Contents

1 Spectral Clustering for Data Streams **1**

1.1 Introduction 1

1.2 Spectral Clustering Background 1

1.2.1 Motivation 1

1.2.2 Graph cut problems 3

1.2.3 Choice of affinity matrix 8

1.3 Advanced Spectral Clustering 9

1.3.1 Large-scale Spectral Clustering 10

1.3.2 Incremental methods for Spectral Clustering 13

1.4 Online Spectral Clustering 15

1.4.1 The challenges of clustering data streams 15

1.4.2 Clustream 16

1.4.3 Weighting the Micro-Clusters 23

1.5 Experimentation 29

1.5.1 Methodology 30

1.5.2 Performance Measures 31

1.5.3 Simulated Results 33

1.5.4 Real texture data 37

1.5.5 Real Pendigit data 38

1.5.6 Evolving Pendigits 40

1.6 Conclusion 47

Bibliography **48**

Chapter 1

Spectral Clustering for Data Streams

1.1 Introduction

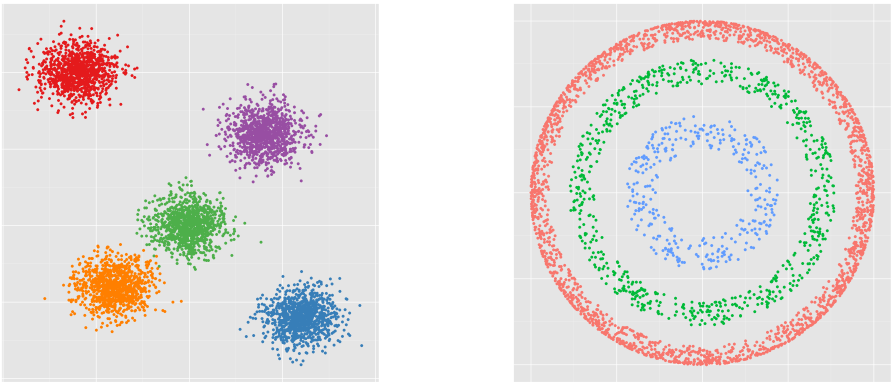
1.2 Spectral Clustering Background

In this section we motivate spectral clustering and introduce the spectral clustering algorithm by first noting the link between spectral clustering and graph partitioning problems.

1.2.1 Motivation

As discussed in Section 1.1 the goal of clustering algorithms is to partition data $X = \{x_1, \ldots, x_n\}, x_i \in \mathbb{R}^d$, into k disjoint classes such that each x_i belongs to exactly one class. Data sets can have underlying true clusters of all shapes and sizes, they can be spherical and convex as in Figure 1.2.1a or connected but non-convex as in Figure 1.2.1b.

Data which is convex may be simple to cluster as the gaps between clusters are easy for simple clustering algorithms like k-means to identify. Connected but non-convex data sets



(a) Convex clusters (b) Non-convex clusters

Figure 1.2.1: Examples of different types of clusters

can be much more challenging than convex data sets, and can cause some simple clustering algorithms to fail. The reason that centroid based clustering algorithms such as k-means struggle with data sets like that shown in Figure 1.2.1b is that k-means clusters the data based on how similar they are to cluster centroids.

Spectral clustering clusters data based on how *similar* they are to all other data points, which can lead to good quality segmentation on even these difficult cases. We do not formally address what is meant by similarity here, but will define this fully in Section 1.2.3.

The similarity between data points can be neatly represented in a *similarity graph*. We can then restate the clustering problem as a graph partitioning problem where we wish to find a partition of the graph such that the edges between different groups have low weights (which corresponds to data points in a cluster being dissimilar) and the edges within a group have high weights (the data points are similar).

In order to introduce spectral clustering we first discuss introduce some graph notation and discuss graph cut problems. We will then describe the spectral clustering algorithm, and

discuss in more detail the notion of similarity.

1.2.2 Graph cut problems

Data can be represented as a similarity graph, $G = (V, E)$ where each vertex $v_i \in V$ represents a data point x_i. The graph can then be described by an *adjacency matrix*. Adjacency matrices are a way of depicting the graph structure, showing which vertexes are connected by edges and which are not by creating a binary $n \times n$ matrix. Figure 1.2.2 depicts two similarity graphs. Their corresponding adjacency matrices are given in equation (1.2.1). A value of 1 in cell (2,3) implies that vertexes v_2 and v_3 are connected by an edge. Note that both of the example adjacency matrices given below are symmetric, which can be expected as we are dealing with undirected graphs.

Figure 1.2.2: Two similarity graphs

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{1.2.1}$$

The *weighted adjacency matrix* of a similarity graph is the matrix $W = (w_{ij})_{i,j=1,\ldots,n}$. The weight w_{ij} is the similarity between vertexes v_i and v_j. If $w_{ij} = 0$ this means that the vertexes v_i and v_j are not connected by an edge. Again the weighted adjacency matrix, which is also

called an *affinity matrix* will be symmetric $w_{ij} = w_{ji}$.

In order to create a graph partition we need to cut the edges in the graph. Non empty subsets of V, A and B will form a partition of the graph G if $A \cap B = \emptyset$ and $A \cup B = V$. The weight of the cut can be calculated by summing the weights of the edges which will be broken when a cut is made. In order to find a good partition of the graph, we wish to choose A and B such that some cut criterion is minimised. The simplest cut criterion is

$$\text{cut(A,B)} = \sum_{i \in A, j \in B} w_{ij}, \tag{1.2.2}$$

where the notation $i \in A$ is short hand to mean the set of indexes $\{i | v_i \in A\}$.

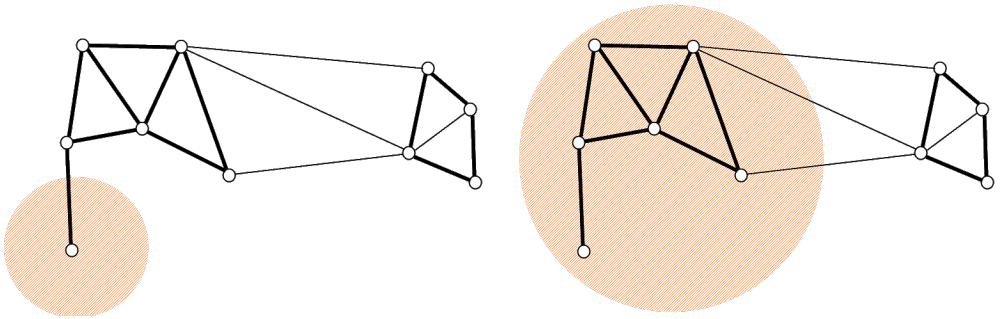
The Minimum cut (Wu and Leahy, 1993) is the cut which minimises equation (1.2.2). This is fairly easy to solve (Stoer and Wagner, 1997) however the minimum cut does not always produce a desirable graph partitioning; it tends to create unbalanced partitions, separating one vertex from the rest of the graph. To understand why this happens, note that the number of edges cut in mincut will be $|A| \times |B|$ which is minimised by the solutions $|A| = 1$ or $|B| = 1$. In order to avoid this, we can specify that the sets A and B are reasonably large in some way. Two common objective functions used to avoid this issue are the RatioCut (Hagen and Kahng, 1992) and the normalised cut, Ncut (Shi and Malik, 2000).

Both RatioCut and Ncut attempt to normalise the weight of the cut by introducing the size of sets A and B. In RatioCut, the size of A is measured by its number of vertexes $|A|$, while in Ncut the size is measured by the weights of its edges $\text{vol}(A) = \sum_{i \in A} d_i$ where $d_i = \sum_{j=1}^{n} w_{ij}$ is the *degree* of a vertex $v_i \in V$. The definitions of RatioCut and Ncut are as follows,

$$\text{RatioCut(A,B)} = \frac{\text{cut}(A,B)}{|A|} + \frac{\text{cut}(A,B)}{|B|}, \tag{1.2.3}$$

$$\text{Ncut(A,B)} = \frac{\text{cut}(A,B)}{\text{vol}(A)} + \frac{\text{cut}(A,B)}{\text{vol}(B)}. \tag{1.2.4}$$

The main idea in Ncut is that large clusters will increase the denominator $\text{vol}(A)$ and thus decrease $\text{Ncut}(A,B)$. This will encourage splitting the data into fairly evenly sized clusters, and avoid the minimum cut issue of segmented isolated points. This can be seen in Figure 1.2.3 which depicts both the minimum cut and Ncut solutions for a particular graph. The shaded/non-shaded regions represent the partitioning. The minimum cut isolates one vertex from the rest of the graph, whilst the Ncut provides a more balanced and sensible partition.



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

Figure 1.2.3: Two solutions to the bi-partition problem. The partitioning is indicated by shading/non shading of nodes.

