USING DISTRIBUTION ANALYSIS FOR PARAMETER SELECTION IN REPSTREAM

ROSS CALLISTER*, DUC-SON PHAM AND MIHAI LAZARESCU

Curtin University, Kent Street, Perth
Western Australia 6102, Australia

(Communicated by Santu Rana)

Abstract. One of the most significant challenges in data clustering is the evolution of the data distributions over time. Many clustering algorithms have been introduced to deal specifically with streaming data, but common amongst them is that they require users to set input parameters. These inform the algorithm about the criteria under which data points may be clustered together. Setting the initial parameters for a clustering algorithm is itself a non-trivial task, but the evolution of the data distribution over time could mean even optimally set parameters could become non-optimal as the stream evolves.

In this paper we extend the RepStream algorithm, a combination graph and density-based clustering algorithm, in a way which allows the primary input parameter, the K value, to be automatically adjusted over time. We introduce a feature called the edge distribution score which we compute for data in memory, as well as introducing an incremental method for adjusting the K parameter over time based on this score. We evaluate our methods against RepStream itself, and other contemporary stream clustering algorithms, and show how our method of automatically adjusting the K value over time leads to higher quality clustering output even when the initial parameters are set poorly.

1. Introduction. A common feature amongst even newer clustering algorithms is that they require user-set parameters to perform their clustering [5, 9, 21]. These user-set parameters affect how the clustering algorithm in question handles the data as it arrives from the stream, for example how the formation of core-micro-clusters are affected by the ε radius parameter and µ density threshold in Denstream [7]. Another example is D-Stream [8] which is a grid based clustering algorithm, using a grid granularity parameter len and threshold parameters Cm and Ct to determine when cells in the grid are sparse, transitional, or dense. Parameters like these can greatly affect the output of the algorithms, and if set poorly can result in low quality output. This leads to a situation in which setting parameters at appropriate values is desirable, but unfortunately a non-trivial one due to the nature of clustering as an exploratory process.

The problem of parametrising algorithms is even more of a challenge in a stream clustering context. Whereas batch data is expected to have a single distribution which a clustering algorithm is attempting to find, a data stream can have the data

2010 Mathematics Subject Classification. Primary: 62H30.
Key words and phrases. Cluster analysis, data clustering, parameter adjustment, graph-based clustering, KNN graphs.
* Corresponding author: Ross Callister.

215
distribution change over time. Clusters can merge, split, grow, shrink, change in density, or move arbitrarily around the data space. With these types of changes in distribution clustering becomes very difficult, even with prior expertise in the data. An example of such a data set is a stream monitoring a sensor network, in which abnormal or noteworthy activity can cause the distributions to shift rapidly. Naturally, this means that selecting initial parameters is challenging, but additionally an algorithm that has data-dependent input parameters can face the problem of the selected values being inappropriate later during a stream. Even if a user could guarantee optimally selected input parameter values initially, issues such as concept drift [19] in a stream mean that they could result in poor quality clustering at later points in the stream as various changes and evolution occurs.

In this paper we propose an extension to the RepStream algorithm [16] which will allow the primary input parameter, the $K$ value, to be automatically varied over time in response to the structure and distribution of the incoming data. The RepStream algorithm works by representing data in a $K$-nearest neighbour directed sparse graph form, and creating outgoing edges from each data point to its $K$ closest neighbouring data points, according to a selected distance metric. The $K$ parameter, therefore, has a big impact on how the data is clustered together, as a higher $K$ value means a more connected graph, while a lower $K$ value results in a graph with fewer edges in it. Setting the $K$ parameter is vital to having high quality clustering output in the RepStream algorithm. Our proposed method allows RepStream to self-adjust to changes in the distribution of the data stream, allowing for recovery if the parameter is set poorly initially, as well as being able to adjust the value to a more appropriate value over time in response to changes in the stream’s distribution.

A data stream is defined as a set of $d$-dimensional data points $x_1, x_2, ..., x_m ...$ arriving at time stamps $t_1, t_2, ..., t_m ...$ where a data point $x_i = [x_{i1}^1, ..., x_{id}^i]$ is a $d$-dimensional vector [14]. The data stream is potentially unlimited in length and data points are sampled in an unpredictable way from a data distribution that can change and shift over time. Because of the nature of data streams as being evolving and unpredictable the input parameters for clustering algorithms, and in our case RepStream, may be of varying usefulness at different times during the stream. Parameter values which initially may provide high quality clustering output may at a later point in the stream be less optimal than different values because of how the data evolves. RepStream in specific requires its $K$ parameter to be set such that the level of connectivity between data points is not too high - resulting in regions of data being grouped together when they should not be, and also not too low - which results in the algorithm fracturing the data points into too many small clusters. Even within the same dataset the optimal $K$ value can change over time. If an inappropriate $K$ value is selected then RepStream cannot recover.

Selecting an appropriate $K$ value is difficult if one has no prior knowledge of the dataset, and even such knowledge, if available, might be of no help when setting an input parameter for a clustering algorithms due to evolution over time in the data stream. Data clustering is an exploratory and unsupervised process [13], relying on internal validation metrics, and so no knowledge can be assumed before applying the algorithm to a given data stream. We consider high quality clustering output to be when the clustering algorithm is able to accurately cluster contiguous groups of roughly uniform density data points which are separated from each other by a region of different density data points, or regions of space containing no data points.
Table 1. Table showing the input parameters used for each algorithm.

| Algorithm name | Density Parameters | Grid Granularity | Reclustering Method | Decay Parameter | Distance Parameter | Other Parameters |
|----------------|--------------------|------------------|--------------------|-----------------|--------------------|------------------|
| CluStream      | \( \delta \)       | e-mean           | k-means            | \( \beta \)     | \( \epsilon \)     | c/d Snapshot parameters |
| DenStream      | \( \mu, \beta \)    | \( \epsilon \)   | \( \lambda \)      |                |                   | \( \delta \) Projected Dimension |
| D-Stream       | \( \epsilon, \mu \) | len              | Homoscedastic      | \( \lambda \)   | \( \delta \)       | \( \lambda \) Projected Dimension |
| ExCC           | \( \epsilon, \mu \) | Grid granularity | \( \lambda \)      |                |                   | \( \lambda \) Graph connectivity |
| HPStream       | \( \alpha, \beta \) |                  |                    | \( \lambda \)   | \( \epsilon \)     | Hierarchical limit   |
| PASCAL         |                     | Evolutionary     |                    | \( \lambda \)   | \( \epsilon \)     | Evolutionary EDA hyper-parameters |
| BEStream       | \( \Delta, \tau \)  |                  |                    | \( \lambda \)   | \( \xi \)           | Hierarchical limit   |
| SNCStream      | \( \psi, \beta \)   |                  |                    | \( \lambda \)   | \( \epsilon \)     | K Graph connectivity |
| PatchWork      | \( \rho \)          | minCells         |                    | \( \alpha \)    | \( \lambda \)       | K Graph connectivity |
| RepStream      | \( \lambda \)       | \( \alpha \)     |                    | \( \lambda \)   | \( \epsilon \)     | K Graph connectivity |

Our proposed method involves using a computed measure that we call the *edge distribution score*, which reveals some information about the properties and distribution of the data points. The edge distribution score is a measure which is intended to estimate the theoretical distribution of the edges connecting to nearest neighbours, and whether increasing or decreasing the \( K \) value might be appropriate. By computing and measuring the average edge distribution score computed across all data points in memory we present a method for tuning and adjusting the \( K \) value in RepStream dynamically over time. Starting from a given initial \( K \) value we show that our method successfully allows the algorithm to adapt to changes in the stream, yielding higher quality clustering results, and without the need for the user to tune the \( K \) parameter to the data stream itself. We show that our proposed method allows the algorithm to recover and produce high quality results even if an initially poor value for \( K \) is specified.

Our contributions are as follows:

- A measure known as ‘edge distribution score’ which we extract from the \( K \)-nearest neighbour sparse graph structure of RepStream.
- A method of \( K \) selection in RepStream based on analysis of the edge distribution score across multiple \( K \) values.
- Evaluation of the \( K \) selection method on synthetic and real-world datasets to evaluate its performance in comparison to base RepStream as well as other stream clustering methods.

The paper is organised as follows. Section 2 examines previous literature related to stream clustering and parameter selection, including parameterless algorithms. Section 4 explains the concept of the edge distribution score, and details our \( K \) selection method. In Section 5 we test the selection method on synthetic datasets and the well known benchmark KDD and Tree Cover Type datasets. Section 6 is a discussion and analysis of the results, and Section 7 is a summary and conclusion of the paper.

