Efficient Analysis of Data Streams

Rhian Natalie Davies

Submitted for the degree of Doctor of Philosophy

at Lancaster University.

September 2017

Abstract

Data streams provide a challenging environment for statistical analysis. Data points can arrive at a high velocity and may need to be deleted once they have been observed. Due to these restrictions, standard techniques may not be applicable to the data streaming scenario. This leads to the need for data summaries to represent the data stream. This thesis explores how data summaries can be used to perform clustering and classification on data streams across a broad range of applications.

Spectral clustering is one such technique which prior to this work has not been applicable to the data streaming setting due to the high computation involved. CluStream is an existing method which uses micro-clusters to summarise data streams. We present two algorithms which utilise these micro-cluster summaries to enable spectral clustering to be performed on data streams. The methods were tested on simulated data streams, as well as textured images and hand-written digits.

Distributed acoustic sensing is used to monitor oil flow at various depths throughout an oil well. Vibrations are recorded at very high resolutions, up to 10000 observations a second at each depth. Unfortunately, corruption can occur in the signal and engineers need to know where corruption occurs. We develop a method which treats the multiple time series as a high-dimensional clustering problem and uses the cluster labels to identify changes within the signal.

The final piece of work concerns identifying areas of activity within a video stream, in particular CCTV footage. It is more efficient if this classification stage is performed on a compressed version of the video stream. In order to reconstruct areas of activity in the original video a recovery algorithm is needed. We present a comparison of the performance of two recovery algorithms and identify an ideal range for the compression ratio.

Acknowledgements

I would like to thank my supervisors Dr Nicos Pavlidis and Professor Idris Eckley for their endless support, guidance and patience throughout this project. This final version of the thesis has been improved thanks to the helpful comments of my viva examiners, Dr Sotirios Tasoulis and Professor Kevin Glazebrook. Thanks are also due to Professor Lyudmila Mihaylova for her help with the conference paper presented in Chapter 4. The data analysed in Chapter 3 was kindly provided by Shell.

My PhD has been supported by the EPSRC funded Statistics and Operational Research (STOR-i) Centre for Doctoral Training. The STOR-i CDT has provided an excellent research environment and many additional training opportunities which have allowed me to develop my research skills. Particular thanks go to the STOR-i director Professor Jonathan Tawn for his constant support, especially during the late PhD stages.

I would also like to give special thanks to the '11-'15 cohort of STOR-i: Ben, Dave, Emma, Hugo, James, Judd, Rob and Tom. My PhD experience was greatly enhanced by their knowledge, encouragement and friendship.

Finally, I would like to thank my family, Geraint, Joan, Sian and Tom. Dach chi'n werth y byd.

Declaration

I declare that the work in this thesis has been done by myself and has not been submitted elsewhere for the award of any other degree.

I declare that the word count of this thesis is 23469 words.

Chapter 4 has been accepted for publication as R. Davies, L. Mihaylova, N. Pavlidis, and I. A. Eckley. The effect of recovery algorithms on compressive sensing background subtraction. In *Sensor Data Fusion: Trends, Solutions, Applications*, 1 - 6, 2013.

Rhian Davies

Contents

Chapter 1

Introduction

The volume of data collected on a daily basis is staggering. In 2013, IBM stated that over 90% of the world's data was created in the last two years. This creates a challenge for researchers and practitioners as computers may not have the memory requirements to deal with such large quantities of data, and their algorithms may not run fast enough or even at all. Big data (Buhlmann et al., 2016) is a term used to refer to data sets so huge that traditional statistical techniques may not be directly applicable.

The challenge of big data can be defined in terms of the three V's (Laney, 2001); volume, variety and velocity. Volume refers to the vast amount of data collected. Variety relates to the many different types of data it is possible to collect from multiple sources such as text, images, GPS location and tweets. The velocity is the speed at which the data is observed. Velocity is perhaps the least studied of the three V's within Statistics and is most commonly found in the analysis of data streams. A *data stream* (Aggarwal, 2007; Gama and Gaber, 2007) is data which is observed continuously in an ordered sequence. Data streams arise in many different applications. Examples include retail, e.g. Macy's use of advanced data collection (Shankar et al., 2016); oil and gas including the development of the digital oil field which uses sensors throughout oil wells to monitor flow and other operational characteristics (Cramer et al., 2008; Patri et al., 2012).

The rate at which data arrives could be as fast as millions of data points each hour such as in the Macy's and oil company examples. However, even if the data arrives more slowly, this can still provide a challenge if the available processing power, storage capabilities or transmission rates are limited.

It is possible for a data stream to be unbounded in size by which we mean that there is no time point where the data stream ends. This is common in data streams found in meteorology, the stock market, online shopping and social media. Since these data streams are potentially endless in size it is not possible to store the data in its entirety. This means that instead of performing analysis once the data has been collected, analysis must be performed and updated in real time as new data points are observed. This leads to another issue sometimes referred to as the one-pass-access problem. As data streams are processed serially, once a data point has been seen it is discarded and cannot be accessed again. Some seemingly trivial analyses such as computing the median of the data become impossible in the data stream setting because of this one-pass-access issue. In fact, many statistical techniques make assumptions which do not hold in the data streaming scenario. A summary of the restrictions imposed by data streams is given below (Silva et al., 2013):

1. Data objects arrive continuously.

2. There is no control over the order in which the data objects should be processed.

3. The size of the stream is potentially unbounded.

4. Data objects are discarded after they have been processed.

5. The unknown data generation process is possibly non-stationary.

Given the restrictions above, how can we perform analysis on a data stream? If the issue was just the size of the data then we could sample and perform analysis on that sample. However as the data stream cannot be stored, it is not possible to take a sample of the whole stream. We could sub sample as we go along based on the frequency storing every 100th data point. However, if the data stream is unbounded then eventually this method will fail.

What is needed is a *representative data summary* of the data stream that captures the new data points but still retains some historical information. An ideal representative data summary will:

1. Be computationally easy to update.

2. Store historical information.

3. Forget historical information if it becomes obsolete.

4. Adapt if the underlying generating process of the data stream changes.

5. Be informative enough to use in statistical analysis.

One possible representative data summary is a window of the most recently observed data points. A window is computationally easy to update as it uses a one-in-one-out update policy. However, the way that it deals with historical information is naive. Any data points outside the window are discarded, which makes the choice of window size particularly important. A small window will react to new data quickly but will have no older knowledge, whilst a large window will retain historical information but may be slow to update to a change in the data stream. This will naturally infer a temporal bias and a poorly selected window size may lead to missing important seasonal trends such as the effect of Christmas on shopping sales.

We discussed above how it is not possible to calculate the median of a data stream. It is however quite simple to keep a track of the running mean by storing the sum of the data points and the number of data points observed so far. This is another simple example of a representative data summary. Again it is computationally easy to update and all historical information can be retained. However, in order to perform clustering or regression on the data stream we will need a richer data summary than just a running mean.

Once a suitable representative data summary has been selected, this can then be used to perform analysis such as clustering. However standard techniques might need to be adapted to work on these data summaries as opposed to on the raw data. In this thesis we explore the use of representative data summaries on the analysis of data streams.

1.1 Thesis Outline

In Chapter 2 we consider the problem of identifying groups or clusters in a data stream. Many different types of clustering algorithms exist however, we restrict our interests to *spectral clustering*. Spectral clustering is popular, offers good empirical performance and can handle tricky, non-Gaussian data sets. However, due to it's complexity it cannot be performed on very large data sets and is not suitable for data streams.

CluStream (Aggarwal et al., 2003) is a popular algorithm used to handle data streams by creating and updating a representative data summary of the stream. We present a new algorithm *spectral CluStream*, which summarises data streams and performs spectral clustering on those data summaries. A spectral clustering algorithm which can cluster data streams does not currently exist in the literature. We consider both weighted and un-weighted variants of online spectral clustering. A number of different data sets are analysed including handwritten digit data and wavelet based texture features from an image data set.

Chapter 3 uses CluStream to identify corruption within acoustic sensing signals. Distributed acoustic sensing (DAS) is a modern technique used to monitor oil flow at various depths throughout an oil well. DAS uses a fibre-optic cable to record vibrations at very high resolutions, up to 10000 observations a second. DAS is fairly cheap to implement and offers high frequency data, but unfortunately corruption can occur in the signal. Our challenge is to identify the locations in the signal where corruption occurs. Existing methods for detecting and removing interference in DAS signals involve using offline, uni-variate changepoint detection. However DAS signals are multivariate and require online processing. We show that CluStream provides an alternative approach to changepoints analysis to identify corruption within DAS signals.

Chapter 4 considers a different challenge, *background subtraction*, which relates to the identification of activity (foreground) in surveillance videos. Current methods for this require storing each pixel of every video frame which can be wasteful as most of this information relates to the uninteresting background. *Compressive sensing*, a method for summarising data, can offer an efficient solution by using the fact that the foreground often only makes up a small proportion of an image. Using compressive sensing background subtraction it is possible to identify regions of foreground activity in a compressed image by applying a recovery algorithm. This chapter presents a study of the performance of two different recovery algorithms on CCTV surveillance footage.

All three chapters are linked by the challenge of gaining insight from data streams by analysing a simple summary of the stream and extrapolating the analysis to learn something about the data stream as a whole. Contributions of this thesis and observations on data stream analysis are gathered in Chapter 5 with suggestions for future areas of research.

In summary, the main contributions of this thesis are:

- A new method, *Spectral CluStream* that enables Spectral clustering to be performed in the data streaming environment.
- A study of the weighting micro-clusters in Spectral CluStream.
- A novel application of online clustering to segment Distributed Acoustic Sensing data.
- An empirical comparison of two competing recovery algorithms in the compressive sensing background subtraction setting.

Chapter 2

Spectral Clustering for Data Streams

2.1 Introduction

The goal of clustering algorithms is to separate data into groups or clusters such that data points within a cluster are similar, and data points in different clusters are dissimilar (Everit et al., 2001). Many different types of clustering algorithms exist including centroid type methods such as k-means (MacQueen, 1967; Lloyd, 1982) and density based algorithms like DB-Scan (Ester et al., 1996). In this chapter, we consider the challenge of data stream clustering. A *data stream* (Gama, 2010; Silva et al., 2013) is data which arrives in an ordered sequence. It is potentially unbounded in length and arrives continuously. Examples can be found in many applications such as telecommunications, shopping transactions and customer click data. For example, an insurance company will receive millions of quote requests an hour, and by clustering these quotes they may be able to better understand their customer base and adapt their services to meet requirements.

There exist many clustering algorithms for data streams (Yang et al., 2013; Maung and

Schweitzer, 2013; Song et al., 2013), which are able to effectively handle a large volume data stream. However, the performance of these streaming clustering algorithms tends to diminish when the underlying nature of the data stream is non-Gaussian. For example, the clusters generated from a streaming k-means algorithm are always convex sets. There is the need for a clustering algorithm which can handle data streams with less restrictive assumptions on the form of the clusters. To overcome this problem, we propose a streaming adaptation of the spectral clustering algorithm. *Spectral clustering* (von Luxburg et al., 2008) is a clustering method which uses the eigenvalues of an *affinity matrix* of the data to perform dimension reduction and cluster in the lower dimensional space. Spectral clustering is popular, offers good empirical performance and crucially, can the handle tricky, non-Gaussian data which other streaming clustering algorithms struggle to cluster. An introduction to spectral clustering is given in Section 2.2.

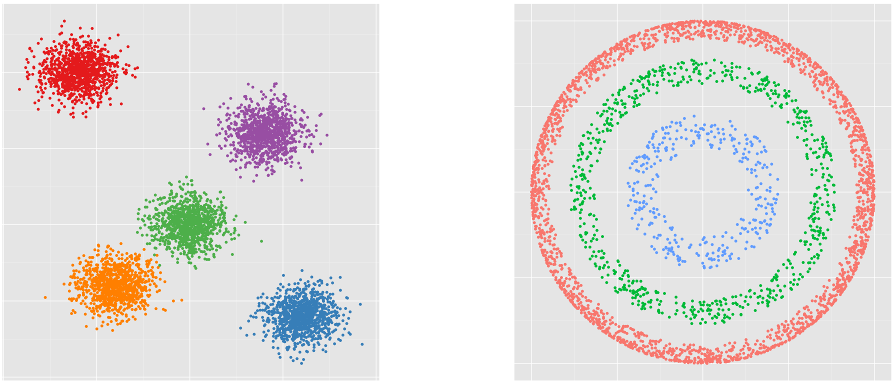
There does not exist an algorithm for performing spectral clustering on data streams, however, advanced spectral clustering techniques exist, which are discussed in Section 2.3. These algorithms address some of the issues met in data streaming, by working with large scale data or incremental data sets but do not meet all the challenges of implementing spectral clustering in data streams. In particular, these algorithms cannot deal with the fact that data streams can be potentially unbounded in size. The need for a spectral clustering algorithm which can truly be applied to data streaming is apparent.

An algorithm which can handle general clustering of data streams is the CluStream algorithm (Aggarwal et al., 2003), the details of which are explained in Section 2.4. Combining CluStream with spectral clustering on data streams has not been explored before. We introduce two *spectral CluStream* algorithms, weighted and unweighted variants and explore their performance on simulated and real data in Section 2.5. Conclusions and future work are discussed in Section 2.6.

2.2 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.

As discussed in Section 2.1 the goal of clustering algorithms is to partition data $X = \{x_1, \ldots, x_n\}, x_i \in \mathbb{R}^d$, into k disjoint clusters such that each x_i belongs to exactly one cluster. Data sets can have underlying true clusters of all shapes and sizes, for example, they can be spherical and convex as in Figure 2.2.1a or connected but non-convex as in Figure 2.2.1b.



(a) Convex clusters.

(b) Non-convex clusters.

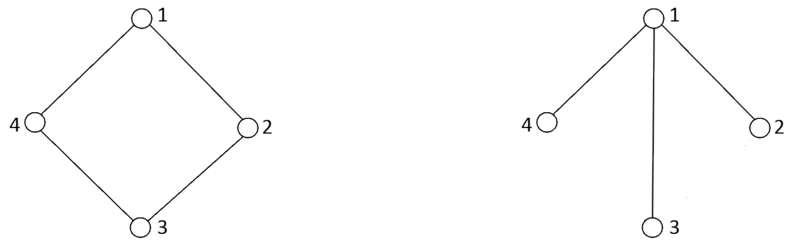
Figure 2.2.1: Examples of different types of clustering problems. The clusters in Figure 2.2.1a are generated by a multivariate Gaussian distribution. The clusters in Figure 2.2.1b are non-convex which makes the clustering problem more difficult.

Simple clustering algorithms such as k-means are good at clustering data in which the underlying true clusters are convex (Everit et al., 2001). An example of convex clusters is given in Figure 2.2.1a. However k-means can fail when the true clusters are non-convex like those shown in Figure 2.2.1b. This is because k-means is a centroid based clustering algorithm that clusters data based on how similar they are to cluster centroids. Spectral clustering instead 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 2.2.2.

The similarity between data points can be neatly represented in a graph structure. 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 being dissimilar) and the edges within a group have high weights (the data points are similar). In order to introduce spectral clustering we first 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.

2.2.1 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 will be undirected, by which we mean the edges denote a two-way relationship. The graph can then be described by an *adjacency matrix*. Adjacency matrices are a way of depicting the graph structure with binary entries denoting which vertices are connected by edges and which are not. Figure 2.2.2 depicts two similarity graphs and their corresponding adjacency matrices. A value of 1 in cell (2,3) implies that vertices 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.



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

Figure 2.2.2: Two example similarity graphs and their corresponding adjacency matrices.

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

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

$$\text{cut(A,B)} = \sum_{i \in A, j \in B} w_{ij}, \tag{2.2.1}$$

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

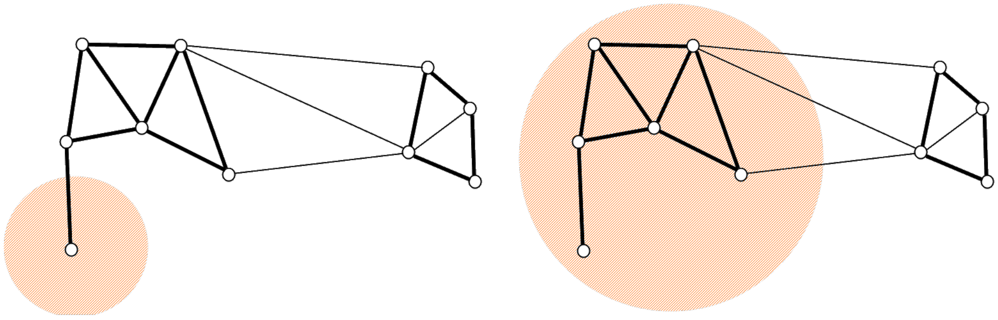
The Minimum cut (Wu and Leahy, 1993) is the cut which minimises equation (2.2.1). This can be solved in polynomial time (Stoer and Wagner, 1997) however the minimum cut does not always produce a desirable graph partitioning; it tends to create unbalanced partitions, separating one vertex from the rest of the graph. To understand why this happens, note for a fully connected graph where all vertices are joined by an edge, 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 vertices $|A|$, 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,

$$\text{RatioCut(A,B)} = \frac{\text{cut}(A,B)}{|A|} + \frac{\text{cut}(A,B)}{|B|}, \tag{2.2.2}$$

$$\text{Ncut(A,B)} = \frac{\text{cut}(A,B)}{\text{vol}(A)} + \frac{\text{cut}(A,B)}{\text{vol}(B)}. \tag{2.2.3}$$

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 2.2.3 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 2.2.3: 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 (equation 2.2.1) has been replaced with minimising the normalised cut (equation 2.2.3) which is an NP-hard problem (Wagner and Wagner, 1993). Therefore, a continuous relaxation of the Ncut is solved instead. The solution to the relaxed problem of equation (2.2.3) is given by the second eigenvector of the symmetric graph Laplacian defined in equation (2.2.5) (von Luxburg et al., 2008). Similarly, the solution of the relaxed problem the ratio cut (equation 2.2.2) is given by the second eigenvector of the unnormalized Laplacian defined in equation (2.2.4). Here W is the affinity matrix of the similarity graph and the degree matrix D is defined as the diagonal matrix with the degrees $d_1, \dots d_n$ on the diagonal. The relaxation of these graph cut problems into the eigen-decompostion of Laplacian matrices is the basis of spectral clustering.

$$L = \quad D - W. \tag{2.2.4}$$

$$L_{\text{symm}} = \quad D^{-1/2} L D^{-1/2}. \tag{2.2.5}$$

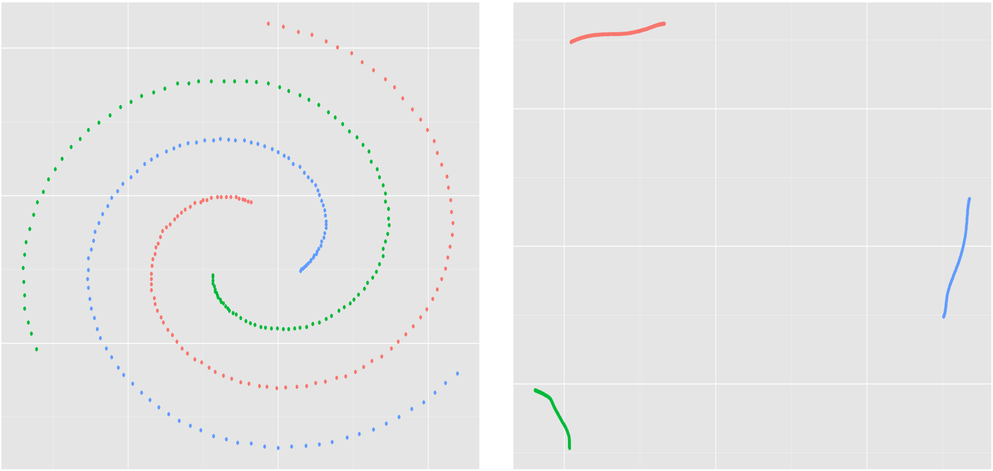
Unfortunately, there is no guarantee on the quality of the solutions of the relaxed problems compared to the exact solutions (Chung, 1997). Consequently, it has been shown that some pathological cases exist which are arbitrarily bad. However, several papers which investigate the quality of the clustering of spectral clustering (Spielman and Teng, 1996; Kannan et al., 2004) find spectral clustering to provide good solutions.

The spectral clustering algorithm that we use (Ng et al., 2001) uses the symmetric Laplacian L_{symm} as defined in equation 2.2.5. The full spectral clustering algorithm is given in Algorithm 1. Note that the number of clusters is assumed to be known and this will be the case throughout this chapter.

Algorithm 1 NJW spectral clustering algorithm

```
Input: Data set $X = \{x_1, \dots, x_n\}$, number of clusters $k$.
Output: $k$-way partition of the input data.
Construct the affinity matrix $W = (w_{ij})_{i,j=1,\dots,n}$.
Compute the symmetric Laplacian matrix $L_{\text{symm}} = D^{-1/2}(D - W)D^{-1/2}$, where $D$ is the diagonal matrix with $D_{ii} = \sum_{j=1}^{n} w_{ij}$.
Compute the $k$ eigenvectors of $L_{\text{symm}}$, $v_1, v_2, \dots, v_k$, associated with the $k$ smallest eigenvalues, and form the matrix $V = [v_1, v_2, \dots, v_k]$.
Renormalise each row of $V$ to form a new matrix $Y$.
Partition the $n$ rows of $Y$ into $k$ clusters using k-means.
Assign the original data point $x_i$ to the cluster $l$ if and only if the corresponding row $i$ of the matrix $Y$ is assigned to the cluster $l$.
```

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. For example, Figure 2.2.4a shows a data set of three spirals depicted in the original feature space. This is visually quite difficult to cluster. Figure 2.2.4b plots the same data set but embedded in the lower dimension, plotting the first eigenvector of the Laplacian against the second eigenvector. Clustering in the embedded space is easy even for k-means to solve.



(a) Data viewed in the original feature space.

(b) Data viewed in the 2-dimensional embedding space.

Figure 2.2.4: Triple spiral data set viewed in feature space (a) and eigenvector space (b).

After the embedding, k-means can be used to cluster the n rows of Y into k clusters. Finally, assign the cluster label given to each row Y_i to the corresponding original data point x_i. Now that we have introduced the general spectral clustering algorithm, we will discuss the notion of similarity in more detail.

2.2.2 Choice of affinity matrix

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

$$w_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right),\ i, j = 1, \dots, n, \tag{2.2.6}$$

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 of σ. They then select the σ which provides least distorted k-means clusters in step 5 of Algorithm 1. Zelnik-manor and Perona (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 in order to calculate σ_i.

If we mainly wish to model the local relationships then using all of the possible pairwise data similarities 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 similarities are greater than this threshold ϵ. This is an ϵ-neighbourhood graph as shown in equation (2.2.7).

$$w^*_{ij} = \begin{cases} 1, & \text{if } w_{ij} > \epsilon \\ 0, & \text{otherwise.} \end{cases} \tag{2.2.7}$$

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

2.3 Advanced Spectral Clustering

In the previous section we introduced spectral clustering via graph partitioning and discussed options for creating affinity matrices. Our overall aim is to perform spectral clustering on data streams. First, we consider some of the challenges that make clustering in data streams so difficult, and look at the approaches that exist to deal with these challenges in the spectral clustering setting.

