Connectivity Based Stream Clustering Using Localised Density Exemplars

Sebastian Lühr and Mihai Lazarescu

Department of Computing, Curtin University of Technology,
Kent Street, Bentley 6102, Western Australia
{S.Luhr, M.Lazarescu}@curtin.edu.au

Abstract. Advances in data acquisition have allowed large data collections of millions of time varying records in the form of data streams. The challenge is to effectively process the stream data with limited resources while maintaining sufficient historical information to define the changes and patterns over time. This paper describes an evidence-based approach that uses representative points to incrementally process stream data by using a graph based method to cluster points based on connectivity and density. Critical cluster features are archived in repositories to allow the algorithm to cope with recurrent information and to provide a rich history of relevant cluster changes if analysis of past data is required. We demonstrate our work with both synthetic and real world data sets.

1 Introduction

Stream mining is an increasingly important area of research that aims to discover interesting information from continually evolving data sets whose size, combined with limitations in available memory and computational resources, typically constrains our ability to perform timely batch processing of the data. Instead, we desire means by which to incrementally maintain current and historical models of the data with which to perform queries. Stream data mining has been heavily investigated in the past five years with most efforts concentrated on the clustering aspect of the problem. Of the algorithms developed, however, only a small number can handle difficult clustering tasks without expert help, typically provided in the form of the number of partitions expected or the expected density of clusters. Moreover, none of these attempt to build a *selective* history to track the underlying changes in the clusters observed.

We present a sparse-graph based stream mining approach that employs representative cluster points to incrementally process incoming data. The graph based description is used because it allows us to model the spatio-temporal relationships in a data stream more accurately than is possible via summary statistics. A critical aspect of our research has been to avoid rediscovery of previously learned patterns by reusing useful cluster information. For this reason, a repository of knowledge is used to capture the history of the relevant changes occurring in the clusters over time. The use of the repository offers two major benefits.

First, the algorithm can handle recurrent changes in the clusters more effectively by storing a concise representation of persistent and consistent cluster features. These features assist in the classification of new data points belonging to historical cluster distributions within an evolving data stream. The retention of such features is important as they permit the algorithm to discard older data points in order to adhere to constraints in available memory and computational resources while continuing to store useful cluster features.

Second, the repository provides a concise knowledge collection that can be used to rebuild a cluster's overall shape and data distribution history. It is therefore possible to archive core cluster features for future off-line analysis when a recall of historical changes is desired.

2 Related Work

Several important stream mining algorithms have been introduced in recent years. One of the first data stream mining methods to consider the archival of cluster information was CluStream [1]. The algorithm uses microclusters to capture and record statistical summary information suitable for off-line analysis. CluStream is, however, best suited to situations in which clusters are spherical, reducing the algorithm's suitability to many real world data sets.

HPStream, a modification of CluStream to enable clustering of high dimensional data, was proposed in [2]. The algorithm employs a data projection method to reduce the dimensionality of the data stream to a subset of dimensions that minimise the radius of cluster groupings. However, the underlying assumption remains that clusters in the projected space remain spherical in nature.

Most recently, a multi-density clustering technique that extends the DBSCAN [3] density-based clustering approach to stream mining was proposed in [4]. The algorithm, DenStream, extends DBSCAN by adapting the original density based connectivity search to a microcluster approach.

An incremental version of the DBSCAN was earlier proposed in [5]. As with DBSCAN, the algorithm obtains groupings based on the nearest neighbourhood connectivity of points within an a priori defined radius known as the ϵ-neighbourhood. Incremental DBSCAN is limited to keeping only the most recent data points in memory and is therefore likely to discard possibly reusable cluster information without consideration towards its value.

A well known algorithm, Chameleon [6] uses the hMETIS [7] multilevel graph partitioning algorithm to recursively divide a sparse graph into micro-clusters. These clusters are then iteratively merged based on user specified thresholds for measures of relative interconnectivity and closeness.

None of the algorithms mentioned provide a means to *selectively* archive historical information. Those algorithms that facilitate archiving instead tend to store summary statistics with which general changes in clusters can be revisited.