2. Previous work.

2.1. Density-based clustering methods. CluStream [3] is an early example of a distance-based micro-clustering framework for use in stream clustering. CluStream uses Cluster Feature Vectors (CFVs) to maintain information about groups of data points. The CFVs are made up of a tuple, which includes the sum of the data values
for each associated data point in each dimension, the sum of squares of the data values for each associated data point in each dimension, as well as the sum and sum squared of the time-stamps of each data point associated with the CFV. This type of micro-cluster has desirable properties, being both additive, so that CFVs can be easily merged, and also being subtractive such that a snapshot of a CFV at a previous time can be subtracted from the current time to yield data about how the CFV has evolved. CluStream maintains snapshots of its CFVs in a pyramidal way to make use of this property - with more recent snapshots being kept than older ones. A parameter $\alpha$ affects the frequency of snapshots, and another parameter $l$ determines how many snapshots are stored. New points incoming from a stream are added to existing CFVs if they are within a boundary defined by a parameter $t$, or become new CFV micro-clusters otherwise. A relevance threshold $\delta$ determines when an existing CFV can be removed and replaced by a new one. The CFV micro-clusters are used in an offline stage to build clusters with a variant of the $k$-means clustering algorithm that treats the CFV micro-clusters as pseudo-points. The micro-cluster structure is a very common concept which is used in many other algorithms.

Another algorithm which builds on CluStream and uses a similar micro-cluster structure is SWClustering [21]. The SWClustering algorithm uses Temporal Cluster Features (TCFs), that contain similar information to the tuple used by CluStream - sum and sum squared of data values for each associated data point in each dimension, number of records, and most recent time-stamp. TCFs are stored in an exponential histogram, such that more recent records are stored with more detail and granularity than older TCFs. The older TCFs become merged together if the current exponential bucket exceeds $\frac{1}{\epsilon} + 1$ records, where $\epsilon$ is a limiting parameter. Merges are cascaded through the exponential levels, and the last TCF is deleted if its time-stamp is no longer one of the most recent. These Exponential Histogram of Cluster Features (EHCF) act like micro-clusters, and new data points are added to their nearest EHCF based on a radius threshold $\beta$, and at most $N$ EHCFs can be in memory. EHCFs are deleted or merged when the number exceeds the threshold. The EHCF structure allows more granularity on more active clusters over time than the pyramidal snapshots used in CluStream. SWClustering, like CluStream, uses the $k$-means algorithm to cluster the EHCFs as pseudo-points, where each EHCF is weighted based on the number of records it contains.

The BEStream algorithm [18] takes the micro-cluster concept and extends it by introducing the idea of elliptic micro-clusters. In BEStream clusters are in the form of hyper-dimensional ellipses arranged along the eigenvectors of the data distribution, which differs from the standard hyper-spherical micro-clusters used in other algorithms. Additionally, BEStream can handle individual data points from a data stream, or batches of data at a time. Data points are added into existing micro-clusters, with their elliptical shapes being adjusted if necessary to fit the data. Data is captured into micro-clusters, which have a radius according to a parameter $\xi$ affecting the elliptical micro-cluster’s size. To merge micro-clusters together a direction threshold $\theta$ and a distance threshold $\Delta$ are used to determine when micro-clusters can safely be combined into a single elliptical micro-cluster. In the macro-clustering phase a density threshold $\tau$ is used, and overlapping clusters with a density at least $1 - \tau$ similar may be considered part of the same cluster.

2.2. **Graph-based clustering methods.** Nearest neighbour graph-based clustering algorithms are computationally intensive due to having to update and maintain
edges connecting to nearest neighbours for each vertex in a graph. SNCStream [4] is an algorithm which uses social network principles to make this connectivity maintenance more efficient. SNCStream uses a $K$-nearest neighbour clustering approach, in which each vertex has edges connecting to the $K$ neighbours which are closest using a given distance measure. Instead of using the strict nearest neighbours, the maintenance phase uses two-hop neighbours - that is, it searches only data points that connect to its current neighbours when searching to update its list of nearest neighbours. This process is not guaranteed to result in an exact $K$-nearest neighbour graph, but it is likely to be similar to the true KNN graph, and requires that at most $K^2$ neighbours be searched instead of all vertices in the graph. The vertices in SNCStream are treated like micro-clusters similar to DenStream after an initial window size $N$ points have been processed. New data points are added to existing micro-clusters, or become new outlier micro-clusters if no existing micro-cluster is nearby. Similar to DenStream, a micro-cluster must have sufficient weight to become a potential micro-cluster, and be added to the graph. The parameter $\beta$ is the same as in DenStream and $\psi$ replaces the $\mu$ parameter for determining when a micro-cluster is an outlier. The $\epsilon$ parameter determines the radius of micro-clusters, and the $\lambda$ parameter controls a fading function. The benefit of SNCStream is the increased efficiency over other graph-based clustering methods, resulting in faster neighbour searches at a small risk of incorrect graph structure.

2.3. Grid-based clustering methods. An example of grid-based clustering is the Exclusive and Complete Clustering (ExCC) algorithm [5]. ExCC is another fixed granularity approach, but unlike D-Stream its only input parameters are grid granularity for each dimension. Cells in the grid are stored in a tree structure, with each leaf storing the number of data points in the cell, time of the first point to arrive, and the time of the most recent point. The average inter-arrival time of points for each cell can be calculated based on the number of points, and the two previously mentioned stored timestamps. ExCC prunes old cells on calls for clustering. Prior to clustering cells are pruned from the tree structure if they have not been updated in more than the average inter-arrival time. This helps reduce the memory usage of the algorithm, and makes sure older less relevant data is not considered in clustering. Cells which are not pruned at a clustering call are merged together into clusters if they are adjacent. Cells are considered to be dense cells when their weight surpasses an internal threshold $\psi$, which is defined as $\psi = \lceil \frac{\mu}{\ln(g+d)} \rceil$ where $\mu$ is the average number of points in each cell, $g$ is the average granularity of the dimensions, and $d$ is the number of dimensions in the data. Cells which are above this computed threshold form the core of clusters in ExCC, with adjacent dense cells being grouped together. To satisfy the goal of complete clustering, cells which are not dense are added onto adjacent clusters, such that all data points belong to a cluster. ExCC also has a mechanism for handling data drift outside of defined maximums and minimums in each dimensions. When a data point from the stream is outside the bounds of a dimension it is considered to be an anomalous point and is added to a hold queue. This hold queue is periodically processed, and if the number of anomalous points exceeds the number of points in the boundary cells, then the grid’s boundaries are expanded to deal with the data drift.

2.4. Other clustering approaches. The DeBaRa algorithm [17] is another parameter-free algorithm. It was proposed alongside a variation of the OPTICS density-based algorithm and clusters based on the relative density of points,
allowing more dense points to merge clusters together, while low density points (points that have lower densities than their surrounding data points) may only join existing clusters. While this approach uses no data-dependent input parameters the algorithm operates by incrementally adjusting an internal distance parameter to produce different granularity of clustering results.

3. Preliminaries.

3.1. RepStream. In this paper we extend the RepStream algorithm [16], which is a combination density and graph-based stream clustering algorithm. The RepStream algorithm has been shown to perform very well against other similar algorithms, outperforming them in terms of purity of clustering results. At the algorithms core are two directed $K$-nearest neighbour ($K$-NN) sparse graphs which inform the clustering of the algorithm, and as such the $K$ parameter must be specified by the user at runtime. We start by describing the operational method of RepStream and then present our proposed changes which allow our extended algorithm to automatically determine the $K$ value as time goes on during the stream.

RepStream takes as its input a stream of $d$-dimensional data points, where each dimension of the data point represents a numerical feature. These data points are time-ordered and arrive one by one over time. The two $K$-NN directed sparse graphs are constructed using the data points as the vertices, and are referred to as the point-level and the representative-level graphs.

The **point-level graph** is a $K$-NN directed sparse graph in which each vertex of the graph corresponds to one of the data points in the input data stream. When a new vertex is processed by RepStream the data point is added to the point-level $K$-NN sparse graph - outgoing edges are created linking to the $K$ nearest other vertices, and other nearby vertices also have their nearest neighbours readjusted if the new vertex is closer than its existing nearest neighbours. The edges between vertices are directed, in that each edge has an explicit start and end point, and are not bi-directional, however it is possible for a hypothetical vertex $v_i$ to have an outgoing edge to another vertex $v_j$, while $v_j$ has its own outgoing edge linking to vertex $v_i$. In this way it is possible for vertices to be reciprocally connected together. An example of a reciprocal connection is shown in Figure 2, in which edges $E_1$ and $E_2$ are edges that connect their respective vertices. Because of the nature of $K$-nearest neighbour graphs it is also possible for a vertex to have zero edges pointing to it, even though the number of outgoing edges is $K$; this tends to happen when a vertex is relatively very far away from other vertices. Edges also have a length which is dependent on the distance metric used, and while RepStream has several options for distance metrics, including the Manhattan distance, Euclidean distance, Euclidean-squared distance, and Mahalanobis distance. For our purposes in this paper we use the Euclidean distance, which is the standard and most intuitive distance measure.