Although the partitioning has been improved, the previously easy to solve mincut problem has been replaced with minimising the normalised cut which is an NP-hard problem (Wagner and Wagner, 1993). Spectral Clustering is a way to solve relaxed versions of these problems von Luxburg et al. (2008). It has been shown that the two graph cut optimisations given

in equations (1.2.3), (1.2.4) can be formulated in terms of the spectral decomposition of the graph Laplacian matrices given in equations (1.2.5) and (1.2.6).

$$L = \quad D - W \tag{1.2.5}$$

$$L_{\text{symm}} = \quad D^{-1/2} L D^{-1/2} \tag{1.2.6}$$

Here W is the affinity matrix of the similarity graph, The degree matrix D is defined as the diagonal matrix with the degrees $d_1, \dots d_n$ on the diagonal. The L Laplacian relates to a relaxation of minimising the RatioCut and L_{symm} relates to the relaxation of the Ncut (Chung, 1997). There is no guarantee on the quality of the solution of the relaxed problem compared to the exact solution. There do exist pathological cases which are arbitrarily bad. However, several papers which investigate the quality of the clustering of spectral clustering Spielman and Teng (1996) and Kannan et al. (2004) find spectral clustering to provide good solutions, and is much easier to solve.

The spectral clustering algorithm that we use (Shi and Malik, 2000) uses the symmetric Laplacian L_{symm}. The full spectral clustering algorithm is given in Algorithm 1.

Once the Laplacian has been calculated, one computes the k eigenvectors which correspond to the k smallest eigenvalues of the Laplacian. A matrix $Y \in \mathbb{R}^{n \times k}$ is created, where each column is an eigenvector of the Laplacian, with length n. We can view this matrix Y as an embedding of the original data X into a lower dimensional subspace. When represented in this low subspace the clustering problem is often easier, and can be solved with a simple clustering algorithm such as k-means. Apply k-means to cluster the n rows of Y into k clusters. Finally, to assign cluster labels to the original data, one assigns x_i to the cluster

Algorithm 1 NJW spectral clustering algorithm

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

Output: k-way partition of the input data

1: Construct the affinity matrix $W = (w_{ij})_{i,j=1,\ldots,n}$

2: Compute the normalised Laplacian matrix $L_{\text{symm}} = D^{-1/2}(D - W)D^{-1/2}$, where D is the diagonal matrix with $D_{ii} = \sum_{j=1}^{n} w_{ij}$.

3: Compute the k eigenvectors of L_{symm}, $v_1, v_2, \ldots, v_k$, associated with the k smallest eigenvalues, and form the matrix $Y = [v_1, v_2, \ldots, v_k]$.

4: Renormalise each row of V to form a new matrix Y.

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

6: Assign the original data point x_i to the cluster l $(1 \leq l \leq k)$ $\iff$ the corresponding row i of the matrix Y is assigned to the cluster l.

that row Y_i is assigned to in the k-means step.

Now that we have introduced the general Spectral Clustering algorithm, we will discuss the notion of similarity in more detail.

1.2.3 Choice of affinity matrix

One of the key factors of Spectral Clustering is the affinity matrix $W = (w_{ij})_{i,j=1,\ldots,n}$ which represents the pairwise similarities or distances between all data points x_i and x_j. A popular choice is to use the Gaussian kernel,

$$w_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right),\ i,j = 1,\ldots,n, \tag{1.2.7}$$

where the parameter σ controls the width of the local neighbourhoods which we want to model. If x_i and x_j are very close, then $w_{ij} \to 1$, and if they are far apart $w_{ij} \to 0$. A Gaussian kernel affinity matrix will have ones along the diagonal and is symmetric ($w_{ij} = w_{ji}$).

The scaling parameter, σ, is usually chosen manually. Ng et al. (2001) automatically choose 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 its neighbourhood. Using a localised σ_i can deal well with multi-scale data, but requires the user to choose the size of the neighbourhood to calculate σ_i.

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 (von Luxburg et al., 2008) to build the affinity matrix once corrections have been made to ensure that this matrix is symmetric. 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 so small that weighting may become eligible. Therefore a simple construction of the ϵ-neighbourhood graph is as shown in equation (1.2.8).

$$w_{ij}^{*} = \begin{cases} 1, & \text{if } w_{ij} < \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{1.2.8}$$

Using this construction will give a sparse affinity matrix instead of a fully connected graph, which will help lower the computational complexity.

1.3 Advanced Spectral Clustering

In the previous section we introduced spectral clustering via graph partitioning and discussed options for creating affinity matrices. In this section we consider settings where standard spectral clustering cannot be applied, large data sets, and data sets to which new data is added incrementally. An even more challenging setting, data streams, will be addressed in Section 1.4. RD: Sign post more large datasets, incremental and data streaming justifying why these are stepping stones towards data streaming.

1.3.1 Large-scale Spectral Clustering

Although spectral clustering has been shown to perform well empirically on simple data sets, computational problems arise as the data set size increases. Spectral clustering can be challenging for very large data sets since constructing the affinity matrix W and computing the eigenvectors of L have computational complexity $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ respectively.

The Nyström method (Williams and Seeger, 2001) is a general method for generating good quality low rank approximations of large matrices. The Nyström approximation method for spectral clustering (Fowlkes et al., 2004) randomly samples the columns of the affinity matrix W and approximates the eigen decomposition of the full matrix directly using correlations between the sampled columns and the remaining columns. Effectively this can be thought of as a dial which the user has control over, sampling more columns will provide better results but at a higher computational cost. The downsides with this method are that there are no accuracy guarantees, the memory requirements can be high, and the random sampling of columns may lead to small clusters being under represented or completely missed in the final clustering.

An alternative to the Nyström method is to use a pre-processing technique to reduce the size of the data. A natural way to do this is to select certain representative points to summarise the whole data set. Yan et al. (2009) proposed KASP and RASP, algorithms which use k-means and random forest methods respectively to select q representative points to apply spectral clustering on. Similarly Shinnou and Sasaki (2008) also use k-means to identify representative points. In addition to using these points, the Spectral Clustering also uses any data points which are deemed to be suitably far from any representative point.

Both Yan et al. (2009) and Shinnou and Sasaki (2008) the cluster labels given to the original data points are the same as the label assigned to their nearest representative point. As an alternative, Chen and Cai (2011) represents the data as a linear combination of representative points. A sequential reduction algorithm is adopted in Chen et al. (2006); Liu et al. (2007) with an early stopping criteria based on the observation that well separated data points converge to the final embedding more quickly. However this is only suitable for binary clustering. Other possibilities include random projection with sampling methods (Sakai and Imiya, 2009) and shortest path methods (Liu et al., 2013).

We discuss the KASP algorithm in more detail as it is the most popular speed up method for spectral clustering and it inspired our work in online spectral clustering which is introduced in Section 1.4.

In KASP, k-means is applied with q clusters to the data set X, where q is chosen such that $k \ll q \ll n$. Therefore each point in X belongs to a cluster $y_j, (j \in 1, \ldots, q)$. Let the centres of these q clusters be $\widehat{y_1}, \ldots, \widehat{y_q}$. These are used as representative points for the whole data set. Spectral clustering is performed on the representative points, reducing the complexity of the eigendecomposition from $\mathcal{O}(n^3)$ to $\mathcal{O}(q^3)$. Finally, the original data points are assigned the cluster label that their closest representative point $\widehat{y_j}$ was assigned in the spectral clustering. The KASP algorithm is given in Algorithm 2.

Both KASP and RASP have been shown to perform well empirically on large data sets (Yan et al., 2009), retaining good clustering performance even as the *data reduction ratio* increases. We can express the data reduction ratio as $\gamma = \frac{n}{q}$. As in many of the sampling methods discussed above, in KASP the user has control over the data reduction rate. A larger value of q will give a better performance but at a computational cost. The KASP

Algorithm 2 KASP

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

Output: k-way partition of the input data

1: Perform k-means with q clusters on $x_1, \ldots, x_n$ to create clusters $y_1, \ldots y_q$.

2: Compute the cluster centroids $\widehat{y_1}, \ldots, \widehat{y_q}$ as the q representative points.

3: Build a correspondence table to associate each x_i with the nearest cluster centroids $\widehat{y_j}$.

4: Run a spectral clustering algorithm on $\widehat{y_1}, \ldots, \widehat{y_q}$ to obtain an k-way cluster membership for each of $\widehat{y_j}, (j \in 1 \ldots q)$.

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

authors present an upper bound on the misclustering rate given the perturbation to the original data. However as the bound depends on eigengaps and the misclustering rate is bound between 0 and 1, this upper bound can be weak. Finally, the method of assigning data points to clusters based on the cluster label of their representative point can lead to poor segmentation as shown in Cao et al. (2014). They propose a local interpolation in their algorithm Local Information-based Fast Approximate Spectral Clustering (Li-ASP) to prevent this poor segmentation issue. They achieve this by assigning data points based on a weighted version of their p closest representative points labels, rather than labelling based just on the label of the single closest representative point.

The methods discussed above only address dealing with large data sets which are static. Our aim is to investigate methods which can update the spectral clustering partitioning when new data points arrive.

