Contents

Chapter 1

Spectral Clustering for Data Streams

1.1 Introduction

Spectral clustering has been empirically shown to perform well under difficult cluster scenarios, but is computationally intensive to run for large data sets. KASP offers an approach to deal with this burden, by summarising the data using k-means, and feeding the centroids as input to spectral clustering. We show how spectral clustering can be applied to online data by combining the Clustream framework with the concepts developed in KASP Yan et al. (2009). Clustream is a framework for dealing with a constant stream of data, and splits the clustering process into two stages: micro-clustering and macro-clustering. We analyse the performance of spectral clustering in an online setting on simulated and real texture-based data sets, both for static and evolving streams. We discuss how to quantify the information loss from summarising the data stream and investigate empirically and algebraically how this perturbation may effect the overall performance. RD: Rewrite, excluding any references to KASP, focus more on unweighted / weighted.

We discuss the literature in Section 1.2. Spectral clustering is introduced in Section 1.3. Online clustering and the Clustream algorithm are discussed in Section 1.4 and experimental results shown in Section 1.5.

1.2 Literature Review

This review introduces the historical origins of Spectral Clustering, discusses methods to speed up the computation time of offline Spectral Clustering algorithms and comments on existing incremental spectral clustering methods.

Spectral Clustering is a technique for separating data into groups or clusters. The elements in each group should be similar to each other, and elements in different groups should be dissimilar. In Spectral Clustering a weighted distance matrix of the data points called the affinity matrix is formed and by evaluating the spectral properties of this affinity matrix clusters can be estimated. The idea of splitting a data set into groups or clusters with high within-cluster similarity and between-cluster dissimilarity, can also be thought of as a graph partitioning problem with the data points as graph vertexes and the elements in the affinity matrix representing weights on the graph edges. Imagine that all of the data points in a data set are joined together by string. If you were asked break this graph into two distinct pieces, which strings would you cut?

Graph partitioning problems have long been of interest to mathematicians, ever since Euler solved the popular "Seven Bridges of Königsberg" problem in 1736. However it was not until the 1970's that the link between graph partitioning and matrix spectra was observed when Donath and Hoffman (1973) established a lower bound for the number of edges

which must be cut in a graph partitioning problem, based on a factor of the eigenvectors of the affinity matrix. Around the same time Fieldler (1973) published work highlighting the importance of the value of the second smallest eigenvector of the affinity matrix when producing a two way graph partition, also known as a bi-partition. As bi-partitioning was studied the question arose, how to step from bi-partitioning problems to n-partitioning problems? The obvious two approaches are to either recursively apply bi-partitioning on the graph or calculate an n-partition directly from the n smallest eigenvectors. This is discussed further in Alpert (1995) where they claim that recursive bi-partitioning, which was more popular at the time, is inferior compared with creating one partition using many eigenvectors. For our purposes we will consider only the direct n-partitioning problem. For a full review of the historical origins of spectral clustering, see the tutorial (von Luxburg et al. (2008)).

The first paper to bring Spectral Clustering to the Machine Learning community was Shi and Malik (2000) which treated image segmentation as a graph-cut problem. Mathematically, a cut of a graph G = (V,E) is the set of edges S, such that the removal of S from G disconnects G. A graph is connected if there is a path from every vertex to every other vertex in G. The weight of the cut can be calculated by summing the weights of the edges which will be broken when a cut is made, as given in equation (1.2.1),

$$\text{cut(A,B)} = \sum_{p \in A, q \in B} w(p, q) \tag{1.2.1}$$

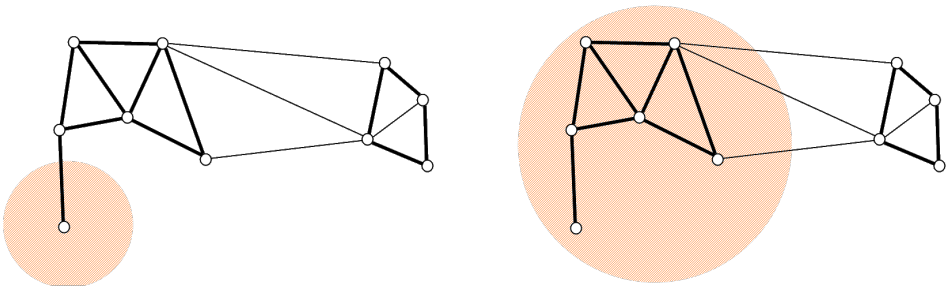
where A and B are the sub-graphs formed by cutting graph G, and p and q are the edges belonging to sub-graphs A and B respectively. The Minimum cut (Wu and Leahy, 1993) is simply the cut which generates the minimum cut weight. However the minimum cut does

not always provide the best solution; it tends to favour removing small sections and isolated nodes of the graph. The main proposal from Shi and Malik was to consider the cut function as a fraction of the total edge connections to all nodes in the graph, instead of using the total edge weight connecting the two partitions. This subtle yet important distinction provides us with the highly acclaimed normalised cut (NCut), defined in equation (1.2.2), which is an integral part of spectral clustering algorithms. The main idea is that big clusters will increase assoc(A,V), thus decrease Ncut(A,B). This will encourage splitting the data into fairly evenly sized clusters, and avoid the minimum cut issue of segmented isolated points. Minimising the normalised cut is an NP-complete problem but an approximate solution can be found efficiently by finding the eigenvector with the second smallest eigenvalue. Figure 1.2.1 shows both the minimum cut and normalised cut solutions for a particular graph. The shaded/non-shaded regions represent the partitioning. The minimum cut isolates one node from the rest of the graph, whilst the normalised cut provides a more balanced and sensible partition.

$$\text{Ncut(A,B)} = \frac{\text{cut(A,B)}}{\text{assoc(A,V)}} + \frac{\text{cut(A,B)}}{\text{assoc(B,V)}} \tag{1.2.2}$$

$$\text{assoc(A,V)} = \sum_{u \in A, t \in V} w(u,t)$$

On Spectral Clustering Ng et al. (2001) was one of the first papers to provide theoretical guarantees on performance for spectral clustering algorithms. Unlike previous authors who had only shown empirical results to justify spectral clustering abilities, they prove that their spectral clustering algorithm will produce a reasonable clustering, given certain assumptions



(a) Minimising the cut. (b) Minimising the normalised cut.

Figure 1.2.1: Two solutions to the bi partition problem. The partitioning is indicated by shading/non shading of nodes.

that the clusters are well-spaced. This version of the spectral clustering algorithm has been popularly cited throughout the literature and is the one that we shall reference in Section 1.3.

Although spectral clustering has been shown to perform well empirically on simple data sets, computational problems arise as the data set size increases. There have been a number of ways to deal with speed up in the static case, one of the most popular methods is to use Nyström efficiency methods (Williams and Seeger (2001); Fowlkes et al. (2004)). The Nyström method samples the columns of the affinity matrix and approximates the full matrix by using correlations between the sampled columns and the remaining columns. Effectively we can think of this sampling as a dial which the user has control over. Sampling more columns will provide a better results but at a cost. Although Nyström methods are approximation techniques to speed up the computation of spectral clustering, the working memory can still be high. Another drawback with Nyström is that due to random sampling it is possible to under represent or entirely miss smaller clusters. One alternative to Nyström methods is to perform a permutation on the data to act as a pre-processing step

to generate a smaller summary data set. By feeding a smaller set of representative points into the spectral clustering algorithm instead of the whole data set, we can lessen the effect of the computational bottleneck that comes with eigen analysis. Fast Approximate Spectral Clustering Yan et al. (2009) explores the theoretical guarantees on misclustering the data set, given that some permutation has been performed on the data before the spectral clustering. Specifically they exactly quantify the misclustering of data sets, given that the original data is summarised using K-means (KASP) or a Random Project tree (RASP) as preprocessors. This is explored further in Section 1.3.

Another method to deal with the computational challenge in spectral clustering is Local Information-based Fast Approximate Spectral Clustering (Li-ASP) introduced in Cao et al. (2014). Li-ASP consists of two upgrades; a sparse affinity graph to speed up computation and local interpolation to improve clustering performance. The sparse affinity graph uses a k nearest neighbour or an ϵ neighbourhood to set many elements in the affinity graph to zero. Local interpolation is suggested based on an issue identified about KASP Yan et al. (2009) that if a representative point is miss-assigned to the wrong cluster, then all data points represented by that point will also be miss-assigned. Local interpolation can avoid these situations by using labelling data points with a weighted version of their p closest representative points labels, rather than labelling based just on the label of the single closest representative point.