3 Clustering Stream Data via Representative Points

Our cluster representation involves the use of dynamically updated sparse graphs that, when used in conjunction with a repository of representative vertices, allows us to rebuild a cluster's history and to rapidly adapt to significant changes previously observed. The RepStream algorithm that we propose aims to capture such change in order to recall it at some future time should the change reoccur. RepStream is a single phase incremental algorithm that updates two sparse graphs of k-nearest neighbour connected vertices in order to identify clusters among data points. The first graph captures the connectivity relationship amongst the most recently seen data points and to select a set of representative vertices. The second graph is used to track the connectivity between the chosen representative vertices. The connectivity of the representative vertices on both graphs then forms the basis for the algorithm's clustering decision making.

The representatives we use offer two major advantages. First, since representative vertices typify a set of nearby data points, decisions made at this level improves performance by requiring only a subset of the data to be considered. Second, representative vertices are associated with a measure of usefulness which allows the algorithm to selectively retain highly valued representatives as historical descriptors of the cluster structures. This retention allows the algorithm to accurately classify new data points arriving within a region of the clustering space where the non-representative vertices have since been retired.

3.1 Preliminaries

Given a data stream P of time ordered points $P = \{p_1, \dots p_{|P|}\}$, we wish to find groupings of points sharing similar properties. We define a cluster c to be a set of points $c = \{p_1 \dots p_{|c|}\}$ where each point p_i is a multidimensional vector $p_i = \{p_{i,1} \dots p_{i,D}\}$ of D dimensions. Let C be the set of clusters $C = \{c_1 \dots c_{|C|}\}$.

Let the set $G = \{g_1 \dots g_{|P|}\}$ be the ideal cluster assignments for points P such that the j^{th} element g_j correctly labels point p_j. We aim to assign labels to data points such that each point is correctly classified or any misclassification is minimised. The distance between point p_i and point p_j is given as $\mathrm{D}(p_i, p_j)$.

Points are inserted into a directed k-nearest neighbour (K-NN) sparse graph $\mathrm{SG}(V, E)$ of vertices $V = \{v_1, \dots v_{|V|}\}$ and edges $E = \{e_1, \dots e_{|E|}\}$ such that the i^{th} vertex v_i corresponds to point $p_i \in P$. Each edge is an ordered pair $\langle u, v \rangle$ of vertices such that $u, v \in V$. The sparse graph representation is used as it provides a rich representation of relationships that is otherwise not available by only labelling data points.

Updates to the sparse graph requires knowledge of each vertex's nearest neighbours. Let $\mathrm{NN}(v_i)$ be a function that provides an ascending distance ordered array of the nearest neighbours of a vertex v_i and let $\mathrm{NN}(v_i, j)$ be a function that gives the j^{th} nearest neighbour of v_i. Let $\mathrm{RC}(v_i)$ be a function that provides a set of vertices reciprocally connected to a vertex v_i. We also let $\mathrm{IE}(v_i)$ be a function for determining the incoming edges directed at vertex v_i.

Let $R = \{r_1, \dots r_{|R|}\}$ be a set of representative vertices on SG such that $\forall x, r_x \in V$ and let $\mathrm{RSG}(W, F)$ be a directed k-nearest neighbour representative sparse graph which links the vertices $W = \{w_1, \dots w_{|W|}\}$ via edges $F = \{f_1, \dots f_{|F|}\}$. An edge in F is an ordered pair $\langle u, v \rangle$ of vertices such that $v, u \in R$. Let $\mathrm{NN}_{\mathrm{R}}(r_i)$, $\mathrm{NN}_{\mathrm{R}}(r_i, j)$ and $\mathrm{RC}_{\mathrm{R}}(r_i)$ be functions that provide the nearest neighbours, the nearest j^{th} neighbour and the set of vertices that are reciprocally linked to a representative vertex r_i on RSG.

Definition (predictor) Let a representative r_i be a predictor if r_i satisfies the condition that $|\mathrm{IE}(r_i)| < \frac{k}{2}$.