1.3.2 Incremental methods for Spectral Clustering

RD: Restructuring: shift any mentioning of datastreaming to the signposting at start of 1.3. Only mention incremental/evolving networks here - and explain exactly what I mean by that. A data stream is a potentially endless sequence of observations obtained at high frequency relative to the available processing and storage capabilities. Data streams arise in many applications such as online purchases, 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. Therefore there is a need for an online spectral clustering algorithm. To our knowledge, there is not currently a method for spectral clustering of data streams in a fully online manner. 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. So far there have been two different approaches to this problem (i) updating the cluster membership directly (ii) incrementally updating the eigenvectors.

The first method is described in Valgren and Lilienthal (2008). When new points arrive, the spectral clustering is updated directly using a similarity threshold to assign points to clusters. If a new data point is sufficiently far from its closest representative points, it is considered the start of a new cluster. This means that the number of overall clusters must always increase. Therefore it is not feasible for data streams. In addition there is no method for splitting clusters. RD: Clarify

An algorithm that incrementally updates the eigenvectors is proposed in Ning et al. (2007) and Ning et al. (2010). Their algorithm 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 - potentially leading to poor cluster detection. 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 reclustering 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. We did intend to use this algorithm as a competing algorithm in our experiments section. However, the computational costs for Ning were so great for data streaming examples that it was not possible to run the study, even when not performing a full reclustering.

Another incremental update algorithm is detailed in Dhanjal et al. (2011) Dhanjal et al. (2014) which approximates the eigen decomposition of the Laplacian incrementally but still requires regular full reclustering.

Kong et al. (2011) a mixture of both Ning and Valgren, using representative points like Valgren but the eigen updating of Ning. Although it can be quicker that Ning, it still has the same problem that the number of clusters will increase making it unsuitable for data streams.

As discussed, none of the above methods are suitable for clustering a data stream. We

consider an online spectral clustering algorithm for data streams based on the Clustream model of Aggarwal et al. (2003). The Clustream algorithm is introduced in full in Section 1.4.

1.4 Online Spectral Clustering

In this section we discuss general online clustering methods for data streams, 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 the data.

The first challenge is storing all of the data. As a data stream is a potentially endless sequence of observations obtained at a high frequency 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 which is a problem if we wish to incorporate historical data into the clustering.

Secondly, 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 in length, 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 as 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-clusters 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). $\boldsymbol{CF1^x_j}$ is the sum of all observed data in micro-cluster j, $\boldsymbol{CF2^x_j}$ is the sum of the squares of the data and n_j is the number of elements assigned to that micro-cluster. $CF1^t_j$ and $CF2^t_j$ refer to the sum of the time stamps, and the sum of squared time stamps respectively. Note that both $\boldsymbol{CF1^x_j}$ and $\boldsymbol{CF2^x_j}$ are d-dimensional vectors.

Each micro-cluster M_j will have

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

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.

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

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.

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

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)

$$RD : Deletionpolicy \tag{1.4.5}$$

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 (1.4.6) 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.

$$\begin{aligned} \boldsymbol{CF1^x_r} &\leftarrow \boldsymbol{CF1^x_r} + \boldsymbol{CF1^x_s} \,, \\ \boldsymbol{CF2^x_r} &\leftarrow \boldsymbol{CF2^x_r} + \boldsymbol{CF2^x_s} \,, \\ CF1^t_r &\leftarrow CF1^t_r + CF1^t_s \,, \\ CF2^t_r &\leftarrow CF2^t_r + CF2^t_s \,, \\ n_r &\leftarrow n_r + n_s \,. \end{aligned} \tag{1.4.6}$$

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

Algorithm 3 Clustream Micro clustering

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

Output: Micro clusters $M_1, \ldots, M_q$

Initialise the micro-clusters k-means$(x_1, \ldots x_{init}, q)$ 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 (1.4.6)

end if

end if

end for

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.

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 $M_1, \ldots, M_q$.

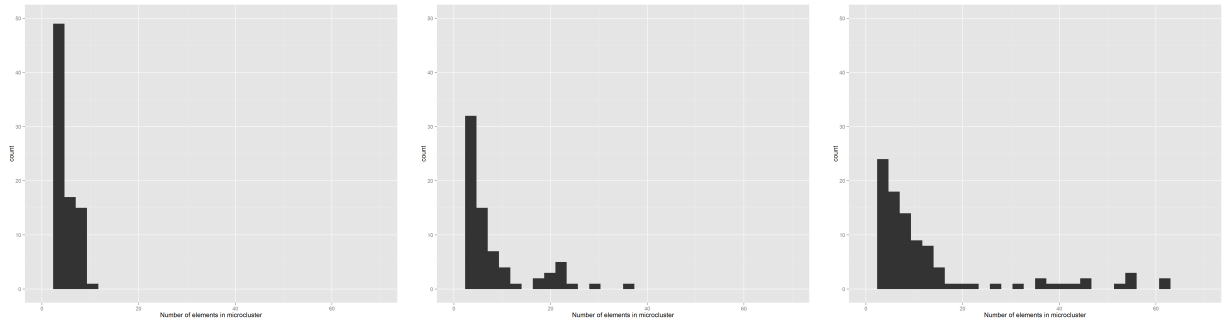
1: Initialise the micro-clusters using k-means$(x_1, \ldots x_{init}, q)$ and equations (1.4.1)
2: **for** each new data point x_i **do**
3: Apply Clustream update as in Algorithm 3
4: **if** A Macro-clustering is required **then**
5: Perform spectral clustering on $M_1, \ldots, M_q$ with k clusters.
6: **end if**
7: **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 using larger affinity consisting of repeated points.

Why would it be beneficial to weight the micro-clusters? For example Figure 1.4.1 shows how the distribution of points to micro clusters 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: Histograms showing the number of points assigned to micro clusters

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 consider the Affinity and Laplacian matrices more closely for a very simple case.

Assume that 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 $\triangle = n_1 s + n_2$.

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

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.

Let's now consider a more general example. 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 $\|u^*\|^2 = \sum_{i=1}^{n} (u_i^*)^2 = \sum_{i=1}^{k} n_i \cdot \frac{\tilde{u}_i^2}{n_i} = \sum_{i=1}^{k} \tilde{u}_i^2 = 1$.

First we recall the formula given in von Luxburg et al. (2008).

The normalised Laplacian has the following property. For every $f \in \mathbb{R}^n$ we have

$$f' L_{\text{symm}} f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2 . \tag{1.4.7}$$

We show that the second smallest eigenvector u of L is linked to the second smallest eigenvector $\tilde{u}$ of an enlarged Laplacian $\tilde{L}$ where points are repeated. In fact this will relationship will hold for all eigenvectors, but here we show the proof for the second smallest eigenvectors only.

In order to show this we need to show First we will show that

1. $\|u\| = 1$
2. $u \perp D^{1/2}\mathbb{1}$
3. $u^\top L u$
4. Part 4

Part 1

$$\begin{aligned}\|u\|^2 &= \sum_{i=1}^{n} u_i^2 \\ &= \sum_{i=1}^{k} n_i \frac{\tilde{u}_i^2}{n_i} \\ &= \sum_{i=1}^{k} \tilde{u}_i^2 = 1\end{aligned}$$

since $\tilde{u}$ is an eigenvector. Therefore $\|u\| = 1$.

Part2

$$\begin{aligned}\sum_{i=1}^{n} u_i D_{ii}^{1/2} &= \sum_{i=1}^{k} n_i \frac{\tilde{u}_i}{\sqrt{n_i}} \frac{\tilde{D}_{ii}^{1/2}}{\sqrt{n_i}} \\ &= \sum_{i=1}^{k} \tilde{u}_i \tilde{D}_{ii}^{1/2} = 0\end{aligned}$$

since $\tilde{u} \perp \tilde{D}^{1/2}\mathbb{1}$, therefore $u \perp D^{1/2}\mathbb{1}$.

Part 3

The first part arises from Von Luxburg.

$$\begin{aligned}\tilde{u}^\top \tilde{L} \tilde{u} &= \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} \left(\frac{\tilde{u}_i}{\sqrt{\tilde{D}_{ii}}} - \frac{\tilde{u}_j}{\sqrt{\tilde{D}_{jj}}} \right)^2 \tilde{A}_{ij} \\ &= \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} \left(\frac{\tilde{u}_i/\sqrt{n_i}}{\sqrt{D_{ii}}} - \frac{\tilde{u}_j/\sqrt{n_j}}{\sqrt{D_{jj}}} \right)^2 n_i n_j A'_{ij}\end{aligned}$$

which we can see is equal to $u'Lu$.

Part4

if $\exists\ v \perp D^{1/2}\mathbb{1}$ with $\|v\| = 1$ and $v^\top Lv < u^\top Lu$, then $\exists\ \tilde{v}$ with $\tilde{v}^\top \perp \tilde{D}^{1/2}\mathbb{1}$ and $\tilde{v}^T\tilde{L}\tilde{v} < \tilde{u}^\top\tilde{L}\tilde{u}$ which contradicts the fact that $\tilde{u}$ is the second smallest eigenvector of $\tilde{L}$.

Therefore u is the second smallest eigenvector of L.