One constant challenge in clustering is selecting the number of clusters to search for. Zelnik-manor et al. (2004) present a method for automatically choosing the true number of clusters using the eigenvectors to inform their choice. More commonly the eigenvalues are used to estimate the number of clusters, but if the clusters are not clearly separated

identifying the number of clusters from eigenvalues alone is not trivial. We shall assume that the true number of clusters is known.

So far, we have only discussed spectral clustering of static data sets. We are interested in data streams, which are of great interest in today's world of communication graphs such as the Internet and social networks. Applications also include health care such as modelling epidemics and understanding sensor networks. Re-clustering the graph at each time step whenever new information arrives is not feasible especially if data is arriving rapidly.

To my knowledge, there is not currently a fully online method for Spectral Clustering. The problem of performing spectral clustering in data streams has been considered, but framed as an evolving network rather than a stream where data appears rapidly. Time evolving graphs are still an interesting problem, and are often found in social networks or biological applications.

The first incremental spectral clustering algorithm concerns topological mapping (Valgren et al. (2007)). Their algorithm updates the cluster estimates whenever ever a new data point (or batch of data points) arrives. The cluster membership is updated directly. The affinity matrix is periodically compressed to deal with large data sets. If a new data point is sufficiently far from it's closest representative points, it is considered the start of a new cluster, this means that the number of overall clusters must always increase.

An incremental update algorithm is proposed in Ning et al. (2007) and Ning et al. (2010) which can deal with both additional data points joining the network, and similarity weights changing between existing data points. The algorithm updates the eigenvectors and eigenvalues directly without performing a full eigen-decomposition. The addition of a new data point is treated as a series of n weight changes, where is n is the number of currently observed data

points. However the authors recommend a full re-clustering in batch to minimise cumulative errors. There are some issues with update method, mainly that the updating of eigenvectors means that the orthogonality property may be lost. Also if the spatial neighbourhoods of often changing vertices are large it can still be computationally difficult as the eigenvector update step involves the inversion of a matrix. Finally the authors recommend a full spectral re-clustering occasionally to prevent the accumulation of errors in the eigenvectors, this is not feasible in the streaming setting. Generally this method is not suitable for data streaming, as the size of the Laplacian can grown unbounded for an infinite data stream. However since it is the most well-known relevant "online" algorithm existing, we will compare performance against it in Section 1.5. RD: Might move Ning from experiments to 1.4.

Other incremental spectral clustering algorithms include Kong et al. (2011), Langone et al. (2014) and Dhanjal et al. (2011) which approximates the eigen decomposition of the Laplacian incrementally but still requires regular full re-clustering.

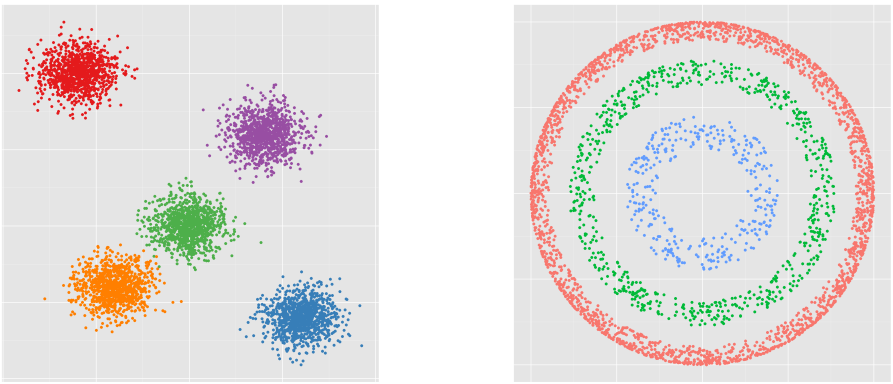
We consider an online spectral clustering algorithm based on the Clustream model of Aggarwal et al. (2003). The Clustream algorithm is introduced in full in Section 1.4, but first we discuss spectral clustering in more detail.

1.3 Spectral Clustering

In this Section we provide a brief introduction to Spectral Clustering, and discuss the choice of affinity matrices. A fast offline spectral clustering algorithm Yan et al. (2009) algorithm will be introduced, which inspired our implementation in Section 1.4.

The goal of clustering algorithms is to partition data ($\boldsymbol{X} = \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$, $\boldsymbol{x_i} \in \mathbb{R}^d$) into

k disjoint classes such that each $\boldsymbol{x_i}$ belongs to one and only one class. Clusters come in all shapes and sizes, they can be spherical and compact as in Figure 1.3.1a or connected but not visually compact as in Figure 1.3.1b. Data which is compact may be simple to cluster as the gaps between clusters are easy for simple clustering algorithms like k-means to identify. Connected but non-compact data sets are can be much more challenging than compact data sets, and can cause some simple clustering algorithms to fail. Spectral clustering can provide good quality segmentation on even these difficult cases, however its performance comes at the cost of computational complexity. We shall use the Jordan-Weiss (NJW) spectral clustering framework Ng et al. (2001) which is described in Algorithm 1.



(a) Compact clusters (b) Connected clusters

Figure 1.3.1: Examples of different types of clusters

The affinity matrix $A = (a_{ij})_{i,j=1}^{n}$ represents the pairwise similarities or distances between all data points x_i and x_j. A popular choice is to define A to be the Gaussian kernel, as defined in equation (1.3.1), where the parameter σ controls the width of the local neighbourhoods which we want to model.

$$a_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), i, j = 1, \ldots, n. \tag{1.3.1}$$

If x_i and x_j are very close, then $a_{i,j} \to 1$, and if they are far apart $a_{i,j} \to 0$. A gaussian kernel affinity matrix will have ones along the diagonal and is symmetric ($a_{ij} = a_{ji}$).

The value scaling parameter σ is usually chosen manually. Ng et al. (2001) automatically chose σ by running their clustering algorithm repeatedly for a number of values and selecting the one which provides least distorted clusters. Zelnik-manor et al. (2004) argue that for data which has a cluttered background, or multi-scale data, one global parameter choice for σ is not sufficient. They calculate a localised parameter σ_i for each data point x_i based on it's neighbourhood. In our experimentation we did not find it necessary to do this and use a global σ.

If we mainly wish to model the local relationships, using all of the possible pairwise data connections may not be necessary. It is possible to use a weighted k-nearest neighbour structure to build the affinity matrix once corrections have been made to ensure that this matrix is symmetrical. Another option is to choose some threshold ϵ and only consider connections between data points whose pairwise distances are smaller than ϵ. This is an ϵ-neighbourhood graph. Although it is possible to weight this graph by ϵ, if we choose ϵ to generate a small ϵ-neighbourhood, then the differences between the weights will be small that weighting may become eligible. Therefore a simple construction of the ϵ-neighbourhood graph is as shown in equation (1.3.2).

$$a^*_{ij} = \begin{cases} 1, & \text{if } a_{ij} < \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{1.3.2}$$

Using this construction will give a sparse affinity matrix instead of a fully connected graph, which will help lower the computational complexity. In Section 1.5, we use a fully connected graph for all experiments.RD: Possibly KNN now The degree of each vertex is defined as, $d_i = \sum_{j=1}^{N} A_{ij}$, the sum of the rows of the affinity matrix. The degree matrix, D, is then defined as a diagonal matrix with i^{th} diagonal element equal to d_i. We use the Normalised Symmetric Laplacian (Chung, 1997) defined in equation (1.3.3), which relates to an approximation of minimising the Normalised Cut, discussed in Section 1.2. RD: Should I introduce unnormalised and random walk normalised too?

$$L = D^{-1/2} A D^{-1/2} \tag{1.3.3}$$

RD: Would this be clearer if I explain unnormalised and state $L_{\text{symm}} = I - D^{-1/2} A D^{-1/2}$

Spectral clustering can be challenging for very large data sets, constructing the affinity matrix A and computing the eigenvectors of L have computational complexity $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ respectively. A fast approximate spectral clustering algorithm is proposed (KASP) is proposed Yan et al. (2009) which uses a k-means pre processing step to lessen the computational complexity whilst retaining good clustering performance. Firstly k-means is run on the whole data set where q is chosen to be large but such that $k \ll q$. The centres of the clusters are then used as representative data points for the whole data set. Spectral clustering is performed on the representative set only, which is significantly faster than performing spectral clustering on the full data set. The resulting cluster labels for the representative data are linked back to the original data set such that every original data point acquires

Algorithm 1 NJW spectral clustering algorithm

Input: Data set $X = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$, number of clusters k

Output: k-way partition of the input data

1: Construct the affinity matrix A by the following Gaussian kernel function:

$$a_{i,j} = \exp\left(-\frac{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|^2}{2\sigma^2}\right), i, j = 1, \ldots, n.$$

2: Compute the normalised affinity matrix $L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where D is the diagonal matrix with $D_{ii} = \sum_{j=1}^{n} a_{ij}$.