The first challenge that is addressed is dealing with big data. One of the main difficulties in data streaming is the pure volume of data available and the methods discussed in Section 2.3.1 offer methods to perform spectral clustering on big data. Another difficulty that arises in data streaming is the ability to update the current clustering result as new data arrives. Incremental spectral clustering is a potential solution to this challenge and is discussed in Section 2.3.2.

It is stressed that these problems (large data and incremental data) are just sub-problems of what makes data streaming difficult. In particular, none of the methods described below are capable of dealing with data streams which are unbounded in length. The solutions offered in the next section to speed up computation for large data sets cannot update the clustering result if a new data point arrives. The incremental spectral clustering algorithms can update cluster membership as new data arrives but do not scale as for very large or possibly infinite n.

2.3.1 Large-scale Spectral Clustering

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 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, but in addition to using these points, Shinnou and Sasaki (2008) also include any data points which are deemed to be suitably far from any representative point in the spectral clustering. In 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. Random sampling has been applied to reduce the size of the data points within the eigen-decomposition step. Chen et al. (2006); Liu et al. (2007) introduce early stopping strategies to speed up eigen-decomposition based on the observation that well-separated data points will converge more quickly to the final embedding. 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 2.4.

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 eigen decomposition 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

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.
```

larger value of q will give a better performance but at a computational cost. The KASP authors present an upper bound on the misclustering rate given the perturbation to the original data. This bound tells us how different the clustering output is if we use the full data set to cluster compared with if we just use the representative points. However, there are a number of assumptions required (Huang et al., 2008) relating to the ability for the data to be separated into clusters. Also, this bound does not inform us about the quality of the clustering generally, only as a comparison to clustering the full data set. 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.

2.3.2 Incremental methods for Spectral Clustering

Incremental spectral clustering methods are spectral clustering algorithms which are able to update their cluster partitioning when a new data point or batch of data point arrives. The term incremental is used here rather than online because although these algorithms can update when new data points arrive, they are not designed to deal with the full data streaming scenario. As a reminder, data streams are defined by the constant arrival of data points at a high velocity relative to the available processing power. On the other hand, incremental spectral clustering algorithms are designed for the problem of an evolving data set. For example, lets say we are interested in understanding academic relationships within Lancaster University. We may first define a similarity score between two academics based on the number of papers that they have authored together. We could then use this similarity to create an affinity matrix and perform spectral clustering to discover the academic clustering within the University. However this affinity matrix will not remain static. A new academic may join the University, or two existing academics may author a new paper changing their similarity score. Performing a full re-clustering of the whole University may be costly and potentially not computationally feasible. Instead, we may wish to update our clustering solution just by incorporating this new information. This is the type of problem that incremental spectral clustering algorithms seek to address. 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 existing clusters as new data points arrive.

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

Kong et al. (2011) is a mixture of both Ning and Valgren's methods, using representative points like Valgren but the eigen-updating of Ning. Although it can be quicker that Ning it retains the other issues of Ning's method discussed above. In addition it has the same problem of Valgren's method that the number of clusters increases over time. This makes it unsuitable for data streams.

Although the methods discussed deal with some aspects of difficulties in data streams, none of them are suitable for the full problem of clustering a data stream. We introduce an online spectral clustering algorithm for data streams based on the CluStream model of Aggarwal et al. (2003) in Section 2.4.

2.4 CluStream for Spectral Clustering

In this section, we first review a number of general methods for data stream clustering, although none of these offer spectral clustering for data streams. We then discuss the CluStream algorithm in more depth and introduce our algorithm, spectral CluStream. Finally, we consider whether to weight micro-clusters within the spectral CluStream algorithm.

2.4.1 Data Stream Clustering

Data stream clustering algorithms take classic clustering algorithms such as k-means (MacQueen, 1967) and DBSCAN (Ester et al., 1996) and adapt them to work in the data streaming environment. In order to do this they need to address the many challenges involving data streams, such as one-pass access, non-stationarity and the potential for the stream to be unbounded.

One of the first algorithms able to identify clusters in large and incremental data sets was BIRCH (Zhang et al., 1996). BIRCH achieved this by using cluster feature vectors to summarise the data stream and perform hierarchical clustering. BIRCH exploits the observation that the feature space is not usually uniformly occupied, and therefore not every data point is equally important in terms of clustering. The fact that BIRCH generally only requires one pass of the data made it faster than existing clustering methods and allowed it to be used to cluster data streams. However, BIRCH does not perform well if clusters are not spherical as it uses the cluster radius to define clusters boundaries.

The cluster feature vector concept developed in BIRCH was developed and re-named as a micro-cluster in the CluStream framework (Aggarwal et al., 2003). CluStream, like BIRCH separates the clustering process into two stages, referred to in CluStream as a micro-clustering stage and a macro-clustering stage. The main difference between BIRCH and CluStream is that CluStream stores temporal as well as spatial statistical summaries in the micro-clusters. By incorporating temporal information, it is able to handle non-stationarity in data streams. CluStream has been very influential in the data streaming community; since the paper was first published in 2003 it has been cited over 1800 times. The CluStream algorithm will be discussed in more detail in the next section as we use it as the basis for our spectral CluStream algorithm.

Other methods which has been inspired by BIRCH's cluster feature vectors include ClusTree (Kranen et al., 2011), a hierarchical method which can adapt its performance depending on the stream velocity and and scalable k-means (Bradley and Fayyad, 1998).

DenStream (Cao et al., 2006) is a density-based data stream clustering algorithm that also uses micro-clusters in the online stage. The offline component of DenStream is a variation of DBSCAN, enabling the detection non-linearly separable clusters of arbitrary shape. However, DBScan cannot cluster data sets well with large differences in densities, due to global parameterisation. DStream (Chen and Tu, 2007) is similar to DenStream, except that the data summarisation stage involves partitioning the feature space into dense grid cells instead of micro-clusters. However, the number of grid cells depends exponentially on the dimension of the data meaning that this algorithm is not suitable for high-dimensional data.

An extensive review of existing data stream clustering algorithms is provided in Silva et al. (2013). However, none of the algorithms discussed above or included in Silva's review enable spectral clustering to be performed in a data streaming context. Our spectral CluStream algorithm uses the micro-clustering approach of CluStream with a spectral clustering algorithm for the macro-clustering stage. We chose to use the CluStream micro-clustering method due to it's universal popularity, good performance, and ability to handle evolving data. In the next section we detail how the micro-clustering step in the CluStream algorithm works as described in Aggarwal et al. (2003).

CluStream: Micro-clustering

CluStream is a framework for clustering data streams which separates the clustering process into 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. 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. To initialise the algorithm a k-means is performed on a training set with q clusters. The value of q should be chosen to be much larger than the expected number of true macro-clusters k. The aim 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. However unfortunately this is not guaranteed by the method.

Assume that we have a data stream S which consists of d-dimensional data $\boldsymbol{x_i}$ arriving in sequence, $S = \{\boldsymbol{x_i}\}_{i\in\mathbb{N}}, \boldsymbol{x_i} \in \mathbb{R}^d$. Each micro-cluster $M_j, (j \in 1\ldots, q)$ is stored as a $(2 \cdot d + 3)$ tuple $(\boldsymbol{CF1^x_j}, \boldsymbol{CF2^x_j}, n_j, CF1^t_j, CF2^t_j)$. The definitions are given in equation (2.4.1). $\boldsymbol{CF1^x_j}$ 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

$$\begin{aligned}
\boldsymbol{CF1_j^x} &= \sum_{x_i \in M_j} \boldsymbol{x_i} \ , \\
\boldsymbol{CF2_j^x} &= \sum_{x_i \in M_j} (\boldsymbol{x_i})^2 \ , \\
CF1_j^t &= \sum_{i|x_i \in M_j} t_i \ , \\
CF2_j^t &= \sum_{i|x_i \in M_j} (t_i)^2 \ , \\
n_j &= \sum_{x_i \in M_j} 1 \ . \qquad (2.4.1)
\end{aligned}$$

Here the notation $(\boldsymbol{x_i})^2$ means the vector where each element is the square of the corresponding element in $\boldsymbol{x_i}$. 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 (2.4.2) is applied.

$$\begin{aligned}
\boldsymbol{CF1_j^x} &\leftarrow \boldsymbol{CF1_j^x} + \boldsymbol{x}_{\text{new}} \ , \\
\boldsymbol{CF2_j^x} &\leftarrow \boldsymbol{CF2_j^x} + (\boldsymbol{x}_{\text{new}})^2 \ , \\
CF1_j^t &\leftarrow CF1_j^t + t_{\text{new}} \ , \\
CF2_j^t &\leftarrow CF2_j^t + (t_{\text{new}})^2 \ , \\
n_j &\leftarrow n_j + 1 \ . \qquad (2.4.2)
\end{aligned}$$

Note that updating the micro-clusters requires only addition therefore updating is computationally efficient. Critically it is possible to use these summaries to calculate the centre of each micro-cluster as in equation (2.4.3).

$$\text{Centre of micro-cluster j} = \boldsymbol{\bar{M}_j} = \frac{\boldsymbol{CF1_j^x}}{n_j}. \qquad (2.4.3)$$

It is these centres which are 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.

When a new data point arrives it's nearest micro-cluster M_* is identified using the Euclidean distance metric given in equation (2.4.4).

$$M_* = \min_{M_j, j \in 1:q} \|\boldsymbol{x_i} - \bar{\boldsymbol{M_j}}\|^2. \tag{2.4.4}$$

To determine whether the new data point is suitably close enough to M_* we need to consider the maximum boundary factor (MBF) of M_*. In CluStream, the maximum boundary factor is defined as a factor of τ of the root-mean-square deviation of the data points in M_* from the centroid of M_*. The value of τ should be chosen small enough so that it can successfully detect most of the points representing new clusters or outliers. At the same time, it should not generate too many unpromising new micro-clusters. Aggarwal et al. (2003) compared values of $\tau \in (1, 8)$ and recommend setting $\tau = 2$.

If the new data point falls within the MBF of it's nearest micro-cluster M_* then it is absorbed as part of that cluster. If not, a new micro-cluster is created. However, as the number of micro-clusters must remain fixed at all times, if a new micro-cluster is formed, either an existing micro-cluster must be deleted, or two close micro-clusters should be merged.

The first step is to see if an existing micro-cluster can be deleted to make room for the new micro-cluster. The criteria for deleting micro-clusters is their *relevancy.* CluStream approximates the average time stamp of the last m data points of the cluster M_j (where m is a user chosen parameter) and judges if the cluster is old enough to discard. Let the mean and standard deviation of the arrival times for a micro-cluster M_j be given by μM_j and σM_j. These can easily be calculated as we store $CF1^t$ and $CF2^t$. The *relevancy stamp* $r(M_j)$ is defined to be the arrival of the $(m/2n_j)^{th}$ percentile of the points in M_j assuming the time stamps are Normally distributed. We check if the micro-cluster with the smallest relevancy stamp has $r(M_j) < \delta$, where δ is some user-chosen deletion threshold as given in equation (2.4.5).

$$\min_{M_j, j \in 1:q} (r(M_j)) < \delta. \tag{2.4.5}$$

If the inequality in equation (2.4.5) holds then the micro-cluster with the minimum relevancy stamp is deleted. If not, then no micro-clusters are deleted and instead the two closest micro-clusters are merged. If two micro-clusters M_r and M_s are to be merged, the updates given in equation (2.4.6) 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.

$$\begin{aligned}
\boldsymbol{CF1_r^x} &\leftarrow \boldsymbol{CF1_r^x} + \boldsymbol{CF1_s^x}\ , \\
\boldsymbol{CF2_r^x} &\leftarrow \boldsymbol{CF2_r^x} + \boldsymbol{CF2_s^x}\ , \\
CF1_r^t &\leftarrow CF1_r^t + CF1_s^t\ , \\
CF2_r^t &\leftarrow CF2_r^t + CF2_s^t\ , \\
n_r &\leftarrow n_r + n_s\ . \qquad (2.4.6)
\end{aligned}$$

With this online micro-cluster maintenance, the data stream should remain well represented over time. The micro-clustering update algorithm for CluStream is given in Algorithm

3.

Algorithm 3 CluStream Micro-clustering

```
Input: Data Stream $S = \{\boldsymbol{x_i}\}_{i \in \mathbb{N}}, \boldsymbol{x_i} \in \mathbb{R}^d$, number of micro-clusters $q$, parameters $\delta$, $\tau$, $m$.
Output: Micro-clusters $M_1, \dots, M_q$.
1: Initialise the micro-clusters k-means($x_1, \dots x_{init}, q$) and equations (2.4.1).
2: for each new data point $x_i$ do
3:     Find the closest micro-cluster to $x_i$, $M_*$ using equation (2.4.4).
4:     if $x_i$ falls within the maximum boundary for $M_*$ then
5:         absorb $x_i$ into micro-cluster $M_*$ using equations (2.4.2).
6:     else
7:         Use $x_i$ to initialise it's own new micro-cluster using equations (2.4.1).
8:         if any micro-cluster is suitably old according to equation (2.4.5) then
9:             Remove the oldest micro-cluster.
10:        else
11:            Merge the two closest micro-clusters using equation (2.4.6).
12:        end if
13:    end if
14: end for
```

The efficiency of updating the micro-clusters

We now consider the computational cost of the micro-clustering update described in the loop in Algorithm 3. The most computationally expensive step is step 3. This step involves computing the Euclidean distance which has complexity $\mathcal{O}(n)$. Therefore, we would expect the run time to increase linearly as the number of micro-clusters is increased.

We ran an experiment on simulated data to see how the run time scales with the number of micro-clusters. The time to run one update of the micro-clustering algorithm was recorded 100 times using the microbenchmark package in R. The experiment was run on a ThinkPad laptop with Intel Core i5-7200U CPU @ 2.50GHz running Ubuntu 16.04. The results are shown in Figure 2.4.1.

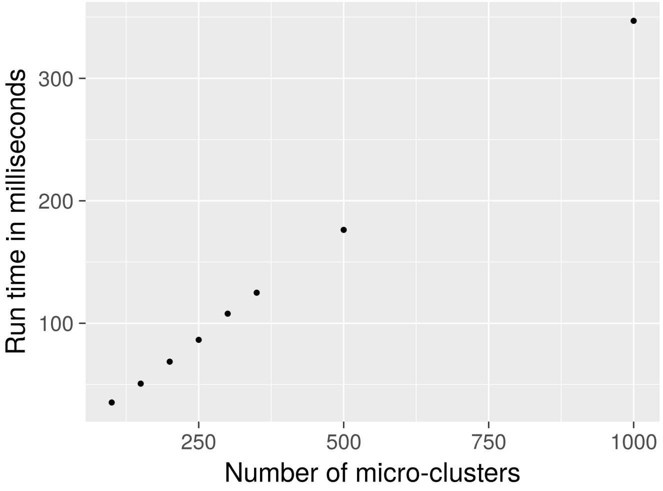


Figure 2.4.1: The effect of the number of micro-clusters on run time for one update.

Each data point represents the median run time in milliseconds over 100 runs. As we can see, the run time does increase linearly with the number of micro-clusters.

We also investigated the size of dimension of the data set on the run time of one update. The dimension of the data set had no effect on run time.

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. By using the micro-clusters to summarise the data, we can therefore perform spectral clustering on data streams. The full algorithm is given in Algorithm 4.

Algorithm 4 Spectral CluStream