RD: Sum up this section and signpost

1.5 Experimentation

In the experimentation section, we investigate the performance of both Spectral Clustream Unweighted and Weighted, along with a simple windowed approach. First the algorithms and methodology are introduced and the performance metrics defined. Then the algorithms are compared on simulated data, two real image based data sets and an evolving data set.

The Algorithms

Spectral Clustream is given in Algorithm 4. As described in Section 1.4, there are two ways we can incorporate the micro-cluster centres into the macro-clustering; not weighting, or weighting. 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 summarise the data stream.

Windowed Spectral Clustering is a simple algorithm which retains a window of only the w 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 $t \in 1, \ldots, n$. Spectral Clustream requires the micro-clusters to be initialised. This is achieved by applying k-means 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).

At each time point t, a new data point $\boldsymbol{x_t}$ is observed and is used to update the streaming algorithms. For Spectral Clustream, this means updating the micro-clusters as outlined in Algorithm 3. Windowed Spectral Clustering shifts the window along by one, forgetting the oldest data point $\boldsymbol{x_{(t-150)}}$ and including $\boldsymbol{x_t}$.

Every ten time steps we use the current snapshot of the data to apply spectral clustering and generate cluster labels for the data. In Spectral Clustream this is done by feeding the centres of the micro-clusters (weight adjusted or not) into a standard spectral clustering algorithm. In windowed spectral clustering, all w of the data points in the window as used as input into a standard spectral clustering algorithm.

We then evaluate the spectral clustering performance. The following step is repeated 10 times and the results are averaged out. Define the *test set* to be the next 200 data points in the data set($\boldsymbol{x_{t+1}}$, $\ldots \boldsymbol{x_{t+200}}$). 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 of the test data using the known

true clusters of the simulated data. Purity and V-measure are defined in the next section.

1.5.2 Performance Measures

The two measures that are used 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.

Let N be the number of data points, $C = \{c_i|1, \dots n\}$ be the set of true classes and $K = \{k_j|1, \dots m\}$ be the set of clusters assigned by the clustering algorithm. Define A to be the contingency table produced by the clustering algorithm representing the clustering solution such that a_{ij} is the number of points that are members of class c_i and assigned to cluster k_j.

Purity is the more intuitive of the two measures to understand as it quantifies how pure clusters are. A cluster which only contains data points associated with one true class will be given a high purity value. A cluster which consists of data points from many different true classes will receive a low purity value. Purity is calculated as follows. For each cluster find the true class which is most prevalent in that cluster and count how many data points of that class there 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).

$$\text{Purity} = \frac{1}{N} \sum_j \max_i |k_j \cap c_i|. \tag{1.5.1}$$

Generally this is a useful measure, however if we had a data set where each data point was assigned to a different cluster then the purity would be perfect even though this isn't a

particularly good clustering of the data. Purity can be unreliable if the number of clusters is much larger than the number of true classes.

To account for this we also use the V-measure (Rosenberg and Hirschberg, 2007) which takes the harmonic mean of two other performance measures, homogeneity and completeness. Homogeneity assesses if each cluster contains members of only a single class (in a similar way to purity), whilst completeness checks 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 as defined below in equation (1.5.3) and equation (1.5.6) respectively.

For homogeneity to be perfect, the clustering algorithm must assign only those data points that are members of a single class to a single cluster.

$$h = \begin{cases} 1 & \text{if } H(C|K) = 0 \\ 1 - \frac{H(C|K)}{H(C)}, & \text{else} \end{cases} \tag{1.5.3}$$

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}} \tag{1.5.4}$$

$$H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{N}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{N} \tag{1.5.5}$$

For perfect completeness the clustering must assign all data points which are members of a single class to a single cluster.

$$c = \begin{cases} 1 & \text{if } H(K|C) = 0 \\ 1 - \frac{H(K|C)}{H(K)}, & \text{else} \end{cases} \tag{1.5.6}$$

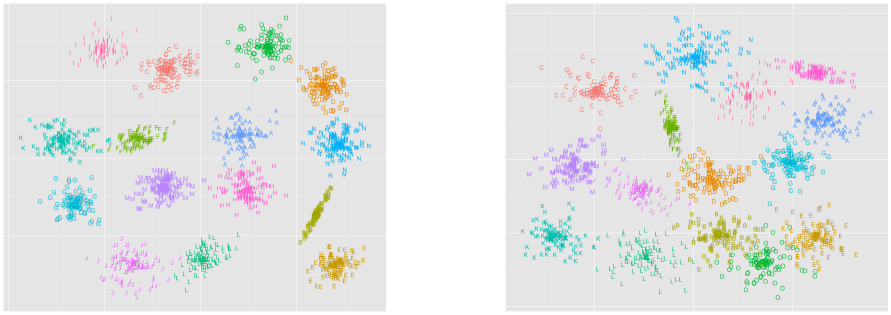
$$H(K|C) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{k=1}^{|K|}a_{ck}} \tag{1.5.7}$$

$$H(K) = -\sum_{k=1}^{|K|}\frac{\sum_{c=1}^{|C|}a_{ck}}{N}\log\frac{\sum_{c=1}^{|C|}a_{ck}}{N} \tag{1.5.8}$$

Both purity and V-measure are bound between 0 and 1, where 1 indicates perfect performance.

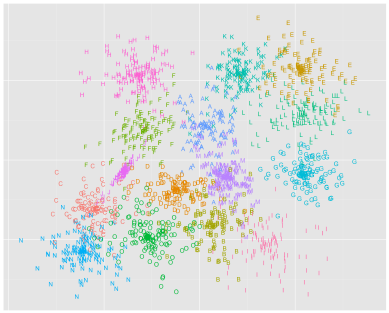
1.5.3 Simulated Results

The first data sets tested are the popular S-sets, first introduced in Fränti and Virmajoki (2006). The sets consist of synthetic 2d data with n=5000 vectors and k=15 Gaussian clusters with different degrees of cluster overlapping. The four data sets are shown in Figure 1.5.1. Set S1 is fairly easy to cluster as all 15 clusters are well separated. The sets become increasingly more challenging and S4 can be difficult even for humans to separate correctly.

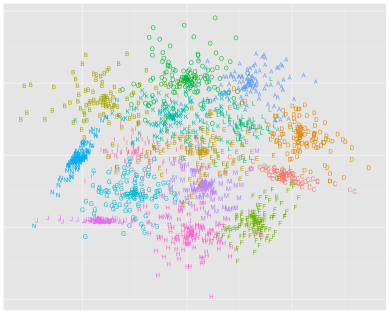


(a) S1

(b) S2



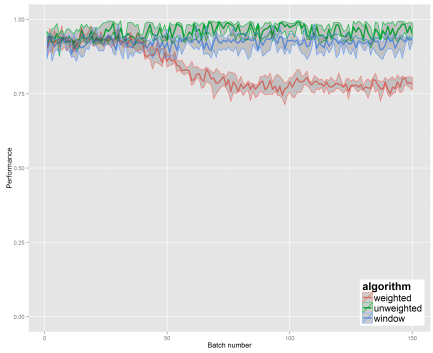
(c) S3



(d) S4

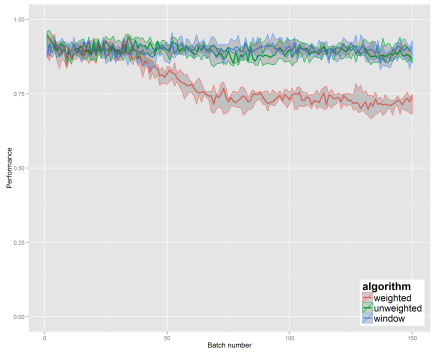
Figure 1.5.1: The S sets with true labels shown

The results from these sets are shown in Figure 1.5.2. The data sets S1, . . . , S4 are shown in the rows. The first column shows purity and the second column shows V-measure. The x-axis shows the batch number and therefore we can see the performance changing over time. However these data sets are static so we would not expect to see performance change with time. The shaded area depicts the inter quartile range. We can see that the unweighted Spectral Clustream (green) and windowed Spectral (blue) perform similarly for all sets. As expected, performance of all algorithms deteriorate as the data sets become harder, with more cluster overlap. The Weighted Spectral Clustream (red) initially starts with performance on par with the competing algorithms, but quickly drops to poor performance and does not recover. Given that the S sets are static, this behaviour is unusual.