3: Compute the k eigenvectors of L, $v_1, v_2, \ldots, v_k$, associated with the k largest eigenvalues, and form the matrix $X = [\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_k}]$.

4: Renormalize each row of X to form a new matrix Y.

5: Partition the n rows into k clusters via a general cluster algorithm, such as the k-means algorithm.

6: Assign the original point $\boldsymbol{x_i}$ to the cluster k $\iff$ the corresponding row i of the matrix Y is assigned to the cluster k.

the same label as its associated k-means cluster centre. The KASP algorithm is repeated in Algorithm 2 with unified notation.

Algorithm 2 KASP

Input: Data set $X = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$, number of clusters k, number of representative points q

Output: q-way partition of the input data

1: Perform k-means with q clusters on $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$

2: Compute the cluster centroids $y1, \ldots, y_q$ as the q representative points.

3: Build a correspondence table to associate each x_i with the nearest cluster centroids y_j .

4: Run a spectral clustering algorithm on $y_1, \ldots, y_q$ to obtain an k-way cluster membership for each of y_q.

5: Recover the cluster membership for each x_i by looking up the cluster membership of the corresponding centroid y_j in the correspondence table.

The KASP algorithm has been shown to perform well empirically, and using representative points is a sensible way to lessen the computational burden. As the number of representative points increases, the better performance should be, but the greater the computational expense.

In online spectral clustering algorithm, we will use representative points as an input for spectral clustering. Rather than generating representative points from a k-means step, we constantly update the representative points by using the streaming algorithm, Clustream which is introduced in Section 1.4.

1.4 Micro-cluster based spectral clustering in data streams

In this Section we discuss online streaming methods, introduce the streaming algorithm Clustream, and consider how to use incorporate the streaming data into a spectral clustering algorithm.

1.4.1 The challenges of clustering data streams

A relatively new challenge to clustering is working with data streams Gama (2010); Silva et al. (2013). A data stream is data which arrives in an ordered sequence, continuously; for example, sensor data or online shopping transactions. There is no control over the order in which data objects should be processed. A data stream may be potentially unbounded, and the data points are often discarded after processing. Much work has been done developing offline clustering methods, such as spectral clustering, but it is not suitable to apply these offline methods directly to the streaming scenario. Simply running an offline clustering algorithm on all the data observed so far may not be feasible for three main reasons, storage capacity, computational costs and ability to access to the data.

The first challenge is storing all of the data. As a data stream is a potentially an endless sequence of observations obtained at a high frequency often, it may not be possible to store all of the data in its entirety. Therefore the older data has to be thrown away to make room for the new arrivals. This might not seem like a big issue as the data will naturally become weighted temporally, but this is challenging if we wish to incorporate historical data into the clustering.

Secondly, as we have discussed, clustering algorithms can be computationally expensive.

For example, computing the eigenvectors for spectral clustering has complexity $\mathcal{O}(n^3)$. As data streams are potentially unbounded, standard clustering algorithms cannot be used. Therefore we need to be able to update our idea of the data as new points arrive efficiently and simply with little computational issues.

Finally, data streaming is often classed as a "one-pass-access" problem. Imagine a constant stream of data flying past your window, you can view the data as it flies by the window, but once it has passed by, it cannot be accessed again. Some traditional clustering algorithms such DBScan (Ester et al. (1996)) require many passes or iterations of the data, therefore these type of clustering methods are not directly suitable for the online data streaming case.

1.4.2 Clustream

Clustream Aggarwal et al. (2003) offers a framework which allows quick and easy updates and the ability to perform sophisticated clustering algorithms. Clustream has proved popular, since the paper was first published in 2003 it has been cited over 1400 times. The main idea is to separate the clustering process in two stages, a micro-clustering stage and a macro-clustering stage. The micro-clustering stage continuously updates statistical summaries of the data stream, and the macro-clustering is more computationally intensive and run in batch or on a user request.

Micro-clustering

The micro-clustering stage is a way of maintaining an active, evolving representative summary of the data, without storing the absolute values of the data points. Micro-clusters are defined as a temporal extension of the cluster feature vector first described in Zhang et al. (1996).

The data stream is summarised by many small clusters, which are initially generated by k-means. The online phase stores q micro-clusters in memory, where q is an input parameter. We take an initial training set, and perform k-means with q clusters but choose the value of q to be much larger than the expected number of true macro-clusters k, ($k \ll q$). The aim here is to create a fine scale summary of the data. The value of q should be chosen to be as large as computationally comfortable. The larger q is, the finer scale that the summaries will be. It is vital to ensure that the micro-cluster well represent the underlying data set or else the macro-clustering will under perform. These q clusters are our first micro-clusters. Over time, we will update these micro-clusters, adding new data points to them, merging them and removing old micro-clusters, although the number of micro-clusters should stay fixed throughout.

The micro-clusters can then be used on a user request to perform a macro-clustering using the summarised data rather than the full data set. If the micro-clusters represent the true underlying data stream well, then the difference between the clustering on the summarised data and the true full data should be small.

Assume that we have a data stream S which consists of d-dimensional data $\boldsymbol{x_i}$ arriving in sequence. $S = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \ldots \boldsymbol{x_i}, \ldots, \}, \boldsymbol{x_i} \in \mathbb{R}^d$. Each micro-cluster M_j for $(j \in 1 \ldots, q)$ is stored as a $(2 \cdot d + 3)$ tuple $(\boldsymbol{CF1_j^x}, \boldsymbol{CF2_j^x}, n_j, CF1_j^t, CF2_j^t)$. The definitions are given in equation (1.4.1). $\boldsymbol{CF1_j^x}$ is the sum of all observed data in that micro-cluster, $\boldsymbol{CF2_j^x}$ is the sum of the squares of the data and n_j is the number of elements assigned to that micro-cluster. $CF1_j^t$ and $CF2_j^t$ refer to the sum of the time stamps, and the sum of squared time stamps respectively. Note that both $\boldsymbol{CF1_j^x}$ and $\boldsymbol{CF2_j^x}$ are d-dimensional vectors.

Each micro-cluster M_j will have

$$\begin{aligned} \boldsymbol{CF1^x_j} &= \sum_{x_i \in M_j} \boldsymbol{x_i} \ , \\ \boldsymbol{CF2^x_j} &= \sum_{x_i \in M_j} (\boldsymbol{x_i})^2 \ , \\ CF1^t_j &= \sum_{i | x_i \in M_j} t_i \ , \\ CF2^t_j &= \sum_{i | x_i \in M_j} (t_i)^2 \ , \\ n_j &= \sum_{x_i \in M_j} 1 \ . \end{aligned} \tag{1.4.1}$$

If a new data point x_{new} arrives at time t_{new} and is assigned to micro-cluster M_j, the update given in equation (1.4.2) is applied.

$$\begin{aligned} \boldsymbol{CF1^x_j} &\leftarrow \boldsymbol{CF1^x_j} + \boldsymbol{x}_{\text{new}} \ , \\ \boldsymbol{CF2^x_j} &\leftarrow \boldsymbol{CF2^x_j} + (\boldsymbol{x}_{\text{new}})^2 \ , \\ CF1^t_j &\leftarrow CF1^t_j + t_{\text{new}} \ , \\ CF2^t_j &\leftarrow CF2^t_j + (t_{\text{new}})^2 \ , \\ n_j &\leftarrow n_j + 1 \ . \end{aligned} \tag{1.4.2}$$

Note that updating the micro-clusters requires only addition therefore updating is cost effective. Critically it is possible to use these summaries to calculate the centre of each micro-cluster as in equation (1.4.3). It is these centres which as used as representative points for input into the macro-clustering. As new points in the data stream arrive, they are either allocated to a micro-cluster and the update procedure discussed above is carried out, or a new micro-cluster is created. The decision for a new micro-cluster to be created is based on whether the new data point is close enough to it's nearest cluster centre.

$$\text{Centre of micro-cluster j} = \bar{\boldsymbol{M}}_{\boldsymbol{j}} = \frac{\boldsymbol{CF1^x_j}}{n_j} \tag{1.4.3}$$

When a new data point arrives it's nearest micro-cluster M_* is identified using the Euclidean distance metric given in equation (1.4.4). If the data point falls within the Maximum Boundary Factor of it's nearest cluster centre, then it is absorbed as part of that cluster. If not, it is used to create a new micro-cluster. However we stated earlier than the number of micro-clusters must remain fixed throughout the process. Therefore if a new micro-cluster is formed, either an existing micro-cluster must be deleted, or two close micro-clusters should be merged. We follow the methodology in Clustream by first looking for an old micro-cluster to delete using the time-stamp references detailed in the original paper and otherwise combine the two nearest micro-clusters. In this way, the algorithm tracks the data stream as it evolves.

$$M^* = \underset{M_j, j \in 1:q}{\arg\min} \|\boldsymbol{x_i} - \bar{\boldsymbol{M}}_{\boldsymbol{j}}\|^2 \tag{1.4.4}$$

In Clustream, the maximum boundary factor is defined as a factor of t of the RMS deviation of the data points in M_j from the centroid of M_j. The RMS deviation can only be defined for a cluster with more than one point. For a cluster with only one previous point, a heuristic is used to define the MBF. Details of the MBF can be found in Aggarwal et al. (2003)

With this online micro-cluster maintenance, the data stream should remain well represented over time. When a new data point arrives if it is the start of a new evolving cluster it will be allowed to grow however if it is an outlier no more points will be added to it and

over time it may be deleted from the system all together.