```
Input: Data Stream $S = \{\boldsymbol{x_i}\}_{i \in \mathbb{N}}, \boldsymbol{x_i} \in \mathbb{R}^d$, number of clusters $k$, number of micro-clusters
    $q$, parameters $\delta$, $\tau$, $m$.
Output: A $k$ way clustering of the micro-clusters $M_1, \ldots, M_q$.
Initialise the micro-clusters using k-means($x_1, \ldots x_{init}, q$) and equations (2.4.1).
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 $M_1, \ldots, M_q$ with $k$ clusters.
    end if
end for
```

The frequency with which the macro-clustering stage is run depends on how often the user requires a summary of the clusters. This will be determined by the nature of the application. As the macro-clustering stage is considered an offline step in CluStream (Aggarwal et al., 2003), the efficiency of the algorithm is not compromised by running the macro-clustering more frequently.

There are a couple of possible ways to feed the micro-clusters into a spectral clustering algorithm (step 5 of Algorithm 4). Two of the options suggested in Zhang et al. (1996) listed here.

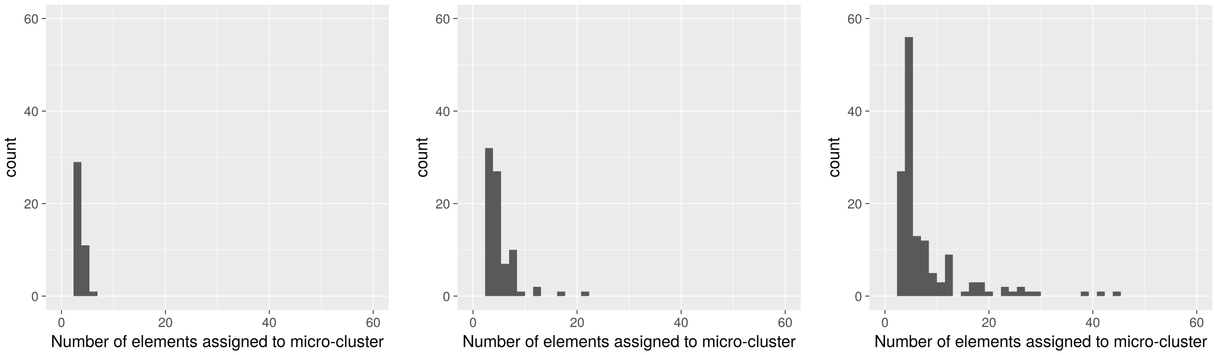
1. Calculate the centre of each micro-cluster $\bar{M}_j$ and use it as an object to be clustered by the macro-clustering algorithm.

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.

No guidance is given in Zhang et al. (1996) to how these two different approaches might affect the final clustering result. Next, in Section 2.4.2 we describe how to weight the micro-cluster centers. Later in Section 2.5 we will analyse the performance of both unweighted and weighted online spectral clustering.

2.4.2 Weighting the Micro-Clusters

In this section, we discuss how to 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. Weighting the micro-clusters is suggested in Zhang et al. (1996) but why might it be beneficial to weight the micro-clusters? The first thing to note is that the number of data points assigned to micro-clusters is not uniform across the micro-clusters, and this distribution will change as the stream progresses. For example, Figure 2.4.2a shows a histogram of the number of points assigned to micro-clusters at the start of a data stream, Figure 2.4.2b shows the middle of the stream, and Figure 2.4.2c 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 2.4.2: Histograms showing the number of points assigned to micro-clusters.

Secondly, imagine the scenario pictured in Figure 2.4.3 where we have two clusters, one much more dense that then other. In the example, many micro-clusters are used to represent the cluster on the right, although each micro-cluster only has a few data points assigned to it. The more dense cluster in the bottom left of the plot has only 3 micro-clusters representing it, but each micro-cluster has hundreds of data points assigned to it. Weighting by the number of points assigned to a micro-cluster may help balance out this scenario for the spectral clustering stage.

In order to weight the micro-clusters, we simply construct an affinity matrix as described. Let $W \in \mathbb{R}^{q \times q}$ 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,

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

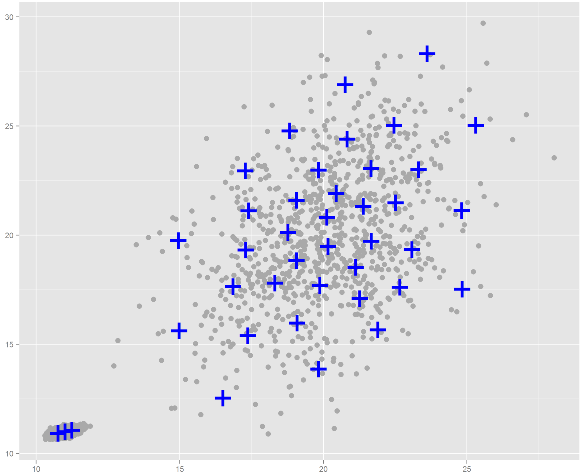


Figure 2.4.3: Possible micro-cluster locations in a toy example.

Define the *weighted affinity matrix* to be $\tilde{W} \in \mathbb{R}^{q \times q}$ where $\tilde{W_{ij}} = n_i n_j W_{ij}$. We can see that $\tilde{W}$ is a valid affinity matrix since it is symmetric with non-negative entries. If we wish to have $\tilde{W}_{ij} \leq 1$ then simply divide $\tilde{W}$ by $\max_i n_i^2$, but this makes no difference to the spectral decomposition (von Luxburg et al., 2008).

There exists a link between the spectral decomposition of the Laplacian generated by $\tilde{W}$ and the Laplacian arising from a data set of repeated points, which we define as follows. Let $W^* \in \mathbb{R}^{n \times n}$ 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 W^* 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.

Let us 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-cluster 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. The affinity, degree and Laplacian matrices (W, D and L) for the two micro-cluster centres are given in equation (2.4.7).

$$W = \begin{pmatrix} 1 & s \\ s & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1+s & 0 \\ 0 & 1+s \end{pmatrix}, \quad L = \begin{pmatrix} \frac{1}{1+s} & \frac{s}{1+s} \\ \frac{s}{1+s} & \frac{1}{1+s} \end{pmatrix}. \tag{2.4.7}$$

In order to create a weighted version of the affinity matrix, we simply multiply through by n_1 and n_2. The weighted affinity matrix $\tilde{W}$ and related degree and Laplacians ($\tilde{D}$ and $\tilde{L}$) are given in equation (2.4.8).

$$\begin{gathered} \tilde{W} = \begin{pmatrix} n_1^2 & sn_1n_2 \\ sn_1n_2 & n_2^2 \end{pmatrix}, \quad \tilde{D} = \begin{pmatrix} n_1^2 + sn_1n_2 & 0 \\ 0 & n_2^2 + sn_1n_2 \end{pmatrix}, \\ \tilde{L} = \begin{pmatrix} \frac{n_1^2}{n_1^2+sn_1n_2} & \frac{sn_1n_2}{\sqrt{(n_1^2+sn_1n_2)(n_2^2+sn_1n_2)}} \\ \frac{sn_1n_2}{\sqrt{(n_1^2+sn_1n_2)(n_2^2+sn_1n_2)}} & \frac{n_2^2}{n_2^2+sn_1n_2} \end{pmatrix}. \end{gathered} \tag{2.4.8}$$

Finally we observe the construction of the repeated affinity matrix given in equation (2.4.9). Here the block nature in W^*, D^* and L^* is clear. The first n_1 rows of W^* relate to the centre of micro-cluster 1, and the bottom n_2 rows relate to the centre of micro-cluster 2.

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

$$L^* = \begin{pmatrix} \frac{1}{\star} & \dots & \frac{1}{\star} & \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\star} & \dots & \frac{1}{\star} & \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} \\ \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} & \frac{1}{\triangle} & \dots & \frac{1}{\triangle} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{s}{\sqrt{\star\triangle}} & \dots & \frac{s}{\sqrt{\star\triangle}} & \frac{1}{\triangle} & \dots & \frac{1}{\triangle} \end{pmatrix}, \tag{2.4.9}$$

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

If we evaluate these expressions for a particular numerical case, we can see how the spectral decomposition of the matrices $\tilde{L}$ and L^* are linked. Let $s = 0.5$, $n_1 = 3$, $n_2 = 2$. The 2nd smallest eigenvector of L^* is

$$e_2^* = \begin{bmatrix} -0.350 & -0.350 & -0.350 & 0.562 & 0.562 \end{bmatrix}. \tag{2.4.10}$$

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

$$\tilde{e}_2 = \begin{bmatrix} 0.607 & -0.795 \end{bmatrix}. \tag{2.4.11}$$

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

$$\left[\begin{array}{c} \overbrace{\frac{0.607}{\sqrt{3}}\ \frac{0.607}{\sqrt{3}}\ \frac{0.607}{\sqrt{3}}}^{n_1}\ \overbrace{\frac{-0.795}{\sqrt{2}}\ \frac{-0.795}{\sqrt{2}}}^{n_2} \end{array}\right] = \left[\begin{array}{ccccc} 0.350 & 0.350 & 0.350 & -0.562 & -0.562 \end{array}\right]. \tag{2.4.12}$$

We can see that the right hand vector in equation (2.4.12) is the negative of the 2nd smallest eigenvector of L^*, e_2^* given in equation (2.4.10).

In fact, we have seen empirically that the expanded repeated eigenvector of the weighted Laplacian is always either equal to e_k^* or $-e_k^*$ for all k. This means that the partition generated by performing spectral clustering on the weighted micro-cluster centers will be the same as the partition generated by performing spectral clustering on a set of repeated micro-cluster centers.

Proposition 2.4.1 proves that the second smallest eigenvectors of spectral decomposition of L^* and $\tilde{L}$ are linked in this way.

Proposition 2.4.1. *Let L be a symmetric Laplacian matrix. Define $\tilde{L}$ and L^* to be the corresponding weighted and expanded Laplacians respectively. Let $\tilde{u}$ be the second smallest eigenvector of $\tilde{L}$. Then u^* is the second smallest eigenvector of L^*, where u^* is defined as* $u^* = \left(\overbrace{\frac{\tilde{u}_1}{\sqrt{n_1}}, \frac{\tilde{u}_1}{\sqrt{n_1}}, \frac{\tilde{u}_1}{\sqrt{n_1}}}^{n_1}, \ldots, \overbrace{\frac{\tilde{u}_k}{\sqrt{n_k}}, \frac{\tilde{u}_k}{\sqrt{n_k}}}^{n_k}\right)$.

Proof. We wish to show that u^* is the second smallest eigenvector of L^*. First we will show that u^* is an eigenvector of L^* which is orthogonal to the smallest eigenvector of L^*. Then we will show by proof by contradiction that u^* must be the second smallest eigenvector of L^*.

i) $\|u^*\| = 1$.

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

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

ii) $u^* \perp D^{*1/2}\mathbb{1}$.

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

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

iii) $u^{*\top} L^* u^* = \tilde{u}^\top \tilde{L} \tilde{u}$.

First we state the general property given in von Luxburg et al. (2008) for the normalised Laplacian. For every $f \in \mathbb{R}^n$ we have,

$$f' L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2. \tag{2.4.13}$$

Using this property,

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

which we can see is equal to $u^{*\top} L^* u^*$ by observing that W^* has repeated elements from W.

Now that all of the above criteria have been satisfied, we know that u^* is an eigenvector of L^* and is orthogonal to the smallest eigenvector. Assume there exists some v^* such that $v^* \perp D^{*1/2}\mathbb{1}$ with $\|v^*\| = 1$ and $v^{*\top} L^* v^* < u^{*\top} L^* u^*$. Then $\exists\ \tilde{v}$ with $\tilde{v}^\top \perp \tilde{D}^{1/2}\mathbb{1}$ and $\tilde{v}^T \tilde{L} \tilde{v} < \tilde{u}^\top \tilde{L} \tilde{u}$. This contradicts the fact that $\tilde{u}$ is the second smallest eigenvector of $\tilde{L}$. Therefore u^* is the second smallest eigenvector of L^*.

□

2.5 Experimentation

In this section, we investigate the performance of both unweighted and weighted spectral CluStream and a simple windowed approach to spectral clustering. We intended to compare against the incremental spectral clustering method of Ning et al. (2010) discussed in Section 2.3.2 however due to the computational cost of the method this was not possible. This underlines the point made previously that incremental methods struggle in streaming settings. First the algorithms and methodology are introduced and the performance metrics defined. Then the algorithms are compared on simulated data, two image based data sets and an evolving data set.

The Algorithms

Spectral CluStream is given in Algorithm 4. As described in Section 2.4, there are two ways that 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 spectral CluStream algorithms we use $q = 150$ micro-clusters to summarise the data stream. This value of q was chosen as initial experiments showed that 150 micro-clusters was sufficient to represent the underlying data stream without being computationally expensive to update. This is demonstrated in Section 2.5.3. In Section 2.5.7 we compare results for varying values of q.

Windowed approaches are often used in data streaming as a computationally simple way to monitor a data stream. A window of the w most recently observed data points is retained. When a new data point is observed, the oldest data point in the window is discarded to make room for the new data point. Windowed spectral clustering uses the data points in the current window as input into the spectral clustering algorithm given in Algorithm 1. Choosing a suitable window size w can be difficult. A large window size will contain lots of historical information but will be slow to adapt to changes in the data stream, whilst a small window can update quickly but may not be informative enough. There do exist methods for adaptive window sizes but we decided to fix the window size. This was decided for two reasons, it is computationally more efficient and it makes a fairer comparison with CluStream which uses a fixed number of micro-clusters. The window size selected was $w = 150$ as it is provides a good trade off between retaining information and adapting to changes in the data stream. In Section 2.5.7 we compare results for different values of w.

The data sets

The data sets used in this section are the S sets (Fränti and Virmajoki, 2006), a texture data set (Kylberg, 2011) and the UCI Pendigits (Lichman, 2013). The full details of these will be introduced when we come to them. However, these data sets exist in a static form. In order to use these data sets as data streams we use the following online generating mechanism. For each data set, ten data streams are generated by randomly sampling from the data set without replacement.

2.5.1 Methodology

A data stream $S = \{\boldsymbol{x_i}\}_{i \in \mathbb{N}}$ is observed sequentially, with one data point $\boldsymbol{x_t}$ observed at each time point. The order in which the data stream is observed is randomised to generate 10 different runs. Results are averaged out over these runs.

Each run consists of four stages; initialisation, updating, generating cluster labels and evaluation. Spectral CluStream is initialised by applying k-means with 150 clusters to the first 500 data points and applying equations (2.4.1). When a new data point $\boldsymbol{x_t}$ is observed the streaming algorithms are updated. In spectral CluStream this is achieved by applying Algorithm 3. Windowed spectral clustering shifts the window along by one, discarding the oldest data point $\boldsymbol{x_{(t-150)}}$ and including $\boldsymbol{x_t}$. Every ten times steps we generate cluster labels for the data stream by applying spectral clustering on the *representative points* of the data stream. In spectral CluStream the representative points are the micro-clusters (weight adjusted or not). In windowed spectral clustering the representative points are the w data points currently in the window. We then evaluate the spectral clustering performance on a *test set* defined to be the next 200 data points in the data stream ($\boldsymbol{x_{t+1}}$, $\ldots \boldsymbol{x_{t+200}}$). The test data points are first assigned to their nearest representative point using the Euclidean distance metric and then take on the cluster label of that representative point. Performance is measured in terms of purity and V-measure of the test data using the known true clusters.

Purity and V-measure are defined in Section 2.5.2.

