as:

$$\text{Euclidean}\,(a,b) = \sqrt{\sum_{i=1}^{k} (a_i - b_i)^2}$$

, where a and b are k-dimensional points. Note that Euclidean(a,b) may also be denoted as $\| a - b \|_2$, which is also known as the L_2 distance/norm. To help explain this distance measure, to illustrate the radius of a point p using the Euclidean distance, one could imagine a circle around p.

Definition 7. *L2 Norm*

According to [13], the L2 norm is defined as:

$$\text{L2 norm of the } k\text{-dimensional vector } a = \| a \|_2 = \sqrt{\sum_{i=1}^{k} |a_i|^2}$$

, where a is a point in k-dimensional distance. Note that $\| a \|_2$ = Euclidean$(a, 0)$, where 0 in this case is the point (0,0...,0).

Definition 8. *L∞ Norm*

According to [13], the L∞ norm can be defined as:

$$\text{L}\infty \text{ norm of the } k\text{-dimensional vector } a = \| a \|_\infty = \max_{i \in \{1,2,\ldots,k\}} |a_i|$$

, where a is a point in k-dimensional space. The L∞ norm is the maximum element in the vector of a.

2.3 Data stream processing

Data stream processing typically involves performing a series of computations in one pass (only iterating over the elements once) on a large number of stream elements. There may be multiple distributed streams and real-time requirements involved, which makes data stream processing not a trivial problem to solve [18].

According to [23], one should consider the cost to transmit, store, and compute when analyzing and designing data stream processing algorithms. As data streams become large and the data rates increase, the streams will stress modern static data processing algorithms, since the datasets are dynamic in nature as opposed to the standard dataset such as a database, for example [26]. Therefore, efficient strategies for data stream processing must be considered, since applications of data streaming include real-time stream analysis of evolving streams (for example, stock exchange transactions). Communication is also an important consideration, since in the context of distributed data streaming, the data streams may be physically separated, requiring communication to aggregate streams together [26]. Some algorithms can be computed offline, but the real-time analysis requirements demand stream processing analytic algorithms to be efficient and quick. Stream processing may be considered from a hierarchical point of view, where the analytics and aggregations closer to the stream (from a latency point of view, starting from the data source) must be fast and efficient; while moving design logic away from the data sources that are producing the streams (up the data processing hierarchy with higher latency) will relax some of the real-time requirements and other constraints of the stream processing algorithms. [18] present the following **requirements** on stream processing systems:

- Data should keep moving
- It should be possible to query a data stream
- Delayed, missing, and out-of-order data should be handled
- Processing of a stream should be predictable and deterministic regardless of the time of data instance input
- It should be possible to aggregate/merge stored data with streaming data
- The safety and availability of the data should be ensured
- Applications should be automatically distributed and scaled with respect to the number and size of streams
- Real-time design must be considered

According to [26], the notion of time and data staleness must also be considered. **Forgetting mechanisms** (or time-decay models) are used to compute statistics on the recent past of a data streams [12]. A literature review of [12] and [26] reveals that there are a variety of time-decay models used to address the dimension of time in stream processing, for example, exponential or polynomial decay models, sliding windows, and fading factors, which are discussed below:

- Exponential or polynomial decay are examples of such models, which may be used to attribute less weight/significance to older data instances.
- Sliding windows are another example of a time-decay model, which are either time-based (the data instances in last N time units, for example) or counter-based (the last N items). In fact, sliding windows are one of the most popular forgetting mechanisms used in the literature [12]. Interested readers are referred to example 2.1, which illustrates how the mean of the elements in a stream can be computed over a sliding window.
- Fading factors are another time-decay model [12] and are explained further in section 2.3.1.

Applications of stream processing include intrusion detection systems, financial data systems, and sensor networks, which probably involve some form of stream learning [26].

Challenges with learning from streams include the following [12]:

- There is a continuous data flow with data streams instead of a static sample of identically independently distributed (i.i.d.) data instances from standard static datasets (a database, for example).
- Instead of being static, the decision model evolves over time.
- The data instances are generated from a non-stationary distribution instead of being generated from a stationary distribution.

Example 2.1. Sliding window average

Compute the mean of the last w stream elements, μ_w, as follows:

$$\mu_w = \frac{1}{w} \sum_{k=i-w+1}^{i} x_i$$

, where i is the time, and x_i is the stream element at time i. Note that this example was inspired from [12].

2.3.1 Fading Factor. Fading factors are also another forgetting mechanism. According to [12], fading factors puts less weight on older information as time moves forward. The *fading sum*, *fading average*, and *fading increment* are examples of fading factors. These fading factors are defined below, where x_i is the data instance from a stream at time i, and α, a forgetfulness parameter, is defined as $0 \ll \alpha \leq 1$, where '$\ll$' means "much smaller than" (for example, $\alpha = 0.999$).

Definition 9. *Fading sum*

The fading sum $S_\alpha(i)$ is computed at time i as:

$$S_\alpha(i) = x_i + \alpha \cdot S_\alpha(i-1)$$

, where $S_\alpha(1) = x_1$.

Definition 10. *Fading increment*

The fading increment $N_\alpha(i)$ is computed as:

$$N_\alpha(i) = 1 + \alpha \cdot N_\alpha(i-1)$$

, where $N_\alpha(1) = 1$. It is important to note that the following expressions holds:

$$\lim_{i \to \infty} N_\alpha(i) = \frac{1}{1-\alpha}$$

Definition 11. *Fading average*

By combining the fading increment and fading sum, the fading average $M_\alpha(i)$ is computed as:

$$M_\alpha(i) = \frac{S_\alpha(i)}{N_\alpha(i)}.$$

2.3.2 Uncertain data streams. A literature review of [2], [6], [7], [14] reveals that uncertain data streams consist of objects who's values are typically represented using probability distributions (as opposed to a specific value), where a series of noisy/fuzzy data instances sampled from the object's probability distribution are found in the stream. In other words, each object o_i in the data stream(s) S has an uncertain dataset $D_i \subseteq S$, and the stream elements may take the form $\{x, p_x, t\} \in D_i$, where x is the data instance's value, p_x is the probability of the value x, and t is the timestamp of reading/receiving the element

Domains where data sources can be treated as uncertain data streams typically occur when the data sources have underlying uncertainty (inaccuracies/noise/fuzziness) in the data elements of the stream. Three examples of uncertain data streams follow:

- A stream of sensor data may be represented as an uncertain data stream, since sensors may be noisy and their readings may be inaccurate. In the domain of vehicle diagnostics, an object o_i could be the engine temperature of the vehicle v_i, and the engine temperature has sensor readings $r \in D_i$, which represent the temperature readings of v_i's engine temperature. This is an example of a continuous case of uncertain data streams (temperature readings are real numbers). Since the engine temperature is continuously changing, a probability distribution is an effective way to represent the vehicle's temperature (a single sensor reading would soon become stale, as it would probably not be correct to conclude the temperature reading at time $t+1$ is equal the reading from t). Furthermore, there may be many objects present in this uncertain data stream (note that this stream may be distributed, one stream for each vehicle, for example), each object representing the probability distributions of the engine temperature of various vehicles in a fleet.
- A stream of user review scores of various camera brands have an element of uncertainty. Users in this example rate cameras with a score from 1 to 5 via an e-commerce website. This is an example of a discrete case of uncertain data streams (the user scores are integers). The objects in this example are the ratings of cameras, which are represented using the mean and variance of user scores for each camera, and the series of user scores are the data instances. The uncertainty in this example can be modeled by the uncertainty in the user score space; that is, two cameras may have the same mean rating, but they may in fact have different variances, which introduces uncertainty when comparing camera ratings.
- Uncertainty may at times be introduced into a data stream intentionally. For example, in a privacy domain where the user data should be anonymous/private for each user, a user's data stream may have synthetic noise added to it to provide privacy. This makes it difficult to deduce the true underlying values of the users/objects, and instead, only the probability distributions of the objects may be manipulated and processed.

