Divisive Clustering of High Dimensional Data Streams

David P. Hofmeyr   Nicos G. Pavlidis   Idris A. Eckley

August 11, 2015

Abstract

Clustering streaming data is gaining importance as automatic data acquisition technologies are deployed in diverse applications. We propose a fully incremental projected divisive clustering method for high-dimensional data streams that is motivated by high density clustering. The method is capable of identifying clusters in arbitrary subspaces, estimating the number of clusters, and detecting changes in the data distribution which necessitate a revision of the model.

The empirical evaluation of the proposed method on numerous real and simulated datasets shows that it is scalable in dimension and number of clusters, is robust to noisy and irrelevant features, and is capable of handling a variety of types of non-stationarity.

1 Introduction

High dimensional data stream clustering is increasingly relevant as automatic data generation and acquisition technologies are adopted in diverse applications. Data streams are encountered in a variety of settings. These include: computer network traffic monitoring, Web page requests, customer click streams, sensor networks, as well as transactions data from stock and foreign exchange markets, to name a few. The volume of data involved in these applications is far too large to fit in main memory. Hence random access to past observations is costly. Linear scans are the only acceptable access method in terms of computational efficiency (Guha et al., 2003; Silva et al., 2013). A defining property of data streams is that the population distribution is subject to changes over time. This phenomenon is known as *population drift* (Babcock et al., 2002). These characteristics pose significant challenges for clustering. Streaming clustering algorithms must be incremental, with time and storage requirements independent of the size of the stream. In addition they must be capable of adapting to population drift by revising the cluster structure, distinguishing emerging clusters from noise, and discarding expired clusters (Jain, 2010).

The majority of existing streaming clustering algorithms consist of two components. The first is an online component that incrementally updates data structures that summarise the data sample. The second component operates offline, and performs the actual clustering, operating on the data summaries, rather than the original data samples (Aggarwal et al., 2003; Cao et al., 2006). However, clustering algorithms for static datasets invariably involve parameters which are application dependent, e.g. the number of clusters in k-means. Using such methods in the offline component of a streaming algorithm implicitly assumes that appropriate values for these parameters remain constant over the duration of the data stream. This is hard to justify in the presence of population drift. Existing algorithms attempt to handle population drift through simple heuristics, like sliding windows (Aggarwal et al., 2003; Kranen et al., 2009) and forgetting factors (Aggarwal et al., 2004; Cao et al., 2006), which are user-determined and static over the length of the stream. The important aspect of change detection is largely ignored (Silva et al., 2013). Finally, the majority of traditional clustering algorithms rely on the Euclidean distance between data samples, which becomes less meaningful as dimensionality increases (Kriegel et al., 2009). Few streaming algorithms are capable of handling very high dimensional data, and in general these are limited to detecting clusters only in axis parallel subspaces (Aggarwal et al., 2004; Ntoutsi et al., 2012; Hassani et al., 2102, 2014).

The framework we propose draws on the standard nonparametric statistical definition of clusters as regions of high density in the underlying probability distribution (Hartigan, 1975; Cuevas and Fraiman, 1997; Rigollet and Vert, 2009). In this approach, a *high-density cluster* is defined as a connected component of the level set of the (unknown) density function. When the density is unimodal the level set is connected, otherwise it can be connected or not. If it is disconnected, high-density clusters correspond to regions around modes of the density (Menardi and Azzalini, 2014). Identifying connected regions of high density of an unknown density is a challenging task even in moderate dimensions. Existing methods rely on an approximation of the density (Azzalini and Torelli, 2007; Cuevas et al., 2001), or attempt to infer local properties of the density (Menardi and Azzalini, 2014; Stuetzle and Nugent, 2010). Due to the curse of dimensionality such approaches are only effective on problems in up to tens of dimensions (Menardi and Azzalini, 2014). In higher dimensions, graph theoretic formulations have been used to approximate these high density regions (Cuevas et al., 2001; Rinaldo and Wasserman, 2010).

The influential DBSCAN algorithm (Ester et al., 1996), combines a kernel density estimate using uniform kernel with a graph theoretic approach to determine clusters. Data points whose density exceeds a chosen threshold, $\lambda > 0$, are considered high-density points. A graph is constructed by connecting each high density point with each other point within a radius equal to the bandwidth of the kernel density estimator. Connected components of the graph define clusters, while singletons are interpreted as noise. This approach is efficient from a computational perspective, but only applies to the uniform kernel.

A basic weakness of algorithms that attempt to identify high-density clusters for a single, user-defined density level, is that both the number and the shape of the clusters depends on the choice of this parameter. In addition using a single density threshold can fail to detect clusters of varied densities (Ankerst et al., 1999; Stuetzle, 2003). To overcome this limitation one can compute the clustering structure that arises by considering all possible values of the density level. The collection of clusters which arises is known as the *cluster tree* (Hartigan, 1975). Recent algorithms that attempt to estimate the cluster tree include OPTICS (Ankerst et al., 1999), and Gslclust (Stuetzle and Nugent, 2010). A general approach to detect clusters at a local level from a cluster tree is discussed in (Campello et al., 2013).

In this article we propose a framework for streaming data clustering that relies on a different approach to high-density clustering. Instead of attempting to estimate high-density clusters directly, it partitions the data sample hierarchically using linear separators (hyperplanes) that pass through regions of low density. It thereby avoids splitting high-density clusters. This was first proposed for static data clustering by Tasoulis et al. (2010). An attractive feature of this approach from a computational perspective is that it requires only one-dimensional projections to identify low density separators. Expanding on this we propose a framework for streaming data clustering, which we refer to as High-dimensional Streaming Divisive Clustering (HSDC), that is able to: (i) identify clusters of arbitrary orientation; (ii) estimate the number of clusters automatically using a statistically motivated divisive procedure; (iii) update the clustering result incrementally without any offline component; (iv) utilise information about structural variation of the model to influence forgetting, instead of relying on static parameters; and (v) identify changes in the population distribution that require the revision of the clustering model. This is achieved through synthesising and extending results from incremental dimensionality reduction, kernel density estimation, and change detection.

The remaining paper is organised as follows. In Section 2 we discuss some existing data stream clustering algorithms. Section 3 gives a more formal description of the problem we consider and discusses challenges associated with a data stream implementation. Section 4 describe our methodology for introducing components to the model, as well as how to accommodate population drift. In Section 5 we give a brief summary of the algorithmic structure of the method, as well as investigate the computational complexity of the model updates. In Section 6 we document the results of an extensive simulation study and performance on publicly available data sets. Finally in Section 7 we give some concluding remarks.

2 Related Work

Many existing data stream clustering algorithms extend classical clustering algorithms, such as k-means, k-medians, fuzzy c-means, and DBSCAN, to the data stream framework (Guha et al., 2003; Zhang et al., 1996; Aggarwal et al., 2003, 2004; Cao et al., 2006; Kranen et al., 2009).

One of the most influential data stream clustering algorithms is CluStream (Aggarwal et al., 2003). CluStream uses *microclusters* to summarise the data received by the algorithm incrementally. These microclusters are then clustered offline using a weighted k-means algorithm. Microclusters store first and second order summary statistics of spatial and temporal information, and possess useful additive, subtractive and multiplicative properties, making them well suited to windows and forgetting factors. To handle non-stationarity, CluStream stores snapshots of the microclusters which enable it to approximate the clustering result over a window, which is specified by the user.

HPStream (Aggarwal et al., 2004) is a modification of CluStream to handle high dimensional data. Distance calculations are performed within axis parallel subspaces so as to minimise the radii of the microclusters. Assigning potentially differing subspaces to the clusters negates the additive and subtractive properties of the microclusters, and so HPStream treats the microclusters as actual clusters rather than data summaries. Snapshots also become meaningless, and so temporal variation is handled by fixed forgetting factors.

DenStream (Cao et al., 2006) is a density-based algorithm that uses microclusters. To handle noise it distinguishes between *outlier* and *potential* microclusters, the latter defined by a threshold on the weighted number of points falling within a sphere of fixed radius. Weights are exponentially decreasing functions of time, enabling the algorithm to adapt to population drift. The offline component of DenStream is a variant of DBSCAN, thus, enabling the estimation of the number of clusters, and the detection of clusters of arbitrary shape. Recently proposed extensions of DenStream include HDDStream (Ntoutsi et al., 2012), PreDeConStream (Hassani et al., 2102), and DMMStream (Amini et al., 2014). HDDStream and PreDeConStream handle high dimensional data by scaling up the contribution of *preferred* dimensions within distance calculations. DMMStream enables the detection of density connected clusters on differing scales, through the use of mini microclusters.

Grid based density clustering algorithms have also been proposed. DStream (Chen et al., 2007) is similar to DenStream, except that dense grid cells (cells containing relatively high approximate integrated density) are used instead of microclusters. The number of grid cells however depends exponentially on the dimension of the data. DDStream (Jia et al., 2008) is an extension of DStream which allows the absorption of data at the boundaries of clusters into adjacent dense cells, thereby reducing the number of *active* grid cells.

A potential feature of data streams is that the rate at which new data is observed can vary over time. *Anytime* algorithms are able to produce a clustering result after any amount of processing time, but are also capable of refining this result when more time is available (Kranen, 2011). Anytime stream clustering was first discussed in relation to the ClusTree algorithm (Kranen et al., 2009). ClusTree stores a hierarchy of microclusters, with each internal node representing an aggregation of its children. Arriving data are inserted at the root and traverse the hierarchy via the nearest microcluster at each level. This insertion is halted if there is insufficient time. Halted data can "hitchhike" further down the hierarchy with similar arriving data at a later stage. In this way

CluStree is capable of handling not only extremely high velocity data streams, but also streams in which the velocity varies. SubClusTree (Hassani et al., 2014) extends ClusTree to high dimensional applications by establishing multiple hierarchies, each existing within a different subspace.

A comprehensive review and categorisation of existing data stream clustering algorithms is provided in two recent surveys (Aggarwal, 2013; Silva et al., 2013). A review focused on density based methods for streaming data is provided in (Amini et al., 2014).

3 Problem Description

Our aim is to generate a hierarchical partition of the Euclidean space $\mathbb{R}^d$ such that the modes of a probability density, f, over $\mathbb{R}^d$, are uniquely contained within different cells of the partition. The density, f, is not known, and instead we receive a sequence of realisations of the random variable X with density f. The learning process is constrained by standard memory and computation limits associated with data stream learning. We do not assume that f is constant in time, and so modify the model as changes in the empirical distribution of realisations suggests is necessary.

This problem can be formulated in the context of high density clustering, wherein the modes of the density f can be associated with its *level sets*.

Level Set For level $\lambda \geq 0$ and density function f, the level set of f above λ, or λ *level set of* f, is defined as $\{x \in \text{Support}(f) | f(x) \geq \lambda\}$.

As λ increases, the λ level set centers around the modes of f above λ, and therefore the number of modes, or clusters, can be associated with the number of maximal connected subsets of the level sets. We refer to these maximal connected subsets as the *components* of the level set. Identifying the components of level sets of a high dimensional probability density function is extremely costly in terms of computational effort, and often these are approximated using graph theoretic formulations (Cuevas et al., 2001; Rinaldo and Wasserman, 2010). In addition, the density function f is unknown and must be approximated. Standard methods, such as Kernel Density Estimation (KDE), become less effective at accurately representing the underlying probability density as dimensionality increases (Scott, 2009). Building an approximation of f incrementally in a data stream setting introduces yet further challenges, since only summaries of the data can be stored and hence further approximations are necessary.

The dePDDP algorithm (Tasoulis et al., 2010) attempts to separate the modes of a distribution via a hierarchy of low density separating hyperplanes. The algorithm recursively projects (subsets of) the data into a one dimensional subspace and splits the projected data above and below the lowest antimode of their estimated density. The KDE of the projected data provides an upper bound on the value of the full dimensional KDE, as shown in the following lemma, which is adapted from Tasoulis et al. (2010).