If two micro-clusters M_r and M_s are to be merged, the updates given in equation (1.4.5) are used to merge them into M_r, and M_s will be deleted. Again as all of these updates only involve addition steps, they are fast to implement.

$$\begin{aligned} \boldsymbol{CF1^x_r} &\leftarrow \boldsymbol{CF1^x_r} + \boldsymbol{CF1^x_s} \,, \\ \boldsymbol{CF2^x_r} &\leftarrow \boldsymbol{CF2^x_r} + \boldsymbol{CF2^x_s} \,, \\ CF1^t_r &\leftarrow CF1^t_r + CF1^t_s \,, \\ CF2^t_r &\leftarrow CF2^t_r + CF2^t_s \,, \\ n_r &\leftarrow n_r + n_s \,. \end{aligned} \tag{1.4.5}$$

The micro-clustering update algorithm for Clustream is given in Algorithm 4.

Macro-clustering stage

The second stage of clustream is a macro-clustering stage, where we take the current micro-cluster feature vectors, and use these as input into global clustering algorithm. The macro-clustering step is where the general data summary is transformed into a snapshot of the true underlying clusters at that point in the stream. The q micro-cluster centres $\bar{M}_j, (1 \leq j \leq q)$ are treated as representative points for the data stream S, and a standard clustering algorithm can be used to determine clusters. The nature of this algorithm allows the user to get close to online streaming and perform spectral clustering on a summary of the whole of the data set. The full algorithm is given in Algorithm 4.

There are a number of possible ways to feed the micro-clusters into a spectral clustering algorithm. Two of the options suggested in Zhang et al. (1996) are implemented here.

Algorithm 3 Clustream Microclustering

Input: Data Stream $S = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$, number of micro-clusters q

Output: Microclusters $M_1, \ldots, M_q$

Initialise the micro-clusters kmeans$(x_1, \ldots x_{init}, q)$ and equations (1.4.1)

for each new data point x_i **do**

Find the closest micro-cluster to x_i, M_* using equation (1.4.4)

if x_i satisfies the MBF criterion for M_* **then**

absorb x_i into micro-cluster M_* using equations (1.4.2)

else

Use x_i to initialise it's own new micro-cluster using equations (1.4.1)

if any micro-cluster is suitably old **then**

Remove it

else

Merge the two closest micro-clusters using equation (1.4.5)

end if

end if

end for

1. Calculate the centre of each micro-cluster $\bar{M_j}$ and use it as an object to be clustered by the macro-clustering algorithm.

2. Do the same as before, but weighting each micro-cluster centre $\bar{M_j}$ proportionally to n_j, the number of points assigned to that micro-cluster, so that micro-clusters with more objects will have more influence on the final clustering.

Algorithm 4 Online Spectral Clustream

Input: Data Stream $S = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$, number of clusters k, number of micro-clusters q

Output: A k way clustering of the microclusters $M_1, \ldots, M_q$.

```
Initialise the micro-clusters kmeans(x_1, ... x_init, q) and equations (1.4.1)
for each new data point x_i do
    Apply Clustream update as in Algorithm 3
    if A Macro-clustering is required then
        Perform spectral clustering on M_1, ..., M_q with k clusters.
    end if
end for
```

In Section 1.5 we analyse the performance of both unweighted and weighted Online Spectral Clustering, but first we further investigate how to use weighted micro-clusters in Spectral Clustering.

1.4.3 Weighting the Micro-Clusters

In this section, we discuss how can create a weighted affinity matrix, look at the effect this has on the Laplacians, and note a spectral link between weighting in this manner, and a

larger affinity of repeated points.

Why would it be beneficial to weight micro-clusters? For example Figure 1.4.1 shows how the distribution of points to microclusters may change as the stream progresses. Figure 1.4.1a shows a histogram of the number of points assigned to micro-clusters at the start of the stream, Figure 1.4.1b shows the middle of the stream, and Figure 1.4.1c shows the end of the stream. We can see that the distribution is not uniform. Therefore some information is contained in the number of points assigned to a micro-cluster.

(a) Start of the stream (b) Middle of the stream (c) End of the stream

Figure 1.4.1: CREATE PLOTS Histograms showing the number of points assigned to microclusters

In order to weight the the micro-clusters, we simply construct an affinity as described. Let $A \in \mathbb{R}^{q \times q}$ be the affinity matrix of the micro-cluster centres with i, j-th element equal to the similarity between micro-cluster M_i and M_j,

$$A_{i,j} = \exp\left(-\frac{\|\bar{M}_i - \bar{M}_j\|^2}{2\sigma^2}\right), \quad i, j = 1, \ldots, q.$$

.

Let $\tilde{A} \in \mathbb{R}^{q \times q}$ have i, j-th element equal to $n_i n_j A_{ij}$. $\tilde{A}$ is a valid affinity matrix since it is symmetric with non-negative entries. If we wish to have $\tilde{A}_{ij} \leq 1$ then simply divide $\tilde{A}$ by $max_i n_i^2$, but this makes no difference to the spectral decomposition.

There exists a link between the spectral decomposition of the Laplacian generated by $\tilde{A}$ and the Laplacian arising from a data set of repeated points, which we define as follows. Let $A^* \in \mathbb{R}^{n \times n}$ be the repeated affinity matrix with the micro-cluster centres repeated based on the number of points assigned to them. Assume that the columns (and therefore rows) of A^* are ordered such that the first n_1 are associated with the data assigned to micro-cluster 1, which has size n_1 and the next n_2 with those assigned to micro-cluster 2, and so on.

Let $D, \tilde{D}, D^*$, be the corresponding degree matrices and $L, \tilde{L}, L^*$ be the corresponding normalised symmetric Laplacians.

Lets look at the Affinity and Laplacian matrices more closely for a very simple case.

Assume we have two micro-clusters, M_1 and M_2, which have n_1 and n_2 points assigned to them respectively. Let the similarity between the two micro-clusters centres be s, and assume that we are using the standard Gaussian kernel to generate affinity matrices, so therefore the diagonal elements will be equal to 1.

$$A = \begin{pmatrix} 1 & s \\ s & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1+s & 0 \\ 0 & 1+s \end{pmatrix}, \quad L_{\text{symm}} = \begin{pmatrix} \frac{1}{1+s} & \frac{s}{1+s} \\ \frac{s}{1+s} & \frac{1}{1+s} \end{pmatrix}$$

In order to create a weighted version of the affinity matrix, we simple multiply through by n_1 and n_2. We can see how this is incorporated into the Laplacian. Note that we are working with the Symmetric Normalised Laplacian. In order to weight in this manner for the unnormalised Laplacian a scaling of the diagonal matrix is required.

$$\tilde{A} = \begin{pmatrix} n_1^2 & sn_1n_2 \\ sn_1n_2 & n_2^2 \end{pmatrix}, \quad \tilde{D} = \begin{pmatrix} n_1^2 + sn_1n_2 & 0 \\ 0 & n_2^2 + sn_1n_2 \end{pmatrix}$$

$$\tilde{L}_{\text{symm}} = \begin{pmatrix} \frac{n_1^2}{n_1^2+sn_1n_2} & \frac{sn_1n_2}{\sqrt{(n_1^2+sn_1n_2)(n_2^2+sn_1n_2)}} \\ \frac{sn_1n_2}{\sqrt{(n_1^2+sn_1n_2)(n_2^2+sn_1n_2)}} & \frac{n_2^2}{n_2^2+sn_1n_2} \end{pmatrix}$$

Now lets look at the repeated affinity matrix. Here we can see the block nature in A^*, D^* and L^*.

$$A^* = \begin{pmatrix} 1 & \dots & 1 & s & \dots & s \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \dots & 1 & s & \dots & s \\ s & \dots & s & 1 & \dots & 1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ s & \dots & s & 1 & \dots & 1 \end{pmatrix}, \quad D^* = \begin{pmatrix} \star & 0 & 0 & 0 & \dots & 0 \\ 0 & \ddots & 0 & \vdots & \ddots & \vdots \\ 0 & 0 & \star & 0 & \dots & 0 \\ 0 & \dots & 0 & \triangle & 0 & 0 \\ \vdots & \ddots & \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & 0 & 0 & \triangle \end{pmatrix}$$

where $\star = n_1 + n_2 s$ and $\triangle = n_1 s + n_2$.

$$L^*_{\text{symm}} = \begin{pmatrix} \frac{1}{\star} & \dots & \frac{1}{\star} & \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\star} & \dots & \frac{1}{\star} & \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} \\ \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} & \frac{1}{\triangle} & \dots & \frac{1}{\triangle} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} & \frac{1}{\triangle} & \dots & \frac{1}{\triangle} \end{pmatrix}$$

If we evaluate these expressions for a particular case, we can see how the spectral decomposition of the matrices are linked.