2.3.3 Sketch. Sketch techniques are used to store high velocity data streams while only using little memory footprint, which enables efficient stream summarizing. Sketches also typically have efficient update complexity [26]. However, sketching techniques come at the cost of a trade-off between efficiency (memory and computational cost) vs. accuracy. The applications of sketch techniques include clustering, anomaly detection, and frequent item mining [6].

2.4 Coresets

A literature review of [1] and [6] reveals that coresets are geometric approximations of a set of points P; that is, a coreset $Q \subseteq P$ is approximated to be geometrically similar to P, which enables inefficient computations to be run on Q while approximating the result as if the computations were done on P. An extent measure is some measure on the extent of a set of points; for example, the diameter of a set of points is an extent measure. It can be costly to compute an exact extent measure on a set of points P; for example, finding the volume (the extent measure) of the smallest volume bounding box over $P \in \mathbb{R}^3$ is $O(n^3)$ with state-of-the-art algorithms.

As a result, approximation algorithms have been a research topic of interest, where a $(1+\epsilon)$-approximation algorithm, where $0 < \epsilon < 1$, estimates a extent measure with complexity $O(nf(\epsilon))$ or $O(n + f(\epsilon))$, for some function of ϵ, f. For such algorithms, it is assumed that there exists a coreset $Q \subseteq P$, where the space complexity of Q is $1/\epsilon^{O(1)}$ and computing the extent measure on Q approximates the measure on P, using some inefficient extent measure computing algorithm. For example, in the shape fitting problem family, for shape γ and an extent measure μ, a shape γ^* can be estimated by fitting γ to Q, such that $\mu(Q, \gamma) \geq (1-\epsilon)\,\mu(P, \gamma)$.

2.5 Clustering

A literature review of [5], [20], [21], [4], and [14], reveals that clustering involves grouping similar objects (or data points) into relevant groups; that is, similar data points should be grouped together while objects that are different should be in different clusters. There are two dimensions of clustering methods, namely:

- hierarchical clustering algorithms, and

- partitioning clustering algorithms.

Partitioning clustering algorithms partition a dataset into a flat (or one-level) partition of k clusters, where similar objects are partitioned together. **Hierarchical clustering** splits the dataset, in an iterative fashion, into different levels/subsets and further splits these levels into sublevels. The splitting continues until each subset has a single object. A tree can be used to represent this clustering, and each node in the tree represents a cluster. OPTICS, which is discussed in section 3.2.4, is an example of a hierarchical clustering algorithm.

Another clustering dimension is the type of clustering; that is, there are two main types of clustering:

- center-based (or optimization-based/distance-based) clustering, and
- density-based clustering

, which are used to cluster certain data (as opposed to uncertain data). **Center-based clustering** algorithms group data points to the closest center (the k-means algorithm, which is discussed in section 3.2.1, is an example of this type of clustering algorithm), while **density-based clustering** algorithms group objects by region/density instead of using a center (DBSCAN, which is discussed in section 3.2.3, is an example of this type of clustering algorithm), where dense regions (clusters) are separated by non-dense regions. Furthermore, for any clustering algorithm, the resulting cluster correctness produced from the algorithm depends on the problem at hand [24].

The remainder of this section presents two special types of clustering problems, namely, fuzzy clustering and uncertain data stream clustering.

2.5.1 Fuzzy Clustering. Fuzzy clustering is based on the idea that typically there is no perfect separation of objects in datasets. Fuzzy clustering assigns a degree of membership (a value between 0 and 1, where 0 indicates no membership and 1 indicates perfect membership) to objects for each cluster. In other words, an object may belong to multiple clusters at once [20].

2.5.2 Uncertain Data Stream clustering. According to [24], in traditional data clustering, the objects are points in space and a distance metric is used to compare the objects. In uncertain data stream clustering, objects are no longer represented as data points in space, they are instead represented using a probability density functions (or probability mass function). Measuring the distance value (using a standard metric, for example, Euclidean distance) between two uncertain objects loses its meaning, since the single-valued distance measure fails to represent the uncertainty between both object [21]. Imagine the case where we are trying to cluster two sets of points that have the same mean. Clearly the center-based and density-based clustering approaches will fail, since both sets will have large overlap. [14] propose using KL-divergence to address this, which enables the comparison between probability distributions. They also mention that the world approach clustering technique can be used to cluster uncertain data.

An example of uncertain data stream clustering follows, which is an extension of the vehicle diagnostics domain example from section 2.3.2: in this example, clustering uncertain data streams would involve grouping together the vehicles that have similar engine temperature probability distributions (as opposed to grouping the temperature readings)

3 DATA MINING ALGORITHMS

This section explores key data mining algorithms: section 3.1 presents stream processing algorithms, section 3.2 presents popular clustering algorithms, and section 3.3 presents uncertain data stream clustering algorithms.

3.1 Data stream processing algorithms

Important data stream processing algorithms are discussed in this section.

3.1.1 Exponential Histograms. A literature review of [26] and [9] reveals exponential histograms are histograms that summarize aggregates over sliding windows. A sliding window of the most recent N bits from a stream are partitioned into exponentially sized (2^i) buckets. Whenever a bit '1' is received, the buckets are re-arranged, possibly merging buckets, and when a '0' is received, the buckets have no change. These buckets are used to query an

estimation of the number of '1's in the sliding window. Querying a section of the window (for example, querying the number of true bits in some range), involves aggregating the buckets that overlap with the queried range.

Error Estimation: the last bucket may be partially outside the sliding window, which can introduce inaccuracy when querying the exponential histogram. When a bucket no longer intersects with the sliding window, it is forgotten. Note that there may be some error due to the last bucket only partially overlapping with the sliding window, although this bucket's size bounds the error. At least half of the last bucket has to be overlapping with a query's range to be included in the query. Furthermore, the invariant below is maintained for all the buckets j:

$$\frac{C_j}{2\left(1+\sum_{i=1}^{j-1} C_i\right)} \leq \epsilon \tag{8}$$

, where ϵ is the maximum acceptable relative error, and C_j denotes the size (or number of true bits that have arrived in the bucket range).

Complexity analysis: let N denote the length of the sliding window, $u(N,S)$ denote the upper bound on number of items that can arrive in one time unit for stream S on a sliding window of length N, and $g(N,S) = \max(u(N,S), N)$. The time and memory complexity analysis are found below:

- Memory: $O\left(\log^2(g(N,S))/\epsilon\right)$ = # buckets × bucket size = $O(\log(u(N,S))/\epsilon) \times O(\log(N) + \text{loglog}(u(N,S)))$
- Amortized (usual) update: O(1)
- Worst case update: $O(\log(u(N,S)))$
- Query over sliding window
  - entirety: O(1)
  - subset with bucket linear search: $O(\log(u(N,S)/\epsilon))$
  - subset with bucket binary search: $O(\log(\log(u(N,S)/\epsilon)))$

3.1.2 Count-min sketch. A literature review of [8], [6], [3], and [26] reveals that count-min sketch is a space-aware frequency estimation technique based on hashing. There is a trade-off between the accuracy of the estimated frequency of elements found in a set and the space required to store the count-min sketch data structures.