Lemma 1 *Let $\mathcal{X} = \{x_1, ..., x_N\}$ be a d-dimensional data set, and let $v \in \mathbb{R}^d$ have unit length and let $b \in \mathbb{R}$. Let $\hat{f}$ denote the d-dimensional kernel density estimate of the distribution of $\mathcal{X}$ with bandwidth matrix hI using the multivariate Gaussian kernel. Let $\hat{f}_1$ be the univariate kernel density estimate of the distribution of $v \cdot \mathcal{X}$ with bandwidth h using the univariate Gaussian kernel. Then for any $x \in \mathbb{R}^d$ s.t. $v \cdot x = b$,*

$$\hat{f}(x) \leq h^{d-1} \hat{f}_1(b).$$

Splitting at the lowest antimode of the projected density estimate, therefore, avoids intersecting high level sets of the full dimensional estimated density. Separating clusters by regions of low density has also been shown to yield more stable clusters (von Luxborg, 2010), which fits well with the possibility of smooth time variations in f. While this approach avoids the explicit estimation of the full dimensional density, the model structure is limited to cases where the modes of the density can be separated by a hierarchy of hyperplane separators. In particular it is limited to cases in which the convex hulls of the modes are non-overlapping. Despite this limitation, the approach has shown good empirical performance in a number of high dimensional applications (Boley, 1998; Tasoulis et al., 2010). The dePDDP algorithm projects data onto their first principal component, using the justification that directions with high variability are likely to display high *between cluster* variability. This will tend to lead to good separation of the clusters, and hence low antimodes in the projected density estimate.

A hyperplane can be parameterised by a vector $v \in \mathbb{R}^d$ and scalar $b \in \mathbb{R}$ as the set $\{x|v \cdot x = b\}$. No generality is lost by assuming that v is unit length. We adopt the approach of learning v, which we will refer to as the *projection vector*, from the full dimensional realisations of X, and then using the projections of these realisations onto v to inform our choice of b, which we refer to as the *split point*. The problem of incremental principal component analysis is well studied (Artac et al., 2002; Li et al., 2003; Weng et al., 2003), and computationally efficient algorithms exist. In order to determine the split point, projections of the data onto the projection vector are used to approximate the density of the random variable $v \cdot X$. Low empirical density regions in this density approximation suggest the location of low density hyperplanes. Combining these provides a readily available framework for an online version of dePDDP, such as that adopted by SPDC (Tasoulis et al., 2012). Such a straightforward implementation, however, has important limitations.

Incremental updates to the projection vector, which we henceforth index by t to indicate the t-th step estimate, mean that the sample of projections at time t is given by $\{v_1 \cdot x_1, v_2 \cdot x_2, ..., v_t \cdot x_t\}$, which is not a sample from the random variable $v_t \cdot X$. With successive updates to v_t the accuracy of the projected points at estimating the empirical distribution of $v_t \cdot X$ diminishes, which affects the accuracy of the spit point. Futhermore, if the splitting rule at a node in the hierarchy is updated with each observation, then the sets of data being passed to its children will vary. This variability propagates down the hierarchy and renders projection vector updates and splitting decisions at lower levels increasingly inaccurate and unstable. Moreover, if the underlying distribution changes in time, these projections and splitting rules are rendered even more inaccurate, unless these changes are suitably accommodated.

In what follows we detail our approach to overcoming these limitations. We describe the incremental updates to a node of the hierarchy. The time indices, t, relate to the t-th update to the node, and not the t-th update to the entire hierarchy. Similarly, the t-th datum refers to the t-th datum received by the node, and not the t-th datum in the entire stream.

4 Methodology

In this section we discuss in detail the three components of the High-dimensional Streaming Divisive Clustering (HDSC) framework. HSDC constructs incrementally a hierarchical clustering model which consists of a collection of separating hyperplanes. The hyperplanes pass through regions of low density, and are orthogonal to directions of high variance. These hyperplane separators are maintained within the model until there is sufficient evidence indicating that they no longer pass through regions of low density. Whenever this occurs the hyperplane in question, and therefore the part of the hierarchy rooted at it, is removed from the model and the corresponding node is re-initialised.

Section 4.1 details how we find high variance projection directions incrementally using the CCIPCA algorithm (Weng et al., 2003). Section 4.2 describes how low density hyperplanes are identified using information from the projected data only. In Section 4.3 we discuss how population drift can be handled within this framework.

4.1 Learning High Variance Projections

The k-th principal component of a data set, $\mathcal{X}$, is given by the k-th largest eigenvector of its covariance matrix. Many incremental methods for principal component estimation require that the full covariance matrix be approximated, leading to high computation and storage costs for large problems. The CCIPCA algorithm (Weng et al., 2003) instead focuses directly on the eigenproblem $\mathrm{Cov}(\mathcal{X})u = \lambda u$. The algorithm is based on the recursion,

$$v_t = \frac{t-1}{t} v_{t-1} + \frac{1}{t} \frac{x_t \cdot v_{t-1}}{\|v_{t-1}\|} x_t, \tag{1}$$

where $\{x_t\}_{t=1}^{\infty}$ is a sequence with zero mean. Weng et al. (2003) have shown that the recursion of Eq. (1) converges almost surely to $\pm\lambda u$ for the maximum value of λ, i.e., u is the first principal component. Lower order eigenvectors are found by first projecting data into the null space of all approximate higher order eigenvectors.

Early updates to the projection vector are highly variable, making it more challenging to approximate the marginal distribution along it. Passing promising projection vectors down the hierarchy to act as initial projection vectors in the child nodes can help to ameliorate this problem. When a node is split, a hyperplane orthogonal to its projection vector is introduced to the model. The truncation induced by a separating hyperplane tends to reduce the variability in the normal direction (i.e., along the projection direction) more than in directions orthogonal to it. The second most highly variable direction is therefore a good candidate for a high variance projection in the child nodes. We investigate a variation on the basic HSDC model, which we call *projection inheritance*, in which each node (besides the root node) learns both its own projection and a high variance projection to be passed to its children. So that this inheritance is not lost to natural variations in the data, it is given additional weight in subsequent updates equal to the number of updates already undergone. Orthogonality to the parent's projection vector is also enforced in subsequent updates after being passed down. This increased stability in the projection vector can be crucial to estimating the distribution along it. If the updates are highly variable, then the projections made onto it will be less reliable at representing the target distribution.

Below we describe formally the updates to the projection and inheritance vector with the arrival of the t-th datum x_t. Let u_t be the (unnormalised) projection vector at time t, and z_t the inheritance vector. Also, if u_t was initialised by inheritance, let N be the number of updates undergone prior to inheritance and let p be the projection vector of the parent node (otherwise assume $p = \mathbf{0}$ and adopt the convention that $0/0 = 0$).

$$\begin{aligned}
\bar{x}_t &= \frac{t-1}{t}\bar{x}_{t-1} + \frac{1}{t}x_t \\
x_C &:= x_t - \bar{x}_t \\
x_0 &:= x_C - \frac{x_C \cdot p}{\|p\|^2} p \\
u_t &= \frac{t+N-1}{t+N} u_{t-1} + \frac{1}{t+N} \frac{x_0 \cdot u_{t-1}}{\|u_{t-1}\|} x_0 \\
x_C &= x_C - \frac{x_C \cdot u_t}{\|u_t\|^2} u_t \\
z_t &= \begin{cases} x_C, & t = 1 \\ \frac{t-1}{t} z_{t-1} + \frac{1}{t}\frac{x_C \cdot z_{t-1}}{\|z_{t-1}\|} x_C, & \text{otherwise.} \end{cases}
\end{aligned}$$

In subsequent sections we will assume that the projection vector is normalised, and denote it by v_t.

4.2 Splitting Based on a Projected Sample

High density clustering associates clusters with modes of the underlying probability density. Introducing a new split to the hierarchical model is therefore only done when the corresponding truncation of the density, induced by the hierarchical partition of $\mathbb{R}^d$, contains multiple modes. Assessing the modality of a high dimensional probability density is difficult, however by considering one dimensional projections of the underlying random variable, X, we only need to assess the modality of a univariate sample. Cases exist in which the full dimensional density is unimodal, but in which the marginal distribution of a univariate projection is multimodal, and vice versa, and in these cases the accuracy of our model will be compromised as components may be split between different elements of the partition.

4.2.1 Assessing Modality

In this subsection we assume that we observe a univariate sample corresponding to the projections of the underlying random variable. The *excess mass* test (Müller and Sawitzki, 1991) is used to asses the modality of a sample from an unknown distribution function F over $\mathbb{R}$, with density function f. The excess mass of f at level λ is defined as,

$$E(\lambda) = \int_{\mathbb{R}} (f(x) - \lambda)^+ \mathrm{d}x.$$

The excess mass therefore measures the integrated density above level λ. The excess mass can also be formulated in terms of the distribution function F,

$$E(\lambda) = \sup_{I_1,\dots,I_{c(\lambda)}} \sum_{i=1}^{c(\lambda)} \left(F(I_i) - \lambda\|I_i\|\right), \tag{2}$$

where $c(\lambda)$ is the number of connected components of the λ level set of f and $\|I\|$ is the diameter of the set I. The supremum is taken over all collections of size $c(\lambda)$ of disjoint intervals. The latter formulation allows for the empirical excess mass based on a sample, $\mathcal{X}$, from F to be calculated by replacing F in (2) with the empirical distribution $F_{\mathcal{X}}$, defined as

$$F_{\mathcal{X}}(z) = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \mathbb{I}[z \geq x].$$

In practice, the number of connected components $c(\lambda)$ will not be known, and so the empirical excess mass is compared for different values. We use the notation $\hat{E}_c(\lambda)$ to mean the empirical excess mass for c intervals. The excess mass statistic for comparing c_1 with $c_2 > c_1$ components is defined as

$$\Delta(c_1, c_2) = \sup_{\lambda} \{\hat{E}_{c_2}(\lambda) - \hat{E}_{c_1}(\lambda)\}.$$

The larger $\Delta(c_1, c_2)$, the more evidence in favour of c_2 over c_1. In our context, we are interested in whether or not a density has more than one mode, and therefore are interested in the case $c_1 = 1, c_2 = 2$. This case can equivalently be assessed via the *dip* (Hartigan and Hartigan, 1985). The dip of a distribution function F over $\mathbb{R}$ measures the departure from unimodality of F and is given by the supremal distance between F and the distribution function with unimodal density for which this supremal distance is minimal. Formally,

$$\mathrm{Dip}(F) = \min_{U \in \mathcal{U}} \|F - U\|_\infty,$$

where $\mathcal{U}$ is the class of distribution functions with unimodal density. It has been shown that the dip is equal to half the *excess mass* statistic $\Delta(1, 2)$, and can therefore be used equivalently to assess multimodality when the dip of $F_{\mathcal{X}}$ is considered. To assess the significance of the dip or excess mass, a null unimodal distribution is specified and the quantiles under this null distribution estimated using Monte Carlo simulation. The benefits of using the dip relate to the computationally efficient algorithm given by Hartigan (1985), which is linear in the sample size. The corresponding unimodal distribution can be extracted from the algorithm, and hence the value of λ corresponding to the excess mass can be obtained. We make use of this value of λ in the approximation of the underlying density (Section 4.2.4).

While the calculation of the dip is linear in the size of the sample, it cannot be calculated incrementally. In the context of a data stream, this violates the fixed storage and computation limits. To remedy this, we propose an approximation method which requires bounded memory and computation time. This is achieved by approximating the sample using a fixed number of compact intervals which are dynamically adjusted to always contain the entire sample. Storing only the endpoints of the intervals and the number of data falling in them allows us to construct an approximation of the sample which leads to a lower bound on the dip of the empirical distribution of the sample. Thus, this approximation method leads to a uniformly more conservative test of unimodality, which fits well with our objective to avoid prematurely splitting clusters.