Let $s = 0.5$, $n_1 = 3$, $n_2 = 2$.

The 2nd smallest eigenvector of L^*_{symm} is

$$e_2^* = \begin{bmatrix} -0.350 & -0.350 & -0.350 & 0.562 & 0.562 \end{bmatrix}$$

The 2nd smallest eigenvector of $\tilde{L}_{\text{symm}}$ is

$$\tilde{e}_2 = \begin{bmatrix} 0.607 & -0.795 \end{bmatrix}$$

If we expand the eigenvector $\tilde{e}_2$ by repeating it's elements and block divide by $\sqrt{n_1}$ and $\sqrt{n_2}$ respectively, we get

$$\left(\overbrace{\frac{0.607}{\sqrt{3}} \; \frac{0.607}{\sqrt{3}} \; \frac{0.607}{\sqrt{3}}}^{n_1} \; \overbrace{\frac{-0.795}{\sqrt{2}} \; \frac{-0.795}{\sqrt{2}}}^{n_2} \right) = \begin{pmatrix} 0.350 & 0.350 & 0.350 & -0.562 & -0.562 \end{pmatrix}$$

which is the negative of the 2nd smallest eigenvector of L^*_{symm}, e_2^*. In fact, we will always have that the expanded repeated eigenvector of the weighted Laplacian equal to either e_k^* or $-e_k^*$ for all k.

We can see more generally that, RD: What can we say generally?!

Let $\tilde{u}$ be the second smallest eigenvector of $\tilde{L}$.

Define $u^* = \left(\overbrace{\frac{\tilde{u}_1}{\sqrt{n_1}}, \frac{\tilde{u}_1}{\sqrt{n_1}}, \frac{\tilde{u}_1}{\sqrt{n_1}}}^{n_1}, \ldots, \overbrace{\frac{\tilde{u}_k}{\sqrt{n_k}}, \frac{\tilde{u}_k}{\sqrt{n_k}}}^{n_k} \right)$.

We can see that $\|u^*\|^2 = \sum_{i=1}^{n} (u_i^*)^2 = \sum_{i=1}^{k} n_i \cdot \frac{\tilde{u}_i^2}{n_i} = \sum_{i=1}^{k} \tilde{u}_i^2 = 1$.

1.5 Experimentation

In the experimentation section, we investigate the performance of Spectral Clustream, both unweighted and weighted variants, along with a simple windowed approach.

The Algorithms

Spectral Clustream is our implementation of Algorithm 3. As highlighted in Section 1.4, there are two ways we can incorporate the micro-cluster centres into the macro-clustering. Unweighted clustream takes the micro-cluster centres as direct input into the spectral clustering algorithm. Weighted clustream weights the micro-cluster centres by the number of data points assigned to that micro-cluster. In both clustream algorithms, we use 150 micro-clusters to monitor the streams.

Windowed Spectral Clustering is a simple algorithm stores a window of size w of the most recently observed data points. When a new data point is observed, the oldest data point in the window is discarded to make room for the new data point. Only the data points in the current window are available for input into a standard Spectral Clustering algorithm. We use a fixed window size of 150 data points for the duration of the stream.

1.5.1 Methodology

A simulated data stream $S = \boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ is observed sequentially, with one data point $\boldsymbol{x_t}$ observed at each time point $t \in 1, \ldots, n$. Clustream requires the micro-clusters to be initialised. We do this by applying kmeans with 150 clusters to the first 500 points. The initial micro-cluster components are then calculated using the equations described in equation (1.4.1). Spectral Windowed uses 150 data points to initialise the data window.

At each time point t, a new data point $\boldsymbol{x_t}$ is observed and is used to update the streaming algorithms. For clustream algorithms, this means updating your micro-clusters in the way outlined in Algorithm 3. For windowed spectral clustering this is done by just shifting your

window along by one, forgetting the oldest data point $\boldsymbol{x_{(t-150)}}$ and including $\boldsymbol{x_t}$.

In batch, every ten time steps, we use the current snapshot of the data to apply spectral clustering. In clustream this means feeding the centres of the micro-clusters (weight adjusted or not) into a standard spectral clustering algorithm. In windowed spectral, this means using all data points in your window as input into a standard spectral clustering algorithm.

We then evaluate the spectral clustering performance. The following step is repeated 10 times, and results are averaged out. Simulate a new test set of 200 data points using the underlying, unknown data stream parameters at time t. Assign each of the 200 points to their closest micro-cluster centre (or data point in the current window) using the nearest neighbour algorithm. These 200 points will then take on the macro-cluster assignment given to their nearest neighbour. Performance is measured in terms of purity and V-measure using the known true clusters of the simulated data.

Note that when comparing performance on the real data sets obviously we cannot simulate from the underlying distribution. Instead we look forward to the next 200 data points, assign them using nearest neighbour and continue in the manner above.

Performance Measures

The two measures we use to quantify cluster performance are purity and V-measure, both of which are well used in the clustering literature. Both measures require knowledge of the "true class" of the data points, which may not always be available for real data sets. Purity is the more intuitive of the two measures to understand as it quantifies how pure clusters are. Clusters which consist of just data points of one cluster will contribute towards a high purity score, whilst clusters consisting of data points from many different true classes will lower the

purity. To calculate purity, for each cluster find the true class which is most prevalent in that cluster and count how of that class are in that cluster. Do this for all clusters, sum these counts and divide by the total number of data points. This is shown mathematically in equation (1.5.1).

$$\text{Purity} = \frac{1}{N}\sum_{k}\max_{j}|\omega_k \in c_j|, \tag{1.5.1}$$

where we have clusters c_j and true classes ω_k.

V-measure (Rosenberg and Hirschberg (2007)) takes the harmonic mean of two other performance measures, homogeneity and completeness. Homogeneity assesses if each cluster contains members of only a single class, whilst completeness checked that all members of the same class are assigned to the same cluster. This is shown in equation (1.5.2).

$$\text{V-measure} = 2\frac{h \times c}{h + c}\,, \tag{1.5.2}$$

where h and c are homogeneity and completeness measures respectively defined in Rosenberg and Hirschberg (2007). Both purity and V-measure take values between 0 and 1, where 1 indicates perfect performance.

1.5.2 Simulated Results

The three simulated data sets we use are all normally distributed, one is static for the duration of the stream, one exhibits an abrupt change, and another illustrates concept drift.

The static data set is a simulated from a multivariate normal distribution with parameters $\pi = (0.5, 0, 5)$, $\mu_1 = (0, 0)$, $\mu_2 = (5, 5)$, and unit variance. There are 2 dimensions, 2 clusters,

and the data set remains static for the duration of the stream. This is shown in Figure 1.5.1a.The abrupt change data set is initially distributed the same as the static data set, but at $t = 1000$, μ_1 changes from $(0, 0)$ to $(15, 15)$, and stays as such until the end of the stream. Figure 1.5.1b shows the abrupt data set before the change point and Figure 1.5.1c shows the data at a point in the stream after the change has occurred. The model used for simulating gradual concept drift is

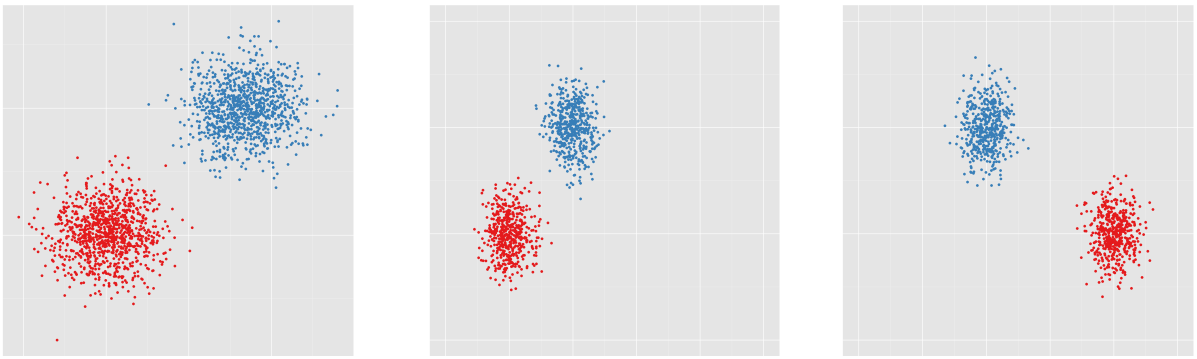
$$(\boldsymbol{X}_t|Y_t = 0) \sim \mathcal{N}(\boldsymbol{\mu}_t^0, \boldsymbol{\Sigma}) \tag{1.5.3}$$

$$(\boldsymbol{X}_t|Y_t = 1) \sim \mathcal{N}(\boldsymbol{\mu}_t^1, \boldsymbol{\Sigma}), \tag{1.5.4}$$