For some given input data stream S, the frequency f_v of a data instance $v \in S$ is estimated by count-min sketch as $\widehat{f_v}$ such that $\Pr(|\widehat{f_v} - f_v| > \epsilon f_v) < \delta$, where ϵ and $\delta > 0$ are tuning parameters. The frequency estimation is performed by using a $t \times k$ 2D-array, CMS, of counters, which are incremented using t 2-universal hash functions (one hash function h_i for each row), where $CMS[i,j]$ denotes the counter/cell at row i and column j, $k = \frac{2}{\epsilon}$, and $t = \left\lceil \log(\frac{1}{\delta}) \right\rceil$. Each time an object o with value v is read from S, the counters $CMS[i, h_i(o)]$, $\forall\, i \in \{1,2,...,t\}$, are incremented by v (incrementing by 1 keeps track of an object's frequency). Computing $\widehat{f_o}$ is done by finding the minimum counter in all rows with respect to the hashes of o; that is, $\widehat{f_o} = \min_{i \in \{1,2,...,t\}} CMS[i, h_i(o)]$ [26].

3.1.3 Sketch-metric.* The sketch*-metric, proposed by [3], is a metric for estimating the distance between two streams. It is based on count-min sketch and is defined by equation 9:

$$\hat{\phi}_k(p \parallel q) = \max_{p \in P_k(\Omega)} \phi\left(\widehat{p_\rho} \| \widehat{q_\rho}\right) \text{ with } \forall a \in \rho,\ \widehat{p_\rho}(a) = \sum_{i \in a} p(i) \tag{9}$$

, where k is a given precision parameter, ϕ is a generalized metric that satisfies the metric properties defined in equation 2 on the set of all Ω-point distributions.

To compute the sketch *-metric between two streams σ_1 and σ_2, the count-min sketch algorithm (see section 3.1.2) is applied to both streams. Let CMS_1 and CMS_2 denote the count-min array structures used to represent σ_1 and σ_2, respectively, $CMS_i[j]$ denote the jth row of CMS_i, and let $CMS_i[j][k] = CMS_i[j,k]$ for notation convenience. The sketch *-metric computation algorithm proceeds by computing the metric ϕ between all the t rows of CMS_1 and CMS_2 and finding the maximum $\phi(CMS_1[i] \parallel CMS_2[i])$, $\forall\, i \in \{1,2,...,t\}$.

Algorithm 1: Count-min sketch frequency estimation [22]

Input: a stream S of n numbers, r hash functions h_1, h_2, ..., h_r, where $h_i : \mathbb{N} \rightarrow \{1,2,...,b\}$, where b is the number of buckets.
Output: estimated frequency of an integer k found in S;
`#initialize count-min structure`
for $i \in \{1, 2, ..., r\}$ **do**
  for $j \in \{1, 2, ..., b\}$ **do**
    $CMS[i, j] := 0$;
`#populate count-min structure`
for $i \in \{1, 2, ..., n\}$ **do**
  for $j \in \{1, 2, ..., r\}$ **do**
    $CMS[j, h_j\,(S[i])] := CMS[j, h_j\,(S[i])] + 1$;
On frequency estimation query of integer k **return** $\min_{i \in \{1,2,...,r\}} CMS[i, h_i\,(k)]$;

To test their proposed sketch *-metric, the authors compare their metric to the Bhattacharyya distance, and to the Kullback-Leibler and Jensen-Shannon divergences, using synthetic datasets and datasets obtained from the web.

Complexity analysis: this algorithm uses $O\,(t\,(\log n + k \log m))$ memory, where k and t are given parameters, n is the size of the stream objects (for example, the dimension of geometric points), and m is the unknown number of data instances in the streams.

Proof: since there are m objects in each stream and count-min sketch counts these objects, $O(kt \log m)$ bits can be used to represent the $t \times k$ counters in each CMS_1 and CMS_2, and the hash functions can be stored using $O(t \log n)$ bits.

Algorithm 2: Sketch *-metric computation [3]

Input: two input streams σ_1 and σ_2, the distance metric ϕ, space vs. accuracy parameters k and t;
Output: the distance $\bar{\phi}$ between σ_1 and σ_2;
Choose t hash functions h: $[n] \rightarrow [k]$, each from a 2-universal hash function family;
`#initialize count-min structures`
for $x \in \{1, 2\}$ **do**
  for $i \in \{1, 2, ..., t\}$ **do**
    for $j \in \{1, 2, ..., k\}$ **do**
      $CMS_x[i, j] := 0$;
`#fill count-min structures`
for $x \in \{1, 2\}$ **do**
  for $a_j \in \sigma_x$ **do**
    $v = a_j$;
    for i = 1 ***to*** t **do**
      $CMS_x\,[i, h_i\,(v)] := CMS_x\,[i, h_i\,(v)] + 1$;
`#compute sketch *-metric between` σ_1 `and` σ_2 `streams`
On query $\bar{\phi}_k\,(\sigma_1 \parallel \sigma_2)$ **return** $\bar{\phi} = \max_{i \in \{1,2,...,t\}} \phi\,(CMS_1[i], CMS_2[i])$;

3.1.4 ECM-sketch - Exponential Count-Min Sketch. ECM-sketch, proposed by [26], is a method for summarizing distributed data streams by using sliding windows. It also supports accuracy guarantees [6]. It can find frequencies,

find heavy hitters (frequent elements), and calculate quantiles in the sliding windows. It is based on count-min sketches (discussed in section 3.1.2) and exponential histograms (discussed in section 3.1.1).

A ECM-sketch differs from count-min sketch as the counters in the 2D array, *CMS*, are replaced with sliding windows (implemented using exponential histograms) of size N, where N is either the number of time units (time-based sliding windows) or the number of items (counter-based). This means that when querying a range in the length of a sliding window, the estimated counter $\hat{x}$ will be in the range of $(1 \pm \epsilon)\, x$ of the actual value x of the counter. In other words, when a stream element o with value x arrives, instead of incrementing all the counters $B = CMS[j, h_j(o)], \forall\ j \in \{1, 2, ..., t\}$ by x (where h_j is the jth hash function and t is the number of hash functions, or rows, in the count-min structure), ECM-sketch inserts x into all the sliding windows in B.

ECM-sketch supports a few types of **queries**, namely, point queries, inner product queries, and self-join queries (also known as second frequency moment F_2). It is worth noting that by setting the count-min sketch parameters $t = \lceil \ln \frac{1}{\delta} \rceil$ and $k = \lceil \frac{e}{\epsilon} \rceil$, the Pr(the result of point queries will have an error less than $\epsilon \parallel a \parallel_1$) $\geq 1 - \delta$, where $\parallel a \parallel_1$ in this case is the number of processed items in stream a, and ϵ and δ are ECM-sketch parameters. These types of queries are discussed below:

- **Point query:** suppose we are working with a stream a. This type of query takes the form (o, r), where o is the identifier of the object/item and r is the range, which takes the form of the number of time units or number of items. Such queries attempt to answer how many times the value of o has occurred in the range r in the sliding windows, which is denoted as $f_a(o, r)$, and let $\hat{f}_a(o, r)$ denote the estimated result. To compute such queries, all the sliding windows with respect to the hash (for all t hash algorithms) of o are used to compute the estimated frequency of o for h_j, denoted as $E_a\left(j, h_j(o), r\right)$, and $\hat{f}_a(o, r)$ is computed as $\hat{f}_a(o, r) = \min\limits_{j \in \{1,2,...,t\}} E_a\left(j, h_j(o), r\right)$. The **space complexity** will now be discussed:
  - Let ϵ_{sw} denote the parameter ϵ for the exponential histograms (sliding windows), ϵ_{cm} denote the parameter ϵ of count-min sketch, and δ_{cm} denote the parameter δ of count-min sketch. The authors show that for all ϵ_{cm} and ϵ_{sw} satisfying $\epsilon = \epsilon_{sw} + \epsilon_{cm} + \epsilon_{sw}\epsilon_{cm}$, the error is at most $\epsilon \parallel a \parallel_1$ and the space complexity is $O(\frac{\ln^2 Z \ln(1/\delta_{cm})}{\epsilon})$, where $\Pr(|\hat{f}(o, r) - f(o, r)| \geq \epsilon \parallel a \parallel_1) \leq \delta = \delta_{cm}$ and Z is the maximum count of elements in the sliding windows.
- **Inner product and self-join query:** inner product queries are denoted as $a_r \odot b_r = \sum\limits_{o \in D} f_a(o, r) \times f_b(o, r)$, where a_r and b_r are sub-streams of streams a and b, respectively, in the range r, and D is the domain of input elements. Self-joins are special inner products of the form $a_r \odot a_r$. The authors show that inner joins in a given range r have the following properties:

$$Pr\left(|\widehat{a_r \odot b_r} - a_r \odot b_r| \geq \left(\epsilon_{sw}^2 + 2\epsilon_{sw} + \epsilon_{cm}\left(1 + \epsilon_{sw}\right)^2\right) \parallel a_r \parallel_1 \parallel b_r \parallel_1\right) \leq \delta = \delta_{cm}$$