Definition (exemplar) Let a representative r_i be an exemplar if r_i satisfies the condition that $|\mathrm{IE}(r_i)| \geq \frac{k}{2}$.

Definition (representative vertex) Representative vertices represent at most k non-representative vertices on the sparse graph SG. A vertex v_i is made representative if at any time $\nexists j, v_j \in \mathrm{RC}(v_i), v_j \in R$, that is, if it is not reciprocally connected to an existing representative. Representatives are further categorised into a set of exemplar representatives $R^E = \{r_1^E, \ldots r_{|R^E|}^P\}$ and predictor representatives $R^P = \{r_1^P, \ldots r_{|R^P|}^P\}$ such that $R^P \cup R^E = R$ and $R^P \cap R^E = \emptyset$.

Clustering decisions in RepStream are made via vertices representative of regions within the cluster space. At each time step a new point p_i is observed in the data stream and added to the sparse graph $\mathrm{SG}(V, E)$ as vertex v_i. A new vertex joins an existing cluster if it is reciprocally connected to a representative $v_j \in R$. Should no such representative vertex exist then v_i is itself made representative. The creation of the new cluster may trigger an immediate merge with an existing cluster if the conditions for merging are met.

3.2 Merging and Splitting Clusters

Cluster splits and merges are made by monitoring both the reciprocal connectivity of vertices on the representative sparse graph as well as their *relative density* based on the proximity of their nearest neighbours on SG. The trigger condition for either of these events is the creation or removal of *density-related* links.

Definition (relative density) The density of representative vertex $r_i \in R$ is determined by the function $\mathrm{RD}(r_i) = \frac{1}{|\mathrm{NN}(r_i)|} \sum_{j=1}^{|\mathrm{NN}(r_i)|} \mathrm{D}(r_i, \mathrm{NN}(r_i, j))$.

Definition (density-related) Given a density scaler α, two representatives r_i and r_j are density-related if: $\mathrm{D}(r_i, r_j) \leq \mathrm{RD}(r_i) \cdot \alpha$, and $\mathrm{D}(r_i, r_j) \leq \mathrm{RD}(r_j) \cdot \alpha$ and $r_j \in \mathrm{RC}_{\mathrm{R}}(r_i)$.

Merges are therefore triggered when an update to the connectivity of vertices on RSG sees the creation of a new reciprocal connection that is also density-relate or when the addition or removal of a vertex affects the density of two existing representatives that are reciprocally connected such that their density-related status is altered. Monitoring the connectivity and relative density of representatives enables the algorithm to evolve with changes in the data.

Split checks are executed when the loss of a density-related link between two vertices on RSG is detected. A standard $O(n^2)$ region growing algorithm that follows the density-related links of the representative vertices was employed to perform split checks.

3.3 Knowledge Repository

A significant aim of RepStream is to retain those representative vertices that prove, over time, to be useful in representing the shapes and distributions of clusters. Such vertices are retained for as long as possible (subject to available resources) via a repository defined as an ordered vector of vertices $S = \langle s_1, \ldots s_{|S|} \rangle$ sorted in ascending usefulness.

Definition (representative usefulness) The usefulness of a representative vertex r_i is defined by the decay function: $\text{usefulness}(r_i, \mathit{count}) = \log(\lambda) \cdot (\text{current time} - \text{creationTime}(r_i) + 1) + \log(\mathit{count} + 1)$. Here λ is a user specified decay rate and *count* is the representative vertex's reinforcement count. This count is incremented when an incoming vertex is found to be a nearest neighbour of r_i.

The decay function ensures a monotonic ordering of vertices in the repository with respect to the passing of time. In our implementation of RepStream we chose to index the repository using a AVL binary search tree [8]. Updating the reinforcement count of a representative vertex that has already been added to the repository requires only two tree operations: the removal of the vertex and then its subsequent reinsertion following an increment to its reinforcement count. The least useful representative vertex can be rapidly found by traversing to the AVL tree node with the lowest usefulness score.