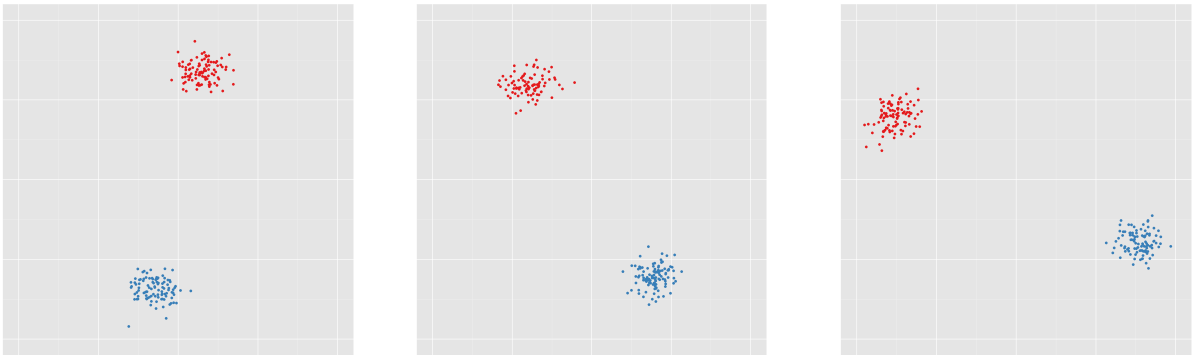
where $Y_t \sim Bern(p)$ for some user defined p and $\boldsymbol{\Sigma}$ is a multiple of the identity matrix. For our data set $p = 0.5$ and $\boldsymbol{\Sigma} = 0.005 * \mathbb{1}$. In order to model gradual concept drift the mean vectors are chosen such that the paths $\boldsymbol{\mu}_1^{(0)}, \ldots, \boldsymbol{\mu}_t^{(0)}$ and $\boldsymbol{\mu}_1^{(1)}, \ldots, \boldsymbol{\mu}_t^{(1)}$ trace out complimentary orbits on a d dimensional hyper-sphere. We have $d = 3$, and set the means such that a quarter of a sphere is covered during the stream. The path traces can be viewed in Figures 1.5.1d, 1.5.1e and 1.5.1f.

Figure 1.5.2 shows performance of unweighted spectral clustream, weighted spectral clustream and windowed spectral clustering on a number of simulated data sets.

The first thing to note is that unweighted clustream and the windowed algorithm both generally perform excellently on all data sets for the duration of the stream. The gradual drift data set doesn't seem to be too challenging for these algorithms, and although there is a downward spike in the abrupt data set at the point of change, both algorithms adapt and recover quickly. The weighted version of clustream starts with good performance in all three



(a) Static Normal (b) Jump start (c) Jump End



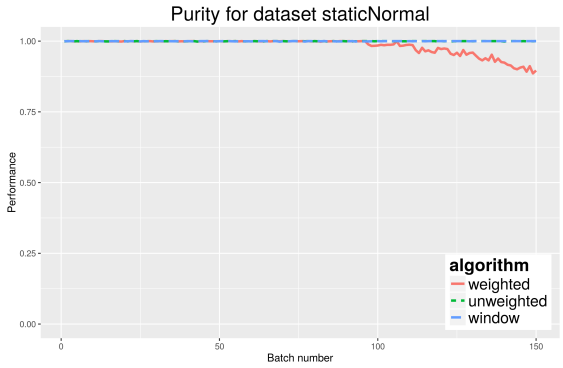
(d) Evolving Start (e) Evolving Middle (f) Evolving End

Figure 1.5.1: Descriptions of the simulated data sets. Figure 1.5.1a shows the Static Normal Data set. Figures (1.5.1b, 1.5.1c) shows the Jump data set before and after the changepoint. Figures (1.5.1d, 1.5.1e,1.5.1f) show the evolving data set at the start, in the middle and at the end of the stream.

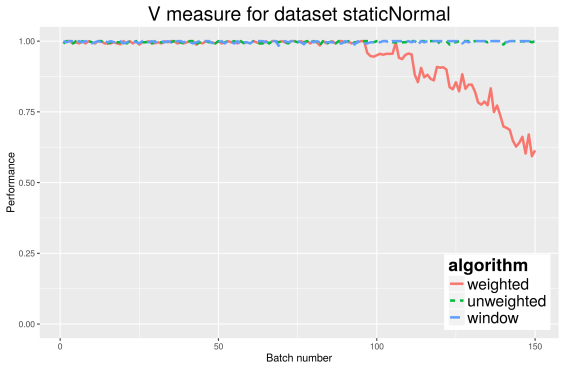
sets, but rapidly deteriorate as the streams progress, at times dropping to V-measure values of below 0.5.

1.5.3 Real Data Results

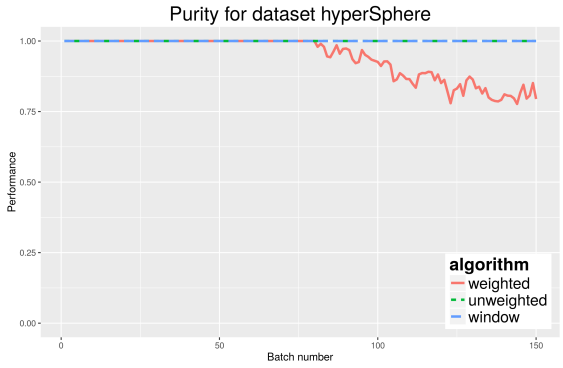
We investigate the performance of Clustream spectral clustering and Windowed Spectral clustering on real data. The data set is taken from the UCI Pendigits data set and consists



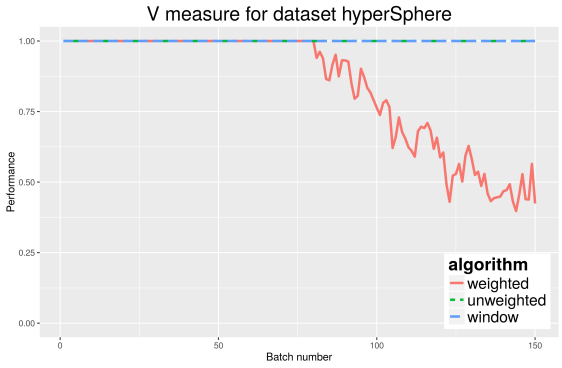
(a) Static Normal Purity



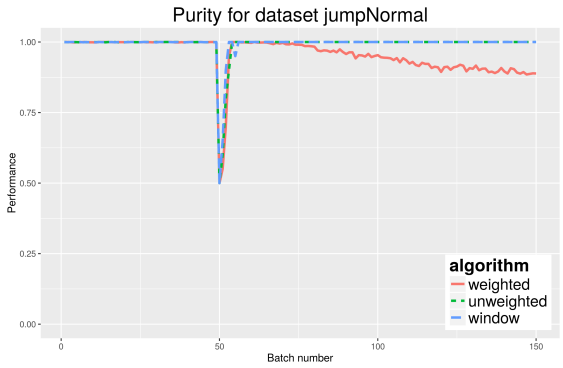
(b) Static Normal V-measure



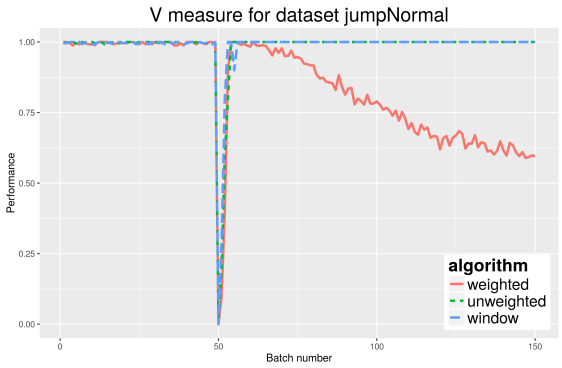
(c) Evolving Normal Purity



(d) Evolving Normal V-measure



(e) Jump Normal



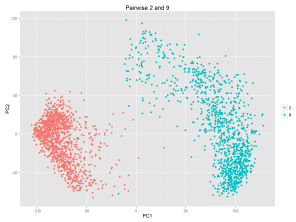
(f) Jump Normal V-measure

Figure 1.5.2: Performance of Spectral Clustream (weighted and unweighted) and windowed spectral

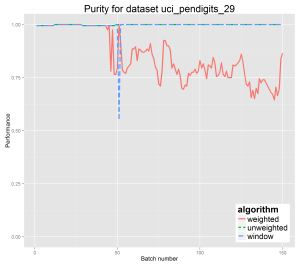
of hand drawn digits of the numbers 0-9. There are 250 samples taken from 44 writers. The data was collected using a pressure sensitive tablet. There are 16 features, each relating to

the co-ordinate information taken from the input tablet. We restrict our analysis to pairwise comparison of digits. For example we attempt to cluster the digits 0 and 1, and treat the data as if it is arriving in a constant data stream.