, where $\widehat{a_r \odot b_r}$ is the estimation of $a_r \odot b_r$. Note that $\widehat{a_r \odot b_r} = \min\limits_{j \in \{1,2,...,t\}} \sum\limits_{i=1}^{k} E_a(i, j, r) \times E_b(i, j, r)$.

The authors analyzed ECM-sketch when replacing exponential histograms with Deterministic Wave and Randomized Wave, which are also sliding window algorithms.

Aggregation of Exponential Histograms: the authors also propose an aggregation algorithm to aggregate many exponential histograms EH_i into a single aggregated histogram $EH_\oplus$; that is, $EH_\oplus = EH_1 \oplus EH_2 \oplus ... \oplus EH_n$, for some aggregation operator '$\oplus$'. This proves that by using their aggregation algorithm and by initializing $EH_\oplus$ with error parameter ϵ', they can answer queries (over the n data streams) with a maximum relative error of $(\epsilon + \epsilon' + \epsilon\epsilon')$, where ϵ is the parameter shared by all EH_i.

Complexity analysis for ECM-sketch using exponential histograms: let N denote the length of the sliding window, $u(N, S)$ denote the upper bound on number of items that can arrive in one time unit for stream S on a sliding window of length N, and $g(N, S) = \max(u(N, S), N)$.

- Memory: $O\left(\frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right) \ln^2\left(g(N, S)\right)\right)$

- Amortized (usual) update: $\mathrm{O}\left(\ln\left(\frac{1}{\delta}\right)\right)$
- Worst case update: $\mathrm{O}\left(\ln\left(\frac{1}{\delta}\right)\ln\left(u\left(N,S\right)\right)\right)$
- Query: $\mathrm{O}\left(\ln\left(\frac{1}{\delta}\right)\ln\left(u\left(N,S\right)\right)/\sqrt{\epsilon}\right)$

3.1.5 EU-sketch - Extended Uncertain Sketch. [6] propose EU-sketch in their approach, which is similar to the ECM-sketch algorithm discussed in section 3.1.4. EU-sketch is based on count-min sketch (discussed in section 3.1.2), and instead of storing an object frequency counter in CMS, the probability of the data sample and time of observation are stored in time-based sliding windows instead; that is, the cells of CMS are sliding window buckets. Buckets are organized into two parts, namely, part A and part B. The probabilities of data instances, before some time t', are stored in part A of a bucket, and data instances arriving after t' are stored in part B of the bucket. All probabilities in A are eventually removed (fall out of the sliding window). To avoid impacting the quality of the clustering results, t' must be chosen carefully based on the size of part A (denoted as $|A|$) and part B (denoted as $|B|$) of the bucket. As data instances arrive, $t' = (t_1 + 0.5\,(t_n - t_1))$, and when a bucket is full, probabilities in part A are removed if $|A| > |B|$. Otherwise, in the case where a bucket is full and $|A| \leq |B|$, let $t' = (t' + \log(t_n - t_1))$ and the probabilities in part A are removed.

3.2 Clustering Algorithms

Important data clustering algorithms are discussed in this section.

3.2.1 K-Means. A literature review of [17] and [15] reveals that k-means is an efficient and popular unsupervised clustering algorithm. It clusters a dataset into k clusters, where k is a user-defined parameter representing the number of desired clusters. First, k centroids are chosen randomly from among points (or they can be randomly chosen) in the dataset, where each centroid represents a cluster center. Each point is then compared to the k centroids and are assigned to the cluster of the nearest centroid. The centroids are recomputed as the mean Euclidean distance of all points to the cluster's centroid. The process is restarted until convergence occurs; that is, when the centroids do not change after an iteration. The performance of k-means is largely dependent on the number of iterations required until convergence.

Complexity: $\mathrm{O}\left(ikn^2\right)$ [25], where i is the number of iterations, k is the desired number of clusters, and n is the number of points. Note that $i \simeq n$, so the complexity is essentially $\mathrm{O}\left(n^2\right)$.

Correctness: k-means may converge to a local minima and it can also create empty clusters due to its greedy nature [27]. The sum of squares error (see equation 3) is used to evaluate the quality of clusters [17]. High quality clusters should minimize:

$$SSE = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left(x_j - \mu_i\right)^2$$

, where k is the number of clusters, x_j is a point in cluster C_i, and μ_i is the mean distance of all points to centroid i.

3.2.2 K-Medoids. According to [14], k-medoids is a clustering problem that is similar to the problem the k-means algorithm (discussed in section 3.2.1) addresses, but instead of representing a cluster as the mean of all objects in the cluster, the k-medoids problem represents the cluster using an object in the cluster. K-medoids is a center-based partitioning clustering algorithm. An object representative, named a medoid, is chosen to represent a cluster, instead of using a centroid [27]. A medoid of a cluster C is defined as the data instance (or object) with the smallest sum of dissimilarities compared to all other instances in C:

$$\text{medoid}\,(C) = \min_{x_i \in C} \sum_{x_j \in C} d\left(x_i, x_j\right)$$

Algorithm 3: K-means

Input: desired number of clusters k, and dataset of d-dimensional points X of size n.
Output: a set of k clusters (subsets) of X
`#initialize k random centroids`
$C := \{c_1, c_2, ..., c_k\}$, where c_i is a random point from $\mathbb{R}^d$;
`#initialize clusters (sets of object/data points)`
$O := \{o_1, o_2, ..., o_k\}$, where $o_i = \emptyset$;
`#keep looping until centroids remain the same after 2 iterations`
while *centroids have not converged* **do**
  `#iterate the points`
  for $i \in \{1, 2, ..., n\}$ **do**
    `#find the index of nearest cluster to point` x_i
    find j such that $\min_{j \in \{1,2,...,k\}} \| x_i - c_j \|_2$
    `#assign the point` x_i `to nearest nearest cluster`
    assign x_i to cluster o_j
  `#recompute centroids`
  for $j \in \{1, 2, ..., k\}$ **do**
    $c_j = \frac{1}{|o_j|} \sum_{\forall x \in o_j} \| x - c_j \|_2$
return O

, where d is the dissimilarity function (a distance function), and x_i is the ith instance in C.

The quality of clusters are evaluated in a similar way to k-means, but instead the absolute error (total distance) is used:

$$TD = \sum_{i=1}^{k} \sum_{x_j \in C_i} d\left(x_j, m_i\right)$$

, where m_i is the representative object/instance of cluster i. Finding the global optimum for this problem is NP-hard [27].

A popular algorithm to solve the k-medoids problem is the **Partitioning Around Medoids** (PAM) algorithm, and the authors present some of its variations. PAM consists of two algorithms, BUILD and SWAP [27].