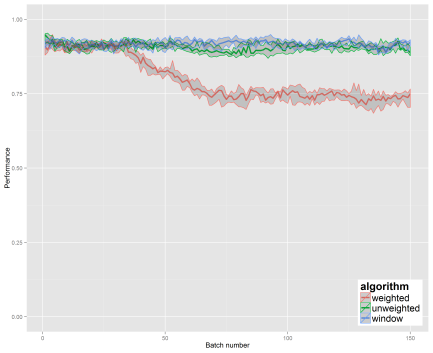


(a) S1 Purity

(b) S1 V measure

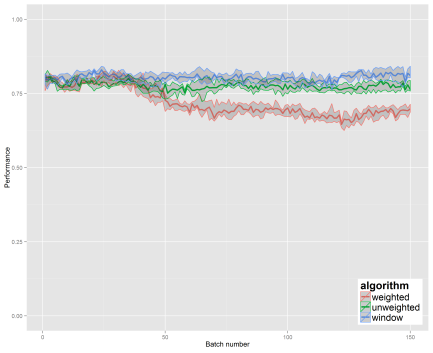


(c) S2 Purity

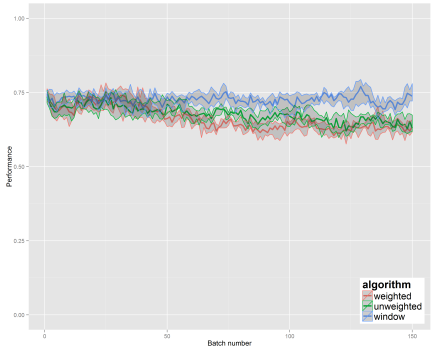


(d) S2 V measure

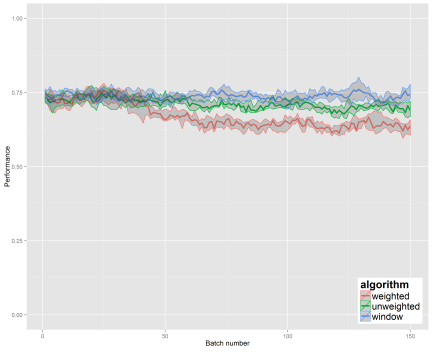
(e) S3 Purity



(f) S3 V measure



(g) S4 Purity



(h) S4 V measure

Figure 1.5.2: S set results

Figure 1.5.3 shows a snapshot of the Weighted Spectral Clustream algorithm on the S1 data set in the middle of the stream. The grey points are the data points observed so far, the location of the letters represent microcluster centres which are labelled with the results of the Weighted Spectral Clustering. We can see that one cluster (letter G) is dominating the points. Many of the outliers of the other clusters have been represented by the G cluster label. From observation we noted that once this behaviour of an "absorbing outlier cluster" began, the algorithm did not recover, hence the drop in performance seen in Figure 1.5.2.

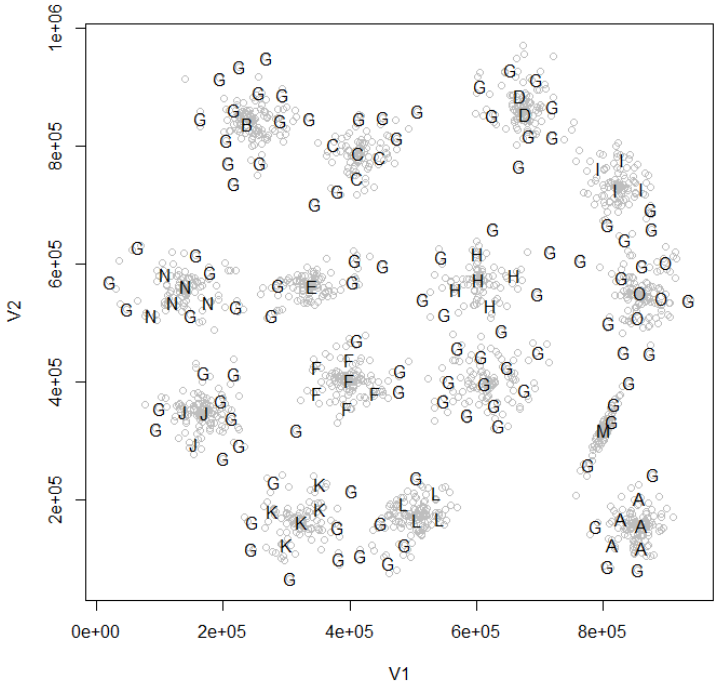


Figure 1.5.3: Snapshot from Weighted Spectral Clustream on S1

The use of the localised scaling parameter (See Section 1.2.3) used in the spectral clustering step may be interfering with the weighting of micro clusters. RD: Explain why. We did try using a global scaling parameter instead of the localised one, however this then brought up the issue of tuning the σ parameter, which is known to be very sensitive and is a diffi-

culty with Spectral Clustering algorithms in general (von Luxburg et al., 2008). Although performance did seem to improve with the global scaling parameter when chosen carefully, the performance was still very poor compared to Windowed Spectral Clustering and Spectral Clustream.

1.5.4 Real texture data

We now investigate the performance of the clustering algorithms on real data. This new data set consists of features extracted from textured images. The Kylberg texture data set (Kylberg, 2011) consists of 28 texture classes with 160 unique texture patches per class. The patches consist of 576×576 pixel images. Features for clustering were extracted using the LS2W (Eckley and Nason, 2011) which creates wavelet features from the textured images.

A subset of 6 classes were selected, examples of which are shown in Figure 1.5.4 . The classes selected are images of two types of blanket, some canvas, a ceiling, some lentils and a screen.

The performance plots for the texture data are shown in Figure 1.5.5. Here we do 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 declines as the stream progresses.

The poor performance of Weighted Spectral Clustream was observed on all other data sets investigated and therefore the results from this algorithm will be dropped from any further performance plots in order to focus more on the behaviour of the other two algorithms.

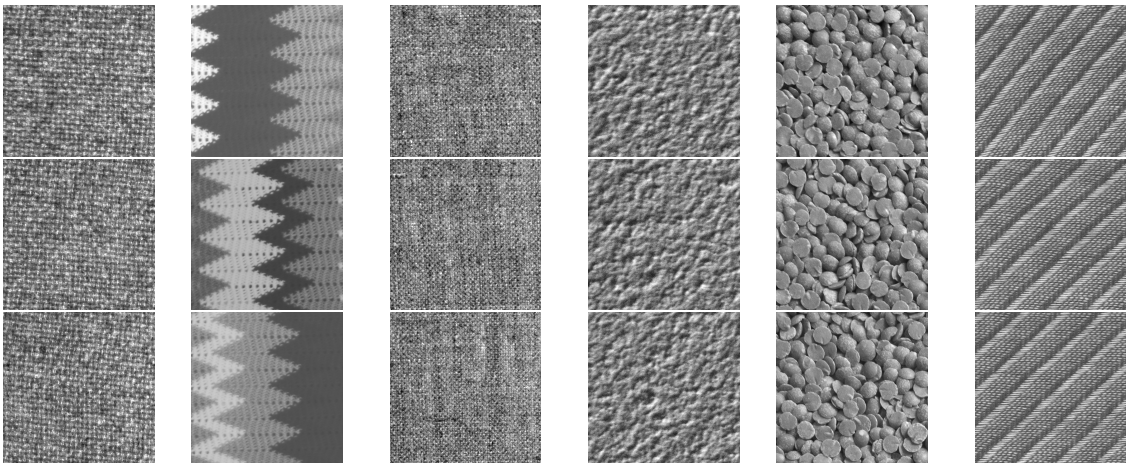
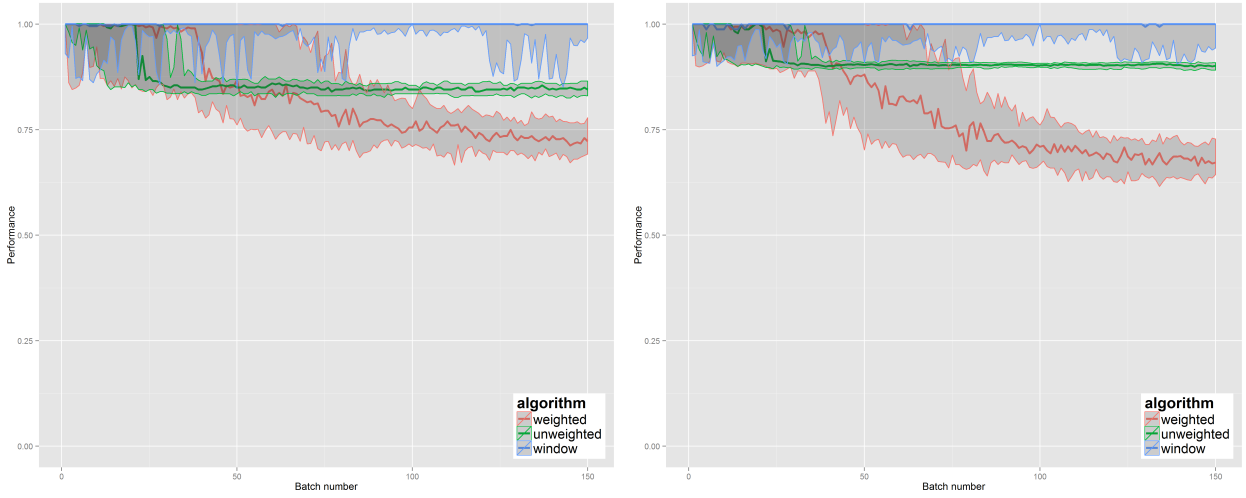