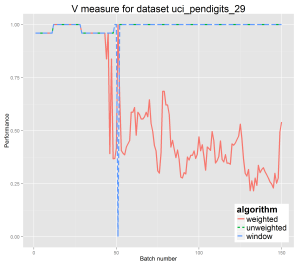
The results for a selection of the pairwise digits are shown in Figure 1.5.3. The first column displays the digit data in PCA space, the second column shows the purity, and the third column shows the V-measure. The pendigit data is more challenging than the simulated data, however both windowed spectral and the unweighted spectral clustream generally do well. Again we observe that weighted Clustream has poor performance, and it seems that once the performance drops, it struggles to improve again. Interestingly, occasionally windowed spectral clustering seems to heavily drop in performance, making a bad clustering, but it recovers quickly due to the nature of the relatively small window size.



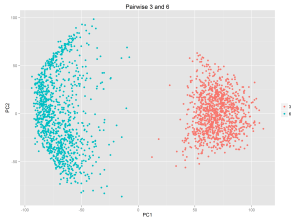
(a) PCA of digits 2 and 9



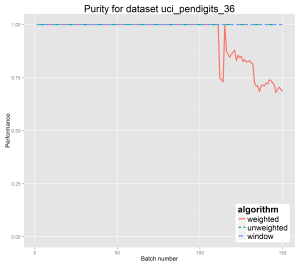
(b) Purity digits 2 and 9



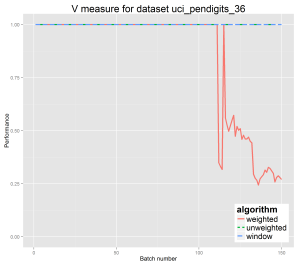
(c) V-measure digits 2 and 9



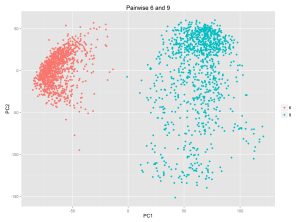
(d) PCA of digits 3 and 6



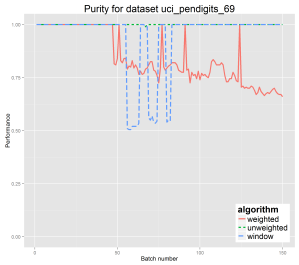
(e) Purity digits 3 and 6



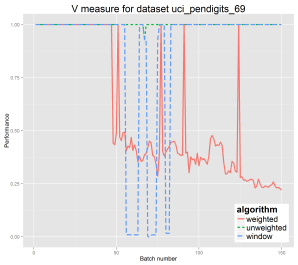
(f) V-measure digits 3 and 6



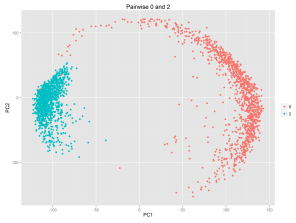
(g) PCA of digits 6 and 9



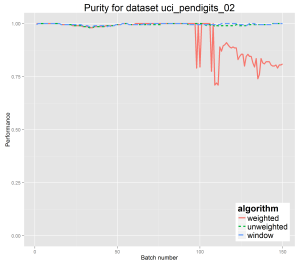
(h) Purity digits 6 and 9



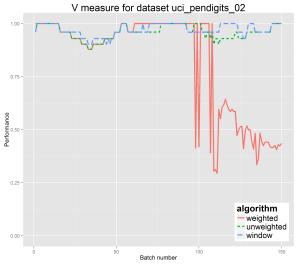
(i) V-measure digits 6 and 9



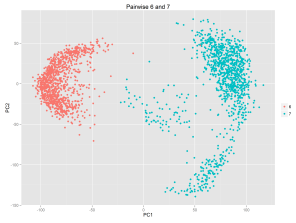
(j) PCA of digits 0 and 2



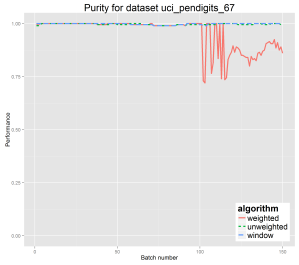
(k) Purity digits 0 and 2



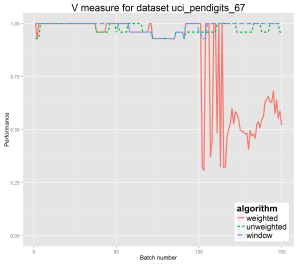
(l) V-measure digits 0 and 2



(m) PCA of digits 6 and 7



(n) Purity digits 6 and 7



(o) V-measure digits 6 and 7

Figure 1.5.3: Pendigits Pairwise - Clustream and windowed spectral

Texture data

We also tested there methods on another set of real data based on textured images. The Kylberg (Kylberg, 2011) texture data set consists of 28 texture classes with 160 unique texture patches per class. The patches consist of 576×576 pixel images.

In order to extract features to cluster, I used the LS2W (Eckley and Nason, 2011) method to extract wavelet features of from the textured images.

I selected a subset of 6 classes to run the algorithms on, examples of all 6 textures are shown in Figure 1.5.4 . The classes selected represent, two types of blanket, canvas, a ceiling, some lentils and a screen.

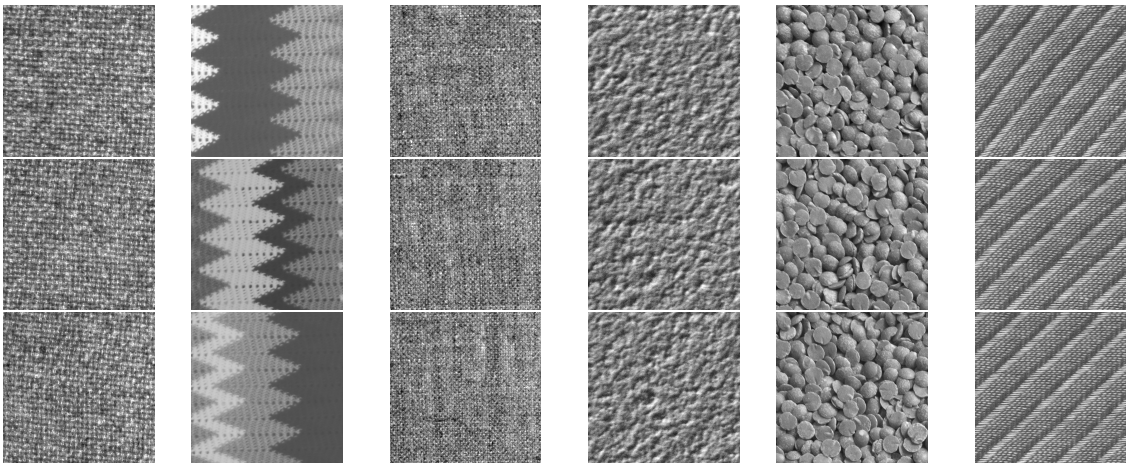
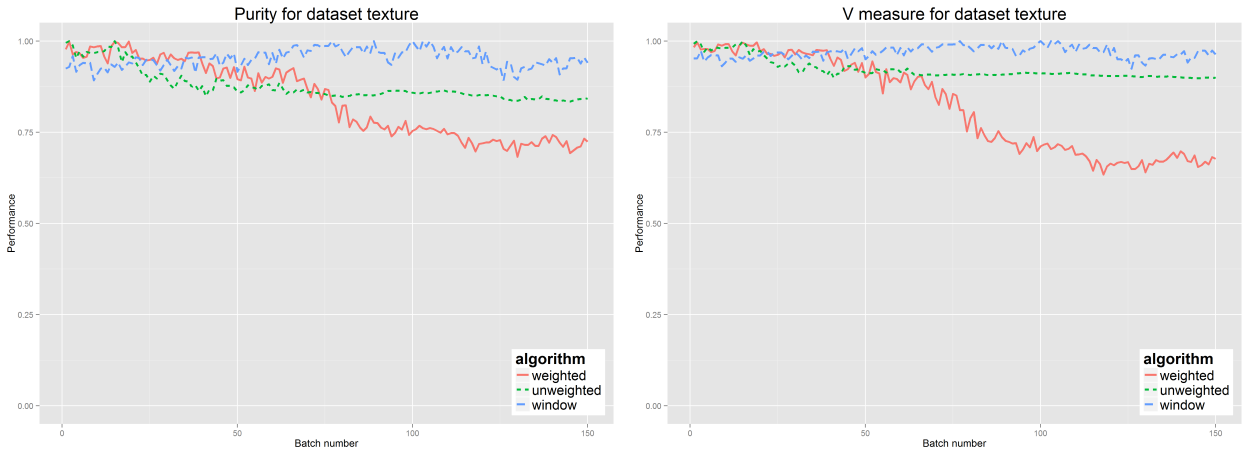