- BUILD will create the initial clustering. It will find the instance/object that minimizes TD, and once found, this object is assigned as the representative object for the first medoid. This process continues for the new object that minimizes TD to find the next medoid, until k representative objects are found.
- SWAP will optimize the clusters by swapping out medoid objects with non-medoid objects. There are $k\ (n-k)$ candidate swaps.

PAM Complexity: It usually takes $O(n^2 d)$ time complexity, where n is the number of points and d is the cost for computing a distance measure, and $O(n^2)$ memory is required for storing the dissimilarity matrix [27].

3.2.3 DBSCAN. DBSCAN was originally proposed in [10]. According to [20] and [10], DBSCAN is a density-based clustering algorithm that clusters dense regions of d-dimensional points and detects outliers. The neighborhood around each point in a cluster must contain a minimum number of points to be included in the cluster; that is, a point is only clustered if the number of points in its neighborhood exceeds some threshold. Furthermore, the distance function $dist(x_i, y_j)$ used (for example, Euclidean distance or Manhattan distance), will determine the

shape of the neighborhoods: circles using the Euclidean distance and squares using Manhattan distance, for example. Points that do not have a dense-enough neighborhood are considered to be outliers.

Parameters: the parameters of DBSCAN are:

- ϵ: the maximum distance (with respect to the chosen distance function) required for a point q to be considered in the neighborhood of a point p
- $MinPts$: the minimum number of points in the neighborhood of a point p to consider p to be part of a cluster.

Let D denote the dataset of objects/points.

Definition 12. *ϵ-neighborhood*

The ϵ-neighborhood $N_\epsilon(o)$ denotes $\forall o' \in D, o \neq o'$ that have dist $(o, o') \leq \epsilon$, where $dist(\cdot, \cdot)$ is the chosen distance function.

Definition 13. *Core objects*

An object o is a core-object with respect to ϵ and $MinPts$ in D, if $|N_\epsilon(o)| \geq MinPts$.

Definition 14. *Directly Density-reachable*

An object p is directly density-reachable from an object o with respect to ϵ and $MinPts$ in D, if o is a core-object and $p \in N_\epsilon(o)$.

Definition 15. *Density-reachable*

An object p is density-reachable from an object o with respect to ϵ and $MinPts$ in D, if there exists a chain of objects $p_1, p_2, ..., p_n$ such that $p_1 = o$ and $p_n = p$, where $p_i \in D$ and p_i is directly density-reachable to p_{i+1} with respect to ϵ and $MinPts$ in D.

Definition 16. *Density-connected*

An object p is density-connected to an object q with respect to ϵ and $MinPts$ in D, if there exists an object $o \in D$ such that both p and q are density-reachable to o with respect to ϵ and $MinPts$ in D.

Algorithm: DBSCAN begins to assign points to the first cluster c_i. It iterates through the n points $p \in D$, checking whether p is a core-object. If p is not a core-object then it is flagged as noise. Otherwise, p is a core-object and is assigned to cluster c_i, and all points $p' \in N_\epsilon(p)$ are recursively processed in the same fashion. Once the ϵ-neighbors of core-objects have all been processed, all the points in the currently processed dense region will have been either labeled as noise or assigned to the cluster c_i. The next cluster c_{i+1} is then created and DBSCAN applies the same logic for the next unprocessed point $p \in D$ (the next point that has not been labeled as noise nor assigned to a cluster). Once all points in D have been labeled, the algorithm ends.

Complexity: Querying a region (finding the points in a N_ϵ-neighborhood) requires $O(\log n)$ time complexity when using an indexing structure, and requires $O(n^2)$ worst case time complexity without an indexing structure. The regions are queried for each n point, so the average runtime is $O(n \log n)$ [20] when ϵ is chosen such that $O(\log n) \equiv |N_\epsilon(p)|$ on average , for some point p [28].

In terms of space requirements, if a distance matrix is used (to avoid recomputing distances) DBSCAN requires a matrix of size $\frac{n^2-n}{2}$, which is $O(n^2)$; while the memory required without the distance matrix is $O(n)$ [28].

3.2.4 OPTICS. [4] extend the DBSCAN algorithm and name it OPTICS, which is a hierarchical density-based cluster algorithm [21]. It uses the idea of having an infinite number of ϵ_i parameters that are smaller than the generating ϵ to create an ordering of the dataset; that is, OPTICS does not directly cluster the data instances, but instead creates an ordering that could be used by a clustering algorithm to create density-based clusters. OPTICS is based on the idea that denser regions (with respect to a smaller ϵ) are contained in less dense regions (with respect to a larger ϵ). This ordering consists of two pieces of information that are assigned to each point in the dataset D, namely, core-distance and reachability-distance, which are defined in definition 17 and 18, respectively. Using these distances, it is sufficient for any ϵ' smaller than the generating ϵ to be used by another density-based clustering

algorithm to generate a clustering using only, the core-distance, the reachability-distance, and the rank (index of the point) of each data point output by OPTICS.

Definition 17. *core-distance*

Let p be a point from the dataset D, let ϵ be a distance value and $MinPts$ be a natural number (both parameters from DBSCAN discussed in section 3.2.3) and let $MinPts - distance\,(p)$ be defined as the minimum ϵ_i distance such that p is a core-object (see definition 13); that is, taking any $\epsilon_i' < \epsilon_i$ would result in p not having enough points in its ϵ_i'-neighborhood (see definition 12) to be considered a core-object. The core-distance of p is computed as:

$$Core\text{-}distance\,(p) = \begin{cases} \text{UNDEFINED}, & \text{if } |N_\epsilon\,(p)| < MinPts \\ MinPts - distance\,(p), & \text{otherwise} \end{cases}$$

Definition 18. *Reachability distance*

The reachability-distance of data point p with respect to data point o is defined as

$$reachability\text{-}distance_{\epsilon,MinPts}\,(p,o) = \begin{cases} \text{UNDEFINED}, & \text{if } |N_\epsilon\,(o)| < MinPts \\ \max\left(core - distance\,(o), distance\,(o,p)\right), & \text{otherwise} \end{cases}$$

, where ϵ is a distance value, $MinPts$ is a natural number, and $distance\,(o,p)$ is the distance between the data points o and p. In other words, the reachability-distance(o,p) can be thought of as the minimum distance required such that p is directly-density reachable from o, assuming o is a core-object.

Algorithm: OPTICS is similar to DBSCAN from an algorithmic point of view. All the n points $p \in D$ are processed. When p is not a core-object, it is labeled as 'UNDEFINED'. Otherwise, for core-object p, the core-distance of p is computed and recorded. OPTICS proceeds to iterate through the points $p' \in N_\epsilon\,(p)$, and computes the reachability-distance$_{\epsilon,MinPts}\,(p,p')$ for each p'. This same logic is recursively applied to all points in $N_\epsilon\,(p)$, until all points in a region have their core-distance and reachability-distance computed. Finally, OPTICS proceeds to processing the next unprocessed point. The points are finally recorded/stored along with their core-distance and reachability-distances.

The authors illustrate how the dataset ordering created by OPTICS could be used to cluster the dataset using a DBSCAN-like algorithm they call *ExtractDBSCAN-clustering*, which proceeds as follows: given $\epsilon' < \epsilon$ and $MinPts$, iterate through all the objects o ordered by OPTICS and examines the core-distance and reachability-distance to decide how to cluster each object o. In other words, a density-based clustering can be achieved for any ϵ' less than the generating ϵ that OPTICS used to create the ordering, without having to recalculate the neighborhood region queries.