The **representative-level graph** is the second sparse graph maintained in the RepStream algorithm. The representative graph uses a subset of vertices from the point-level graph to make a second directed sparse graph used for clustering. A vertex becomes part of the representative-level sparse graph when it becomes a representative point. Vertices become representative points when they are inserted into the graph and have no outgoing neighbour which connects to an existing representative point. This method of selection allows the representative vertices to be more or less evenly spread through the data space, being added incrementally as
Representative vertices are added to the point-level graph. Figure 1 shows the relation between the two different levels of sparse graphs in RepStream, and how a subset of the points in the point-level graph are used to construct a separate graph, which is later used in the clustering process of RepStream. RepStream must maintain both of these graphs and update them as new data points are added to the stream which are added to the point-level graph and representative-level graphs as applicable.

To limit the amount of space used by RepStream the user may specify a limit to the number of data points kept in memory. This is done by indicating a maximum number of data points to be maintained in the point-level sparse graph. When a new vertex is added to the point-level graph, the oldest vertex which is not a representative will be removed from the sparse graph, and nearby neighbours will have their outgoing edges rearranged to maintain the $K$-nearest neighbour structure. Representatives vertices are not deleted in the same first in, first out way, instead, they are kept in a repository. Representatives are added to the repository as they are created, until the repository is full, once full the least useful representative is removed each time a new one must be added. This is done by computing a representative usefulness value, which is dependent on the age of the representative, and the number of vertices that have linked to that representative.

The $K$ value in this case is incredibly vital to the RepStream algorithm, because it determines the level of connectivity in both the point-level and representative-level sparse graphs. The edges which connect vertices in these graphs are a major component in the clustering of data points in RepStream, described in the next subsection. Setting the $K$ parameter appropriately is important for achieving high quality clustering as it affects the connectivity as well as the number and density of representative points in RepStream.

3.2. Clustering in RepStream. Forming individual clusters in RepStream is done at the representative level sparse graph, but using additional information
Figure 2. The representative vertex $R_1$ and $R_2$ share a reciprocal connection at the representative level, and are also density related. The density radius of the vertices is shown as $DR_1$ and $DR_2$ respectively, which is the average distance to neighbours at the point level, multiplied by the alpha scaling factor.

from the point level. This forms a combination graph and density based clustering method.

Clusters are defined as groups of representative points which are both reciprocally connected, as we define above, at the representative level, and which are also mutually density related.

**Definition 3.1.** A representative vertex $v_i$ is said to be density related to a vertex $v_j$ if the following condition is met:

$$D(v_i, v_j) < RD(v_i) \cdot \alpha$$

where $D(v_i, v_j)$ is the distance between the two vertices, $RD(v_i)$ is the average distance of the vertices in the vertex $v_i$’s $K$ neighbourhood, and $\alpha$ is a configurable parameter for tuning the density relation.

When two representatives have reciprocal outgoing edges which connect to each other, and both vertices are density related then they are considered to be part of the same cluster. This works in a transitive way, such that if vertex $v_1$ is in a cluster with $v_2$, and $v_2$ is also in a cluster with $v_3$, then both $v_1$ and $v_3$ are in the same cluster. This is demonstrated in Figure 2, in which the two representatives $R_1$ and $R_2$ are reciprocally connected via edges $E_1$ and $E_2$, and are also reciprocally connected, by being within each other’s density relation radius, denoted by the radii around the vertices.

As vertices and edges are added and removed from both the point and representative level sparse graphs over time, these clusters can change. Two vertices can lose
their reciprocal connections as the $K$ neighbourhood shifts, or can lose their density relation if the density near a vertex changes. When this happens clusters can split. Similarly, clusters can merge together when vertices gain reciprocal connections or gain mutual density relation.

3.3. RepStream algorithm. To demonstrate how RepStream processes new data points, Algorithm 1 shows the high level steps that a new vertex goes through. A new vertex $v_{new}$ is added to the point-level sparse graph, having edges created between neighbouring vertices using the $createLink(v_i, v_j)$ function. The function $NN(v_i)$ denotes the set of existing neighbours for the vertex $v_i$, whilst $FarEdge(v_i)$ will return the edge connecting to its farthest neighbour in its $K$ neighbourhood, which is determined by the distance between two vertices, denoted by $dist(v_i, v_j)$. Adding a vertex to the sparse graph can cause the $K$ neighbourhoods of existing vertices to shift, which may cause edges of existing vertices to be replaced. If a vertex being linked into the sparse graph has no existing reciprocal connection to a representative vertex, then it itself becomes a representative, which is shown as $makeRepresentative(v_{new})$.

Linking a new representative vertex into the representative level sparse graph follows a similar process, shown in Algorithm 2. The vertex has edges created to nearby representatives in its $K$ neighbourhood, using the representative level function $createLinkR(r_i, r_j)$. The functions $NNr(r_i)$ and $FarEdgeR(r_i)$ return, respectively, the set of $K$ nearest representative vertices, and the farthest existing representative in the $K$ neighbourhood. Once the new representative is connected to its $K$ nearest representative neighbours, it may merge with existing clusters if it shares a reciprocal connection to a neighbour, whilst also being density related. As we mention before, this is a transitive property, so it’s possible for groups of representatives to be merged into a large cluster.

4. Proposed method. To select suitable $K$ values over time we extract and analyse a feature in the $K$-nearest neighbour structure of RepStream which we call ‘edge distribution score’. This computed feature is then fed into an incremental algorithm to determine which $K$ value to use at each time step. The edge distribution score of a graph is a measure which will reflect the rough distribution of nearest neighbour data, and gives us an idea of whether there exist outgoing edges from a vertex which connect to data points belonging to separate ground-truth classes. We wish to minimise the number of such edges, which we refer to as inter-class edges, while maximising the connectivity of data points belonging to the same theoretical ground-truth class - so called intra-class edges.

4.1. Inter versus intra class edges. For our purposes in this paper we will use the term classes to refer to the theoretically perfect groupings of data points as determined by the distributions in the stream. Information on these ground-truth classes is typically not available because cluster analysis deals with unlabelled data, however evaluation and test datasets can be produced which have information on the ground-truth classes. In this paper we refer to classes as the theoretically perfect cluster groupings, which are unknown to the algorithm and to the user. We also refer to clusters, which are the groupings of data points produced by the algorithm on demand during its runtime. These clusters are not necessarily the same as the ground truth classes, but if the information is available then an external validation
Algorithm 1 Algorithm for RepStream Algorithm

1: procedure LINKINTOGRAPHSG($v_{new}$, $neighbours$) \Comment*[t]{new vertex and neighbours}
2: \quad for each $v_j$ in $NN(v_{new})$ do
3: \quad createLink($v_j$, $v_{new}$)
4: \quad if $|NN(v_j)| < k$ or $dist(v_{new}, FarEdge(v_j)) < dist(v_j, FarEdge(v_j))$
5: \quad createLink($v_j$, $v_{new}$)
6: \quad if $|NN(v_j)| > k$ then
7: \quad Remove edge to farthest neighbour of $v_j$
8: \quad update local density of $v_j$
9: \quad end if
10: \quad end if
11: \quad end for
12: \quad for Each each vertex $v_j$ which has a new reciprocal link do
13: \quad if $v_j$ is a representative vertex then
14: \quad Update representative reinforcement of $v_j$
15: \quad Delete and unlink least useful representative if repository is full
16: \quad end if
17: \quad end for
18: \quad if $v_{new}$ not reciprocally connected to a representative then
19: \quad makeRepresentative($v_{new}$)
20: \quad end if
21: \quad if $v_j$ not reciprocally connected to a representative then
22: \quad makeRepresentative($v_{new}$)
23: \quad else
24: \quad assign $v_{new}$ to nearest representative
25: \quad end if
26: \quad for each vertex $v_j$ with removed edges do
27: \quad if $v_j$ not reciprocally connected to a representative then
28: \quad makeRepresentative($v_j$)
29: \quad end if
30: \quad end for
31: \quad end procedure

In cluster analysis we wish to find regions of data points in a more or less uniform density separated by space at a significantly different density of data points. Most often this takes the form of clusters of data points in specific arbitrarily shaped regions of the data space, separated by regions of empty space. This empty space between classes, which may sometimes contain noise data points,