New additions to the repository are made whenever a new representative vertex is created until resource constraints have been reached. At this point only the most useful repository members are retained. This is achieved by comparing the least useful repository member with other non-repository representatives whenever their reinforcement count is incremented. Vertices retired from the repository are immediately unlinked from both graphs and archived to disk.

3.4 Singularities

The occurrence of many identical points within a data stream is captured via *singularities*, a special case of representative vertices intended to succinctly and efficiently represent such occurrences.

Definition (singularity) A representative vertex $r_i \in R$ is termed a singularity when $\sum_{j=1}^{k} \mathrm{D}\left(r_i, \mathrm{NN}\left(r_i, j\right)\right) = 0$ and $\left|\mathrm{NN}\left(r_i\right)\right| = k$.

Singularities represent a collection of identical points that offer no new information to the clustering process, yet whose inclusion in the sparse graphs would require the retirement of otherwise useful vertices. New points that are identical to a singularity are therefore immediately deleted in order to avoid the overhead of unnecessary sparse graph updates and to maintain the information value of the repository. The occurrence of identical points is not lost, however, as they are represented by a singularity's reinforcement count.

Singularities are unable to be assigned non-zero density measures and as such do not lose their singularity status once it is acquired. This ensures that the presence of a singularity is permanently captured by the algorithm even though its nearest neighbours may be retired over time. Representative vertices are unable to form density-related links to singularity vertices.

3.5 Data Retirement

Processing and memory constraints require the algorithm to discard information over time. This is accomplished by prioritising the disposal of data such that the least useful information for clustering is removed first. Non-representative vertices are queued on a first in, first out basis and removed whenever resource limitations are reached. Representative vertices that are not stored in the repository are considered to have little retentive value and are also removed via the deletion queue. All other representative vertices remain in memory; their deletion is instead managed via the repository update procedure.

The removal of a vertex requires updates to the sparse graph and the representative sparse graph. Graph updates are made to ensure that any vertices with edges directed at the removed vertex are updated with a new nearest neighbour. Representative vertices are also updated to ensure that their local density is adequately maintained.

4 Experimental Results

The performance of RepStream was evaluated using synthetic and real world data sets. Our real world data sets consisted of the KDD-99 Cup network intrusion data and the forest canopy type data described in [9]. The synthetic data was designed to test the algorithm's capacity to cluster a difficult set containing a variety of arbitrarily shaped clusters of different densities. The real world data sets, in contrast, were selected to investigate the practical application of the approach on large evolving data streams.

Cluster purity [10] was used to measure how well data is classified over a horizon of the previous h data points. The purity of a cluster c_i is defined as: $\mathrm{CP}(c_i) = \frac{1}{|c_i|} \max_k (\sum_{j=1}^{|c_i|} \mathrm{r}(v_j, k))$ where $\mathrm{r}(v_j, k)$ is 1 if $\mathrm{class}(v_j) = k$, else 0.

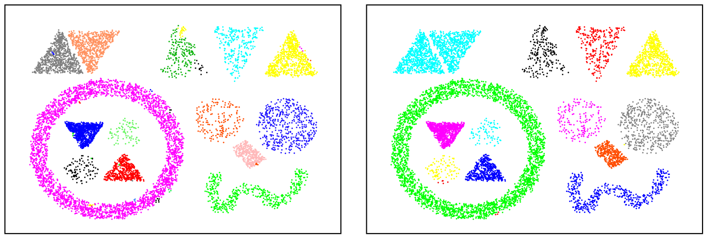
The total clustering purity is then found by averaging over all clusters via: $\mathrm{TCP}(C) = \frac{1}{|C|} \sum_{i=1}^{|C|} \mathrm{CP}(c_i)$.

The algorithm was constrained to using only 10 MiB of memory and the decay factor used in all experiments was set to $\lambda = 0.99$. The chosen purity horizons were selected to correspond with previous work in clustering data streams [1, 2]. The KD-Tree [11] was used to perform nearest neighbour searches.