Complexity: The complexity is almost identical to DBSCAN, since OPTICS has an equivalent structure, where the bottleneck in the run-time is the time taken to execute the $\epsilon - neighborhood$ query (see complexity part of DBSCAN section 3.2.3).

3.3 Uncertain Data Stream Clustering Algorithms

Uncertain data stream clustering algorithms are discussed in this section.

3.3.1 FDBSCAN. [20] propose FDBSCAN, a fuzzy clustering method based on DBSCAN, which is discussed in section 3.2.3, that clusters uncertain data using fuzzy distance measures. In FDBSCAN, instead of determining whether an object is a core-object or not, they assign a probability that an object o is a core-object. In the standard DBSCAN, one could represent that o is core-object with a '1' and '0' when o is not a core-object. Therefore, assigning a probability value of being a core-object follows this same idea, with the exception that o can now partially be a core-object, represented with a p-value between '0' and '1'. In standard DBSCAN, a core-object is 100% likely to be

a core-object, for example. Similarly, the authors also assign a probability value that an object q is density-reachable to an object p.

Algorithm: Similarly to DBSCAN, FDBSCAN forms a cluster by adding density-reachable points to the cluster, but instead the points are added to the cluster if their probability of being density reachable exceeds $\frac{1}{2}$. FDBSCAN proceeds to computing the distance between the discrete probability density functions of the objects. Since objects are represented by many instances from a data stream, it becomes impractical to compute all of these distances. As such, FDBSCAN uses minimum bounding rectangles (MBR) over the data instances, and instead computes the distances between MBRs.

Complexity: Without any index structure over the regions when querying ϵ-neighborhoods, FDBSCAN requires $O(n)$ range scans, and requires between $O(n^2)$ and $O(s^2n^2)$ distances computations for d-dimensional data instances, where n is the number of objects and s is the sampling rate (number of data instances) for the objects. When the objects are not too fuzzy, FDBSCAN requires $O(n^2)$ distance calculations. When using an indexing structure, the number of distance calculations can be reduced to $O(n\log n)$, which is the same complexity as DBSCAN.

3.3.2 FOPTICS. FOPTICS is proposed by [21], and it uses fuzzy distance measures to measure similarity between objects (as opposed to using standard distance measures, for example, Euclidean distance), and it is an improvement of the OPTICS algorithm. The objects in FOPTICS can be treated as discrete probability density functions (pdf) or continuous pdf. In the case where the objects are treated as continuous pdf, an object x is represented as a set of s samples $\{x_1, x_2, ..., x_s\}$.

Complexity: $O(n)$ range scans are required, and $O(sn^2)$ distance computations on d-dimensional points are required, where s is the sample rate used and n is the number of data points in the dataset.

3.3.3 Uncertain K-Medoids. [14] present uncertain k-medoids for clustering uncertain data using the KL-divergence, which has complexity $O((k + r)n^2E)$, where r is the number of swapping phase iterations, E is the cost to compare two objects using KL-divergence, n is the number of objects, and k is the number of medoids. The authors also present a randomized uncertain k-medoids to reduce this complexity using approximations, which reduces the complexity to $O(rnE)$ complexity.

3.3.4 Uncertain DBSCAN. [14] propose uncertain DBSCAN, where the KL-divergence is used to decide whether objects are in a dense region. The complexity uncertain DBSCAN is $O(n^2E)$, where n is the number of uncertain objects and E is the cost to compute the KL-divergence between two objects.

3.3.5 UK-Means. According to [24], Uncertain k-means (UK-means) is similar to k-means, but instead of using the Euclidean distance to measure the distance between an object and the cluster's center, the expected distance is used [6]; that is, an object o_i that has the shortest expected distance to c_i's representative point p_{c_i} will be assigned to a cluster c_i. To compute the expected distance between o_i and p_{c_i}, $\mathrm{ED}(o_i, p_{c_i})$, the integral $\int f_i(x)\, d\left(x, p_{c_i}\right) dx$ will need to be calculated. In other words, we want to:

$$\text{minimize} \sum_{i=1}^{n} \left(\int f_i(x)\, d\left(x, p_{c_i}\right) dx \right)$$

, where n is the number of objects of m-dimensional space, x is an uncertain data sample of object o_i, f_i is the pdf of the data sample x under the uncertainty region of o_i, and $d(x, y)$ is a distance metric measuring the distance between points x and y.

Complexity: nkt expected distance calculations, where k is the number of clusters, and t is the number of iterations before convergence. Note that a expected distance calculation ($\mathrm{ED}(o_i, p_{c_i})$) requires more computations than basic distance computations ($d(x, p_{c_i})$); that is, the integral is approximated by summing the distance from p_{c_i} to all sample points, where the distance is weighted by the point's probability density.

The authors propose four possible improvements to UK-means:

- **Improvement** U_{pre}: the authors attempt to improve the efficiency of UK-means by using **min-max distance** pruning to reduce unnecessary expected distance computations. A MBR is placed around the region of some o_i, such that the probability that a data instance of o_i is outside the MBR is approximately 0. An upper bound and lower bound are placed on the expected distance from some o_i to a cluster representative p_j by computing the minimum distance MIN_{ij} and maximum distance MAX_{ij} from p_j to the MBR of o_i. Using this notion, some expected distance computations can be ignored. A min-max distance $\widehat{d_i}$ is defined as the minimum MAX_{ij}. Using $\widehat{d_i}$, it can be shown that any p_j with $MIN_{ij} > \widehat{d_i}$ cannot be the cluster representative of o_i with the smallest distance, which means such expected distance computations can be ignored. This pruning technique suffers from poor expected distance estimations when the MBR of an o_i is large. The authors address this estimation inaccuracy by creating tighter bounds, by using pre-computations with the help of the triangle inequality (see expression *d)* in definition 2). An upper bound can be placed on $\mathrm{ED}(o_i, p_j)$ as follows:

$$\mathrm{ED}(o_i, p_j) \leq \mathrm{ED}(o_i, y) + d(y, p_j)$$

, where y is an anchor point, which is some chosen point to use in the triangle inequality. As such, if $\mathrm{ED}(o_i, y)$ is pre-computed, computing an upper bound on $\mathrm{ED}(o_i, p_j)$ only requires a distance computation $\mathrm{d}(y, p_j)$ instead of a inefficient expected distance computation. The authors call this the U_{pre} method, which they apply to the min-max distance method by comparing MAX_{ij} to U_{pre}, and choosing the smallest. Note that the upper bound on $\mathrm{ED}(o_i, p_j)$ will be the same as the min-max distance method or better.
- **Improvement** L_{pre}: similarly, they apply the same logic for lower bounds on $\mathrm{ED}(o_i, p_j)$ using the inequality below:

$$\mathrm{ED}(o_i, p_j) \geq |d(y, p_j) - \mathrm{ED}(o_i, y)|. \tag{10}$$

, where $\mathrm{ED}(o_i, y)$ can be pre-computed. They call this lower bound L_{pre}. Incorporating L_{pre} into the min-max distance method can be done by comparing L_{pre} to MIN_{ij} and choosing the largest among the two. The lower bounds are equal to or better than the lower bounds produced by the min-max distance method. The y that minimizes $\mathrm{ED}(o_i, y)$ must be chosen wisely, since this improvement only works when more than nr expected distances are pruned, where n is the number of objects and r is the number of anchor points used, since there are nr pre-computed expected distances performed in min-max distance optimization. Therefore, y must be chosen wisely to minimize $\mathrm{ED}(o_i, y)$ to increase the number of pruned expected distance computations. Such a y is found inside the MBR of o_i, which is proven by the authors. It is also the case that choosing such $y \in$ MBR is ideal for L_{pre}, since it minimizes $\mathrm{ED}(o_i, y)$ and maximizes $d(y, p_j)$, which maximizes $\mathrm{ED}(o_i, p_j)$ (see equation 10). It is not trivial to pick an appropriate $y \in$ MBR of o_i. A scheme could be used where y is picked as the center of MBR, picked from the points on edges of MBR, or picked from points on edges and corners of MBR.
- **Improvement** U_{cs}: the authors also propose a method for avoiding pre-computations and automatically choosing y based on previous iterations of UK-means, which uses the idea that the cluster representative p_j will be similar to the representative in the next iteration p_j'. By examining the inequality below:

$$\mathrm{ED}(o_i, p'_j) \leq \mathrm{ED}(o_i, p_j) + d(p_j, p'_j) \tag{11}$$

, it can be seen that choosing $y = p_j$ should be a reasonable choice for the next iteration, since we want to minimize $d(p_j, p'_j)$, and in k-means, cluster centroids tend to be similar in-between consecutive iterations. The authors call this method U_{cs}, for upper bound estimation based on cluster shift.
- **Improvement** L_{cs}: similarly, for the lower bound, the inequality is shown below:

$$\mathrm{ED}(o_i, p'_j) \geq |\mathrm{ED}(o_i, p_j) - d(p_j, p'_j)| \tag{12}$$

, which is denoted as the L_{cs} method. It is clear that minimizing $d(p_j, p'_j)$ will maximize the lower bound.

To summarize, the U_{pre}, U_{cs}, L_{pre}, and L_{cs} improvements can be used interchangeably for pruning expected distance calculations in UK-means using min-max distance method.

In their experiments, the authors use k, the number of clusters, as the baseline, since the brute-force approach to performing UK-means will compare expected distances of all objects to k clusters. The author's goal was to show that their improvements executed fewer expected distance calculations compared to the baseline. In their results, they were able to prune 97% of expected distance calculations.

3.3.6 UESStream Clustering Algorithm. [6] propose UESStream, a clustering algorithm for uncertain data streams. They propose EU-sketch (discussed in section 3.1.5) to summarize uncertain data streams. By using the sketch *-metric (discussed in section 3.1.3) and the KL-divergence (discussed in section 2.2.2) as the stream comparison measure, the authors define a measure that may be treated as a distance measure when comparing two uncertain data streams. The streams are represented using EU-sketch structures. Note that the authors did not discuss the complexity of their proposed UESStream clustering algorithm.

Algorithm: The UESStream algorithm proceeds as follows:

(1) Represent every object using an EU-sketch structure by storing the probability of data instances in the sliding windows.
(2) Partition the objects into coresets (discussed in section 2.4), and choose a representative object for each coreset. To address outliers, local distance-based outlier scores are used to prevent outlier objects from representing a coreset, by replacing the outlier representatives with a more appropriate representative object.
(3) Choose initial cluster centers using max-min distances.
(4) As uncertain data instances arrive, assign the instances to the nearest cluster, and re-calculate the cluster centers.

4 EXERCISES AND SOLUTIONS

This section contains a variety of questions and answers related to the concepts and algorithms found in the present work. This section is separated into two parts: section 4.1 has exercises for the reader, and these exercises are organized by algorithm; and section 4.2 has the answers to these exercises.

4.1 Exercises

4.1.1 K-means.

Exercise 1. What is the time complexity of k-means?

Exercise 2. Is the k-means algorithm a distance/center-based or density-based clustering algorithm?

4.1.2 K-medoids.

Exercise 3. What is the time complexity of solving the k-medoids problem using the PAM algorithm?

Exercise 4. What is the space complexity of solving the k-medoids problem using the PAM algorithm?

4.1.3 DBSCAN.

Exercise 5. What is the average time complexity of the DBSCAN algorithm when ϵ is chosen appropriately and an indexing structure is used over the regions when querying ϵ-neighborhoods?

Exercise 6. What is the worst-case time complexity of the DBSCAN algorithm without using an indexing structure over the regions when querying ϵ-neighborhoods?

Exercise 7. What is the space complexity of the DBSCAN algorithm if a distance matrix is used?

Exercise 8. What is the space complexity of the DBSCAN algorithm without using a distance matrix?

Exercise 9. What is the advantage of using a distance matrix in the the DBSCAN algorithm algorithm?

Exercise 10. Is the DBSCAN algorithm a distance/center-based or density-based clustering algorithm?

4.1.4 FDBSCAN.

Exercise 11. What is the time complexity of the FDBSCAN algorithm if the objects are very fuzzy and an indexing structure is used on the regions when querying ϵ-neighborhoods?

Exercise 12. What is the time complexity of the FDBSCAN algorithm if the objects are not very fuzzy?

4.1.5 Uncertain K-medoids.

Exercise 13. What is the time complexity of the Uncertain K-medoids algorithm without using randomization?

Exercise 14. What is the time complexity of the Uncertain K-medoids algorithm when randomization is used?

4.1.6 Uncertain DBSCAN.

Exercise 15. What is the time complexity of the Uncertain DBSCAN algorithm?

4.1.7 UK-means.

Exercise 16. What is the time complexity of the UK-means algorithm in terms of number of expected distance computations?

4.1.8 Exponential Histograms.

Exercise 17. What is the space complexity of exponential histograms?

Exercise 18. What is the amortized (usual) time complexity for updates of exponential histograms?

Exercise 19. What is the worst-case time complexity for updates of exponential histograms?

Exercise 20. What is the time complexity for querying an entire sliding window of an exponential histogram?

Exercise 21. What is the time complexity for querying a subset of a sliding window of an exponential histogram using a linear search?

Exercise 22. What is the time complexity for querying a subset of a sliding window of an exponential histogram using a binary search?

4.1.9 ECM-sketch. Assume exponential histograms are used to implement the sliding windows of ECM-sketch.

Exercise 23. What is the space complexity of the ECM-sketch structure?

Exercise 24. What is the amortized (usual) time complexity of updating the ECM-sketch structure?

Exercise 25. What is the worst-case time complexity for updating the ECM-sketch structure?

Exercise 26. What is the time complexity for querying the ECM-sketch structure?

4.1.10 FOPTICS.

Exercise 27. What is the time complexity for range scans of FOPTICS?

Exercise 28. What is the time complexity for distance computations of FOPTICS?

4.2 Solutions

4.2.1 K-means.

Answer 1. $\mathrm{O}(n^2)$

Answer 2. center-based

4.2.2 K-medoids.

Answer 3. $\mathrm{O}(n^2 d)$

Answer 4. $\mathrm{O}(n^2)$

4.2.3 DBSCAN.

Answer 5. $\mathrm{O}(n \log n)$

Answer 6. $\mathrm{O}(n^2)$

Answer 7. $\frac{n^2-n}{2} \equiv \mathrm{O}(n^2)$

Answer 8. $\mathrm{O}(n)$

Answer 9. At the cost of more memory being used, fewer distance computations are performed.