Figure 1.5.4: Three examples from each of the 6 different texture tiles used in the streaming data set. The texture classes are (L-R) Blanket 1, Blanket 2, Canvas, Ceiling, Lentils and Screen

The performance plots for the texture data are shown in Figure 1.5.5. Here we can see a difference between windowed spectral clustering and unweighted clustream, the windowed approach is generally performing better. Once again weighted clustream does not perform

well, and performance seems to decline as the stream progresses.



(a) Texture Purity

(b) Texture V-measure

Figure 1.5.5: Texture Results

1.5.4 Optional Extras

- Evaluate the number of micro-clusters needed for good performance. Give some insight into choosing the number of micro-clusters q. How does performance change based on the value of q? Aggarwal et al. (2003) suggests using the biggest value of q computationally possible. I suspect diminishing returns when increasing q. Can I say something about this?

- Show the distribution of micro-cluster sizes. Often it is very non-uniform, is it this which is effecting the performance of weighted spectral clustering?

- Explain why Ning was not used in experimental results - too slow!

Spectral Clustream vs Ning (without re-clustering)

In order to implement the Ning algorithm, the authors recommend a regular full re-clustering of the data sets to stop errors in the eigenvector estimates growing too large.

In Figure 1.5.6 I run both Ning and Clustream on the "Norm overlap" data set. We do not perform re-clustering for the Ning algorithm at all, so we can view how bad performance may get over time. This was run over a fairly short stream of just 200 data points. Cluster performance was requested every 10 time steps, resulting in 20 performance points.

Although performance does seems to deteriorate over time, it isn't as dramatic as I expected. It would be nice to see this for a much longer stream (5000 data points?)

We can also see that Spectral Clustream is generally performing much better in terms of both purity and V-measure. It would be good to see something similar on real data.

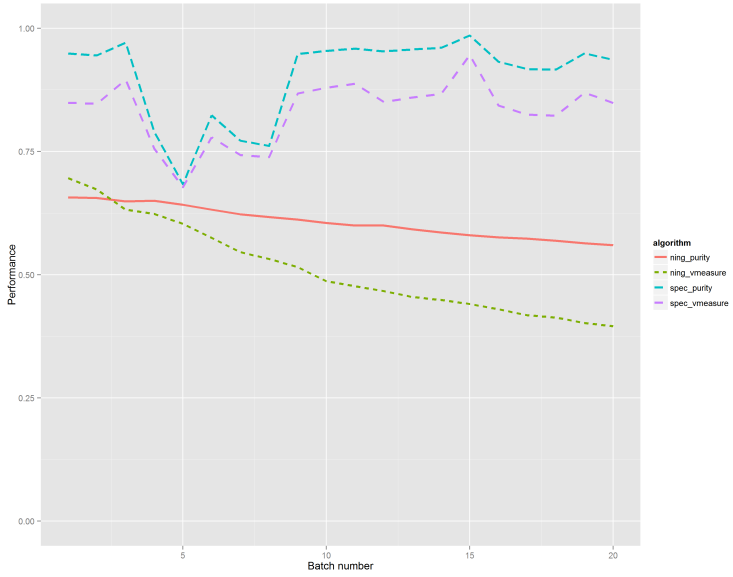


Figure 1.5.6: Ning (no re clustering) vs Spectral Clustream

However, I have found Ning very slow to run. The update method can be very expensive for fully connected networks, which makes it not feasible to use as it is in this study. However

by using an ϵ neighbourhood to force the updates to be minimal I should be able to achieve a vast speed up. This will be tried on my return. I will also implement the regular re-clustering as recommended by the authors but this re-clustering will have to be over a window as it is not possible to performance a truly full re-clustering in a streaming setting.

1.6 Conclusion

- The way to compare clustering algorithms in a streaming setting is not obvious
- We may want a clustering to accurately represent the data stream in it's current state, however we may also wish to incorporate older stream information into the clustering.

Bibliography

Aggarwal, C. C., Watson, T. J., Ctr, R., Han, J., Wang, J., and Yu, P. S. (2003). A Framework for Clustering Evolving Data Streams. *Proceedings of the 29th international conference on Very large data bases*, pages 81–92.

Alpert, S.-Z. Y. C. J. (1995). Spectral Partitioning: The More Eigenvectors, The Better. In *32nd Design Automation Conference*, pages 195–200. ACM.

Cao, J., Chen, P., Dai, Q., and Ling, W. K. (2014). Local information-based fast approximate spectral clustering. *Pattern Recognition Letters*, 38(1):63–69.

Chung, F. R. (1997). *Spectral graph theory*, volume 92. American Mathematical Soc.

Dhanjal, C., Gaudel, R., and Clémençon, S. (2011). Incremental spectral clustering with the normalised laplacian.

Donath, W. E. and Hoffman, A. J. (1973). Lower Bounds for the Partitioning of Graphs. *IBM Journal of Research and Development*, 17(5):420–425.

Eckley, I. and Nason, G. (2011). LS2W: Implementing the locally stationary 2d wavelet process approach in R. *Journal of Statistical Software*, 43(3):1–23.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Kdd*.

Fieldler, M. (1973). Algebraic Connectivity of Graphs. *Czechoslovak Mathematical Jornal*, 23(2):298–305.

Fowlkes, C., Belongie, S., Chung, F., and Malik, J. (2004). Spectral grouping using the Nystr{ö}m method. *IEEE transactions on pattern analysis and machine intelligence*, 26(2):214–225.

Gama, J. (2010). *Knowledge discovery from data streams*, volume 12. CRC Press.

Kong, T., Tian, Y., and Shen, H. (2011). A Fast Incremental Spectral Clustering for Large Data Sets. In *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 1–5. IEEE.

Kylberg, G. (2011). The kylberg texture dataset v. 1.0. External report (Blue series) 35, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden.

Langone, R., Mauricio Agudelo, O., De Moor, B., and Suykens, J. A. (2014). Incremental kernel spectral clustering for online learning of non-stationary data. *Neurocomputing*, 139:246–260.

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On Spectral Clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, pages 849–856.

Ning, H., Xu, W., Chi, Y., Gong, Y., and Huang, T. (2007). Incremental Spectral Clustering With Application to Monitoring of Evolving Blog Communities. *SIAM International Conference on Data Mining*, pages 261–272.

Ning, H., Xu, W., Chi, Y., Gong, Y., and Huang, T. S. (2010). Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113–127.

Rosenberg, A. and Hirschberg, J. (2007). 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*, 1(June):410–420.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

Silva, J. a., Faria, E. R., Barros, R. C., Hruschka, E. R., Carvalho, A. C. P. L. F. D., and Gama, J. (2013). Data stream clustering. *ACM Computing Surveys*, 46(1):1–31.

Valgren, C., Duckett, T., and Lilienthal, A. (2007). Incremental Spectral Clustering and Its Application To Topological Mapping. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4283–4288. IEEE.

von Luxburg, U., Belkin, M., and Bousquet, O. (2008). Consistency of spectral clustering. *The Annals of Statistics*, 36(2):555–586.

Williams, C. K. and Seeger, M. (2001). Using the Nystrom Method to Speed up Kernel Machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688.

Wu, Z. and Leahy, R. (1993). An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113.

Yan, D., Huang, L., and Jordan, M. I. (2009). Fast approximate spectral clustering. In *Work*, page 907, New York, New York, USA. ACM Press.

Zelnik-manor, L., Zelnik-manor, L., Perona, P., and Perona, P. (2004). Self-tuning spectral clustering. *Advances in Neural Information Processing Systems 17*, pages 1601–1608.

Zhang, T., Ramakrishnan, R., and Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Record*, 25(2):103–114.