4.1 Synthetic Data

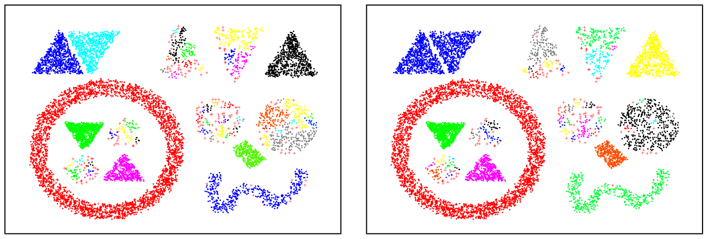
The clustering quality of RepStream was first compared against an incremental version of DBSCAN [5] using the hand crafted synthetic data set. DBSCAN was selected for comparison as this algorithm employs a density based method of clustering known to perform well with arbitrarily shaped clusters. However, DBSCAN is limited to operating at a single density and is therefore expected to exhibit difficulties when dealing with this data set. As DBSCAN relies on a priori knowledge of the optimal cluster density, we repeated each of the DBSCAN experiments using a variety of values for ϵ. The minimum number of points required to form a cluster was set to 5. The data was presented to the algorithms using a randomised point ordering and the Manhattan distance was used to compute the similarity between points.

Figure 1 depicts the RepStream clustering of the data using the optimal parameter set $k = 4$ and $\alpha = 4.0$. These results show that the algorithm was able to cluster the arbitrarily shaped clusters well. The discovered clusters are sub-optimal, however, with some minor fragmentation evident. The separate clustering of these points is not considered an error, however, as their location and density suggests that these points may, indeed, belong to separate clusters when compared to the remaining points.



(a) (b)

Fig. 1: RepStream clustering of the synthetic data highlighting the performance difference between a neighbourhood connectivity of (a) $k = 4$ and (b) $k = 5$ when $\alpha = 4.0$.



(a) (b)

Fig. 2: DBSCAN clustering of the synthetic data set with (a) $\epsilon = 15$ and (b) $\epsilon = 16$.

Increasing the density scaler from $\alpha = 4.0$ to a higher value of $\alpha = 6.0$ did not correct this clustering. Decreasing the scaler did, however, result in increased fragmentation. An increase of the neighbourhood connectivity successfully overcame the fragmentation issue as shown in Figure 1(b).

In contrast, DBSCAN was found to produce well formed higher density clusters with an ϵ-neighbourhood parameter of $\epsilon = 15$. The lower density clusters, however, were found to be highly fragmented with the presence of a significant number of unclustered points as shown in Figure 2(a). Decreasing the density with $\epsilon = 16$ marginally decreased the cluster fragmentation, as seen in Figure 2(b), though at the expense of the incorrect merging of the two top left triangular clusters.

4.2 Network Intrusion Data

The KDD Cup-99 data set features 494,020 network connection records derived from seven weeks of raw TCP logs consisting of both regular network traffic as well as 24 types of simulated attacks within a military local area network. Of the dimensions available, 34 continuous valued features were used for clustering and a single outlier point was removed.

RepStream was tested using a purity horizon of $h = 1,000$. The Manhattan distance function was used to compute the similarity of data points from features that were normalised on-the-fly. A point $p_i = \{p_{i,1} \ldots p_{i,D}\}$ of D dimensions was normalised in each dimension d using the formula $p'_{i,d} = \frac{p_{i,d}}{\sum_{j=1}^{|P|} p_{j,d}}$ where $|P|$ refers to the number of points in memory at any given time. The nearest neighbourhood connectivity was set to $k = 9$ with $\alpha = 1.5$.

The purity results in Figure 3 show that RepStream is able to accurately differentiate between different types of attack connections. The accuracy of RepStream was also evaluated against published results reported on the same data set for the HPStream, DenStream and CluStream algorithms. The results of the comparisons, depicted in Figure 4 and in Figure 5, shows that in most cases RepStream was able to classify network connections as well as or with higher accuracy than HPStream, DenStream and CluStream. The data stream sample times were chosen to match those reported in [1, 2].