The parameter settings throughout are as follows. In spectral CluStream algorithms we set $\delta = 0.1$, $m = 1$ and $\tau = 2$ as suggested in Aggarwal et al. (2003). For the first three experiments we use $q = 150$ micro-clusters. In Section 2.5.7 we compare results for a number of different values of q. Similarly, in the first three experiments we use a window size of $w = 150$ for windowed spectral clustering, but investigate varying w in the final experiment. In all experiments, we set the number of clusters k to be equal to the known true number of classes for the data set.

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

Let n be the number of data points, $U = \{u_i|1, \dots, k\}$ be the set of true classes and $V = \{v_j|1, \dots, k\}$ 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 u_i and assigned to cluster v_j.

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. Repeat this for all clusters, sum these counts and divide by the total number of data points. This is shown mathematically in equation (2.5.1).

$$\text{Purity} = \frac{1}{n}\sum_{j=1}^{k}\max_i(a_{ij}). \tag{2.5.1}$$

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 (2.5.2):

$$\text{V-measure} = 2\frac{h \times c}{h + c}\ , \tag{2.5.2}$$

where h and c are homogeneity and completeness measures as defined below in equation (2.5.3) and equation (2.5.6) 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.

$$h = \begin{cases} 1 & \text{if } H(U|V) = 0 \\ 1 - \frac{H(U|V)}{H(U)}, & \text{otherwise.} \end{cases} \tag{2.5.3}$$

$$H(U|V) = -\sum_{j=1}^{k} \sum_{i=1}^{k} \frac{a_{ij}}{N} \log \frac{a_{ij}}{\sum_{i=1}^{k} a_{ij}}. \tag{2.5.4}$$

$$H(U) = -\sum_{i=1}^{k} \frac{\sum_{j=1}^{k} a_{ij}}{N} \log \frac{\sum_{j=1}^{k} a_{ij}}{N}. \tag{2.5.5}$$

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

$$c = \begin{cases} 1 & \text{if } H(V|U) = 0 \\ 1 - \frac{H(V|U)}{H(V)}, & \text{otherwise.} \end{cases} \tag{2.5.6}$$

$$H(V|U) = -\sum_{i=1}^{k} \sum_{j=1}^{k} \frac{a_{ij}}{N} \log \frac{a_{ij}}{\sum_{j=1}^{k} a_{ij}}. \tag{2.5.7}$$

$$H(V) = -\sum_{j=1}^{k} \frac{\sum_{i=1}^{k} a_{ij}}{N} \log \frac{\sum_{i=1}^{k} a_{ij}}{N}. \tag{2.5.8}$$

2.5.3 Parameter Choices

For the experiments in which the number of micro-clusters (q) and window size (w) is fixed, the value of the parameters must be selected. A brief computational study was carried out on a selection of simulated data sets for the purpose of parameter selection. Table 2.5.1 shows the Unweighted CluStream algorithm performance in terms of purity and V-measure for a number of parameter choices. The results are averaged over 10 runs. We can see that the performance improves as the number of micro-clusters increases, however, the performance seems to plateau above $q = 150$. Given that the computationally time increases linearly (see Section 2.4.1) $q = 150$ is a sensible parameter choice for the purposes of our study.

Now, we consider the choice of window size (w). Table 2.5.1 shows the windowed algorithm performance in terms of purity and V-measure for a number of parameter choices. Again,the results are averaged over 10 runs. We see that the performance improves marginally as the window size increases, however, the performance appears to be flat around $w = 150$ to $w = 200$. As there is no advantage to using a window size greater than 150 we choose to set $w = 150$. Using the same value for both q and w has the added advantage that the memory requirements for both algorithms will be comparable.

Table 2.5.1: Select of nMicro parameter for CluStream.

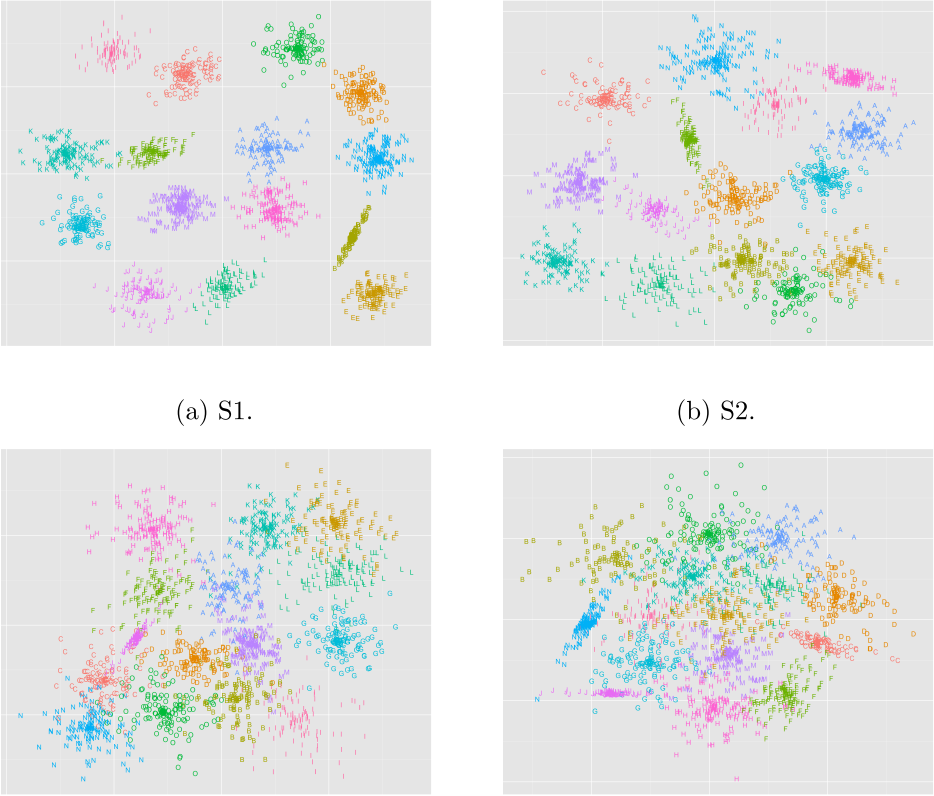
nMicro	Data set	Purity	Vmeasure
50	S1	0.935	0.955
100	S1	0.955	0.969
150	S1	0.957	0.972
200	S1	0.944	0.969
50	S2	0.870	0.891
100	S2	0.888	0.903
150	S2	0.894	0.907
200	S2	0.910	0.924
50	S3	0.705	0.760
100	S3	0.723	0.762
150	S3	0.747	0.776
200	S3	0.746	0.775
50	S4	0.649	0.704
100	S4	0.671	0.711
150	S4	0.677	0.712
200	S4	0.698	0.723

Table 2.5.2: Selection of window size parameter for windowed algorithm.

Window Size	Data set	Purity	Vmeasure
50	S1	0.928	0.957
100	S1	0.912	0.952
150	S1	0.922	0.959
200	S1	0.916	0.955
50	S2	0.872	0.903
100	S2	0.889	0.912
150	S2	0.898	0.918
200	S2	0.897	0.918
50	S3	0.741	0.779
100	S3	0.777	0.793
150	S3	0.794	0.804
200	S3	0.787	0.796
50	S4	0.69	0.723
100	S4	0.713	0.735
150	S4	0.726	0.738
200	S4	0.725	0.739

2.5.4 Simulated Results

The first data sets tested are the popular S-sets, first introduced in Fränti and Virmajoki (2006). The four data sets are shown in Figure 2.5.1.



(a) S1.

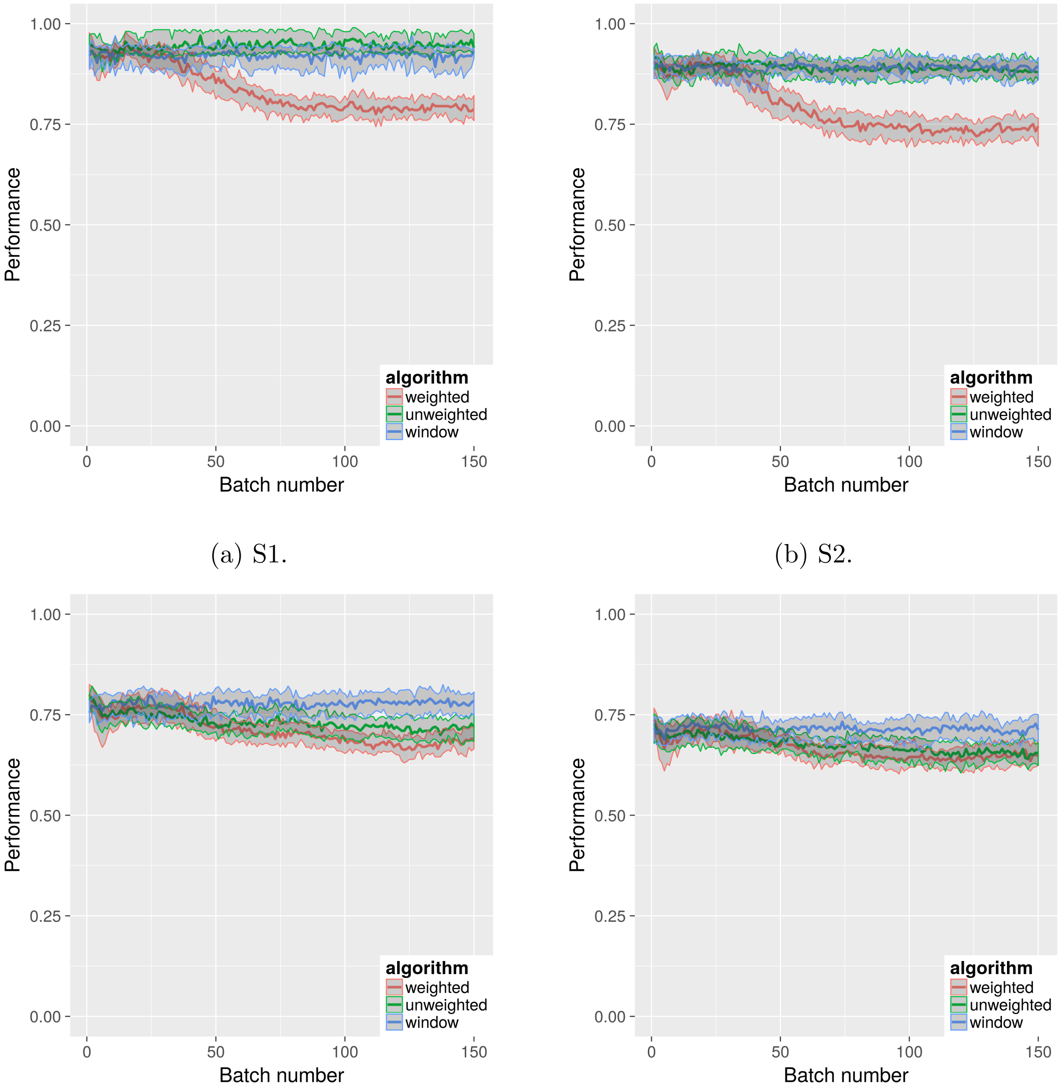
(b) S2.

(c) S3.

(d) S4.

Figure 2.5.1: The S sets (Fränti and Virmajoki, 2006), two-dimensional data sets with varying degrees of overlap. The true clusters labels are shown.

Each set consists of synthetic two-dimensional data with n=5000 data points and k=15 Gaussian clusters with different degrees of cluster overlapping. Data set S1 should be the easiest to cluster as all 15 clusters are well fairly separated. The sets become increasingly more challenging and S4 can be difficult even for humans to separate correctly. We treated each of these data sets as a data stream and used spectral CluStream and windowed spectral clustering to assign data points to clusters. Performance is evaluated in batch every 10^{th} time point. The results are shown in Figures 2.5.2 and 2.5.3.



(a) S1.

(b) S2.

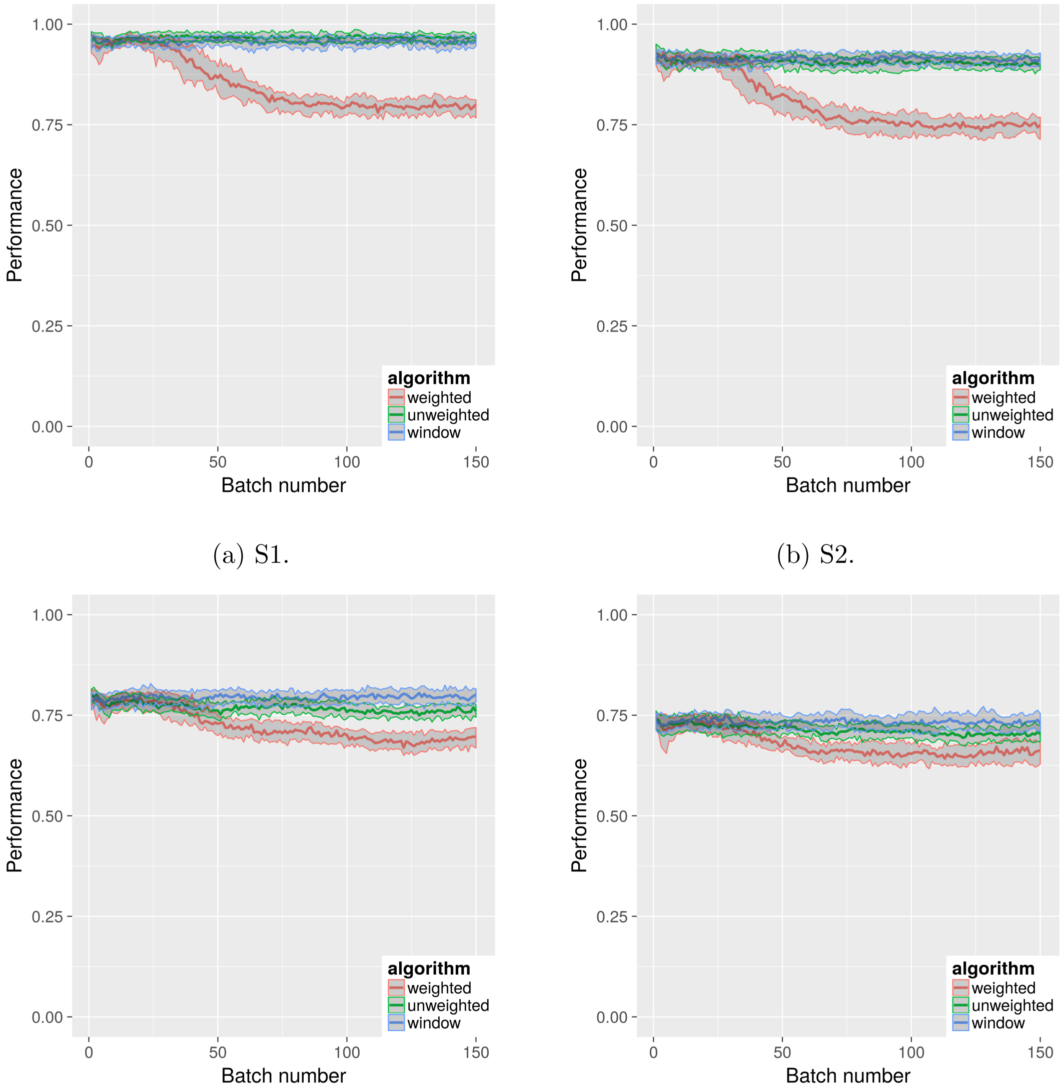
(c) S3.

(d) S4.

Figure 2.5.2: Purity for the S data sets.

Figure 2.5.2 plots the performance in terms of purity on the y-axis and the batch number is given on the x-axis. The average performance for each algorithm over the multiple runs is given by the thick solid lines and the shaded area depicts the inter-quartile range. Figure 2.5.3

shows the performance in terms of V-measure. Both figures show the algorithm performance at each batch step, meaning that we can see how performance changes as the data stream progresses. However, these data sets are stationary in distribution and therefore we would not expect to see algorithm performance vary dramatically with time.



(a) S1.

(b) S2.

(c) S3.

(d) S4.

Figure 2.5.3: V-measure for the S data sets.

Both unweighted spectral CluStream (green) and windowed spectral clustering (blue) perform similarly for all sets. They both perform well on set S1 and set S2 but they struggle with the more challenging sets S3 and S4. 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 as the stream progresses. Given that the underlying distributions for the S sets are stationary, this behaviour is unusual.

In order to discover why weighted spectral CluStream is performing poorly, lets look at the micro-clusters more closely. Figure 2.5.4 shows a snapshot of the weighted spectral CluStream algorithm on the S1 data set in the middle of the stream.

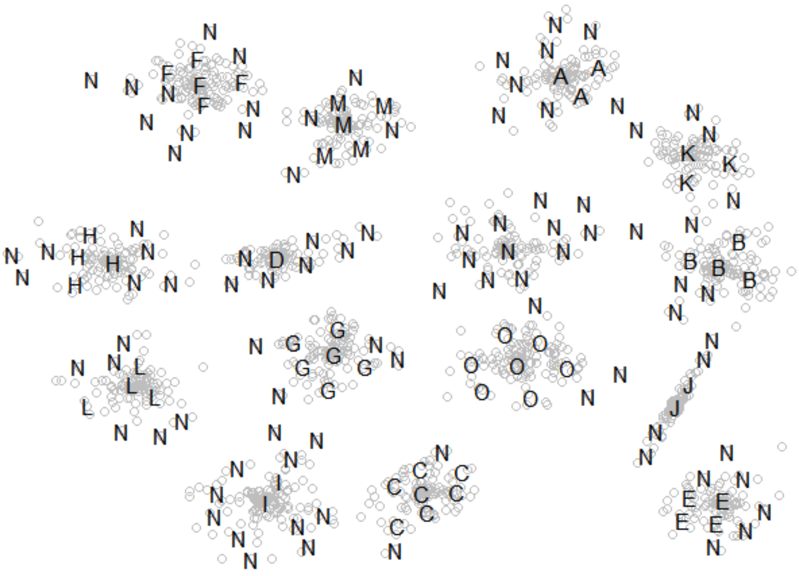
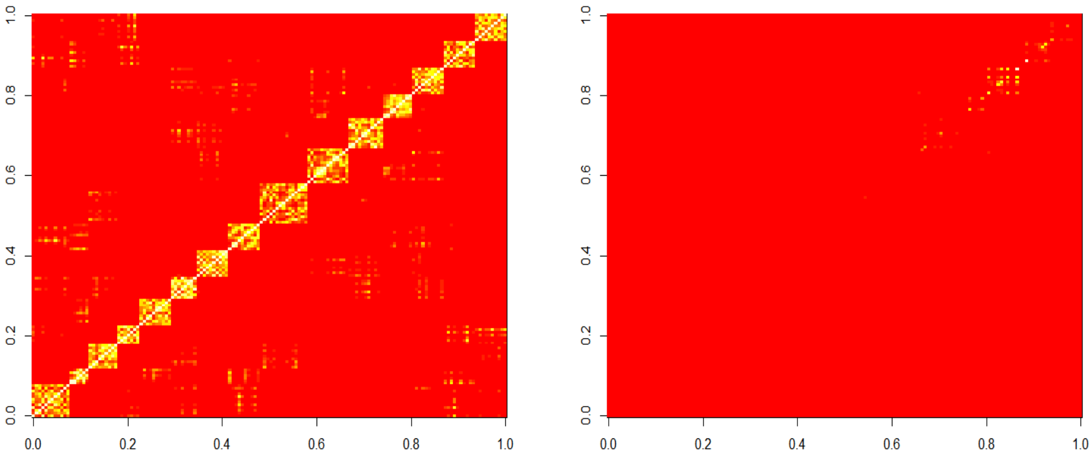


Figure 2.5.4: Snapshot from weighted spectral CluStream on S1.

The grey points are the data points observed so far, the location of the letters represent micro-cluster centres which are labelled with the results of the weighted spectral clustering. This is not a good clustering of the data. We can see that one cluster (letter N) is dominating and many of the outliers of the other clusters have been represented by the N cluster label.

This implies that the affinities between the micro-cluster centres on the outskirts of the cluster C are more similar to the outliers of other clusters (such as cluster I) than to the micro-cluster centres at C. This behaviour is very odd and implies that there might be an issue with the affinity matrix.

We can observe the affinity matrix by plotting it as an image, where bright values imply an affinity value close to one and red means the value is close to zero. Figure 2.5.5 shows an image of both the unweighted and weighted affinity for the example shown in Figure 2.5.4. The affinity matrix has dimension 150 × 150, with each row representing the similarities between one micro-cluster centre and the other 149. The rows and columns have been grouped so that micro-cluster centres from the same true underlying clusters are next to each other.



(a) Unweighted affinity matrix.

(b) Weighted affinity matrix.

Figure 2.5.5: Affinity matrices for S set 1.

We can see that in Figure 2.5.5a the block nature of the unweighted affinity is clear. There are strong affinities between close micro-clusters, and weak affinities between distant micro-clusters making this an informative affinity matrix to use with spectral clustering. However in the weighted affinity matrix (Figure 2.5.5b) the block nature is not visible. Most of the values are close to zero, with only a few strong affinities. The weighting seems to have dampened the affinity matrix, incorrectly reducing the affinity of close micro-clusters. It is possible that the use of the localised scaling parameter (see Section 2.2.2) in the spectral clustering step may be interfering with the weighting of micro-clusters. We did attempt to use 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 difficulty for 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.

2.5.5 Texture data

We now investigate the performance of the clustering algorithms 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 method (Eckley and Nason, 2011) which creates 27 wavelet features from the textured images.

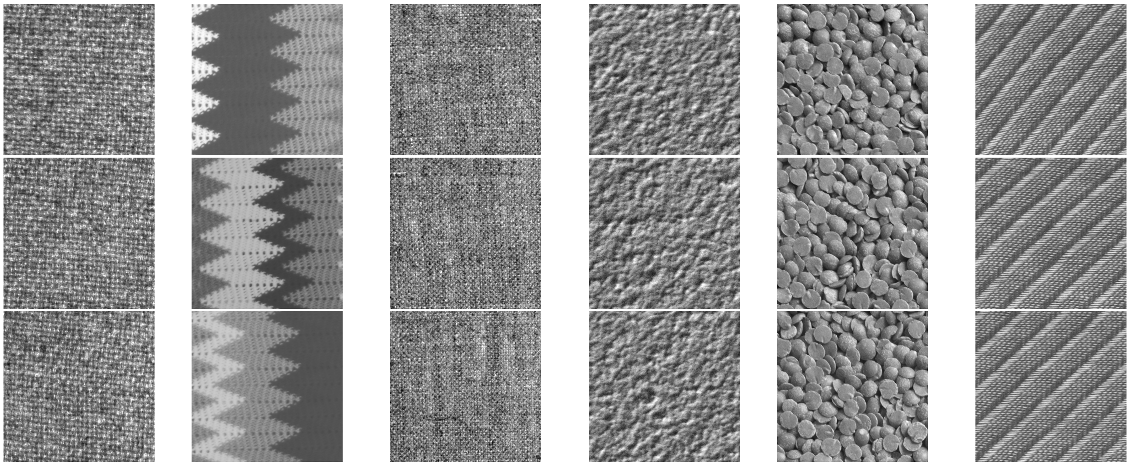
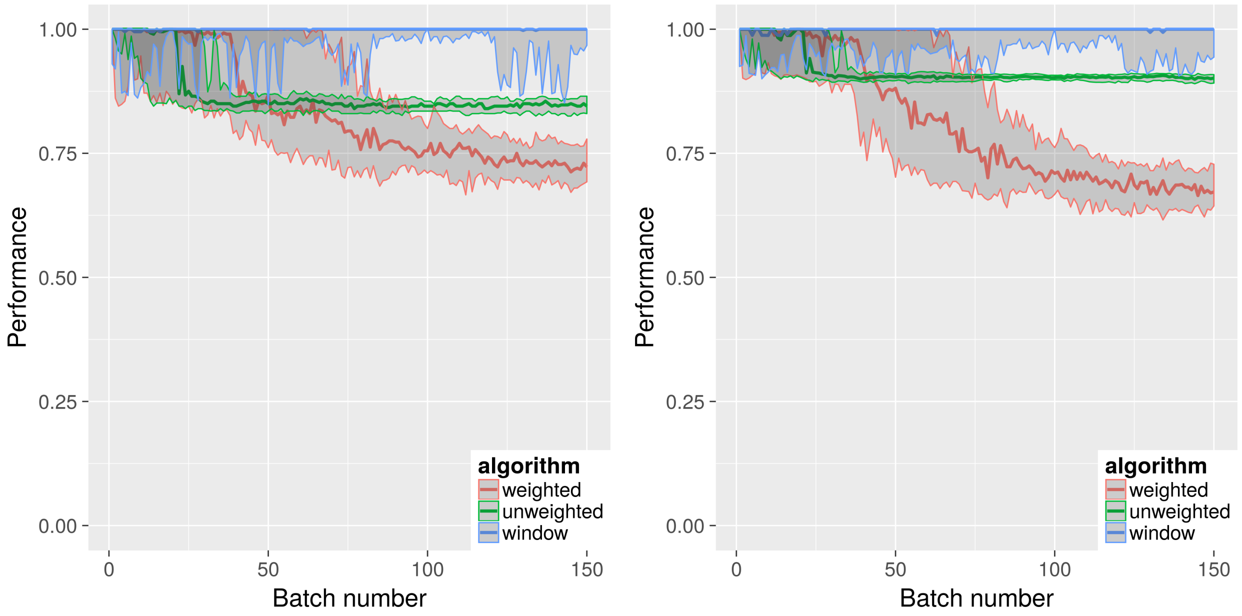


Figure 2.5.6: Three examples from each of the 6 different texture tiles. The texture classes are (L-R) Blanket 1, Blanket 2, Canvas, Ceiling, Lentils and Screen.

A subset of 6 classes was selected, examples of which are shown in Figure 2.5.6. 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 2.5.7. Here we do see a difference between windowed spectral clustering and unweighted spectral CluStream, the windowed approach is generally performing better. Once again weighted spectral 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. From now on, we will refer to unweighted spectral CluStream simply as spectral CluStream.



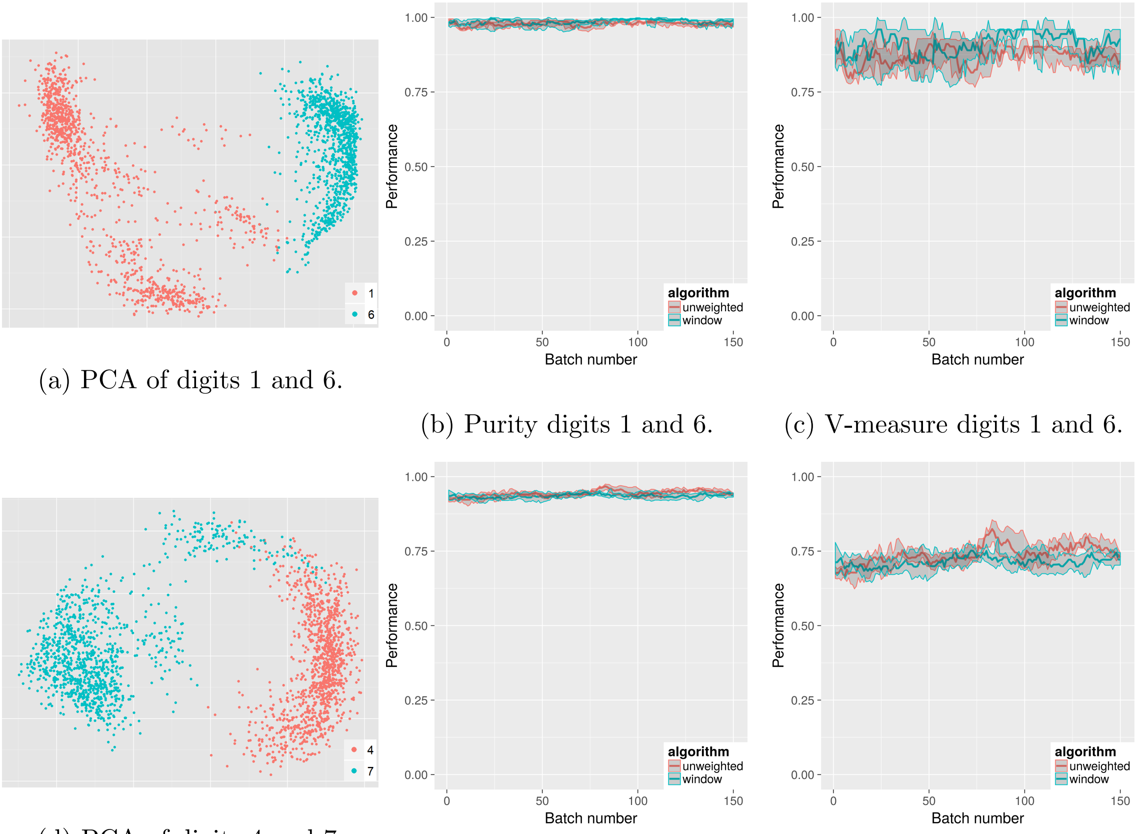
(a) Texture Purity.

(b) Texture V-measure.

Figure 2.5.7: Texture Results.

2.5.6 Pendigit data

This section and the next will use the UCI Pendigit data set 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.



(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 2.5.8: Pendigits Pairwise - spectral CluStream and windowed spectral clustering.

The results for a selection of the pairwise digits are shown in Figure 2.5.8. 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 2.5.2).

2.5.7 Non-stationary data

So far we have considered only stationary data streams. The main challenge for clustering algorithms for data streams is adapting to changes in the data stream. In order to create data streams with a non stationary distribution we introduce a change into the data stream. To construct a data stream we choose three 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.

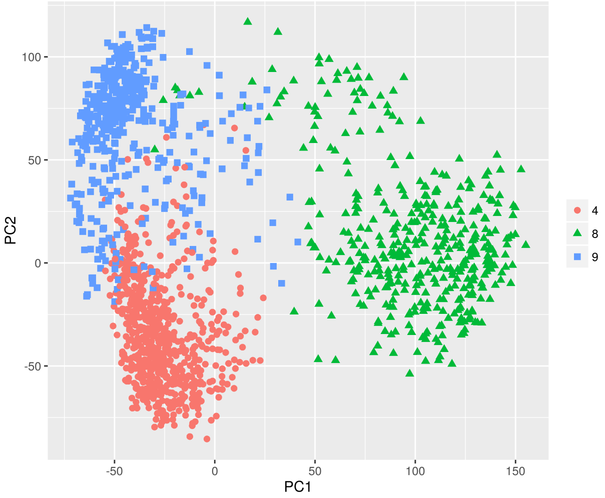
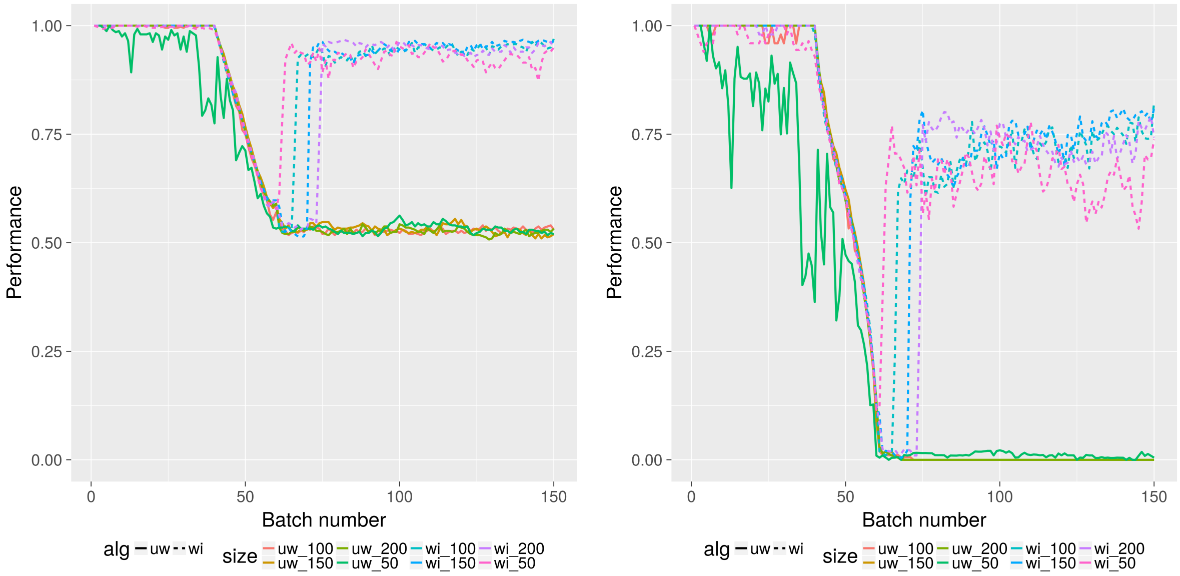


Figure 2.5.9: PCA plot for the Pendigits 4, 8 and 9.

The example set is “Pendigits 48 49” - first we observe features from digits 4 and 8, and we switch to observing features from digits 4 and 9 half way through the stream. Figure 2.5.9 shows the PCA space for all three digits. There is quite a bit of overlap between digits 4 and 9, which makes the data stream quite tricky to cluster. We ran spectral CluStream and windowed spectral clustering on this data stream. We have included a number of different values for the number of micro-clusters $q \in (50, 100, 150, 200)$ and also ran windowed spectral with window sizes $w \in (50, 100, 150, 200)$. The results in terms of purity and V-measure are shown in Figure 2.5.10. In Figure 2.5.10 the dashed lines show the windowed spectral clustering and the full lines are unweighted spectral CluStream. The colours indicate different values for the number of micro-clusters/window size.

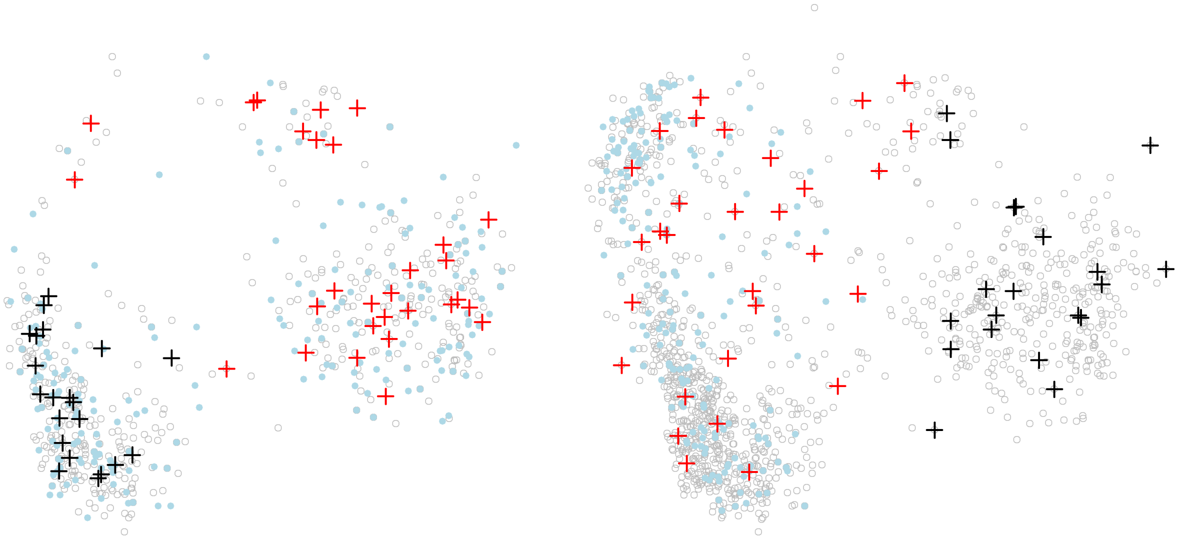


(a) Purity.

(b) V-measure.

Figure 2.5.10: Purity and V-measure for Pendigits 48-49.

The initial results show that spectral CluStream does not perform well after the change is observed. Both purity and V-measure drop dramatically low and do not recover. A fix is required in order for spectral CluStream to deal with the change in the data stream. In order to understand why performance drops at the change and discover how to fix this we need to look closely at the behaviour of the micro-clusters. Figure 2.5.11a shows the micro-clusters for 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 micro-cluster centres, and their colour indicates which overall cluster they have been assigned to using the spectral clustering algorithm.



(a) Stream at time t = 1.

(b) Stream at time t = 1000.

Figure 2.5.11: Micro-cluster centres for the Pendigits 48, 49.

In Figure 2.5.11a, at time step 1, the micro-cluster centres are well distributed over the grey data points and also the blue test points. The spectral clustering mostly segments the micro-cluster centres correctly into the left and right clusters.

At time step 1000 (Figure 2.5.11b), we have begun observing the new cluster (digit 9 in the top left corner) and are no longer receiving data points from 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 micro-cluster 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). The reason that there are still micro-cluster centres located in the cluster 8 region is because of the deletion policy that CluStream uses.

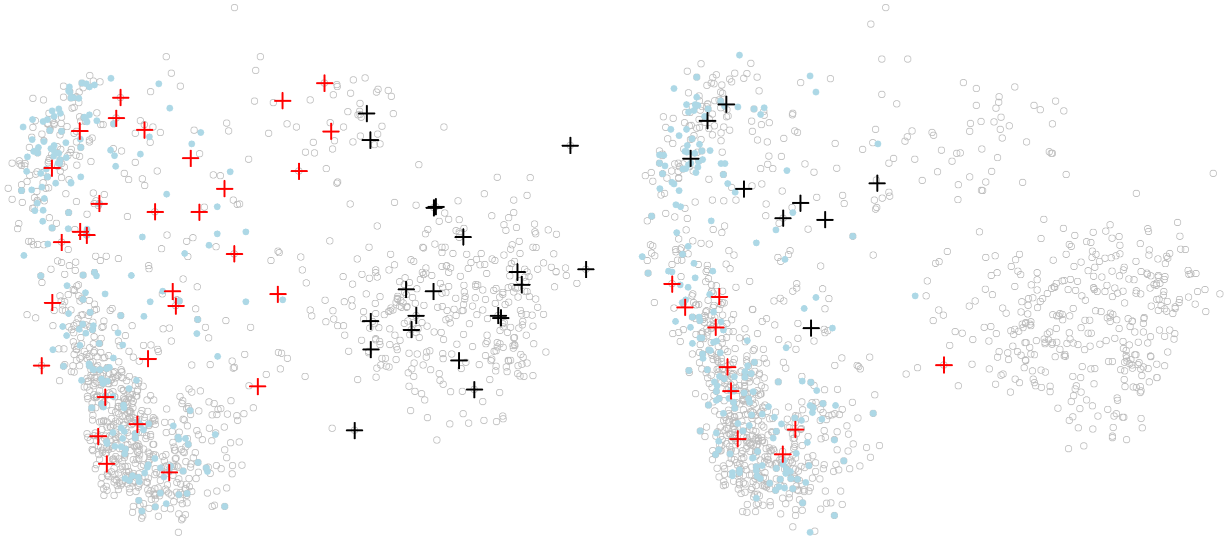
CluStream requires the number of micro-clusters to remain fixed for the duration of the stream. As discussed in Section 2.4, if an arriving data point does not have a suitable micro-cluster to merge into, a new micro-cluster is formed. However, since the total number of micro-clusters is fixed, this means that we need to either delete an old micro-cluster if it is suitably old, or combine two close ones. This is the only way that micro-clusters can be deleted in CluStream. A micro-cluster 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 (2.4.5).

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 discard old micro-clusters.

Often, storing old micro-cluster centres isn't a problem. Keeping old micro-cluster centres is a useful way to retain some historical data about the data stream. Also, in the case where a cluster disappears for some time and then re-emerges later in the stream, retaining old micro-cluster centres may speed up the learning when the old cluster re-emerges. 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 old micro-cluster centres are used in the spectral clustering step. By including these old centres in the spectral clustering, the algorithm technically has centres from three clusters (digits 4, 8 and 9), but we have asked the spectral clustering algorithm to find two clusters. In the example above in Figure 2.5.11b, 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 micro-cluster centres on the right of the plot.