Answer 10. density-based

4.2.4 FDBSCAN.

Answer 11. $\mathrm{O}(n \log n)$

Answer 12. $\mathrm{O}(n^2)$

4.2.5 Uncertain K-medoids.

Answer 13. $\mathrm{O}\left((k+r)\, n^2 E\right)$

Answer 14. $\mathrm{O}(rnE)$

4.2.6 Uncertain DBSCAN.

Answer 15. $\mathrm{O}\left(n^2 E\right)$

4.2.7 UK-means.

Answer 16. $\mathrm{O}(nkt)$

4.2.8 Exponential Histograms.

Answer 17. $\mathrm{O}\left(\log^2\left(g\left(N,S\right)\right)/\epsilon\right)$

Answer 18. $\mathrm{O}(1)$

Answer 19. $\mathrm{O}(\log\left(u\left(N,S\right)\right))$

Answer 20. $\mathrm{O}(1)$

Answer 21. $\mathrm{O}(\log\left(U\left(N,S\right)\right)/\epsilon)$

Answer 22. $\mathrm{O}(\log\left(\log\left(U\left(N,S\right)\right)/\epsilon\right))$

4.2.9 ECM-sketch.

Answer 23. $\mathrm{O}\left(\frac{1}{\epsilon}\ln\left(\frac{1}{\delta}\right)\ln^2\left(g\left(N,S\right)\right)\right)$

Answer 24. $\mathrm{O}\left(\ln\left(\frac{1}{\delta}\right)\right)$

Answer 25. $\mathrm{O}\left(\ln\left(\frac{1}{\delta}\right)\ln\left(u\left(N,S\right)\right)\right)$

Answer 26. $\mathrm{O}\left(\ln\left(\frac{1}{\delta}\right)\ln\left(u\left(N,S\right)\right)/\sqrt{\epsilon}\right)$

4.2.10 FOPTICS.

Answer 27. $\mathrm{O}(n)$

Answer 28. $\mathrm{O}\left(sn^2\right)$

5 CONCLUSION

The present work discusses some background on data stream processing and clustering. Many applications can make use of mining data streams. These applications include stock market analysis, weather predictions, social media analysis, sensor networks, and intrusion detection systems. However, there are things that must be considered, namely, a) modern stream processing algorithms are struggling to keep up with the increasing volume and speed of distributed data streams, b) a forgetting mechanism (time-decay model) must be used to address the dimension of time, and c) the data sources may be noisy and inaccurate. Uncertain data stream processing algorithms can be used to process these types of fuzzy/noisy data streams, which address the uncertainty of the data in an efficient manner. Furthermore, the present work also explores different types of data mining algorithms, and presents their complexity and other important details. The types of algorithms discussed in the present work are:

a) **Data stream processing algorithms**, including exponential histograms [9], count-min sketch [8], sketch *-metric [3], ECM-sketch [26], and EU-sketch [6].
b) **Clustering algorithms**, including k-means [17][15], k-medoids [14][27], DBSCAN [10], and OPTICS [4].
c) **Uncertain data stream clustering algorithms**, including FDBSCAN [20], FOPTICS [21], UK-medoids [14], uncertain DBSCAN [14], and UESStream [6].

Due to time constraints, a few parts of the present work are lacking in detail. As such, **future work** may involve:

- Adding additional details to:
  - the FOPTICS, UK-medoids, and uncertain DBSCAN algorithm sections (section 3.3.2, 4.1.5, and 4.1.6, respectively). Only a short summary and the complexity of these algorithms are presented.
  - the *ExtractDBSCAN-clustering* algorithm proposed by [4] in section 3.2.4.
  - the section on fuzzy clustering (section 2.5.1), since only a brief summary is presented .
  - the complexity analysis of count-min sketch in section 3.1.2.
- Exploring and presenting more details on data stream clustering.
- Adding pseudo code to all the data mining algorithms discussed in the present work .

REFERENCES

[1] Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. 2005. Geometric approximation via coresets. *Combinatorial and computational geometry* 52 (2005), 1–30.
[2] Charu C Aggarwal and S Yu Philip. 2008. A survey of uncertain data algorithms and applications. *IEEE Transactions on knowledge and data engineering* 21, 5 (2008), 609–623.
[3] Emmanuelle Anceaume and Yann Busnel. 2013. Sketch*-metric: Comparing data streams via sketching. In *2013 IEEE 12th International Symposium on Network Computing and Applications*. IEEE, 25–32.
[4] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: ordering points to identify the clustering structure. *ACM Sigmod record* 28, 2 (1999), 49–60.
[5] Avrim Blum, John Hopcroft, and Ravi Kannan. 2018. Foundations of Data Science (2018). *Draft available at https://www.cs.cornell.edu/jeh/book.pdf* (2018).
[6] Jingyu Chen, Ping Chen, and Xian'gang Shen. 2013. A Sketch-based clustering algorithm for uncertain data streams. *Journal of Networks* 8, 7 (2013), 1536–1542.
[7] Reynold Cheng, Dmitri V Kalashnikov, and Sunil Prabhakar. 2003. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 551–562.
[8] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.
[9] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining stream statistics over sliding windows. *SIAM journal on computing* 31, 6 (2002), 1794–1813.
[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.
[11] Brian Everitt and Anders Skrondal. 2002. *The Cambridge dictionary of statistics*. Vol. 106. Cambridge University Press Cambridge.
[12] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On evaluating stream learning algorithms. *Machine learning* 90, 3 (2013), 317–346.

[13] Sara Iris Garcia. 2018. *L0 Norm, L1 Norm, L2 Norm & L-Infinity Norm.* https://medium.com/@montjoile/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18a4f40c
[14] Bin Jiang, Jian Pei, Yufei Tao, and Xuemin Lin. 2011. Clustering uncertain data based on probability distribution similarity. *IEEE Transactions on Knowledge and Data Engineering* 25, 4 (2011), 751–763.
[15] Xin Jin and Jiawei Han. 2017. *K-Means Clustering.* Springer US, Boston, MA, 695–697. https://doi.org/10.1007/978-1-4899-7687-1_431
[16] John D Kelleher, Brian Mac Namee, and Aoife D'Arcy. 2015. *Fundamentals of Machine Learning for Predictive Analytics.* Cambridge, MA: The MIT Press.
[17] Patrick Killeen. 2018. Parallel K-means using MapReduce - IPKMeans vs. PKMeans. (2018). https://www.mudlakebiodiversity.ca/papers/hadoop-paper-2018.pdf Technical report, University of Ottawa.
[18] Patrick Killeen and Alireza Parvizimosaed. 2018. An AHP-Based Evaluation of Real-Time Stream Processing Technologies in IoT. (2018). https://www.mudlakebiodiversity.ca/papers/ahp-based-evaluation-iot-2018.pdf Technical report, University of Ottawa.
[19] Paris Koutris, Jerry Li, and Noah Siegel. 2013. *CSE525: Randomized Algorithms and Probabilistic Analysis.* https://courses.cs.washington.edu/courses/cse525/13sp/scribe/lec5.pdf Scribe notes, lecture 5.
[20] Hans-Peter Kriegel and Martin Pfeifle. 2005. Density-based clustering of uncertain data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining.* 672–677.
[21] Hans-Peter Kriegel and Martin Pfeifle. 2005. Hierarchical density-based clustering of uncertain data. In *Fifth IEEE International Conference on Data Mining (ICDM'05).* IEEE.
[22] Anil Maheshwari. 2020. *Count-Min sketch.* https://people.scs.carleton.ca/~maheshwa/courses/5112/Talk-Slides/cms.pdf COMP5112 slides Fall 2020.
[23] Shanmugavelayutham Muthukrishnan. 2005. *Data streams: Algorithms and applications.* Now Publishers Inc. 11 – 14 pages.
[24] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y Yip. 2006. Efficient clustering of uncertain data. In *Sixth International Conference on Data Mining (ICDM'06).* IEEE, 436–445.
[25] Malay K Pakhira. 2014. A linear time-complexity k-means algorithm using cluster shifting. In *2014 International Conference on Computational Intelligence and Communication Networks.* IEEE, 1047–1051.
[26] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. 2012. Sketch-based querying of distributed sliding-window data streams. *arXiv preprint arXiv:1207.0139* (2012).
[27] Erich Schubert and Peter J Rousseeuw. 2019. Faster k-medoids clustering: improving the PAM, CLARA, and CLARANS algorithms. In *International Conference on Similarity Search and Applications.* Springer, 171–187.
[28] Wikipedia. 2020. *DBSCAN.* https://en.wikipedia.org/wiki/DBSCAN