Figure 1.5.4: Three examples from each of the 6 different texture tiles used in the streaming data set. The texture classes are (L-R) Blanket 1, Blanket 2, Canvas, Ceiling, Lentils and Screen



(a) Texture Purity

(b) Texture V-measure

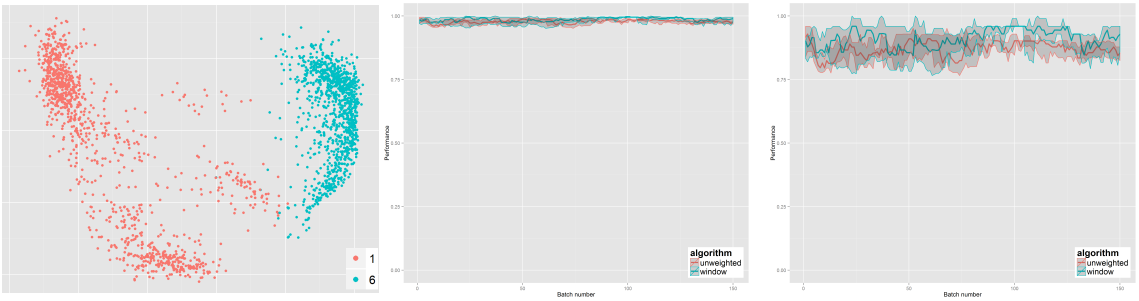
Figure 1.5.5: Texture Results

1.5.5 Real Pendigit data

This section and the next will use a the UCI Pendigit dataset which was introduced in Alimoglu and Alpaydin (1996) and is available is for download (Lichman, 2013). The data

set consists of 250 samples of hand drawn digits of the numbers 0-9 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.

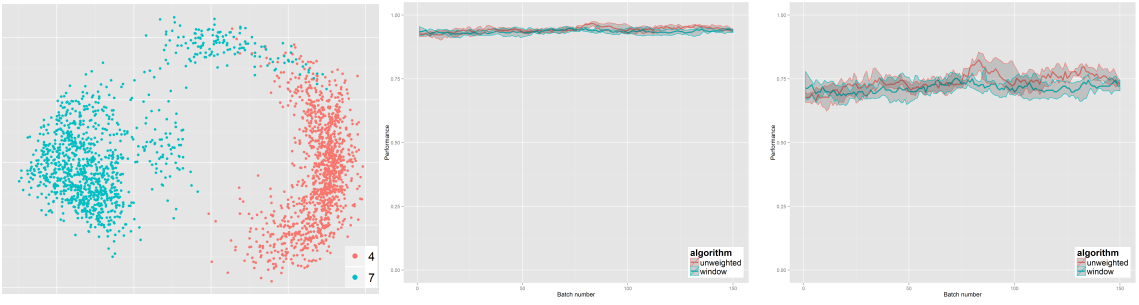
The results for a selection of the pairwise digits are shown in Figure 1.5.6. The first column displays the digit data in PCA space, the second column shows the purity, and the third column shows the V-measure. Both algorithms show similar performance in the plots shown (and in all the other pairwise combinations which were run). It can be noted that V-measure is lower than purity, which might imply that the resulting clusters are homogeneous but not complete (see Section 1.5.2).



(a) PCA of digits 1 and 6

(b) Purity digits 1 and 6

(c) V-measure digits 1 and 6



(d) PCA of digits 4 and 7

(e) Purity digits 4 and 7

(f) V-measure digits 4 and 7

Figure 1.5.6: Pendigits Pairwise - Spectral Clustream and Windowed Spectral Clustering

1.5.6 Evolving Pendigits

So far in this section we have considered only static data sets. The main challenge for clustering algorithms for data streams is adapting to changes in the data set. In order to create real data sets with a non stationary distribution we introduce a change into the data set. To construct a data set we choose 3 digits in the Pendigits data set (for example 4, 8 and 9). The start of the data stream consists only of 2 digits (4 and 8). Half way through the stream, we replace the second digit with the third digits (so now we observe values 4 and 9 rather than 4 and 8). By swapping the digits in this manner we can avoid the difficult issue of having to select a number of clusters, as we always observe only two clusters.

Due to the nature of algorithms with windowing, we would expect Windowed Spectral Clustering to adapt quickly to this change given the window is fairly small. The initial results showed Spectral Clustream performing poorly after the change, and struggling to recover. These results will be shown at the end of the section, and fix is suggested to correct the poor behaviour. First, in order to understand why Spectral Clustream was performing badly lets consider a specific example. The example set is "Evolving 48 49" - first we observe 4 and 8, and we switch to observing 4 and 9 half way through the stream. Figure 1.5.7 shows the PCA space for all three digits. There is quite a bit of overlap between digits 4 and 9, which makes the set quite tricky to cluster.

Figure 1.5.8a shows the Spectral Clustream at the start of the stream (directly after initialisation). The grey points show all data seen up until this time and the blue points indicate the next 200 points to be observed (the test set used for our performance measures.) The crosses indicate locations of microcluster centres, and their colour indicates which overall

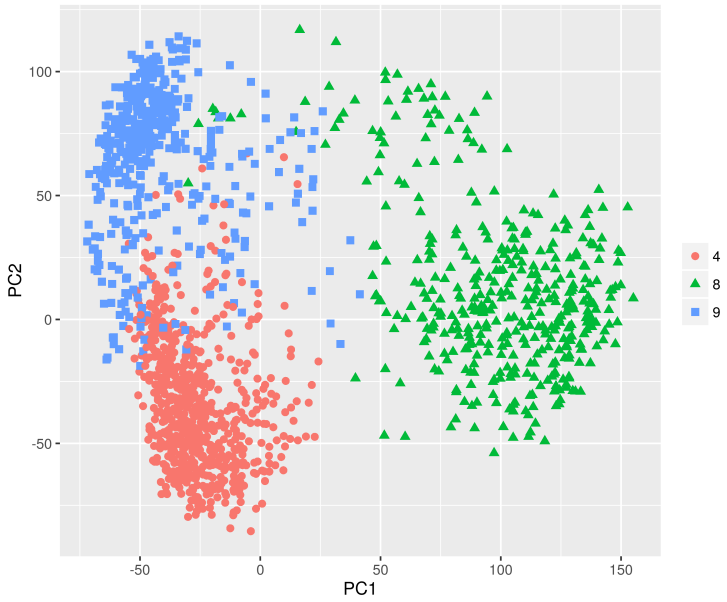


Figure 1.5.7: PCA plot for the Pendigits 4,8 and 9

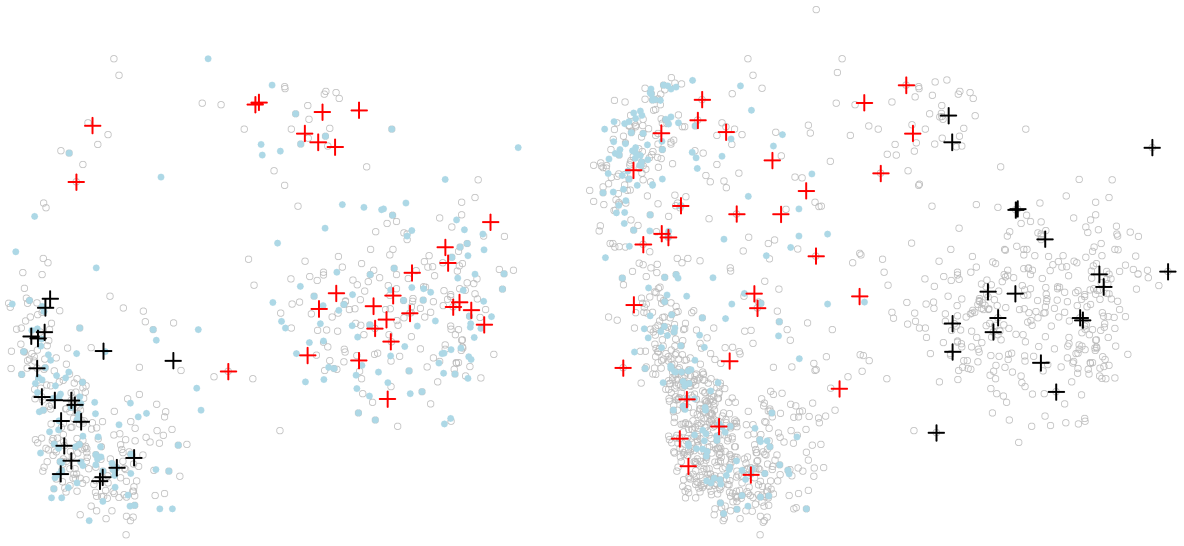
cluster they have been assigned to using the spectral clustering algorithm.

In Figure 1.5.8a, the microcluster centres are well distributed over the grey data points and also the blue test points. The spectral clustering mostly segments the microcluster centres correctly into the left and right clusters.

At time step 1000 (1.5.8b), we have begun observing the new cluster (digit 9) and are no longer receiving data about digit 8. This is shown as all of the test data points (light blue) are in either the bottom left corner (digit 4) and top left corner (digit 9). We see that the microcluster centres (crosses) are spread out over all the data including the defunct cluster of digit 8 (the grey data points on the right hand side of the plot).