We will define inter-class edges and inter-class edges as follows:

- **Inter-Class Edges** are edges which connect between two vertexes belonging to different classes.
- **Intra-Class Edges** are edges which connect between two vertexes belonging to the same class.
**Algorithm 2** RepStream representative level link-in algorithm

1: procedure LinkIntoGraphRSG($r_{new}$, neighbours) \triangleright new representative vertex and neighbours
2: \hspace{1em} for each $r_j$ in NN$_r(r_{new})$ do
3: \hspace{2em} createLinkR($r_j$, $r_{new}$)
4: \hspace{2em} if $|NN_r(r_j)| < k$ or dist($r_{new}$, FarEdgeR($r_j$)) < dist($r_j$, FarEdgeR($r_j$)) then
5: \hspace{3em} createLinkR($r_j$, $r_{new}$)
6: \hspace{3em} if $|NN_r(r_j)| > k$ then
7: \hspace{4em} Remove edge to farthest neighbour of $r_j$
8: \hspace{3em} end if
9: \hspace{3em} if densityRelated($r_{new}$, $r_j$) then
10: \hspace{4em} Merge clusters of $r_{new}$ and $r_j$ if not already in same cluster
11: \hspace{3em} end if
12: end if
13: end for
14: for each vertex $r_j$ with removed edges do
15: \hspace{1em} Check if cluster of $r_j$ must be split
16: end for
17: end procedure

Figure 3 shows examples of these in a $K$-nearest neighbour graph where there are two ground-truth classes, and each vertex is a member of one of the classes. $E_1$ is an edge which connects two points belonging to different classes, and thus it is an inter-class edge. $E_2$ is an edge which connects two points belonging to the same class, and is an example of an intra-class edge.

As the $K$ value of a $K$-nearest neighbour graph increases, the vertices become more and more connected together. Density and graph-based clustering methods assume that there is a level of separation between classes which is greater than the distance between nearest neighbouring vertexes in the same class. Without this level of separation the boundaries between clusters would be virtually impossible to determine. This level of separation means that at lower $K$ values intra-class edges are more likely to form than inter-class edges. As the $K$ value increases the edges will connect to more and more distant vertexes, and inter-cluster edges become more common. The ideal $K$ value is one which connects the vertices of each class as much as possible, while avoiding connections between classes, so that they are not merged into the same cluster. In other words, we want a $K$ value which has many intra-class edges, and few inter-class edges. Some degree of inter-class connection is acceptable and will not result in merging clusters belonging to different classes, however the more inter-class connections can be avoided, the less likely such an event becomes.

Obviously these definitions of intra-class and inter-class edges require knowledge of the ground truth classes. This is something that is not known when performing clustering, so measuring intra-class and inter-class edges must be done indirectly. We propose a method of approximating the presence of inter-class edges by measuring a feature known as edge distribution score.

4.2. **Edge distribution score.** The edge distribution score is a feature we extract from the $K$-nearest neighbour graph structure. It is designed to give us an idea of the distribution of the edge lengths of a given vertex compared to how we would
Figure 3. Intra and Inter-class edges. The Edge $E_1$ is considered an inter-class edge as it connects two vertices $R_1$ and $R_2$ that belong to different ground-truth classes. Edge $E_2$ connects two vertices belonging to the same class and thus is considered an intra-class edge.

Figure 4. An illustration of relative edge lengths of nearest neighbours in the middle of a cluster versus near the edge of a cluster.

Expect it to be in a normal clustering context. Edge distribution score gives us a measure calculated on each vertex which is determined by the relative edge lengths represented as a one-dimensional distribution. When this distribution is not consistent with our expectations of a stable cluster, the measure gives an indication of whether the number of outgoing edges is too high or too low.

Each vertex in RepStream has a number of outgoing edges which link to other vertices in a $K$-nearest neighbour fashion. To compute our distribution score we use the edge lengths of these outgoing edges, and treat them as a one-dimensional distribution. In these computations the direction and relative position of the edges is not taken into account, only the length of each edge.

In what follows, we denote $v_i$ as the vertex of node $i$ in the $K$-NN graph, and $e^1_i, e^2_i, \ldots, e^K_i$ as the outgoing edges from node $i$. The length of an edge $e^j_i$ is $l^j_i$. We introduce the following concepts
Definition 4.1. The span $s_i$ of vertex $v_i$ is the edge length to its farthest node within its $K$ neighbourhood $\mathcal{N}_K(v_i)$ is

$$s_i = \max_{j \in \mathcal{N}_K(v_i)} l_{ij}.$$

The span $s_i$ of a vertex is a non-negative quantity and it represents the support of the local neighbourhood. Denote $r_i$ as the interquartile range, $m_i$ as the median, and $\mu_i = \frac{1}{2}(l_{\max} - l_{\min})$ as the average of the shortest and longest of the edge length values associated with a vertex $v_i$. We formally define the edge distribution score (EDS) as follows

Definition 4.2. The edge distribution score of a vertex $v_i$ is given by

$$EDS(v_i) = \begin{cases} 2\theta & \text{if } s_i = 0, r_i = 0 \\ -1 & \text{if } s_i > 0, r_i = 0 \\ \frac{2(m_i - \mu_i)}{r_i} & \text{if } r_i > 0 \end{cases}$$

where $\theta$ is a fixed threshold independent of the data stream statistics.

In the above definition, the first branch accounts for the rare case when all vertices within the $K$-neighbourhood of a vertex are the same as the vertex itself. In this case, the distribution of the edge length reduces to a singularity. The second branch accounts for a similar and rare case where a large proportion of the vertices within the $K$-neighbourhood are the same and cause the inter-quartile range $r_i$, which is a measure of the spread of the distribution, to become zero. The third branch is what we expect most common: the distribution is either left skew, right skew, or symmetric.

From the definition, it can be seen that EDS is a measure similar to the skewness of a distribution. The difference is that our definition caters for the special case, and the mean value in the usual skewness has been replaced by the average of the extreme values of the edge length.

Let $G_K(\mathcal{V}, \mathcal{E})$ be the $K$-NN graph consisting of the vertices $\mathcal{V}$ and edges $\mathcal{E}$. We introduce the following definition

Definition 4.3. The average distribution score (ADS) for an $K$-NN graph $G_K(\mathcal{V}, \mathcal{E})$ is defined as

$$ADS(G) = \frac{1}{|\mathcal{V}|} \sum_{v_i \in \mathcal{V}} EDS(v_i)$$

where $|\mathcal{V}|$ denotes the total number of vertices in the graph.

We refer to distributions where the median edge length is greater than the midpoint value as being right-heavy, while when the median edge length is less than the midpoint value we refer to it as left-heavy. For a vertex surrounded by roughly evenly distributed vertices we expect that the distribution of edge lengths for the outgoing edges will be right-heavy. This is due to the fact that the volume increases exponentially when radius increases, when the data is 2 dimensional or higher. This leads to a situation where given an arbitrary radius $r$ from a vertex $v_i$, which forms a hyper-sphere containing other vertices, the majority of other vertices should be at a distance greater than $\frac{r}{2}$ from $v_i$. As such we expect a normal distribution of data points to be somewhat right-heavy.

Figure 5 shows several cases which demonstrate the intuition of our method. Case A is an example where the distribution of edge lengths for a given vertex is extremely right-heavy. This means the majority of nearest neighbours are similar.
in distance to the farthest neighbour. In this case it is reasonable to assume that further nearest neighbours would be of a similar distance, and thus increasing the $K$ connectivity would not be a problem. This kind of distribution would imply that the farthest neighbours are more likely to be intra-class edges, which are good for our purposes. Case A would result in a very high distribution score, as per our definition of it, since the distance of the median from the midpoint is significantly higher than the IQR. This is particularly the case in extremely low $K$ values. To calculate this we need a connectivity of at least 4, or more, outgoing edges, due to its reliance on median and inter quartile range, and thus this is the lower-limit. With such a small number of data points the edge length variance is likely to be extremely high, which will result in a very high edge distribution score.

Case B in Figure 5 shows an example of a somewhat right-heavy distribution of edge lengths. In this case the score would be greater than 0, but less than 1 because the median is not significantly higher than the midpoint of the data, with respect to the IQR. In this case, increasing the $K$ connectivity and adding additional neighbours would further push the median towards the midpoint, which, as we established, is not to be expected in an even distribution of data points. Adding additional neighbours when the distribution of edge lengths is like this will result in additional edges which are a significant distance from the centre of the edge length distribution, and therefore are more likely to be inter-class edges due to their significantly longer resulting edges compared to a majority of the other existing edges. In this case we prefer to lower the $K$ connectivity rather than increase it, which will discard points in the farthest quartile, such that the data is likely to become more right-heavy.