The proposed solution to deal with these micro-cluster 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 micro-cluster centres to perform the spectral clustering. Figure 2.5.12 compares the previous method with the proposed alteration. Figure 2.5.12a shows the standard algorithm at t = 1000 (this is a duplicate of Figure 2.5.11b repeated for comparison purposes). Figure 2.5.12b shows the clustering of the micro-cluster centres when the alternative spectral CluStream is used.



(a) Standard spectral CluStream Algorithm.

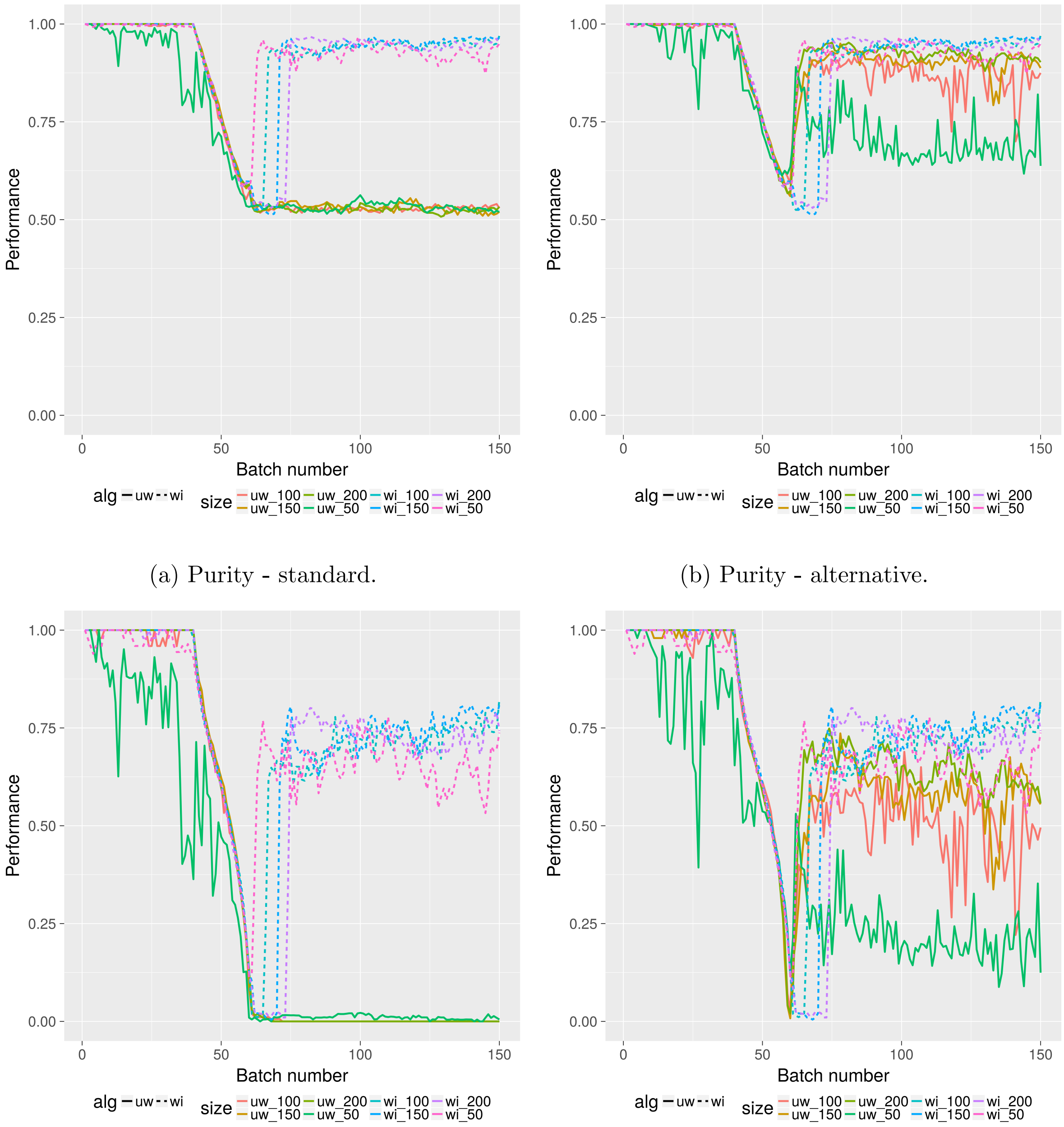
(b) Alternative spectral CluStream Algorithm.

Figure 2.5.12: Micro-cluster centres for the Evolving Pendigits 48, 49 at t = 1000.

We can see that the older micro-cluster 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 micro-cluster 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.

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 toy example above we 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 2.5.12b), 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 2.5.13 and 2.5.14 show the alternative spectral CluStream performance with the standard spectral CluStream performance. In the plots the dashed lines show the windowed spectral clustering and the full lines are unweighted spectral CluStream. 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.



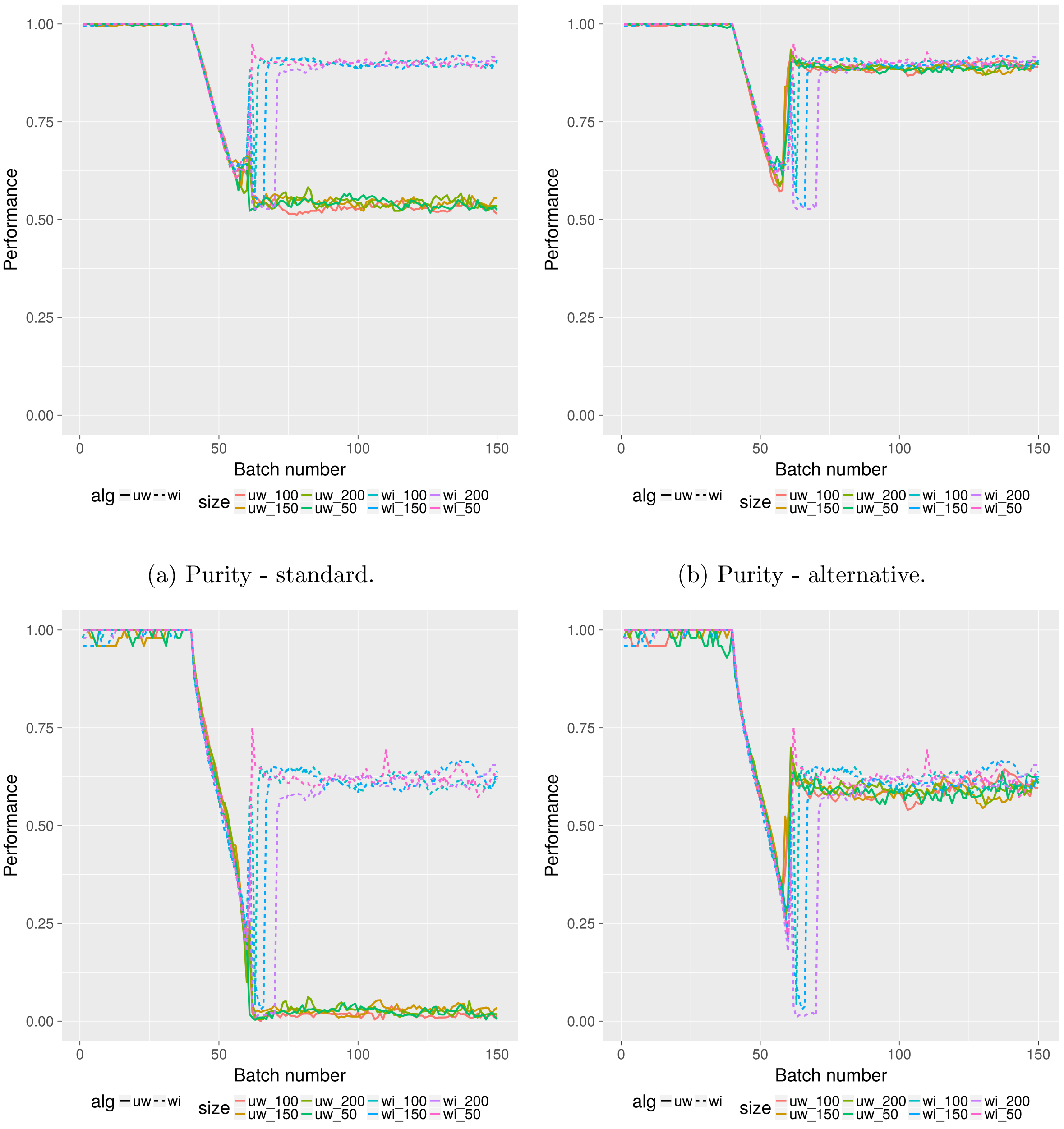
(a) Purity - standard.

(b) Purity - alternative.

(c) V-measure - standard.

(d) V-measure - alternative.

Figure 2.5.13: Standard vs alternative CluStream on Pendigits 48 49.



(a) Purity - standard.

(b) Purity - alternative.

(c) V-measure - standard.

(d) V-measure - alternative.

Figure 2.5.14: Standard vs alternative CluStream on Pendigits 34 37.

The plots show that although unweighted spectral CluStream was performing poorly when using the standard approach, using the alternative method has brought performance back up to being as good as windowed spectral clustering. In fact in both examples all spectral

CluStream algorithms outperforms windowed spectral clustering as it recovers faster after the change. This can be seen for example in Figure 2.5.14b where, after then change, the purity for all the spectral CluStream algorithms recovers almost immediately to 0.8 whilst the windowed algorithms take longer to reach that level of purity. The importance of the number of micro-clusters is evident in this data set. In Figure 2.5.13d 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 for 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 2.5.14b 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.

2.6 Conclusion

In this chapter we have presented spectral CluStream, a clustering method capable of performing spectral clustering on data streams. Under the suggestion from Zhang et al. (1996), we considered two variations of this algorithm, a weighted and an unweighted algorithm. Despite having a mathematically valid affinity matrix, the weighted spectral CluStream was found to have very poor performance and fundamental difficulties clustering even simple simulated data. The unweighted spectral CluStream was shown to have good performance on par with a windowed approach to spectral clustering. An issue with the unweighted spectral CluStream deletion policy was highlighted for non-stationary data streams where we saw historic micro-clusters being retained causing to the spectral clustering to fail. A fix was suggested in order to retain the good performance, but at the cost of using additional micro-clusters to track the stream.

The key difference between spectral CluStream and windowed spectral clustering is that the data summaries kept by CluStream can include historic information. We found that this could be a disadvantage in the non-stationary setting. However, this might not always be the case, for example, in cases where the non-stationarity is seasonal in nature, such as shopping transactions. In this situation, retaining historical information may improve performance. The effect of storing historic micro-clusters is worth investigating for a range of types of non-stationary data streams across different applications.

Chapter 3

Identifying corruption within acoustic sensing signals

3.1 Introduction

In Chapter 2 we introduced the CluStream (Aggarwal et al., 2003) algorithm and demonstrated how it could be used to create an online spectral clustering algorithm. In this chapter we apply CluStream to identify corruption within acoustic sensing signals. Distributed acoustic sensing (DAS) is a modern technique used to monitor oil flow at various depths throughout an oil well. DAS uses a fibre-optic cable to record vibrations at very high resolutions, up to 10000 observations a second. DAS is fairly cheap to implement and offers high frequency data, but unfortunately corruption can occur in the signal. Our challenge is to identify the locations in the signal where corruption occurs. Existing methods for detecting and removing interference in DAS signals involve using offline, univariate changepoint detection. However DAS signals are multivariate and require online processing. In this chapter we show that

CluStream provides an alternative approach to changepoints analysis to identify corruption within DAS signals.

3.2 Motivation

3.2.1 What is Distributed Acoustic Sensing?

Distributed Acoustic Sensing (DAS) is a technique which uses fibre-optic cables to measure vibrations travelling through the ground. DAS systems have recently become popular in the oil and gas industry and are used to monitor oil flow (Silkina, 2014; van der Horst et al., 2014) and to detect leaks in abandoned gas wells (Boone et al., 2014). When vibrations pass through the fibre-optic cable, they induce a change in the intensity of the reflection of the pulses of light being passed through the cable. This provides very high frequency data, potentially as high as 10kHz. It is also possible to collect this data at many different depths in the well simultaneously. Therefore DAS data has both high frequency and high dimensionality.

An example of DAS signal data is given in Figure 3.2.1. In the figure, each plot is a series obtained at a different depth within the oil well. We can see that there are some disturbances in the signal. Engineers refer to these disturbances as corrupted data and the challenge of this application is to locate where the data is corrupted. We are told that if corruption is observed at one depth then the effect is also likely to be observed at multiple other depths simultaneously. This is visible in Figure 3.2.1, particularly at time point $t = 3750$, where there is a big drop in the signal which occurs in all ten series.

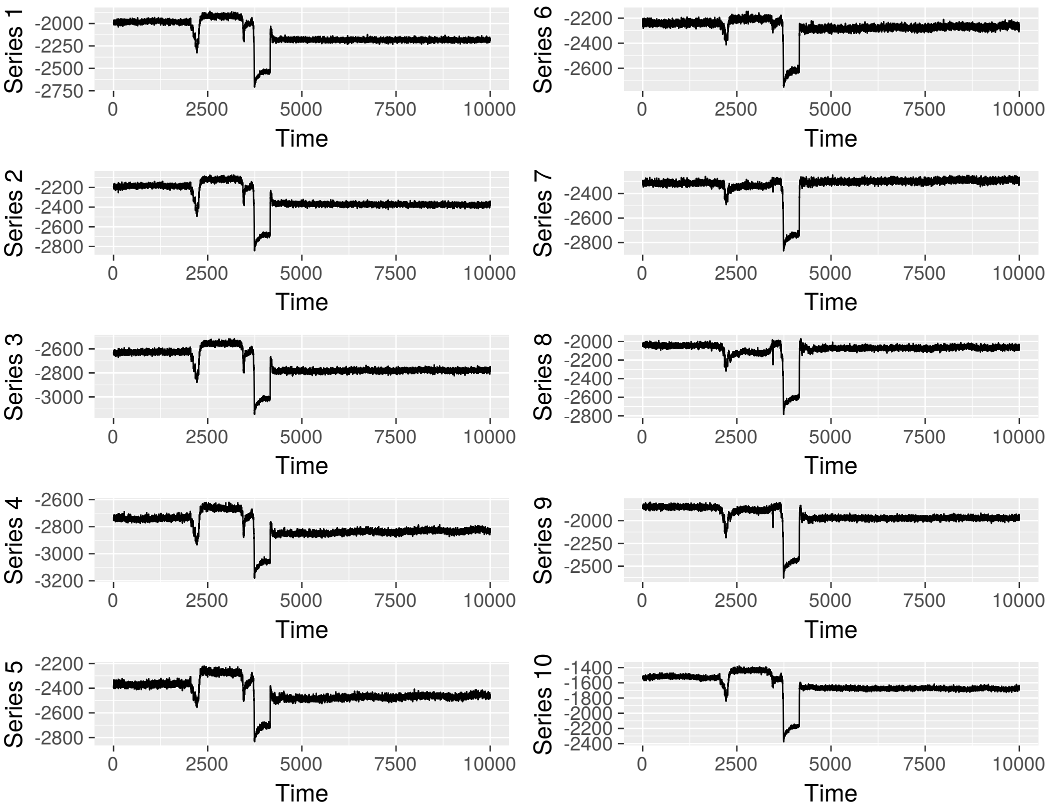


Figure 3.2.1: An example of acoustic sensing data observed at various depths in an oil well.

3.2.2 Relevant literature

Detecting corruption within a time series is usually framed as a changepoint detection problem. A changepoint is defined as a time-point at which a change occurs in one or more of the statistical properties of a time series. The first published article concerning changepoints was in Page (1954) which considered testing for a potential single changepoint and was motivated by a quality control setting in manufacturing. Over the decades, changepoint analysis has developed rapidly with multiple changepoints, different types of data and other assumptions being considered. Many methods for detecting changepoints exist ranging from approximate (heuristic) fast methods, to exact methods which take longer to run. A review of recent changepoint methods can be found in Chen and Gupta (2012); Eckley et al. (2011).

Much of the work in changepoint detection has focused on the scenario where the observations are univariate, although some extensions have been developed for the multivariate setting. The available changepoint algorithms which are multivariate cannot currently deal with the online scenario due to computational restraints. However due to the high frequency and dimensionality of DAS data, an online method is required.

3.2.3 Using CluStream to identify boundary locations

We consider the problem of identifying corruption within a DAS data stream as a two-stage clustering problem. The first stage is purely online, and consists of updating micro-clusters as a way of storing information about the data stream without storing all of the data points. The second stage is applied on a small, recent section of the data stream, and allows the user to request a segmentation of that section of the stream to look for where the signal is corrupted.

3.2.4 Stage one: Micro-clustering

Stage one is essentially the micro-clustering step of CluStream introduced in Section 2.4. CluStream is a method of clustering data streams, based on the concept of micro-clusters. Micro-clusters are data structures which summarise a set of instances from the stream, and are composed of a set of statistics which are easily updated and allow fast analysis. The number of micro-clusters used is a user chosen parameter. Using a large number of micro-clusters will represent the data stream better than a smaller number, at the cost of increased computation. We found using 250 micro-clusters to be sufficient for this application.

3.2.5 Stage two: Identifying corruption

Stage two is an offline procedure which is performed in batch on a recent section of the signal. This step uses the micro-cluster summaries to identify a set B of *boundary locations*, points in the signal where there is a change in the signal. First the k-means algorithm is applied on the micro-cluster centres. The clusters generated by the k-means step are referred to as macro-clusters. Then we consider the N most recently observed data points in the signal, $\{x_1, ..., x_N\}$. Each of these N points is assigned to a k-means macro-cluster using the nearest neighbour algorithm. We can now plot the signal coloured by the macro-cluster assignments. An example of this is given in Figure 3.2.2.

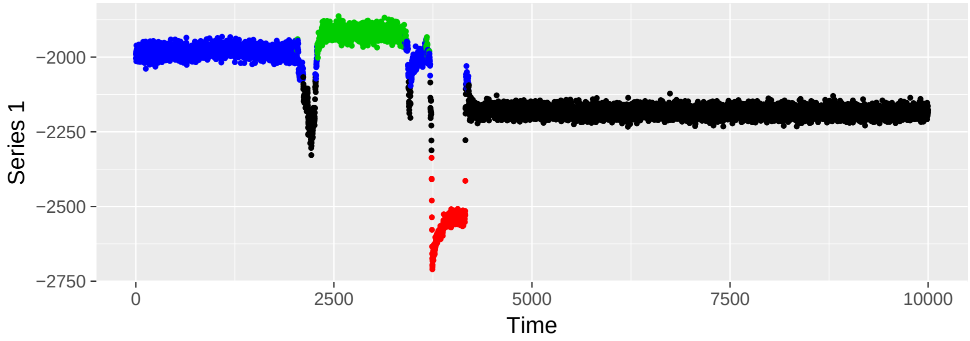
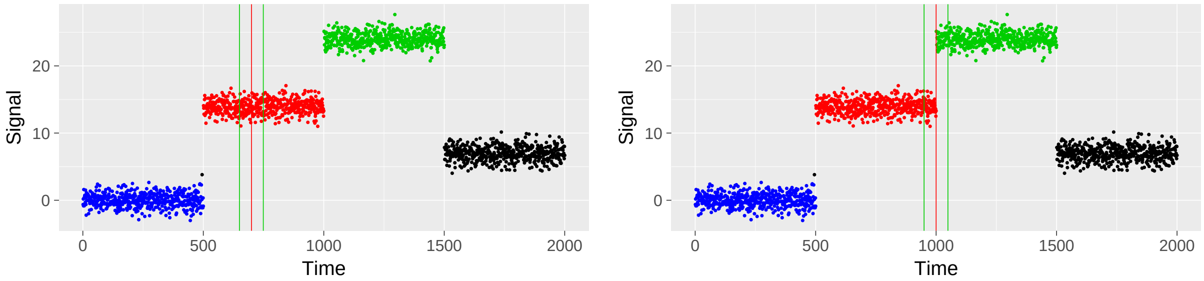


Figure 3.2.2: Series one of the DAS data coloured by macro-cluster assignments.

Informally, it appears that we can visualise where a change in structure occurs as indicated by the change in cluster assignment. In Figure 3.2.2 this is shown as a change in the colour of the signal. However, we would like to output a set of locations in time where change occurs. Note that during the clustering steps, we do not use the timestamps as input to the clustering. This means that we do not specifically tell the clustering to consider points closer in time as more similar. As a result, there is not necessarily a clear change in the cluster assignments. The method that we use to convert the clustering of assignments into a set of boundary locations is as follows. Consider a data point in $\{x_1, ..., x_N\}$, lets call it τ. In order to decide whether it is likely to be a boundary point we consider the k-means assignments of the data points directly before τ and directly after τ. If the assignments of those points are different, τ is likely to be a boundary location, if the assignments either side of τ are similar then τ is not likely to be a boundary location. The number of data points we look either side of τ is given by the search parameter γ. A simple example is shown in Figure 3.2.3.



(a) $\tau = 700$, $\gamma = 50$.

(b) $\tau = 1000$, $\gamma = 50$.

Figure 3.2.3: Example of searching for boundary locations on a simple change in mean example. The value of τ is given by the vertical red line, and the green lines show $\tau - \gamma$ and $\tau + \gamma - 1$.

In Figure 3.2.3a, $\tau = 700$ and we can see that the 50 points before τ are assigned to the same cluster as the 50 points after τ. This implies that τ is not a likely to be a boundary location. In Figure 3.2.3b, $\tau = 1000$ and we can see that the 50 points before τ are mostly different in colour to the 50 points after τ. This implies that τ is likely to be a good boundary location. In order to quantify this, we use a categorical similarity measure. To this end, we define the set L to be the set of points to the left of τ given by $L = \{x_{\tau-\gamma}, \ldots, x_{\tau-1}\}$. Similarly, we define the set R to be the set of points to the right of τ given by $R = \{x_{\tau}, \ldots, x_{\tau+\gamma-1}\}$. In order to calculate how similar the cluster assignments of sets L and R are we calculate the following similarity metric. Let $n_{L,j}$ be the number of data points in set L assigned to cluster j, where $j \in 1, \ldots, k$ and similarly for $n_{R,j}$. The categorical similarity metric is defined in equation (3.2.1). This metric was inspired by the χ^2 test statistic, although since the size of sets L and R are equal we can use a related, more intuitive comparison.

$$\text{sim}(\tau, \gamma) = \frac{\sum_{j=1}^{k} \min(n_{L,j}, n_{R,j})}{\gamma}. \tag{3.2.1}$$

Note that this categorical similarity measure will be bound between 0 and 1, where 0 indicates perfect dissimilarity and 1 indicates that the distribution of the sets is identical. We can think of this similarity measure as an indicator of how likely τ is to be a boundary location. If $\text{sim}(\tau, \gamma) = 0$ then τ is very likely to be a boundary location. We search over all values of $\tau \in \{\gamma + 1 : N - \gamma + 1\}$ and define the set of boundary locations B to be the values of τ which satisfy $\text{sim}(\tau, \gamma) = 0$. The whole procedure for Stage 2 is summarised in Algorithm 5.

The number of boundary locations identified given by $|B|$ will depend on the choice of k in the k-means clustering and the value of γ selected. Generally, the smaller the value of γ, the more boundary locations will be identified. By searching over a range of values of γ and k this will give engineers a number of possible options of boundary locations for their consideration. The effect of these parameters on performance is explored in the next section.

Algorithm 5 Stage Two: Identifying Boundary Locations