4.2.2 Compactly Approximating the Sample

Our approximation method relies on the notion of a uniform set, which is defined as follows.

Uniform Set Let $\mathcal{X}$ be a finite sample in $\mathbb{R}$ and $I = [a, b], a \leq b \in \mathbb{R}$. Then the *uniform set* of $\mathcal{X}$ and I is defined as

$$Unif(\mathcal{X}, I) = \bigcup_{i=1}^{n} \left\{ m + \frac{i-1}{n-1}(M - m) \right\},$$

where $n = |\mathcal{X} \cap I|$, $m = \min\{\mathcal{X} \cap I\}$, and $M = \max\{\mathcal{X} \cap I\}$.

We lose no generality by assuming that the endpoints of the interval I, a and b, are elements of $\mathcal{X}$. For the purpose of approximating the empirical distribution, the uniform set replaces the empirical distribution on I with the distribution function of the random variable $Y_I := \frac{\|I\|}{|\mathcal{X} \cap I| - 1} U + \min\{I\}$, where $U \sim \mathcal{U}[0, |\mathcal{X} \cap I| - 1]$ is the discrete uniform random variable on $\{0, 1, ..., |\mathcal{X} \cap I| - 1\}$. Notice that we have again adopted the convention $0/0 = 0$. For a collection of disjoint intervals $I_1 < I_2 < ... < I_k$, which jointly contain the entire sample, the approximate distribution is given by,

$$\tilde{F}(x) = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{k} |\mathcal{X} \cap I_i| F_{Y_{I_i}},$$

where $F_{Y_{I_i}}$ is the distribution function of Y_{I_i}, defined as above.

With the arrival of a new datum, x, $\tilde{F}$ must be updated such that the number of intervals used does not exceed the predefined limit. If x lies within one of the existing intervals, then no adjustment to the above formulation is necessary. Otherwise, an interval $I = [x, x]$ is added, and then two adjacent intervals are replaced with the convex hull of their union. The intervals *merged* in this way are the adjacent pair which minimise the supremal distance between $\tilde{F}$ before and after the merger. If intervals $i, i+1$ are merged, then this distance is given by,

$$\frac{1}{|\mathcal{X}|} \max \left\{ \left| |\mathcal{X} \cap I_i| - \left\lceil \frac{\|I_i\|}{\|I_{i:i+1}\|} \right\rceil |\mathcal{X} \cap I_{i:i+1}| \right|, \right.$$
$$\left. \left| |\mathcal{X} \cap I_i| + 1 - \left\lceil \frac{\min I_{i+1} - \min I_i}{\|I_{i:i+1}\|} \right\rceil |\mathcal{X} \cap I_{i:i+1}| \right| \right\},$$

where $I_{i:i+1} = I_i \cup I_{i+1}$. With the above formulation of $\tilde{F}$ we arrive at the following result, the derivation of which can be found in the appendix. We also discuss therein how a slight modification to the dip algorithm allows one to calculate the dip of the sample approximation in $\mathcal{O}(k)$ time, where k is the number of intervals.

Lemma 2 *Let $\mathcal{X}$ be a univariate sample of distinct points. For any collection of disjoint, compact intervals $I_1 < I_2 < ... < I_k$ satisfying*

$$\mathcal{X} \subset \bigcup_{j=1}^{k} I_j,$$

we have $Dip(\tilde{F}) \leq Dip(F_{\mathcal{X}})$.

This result ensures that the approximation method used cannot lead to additional false discovery of multimodality when compared with the true sample of observations. While the result is stated for an unweighted sample, it also holds for weighted samples for which the data within each interval are given the same weight. This is important as in the next subsection we describe how reweighting the data can be useful in better approximating the distribution of a sample projected onto a vector which is continually being updated.

4.2.3 Accommodating a Shifting Projection

Our aim is to approximate the distribution of the projected random variable, $v \cdot X$, where $\|v\| = 1$. However, we only observe realisations of a sequence of random variables $v_t \cdot X_t$, where under the assumption that $X_1, X_2, ...$ are i.i.d., we know that v_t converges almost surely to a vector v. Even under this assumption, the realisations $v_t \cdot x_t$ still represent a sample from a nonstationary distribution due to the shifting projection v_t. With consecutive updates to v_t, the accuracy of the observed projections as a representation of the target distribution diminishes. The influence of these observations on the approximate distribution should therefore diminish with subsequent updates. *Forgetting factors* impose a decaying weight mechanism to control the influence of past observations on the current estimate, however they are difficult to tune in practice. If $w_{t,i}$ is the weight associated with observation i at time t, then $w_{t,i} = (1 - \lambda_t)w_{t-1,i}$, where $\lambda_t \in [0, 1]$ is the forgetting factor at time t. Weights associated with new observations are initialised at 1, and so we have,

$$\begin{aligned} W_t := \sum_{i=1}^{t} w_{t,i} &= 1 + (1 - \lambda_t) \sum_{i=1}^{t} w_{t-1,i} \\ &= 1 + (1 - \lambda_t) W_{t-1}. \end{aligned}$$

Our approximate distribution $\tilde{F}$ is a mixture of discrete uniform distributions, in which the weight associated with each component is equal to the number of atoms in its support. Using forgetting factors we can adjust these weights to obtain a more accurate approximation to the distribution on the current projection. The update to $\tilde{F}$, which we now index by t to represent the approximation after t observations, is therefore given by,

$$\tilde{F}_t = \frac{1}{W_t} F_{Y_{[x_t, x_t]}} + (1 - \lambda_t) \frac{W_{t-1}}{W_t} \tilde{F}_{t-1},$$

where x_t is the observation at time t. If the number of intervals then exceeds the upper limit, the merging of two adjacent intervals is performed as described in Section 4.2.2. As λ_t approaches zero, past and present observations become equally weighted, while higher values of λ_t increase the influence of recent observations on $\tilde{F}$. In problems with an explicit loss function, forgetting factors can be tuned using stochastic gradient descent (Haykin, 1999; Anagnostopoulos et al., 2012; Pavlidis et al., 2011), but this is not true in our context. We propose an adaptive scheme which is complementary to the incremental estimation of the projection in that it uses information about the angles between consecutive updates to v_t to quantify the variability of the projected distribution over time. In detail, with the arrival of the $(t+1)$-th datum, first v_t is updated as described in Section 4.1, and then λ_t is set to,

$$\lambda_{t+1} = \min\{\Lambda, \gamma\lambda_t + (1-\gamma)\arccos(v_{t+1}\cdot v_t)\}, \tag{3}$$

where $\Lambda \in (0,1]$ is a chosen maximum forgetting factor. We use an exponentially weighted moving average (EWMA) with parameter $\gamma \in (0,1)$ to smooth the impact on λ_t of isolated large fluctuations in $\arccos(v_{t+1}\cdot v_t)$ arising from natural variation in v_t. The following proposition states that the adaptive scheme of (3) converges almost surely to 0, whenever v_t converges almost surely. The distribution approximation will therefore stabilise as the projection converges, which is almost sure under the CCIPCA algorithm as long as the underlying distribution does not change.

Proposition 3 *If $\{v_t\}_{t=1}^{\infty}$ converges almost surely and $\|v_t\| = 1$ $\forall t$, then $\lambda_t \xrightarrow{a.s.} 0$, where λ_t is as in (3).*

Reweighting the projected data in this way means that the approximate dip or excess mass must be compared with the quantiles for a sample of size equal to the *effective sample size* of the reweighted data, which we calculate as the sum of the weights in the approximation $\tilde{F}$.

When the null hypothesis of unimodality is rejected based on the dip of the approximate sample distribution, the associated node is split. The projection direction and split point are then kept fixed. In the following we describe how to approximate local minima in the density along the projection, thereby allowing one to determine such a split point based on this density.

4.2.4 Approximating Anti-modes

The distribution function $\tilde{F}$ is discontinuous, and therefore using this distribution directly to approximate an antimode in the underlying density is challenging. A standard approach to generating smooth density approximations is to consider a convolution with a smooth distribution function, having density K. The density K is referred to as a *kernel.* The convolution of a discrete distribution associated with a random variable Y, having mass function $p(y)$, with a smooth distribution, gives rise to the canonical kernel density estimate,

$$\hat{f}(x) = \frac{1}{h}\sum_{y\in\text{Support}(Y)} p(y)K\left(\frac{x-y}{h}\right).$$

The parameter h is called the *bandwidth*, and controls the smoothness of the resulting density estimate. A common choice of kernel is the standard Gaussian distribution given by,

$$K(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}.$$

In this case the bandwidth directly relates to the standard deviation of the smoothing density. The support of the random variable underlying our approximation $\tilde{F}$ still contains $|\mathcal{X}|$ atoms, despite the compression of its representation. The evaluation of the associated kernel density estimate at a fixed number of points is $\mathcal{O}(|\mathcal{X}|)$. Knowledge that $\tilde{F}$ represents a mixture of discrete uniform distribution functions leads us to instead consider the convolution of the corresponding continuous uniform distribution functions with the kernel K. For those intervals which contain only a single point, there is no associated continuous distribution, and so the standard kernel convolution above is used. The convolution of the uniform density on $[a,b], a < b \in \mathbb{R}$ with the Gaussian distribution with variance h^2 is given by

$$f(x) = \frac{\Phi((x-a)/h) - \Phi((x-b)/h)}{b-a},$$

where Φ is the distribution function of the standard Gaussian random variable. With this formulation the associated kernel density estimate has k components, rather than $|\mathcal{X}|$.

How to choose h remains a very active area of research, and no universal guidelines exist for every context. We use the equivalence of the dip and excess mass tests, and the implications of the rejection of their null hypotheses, to inform our choice of h. Rejection of the null hypothesis of the excess mass test is equivalent to favouring two λ level set components over one. The corresponding value of λ can be extracted from the dip algorithm. We choose h such that the associated density estimate has λ level set consisting of two components. The minimum such value of h is chosen, as this leads to more sharply defined modes and anti-modes.

4.3 Handling Population Drift

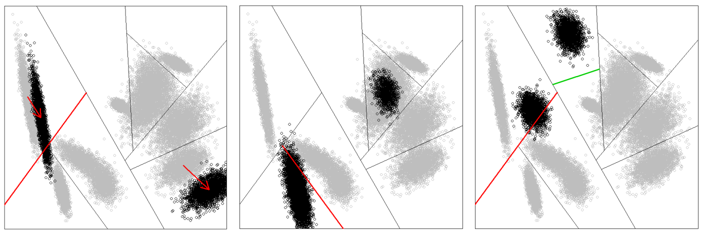
A generic approach to detect time-variations in f that invalidate our clustering model is to sequentially consider the hypothesis that each hyperplane passes through a region of low relative f density (*low-density separation hypothesis*). If this hypothesis is false then a revision of the corresponding part of the hierarchical model is necessary. Notice that the low-density separation hypothesis is not a hypothesis of overall stationarity of f. Indeed, as Figure 1 shows, considerable variation in f is possible without invalidating the model. The figure also shows that testing this hypothesis for all separating hyperplanes enables us to identify and revise only the relevant part of the clustering hierarchy when a change is detected, rather than resetting the entire model. Moreover, the low-density separation hypothesis is independent of the type (abrupt, or gradual) and the speed of drift. Lastly, testing this hypothesis corresponds to a one-dimensional change detection problem, since each low-density hyperplane is identified through an estimate of a one-dimensional marginal density.