In the final Case C in Figure 5 we see a left-heavy distribution. According to our definition, a left-heavy distribution like this will always result in a score less than zero. If the centre median of the distribution of edge lengths is less than our midpoint then that means the farthest neighbours are significantly more distant from the vertex than the majority of nearest neighbours. In this case, the large variation in edge lengths connected to the farthest neighbours means that these are
more likely to be inter-class edges, which we wish to avoid. This case would result in a score below zero, and would mean we wish to decrease our $K$ value, to discard the farthest edges which are more likely to be inter-class edges.

As we can tell from these three cases, in general a higher distribution score means we wish to increase our $K$ connectivity parameter, and a lower distribution score means we wish to decrease our $K$ value. In the next section we will show exactly when we intend to change the $K$ value according to the distribution score.

4.3. $K$ selection. Our parameter selection method uses the edge distribution score in varying the $K$ value over time. The intuition behind edge distribution score is that the distribution of the edge lengths is relatively predictable as $K$ increases, or else the $K$ connectivity parameter should be decreased. Since this score represents how far the distribution of edge lengths deviates from what we expect in an even distribution of data points, there will be some $K$ value at which there are a large number of inter-cluster edges, which result in an edge distribution score different than what we expect. Similarly when the distribution of edge lengths is indicative of having too low a $K$ value, the score should reflect that. As such, to select the appropriate $K$ value, we must just select based on the value of $K$ that produces a score close to what we expect the distribution to be. This comes in the form of a constant threshold, which we describe below.

The edge distribution score is calculated for all data points, represented by vertices in a sparse graph, in memory. The individual score for each vertex is calculated on its $K$-nearest neighbours on the point-level sparse graph, and then the mean value of those distribution scores is used. This is referred to as the average distribution score (ADS). The ADS us used when we determine when to adjust the $K$ value in our method.

We define a constant threshold $\theta = 2.0$, which means that on average the distribution of edge-lengths should be right-heavy, as we describe previously, and ideally be around one inter-quartile range from the midpoint. Our edge distribution score is calculated as the distance of the midpoint from the median in relation to half of the IQR, and so a $\theta$ of 2.0 represents a distance exactly equal to the IQR - that is, when the edge distribution score of a vertex is exactly 2.0, then the distance from the median $m_i$ to the midpoint $\mu_i$ will be exactly equal to the interquartile range $r_i$. When a vertex produces a score greater than this constant it will indicate that a higher $K$ value is required, whereas if the median length is less than the midpoint distance then the score will be below the threshold, which is a sign the $K$ value might need to be decreased. The threshold constant of $\theta = 2.0$ allows us to maintain an expected right-heavy distribution, whilst also reducing the amount of potential inter-class edges.

The algorithm for the dynamic selection method for $K$ is shown in Algorithm 3. In this algorithm the inputs are $X$, the set of all data points in the stream $\langle x_1, x_2, \ldots, x_s \rangle$, $K$ is the initial $K$ connectivity value which our algorithm will adjust over time, and $M$ is the maximum number of points that RepStream is to store in memory. The symbol $\theta$ denotes our threshold constant, which we define as $\theta = 2.0$, and $\text{margin}$ which allows for a margin of error in the distribution score. Our algorithm allows RepStream to work as normal, but periodically computes the average distribution score, ADS, for each vertex currently in RepStream’s point level sparse graph. The $K$ value is adjusted up or down, or kept the same, depending on what the average distribution score is computed as. The $K$ adjustment process
Algorithm 3 Dynamic $K$ selection algorithm for RepStream. $K$ selection occurs periodically before new vertices are linked into the RepStream sparse graph.

1: procedure RepStreamKSelection($X, K, M$) \> $X$ is the set of data points in the stream, $K$ is the initial connectivity value, and $M$ is the maximum number of points to store in memory
2: \> \> $\theta \leftarrow 2.0$
3: \> $\text{margin} \leftarrow 0.2$
4: \> for each $x_i$ in $X$ do
5: \> \> if $i$ is evenly divisible by $\frac{M}{10}$ then
6: \> \> \> $\text{Score} \leftarrow$ Compute ADS for all vertices in memory
7: \> \> \> if $\text{Score} > \theta + \text{margin}$ then
8: \> \> \> \> $K \leftarrow K + 1$
9: \> \> \> else if $\text{Score} < \theta - \text{margin}$ then
10: \> \> \> \> $K \leftarrow K - 1$
11: \> \> end if
12: \> end if
13: \> end for
14: \> $\text{Neighbours} \leftarrow K$ nearest neighbours of data point $x_i$
15: \> Data point $x_i$ becomes a vertex $v_i$
16: \> $\text{LinkIntoGraphSG}(v_i)$
17: end procedure

takes place every $\frac{M}{10}$ data points, which allows time for the change in $K$ to stabilise before the score is re-computed.

Our reasoning for including a margin is that the value of the distribution score is exceedingly unlikely to exactly equal the threshold parameter, and so it is likely that even at the correct level of connectivity $K$ will alternate between two values repeatedly by fluctuating above and below the threshold. Thus a margin is included to allow for more stability in the $K$ values selected, meaning the algorithm is less sensitive to noise, and more likely to respond only to changes in the data distribution over time. We allow a margin of $10\%$ of the threshold constant, meaning for our threshold of $\theta = 2.0$ there will be no changes if the calculated ADS is between 1.8 and 2.2. For our method we must set an initial $K$ value which is then adjusted over time by our distribution score selection method, as we describe in Section 5 we attempt our best to select the least optimal initial $K$ value possible in order to show how our method works under worst-case conditions.

5. Evaluation.

5.1. Synthetic datasets. We begin with a selection of synthetic datasets, which are designed to present difficult clustering problems. The datasets, aside from DS1 and DS2, are datasets that evolve over time - with distributions that move, change size, or change density during different points of the data stream. While synthetic datasets are not necessarily representative of typical real-world datasets, they can be crafted such that they provide specific challenges which are difficult for clustering algorithms to deal with.

DS1 and DS2 are synthetic datasets used in the original RepStream paper [16] which contains static distributions of data which pose difficult challenges for clustering algorithms to deal with. They can be seen in Figures 6 and 7, which shows...
how the distribution contains classes with concave shapes, as well as classes which are contained within other classes. Synthetic datasets like this are a commonly used way to test the general effectiveness of a clustering algorithm on a static dataset.

**Figure 6.** Visualisation of the DS1 dataset.

**Figure 7.** Visualisation of the DS2 dataset.
SynTest is an evolving dataset that consists of one persistent class that slowly shifts its shape and position over the course of the stream, as well as several other smaller, more dense classes which are transient - appearing and disappearing at various periods of the stream. The larger class is present throughout the whole dataset, and makes up the majority of the data points, while the smaller classes exist for a relatively shorter amount of time. Each of these smaller classes are more dense than the main class, but are present for only a few hundred, to a few thousand time-steps at a time.

Figure 9 shows the presence of the classes in the SynTest dataset. Adjacent marks indicate when the given class is present in the given time window. The shape, size, and position of the classes is shown in Figure 8. Class 1 is always present through the dataset, while the other classes are present for shorter time periods.

Closer is a dataset which simulates the separation between classes in a dataset becoming smaller over time. There are three distinct stages to the dataset: The first 10,000 data points alternate between two classes, all data points are on a two-dimensional plane and each class is normally distributed with a significant level of separation between the two classes. In the next 10,000 data points, the two classes suddenly become much closer together, such that the two classes borders are slightly overlapping. The final 10,000 points return the classes to having a greater degree of separation once again, however one of the classes becomes more dense, while the other becomes less dense. Figure 10 shows these three stages. The changes in this dataset are sudden, conforming to the three distinct stages of the dataset.

The three stages of the dataset were sampled like so:

- Between $T = 0$ to $T = 10,000$ class A was centred at 1,1 and normally distributed with a $\sigma$ of 1 in both the x and y axes. Class B was centred at 8,8 with a standard deviation $\sigma$ of 1.5 on both axes. Points were sampled from these distributions alternately between classes.
Figure 9. The class presence of the classes in the SynTest dataset. A marker indicates that the class is present in the dataset during the given time window.

Figure 10. The evolution of the Closer dataset, showing slices of its 3 sections.

- Between $T = 10,000$ to $T = 20,000$ class A remained the same, while class B was moved to 4,4. Points were sampled from these distributions alternately between classes.
- Between $T = 20,000$ to $T = 30,000$ class A remained the same, while class B was moved to 6,6 with a standard deviation $\sigma$ of 1.5 on both axes. Points were sampled from these distributions randomly by alternately selecting 3 points from class A, followed by 1 point from class B.