```
Input: Data points = $\boldsymbol{x_1} \dots \boldsymbol{x_N}$, micro-cluster centres, number of macro-clusters $k$, search parameter $\gamma$.
Output: A set of boundary locations $B$.
Set $B = \emptyset$.
Apply k-means on the micro-cluster centres.
Assign each data point in $\{\boldsymbol{x_1} \dots \boldsymbol{x_N}\}$ to a k-means macro-cluster.
for $\tau \in \{\gamma + 1 : N - \gamma + 1\}$ do
    Calculate $n_{L,j}$ and $n_{R,j}$ for all $j \in \{1, \dots, k\}$.
    Calculate $\text{sim}(\tau, \gamma) = \frac{\sum_{j=1}^{k} \min(n_{L,j}, n_{R,j})}{\gamma}$.
    if $\text{sim}(\tau, \gamma) = 0$ then
        $B = B \cup \tau$ .
    end if
end for
```

3.3 Results on DAS data

We now explore the use of CluStream with Algorithm 5 to identify the location of corrupted data within acoustic sensing data. The data stream we consider consists of 10000 data points (one second of data) and ten different series relating to different depths within the oil well. In order to compare performance of our algorithm we compare against a ground truth. The ground truth is shown in Figure 3.3.1 and consists of six manually chosen boundary locations.

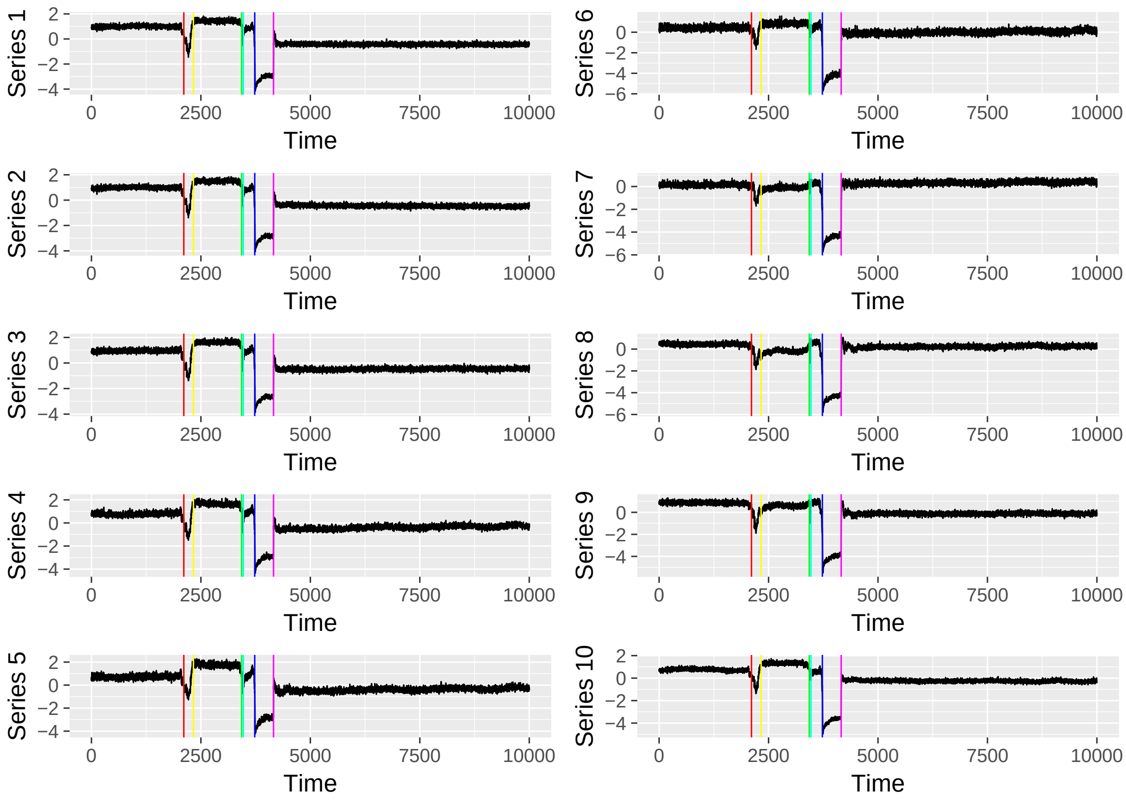


Figure 3.3.1: Ground truth for the DAS data shown for all ten series. The six true boundary locations are shown with coloured vertical lines.

We compare the boundary locations found by Algorithm 5 against the ground truth by using the V-measure. The V-measure (Rosenberg and Hirschberg, 2007) is a performance measure which rates the quality of a given segmentation compared to the ground truth segmentation. V-measure rates the segmentation by balancing both homogeneity and completeness. A larger V-Measure value indicates higher accuracy, with a value of 1 indicating a perfect segmentation. For full details, see in Section 2.5.2.

CluStream was applied on the DAS data stream using 250 micro-clusters. At $t = 10000$ we applied Algorithm 5 on the signal for a range of values of k and γ to identify boundary locations. Due to the randomness induced by the k-means step, the results for each experimental setting are averaged out over 10 runs. The variation between runs was minimal. The average V-measure for each setting is presented in Table 3.3.1.

k	γ									
	5	10	15	20	25	30	35	40	45	50
2	**0.742**	**0.742**	**0.742**	**0.742**	**0.742**	**0.742**	**0.742**	**0.742**	**0.742**	**0.742**
3	**0.967**	0.828	0.828	0.828	0.828	0.828	0.828	0.828	0.828	0.828
4	**0.956**	0.88	0.88	0.88	0.88	0.829	0.829	0.829	0.829	0.828
5	0.937	0.946	**0.961**	**0.961**	**0.961**	0.931	0.931	0.931	0.931	0.828
6	0.798	0.935	**0.961**	**0.961**	**0.961**	0.93	0.93	0.93	0.931	0.828
7	0.802	0.935	0.952	0.952	**0.961**	0.942	0.942	0.93	0.93	0.868
8	0.805	0.936	0.944	0.944	**0.961**	0.955	0.955	0.93	0.93	0.93

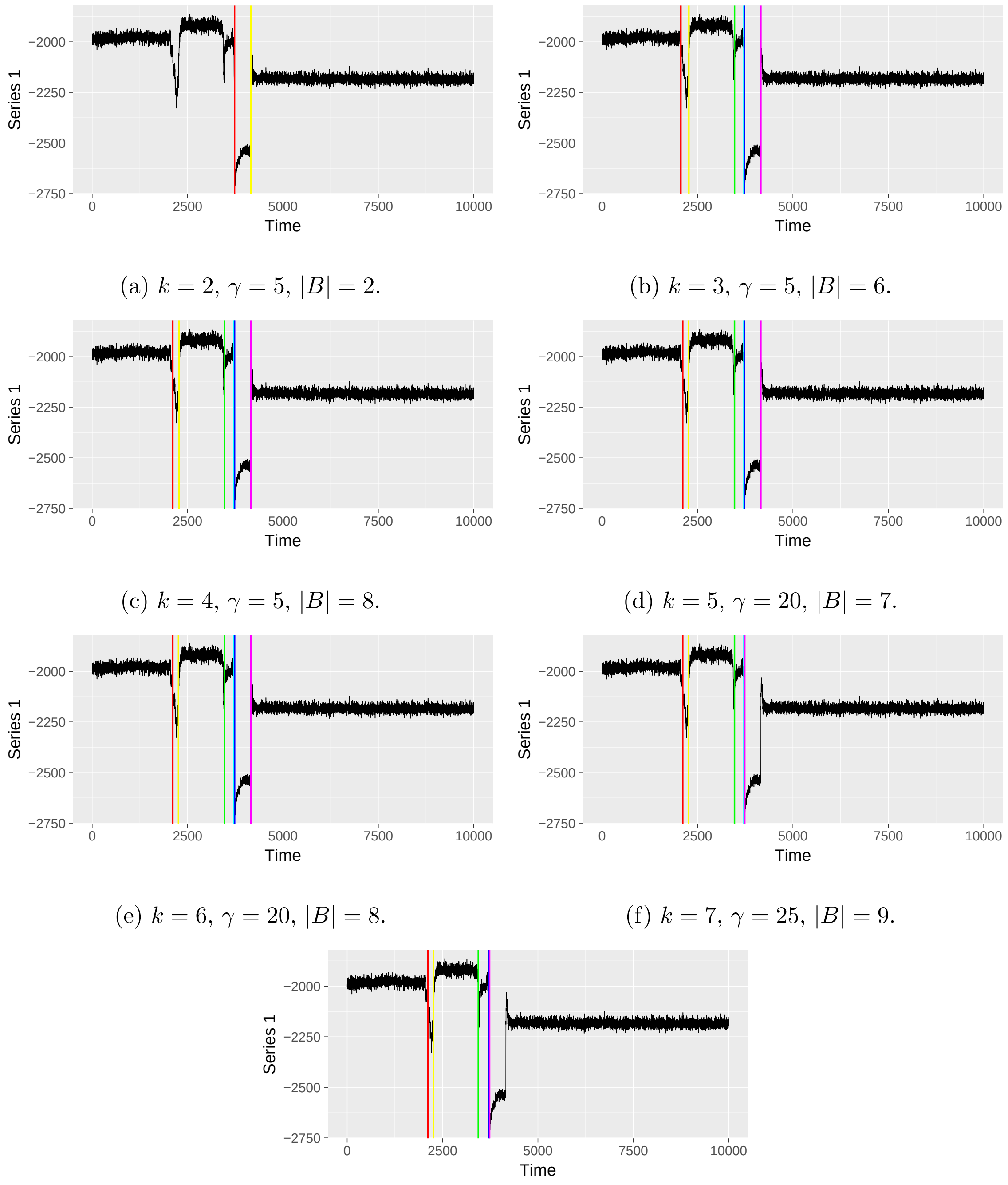
Table 3.3.1: V-measure results on the DAS data for a range of k and γ. The best performance for each value of k is highlighted in bold.

The best performance in terms of V-measure is given by $k = 3$, $\gamma = 5$ however, good performance is found across the settings. The choice of γ had no effect for the experiments where $k = 2$. Generally as k increases, the ideal choice of γ for that value of k also increases. This makes sense as when k increases, $\text{sim}(\tau, \gamma)$ is more likely to take values of 0, resulting in more boundary locations being identified. By choosing a larger γ for larger values of k, this prevents the number of boundary locations chosen by the algorithm from growing too large. The average number of boundary location identified by the algorithm under the different scenarios is given in Table 3.3.2.

k	γ									
	5	10	15	20	25	30	35	40	45	50
2	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
3	**6**	3	3	3	3	3	3	3	3	3
4	**8**	5	5	5	5	4	4	4	4	3
5	11	8	**7**	**7**	**7**	6	6	6	6	5
6	19	11	**9**	**8**	**8**	7	7	7	6	5
7	18.8	11.6	9.6	9	**8.6**	8	8	7.6	7	6.4
8	18.4	11.6	9.6	9.4	**8.6**	8.4	8.4	7.6	7.4	7.4

Table 3.3.2: Average number of boundary locations identified on the DAS data for a range of k and γ. The number of boundary locations corresponding to the best performing V-measure for each value of k is highlighted in bold. The ground truth has $|B| = 6$.

In Table 3.3.2 we can see that the the number of boundary locations found does vary with k and γ. However, the actual boundary locations identified are similar across the different parameter settings. In order to demonstrate this, Figure 3.3.2 plots the boundary locations identified using the most suitable value of γ for each value of k.



(a) $k = 2$, $\gamma = 5$, $|B| = 2$.

(b) $k = 3$, $\gamma = 5$, $|B| = 6$.

(c) $k = 4$, $\gamma = 5$, $|B| = 8$.

(d) $k = 5$, $\gamma = 20$, $|B| = 7$.

(e) $k = 6$, $\gamma = 20$, $|B| = 8$.

(f) $k = 7$, $\gamma = 25$, $|B| = 9$.

(g) $k = 8$, $\gamma = 25$, $|B| = 9$.

Figure 3.3.2: Each plot shows the boundary locations (coloured lines) identified under different parameter settings. We show one plot for each value of k, and use the most appropriate value of γ in each setting. The number of boundary locations identified is given by $|B|$.

Only plots for Series 1 are shown in Figure 3.3.2 here but similar performance was observed across all ten series. We can see that in Figure 3.3.2a, for $k = 2$ only two boundary locations are found but these manage to pick out the most obvious data corruption. As k increases, the number of boundary locations increases, but boundary locations only occur in places of change within the signal. For example, in Figure 3.3.2g we can see that $|B| = 9$ although we can only visibly see five red lines in the plot. This means that some of the boundary locations will be very close together, perhaps even neighbouring time points.

3.4 Conclusion

In this chapter we have demonstrated that data summaries constructed by CluStream can help provide an alternative to changepoint detection methods for identifying corruption in digital acoustic sensing signals. The fact that CluStream can be run efficiently online means that it can cope well with the high frequency and high dimensionality data created by digital acoustic sensing.

In order to frame the time-series as a clustering problem, we treated each time-series as a different data dimension. Interestingly the initial k-means clustering assignments look visually sensible in a temporal sense, despite no temporal information being given to the k-means algorithm.

We have developed an algorithm which uses a change in cluster assignments to identify boundary locations within the DAS signal. This algorithm was tested on the DAS data set for a range of values of k and γ and the results were found to be robust. As engineers are not clear exactly what defines a corrupted signal, offering a range of potential boundary locations by varying γ and k allows a set of solutions to be considered.

When detecting boundary locations, we considered only the time points τ where $\text{sim}(\tau, \gamma) = 0$. It would be possible to instead threshold $\text{sim}(\tau, \gamma)$ to increase the number of boundary locations identified by the algorithm. It would also be possible to consider $\text{sim}(\tau, \gamma)$ as a function of τ and treat identifying boundary locations as a local minima problem. However, we suspect that this would lead to many duplicate boundary locations being identified which would not be of use to the application.

Chapter 4

The Effect of Recovery Algorithms on Compressive Sensing Background Subtraction

This chapter appears in the form of a conference paper (Davies et al., 2013). Supplementary material is provided in Appendix A. This material includes a broader introduction to compressive sensing, a description of the conditions for choosing a stable measurement matrix and intuition for the orthogonal matching pursuit algorithm.

Background subtraction is a key method required to aid processing surveillance videos. Current methods require storing each pixel of every video frame, which can be wasteful as most of this information refers to the uninteresting background.

Compressive sensing can offer an efficient solution by using the fact that foreground is often sparse in the spatial domain. By making this assumption and applying a specific recovery algorithm to a trained background, it is possible to reconstruct the foreground, using only a low dimensional representation of the difference between the current frame and the estimated background scene.

Although new compressive sensing background subtraction algorithms are being created, no study has been made of the effect of recovery algorithms on performance of background subtraction. This is considered by applying both basis pursuit and orthogonal matching pursuit (OMP) to a standard test video, and comparing their accuracy.

4.1 Introduction

Surveillance cameras have become ubiquitous in many countries, constantly collecting a huge amount of data most of which is stored and never analysed. This is due to the challenge of converting a large volume of video data into useful information, particularly as several cameras may be acquiring data simultaneously. One of the main aims of collecting surveillance footage is to track an object or classify its behaviour, therefore the first step in video analysis is to identify the objects of interest from the background.

Background subtraction (Piccardi, 2004) is a method used to separate the foreground from the background of a video sequence. This consists of constructing and updating a model of the background and then subtracting it from the current frame. Background subtraction is not a new development and many methods for modelling a background scene exist, however traditional background subtraction methods are not very efficient. Most surveillance footage consists of a slowly adapting background scene with foreground appearing in a subset of the frames and when foreground does appear in the footage it usually takes up only a small percentage of the overall frame. Traditional background subtraction methods require that all pixels are acquired for each frame in order to correctly segment the two layers, however since foreground is sparse in most surveillance footage this can be seen as a waste of resources.

In order to combat this inefficiency, the single pixel camera (SPC) (Duarte et al., 2008a) was developed; a camera based on the theory of compressive sensing (Candès and Tao, 2006; Candès et al., 2006; Donoho, 2006) and designed to acquire images directly in compressed format. Once surveillance footage is acquired using a camera such as the SPC, background subtraction can be performed on this low dimensional representation of the video frames. Later a recovery algorithm is used to decode a mask of the foreground into the correct dimension. The reason for using this technique is, due to the natural sparsity property of foreground, a mask can be reconstructed without ever storing the current frame, or background model in the full dimensional form. The computational savings available are significant, provided that the segmentation performed is accurate enough for the purpose of the application.

This paper discusses if the choice of recovery algorithm affects the performance of compressive sensing background subtraction. In particular two algorithms are investigated, basis pursuit (Candès and Tao, 2005) which is an algorithm based on convex optimisation (Boyd and Vandenberghe, 2004) and a greedy algorithm called orthogonal matching pursuit (Tropp, 2004). Results are presented from both algorithms on data which was gathered using a conventional camera but which have been simulated to mimic the acquisition process of compressive sensing imaging technology such as the SPC.

The rest of this paper is organised as follows; Section 4.2 discusses the related works in this area in brief before discussing the methodology behind compressive sensing and its application to background subtraction in Section 4.3. Experimentation is conducted in Section 4.4 and conclusions are given in Section 4.5.

4.2 Related Works

There is a vast amount of research available in the literature detailing the many techniques for background subtraction; notable comparative studies include Cheung and Kamath (2004) and Piccardi (2004). Generally, algorithms for background modelling can be categorised into pixel based and region based methods (Bouwmans, 2011). The latter may seem more intuitive as one expects foreground to be clustered and generally not exist as isolated pixels. Using knowledge of neighbouring pixels should therefore improve the classification of pixels into foreground or background, and region based techniques such as Elgammal et al. (2000) and Toyama et al. (1999) do use this knowledge to segment the images into regions and then create a background model based on these regions. However, there are downsides to region based techniques, for example they are often a lot more complex to run than pixel based techniques, and are not generally robust to division of blocks.

Pixel-based methods such as approximate median filtering (McFarlane and Schofield, 1995) assume that the pixels observed are classified as foreground independently of each other. These methods can often detect the contours of foreground well but may be susceptible to making false classifications, especially if the background model is not well tuned.

It is also possible to categorise these algorithms as recursive and non-recursive methods (Bouwmans, 2011). A non-recursive algorithm maintains a buffer or window of N previous video frames and estimates a background model based on the statistical properties of these frames. This sliding window is usually required to be fairly large in order to obtain accurate results, and therefore can quickly become computationally intensive. Recursive techniques update the background subtraction each time a new frame is stored and so there is no need to buffer previous frames. The method used in Section 4.3.4 to model the background is a pixel-based, recursive method.

More recently the new technology of compressive sensing (Candès and Tao, 2006; Candès et al., 2006; Donoho, 2006) has been applied to the background subtraction problem. The theory of compressed sensing states that an under-sampled signal can be recovered almost perfectly, given that the signal itself is sparse in some domain (Baraniuk, 2007), i.e. a large proportion of the signal's elements are close to zero. In compressive sensing for video, a random matrix is used to encode a signal efficiently, and then a recovery algorithm is used to decode the signal and recover the required information fairly accurately.

Most compressed sensing background subtraction methods (Cevher and Duarte, 2008; Cevher and Sankaranarayanan, 2008; Warnell and Chellappa, 2011; Cossalter et al., 2009) make the assumption that the majority of the pixels in a video represent the background, and therefore the foreground is sparse in the spatial domain. This key point allows us to encode a video into a low dimension and then search for a sparse solution, using the knowledge that the foreground mask should be suitably sparse.

The background subtraction problem is first expressed as a sparse signal recovery problem in Cevher and Sankaranarayanan (2008). Their work successfully recovers silhouettes of foreground activity by modelling a compressed form of the background and recovering a foreground mask directly from compressed measurements. Later Warnell and Chellappa (2011) builds upon this work by proposing a method that adapts the encoding process between frames to incorporate the changing size of foreground. Unlike Cevher and Sankaranarayanan

(2008), Warnell and Chellappa (2011) choose to use a static model for the background which could cause problems in the model when applying to real-life video.

More current work has started attempting to incorporate the structure of foreground into segmentation. It is generally expected that foreground will be clustered into particular shapes in most frames and does not consist of isolated pixels. Foreground often consists of humans, animals and vehicles, and so knowledge of this structure could be used to improve segmentation. Currently many different approaches are being investigated such as applying particle filters (Cossalter et al., 2009) and lattice based graphical models (Cevher and Duarte, 2008). Although cluster type methods are not attempted in this work, it is a research area of interest for future investigation.

In this work, two recovery algorithms are compared, one greedy method and one based on convex optimisation when applied to a background subtraction algorithm.

4.3 Methodology

In this section, the main theory is introduced, starting with the explanation of sparse signals in Section 4.3.1, progressing to the theory of compressive sensing in Section 4.3.2 with a focus on the two recovery algorithms of interest in Section 4.3.3 and finally notes on applying this theory to the background subtraction problem in Section 4.3.4.

4.3.1 Sparse and Compressible Signals

A signal is known as being K-sparse if $\boldsymbol{x} \in R^N$ can be represented as a linear combination of K basis vectors (Baraniuk, 2007). The case of interest occurs when K is much smaller than

N as this means that most of the coefficients are zero. No signal is ever truly sparse in the presence of noise and so some signals are described as approximately sparse, or compressible. If a signal is compressible there exist K large coefficients (with $K \ll N$) but the remaining $N - K$ coefficients are only required to be small and not necessarily zero. Fundamentally a signal is compressible if most of the information in the signal is represented by a few coefficients.

4.3.2 Compressive Sensing

According to the framework developed in Candès and Tao (2006), Candès et al. (2006) and Donoho (2006), the measurement $\boldsymbol{y}$ is a linear function of the signal $\boldsymbol{x}$ as shown in equation (4.3.1). The number of measurements M in $\boldsymbol{y}$ is chosen to be smaller than N, so a measurement matrix $\mathbf{\Phi} \in R^{M \times N}$ is chosen, with $M \ll N$. Although it is known from linear algebra that there are infinitely many vectors $\boldsymbol{x}$ that can solve equation (4.3.1), the knowledge that the original signal $\boldsymbol{x}$ was sparse can be used to help the reconstruction process, given that certain conditions hold for $\mathbf{\Phi}$.

$$\boldsymbol{y} = \mathbf{\Phi}\boldsymbol{x}. \tag{4.3.1}$$

It is of vital importance that a stable measurement matrix $\mathbf{\Phi}$ is designed so that the signal information is not damaged by the dimensional reduction from $\boldsymbol{x} \in R^N$ to $\boldsymbol{y} \in R^M$. In order for the reconstruction problem to be well-conditioned, it is sufficient that $\mathbf{\Phi}$ holds the Restricted Isometry Property (RIP) (Donoho, 2006) of order $2K$.

Definition 1. *A matrix $\mathbf{\Phi}$ satisfies the (RIP) of order K if there exists a $\delta_K \in (0, 1)$ such* *that*

$$(1-\delta_k)\|\boldsymbol{x}\|_2^2 \leq \|\boldsymbol{\Phi x}\|_2^2 \leq (1+\delta_k)\|\boldsymbol{x}\|_2^2,$$

for all $\boldsymbol{x} \in \sum_K = \{\boldsymbol{x} : \|\boldsymbol{x}\|_0 \leq K\}$*,*

where $\|\boldsymbol{x}\|_0$ *is the zero pseudo-norm defined as*

$$\|\boldsymbol{x}\|_0 = \#(i|x_i \neq 0).$$

If $\boldsymbol{\Phi}$ satisfies the RIP with order $2K$, then $\boldsymbol{\Phi}$ approximately preserves the distance between any pair of K-sparse vectors. Unfortunately the task of checking that a matrix satisfies the RIP is a NP-hard problem, but fortunately the RIP will hold true with high probability if $\boldsymbol{\Phi}$ is selected as a random matrix (Candès et al., 2006), and $M \geq cK \log \frac{N}{K}$, where c is a small constant.

4.3.3 Recovery Algorithms

Right at the heart of the compressive sensing theory is the ability for recovery algorithms to provide accurate signal estimations in an efficient manner. Recovery algorithms generally fall into two categories, those based on convex optimisation and greedy algorithms. One of the basic algorithms from each category are considered.

Convex Optimisation

Convex optimisation is a minimisation problem subject to a number of constraints where the functions involved are convex. According to Baraniuk (2007), optimisation based on the ℓ_1

norm can exactly recover K-sparse signals and closely approximate compressible signals with high probability using only $M \geq cK \log \frac{N}{K}$ iid Gaussian measurements. The ℓ_1 norm of a vector $\boldsymbol{x}$ is the sum of the absolute values of the elements of $\boldsymbol{x}$ and is defined mathematically in equation (4.3.2).

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{N} |x_i|. \tag{4.3.2}$$

The use of ℓ_1 minimisation to promote sparsity is not a new idea, the links between ℓ_1 minimisation and sparsity were first noted in the field of geophysics when it was observed (Claerbout and Muir, 1973) that minimising the ℓ_1 norm could help detect sparse spike trends in earthquake data. The idea soon caught on, Taylor et al. (1979) utilised ℓ_1 methods in their work on extracting spike trains and eventually in 1996, ℓ_1 minimisation was approached in statistics as Least Absolute Shrinkage and Selection Operator (LASSO) (Tibshirani, 1996). LASSO is a version of least squares with a constraint that the ℓ_1 norm, $\|x\|_1$ cannot be larger than some value ϵ. This penalty encourages more parameters to become zero as it increases, therefore promoting sparsity. Basis pursuit was developed in Chen et al. (2001) and is a type of ℓ_1 minimisation algorithm defined as

$$\min_{\boldsymbol{x}} ||\boldsymbol{x}||_1 \text{ subject to } \boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{x}. \tag{4.3.3}$$

This can be viewed as a least squares problem with an ℓ_1 regularizer. Basis pursuit is considered to have polynomial complexity but in reality this is not always true for standard optimisation packages as they are not tailored for sparse signal recovery.

In Section 4.4 the ℓ_l magic (Candès and Tao, 2005) implementation of basis pursuit is applied, which solves equation (4.3.3) by recasting the problem as a primal dual algorithm based on work by Boyd and Vandenberghe (2004). The Newton-Raphson method is then applied in order to detect an interior point at which the system is linearised and solved.

Greedy Algorithms

A greedy algorithm iteratively makes decisions based on some locally optimal solution. One of the simplest greedy algorithms suitable for the sparse signal approximation problem is orthogonal matching pursuit. Some of the earliest orthogonal matching pursuit algorithms can be found in Pati et al. (1993); Davis et al. (1997), although notable more recent work can be found in Tropp (2004) and Tropp and Gilbert (2007). This method can often perform faster than basis pursuit due to its simplicity. It iteratively computes the local optimum solutions in the hope that these will lead to the global optimum solution. The algorithm determines the column of $\boldsymbol{\Phi}$ which is most correlated with $\boldsymbol{y}$, or which contributes to $\boldsymbol{y}$ most. This is repeated again by comparing the correlation between columns of $\boldsymbol{\Phi}$ with the signal residual, until it reaches some stopping criterion defined by the user. This can be described algorithmically as in Algorithm 6.

In the algorithm, $\hat{\boldsymbol{x}}$ is updated for each iteration with the contributions from the columns of $\boldsymbol{\Phi}$ placed in the indexes stored in Λ_t. So if the algorithm is stopped after the Tth iteration for some positive integer T, $\hat{\boldsymbol{x}}$ will be a T-sparse vector. This emphasises how important it is to run the algorithm for the correct number of iterations, although generally this number is unknown.

Using greedy algorithms can be advantageous for the background subtraction problem as they can be both flexible and speedy. Greedy algorithms are less restrained to a particular

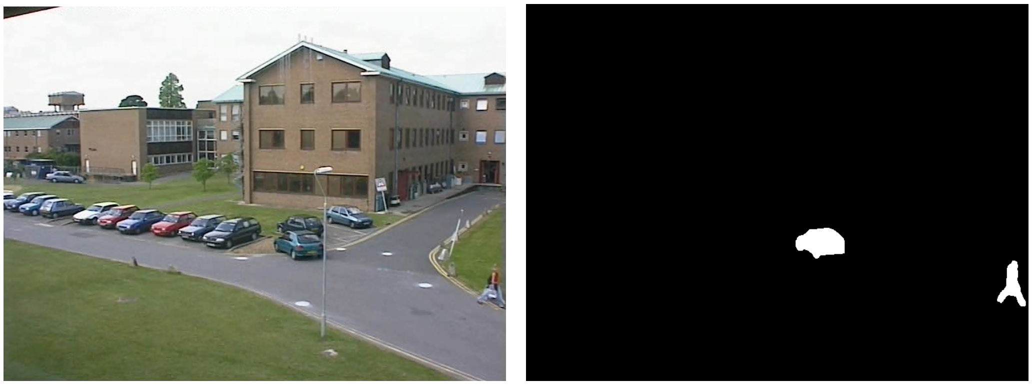
Algorithm 6 Orthogonal Matching Pursuit