With each hyperplane added to the model we initialise a Bernoulli CUSUM change detection regime, as described in Reynolds and Stoumbos (1999), to detect significant increase in the frequency of data arising in a small neighbourhood of the hyperplane. This frequency is determined relative to the frequency in a larger neighbourhood extending to the adjacent modes of the projected density. In this way the local density behaviour is better represented. If such a significant increase is observed, the corresponding node and the subhierarchy it anchors are removed from the model, and the node is re-initialised. We determine the larger region, which we denote by $\mathcal{R}$, by the location of the adjacent modes in the density estimate described in Section 4.2.4. The smaller neighbourhood, $\mathcal{N}$, of the hyperplane is taken to be some proportion, $\beta \in (0,1)$ of the region $\mathcal{R}$.

To implement a Bernoulli CUSUM, a pre- and post-change frequency must be specified. We obtain an initial estimate of the pre-change frequency, p_0, using the estimated density described in Section 4.2.4, and then update this estimate with new observations to obtain a more accurate estimate. With this updating regime, the threshold parameter must be recalculated with each such update. We set the post-change frequency, p_1, equal to the average density over $\mathcal{R}$, since this is the supremal possible frequency while the hyperplane is at an anti-mode of the projected density with adjacent modes beyond the boundaries of $\mathcal{R}$.

The Bernoulli CUSUM statistic, S_0, is initialised at 0. Note that the time index here, unlike previously, relates to the t-th datum since the hyperplane was introduced to the model. With the arrival of datum x_{t+1} the CUSUM statistic is updated as follows. If $x_{t+1} \notin \mathcal{R}$, then the datum is

Figure 1: Different changes in distribution and their impact on the clustering model. Red lines indicate necessity of model revision. Green lines indicate changes which can be addressed by extending the model without revision



(a) Cluster Location (b) Cluster Shape (c) New Clusters

not relevant, and $S_{t+1} = S_t$. If $x_{t+1} \in \mathcal{R}$, then,

$$\begin{aligned} B &= \begin{cases} 1, & \text{if } x_{t+1} \in \mathcal{N} \\ 0, & \text{otherwise} \end{cases}, \\ S_{t+1} &= \max\{0, S_t\} + B \\ &\quad + \log\left(\frac{1-p_1}{1-p_0}\right) \log\left(\frac{p_1(1-p_0)}{p_0(1-p_1)}\right)^{-1}. \end{aligned}$$

A change is detected when S_t exceeds a threshold α. We set α conservatively, corresponding to the maximum for chosen run lengths under p_0 and p_1, which we calculate according to the method in Reynolds and Stoumbos (1999).

5 The HSDC Algorithm

Having detailed the constituent elements of the method in Section 4, in this section we give a brief summary of the overall algorithm. The clustering model constructed by HSDC comprises a hierarchy of separating hyperplanes, each defined by a projection vector, $v \in \mathbb{R}^d$, of norm 1, and split point $b \in \mathbb{R}$. Associated with each hyperplane is a CUSUM statistic, S, in place to detect changes in the underlying distribution which lead to instability of the clustering result. The leaf nodes of the hierarchy (i.e. those for which the truncated density has not been deemed multimodal) each have associated with them an updating projection, as well as an approximate univariate distribution associated with the projection onto it. Leaf nodes might also contain inheritance vectors, z, to be passed to their children as projections in the event that the node is split.

Algorithm 1 describes an update to the hierarchical model with the arrival of a datum x. We associate with each node an identifying tag, ID, which informs the algorithm where to direct data down the hierarchy. Each internal node has 2 children, $LChild$ and $RChild$, associated with the halfspaces induced by the node's hyperplane.

The datum traverses the hierarchical structure until it reaches the appropriate leaf node. At each internal node along its path, the corresponding CUSUM statistic is updated as in Section 4.3. If a change is detected, the associated node is reinitialised, and it becomes a leaf. Otherwise, the datum is projected onto the node's projection vector, and is passed to the appropriate child node. Once the datum arrives at a leaf, the leaf's projection vector is updated as described in Section 4.1. The datum is projected onto this updated vector, and this projected datum is used to update the node's sample approximation, as described in Sections 4.2.2 and 4.2.3. The dip statistic of the sample approximation is then calculated, and if the hypothesis of unimodality is rejected, the node is split. The split point is given by the lowest anti-mode between the components of the associated level set of the estimated density, as in Section 4.2.4.

Algorithm 1: HSDC Update

```
Input: New datum $x$;
[Set index to root node]
$ID$ = root;
[Find relevant leaf node]
while $node_{ID}$ is internal do
    $S_{ID}$ = updateCUSUM($S_{ID}|x$) (Section 4.3);
    if $S_{ID} > \alpha_{ID}$ then
        removeSubhierarchy($ID$), re-initialise node $ID$;
    else
        $p := v_{ID}^{\top}(x - \bar{x}_{ID})$;
        $ID$ = ifelse($p < b_{ID}, LChild_{ID}, RChild_{ID}$);
    end
end
[Update leaf node and split if necessary]
$(v_{ID}, z_{ID})$ = updateProjection($v_{ID}, z_{ID}|x$) (Section 4.1);
$p = v_{ID}^{\top}(x - \bar{x}_{ID})$;
$\mathcal{X}_{ID}$ = updateSample($\mathcal{X}_{ID}|p$) (Sections 4.2.2-4.2.3);
if Reject $H_0 := \tilde{F}_{ID}$ is unimodal then
    $b_{ID}$ := Minimum Antimode Between $\lambda$ Level Set of $\hat{f}_{ID}$ (Section 4.2.4);
    initialise nodes $LChild_{ID}$ and $RChild_{ID}$.
end
```

5.1 Computational Complexity

We consider the worst case cost of updating the HSDC hierarchy. Suppose the model contains C clusters. The maximum depth of the hierarchy is therefore C (in general the depth of the hierarchy is much lower, with minimum value $\log_2(C)$). For each internal node along a path to a leaf node, a datum is projected onto the corresponding projection vector, with a computational cost $\mathcal{O}(d)$. This projected datum is used within an update to the corresponding CUSUM statistic with cost $\mathcal{O}(1)$. The maximal cost of finding the appropriate leaf node is thus $\mathcal{O}(C(d+1))$. Updates to a leaf node include updating its projection (and inheritance) vector, $\mathcal{O}(d)$, updating the sample approximation, $\mathcal{O}(k)$, and calculating the dip statistic, $\mathcal{O}(k)$. For an update which does not result in a new split being introduced, the computational cost is therefore $\mathcal{O}((C+1)(d+1)+k)$. In high dimensional applications, we have $Cd \gg k$, and so the behaviour is $\mathcal{O}((C+1)(d+1))$.

When a node is split, we determine the minimum bandwidth, h, giving the correct level set of the density estimate. This is done by a bisection method, which has $\log_2((h_{max} - h_{min})/\epsilon)$ iterations, where h_{max} and h_{min} are upper and lower bounds on h respectively, and ϵ is the tolerance level. Within each iteration, the kernel density estimate is calculated, at a cost of $\mathcal{O}(k^2)$. The split point is calculated within this procedure.

For existing data stream clustering algorithms based on microclusters, the primary cost associated with the online step lies in determining the nearest microcluster. This has computational cost $\mathcal{O}(md)$, where m is the number of microclusters. The time complexity of the offline step depends on the clustering algorithm being employed. Algorithms based on k-means are technically NP hard, however practical implementations run in $\mathcal{O}(mkd)$ time. Density connectivity methods based on DBSCAN have time complexity $\mathcal{O}(dm\log(m))$.

Density based methods based on grids have time complexity of the online step $\mathcal{O}(g)$, where g is the number of active grid cells. Without pruning methods, this is exponential in d (Aggarwal, 2013). For the offline component, the approach of DStream (Chen et al., 2007) only requires processing those grid cells which changed since the last offline step. If t is the number of time steps since the last offline step, the computational cost is $\mathcal{O}(t)$. This means that the total cost of all offline steps up to time point T has worst case cost $\mathcal{O}(T)$.

In general the number of microcluster and grids cells is substantially larger than the actual number of clusters, i.e., $C \ll m, g$. (The HPStream algorithm is an exception.) For standard updates therefore HSDC compares favourably with existing data stream clustering algorithms in terms of update time, especially on high dimensional examples. Updates to HSDC which result in the introduction of a new split have an additional cost of $\mathcal{O}(k^2)$. However, in practice these updates requiring a new split being introduced are infrequent, and the overall complexity is dominated by standard update steps. Most importantly however, HSDC has no offline clustering component.

6 Experimental Results

We compare the performance of the following methods.

1. CluStream (Aggarwal et al., 2003): We use the implementation in the `R` package `streamMOA`, and the parameters suggested in (Aggarwal et al., 2003).

2. HPStream (Aggarwal et al., 2004): HPStream, like CluStream, requires an offline initialisation step, for which we give it 2000 data and provide it with the correct number of clusters for the initial stream segment. We set the average dimensionality of the clusters to 80% the total dimensionality of the data, and the forgetting factor was set to 0.002. These parameters improved performance over the suggestions made in (Aggarwal et al., 2004).

3. SPDC (Tasoulis et al., 2012): We set the number of kernels for the M-kernel density estimator to 50.

4. Our method, HSDC and HSDC(I) (with projection inheritance): The null distribution for the dip test was the Gaussian, and we use the 95th centile from the Monte Carlo simulations as a threshold. We use 100 intervals for the sample approximation. The smoothing parameter, γ, for the EWMA associated with the forgetting factor was set to 0.9. Expected run lengths for CUSUM statistics were set to 10^6 and 250 for p_0 and p_1 respectively.

We also considered the density based algorithms DenStream (Cao et al., 2006) and DMM-Stream (Amini et al., 2014), however neither produced meaningful clusterings in high dimensional applications and therefore results are omitted.

To avoid ambiguity we will refer to true clusters in the data as *classes* and the assignments made by an algorithm as clusters. An ideal clustering model should (i) correctly cluster data from the same class; and (ii) assign data from each class to a single cluster. It is therefore important to consider the class distribution within each discovered cluster as well as the cluster distribution withing each class. We compare algorithms using Purity (Zhao and Karypis, 2001) and V-Measure (Rosenberg and Hirschberg, 2007). Purity takes values in $(0, 1]$, with higher values indicating a clustering in which each cluster contains observations almost exclusively from a single class. A disadvantage of this measure is that it does not penalise the splitting of data from one class between multiple clusters. V-Measure is defined as the harmonic mean of *homogeneity* and *completeness*. Homogeneity measures the conditional entropy of the class distribution within each cluster. Completeness is symmetric, and measures the conditional entropy of the cluster distribution within each class. V-Measure takes values in $[0, 1]$, with high values indicating a clustering in which each cluster contains almost uniquely data from one class, and each class is contained almost entirely within a single cluster.

To compare the performance of algorithms we evaluate them on stream segments of length 100 taken every 200 time steps. Performance plots show performance evolution through time, indicating convergence rates and the ability of algorithms to react to and recover from non-stationarity, while tables document the overall performance of the algorithms on each stream environment.

6.1 Simulations

For all simulations data were generated from a mixture of C multivariate Gaussian distributions. Covariance matrices were randomly generated according to,

$$\Sigma_i = 4\left(\frac{i}{C}\right)^2 S^\top S,\ S_{j,k} \sim N(0,1).$$

The coefficients $4(i/C)^2$ lead to classes with highly variable scales. The mixture proportions were determined by $p_i \propto u_i, u_i \sim U[1,2]$. The component means were uniformly sampled from a d-dimensional hypercube, $\mu_i \sim U[0, Cd^{1/4}]^d$. Reported results for simulated experiments are averages over 50 experiments.