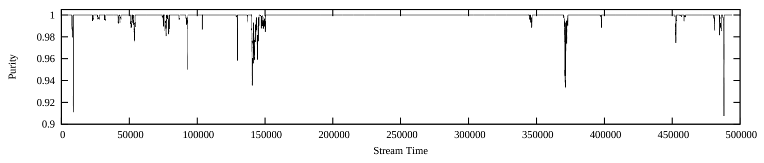
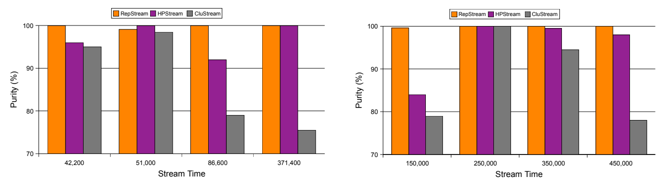
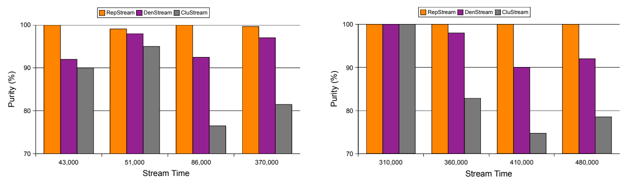


Fig. 3: RepStream purity throughout the network intrusion data stream.



(a) (b)

Fig. 4: Purity measures of RepStream, HPStream and CluStream using available published results on the KDD Cup 1999 data set with (a) $h = 200$ and (b) $h = 1,000$.



(a) (b)

Fig. 5: Purity measures of RepStream, DenStream and CluStream using available published results on the KDD Cup 1999 data set with (a) $h = 200$ and (b) $h = 1,000$.

4.3 Forest Cover Data

The forest cover data set contained 581,012 records consisting of a total of 54 geological and geographical features that describe the environment in which trees were observed. Records also included the ground truth as to which of seven different types of canopy were present on the trees. Attributes consisted of a mixture of continuous and Boolean valued data, the latter taking values from the set $\{0, 1\}$. Dimensions were normalised as described in Section 4.2 and the Manhattan distance function was used to measure the similarity between points. Parameters used on this data set were $k = 9$ and $\alpha = 1.5$.

Figure 6 shows the purity measured over the data stream with $h = 1,000$. RepStream is seen to classify the canopy types with an accuracy typically $\geq 85\%$. The jagged appearance of the purity plots suggest that the algorithm is coping

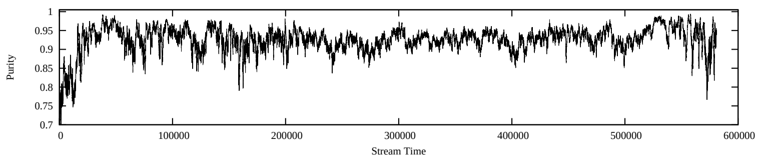


Fig. 6: RepStream purity throughout the tree cover data stream.

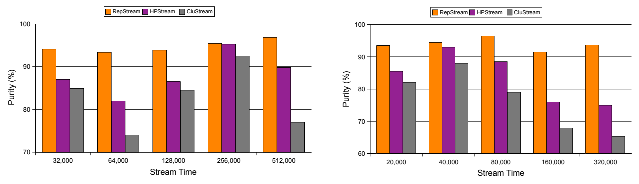


Fig. 7: Purity measures of RepStream, HPStream and CluStream using available published results on the forest tree cover data set with (a) $h = 200$ and (b) $h = 1,000$.

with a more dynamic data set than compared to the network intrusion experiment in Section 4.2; a premise confirmed through inspection of the data. RepStream's purity measurements were evaluated against HPStream and CluStream using the results published in [2]. Figure 7 depicts the result of this comparison, showing that the algorithm was able to classify the tree data with consistently more accuracy than the competing algorithms.