5.2. Benchmark datasets. The real-world benchmark datasets described here are examples of a typical usage of stream clustering algorithms. They contain data taken from real-world sources which has been manually labelled such that external validation metrics can be used to determine the effectiveness of algorithms working on them. Because class labels are available we are able to calculate measures such as Purity, Entropy, $F$-Measure, as well as other scores to show the performance of algorithms in an objective sense. Typically, clustering is itself an exploratory process, and class labels and labelled training data is unavailable, thus an algorithm should ideally perform well on benchmark datasets to be trusted to perform well in real-world situations.
The KDD Cup 1999 dataset. The KDD’99 dataset [12] is a well-known benchmark dataset. It is extracted from logs taken from a smart firewall in a network being subjected to simulated and controlled network attacks. It contains high dimensional data, of which we use the 34 numerical features with each data point presented as a 34-dimensional vector. We use a sub-sampled version of the dataset containing 494,020 data points, which is about 10% of the original KDD Cup 1999 dataset. Most of the data in the sub-sampled version of the dataset falls into either the normal traffic class, or one of two major denial-of-service attack classes. A relatively small percentage of the data - less than 2% - are from 20 other network attack types. Each data point is labelled with the type of traffic (normal, or the type of attack) for evaluation purposes.

This KDD 1999 dataset has been used previously in evaluating stream clustering algorithms [1, 2, 5, 16] due to the high variability between classes in the dataset. The various network attacks interrupting the normal traffic represent changes in the distribution of subsequent data points, known as concept drift. This is a significant challenge for clustering algorithms to deal with, making it an excellent dataset for testing how an algorithm deals with dynamic, unpredictable data point distributions over time.

The Tree Cover Type dataset. The Tree Cover dataset [6] is a real-world data stream of a set of features extracted from satellite photos and geological surveys from forested areas of northern Colorado. It contains over 580,000 entries with ground truth labels corresponding to which type of trees grow in each area, and has been previously used as a benchmark dataset for stream clustering [16, 5, 9]. This data represents a naturally evolving stream of data which changes with the environment and climate of each region.

A particularly challenging feature of the Tree Cover dataset is that the classes overlap to some degree in some of the dimensions. This makes clustering particularly challenging as overlapping classes means there’s no spatial separation which can be used to determine where the edges of the classes are. Instead, changes in the density of the data must be used to find where the different classes lie. Since this is a particularly challenging case it is common that even modern, sophisticated clustering methods have a high error rate in separating data.

5.3. Experimental method. To evaluate our method’s efficacy we will examine external validation metrics, specifically the purity and the $F$-measure scores. External validation metrics are commonly used [14, 15] to evaluate the performance of clustering methods in an objective sense against the stated ideal clustering, which is represented by labels for each data point, showing which belong together and which should belong in different clusters.

Our experimental set-up and parametrisation is as follows for our proposed method:

- Memory parameter set to $M = 1000$, as a maximum number of data points in memory at any time.
- The $\alpha$ scaling factor is set to $\alpha = 1.5$ as suggested in the original paper.
- Vanilla normalisation is enabled.
- Decay parameter $\lambda$ at the default value of $\lambda = 0.99$.
- The initial $K$ value is set to the worst possible $K$ value for the dataset, as described below.

We use these same values for all datasets, aside from varying the $K$ value.
Table 2. Best and Worst $K$ values for RepStream, according to $F$ score.

| Dataset  | Best $K$ | Best $F$ score | Worst $K$ | Worst $F$ score |
|----------|----------|----------------|-----------|-----------------|
| DS1      | 7        | 0.7208         | 18        | 0.2767          |
| DS2      | 7        | 0.6371         | 21        | 0.2594          |
| SynTest  | 9        | 0.8614         | 5         | 0.5435          |
| Closer   | 9        | 0.7989         | 5         | 0.4345          |
| TreeCov  | 29       | 0.6108         | 5         | 0.2978          |
| KDD99    | 30       | 0.7898         | 5         | 0.2636          |

As mentioned, we set the initial $K$ values for our proposed method to be the worst possible $K$ value we can select. We do this in order to show that even under the worst case scenario our method can still adapt and produce useful results. To determine which $K$ value is the worst we have run the original RepStream multiple times on each dataset for a range of $K$ values between $K = 5$ and $K = 30$. We then use the class labels to determine which $K$ value produces the lowest mean $F$-measure score for each dataset, and which produces the highest mean $F$-measure score. We set a lower floor for the $K$ value of our algorithm to $K = 5$ and $K = 30$ because our method requires a minimum number of outgoing edges to calculate a median and IQR, and because $K$ values higher than 30 are too high to produce distinct clusters in most cases.

Table 2 shows the $F$-measure scores produced by running the original RepStream algorithm at different $K$ values, using the same parameters listed above. For our experiments on our own algorithm we use the worst initial $K$ values for each dataset as our initial $K$. As can be seen in the table, the overall $F$-measure score can vary a great deal when a sub-optimal $K$ is selected, especially in the worst case scenario. It is worth noting that selecting a $K$ value is not a trivial task, and that since usually one has no access to class labels it is impossible to know whether an appropriate $K$ has been selected.

5.4. Results vs other algorithms. We examine the results for HPStream [2], CluStream [3], ExCC [5], and STRAP [20], using their published results to compare against our own. Rather than listing purity values at every time in the stream, these papers instead publish purity values for specific time slices. We evaluate our algorithm throughout the entirety of the datasets and record the purity values at these same time slices for comparison. These time slices correspond to attacks in the KDD dataset, times where it is most difficult for clustering to occur. We use purity as the chosen measure because, as a common external validation metric, it is the measure used in the published results for these algorithms. We also use the Manhattan distance metric for these evaluations, as it is efficient to compute.

Tree Cover Type dataset, Figure 13 shows the comparisons against our method (Dynamic $K$) HPStream and CluStream, while Figure 14 shows the comparisons between our method, ExCC, STRAP, and HPStream. As we can see, all algorithms are able to achieve a greater than 0.7 purity for each time slice, except for STRAP which drops produces a less than 0.6 purity value at the 20,000 time-slice. Our Dynamic $K$ RepStream method performs favourably against the other algorithms, achieving a higher purity than the other algorithms in 5 of the time-slices, and performing similarly well to the other algorithms in the other time-slices. Earlier time-slices struggle with the purity values, however as the algorithm stabilises over
time the results become more consistently high, and equal or out-perform the other algorithms. The $K$ value selected by our algorithm is initially set at the worst value of 5, but over time our method increases this value to a more appropriate value, reaching up to the maximum $K$ value of 30 at times.

For the KDD dataset, Figure 11 and Figure 12 show the comparisons of our method against the same dataset listed previously. HPStream performs well on the KDD dataset compared to the other algorithms, which is expected because the
algorithm is designed specifically to handle high dimensionality datasets like the KDD dataset. Our extended version of RepStream with the dynamically selected $K$ value, outperforms HPStream and CluStream in all time slices in Figure 12, and also outperforms ExCC, HPStream, and STRAP in half of the time slices in Figure 11. Our method tends to struggle nearer to the start of the dataset when configured with a poor initial $K$ value, but is able to adjust over time.
Additionally, we also compare against the DBStream [11] and D-Stream [8] algorithms, which had implementations available for the Stream package [10] of the R programming language. We used the recommended parameters for each of these algorithms according to the original papers for each algorithm. D-Stream grid-size parameter was set to $\text{len} = 0.05$, its dense and sparse cell thresholds set to $C_m = 3.0$ and $C_m = 0.8$, the decay value $\lambda = 0.998$, and its sporadic cell deletion parameter $\beta = 0.3$. The DBStream algorithm was set with its micro-cluster radius $r = 0.05$, its decay parameter $\lambda = 0.01$, its clean-up interval $t_{\text{gap}} = 1000$, the minimum weight $w_{\text{min}} = 3.0$, and its intersection factor $\alpha = 0.1$. We then ran DBStream and D-Stream on each of our synthetic and benchmark datasets, calculating the purity values at 100-point intervals.

Figure 15 and Figure 17 show the purity scores calculated for the DS1 and DS2 datasets respectively on the DBStream, D-Stream, and our dynamic $K$ method. In both of these datasets a similar phenomenon happens - at first D-Stream and DBStream outperform our method significantly for the first approximately 2000 data points, with our method having a purity score hovering as low as 0.3 purity. After this time, however, our algorithm shows a steep increase in clustering purity until, in both cases, our method outperforms the other algorithms. The low initial purity of our method is due to our attempts at selecting the worst possible initial $K$ value as the input parameters (in this case $K = 18$ for DS1 and $K = 21$ for DS2). After some time to adjust the internal $K$ value in response to the calculated edge distribution score the algorithm performs better, and for both datasets our methods adjusts the $K$ value downwards, hovering around $K = 9$, which according to Table 2 is close to the optimal average $K$ value.