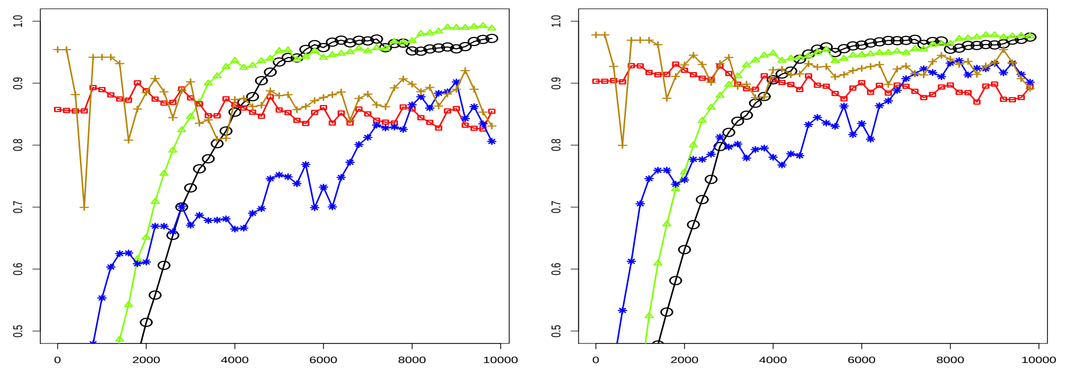
6.1.1 Static Environments

We first consider static environments of dimensionality 50, 100 and 500 with 10, 20 and 30 classes. Streams were of length $500C$ to allow algorithms without an offline component to build their models. Figure 2 shows the case with 20 classes in 500 dimensions. The offline initialisation of CluStream and HPStream ensures good performance from the early stages of the stream. However, our algorithm is quickly able to surpass them. SPDC is less conservative in introducing splits than HSDC, however its instability causes the improvements to tail off rapidly. HSDC(I) generates robust splits more rapidly than HSDC because of projection inheritance, and so it is able to achieve high performance after fewer time steps. Table 1 contains a summary of the algorithms' performance on the final stream segment of length 100. The performance in the final segment of a static stream is most indicative of model performance since the algorithms have been given opportunity to converge and maintain their models. Our algorithms achieve substantially higher performance in the high dimensional examples, while being competitive in every case considered.

6.1.2 Static Environments with Irrelevant Features

In high dimensional applications, often certain features are irrelevant to the class identity of the data. Being able to handle data with irrelevant or noisy features is therefore critical. We consider cases with 20 classes described by 100 relevant features. We explore the robustness of the algorithms to the number of irrelevant features as well as the degree of variability therein. The 100 relevant features were generated as described above. The data were then augmented with scaled standard (zero mean and identity covariance) Gaussian measurements for a variety of dimensions (d) and scaling factors (S). Figure 3 shows the case with 100 irrelevant dimensions with scaling factor 20. The performance of CluStream and HPStream is stable, but substantially diminished by the presence of the irrelevant features. HSDC and HSDC(I) both quickly surpass them and maintain stable performance after convergence. SPDC also outperforms CluStream and HPStream, but cannot achieve the same high levels of performance as our method. Table 2 contains a summary of the algorithms' performance on the final stream segment. HSDC and HSDC(I) are robust to the number of irrelevant features and the degree of noise therein, except in the most extreme case (S = 30, d = 200), where the high level of noise coupled with the large number of irrelevant features leads

Figure 2: Performance on Static Data Stream with 20 Classes in 500 Dimensions



(a) C = 20, d = 500: Purity

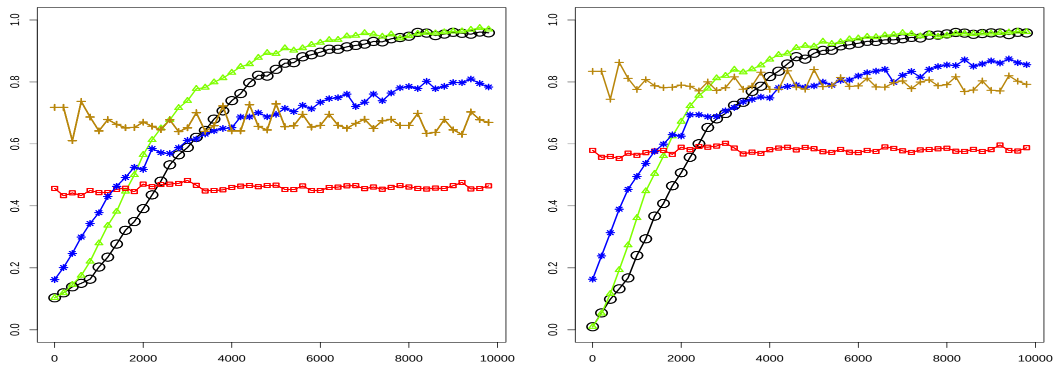
(b) C = 20, d = 500: V-Measure

HSDC (θ-θ), HSDC(I) (△-△), CluStream (+-+), HPStream (⊟-⊟), SPDC(*-*)

Table 1: Clustering Performance. Static environments. Average performance on the final stream segment. Standard deviation in parentheses. Highest performance in bold. Significantly lower performance indicated by *, based on a one sided *t*-test at the 5% level

	d = 50		d = 100		d = 500	
C = 10	Purity	V-Measure	Purity	V-Measure	Purity	V-Measure
HPStream	**0.89** (0.04)	0.85 (0.03)*	0.86 (0.05)	0.84 (0.04)*	0.81 (0.05)*	0.82 (0.04)*
SPDC	0.88 (0.14)	**0.87** (0.10)	**0.89** (0.16)	**0.88** (0.15)	0.79 (0.25)*	0.82 (0.23)*
CluStream	0.44 (0.06)*	0.49 (0.06)*	0.37 (0.05)*	0.41 (0.06)*	0.30 (0.04)*	0.30 (0.05)*
HSDC	0.84 (0.16)*	0.83 (0.15)	0.88 (0.19)	0.86 (0.20)	0.90 (0.14)	0.91 (0.12)
HSDC(I)	0.82 (0.16)*	0.81 (0.15)*	0.88 (0.17)	0.86 (0.15)	**0.94** (0.12)	**0.92** (0.08)
C = 20						
HPStream	0.87 (0.05)*	0.91 (0.03)*	0.86 (0.04)*	0.89 (0.03)*	0.85 (0.03)*	0.89 (0.03)*
SPDC	0.92 (0.15)*	0.93 (0.11)*	0.85 (0.21)*	0.90 (0.15)*	0.81 (0.16)*	0.90 (0.11)*
CluStream	0.94 (0.04)*	**0.97** (0.02)	0.94 (0.04)	**0.97** (0.02)	0.83 (0.13)*	0.89 (0.09)*
HSDC	**0.98** (0.03)	**0.97** (0.02)	**0.96** (0.13)	0.95 (0.10)	0.97 (0.08)	0.97 (0.05)
HSDC(I)	**0.98** (0.02)	0.96 (0.02)*	**0.96** (0.09)	0.95 (0.06)*	**0.99** (0.03)	**0.98** (0.02)
C = 30						
HPStream	0.90 (0.06)*	0.92 (0.04)*	0.85 (0.09)*	0.90 (0.06)*	0.84 (0.06)*	0.90 (0.04)*
SPDC	0.81 (0.20)*	0.89 (0.13)*	0.73 (0.22)*	0.84 (0.16)*	0.77 (0.20)*	0.86 (0.18)*
CluStream	0.94 (0.03)*	0.97 (0.01) *	0.95 (0.02)	**0.98** (0.01)	0.94 (0.04)*	0.97 (0.04)
HSDC	**0.98** (0.06)	**0.98** (0.03)	**0.97** (0.12)	0.97 (0.07)	0.96 (0.14)	0.97 (0.11)
HSDC(I)	0.95 (0.13)	0.96 (0.08)	**0.97** (0.09)	0.97 (0.05)	**0.98** (0.08)	**0.98** (0.05)

Figure 3: Clustering Performance. Static Environment with Irrelevant Features. 20 Classes in 100 Relevant and 100 Irrelevant Dimensions with Moderate Variability



(a) S = 20, d = 100: Purity (b) S = 20, d = 100: V-Measure

HSDC (θ-θ), HSDC(I) (△-△), CluStream (+-+), HPStream (⊟-⊟), SPDC(∗-∗)

to slower splitting and a much higher incidence of false detection of change. CluStream achieves the highest performance in this most extreme case. HPStream seems unable to distinguish classes when the number of irrelevant features dominates the number of relevant ones (d=200).

6.1.3 Non-Stationary Environments

For these experiments we simulated environments in which the distribution undergoes abrupt changes at discrete points in time. Plots for the 500 dimensional cases are found in Figures 4-6, and a full summary of the results from non-stationary examples is found in Table 3. For these experiments we consider the performance of algorithms throughout the data streams. The table therefore reports the average and standard deviation of the average performance of each algorithm on stream segments of length 100 taken every 200 time steps.

Variable Number of Classes For results which lend themselves better to interpretation, we simulate two separate cases; streams with an increasing number of classes and streams with a decreasing number. For the former we split a randomly selected class every 1000 time steps, beginning with 20 classes and ending with 60. The reverse procedure was adopted for the case of decreasing number of classes. CluStream requires a fixed number of classes throughout. This was set to 40, the average number of classes over the stream. HPStream was initialised with the correct number for the initial stage of the stream.

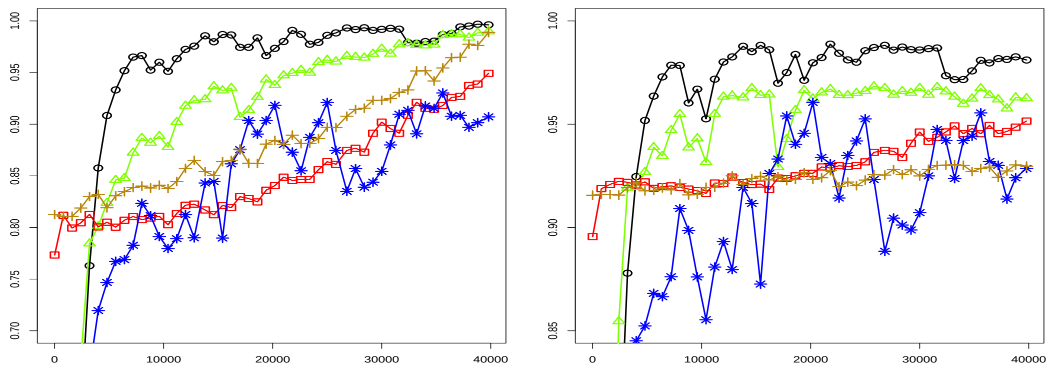
For a decreasing number of classes (Figure 4) the performance of all algorithms improves as the stream progresses, since the environment becomes easier to model. Following initial convergence, the performance of our method surpasses the others. In the case of increasing number of classes (Figure 5) the performance of CluStream and HPStream deteriorates as the stream progresses. This highlights the limitation of having to specify a fixed number of clusters for the entire data stream. In contrast the performance of our method is stable after initial convergence. The sustained high performance indicates that HSDC is able to identify when clusters are being split.