```
  Define the columns of $\mathbf{\Phi}$ to be $\varphi_1, \varphi_2, \ldots, \varphi_N$.
Input: $\boldsymbol{r_0} = \boldsymbol{y}, \Lambda_0 = \emptyset$ and iteration counter $i = 1$.
  for $i < T$ do
    $\lambda_t = \text{argmax }_{j=1,\ldots,N} | < r_{t-1}, \varphi_j > |$.
    {Find the index for the column of $\Phi$ with the greatest contribution.}
    $\Lambda_t = \Lambda_{t-1} \cup \lambda_t$, $\Phi_t = [\boldsymbol{\Phi_{t-1}}, \varphi_{\lambda_t}]$.
    {Keeps track of the columns used.}
    $\boldsymbol{x_t} = \text{argmin}_{\boldsymbol{x}} ||\boldsymbol{y} - \boldsymbol{\Phi_t x}||_2$.
    {Updates the signal estimate.}
    $\boldsymbol{r_t} = \boldsymbol{y} - \boldsymbol{\Phi_t x_t}$.
    {Updates the measurement residual.}
  end for
  return $\hat{\boldsymbol{x}}$.
```

form than in ℓ_1 minimisation, therefore greedy algorithms can incorporate constraints which do not fit naturally in a convex formulation. Also, when the signal $\boldsymbol{x}$ is exceptionally sparse, only a few iterations are required, which makes the whole process very fast. Tropp and Gilbert (2007) claims that OMP can recover a K-sparse signal of size N by making only $\mathcal{O}(k \ln N)$ observations of the signal. However, the relationship between the sparsity of the foreground mask and the choice of the stopping criterion means that unless the stopping criterion is adaptive, the algorithm may struggle with varying sparsity as discussed in Section 4.4. As the signal length and number of measurements increases, so does the computational complexity - particularly in the identification stage. In this stage, the algorithm attempts to find the column φ of $\boldsymbol{\Phi}$ which is most strongly correlated with the residual. As the number of iterations increases, the sparsity of the signal estimate $\hat{\boldsymbol{x}}$ decreases. Therefore iterating for too long ($T \ll K$) not only leads to a poor estimate but unnecessary computational time.

4.3.4 Background Subtraction with Compressive Sensing

A major assumption when applying compressive sensing techniques to foreground segmentation is the assumption that the foreground is sparse in the spatial domain (Cevher and Sankaranarayanan, 2008). This means that without having to apply any sort of sparsifying transformation, the foreground only takes up a small percentage of the number of pixels in the frame. Figure 4.3.1 shows a test frame and the true foreground or “Ground Truth” where white pixels represent the true foreground and black pixels represent the true background. The number of white pixels (3862) is much smaller than the number of black pixels (438,506) therefore indicating that the foreground is indeed sparse in the spatial domain. The precise sparsity of the foreground will inevitably vary in different videos and even between frames



(a) Test frame.

(b) Ground truth.

Figure 4.3.1: The spatial sparsity of foreground. A frame from the PETS data set (University of Reading, 2001) and the corresponding foreground in white. In this example, less than 1% of the frame is foreground, as N=442,368 and K=3862.

as discussed in Warnell and Chellappa (2011).

Our method takes a vectorised form of the current frame $\boldsymbol{x_t}$, and acquires compressive measurements $\boldsymbol{y_t}$ of the frame using a Gaussian random matrix $\boldsymbol{\Phi}$. The foreground mask $\boldsymbol{x_t^f}$ is then reconstructed by applying a recovery algorithm to $(\boldsymbol{y_t} - \boldsymbol{y_t^b})$, where $\boldsymbol{y_t^b}$ is a compressed model for the background at time t. The silhouette is then thresholded to set any small values to zero, and the non-zero pixels are classified as foreground.

In order for this method to work well, it is important that a good model of the background is kept updated. Although a static background could be used for short indoor sequences, most real-world video sequences require a dynamic background model. Piccardi (2004) suggests that a background model should be able to adapt to deal with illumination changes, high frequency repetitive background objects and changes in background geometry. Illumination changes can be separated into gradual changes, or rapid changes. Gradual changes in illumination are mainly due to the light changing throughout the day. Rapid changes in illumination may occur when clouds cover the sun in outdoor scenes or a light being switched on or off in interior scenes. High frequency background objects are mainly found in outdoor footage such as leaves waving in the wind, water rippling or rainy weather. These small movements are actually part of the background so care should be taken to use an algorithm which detects them as such. Changes in background geometry refers to when part of the background begins to move. Examples of this are very common in real videos, such as a chair being repositioned in a room such as in Toyama et al. (1999) or a car driving into a car park as foreground, and then parking therefore becoming part of the background. In this report the background is modelled using an exponentially weighted moving average method as discussed in Piccardi (2004) and Cossalter et al. (2009). This is calculated as in equation (4.3.4).

$$\boldsymbol{y_t^b} = \alpha \boldsymbol{y_t} + (1 - \alpha)\boldsymbol{y_{t-1}^b}. \tag{4.3.4}$$

where $\alpha \in (0, 1)$ is a learning parameter. This background model was chosen, as it is not computationally heavy, and the single learning parameter α keeps tuning the model simple. The initial background $\boldsymbol{y_0^b}$ is calculated by taking an average of scenes from a training set. If this information is not available it is possible to use only the first frame in the data set, but this may not be as accurate. The background model learning parameter α is tuned so that it is sensitive to the challenges discussed above, specifically to avoid the change in geometry problems. This challenge is especially prominent in the experimentation in Section 4.4 as the video used is CCTV footage from a car park.

4.4 Performance Evaluation

The PETS 2001 data set “camera1” University of Reading (2001), is used to compare implementation of the CS_{ℓ_1} and $\mathrm{CS}_{\mathrm{OMP}}$ algorithms. A quantitative way to characterise the accuracy of these algorithms is to use two performance metrics; precision and recall. These measures calculate how accurate the segmentation is for a particular frame by comparing the estimated foreground and background with the true values calculated by a hand-segmented ground truth.

In order to quantify how well an algorithm is working, Type I errors and Type II errors must be considered. A Type I error is a false positive (FP), this is experienced when the algorithm classifies a pixel as foreground incorrectly. A Type II error is a false negative (FN), this where the algorithm has neglected to classify a pixel as foreground correctly. If the algorithm works perfectly there will only be True Positives (TP) which are correctly identified foreground pixels and True Negatives (TN), correctly identified background pixels. Recall is defined as the fraction of correctly identified foreground pixels over the number of ground truth foreground pixels which can be written mathematically as

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{4.4.1}$$

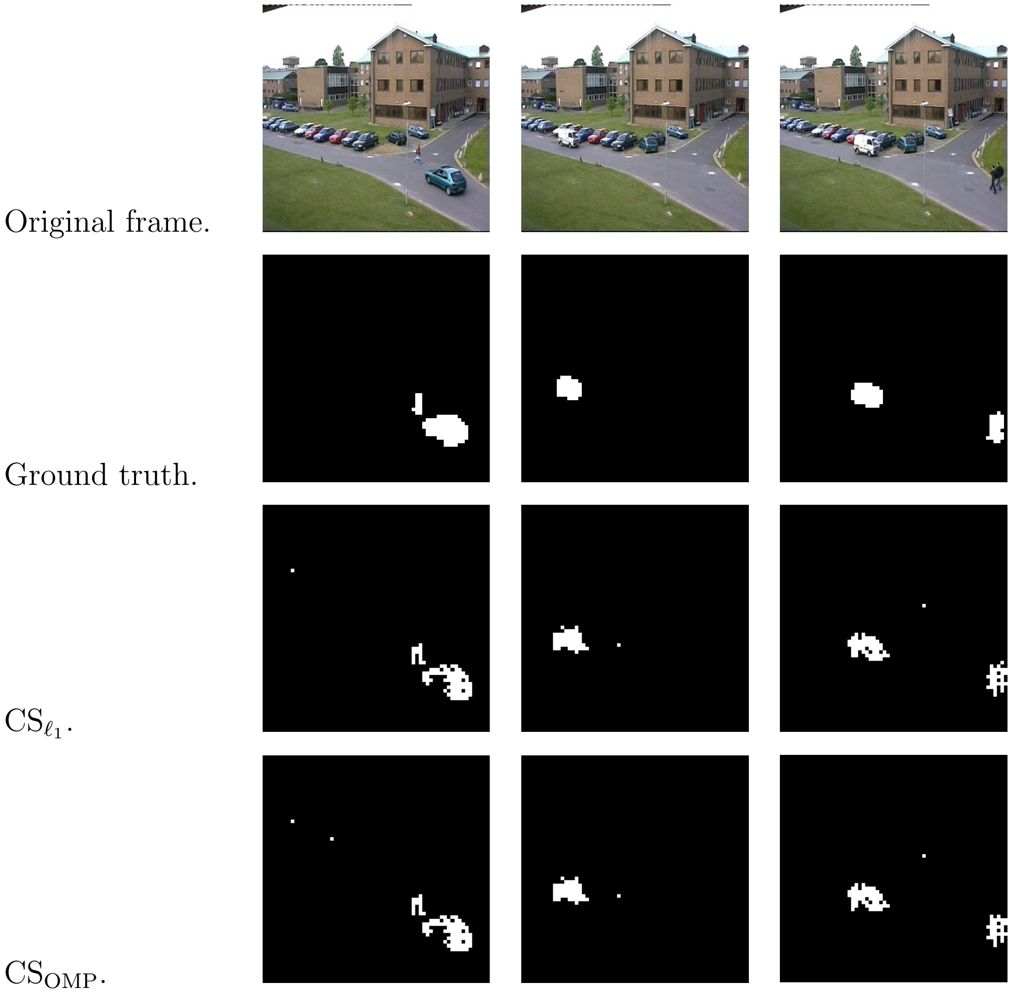
Precision is defined to be the fraction of correctly identified foreground pixels over the number of detected foreground pixels in total, or when written mathematically

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{4.4.2}$$

When the recovery algorithms provide an estimate of a foreground mask, the output is not binary but a probabilistic vector. In order to calculate the precision and recall values, a threshold must first be applied to the estimate. Precision-Recall curves can be used to display information relating to the algorithm's performance across a number of thresholds.

In order to have a singular metric to rank the algorithms across a number of thresholds the Area Under Curve (AUC) (Hanley and McNeil, 1983) is used. This is equal to the probability that an algorithm will rank a randomly chosen positive instance higher than a randomly chosen negative one.

Table 4.4.1: CS_{ℓ_1} and $\mathrm{CS}_{\mathrm{OMP}}$ segmentation for 3 test scenes ($N = 4096, M = 2048, \alpha = 0.05$).



Foreground masks for three test frames can be found in Table 4.4.1. These images have been thresholded so any value above 0.004 is classed as foreground and everything else is set to background. The performance of both algorithms is very similar, although the Precision-Recall curves in Figure 4.4.1 imply that CS_{ℓ_1} may be slightly outperforming $\mathrm{CS}_{\mathrm{OMP}}$ across the different thresholds.

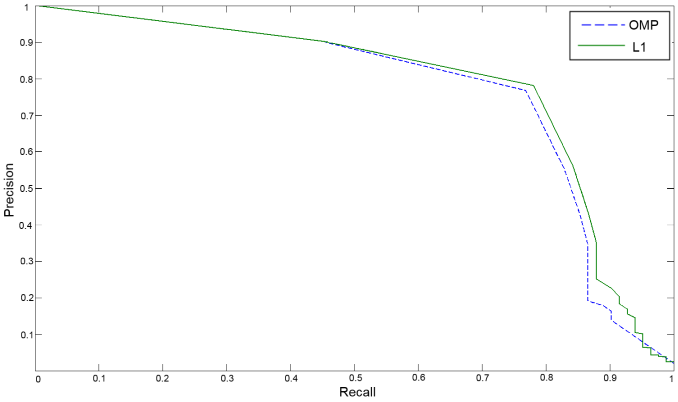


Figure 4.4.1: Precision-Recall Curves for the 3rd test frame.

The effect of the stopping criterion for the orthogonal matching pursuit can be seen in Figure 4.4.2.

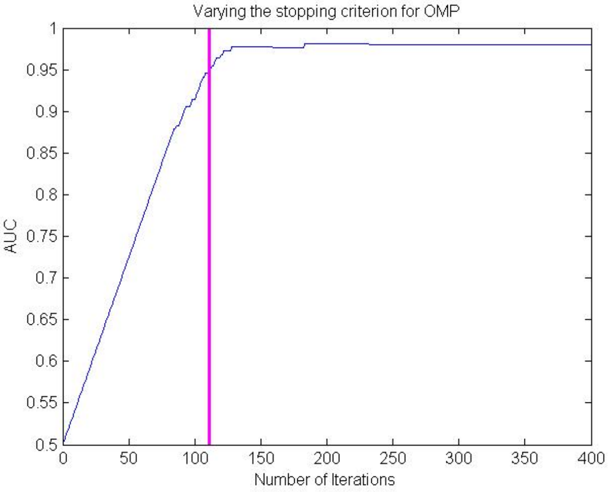


Figure 4.4.2: Selection of the stopping criterion for OMP.

The stopping criterion is increased between 1 and 400 and the AUC is calculated at each stage. As discussed in Section 4.3.3, the selection of the stopping criterion can have an impact on the results; too small and it will underestimate the size of the foreground, too large and computational expense will increase greatly with no noticeable performance improvement. This can be seen in the steep increase of performance in Figure 4.4.2. Note that as the number of iterations reaches approximately the "true" sparsity 111 of the foreground, the performance increases. Ideally, the stopping criterion should be as small as possible to minimise the computational cost, but not at the expense of poor segmentation.

Another consideration is the effect of the choice of M on recovery accuracy. Ideally the number of M should be as small as possible in order to keep computations cost low, whilst still retaining a good enough quality of detection to suit the application, whether it be tracking, classification etc. Figure 4.4.3 shows the performance of both algorithms as the compression ratio $\frac{M}{N}$ increases, with respect to AUC. There is a sharp rise in performance as the compression ratio increases to 20%, and as we approach a compression rating of 30%, both algorithms are performing very well. Note that CS_{ℓ_1} is constantly outperforming $\mathrm{CS}_{\mathrm{OMP}}$ in this experiment over all compression ratios.

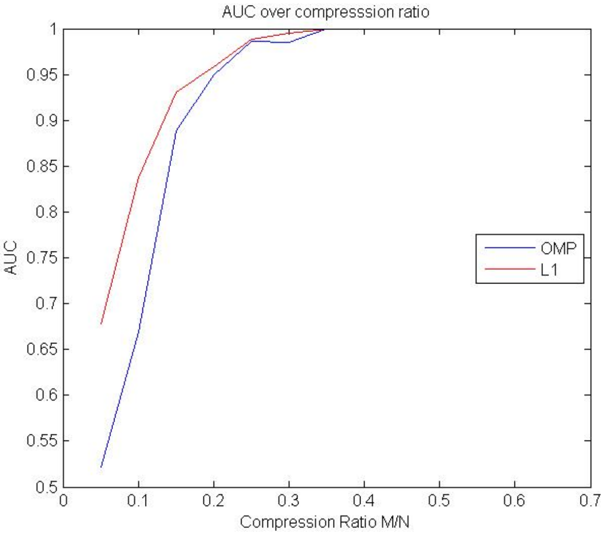


Figure 4.4.3: AUC over Sparsity Ratio.

4.5 Conclusions and Further Work

This paper presents results about the effects of different compressive sensing recovery algorithms on background subtraction for surveillance footage. In our experimentation basis pursuit outperforms orthogonal matching pursuit across the foreground thresholds, although it is hard to distinguish them from foreground masks alone. The effect of the stopping criterion for OMP was seen to have a large impact on performance, which indicates the necessity for adaptive iterations in order to cope with varying sparsity in a video. It is also indicated that ideal boundaries for the compression ratio $\frac{M}{N}$ are between $25 - 35\%$. One important question to address in future is, is it possible to incorporate more prior information about surveillance footage and use this to aid the recovery process? In current methods, there is a failure to use the natural structural properties of foreground, such as the expectation of foreground to take the shape of a person or vehicle. Structured sparsity has not been well explored in the compressive sensing background subtraction literature although a start in this area has been made by La and Do (2006) and Duarte et al. (2008b). Future work seeks to apply a clustering method to these background subtraction algorithms and to build in a natural way to allow the sparsity constraint in the algorithms to vary between frames.

Chapter 5

Conclusion

Data streams provide a challenging environment for statistical analysis. Data points can arrive at a high velocity and may need to be deleted once they have been observed. Due to these restrictions, standard techniques are not applicable to the data streaming scenario. This leads to the need of data summaries to represent the data stream. This thesis has explored how data summaries can be used to perform clustering or classification on data streams across a broad range of applications.

In Chapter 2 we introduced an algorithm for performing spectral clustering on data streams: spectral CluStream. Prior to this work, there did not exist a method for applying spectral clustering to data streams. We considered two variants of this algorithm, a weighted and non-weighted version. Despite having a mathematically valid affinity matrix, the weighted spectral CluStream was found to have poor performance and fundamental difficulties clustering even simple simulated data. This appears to be caused by the weightings dominating the scaling parameter which is essential to spectral clustering. However, the unweighted spectral CluStream was shown to have good performance on par with a windowed approach to spectral clustering even on tricky image data sets. One issue was identified with the unweighted spectral CluStream deletion policy. When dealing with non-stationary data streams, historic micro-clusters could cause poor performance. A correction was proposed in order to retain the good performance but at the cost of using additional micro-clusters to track the stream.

Chapter 3 was motivated by an application which arises in the oil industry where engineers wish to identify corruption within an acoustic signal. This is a difficult problem, particularly as the notion of corruption is not well defined by engineers at present. A common method used to detect changes in structure would be to use changepoint detection methods. However, due to the multiple time series of the DAS data, changepoint methods become computationally in-feasible for large data sets. We re-framed the multiple time series problem as a clustering problem by treating each time series as a different data dimension. We provided a flexible solution combining the streaming capabilities of CluStream with k-means and a similarity metric to detect boundary locations. These boundary locations can be used to isolate sections of the signal which are corrupted. We tested this method on the DAS data set for a range of values of k, the number of clusters and γ, the search parameter. The boundary locations identified by this method were found to be fairly insensitive to the parameter values except at extremities.

Finally in Chapter 4 we considered the different problem of identifying areas of foreground activity within a compressed video stream. Two different recovery algorithms had recently been suggested in the literature. We compared the performance of the two recovery algorithms and showed that Basis Pursuit slightly outperformed Orthogonal Matching Pursuit. However, the pixels identified as foreground were similar for both algorithms. The effect of the stopping criterion for OMP was seen to have a large impact on performance, with the best results being observed when the value of stopping criterion was close to the true sparsity of the current video frame. This reinforces the necessity for adaptive stopping criterion in order to cope with varying sparsity in a video. We also investigated the performance for different compression ratios. The best performance was observed at a compression ratio of between $25 - 35\%$.

Throughout the thesis we assumed that we had access to the true underlying number of clusters k. This is a common assumption made in clustering, however when dealing with real data this is not a realistic assumption. This is particularly an issue in the data streaming setting as the true number of clusters may change as the data stream progresses. Clusters may become irrelevant and stop being updated and new clusters may join the data stream.

One difficulty that we faced throughout the thesis was evaluating algorithmic performance on data streams. If the number of true clusters changes but we are constrained to keeping k fixed this can be an issue. There does not exist an obvious method for evaluating performance of clustering algorithms in the streaming setting. The current standard performance measures take into account only how well the current clustering represents the current state of the stream. It would be useful to develop methods appropriate for evaluating non-stationary data streams. For example, we might wish to judge how well a clustering algorithm learns about the historical state of the stream as well its ability to adapt quickly to new information.

Appendix A

Supplementary Material on Compressive Sensing

The purpose of this appendix is to provide supplementary material on compressive sensing. An introduction to compressive sensing is given in Section A.1. The conditions for choosing a stable measurement matrix $\mathbf{\Phi}$ are described in Section A.2 and more intuition is provided on the orthogonal matching pursuit algorithm in Section A.3.

A.1 Introduction to Compressive Sensing

Compressive sensing (Candès and Tao, 2006; Candès et al., 2006; Donoho, 2006) is based on the discovery that there is a way to accurately sample a signal at a much lower frequency than the Nyquist frequency, provided that the signal that you are trying to sample is either sparse or compressible. The definition of a sparse signal is given below.

Definition 2. *A signal $\boldsymbol{x} \in \mathbb{R}^N$ is known as being K-sparse if it can be represented as a* *linear combination of K basis vectors.*

In simpler terms, we can think of a signal being K-sparse if it can transformed so that it contains at most K non-zero elements, or written mathematically,

$$\|\boldsymbol{x}\|_0 \leq K.$$

For notational purposes, we express the set of all K-sparse signals as

$$\Sigma_K = \{\boldsymbol{x} : \|\boldsymbol{x}\|_0 \leq K\}.$$

Generally the signal of interest is not exactly sparse but approximately sparse, which is also known as compressible. For a signal to be compressible we still require K large coefficients (with $K << N$) but the remaining $N - K$ coefficients are only required to be small and not necessarily zero.