We show an example of the $K$ value adjusting over time in Figure 16. Initially our $K$ parameter begins at the worst possible $K$ value of $K = 18$, but our algorithm very rapidly determines that this $K$ value is too high, and the $K$ value is decreased each time step until it reaches a reasonably stable value after $t = 2000$. The $K$
value doesn’t sit exactly on the optimal value, but hovers at a comparable value, which is why the purity, shown in Figure 15, increases dramatically after $t = 2000$.

For the SynTest dataset, we see in Figure 18 that our method performs comparably well versus the DBStream and D-Stream algorithms. The results are close enough that it is not easy to determine which produces a higher overall purity by looking at the plot alone, so instead we take the mean purity values. D-Stream’s mean purity value is 0.951, our method produces a mean purity value of 0.943,
while DBStream’s mean purity is 0.969. Overall, none of the methods stand out as being clearly superior for this dataset, in fact all achieve high purity clustering throughout the stream, though our method does dip in performance briefly around the 10,000 mark.

The three algorithms perform very differently from each other on the Closer dataset, shown in Figure 19. The DBStream algorithm produces good purity results for the first and last 10,000 data points in the 30,000 point dataset, but performs...
very poorly from 10,000-20,000, dipping as low as 0.5 purity. D-Stream, on the other hand, has purity results which are remarkably consistent throughout the dataset, including in the middle section where the distribution has two overlapping classes. Our dynamic \( K \) method performs exceptionally well for the first and last 10,000 data points, but has variable success during the middle section of the dataset. The purity in this section varies between 0.5 and 0.9, having an average purity of 0.836 during this section. Overall, however our dynamic \( K \) method achieves an overall purity of 0.941, compared to 0.913 and 0.799 for D-Stream and DBStream respectively.

As for the benchmark datasets, Figure 20 shows the purity plots for the comparison algorithms on the Tree Cover Type dataset. The plot is very noisy due to the high level of variability in clustering results over time for all algorithms, however it does show that the DBStream algorithm has the highest amount of variability in its clustering quality, occasionally producing zero clusters, and therefore achieving a nominal purity score of 0. At other times this algorithm produces purity values comparable to D-Stream and our dynamic \( K \) method. Overall, the DBStream method produces a mean purity of 0.508 (or 0.698 if the zero-cluster instances are discarded), the D-Stream algorithm has a mean purity of 0.690 and our dynamic \( K \) method results in a mean purity of 0.843 through the dataset, significantly higher than the other algorithms. While there is still a lot of variability in the results of all algorithms, our method manages to maintain the most consistent and highest purity value on the Tree Cover benchmark dataset. The high degree of overlapping classes in the Tree Cover dataset makes this particularly challenging for algorithms to achieve.

Finally, our method is tested on the KDD Cup 99’ network intrusion benchmark dataset, shown in Figure 21. Note that very clearly all algorithms sit at 1.0 purity throughout the vast majority of the data stream. This is to be expected due to the composition of the KDD dataset, being made up mostly of normal traffic, and two major attack classes. During most of the time, only one class is present and
so any clustering output would result in perfect purity. The sections where there is a decrease in purity corresponds to instances of other classes appearing suddenly in the stream, mixing in with the other traffic. These instances of other classes appearing in the dataset last for a short time before returning to either normal traffic or one of the attack classes. Whilst all algorithms maintain a very high purity overall, the decrease in purity during the short attack instances is smaller in our dynamic $K$ method than in the DBStream and D-Stream methods. Our
method maintains a purity above 0.80 for the whole stream, keeping the majority of the traffic separated from the attack classes more effectively than the comparison algorithms.

Figure 22 shows the $K$ value selected by our method over time while being run on the KDD Cup 99\textquotesingle{} dataset. Initially we set the algorithm to the worst possible initial $K$ value of $K = 5$, but it very rapidly determines that this value is too low, and increases the $K$ value to our cap of $K = 30$. This suggests that it would also tend to go higher if we removed our maximum limit for our evaluations. In general during this dataset the algorithm tends to stay at higher values, between 25 to 30, however it drops down on several occasions, particularly after $t = 3.7 \times 10^5$. Note that while $K = 30$ is the value which provided the highest average purity over the entire length of the dataset, there may be different $K$ values which result in higher purity values at specific times during the stream, as such it is difficult to figure out which is the best exact $K$ value at each time step, especially given that the KDD dataset has a large number of dramatic changes in data distribution over time. The dynamic nature of our algorithm is likely why it performs well compared to standard RepStream, as we show below.

Overall our method performs very well compared to all the algorithms listed above - HPStream, ExCC, CluStream, STRAP, DBStream, and D-Stream. We note that our method was consistently configured such that the initial $K$ value parameter was the worst case possible, while our comparison algorithms were using published results, and parameter values suggested in their original papers. Despite our method having worst-case parametrisation, it was still able to produce comparable results, and even exceed the performance of other stream clustering algorithms after being given time to adjusts its internal $K$ parameter.

5.5. Results vs RepStream. We wish to also evaluate our method against the ideal results which can be produced by RepStream in an ideal case. As such, we run RepStream using the ideal optimal $K$ value, and compare it against our method. Again, we run our dynamic $K$ selection by selecting the most sub-optimal initial $K$ value and let the value automatically change over time according to the computed distribution score.

For these evaluations we use $F$-measure as our external validation metric. One major problem with purity, as noted in [14], is that purity is an unreliable indicator of performance, despite its popularity as a comparison tool between clustering algorithms. Purity has the problem has no penalty for producing too many clusters, and splitting classes up into multiple clusters. Notably, it is possible to reach a perfect purity value when each data point is treated as a separate cluster. Instead we choose to use the $F$-measure score to compare our results, as this avoids this problem, rewarding both the precision and the recall of the clustering output, rather than just the precision as with the purity value.

Figure 23 and Figure 24 show the $F$-measure achieved by RepStream configured with the optimal $K$ value against our dynamic $K$ method set with the worst possible initial $K$ value. As one would expect, optimally configured RepStream performs better than our dataset, after about 2000 data points our method catches up in $F$-measure score dramatically. Our method adjusts its internal $K$ value by $\pm 1$ for every 100 data point, and so it takes a period of time for it to achieve a stable $K$ value if it is initially very far off from where it should be. As such in this case it takes approximately 2000-3000 data points before its performance is able to adjust. Our method shows a dramatic upturn in the $F$-measure score after this time period,
Despite having the worst possible initial setting, once the internal $K$ parameter stabilises the clustering quality is comparable to that of the optimally configured RepStream. Also marked on Figures 23 and 24 is the $F$-measure produced by RepStream configured at the worst possible single $K$ value for comparison purposes.
Figure 25 has our dynamic $K$ method compare against RepStream set to the optimal value of $K = 9$, and also compared to the worst possible $K$ value of $K = 5$. As expected, our method begins sub-optimally, and over the first 2000 data points compares very poorly to the optimally configured RepStream instance. However, after this time period the $F$-measure score produced by our method dramatically improves, often matching the score of our benchmark. There is a slight decrease in $F$-measure compared to the optimal in the 7000 range, but this difference is only about 0.1 in score versus the optimal.

The Closer dataset is shown in Figure 26. RepStream in this experiment has the $K$ value set to $K = 9$ while our dynamic $K$ method has an initial $K$ value of $K = 5$, which the worst initial $K$ value according to Table 2. Again, for the first 2000 data points the difference in $F$-measure values is very noticeable, but after this time our method adjusts and almost entirely matches the optimally configured RepStream instances. Notably, there are brief times when our method even outperforms the base RepStream algorithm. On average though, our method has almost identical performance to the RepStream algorithm over most of the dataset having an overall $F$-measure score of 0.841 against the optimal RepStream overall score of 0.861, differing by only 0.02, even when taking into account the initial 2000 data points of poor performance as our method adjusts.

The results for the Tree Cover Type benchmark dataset is shown in Figure 27. This dataset is particularly difficult to cluster as it contains overlapping classes. Overall clustering purity is likely to be imperfect because of this. As such, even the optimally configured RepStream instance set to $K = 29$ has an overall $F$-measure value of 0.611. This is, however, not far off from our dynamic $K$ method which has an overall average score of 0.592, a difference of only around 0.019 overall. As shown in Table 2 the average $F$-measure score produced by RepStream if it is kept at the
The worst possible $K$ value is 0.298, so our method is able to perform exceptionally well even in the face of having the worst initial parameters.