Distribution Overhaul In this set of experiments the distribution undergoes complete change at regular intervals. We consider the case with 20 classes, whose parameters are reinitialised every 15000 time steps. The performance plots show a sudden deterioration in the performance of our method following each change. This is expected as the separating hyperplanes are likely redundant

Table 2: Clustering Performance. Static environments with irrelevant features. Average performance on the final stream segment. Standard deviation in parentheses. Highest performance in bold. Significantly lower performance indicated by *, based on a one sided t-test at the 5% level

	d = 50		d = 100		d = 200	
S = 10	Purity	V-Measure	Purity	V-Measure	Purity	V-Measure
HPStream	0.42 (0.12)*	0.56 (0.10)*	0.46 (0.08)*	0.59 (0.08)*	0.10 (0.02)*	0.01 (0.04)*
SPDC	0.85 (0.24)*	0.90 (0.17)*	0.80 (0.22)*	0.86 (0.16)*	0.83 (0.20)*	0.89 (0.15)*
CluStream	0.94 (0.03)*	**0.97** (0.02)	**0.94** (0.04)	**0.97** (0.02)	0.91 (0.08)*	0.95 (0.05)
HSDC	**0.99** (0.02)	**0.97** (0.02)	0.92 (0.21)	0.92 (0.19)	0.96 (0.11)	**0.96** (0.07)
HSDC(I)	0.97 (0.07)*	0.96 (0.03)*	0.92 (0.20)	0.91 (0.20)	**0.97** (0.07)	**0.96** (0.05)
S = 20						
HPStream	0.44 (0.10)*	0.58 (0.09)*	0.46 (0.08)*	0.59 (0.08)*	0.11 (0.02)*	0.01 (0.04)*
SPDC	0.83 (0.22)*	0.89 (0.17)*	0.78 (0.22)*	0.86 (0.17)*	0.71 (0.24)*	0.80 (0.19)*
CluStream	0.92 (0.05)*	**0.96** (0.03)	0.67 (0.07)*	0.79 (0.05)*	0.65 (0.08)*	0.78 (0.06)*
HSDC	**0.97** (0.09)	**0.96** (0.06)	0.96 (0.09)	**0.96** (0.05)	0.87 (0.21)*	0.89 (0.19)*
HSDC(I)	0.96 (0.12)	0.95 (0.08)	**0.97** (0.06)	**0.96** (0.04)	**0.94** (0.10)	**0.95** (0.06)
S = 30						
HPStream	0.42 (0.12)*	0.56 (0.09)*	0.40 (0.10)*	0.51 (0.11)*	0.11 (0.02)*	0.02 (0.05)*
SPDC	0.73 (0.23)*	0.81 (0.19)*	0.64 (0.21)*	0.76 (0.18)*	0.49 (0.20)*	0.62 (0.22)*
CluStream	0.72 (0.07)*	0.83 (0.05)*	0.72 (0.07)*	0.75 (0.08)*	**0.71** (0.07)	**0.79** (0.05)
HSDC	**0.88** (0.20)	**0.90** (0.19)	0.70 (0.28)*	0.77 (0.26)*	0.14 (0.10)*	0.07 (0.18)*
HSDC(I)	**0.88** (0.20)	0.89 (0.18)	**0.85** (0.20)	**0.88** (0.17)	0.56 (0.27)*	0.65 (0.28)*

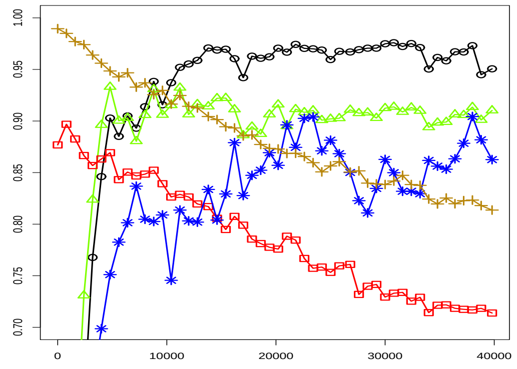
Figure 4: Clustering Performance. Decreasing Classes. 500 Dimensions



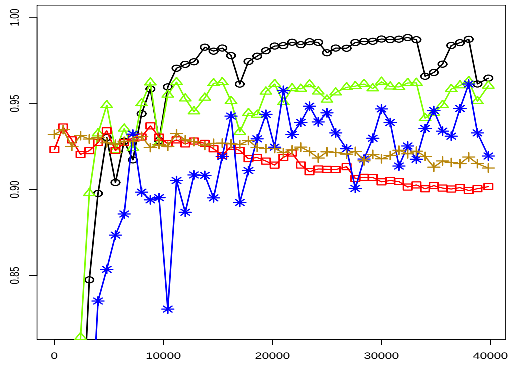
(a) Decreasing Classes: Purity (b) Decreasing Classes: V-Measure

HSDC (θ–θ), HSDC(I) (△–△), CluStream (+–+), HPStream (⊟–⊟), SPDC(✳–✳)

Figure 5: Clustering Performance. Increasing Classes. 500 Dimensions



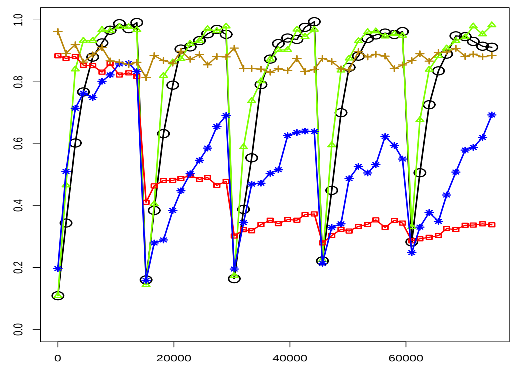
(a) Increasing Classes: Purity



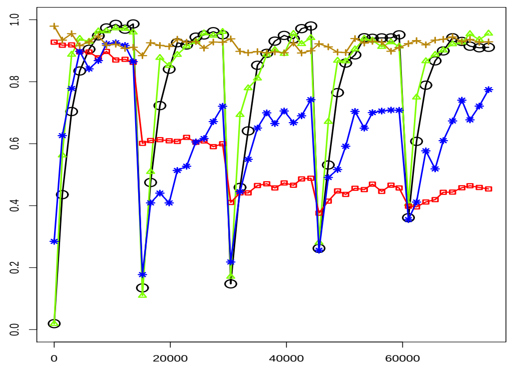
(b) Increasing Classes: V-Measure

HSDC (θ-θ), HSDC(I) (△-△), CluStream (+-+), HPStream (⊟-⊟), SPDC(∗-∗)

Figure 6: Clustering Performance. Distribution Overhaul. 20 Classes in 500 Dimensions



(a) Distribution Overhaul: Purity



(b) Distribution Overhaul: V-Measure

HSDC (θ-θ), HSDC(I) (△-△), CluStream (+-+), HPStream (⊟-⊟), SPDC(∗-∗)

or intersect the new classes, and thus the model must be rebuilt from scratch. In all cases, however, HSDC is able to rebuild good quality clustering models, with no apparent deterioration after multiple changes. The instability of a direct implementation of projected divisive clustering is indicated by the performance of SPDC. CluStream is least affected by the changes, but cannot achieve the high performance of HSDC. Because of the multiple rebuilding stages, CluStream is able to achieve higher average performance than our method when taken over the entire stream.

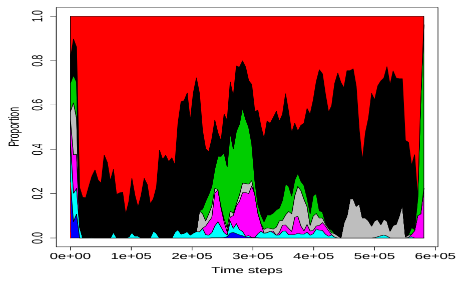
6.2 Publicly Available Data Sets

In this section we compare the performance of the algorithms on two publicly available data sets where the true class label is known. The first, Forest Cover Type, is lower dimensional and so (at best) we hope to achieve results comparable with state of the art standard data stream clustering algorithms such as CluStream. The second, Gas Sensor Array, is higher dimensional and we expect HSDC to perform better.

Table 3: Clustering Performance. Drifting environments. Average performance on segments taken every 200 time steps. Standard deviation in parentheses. Highest performance in bold. Significantly lower performance indicated by *, based on a one sided t-test at the 5% level

	Increasing Classes		Decreasing Classes		Overhaul	
d = 100	Purity	V-Measure	Purity	V-Measure	Purity	V-Measure
HPStream	0.85 (0.03)*	**0.93** (0.01)	**0.93** (0.02)	**0.94** (0.01)	0.91 (0.02)*	0.93 (0.00)*
SPDC	0.83 (0.08)*	0.89 (0.06)*	0.83 (0.07)*	0.89 (0.04)*	0.51 (0.06)*	0.56 (0.07)*
CluStream	0.90 (0.01)	0.92 (0.00)*	0.90 (0.01)*	0.92 (0.01)*	**0.94** (0.00)	**0.97** (0.00)
HSDC	**0.91** (0.04)	**0.93** (0.03)	0.90 (0.04)*	0.92 (0.03)*	0.82 (0.04)*	0.84 (0.04)*
HSDC(I)	**0.91** (0.04)	**0.93** (0.02)	0.89 (0.04)*	0.92 (0.03)*	0.82 (0.04)*	0.83 (0.03)*
d = 500						
HPStream	0.79 (0.03)*	0.92 (0.02)*	0.85 (0.03)*	0.93 (0.01)*	0.46 (0.07)*	0.57 (0.07)*
SPDC	0.81 (0.08)*	0.89 (0.05)*	0.83 (0.07)*	0.89 (0.05)*	0.53 (0.06)*	0.63 (0.05)*
CluStream	0.88 (0.01)*	0.91 (0.01)*	0.88 (0.01)*	0.91 (0.01)*	**0.86** (0.03)	**0.91** (0.03)
HSDC	**0.91** (0.03)	**0.93** (0.03)	**0.93** (0.03)	**0.94** (0.03)	0.78 (0.04)*	0.80 (0.03)*
HSDC(I)	0.87 (0.03)*	0.92 (0.02)	0.89 (0.03)*	0.93 (0.02)*	0.82 (0.03)*	0.83 (0.03)*

Figure 7: Class Proportions of Forest Cover Type



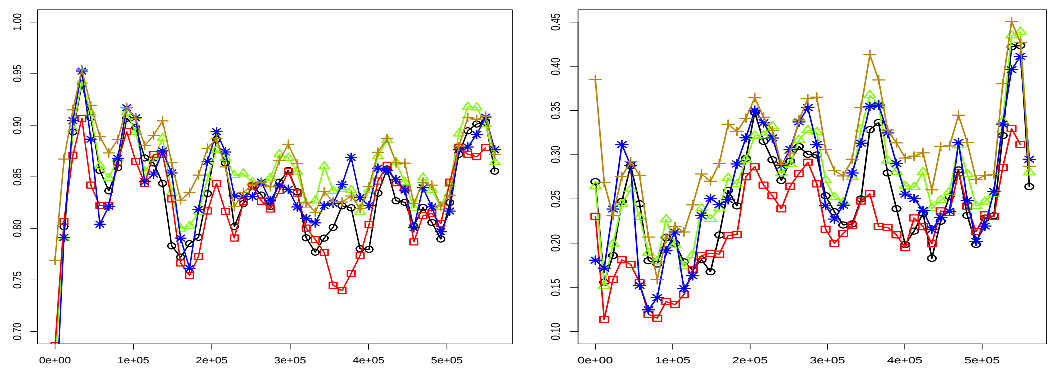
6.2.1 Forest Cover Type

The Forest Cover Type data set, taken from the UCI Machine Learning Repository (Bache and Lichman, 2014), contains 581012 observations characterised by 54 features, where each observation corresponds to one of seven forest cover types. As in the analyses in (Aggarwal et al., 2004; Tasoulis et al., 2012) we use only the ten continuous features. A plot of the class proportions (Figure 7) suggests considerable variability in the data distribution through the stream. Figure 8 shows the performance of the various algorithms through the stream (the series of performance values were smoothed for better interpretability). CluStream achieves the highest performance through the majority of the stream. The average purity of all algorithms is similar; in decreasing order: CluStream = 0.86, HSDC(I) = 0.85, SPDC = 0.84, HSDC = 0.83, HPStream = 0.82. The average V-Measure of CluStream is substantially higher than the other algorithms at 0.31. The other algorithms achieved average V-Measure of: HSDC(I) = 0.27, SPDC = 0.27, HSDC = 0.25, HPStream = 0.22.