4.4 Scale-Up Experiments

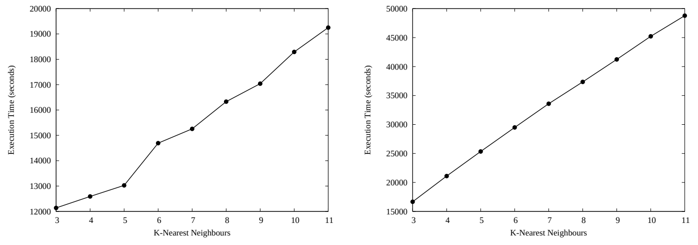
We investigated the execution time of the algorithm with respect to neighbourhood connectivity and the length of the data stream. Scale up experiments were executed on Mac OS 10.4 running on an Intel 2.33GHz Core 2 Duo processor.

A near linear relationship between connectivity and execution time was discovered in the network intrusion results in Figure 8a. The forest data set produced a similar relationship as shown in Figure 8b. Execution time with respect to the length of the data stream is shown in Figure 9.

Whereas the tree data set in Figure 9b shows an expected linear relationship between the number of points processed and the execution time, the network data set in Figure 9a displays significant flattening out due to efficient processing of identical points within the stream. Connectivity was set to $k = 5$ and a density scaler of $\alpha = 1.5$ was used to process both data sets.

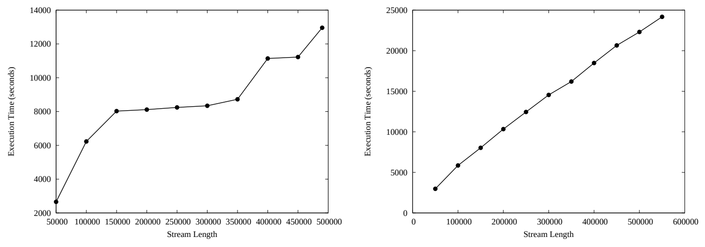
5 Conclusions

This paper has introduced a graph-based incremental algorithm for clustering evolving stream data. Experimental results demonstrated that the algorithm was able to effectively classify both synthetic and real world data sets. The algorithm was compared against an incremental implementation of DBSCAN and shown to robustly handle clusters of complex shapes, sizes and densities. DBSCAN, in contrast, was shown to be hampered by a static density threshold ill suited towards stream processing. Results on real world data sets showed that RepStream was able to more accurately classify well known network intrusion and forest canopy data sets than three of the most popular stream data clustering algorithms: DenStream, HPStream and CluStream.



(a) (b)

Fig. 8: Execution time of RepStream clustering (a) the network intrusion data and (b) the forest canopy data as the k-nearest neighbours are increased.



(a) (b)

Fig. 9: Execution time of RepStream clustering the (a) network intrusion data and (b) the forest canopy data as the stream length is increased.

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proc. 29th Int'l Conf. Very Large Data Bases. (2003)
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for projected clustering of high dimensional data streams. In: Proc. Very Large Data Bases. (2004) 852–863
3. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. 2nd Int'l Conf. Knowledge Discovery and Data Mining. (1996) 226–231
4. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: Proc. Sixth SIAM Int'l Conf. Data Mining. (2006)
5. Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X.: Incremental clustering for mining in a data warehousing environment. In: Proc. 24rd Int'l Conf. Very Large Data Bases. (1998) 323–333
6. Karypis, G., Han, E.H., Kumar, V.: Chameleon: Hierarchical clustering using dynamic modeling. Computer **32**(8) (August 1999) 68–75
7. Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S.: Mutlilevel hypergraph partitioning: Application in VLSI domain. IEEE Trans. Very Large Scale Integration (VLSI) Systems **7**(1) (March 1999) 69–79
8. Knuth, D.: The Art of Computer Programming. 3rd edn. Volume 3. (1997)
9. Blackard, J.A., Dean, D.J.: Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Computers and Electronics in Agriculture **24**(3) (1999) 131–151
10. Aggarwal, C.C.: A human-computer interactive method for projected clustering. IEEE Trans. Knowledge and Data Engineering **16**(4) (2004) 448–460
11. Bentley, J.L.: Mutlidimensional binary search trees used for associative searching. Communications of the ACM **18**(9) (September 1975) 509–517