It isn't necessarily a bad thing that there are some micro clusters in this region. It can inform us about the historical state of the stream. The reason that there are still microcluster centres located in the cluster 8 region is because of the deletion policy that Clustream uses.

Clustream requires the number of micro clusters to remain fixed for the duration of the



(a) Stream at time t = 1

(b) Stream at time t = 1000

Figure 1.5.8: Microcluster centres for the Evolving Pendigits 48, 49

stream. As discussed in Section 1.4, if an arriving data point does not have a suitable microcluster to merge into, a new micro-cluster is formed. However, since the total number of micro clusters is fixed, this means that we need to either delete an old microcluster if it is suitably old, or combine two close ones. This is the only way that micro clusters can be deleted in Clustream. A microcluster i is defined to be suitably old if it's relevancy $r(M_i)$ is less than the relevancy threshold δ, as defined and discussed in equation (1.4.5).

In practice it is difficult to select the best value of δ. If δ is set too high, Clustream will delete too often and therefore emerging new micro clusters may not be allowed to develop fully. If δ is set too small then old clusters will stay in the system much longer than required. This is an example of when δ is possibly too small, as the algorithm seems unwilling to delete old micro clusters.

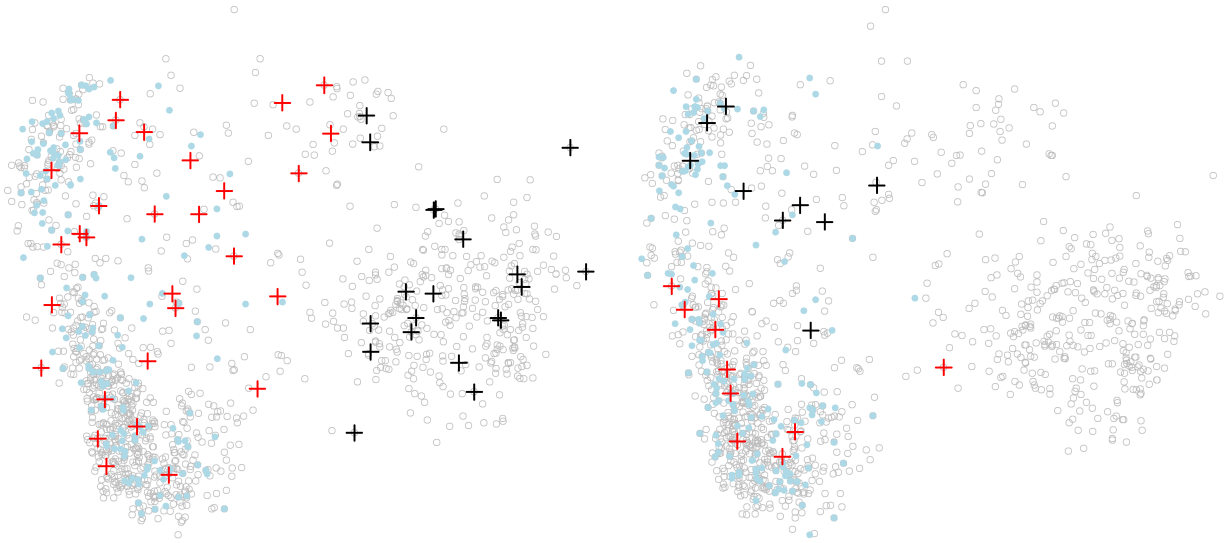
In itself having old microcluster centres remain isn't a problem. In fact this could be

useful. In the case where a a cluster disappears for some time and then re-emerges later in the stream, retaining old microcluster centres would be very useful, as the initial learning should be quick. These type of scenarios can occur regularly in any sort of cyclic data, such as any shopping data which has seasonality.

The problem arises when the centres are used in the spectral clustering. By including these old centres in the spectral clustering, the algorithm technically has centres from three (digits 4,8 and 9) clusters, but we have asked the spectral clustering algorithm to find two clusters. In the example above in Figure 1.5.8b, we see that the two clusters on the left that we are trying to separate are grouped together as one, because of the inclusion of the old microcluster centres on the right of the plot.

The proposed solution to deal with these microcluster centres is as follows. Before the macro clustering step is complete, identify the micro clusters of interest. In this setting, we find the micro clusters which the test data are closest to. Then use only these relevant microcluster centres to perform the spectral clustering. Figure 1.5.9 compares the previous method with the proposed alteration. Figure 1.5.9a shows the standard algorithm at t = 1000 (this is a duplicate of Figure 1.5.8b repeated for comparison purposes). Figure 1.5.9b shows the clustering of the microcluster centres when the Alternative Spectral Clustream is used.

We can see that the older microcluster centres are no longer used in the spectral clustering, and therefore the algorithm is better able to distinguish between the digit 4 and the digit 9. Note that although the old microcluster centres are not shown in the figure, they are technically still stored, but since they are not used in the macro-clustering stage they do not receive a cluster label therefore are not shown on the plot.



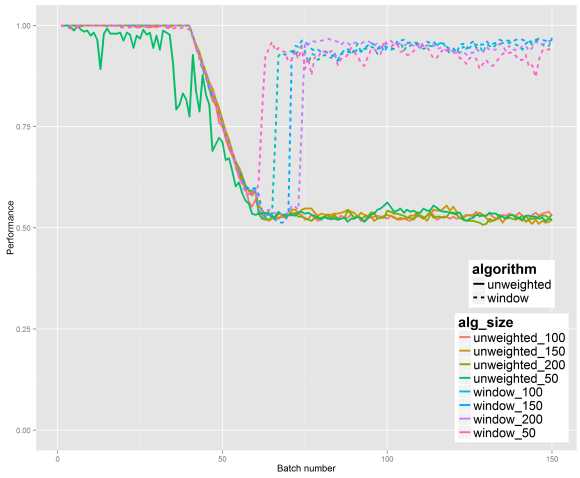
(a) Standard Spectral Clustream Algorithm (b) Alternative Spectral Clustream Algorithm

Figure 1.5.9: Microcluster centres for the Evolving Pendigits 48, 49 at t = 1000

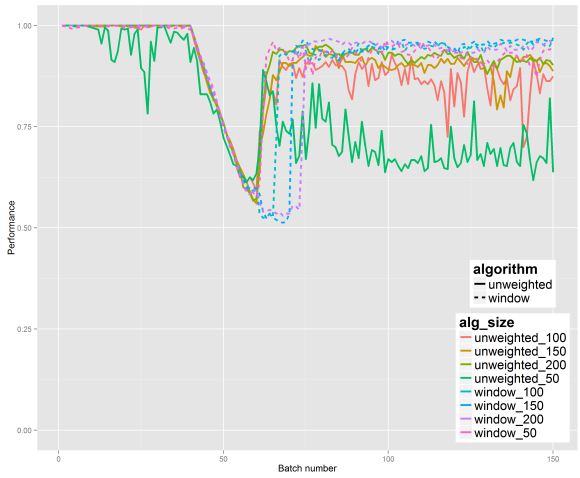
Once obvious issue with this amendment is there may be fewer input data points into the spectral clustering algorithm - this means that the number of micro clusters used becomes more important. In the illustrative example above we initially used 50 micro clusters to represent the stream. This is already fairly small, but given that some centres are now not used in the spectral clustering step, this can be an issue. In fact at time step 1000 (Figure 1.5.9b), we can see that only 19 of the 50 micro clusters are being used in the spectral clustering algorithm. Therefore there may be a need when using this alternative method to select the number of micro-clusters to be larger then required.

Figures 1.5.10 and 1.5.11 show the alternative Spectral Clustream performance with the standard Spectral Clustream performance. We have included a number of different values for the number of micro clusters (50, 100, 150, 200) and also ran Windowed Spectral with window sizes w = 50,100,150, 200. In the plots the dashed lines show the Windowed Spectral Clus-

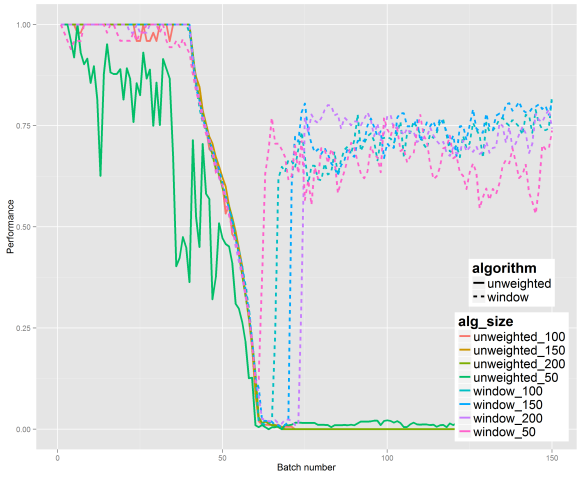
tering and the full lines are Unweighted Spectral Clustering. The colours indicate different values for the number of micro clusters/window size. The plots on the left show the performance for the Standard Unweighted Spectral Clustering, and the plots on the right show the proposed Alternative Unweighted Spectral Clustering. The performance of Windowed Spectral Clustering is repeated in both left and right plots for comparison purposes.