Similar is true with the KDD Cup 99 benchmark dataset. The main challenge in clustering this dataset is combining all normal traffic, and all denial-of-service attack classes into single clusters whilst also successfully differentiating the other attack classes.
classes from each other. Thus, for this non-typical dataset the way to produce the best overall F-measure score is to combine as many points together into the same cluster as possible. This is why the highest $K$ value we tested for RepStream was optimal. Our method overall performs within 0.1 F-measure value of the optimal $K$, producing 0.788 F-measure compared to the 0.790 purity of the optimally configured RepStream. This score is vastly improved from the 0.264 F-measure score which would’ve been produced with standard RepStream set to the same initial $K$ value as our method was set to, which demonstrates how much our dynamic $K$ method can help in cases of poor initial parametrisation. As is evident in the plot there are time periods where our method outperforms standard, optimally-configured RepStream, which as we mentioned before is likely due to the dynamic nature of our method, and the ability to use different $K$ values over time.

6. Discussion. As we noted in Section 1 a major problem with stream clustering algorithms is the sensitivity to user-set initial parameters. Our dynamic $K$ adjustment method allows for the internal $K$ value of RepStream to change over time in response to changes in the data distribution.

Table 3 shows the F-measure value RepStream configured at the optimal $K$ value, in terms of F-measure score, the worst possible $K$ value, as well as our dynamic $K$ method. Our method is configured using the same initial $K$ parameter used in the worst F-measure column of the table. The exact $K$ values used can be found in Table 2. As can be seen, however, our dynamic $K$ method allows significant improvements over RepStream configured at the same values.

Our method works in improving the results and making the RepStream algorithm less sensitive to initial parameters. Through all of our experiments we’ve used the same parameters - $\alpha$ scaling factor is set to $\alpha = 1.5$, vanilla normalisation, decay factor at the default $\lambda = 0.99$, and an initial $K$ set to the worst possible initial $K$.
Table 3. Comparison of our dynamic $K$ method versus RepStream with optimal and worst $K$ values.

| Dataset | Best F-Measure | Worst F-Measure | Dynamic $K$ |
|---------|----------------|----------------|-------------|
| DS1     | 0.7208         | 0.2767         | 0.5153      |
| DS2     | 0.6371         | 0.2594         | 0.4137      |
| SynTest | 0.7989         | 0.5435         | 0.7091      |
| Closer  | 0.8614         | 0.4345         | 0.8410      |
| TreeCov | 0.6108         | 0.2978         | 0.5920      |
| KDD99   | 0.7898         | 0.2636         | 0.7882      |

value for each dataset. For almost all instances this initial $K$ value was set to 5, as per Table 2, and so we would suggest that using an initial $K$ of 5 and letting the value automatically stabilise over time would be a satisfactory way to configure our method. As such our algorithm can perform without need for further tuning as it will automatically adjust its own internal $K$ parameter according to the computed average edge distribution score.

7. Conclusion. In this paper we have introduced a method for automatic parameter selection in RepStream using edge distribution as a computed measure. RepStream [16] is a sophisticated clustering algorithm, employing a combination density and graph based clustering approach, but one notable problem with it, and other stream clustering algorithms, is the reliance on user-set parameters. Here, we have extended the RepStream algorithm, proposing changes which remove the need for users to tune parameters to the dataset. Since data clustering is an exploratory processes this is particularly important, because one can’t assume prior knowledge about the data to be analysed.

Our method consisted of making use of the $K$-nearest neighbour directed sparse graph employed by the RepStream algorithm and computing the edge distribution score. With this measure we gradually raise or lower the $K$ value over time to keep the distribution score close to a threshold, which is set to minimise the number of edges that span between classes.

Our edge distribution score, described in detail in Section 4, is a computed measure which reflects how closely the edge lengths resemble what we would expect from a stable cluster, and makes use of the fact that areas of relatively continuous density have much less variation in the length of edges in a $K$-nearest neighbour context. Using this measure we propose a method for increasing and decreasing the $K$ value over time to adjust to changes in the distribution of data points in a stream.

Our experiments in Section 5 showed that when our method was configured using the $K$ value which would perform poorest, in terms of $F$-measure, in standard RepStream, our dynamic $K$ method was able to recover and produce significantly improved clustering results, shown in Table 3. We propose that using our method, and an arbitrary low initial $K$ value of $K = 5$ we can produce clustering output which is of consistent quality and which matches or even outperforms other sophisticated stream clustering algorithms, with respect to purity, when those algorithms are run with recommended parameters.
REFERENCES

[1] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen and C. Sohler, Streamkm++: A clustering algorithm for data streams, Journal of Experimental Algorithmics, 17 (2012), Article 2.4, 30 pp.

[2] C. Aggarwal, J. Han, J. Wang And P. Yu, A framework for projected clustering of high dimensional data streams, in Proceedings 2004 VLDB Conference, Elsevier, 2004, 852–863.

[3] C. C. Aggarwal, P. S. Yu, J. Han and J. Wang, A framework for clustering evolving data streams, in Proceedings 2003 VLDB Conference, Elsevier, 2003, 81–92.

[4] J. P. Barddal, H. M. Gomes and F. Enembreck, SNCStream: A social network-based data stream clustering algorithm, in Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC ’15, SAC ’15, ACM Press, New York, New York, USA, 2015, 935–940.

[5] V. Bhatnagar, S. Kaur and S. Chakravarthy, Clustering data streams using grid-based synopsis, Knowledge and Information Systems, 41 (2014), 127–152.

[6] J. A. Blackard and D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Computers and Electronics in Agriculture, 24 (1999), 131–151.

[7] F. Cao, M. Estert, W. Qian and A. Zhou, Density-based clustering over an evolving data stream with noise, in Proceedings of the 2006 SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2006, 328–339.

[8] Y. Chen and L. Tu, Density-based clustering for real-time stream data, in Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’07, ACM, ACM Press, New York, New York, USA, 2007, 133–142.

[9] A. Forestiero, C. Pizzuti and G. Spezzano, A single pass algorithm for clustering evolving data streams based on swarm intelligence, Data Mining and Knowledge Discovery, 26 (2013), 1–26.

[10] J. Forrest, Stream: A framework for data stream modeling in r, Bachelor Thesis, Department of Computer Science and Engineering, SMU.

[11] M. Hahsler and M. Bolaos, Clustering data streams based on shared density between micro-clusters, IEEE Transactions on Knowledge and Data Engineering, 28 (2016), 1449–1461, http://ieeexplore.ieee.org/document/7393836/.

[12] S. Hettich and S. D. Bay, The UCI KDD Archive, http://kdd.ics.uci.edu, 1999.

[13] A. K. Jain, M. N. Murty and P. J. Flynn, Data clustering: A review, ACM computing surveys (CSUR), 31 (1999), 264–323.

[14] S. Kaur, V. Bhatnagar and S. Chakravarthy, Stream clustering algorithms: A primer, in Big Data in Complex Systems, Springer, 9 (2015), 105–145, http://link.springer.com/10.1007/978-3-319-11056-1.

[15] H. Kremer, P. Kränen, T. Jensen, T. Seidl, A. Bifet, G. Holmes and B. Pfahringer, An effective evaluation measure for clustering on evolving data streams, in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2011, 868–876.

[16] S. Lühr and M. Lazarescu, Incremental clustering of dynamic data streams using connectivity based representative points, Data & Knowledge Engineering, 68 (2009), 1–27, http://linkinghub.elsevier.com/retrieve/pii/S0169023X08001092.

[17] J. Schneider and M. Vlachos, Fast parameterless density-based clustering via random projections, in Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management - CIKM ’13, ACM, ACM Press, New York, New York, USA, 2013, 861–866.

[18] N. Wattanakitrungroj, S. Maneeroj and C. Lursinsap, Bestream: Batch capturing with elliptic function for one-pass data stream clustering, Data & Knowledge Engineering, 117 (2018), 53–70, http://www.sciencedirect.com/science/article/pii/S0169023X17301027.

[19] G. Widmer and M. Kubat, Learning in the presence of concept drift and hidden contexts, Machine Learning, 23 (1996), 69–101.

[20] X. Zhang, C. Furthleihner and M. Sebag, Data streaming with affinity propagation, in Proc. Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2008, 628–643.

[21] A. Zhou, F. Cao, W. Qian and C. Jin, Tracking clusters in evolving data streams over sliding windows, Knowledge and Information Systems, 15 (2008), 181–214.

Received for publication April 2019.
E-mail address: ross.callister@postgrad.curtin.edu.au
E-mail address: DucSon.Pham@curtin.edu.au
E-mail address: M.Lazarescu@curtin.edu.au