6.2.2 Gas Sensor Array

The Gas Sensor Array Drift data set, available from the UCI Machine Learning repository (Bache and Lichman, 2014), contains 13910 measurements from each of 16 chemical sensors (amounting to 128 features in total per datum) used in simulations of drift compensation for the purpose of discriminating between six different gases (Ammonia, Acetaldehyde, Acetone, Ethylene, Ethanol, and Toluene) at various levels of concentration. The experiment was designed for the task of

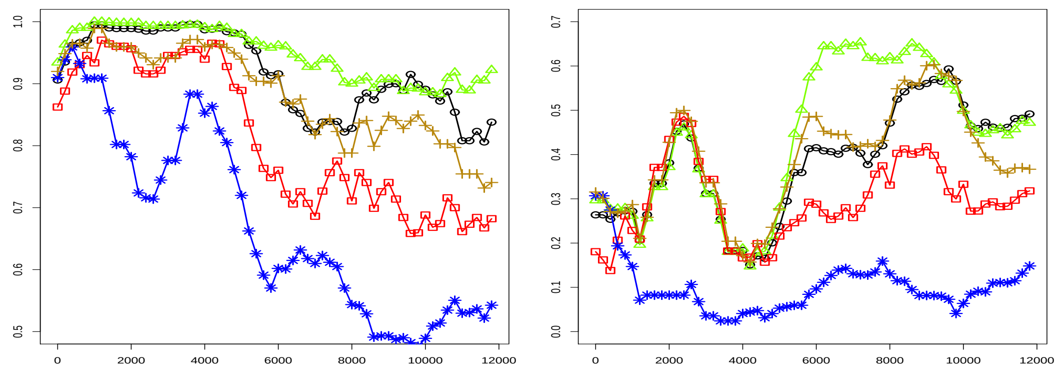
Figure 8: Clustering Performance. Forest Cover Type Data



(a) Forest Cover Type: Purity (b) Forest Cover Type: V-Measure

HSDC (θ-θ), HSDC(I) (△-△), CluStream (+-+), HPStream (⊟-⊟), SPDC(∗-∗)

Figure 9: Clustering Performance. Gas Sensor Array Data



(a) Gas Sensor Array: Purity (b) Gas Sensor Array: V-Measure

HSDC (θ-θ), HSDC(I) (△-△), CluStream (+-+), HPStream (⊟-⊟), SPDC(∗-∗)

achieving al degradation in discriminatory performance over time as possible to evaluate strategies able to handle non-stationarity or drift (Vergara et al., 2012). The average number of dimensions for HPStream was set to 80 after experiments indicated this resulted in better performance. Figure 9 shows the performance of the algorithms through the stream (again the series are smoothed for better interpretability). Our method is the only one to obtain good discriminatory performance in the latter part of the stream, indicated by the purity performance. The instability of SPDC is again highlighted by its severe performance degradation with drift. The average purity (and V-Measure), in decreasing order: HSDC(I) = 0.94 (0.44), HSDC = 0.90 (0.39), CluStream = 0.87 (0.39), HPStream = 0.80 (0.30), SPDC = 0.67 (0.12).

6.3 Discussion of Experimental Results

We compared our method with three existing data stream clustering algorithms from the literature, CluStream, HPStream, and SPDC. We investigated the scalability of the method in terms of dimensionality and number of clusters. HSDC and HSDC(I) outperformed the compared methods in most cases, and especially in the highest dimensional examples. Next we investigated robustness to irrelevant/noisy features. The performance of our method was affected to a lesser degree than the other methods except in the most extreme case. When the number of irrelevant dimensions dominated the number of relevant ones, and in addition the degree of variability in the irrelevant dimensions was large, HSDC failed to detect clusters reliably. The stability added by projection inheritance improved matters, but the performance was still strongly affected. Following that we introduced non-stationarity, considering three types: Clusters dividing, clusters merging, and distribution overhaul. Our method obtained the highest overall performance when the number of clusters varied (clusters dividing/merging). For distribution overhaul, because our method is required to rebuild its model from scratch, the average performance was strongly affected. Despite this fact, HSDC and HSDC(I) were always able to rebuild high quality clustering models, with no apparent degradation with repeated overhauls.

Finally we considered two real world applications: Clustering Forest Cover Types, and Gas Sensor Array Data. The former is lower dimensional (10 dimensions used), and our method obtained similar performance to the other methods considered. The latter is higher dimensional, and our method strongly outperformed the compared algorithms.

It is important to notice that in almost all cases where our approach does not yield the best performance, it is still close to the best performing method, while there are numerous examples in which our method far outperforms all others.

7 Conclusions

We introduced a framework for projected divisive clustering that is consistent with high density clustering, and which accommodates central challenges associated with data streams, including high dimensional data and non-stationarity. The derived method is able to identify clusters in arbitrary subspaces, estimate the number of clusters automatically and identify changes in the data distribution which affect the validity of the model. The algorithm is also fully incremental, requiring no offline component. To our knowledge, no other algorithms achieve all of these simultaneously.

The method constructs a hierarchy of low-density hyperplane separators, thereby enabling it to handle high dimensional data. This is achieved by establishing directions of high variability and approximating the projected distribution along them. We propose a simple approach to speed up the incremental approximation of such directions, hence reducing the time required to construct the cluster hierarchy. We introduce new components to the model only when the marginal distribution along the projection is found to be multimodal. For this purpose, a fixed memory approximation to the dip test is developed, and shown to lower bound the true value.

The framework incorporates a novel formulation to detect arbitrary changes in the population distribution that affect the validity of the current clustering model. The approach relies on detecting increases in the density local to a separating hyperplane, which signal that the hyperplane intersects regions of high density. This enables us to not only provide indication of the timing of such changes but also to isolate the parts of the hierarchical model that require revision. To our knowledge this is the first general purpose clustering algorithm able to detect changes in the underlying distribution.

Experimental results show that algorithms derived from the proposed framework obtain the best overall performance on a range of problem types, when compared with state of the art algorithms for data stream clustering. Among the competing methods, CluStream achieved an on-average better performance when the population distribution underwent the most extreme type of abrupt change. The reason is that reconstructing the entire clustering hierarchy invariably requires more time than the adaptation of centroid-based methods. This result however is conditional on the a priori specification of the correct number of clusters for such algorithms. When this is not the case our methods fare better.

Appendix A. Proofs

Before we can prove Lemma 2, we require the following preliminaries.

The algorithm for computing the dip of a distribution function F constructs a unimodal distribution function G with the following properties: (i) The modal interval of G, $[m, M]$, is equal to the modal interval of the closest unimodal distribution function to F, which we denote by F^U, based on the supremum norm; (ii) $\|F - G\|_\infty = 2\|F - F^U\|_\infty$; (iii) G is the greatest convex minorant of F on $(-\infty, m]$; (iv) G is the least concave majorant of F on $[M, \infty)$. By construction, the function G is linear between its *nodes*. A node $n \leq m$ of G satisfies $G(n) = \liminf_{x\to n} F(x)$, while a node $n \geq M$ of G satisfies $G(n) = \text{limsup}_{x\to n} F(x)$. If F is the distribution function of a discrete random variable, then G is continuous.

The function F^U can be constructed by finding appropriate values $b < m$, $B > M$ s.t. F^U is equal to $G + \text{Dip}(F)$ on $[b, m]$, equal to $G - \text{Dip}(F)$ on $[M, B]$, linearly interpolating between $G(m)$ and $G(M)$ and given any appropriate tails, which we choose to be linearly decreasing/increasing to 0 and 1 respectively.

Before proving Lemma 2, we require the following preliminary result, which relies on the notion of a *step linear* function.

Step Linear A function f is *step linear* on a non-empty, compact interval $I = [a, b]$, if

$$f(x) = \alpha + \beta \left\lfloor (x - a)\frac{n}{b - a} \right\rfloor, \ \forall x \in I,$$

for some $\alpha, \beta \in \mathbb{R}$ and $n \in \mathbb{N}$.

A step linear function is piecewise constant, and has n equally sized jumps of size β spaced equally on I with the final jump ocurring at b. The approximate empirical distribution function $\tilde{F}$ (Section 4.2.2) is therefore step linear over the approximating intervals.

Proposition 4 *Let f be step linear on an interval $I = [a, b]$, and satisfy $\lim_{x\to a^-} f(x) = \alpha - \beta$, where α, β as in the above definition for f. Let g be liner on I and continuous on a neighbourhood of I. Then*

$$\begin{aligned}\sup_{x\in I} |f(x) - g(x)| \quad \leq \quad & \max\{\mathit{limsup}_{x\to a}|f(x) - g(x)|, \\ & \mathit{limsup}_{x\to b}|f(x) - g(x)|\}.\end{aligned}$$

Proof *Let f_m and f^M be linear on a neighbourhood of I s.t. they form the closest lower and upper bounding functions of f on I respectively. Since f is step linear, we have,*

$$\lim_{x\to a^-} f(x) = f_m(a), \quad \lim_{x\to b^-} f(x) = f_m(b),$$
$$f(a) = f^M(a), \qquad f(b) = f^M(b).$$

We therefore have, by above and the fact that g, f_m, and f^M are linear on I,

$$\begin{aligned}\sup_{x\in I} |f(x) - g(x)| \quad \leq \quad & \max\left\{\sup_{x\in I} |f^M(x) - g(x)|,\right. \\ & \left.\sup_{x\in I} |f_m(x) - g(x)|\right\} \\ = \quad & \max\{|f^M(b) - g(b)|, \\ & |f^M(a) - g(a)|, |f_m(a) - g(a)|, \\ & |f_m(b) - g(b)|\} \\ = \quad & \max\{\mathit{limsup}_{x\to a}|f(x) - g(x)|, \\ & \mathit{limsup}_{x\to b}|f(x) - g(x)|\}.\end{aligned}$$

□

We are now in a position to prove Lemma 2, which states that the dip of a compactly approximated sample, as described in Section 4.2.2, provides a lower bound on the dip of the true sample.

Proof of Lemma 2. Let $I = [a, b]$ be any compact interval and F_I the empirical distribution function of $(\mathcal{X} \cap I^c) \cup \text{Unif}(\mathcal{X}, I)$. Assume $|\mathcal{X} \cap I| > 1$, since otherwise $F_I = F_{\mathcal{X}}$ and we are done. We can assume that the endpoints of I are elements of $\mathcal{X}$ since this defines the same uniform set. $F_{\mathcal{X}}$ and F_I are therefore equal on $\text{Int}(I)^c$. In fact, since $\mathcal{X}$ consists of unique points, $\exists \epsilon > 0$ s.t. $F_I(x) = F_{\mathcal{X}}(x)$ $\forall x \notin (a + \epsilon, b - \epsilon)$. Define F'_I to be equal to $F^U_{\mathcal{X}}$ for $x \notin \text{Int}(I)$ and linearly interpolating between $F^U_X(a)$ and $F^U_X(b)$. By construction F'_I is a continuous unimodal distribution function.

We now show $\|F_I - F'_I\|_\infty \leq \|F_{\mathcal{X}} - F^U_{\mathcal{X}}\|_\infty$. To see this, suppose that it is not true, i.e., $\exists x$ s.t. $|F_I(x) - F'_I(x)| > \sup_y |F_{\mathcal{X}}(y) - F^U_{\mathcal{X}}(y)|$. Clearly $x \in \text{Int}(I)$ due to the equalities discussed above and the construction of F'_I. Because of the continuity of $F^U_{\mathcal{X}}$ and F'_I and the equality of $F_{\mathcal{X}}$ and F_I on $(a, a + \epsilon) \cup (b - \epsilon, b)$, we have

$$\text{limsup}_{y \to a}|F_I(y) - F'_I(y)| = \text{limsup}_{y \to a}|F_{\mathcal{X}} - F^U_{\mathcal{X}}(x)|$$

and

$$\text{limsup}_{y \to b}|F_I(y) - F'_I(y)| = \text{limsup}_{y \to b}|F_{\mathcal{X}} - F^U_{\mathcal{X}}(x)|.$$

But by Proposition 4 one of these left hand sides is at least as large as $|F_I(x) - F'_I(x)|$, leading to a contradiction.