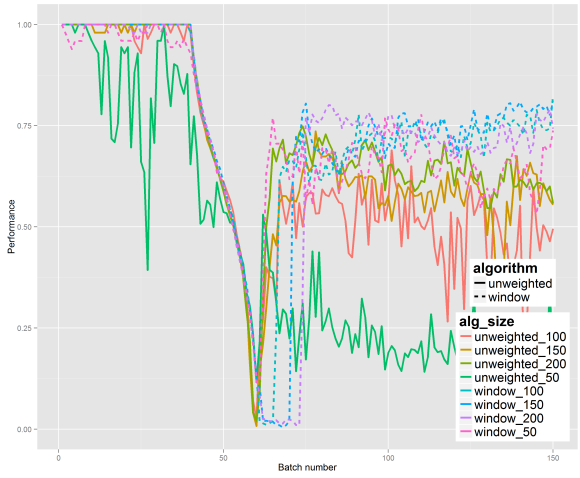
(a) Purity - Standard



(b) Purity - Alternative



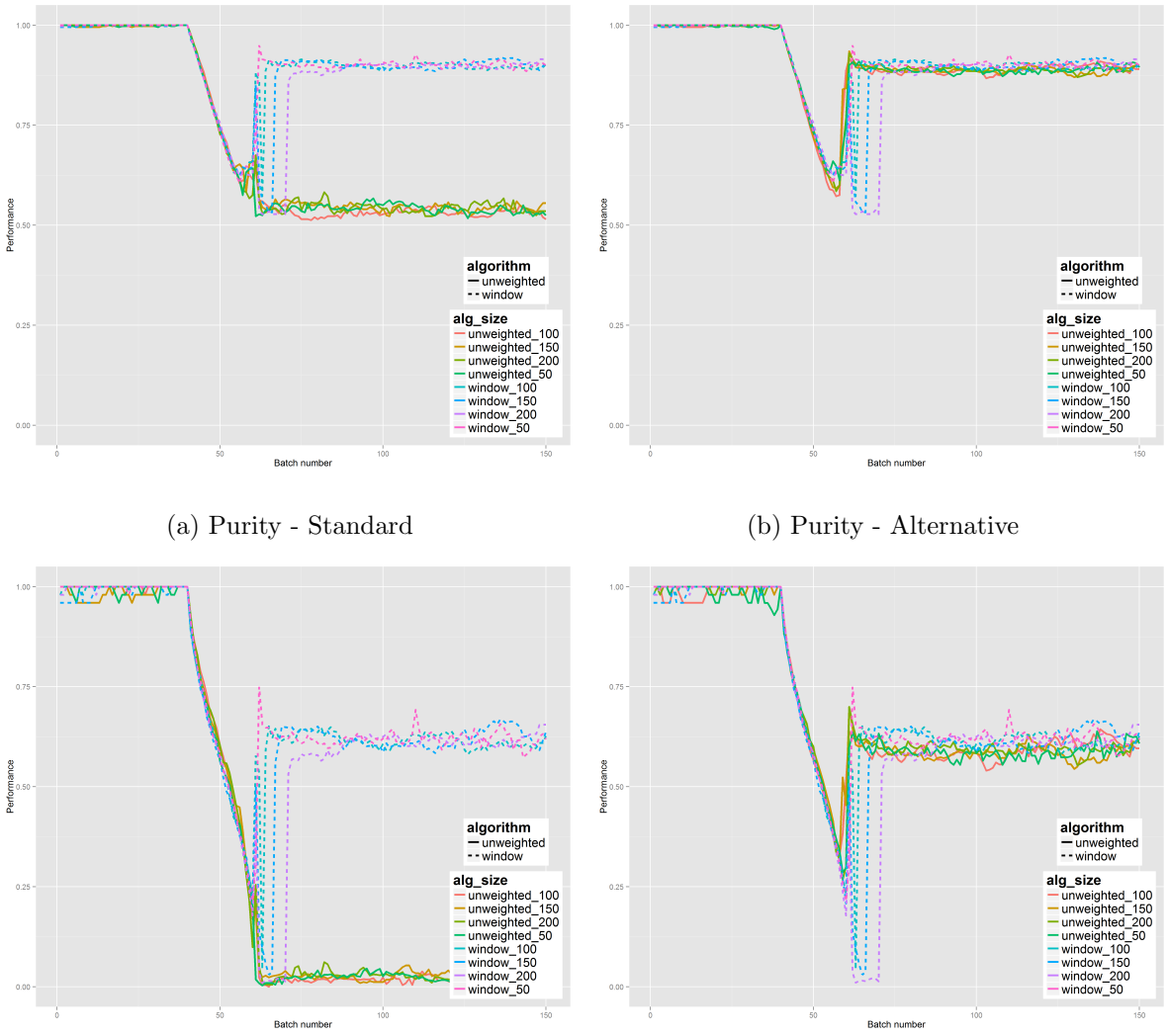
(c) Vmeasure - Standard



(d) Vmeasure - Alternative

Figure 1.5.10: Standard vs Alternative Clustream on Evolving Pendigits 48 49

The plots show that although Unweighted Spectral Clustream was performing poorly



(a) Purity - Standard

(b) Purity - Alternative

(c) Vmeasure - Standard

(d) Vmeasure - Alternative

Figure 1.5.11: Standard vs Alternative Clustream on Evolving Pendigits 34 37

when using the standard approach, using the alternative method has brought performance back up to being as good as Windowed Spectral Clustering. The importance of the number of micro clusters is evident in this change set. In Figure 1.5.10d we can clearly see the number of micro clusters affects the performance of the algorithm. Initially the number of micro clusters doesn't have a large effect on performance but after the change occurs, using 50 micro clusters was not sufficient to the algorithm to adapt. However using 200 micro clusters

brought performance up to a reasonable level. We also observe the effect of window size on performance. In Figure 1.5.11b it is clear at the point of change that the window with the smallest size (50) recovers first, with the largest window size taking the longest to adapt to the change.

1.6 Conclusion

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.

Alimoglu, F. and Alpaydin, E. (1996). Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwritten Digit Recognition. *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium*.

Bouchachia, A. and Prossegger, M. (2012). Incremental spectral clustering. In *Learning in Non-Stationary Environments: Methods and Applications*, volume 9781441980, pages 77–99. Springer New York, New York, NY.

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.

Chen, B., Gao, B., Liu, T.-Y., Chen, Y.-F., and Ma, W.-Y. (2006). Fast spectral clustering of data using sequential matrix compression. *Machine Learning: ECML 2006*, pages 590–597.

Chen, X. and Cai, D. (2011). Large Scale Spectral Clustering with Landmark-Based Rep-

resentation. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence Large*, pages 313–318.

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.

Dhanjal, C., Gaudel, R., and Clémençon, S. (2014). Efficient eigen-updating for spectral graph clustering. *Neurocomputing*, 131:440–452.

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*.

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.

Fränti, P. and Virmajoki, O. (2006). Iterative shrinking method for clustering problems. *Pattern Recognition*, 39(5):761–775.

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

Hagen, L. and Kahng, A. (1992). New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085.

Kannan, R., Vempala, S., and Vetta, A. (2004). On Clusterings: Good, Bad and Spectral. *Journal of the ACM*, 51(3):497–515.

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.

Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Liu, J., Wang, C., Danilevsky, M., and Han, J. (2013). Large-scale spectral clustering on graphs. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1486–1492.

Liu, T.-Y., Yang, H.-Y., Zheng, X., Qin, T., and Ma, W.-Y. (2007). Fast large-scale spectral clustering by sequential shrinkage optimization. In *Proceedings of the 29th European conference on IR research*, pages 319–330, Berlin, Heidelberg. Springer Berlin Heidelberg.

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.

Sakai, T. and Imiya, A. (2009). Fast spectral clustering with random projection and sampling. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, volume 5632 LNAI, pages 372–384.

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

Shinnou, H. and Sasaki, M. (2008). Spectral Clustering for a Large Data Set by Reducing the Similarity Matrix Size. *In proceedings of the Sixth International Language Resources and Evaluation(LREC08)*, pages 201–204.

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.

Spielman, D. A. and Teng, S.-H. (1996). Spectral partioning works: planar graphs and finite element meshes. In *37th Ann. Symp. on Found. of Comp. Science (FOCS)*, pages 96–105.

Stoer, M. and Wagner, F. (1997). A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591.

Valgren, C. and Lilienthal, A. (2008). Incremental spectral clustering and seasons: Appearance-based localization in outdoor environments. In *2008 IEEE International Conference on Robotics and Automation*, pages 1856–1861. IEEE.

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

Wagner, D. and Wagner, F. (1993). Between Min Cut and Graph Bisection. In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science*, pages 744–750. Springer-Verlag.

Williams, C. and Seeger, M. (2001). Using the Nyström 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.