We have shown that the addition of a single interval cannot increase the dip. We can apply the same logic to the now modified sample $(\mathcal{X} \cap I^c) \cup \text{Unif}(\mathcal{X}, I)$, iterating the addition of disjoint intervals to obtain a non-increasing sequence of dips. □

In the above proof, we do not show that F'_I is the closest unimodal distribution function to F_I, however its existence necessitates the closest one being at least as close. Now, the sample approximations we employ still contain a full t atoms after t observations, however, they can be stored in $\mathcal{O}(k)$ for k intervals. We can easily show that the dip of such a sample approximation can be computed in $\mathcal{O}(k)$ time.

Proposition 5 *The dip of a sample consisting of k uniform sets with disjoint ranges can be computed in $\mathcal{O}(k)$ time.*

Proof We begin by showing that there exists a unimodal distribution function which is linear on the ranges of the uniform sets and which achieves the minimal distance to the empirical distribution function of the sample.

Let F be a continuous unimodal distribution function s.t. $\|F - \tilde{F}\|_\infty = \text{Dip}(\tilde{F})$. Define F' similarly to in the above proof to be the continuous distribution function which is equal to F outside and at the boundaries of the intervals defining the uniform sets and linearly interpolating on them. Using the same logic, we know that $\sup_x |F'(x) - \tilde{F}(x)| \leq \sup_x |F(x) - \tilde{F}(x)|$, hence $\|F' - \tilde{F}\|_\infty = \text{Dip}(\tilde{F})$.

Proposition 4 ensures that points in the interior of the intervals will not be chosen by the dip algorithm as end points of the modal interval of G, nor points at which the difference between the functions is supremal. The possible choices for these locations is therefore $\mathcal{O}(k)$, and the algorithm need not evaluate the functions except at the endpoints of the intervals. □

Finally, we provide a proof of Proposition 3.

Proof of Proposition 3. For $s > 1$ we have $\|v_s - v_{s-1}\| = \|v_s\|\|v_s - v_{s-1}\| \geq |v_s \cdot (v_s - v_{s-1})| = |v_s v_{s-1} - 1|$, since $\|v_t\| = 1 \; \forall t$. Therefore, since $\{v_t\}_{t=1}^{\infty}$ is almost surely convergent, and therefore almost surely Cauchy, we have $v_s \cdot v_{s-1} \xrightarrow{a.s.} 1 \Rightarrow \arccos(v_s \cdot v_{s-1}) \xrightarrow{a.s.} 0$. Now, we can easily show that,

$$\lambda_t \leq \gamma^{t-1}\lambda_1 + (1-\gamma)\sum_{i=1}^{t-2}\gamma^i \arccos(v_{t-i} \cdot v_{t-i-1}).$$

Take $\epsilon > 0$ and t large enough that $\gamma^{t-1}\lambda_1 < \gamma\epsilon$, and $t > k+2$, where $k = \lfloor \log(\epsilon(1-\gamma)/2\pi)/\log(\gamma) - 1 \rfloor$. Consider,

$$\begin{aligned}\sum_{i=1}^{t-2}\gamma^i \arccos(v_{t-i} \cdot v_{t-i-1}) \quad &\leq \quad \sum_{i=1}^{k}\arccos(v_{t-i} \cdot v_{t-i-1}) \\ &\quad\quad + \frac{\pi\gamma^{k+1}}{1-\gamma},\end{aligned}$$

and $\frac{\pi\gamma^{k+1}}{1-\gamma} \leq \frac{\epsilon}{2}$. In all,

$$\lambda_t > \epsilon \Rightarrow \sum_{i=0}^{k}\arccos(v_{t-i} \cdot v_{t-i-1}) > \epsilon/2.$$

Notice that k does not depend on t. With probability 1, for any given $\epsilon > 0$ there is a $\mathcal{T}$ s.t. $T > \mathcal{T}$ implies $\sum_{i=0}^{k}\arccos(v_{T-i} \cdot v_{T-i-1}) \leq \epsilon/2$, implying $\lambda_T \leq \epsilon$ for all $T > \mathcal{T}$, and the result follows.$\square$

Acknowledgements

David Hofmeyr gratefully acknowledges funding from both the Engineering and Physical Sciences Research Council (EPSRC) and the Oppenheimer Memorial Trust.

References

Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and Practice. IEEE Transactions on Knowledge and Data Engineering. 15(3), 515-528 (2003)

Silva, J. A., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C. P. L. F., Gama, J.: Data Stream Clustering: A Survey. ACM Computing Surveys. 46(1), 13:1-13:31 (2013)

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. 1-16 (2002)

Jain, A. K.: Data clustering: 50 years beyond K-means. Pattern Recognition Letters. 31(8), 651-666 (2010)

Aggarwal, C. C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. Proceedings of the 29th international conference on Very large data bases. 29, 81-92 (2003)

Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. Proceedings of the 2006 SIAM International Conference on Data Mining. 328-339 (2006)

Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. ACM SIGMOD Conference. 25, 103-114 (1996)

Aggarwal, C. C., Han, J., Wang, J., Yu, P. S.: A framework for projected clustering of high dimensional data streams. Proceedings of the Thirtieth international conference on Very large data bases. 852-863 (2004)

Ntoutsi, I., Zimek, A., Palpanas, T., Kröger, P., Kriegel, H.P.: Density-based projected clustering over high dimensional data streams. Proceedings SiAM International Conference on Data Mining, 987998 (2012)

Hassani M., Spaus P., Gaber M. M., Seidl T.: Density-based projected clustering of data streams. Proceedings of the 6th International Conference on Scalable Uncertainty Management, 311-324 (2012)

Hassani, M., Kranen, P., Saini, R., Seidl, T.: Subspace anytime stream clustering. Proceedings of the 26th International Conference on Scientific and Statistical Database Management, 37 (2014)

Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-Adaptive Anytime Stream Clustering. IEEE International Conference on Data Mining. 249-258 (2009). doi: 10.1109/ICDM.2009.47

Kriegel, H. P., Kröger, P., Zimek, A.: Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Transactions on Knowledge Discovery from Data. 3(1), 1-58 (2009)

Hartigan, J. A.: Clustering Algorithms. Wiley Series in Probability and Mathematical Statistics. Wiley, New York (1975)

Ankerst, M., Breunig, M., Kriegel, H.P., Sander, J.: OPTICS: ordering points to identify the clustering structure. Proceedings of the ACM Sigmod Conference. 49-60 (1999)

Stuetzle, W.: Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. Journal of Classification, 20(5): 25-47 (2003)

Campello, R. J. G. B., Moulavi, D., Zimek, A., Sander, J.: A framework for semi-supervised and unsupervised optimal extraction of clusters from hierarchies. Data Mining and Knowledge Discovery 27(3): 344-371 (2013)

Cuevas, A., Fraiman, R.: A plug-in approach to support estimation. The Annals of Statistics. 2300-2312 (1997)

Rigollet, P., Vert, R.: Optimal rates for plug-in estimators of density level sets. Bernoulli. 15(4), 1154-1178 (2009)

Menardi, G., Azzalini, A.: An advancement in clustering via nonparametric density estimation. Statistics and Computing. 24(5) 753-767 (2014).
doi: 10.1007/s11222-013-9400-x

Azzalini, A., Torelli, N.: Clustering via nonparametric density estimation. Statistics and Computing. 17(1) 71-80 (2007). doi: 10.1007/s11222-006-9010-y

Cuevas, A., Febrero, M., Fraiman, R.: Cluster analysis: a further approach based on density estimation. Computational Statistics & Data Analysis. 36(4), 441-459 (2001)

Stuetzle, W., Nugent, R.: A generalized single linkage method for estimating the cluster tree of a density. Journal of Computational and Graphical Statistics. 19(2), 397-418 (2010)

Rinaldo, A., Wasserman, L.: Generalized density clustering. The Annals of Statistics. 2678-2722 (2010)

Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, 226231 (1996)

Tasoulis, S. K., Tasoulis, D. K., Plagianakos, V.: Enhancing principal direction divisive clustering. Pattern Recognition. 43(10) 3391-3411 (2010)

Amini, A., Saboohi, H., Wah, T.Y., Herawan, T.: Dmm-stream: A density mini-micro clustering algorithm for evolving datastreams. In Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013), 675-682 (2014)

Chen Y., Tu L.: Density-based clustering for real-time stream data. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 133-142 (2007)

Jia, C., Tan, C., Yong, A.: A Grid and Density-based Clustering Algorithm for Processing Data Stream. International Conference on Genetic and Evolutionary Computing (2008)

Kranen, P.: Anytime algorithms for stream data mining. Diese Dissertation, RWTH Aachen University (2011)

Aggarwal, C. C.: A survey of stream clustering algorithms. Data Clustering: Algorithms and Applications, Aggarwal C. C., Reddy C. (eds.), 457-482 (2013)

Amini, A., Wah, T.Y., Saboohi, H.: On density based data streams clustering algorithms: A survey. Journal of Computer Science and Technology, 29(1) 116-141 (2014)

Scott, D. W.: Multivariate density estimation: theory, practice, and visualization. 383. John Wiley & Sons (2009)

von Luxburg, U.: Clustering Stability. Now Publishers Inc (2010)

Boley, D.: Principal direction divisive partitioning. Data Mining and Knowledge Discovery. 2(4), 325-344 (1998)

Artac, M., Jogan, M., Leonardis, A.: Incremental PCA for on-line visual learning and recognition. Proceedings of the 16*th* International Conference on Pattern Recognition. 3, 781-784 (2002)

Li, Y., Xu, L-Q., Morphett, J., Jacobs, R.: An integrated algorithm of incremental and robust pca. Proceedings of the International Conference on Image Processing. 1, 245-248 (2009)

Weng, J., Zhang, Y., Hwang, W-S.: Candid covariance-free incremental principal component analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence. 25(8), 1034-1040 (2003)

Tasoulis, S. K., Tasoulis, D. K., Plagianakos, V. P.: Clustering of high dimensional data streams. Artificial Intelligence: Theories and Applications. 223-230 (2012)

Müller, D. W., Sawitzki, G.: Excess mass estimates and tests for multimodality. Journal of the American Statistical Association. 86(415), 738-746 (1991)

Hartigan, J. A., Hartigan, P. M.: The dip test of unimodality. The Annals of Statistics. 70-84, (1985)

Hartigan, P. M.: Algorithm as 217: Computation of the dip statistic to test for unimodality. Journal of the Royal Statistical Society. Series C (Applied Statistics). 34(3), 320-325 (1985)

Haykin, S.: Neural Networks: A comprehensive foundation. Prentice-Hall International (1999)

Anagnostopoulos, C., Tasoulis, D. K., Adams, N. M., Pavlidis, N. G., Hand, D. J.: Online linear and quadratic discriminant analysis with adaptive forgetting for streaming classification. Statistical Analysis and Data Mining. 5(2), 139-166 (2012)

Pavlidis, N. G.,Tasoulis, D. K., Adams, N. M., Hand, D. J.: λ-Perceptron: An adaptive classifier for data-streams. Pattern Recognition. 44(1), 78-96 (2011)

Reynolds Jr, M. R., Stoumbos, Z. G.: A CUSUM chart for monitoring a proportion when inspecting continuously. Journal of Quality Technology. 31(1) (1999)

Zhao, Y., Karypis, G.: Criterion functions for document clustering: Experiments and analysis. Machine Learning. (2001)

Rosenberg, A., Hirschberg, J.: V-measure: A conditional entropy-based external cluster evaluation measure. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. 410-420 (2007)

Bache, K., Lichman, M.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences. http://archive.ics.uci.edu/m.

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., Huerta, R.: Chemical gas sensor drift compensation using classifier ensembles. Sensors and Actuators B: Chemical. 166, 320-329 (2012)