The compressive sensing theory describes how a finite compressible signal $\boldsymbol{x} \in \mathbb{R}^N$ can be recovered from a set of M linear, non-adaptive measurements $\boldsymbol{y}$ where $M << N$. The measurement $\boldsymbol{y}$ is a linear function of the signal $\boldsymbol{x}$ as shown in equation (A.1.1) or visualised in Figure A.1.1. The number of measurements M in $\boldsymbol{y}$ is assumed to be smaller than signal $\boldsymbol{x}$, so we choose a measurement matrix $\boldsymbol{\Phi} \in R^{M \times N}$ with $M << N$. It is known from linear algebra that there are infinitely many vectors $\boldsymbol{x}$ that can solve equation (A.1.1), but usually $\boldsymbol{x}$ is the only sparse solution, provided that $M \geq 2K$, where K is the true sparsity of the signal. Therefore if $\boldsymbol{x}$ is known in advance to be sparse, it can in theory be reconstructed exactly from M measurements.

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{x}. \tag{A.1.1}$$

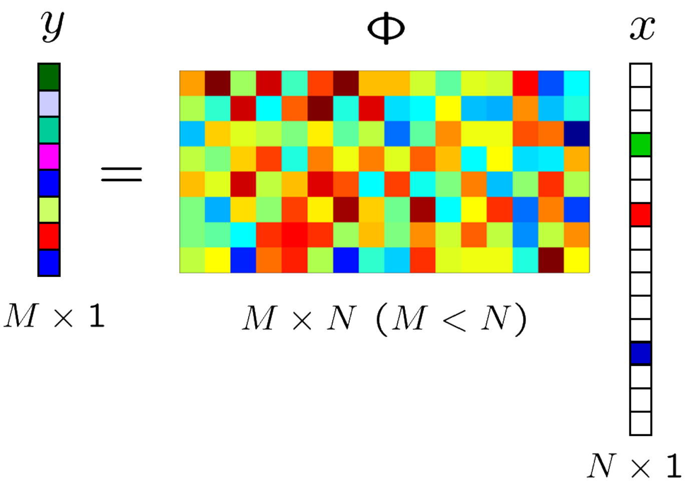


Figure A.1.1: A visualisation of the signal acquirement process in compressive sensing. A sparse signal $\boldsymbol{x}$ of length N is under sampled by multiplication of a random measurement matrix $\boldsymbol{\Phi}$, resulting in the non-sparse measurement vector $\boldsymbol{y}$ of much smaller length M. Courtesy of Volkan Cevher.

Compressive sensing is different from classical sampling in a number of ways. The two main differences are the manner in which measurements are acquired, and how the signal is reconstructed from these measurements.

In classical sampling, measurements are taken at equidistant intervals at a frequency above the Nyquist frequency. The Shannon-Nyquist theorem (Shannon, 1949) allows analogue signals to be digitally sampled and later restored back to an analogue signal accurately. In order for the signal to be reconstructed uniquely from its digital sample, it must be sampled at a rate above the Nyquist Sampling frequency. This frequency is defined to be twice the bandwidth of the signal, where the bandwidth refers to the highest frequency present in the signal. In contrast, in compressive sensing sampling occurs at randomly spaced intervals in accordance with a measurement matrix $\boldsymbol{\Phi}$, the choice of which is discussed in Section A.2.

The method of signal reconstruction also differs between classical and compressive sensing. If the signal has been sampled in accordance with the Shannon-Nyquist theorem, the Whittaker-Shannon interpolation formula is applied in order to reconstruct the signal accurately. However, when dealing with compressed signals things are not so simple. We need to be able to determine the significant coefficients of some sparse representation of the signal and calculate a least squares approximation. This is not straightforward to calculate exactly in practise and instead a recovery algorithm is used to get an optimal solution. In Section A.3 we introduce one such recovery algorithm and provide intuition on how it is able to recover sparse signals.

A.2 Conditions for a Stable Measurement Matrix

The matrix $\mathbf{\Phi}$ represents a dimensionality reduction as it maps the signal from $\mathbb{R}^N$ to $\mathbb{R}^M$. It is of vital importance that we are able to design a stable measurement matrix $\mathbf{\Phi}$ so that the signal information is not damaged by the dimensional reduction from $\boldsymbol{x} \in \mathbb{R}^N$ to $\boldsymbol{y} \in \mathbb{R}^M$. In order to guarantee this, a number of conditions must be true for the measurement matrix $\mathbf{\Phi}$.

A.2.1 Null Space Conditions

The first important property that is required is that the signal $\boldsymbol{x}$ can be uniquely reconstructed. It is possible to show that this will hold true if the null space $\mathcal{N}(\mathbf{\Phi})$ does not contain any vectors in Σ_{2K}. The Null space of $\mathbf{\Phi}$ is defined as follows,

$$\mathcal{N}(\mathbf{\Phi}) = (\boldsymbol{z} : \mathbf{\Phi}\boldsymbol{z} = 0). \tag{A.2.1}$$

In order to preserve $\boldsymbol{x} \in \Sigma_K$, it is required $\mathbf{\Phi}\boldsymbol{x} \neq \mathbf{\Phi}\boldsymbol{x'}$ $\forall \boldsymbol{x'} \in \Sigma_K$, because if $\mathbf{\Phi}\boldsymbol{x} = \mathbf{\Phi}\boldsymbol{x'}$ it would be impossible to distinguish $\boldsymbol{x}$ from $\boldsymbol{x'}$ based only on $\boldsymbol{y}$. Consider the case where $\mathbf{\Phi}\boldsymbol{x} = \mathbf{\Phi}\boldsymbol{x'}$.

$$\begin{aligned}
&\mathbf{\Phi}\boldsymbol{x} = \mathbf{\Phi}\boldsymbol{x'} \\
&\quad \Rightarrow \mathbf{\Phi}(\boldsymbol{x} - \boldsymbol{x'}) = 0 \\
&\quad \Rightarrow (\boldsymbol{x} - \boldsymbol{x'}) \in \Sigma_{2K}.
\end{aligned}$$

This leads us to the conclusion that $\mathbf{\Phi}$ uniquely represents all $\boldsymbol{x} \in \Sigma_K \iff v \notin \mathcal{N}(\mathbf{\Phi})$ $\forall \boldsymbol{v} \in \Sigma_{2K}$. It can be shown that a bound on the spark of $\mathbf{\Phi}$ can characterise this property well. The spark of a matrix is defined in equation (A.2.2) as the smallest number of columns of the matrix that are linearly dependent.

$$\text{Spark}(\mathbf{\Phi}) = \min_{a \neq 0} \|a\|_0 : \mathbf{\Phi}a = 0. \tag{A.2.2}$$

The spark can then be used to give the following guarantee as presented in Eldar and Kutyniok (2012).

Theorem A.2.1. $\forall \boldsymbol{y} \in \mathbb{R}^M \; \exists$ *at most one* $\boldsymbol{x} \in \Sigma_K : \boldsymbol{y} = \mathbf{\Phi}\boldsymbol{x} \iff$ *spark* $(\mathbf{\Phi}) > 2K$.

Proof. Assume $\forall \boldsymbol{y} \in \mathbb{R}^M, \exists$ at most one signal $\boldsymbol{x} \in \Sigma_K : \boldsymbol{y} = \mathbf{\Phi}\boldsymbol{x}$. Assume spark $(\mathbf{\Phi}) \leq 2K$. Due to the bound on the spark of $\mathbf{\Phi}$, there exists a set of at most $2K$ columns that are linearly dependent which implies that there exists some $\boldsymbol{v} \in \mathcal{N}(\mathbf{\Phi}) : \boldsymbol{v} \in \Sigma_{2K}$. It is possible to express $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{x'}$ where $\boldsymbol{x}, \boldsymbol{x'} \in \Sigma_K$. Since $\boldsymbol{v} \in \mathcal{N}(\mathbf{\Phi}), \mathbf{\Phi}(\boldsymbol{x} - \boldsymbol{x'}) = 0$, this implies that

$\boldsymbol{\Phi x} = \boldsymbol{\Phi x'}$. This is a contradiction of the assumption that there is at most one signal $\boldsymbol{x} \in \Sigma_K$ such that $\boldsymbol{y} = \boldsymbol{\Phi x}$. Therefore consider spark $(\boldsymbol{\Phi}) > 2K$.

Assume that there exists $\boldsymbol{x}, \boldsymbol{x'} \in \Sigma_K$ such that $\boldsymbol{\Phi x} = \boldsymbol{\Phi x'} = \boldsymbol{y}$. Therefore it can be seen that $\boldsymbol{\Phi}(\boldsymbol{x} - \boldsymbol{x'}) = 0$. By letting $\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{x'}$ it is possible to write $\boldsymbol{\Phi v} = 0$. But as the spark $(\boldsymbol{\Phi}) > 2K$ this implies that $\boldsymbol{v} = 0 \Rightarrow \boldsymbol{x} = \boldsymbol{x'}$ as expected. □

Since Spark$(\boldsymbol{\Phi}) \in [2, M+1]$, in order to preserve $\boldsymbol{x}$ in the dimensionality reduction, we require $M > 2K$.

A.2.2 The Restricted Isometry Property

Although null space conditions can accommodate for the uniqueness of reconstruction, they do not account for noise. It is possible that the measurement vector $\boldsymbol{y}$ may be corrupted by noise during the measurement process. In order for the reconstruction problem to be well-conditioned, it is necessary that $\boldsymbol{\Phi}$ holds the Restricted Isometry Property (RIP) Donoho (2006) of order $2K$.

Definition 3. *A matrix $\boldsymbol{\Phi}$ satisfies the (RIP) of order K if there exists a $\delta_K \in (0,1)$ such that*

$$(1 - \delta_k)||\boldsymbol{x}||_2^2 \leq ||\boldsymbol{\Phi x}||_2^2 \leq (1 + \delta_k)||\boldsymbol{x}||_2^2,$$

for all $\boldsymbol{x} \in \sum_K = \boldsymbol{x} : ||\boldsymbol{x}||_0 \leq K$.

If $\boldsymbol{\Phi}$ satisfies the RIP with order $2K$, then $\boldsymbol{\Phi}$ preserves the distance between any pair of K-sparse vectors. It is the preservation of distance which will give a stronger guarantee of robustness against noise.

Unfortunately the task of checking that a matrix satisfies the RIP and null space conditions is an NP-hard problem, but fortunately both of these conditions will hold true with high probability if $\boldsymbol{\Phi}$ is selected as a random matrix (Baraniuk, 2007). In order to construct $\boldsymbol{\Phi}$ as such, the elements $\phi_{i,j}$ are chosen as independent realisations of some probability distribution. In Chapter 4, $\phi_{i,j}$ are independent and identically distributed (i.i.d.) Gaussian random variables with mean 0 and variance $\frac{1}{N}$, but other distributions can be used.

A.3 Intuition for Orthogonal Matching Pursuit

Orthogonal matching pursuit (OMP) (Tropp, 2004) is a greedy recovery algorithm which iteratively computes the local optimum solutions in the hope that these will lead to the global optimum solution. Orthogonal matching pursuit determines the column of $\boldsymbol{\Phi}$ which is most correlated with $\boldsymbol{y}$, or which contributes to $\boldsymbol{y}$ most. This is repeated again by comparing the correlation between columns of $\boldsymbol{\Phi}$ with the signal residual, until it reaches some stopping criterion defined by the user. This is given algorithmically in Chapter 4, Algorithm 6.

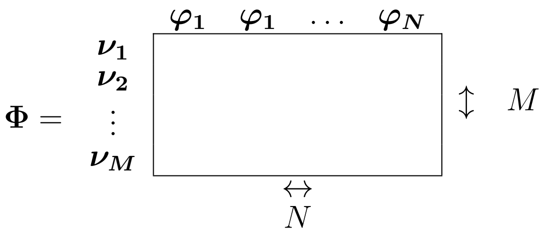
In the algorithm, $\hat{\boldsymbol{x}}$ is updated for each iteration with the contributions from the columns of $\boldsymbol{\Phi}$ placed in the indexes stored in Λ_t. So if the algorithm is stopped after the Kth iteration for some positive integer K, $\hat{\boldsymbol{x}}$ will be a K-sparse vector. This emphasises how important it is to run the algorithm for the correct number of iterations, although generally this number is unknown.

The most important and most computationally intensive step in Algorithm 6 is the first step, solving $\lambda_t = \arg\max_{j=1,\dots,N} | < r_{t-1}, \varphi_j > |$. It may not be obvious at first why this step will promote sparsity and therefore help identify the non-zero components of $\boldsymbol{x}$. A toy example should help explain further.

Example Let $\boldsymbol{x}$ be a 1-sparse vector of length N, therefore there is only one non-zero element in $\boldsymbol{x}$. Let this non-zero element occur at the p^{th} location of $\boldsymbol{x}$, $x(p)$.

We define the measurement matrix $\boldsymbol{\Phi} \in R^{M \times N}$ in the usual manner and define the rows of $\boldsymbol{\Phi}$ as measurement vectors $\boldsymbol{\nu}_1, \boldsymbol{\nu}_2 \ldots, \boldsymbol{\nu}_M$, each with length N. The columns of $\boldsymbol{\Phi}$, defined to be $\boldsymbol{\varphi_1}, \boldsymbol{\varphi_2}, \ldots, \boldsymbol{\varphi_N}$ each of length M, are used to collect observations of our original signal.

So $\boldsymbol{\Phi}$ can be visualised as in,



with the elements of $\boldsymbol{\Phi}$ taken from the Gaussian distribution as described in Section A.2. Measurements $\boldsymbol{y}$ are taken by applying equation (**??**), and the result can be evaluated in terms of columns and rows of $\boldsymbol{\Phi}$.

$$
\begin{aligned}
\boldsymbol{y} &= \boldsymbol{\Phi x} \\
&= \begin{bmatrix} \nu_1(p)\ x(p) \\ \nu_2(p)\ x(p) \\ \vdots \\ \nu_M(p)\ x(p) \end{bmatrix} \\
&= \boldsymbol{\varphi_p}\ x(p).
\end{aligned}
$$

It is now clear that, because $\boldsymbol{x}$ had only one element $x(p)$, than the measurement matrix $\boldsymbol{y}$ only involves contributions from one column $\boldsymbol{\varphi_p}$ of $\boldsymbol{\Phi}$. Although this is a toy example and many signals will not be 1-sparse, the principle idea continues. Many columns of $\boldsymbol{\Phi}$ will not contribute to the measurement vector $\boldsymbol{y}$ as the corresponding elements of $\boldsymbol{x}$ are close to or equal to 0.

Bibliography

C. C. Aggarwal. *Data streams : Models and algorithms*. Springer, 2007.

C. C. Aggarwal, T. J. Watson, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. *Proceedings of the 29th International Conference on Very Large Data Bases*, 81–92, 2003.

F. Alimoglu and E. Alpaydin. 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*, 1996.

R. G. Baraniuk. Compressive sensing. *IEEE Signal Processing Magazine*, 24(4):118–121, 2007.

K. Boone, A. Ridge, R. Crickmore, and D. Onen. Detecting leaks in abandoned gas wells with fibre-optic distributed acoustic sensing. In *International Petroleum Technology Conference*, 2014.

T. Bouwmans. Recent advanced statistical background modeling for foreground detection - a systematic survey. *Recent Patents on Computer Science*, 4(3):147–176, 2011.

S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, 91–99, 1998.

P. Buhlmann, P. Drineas, M. Kane, and M. van der Laan. *Handbook of Big Data*. Springer, 2016.

E. Candès and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 1–22, 2005.

E. Candès and T. Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE Transactions on Information Theory*, 52:5406–5425, 2006.

E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

F. Cao, M. Estert, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, 328–339. SIAM, 2006.

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

V. Cevher and M. F. Duarte. Sparse signal recovery using markov random fields. *Advances in Neural Information Processing Systems*, 1–8, 2008.

V. Cevher and A. Sankaranarayanan. Compressive sensing for background subtraction. *European Conference on Computer Vision*, 155–168, 2008.

B. Chen, B. Gao, T.-Y. Liu, Y.-F. Chen, and W.-Y. Ma. Fast spectral clustering of data using sequential matrix compression. *European Conference on Machine Learning*, 590–597, 2006.

J. Chen and A. K. Gupta. *Parametric Statistical Change Point Analysis.* Springer Science & Business Media, 2012.

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Rev.*, 43(1):129–159, 2001.

X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 313–318, 2011.

Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 133–142. ACM, 2007.

S.-C. S. Cheung and C. Kamath. Robust techniques for background subtraction in urban traffic video. In *Electronic Imaging*, 881–892, 2004.

F. R. K. Chung. *Spectral graph theory.* American Mathematical Society, 1997.

J. F. Claerbout and F. Muir. Robust modelling with erratic data. *Geophysics*, 38(5):826–844, 1973.

M. Cossalter, M. Tagliasacchi, and G. Valenzise. Privacy-enabled object tracking in video sequences using compressive sensing. In *Advanced Video and Signal-Based Video Surveillance*, 436–441, 2009.

R. Cramer, M. Hooimeijer, and A. Franzen. New software moves distributed-temperature sensing data. *Oil & Gas Journal*, 106(38):52–52, 2008.

R. Davies, L. Mihaylova, N. Pavlidis, and I. A. Eckley. The effect of recovery algorithms on compressive sensing background subtraction. In *Sensor Data Fusion: Trends, Solutions, Applications*, 1–6, 2013.

G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997.

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

D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52:1289–1306, 2006.

M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25: 83–91, 2008a.

M. F. Duarte, M. B. Wakin, and R. G. Baraniuk. Wavelet-domain compressive signal reconstruction using a hidden markov tree model. *Acoustics, Speech and Signal Processing*, 5137–5140, 2008b.

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

I. A. Eckley, P. Fearnhead, and R. Killick. Analysis of changepoint models. In D. Barber,

A. T. Cemgil, and S. Chiappa, editors, *Bayesian Time Series Models*, chapter 10, 205–224. Cambridge University Press, Cambridge, 2011.

Y. C. Eldar and G. Kutyniok. *Compressed sensing: theory and applications.* Cambridge University Press, 2012.

A. Elgammal, D. Harwood, and L. Davis. Non-parametric model for background subtraction. *European Conference on Computer Vision*, 751–767, 2000.

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

B. Everit, S. Landau, and M. Leese. *Cluster analysis.* Wiley, 2001.

C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 214–225, 2004.

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

J. Gama. *Knowledge discovery from data streams.* CRC Press, 2010.

J. Gama and M. M. Gaber. *Learning from data streams : Processing techniques in sensor networks.* Springer, 2007.

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

J. Hanley and B. McNeil. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148(3):839–843, 1983.

L. Huang, D. Yan, N. Taft, and M. I. Jordan. Spectral clustering with perturbed data. *Advances in Neural Information Processing Systems*, 705–712, 2008.

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

T. Kong, Y. Tian, and H. Shen. A fast incremental spectral clustering for large data sets. In *International Conference on Parallel and Distributed Computing, Applications and Technologies*, 1–5, 2011.

P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and information systems*, 29(2):249–272, 2011.

G. Kylberg. 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, 2011.

C. La and M. Do. Tree-based orthogonal matching pursuit algorithm for signal reconstruction. In *International Conference on Image Processing*, 1277–1280, 2006.

D. Laney. 3-d data management: Controlling data volume, velocity and variety. Technical report, META Group Research Note, 2001.

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

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

T.-Y. Liu, H.-Y. Yang, X. Zheng, T. Qin, and W.-Y. Ma. Fast large-scale spectral clustering by sequential shrinkage optimization. In *European Conference on Information Retrieval*, 319–330, 2007.

S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 281–297, 1967.

C. Maung and H. Schweitzer. Pass-Efficient Unsupervised Feature Selection. *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 1–9, 2013.

N. J. B. McFarlane and C. P. Schofield. Segmentation and tracking of piglets in images. *Machine Vision and Applications*, 8(3):187–193, 1995.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 849–856, 2001.

H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering with application to monitoring of evolving blog communities. *SIAM International Conference on Data Mining*, 261–272, 2007.

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

E. Page. Continuous inspection schemes. *Biometrika*, 41(1):100–115, 1954.

Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. *Signals, Systems and Computers*, 40–44, 1993.

O. P. Patri, V. S. Sorathia, and V. K. Prasanna. Event-driven information integration for the digital oilfield. In *SPE Annual Technical Conference and Exhibition*, 3530–3543, 2012.

M. Piccardi. Background subtraction techniques: A review. In *Systems, Man and Cybernetics*, 3099–3104, 2004.

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

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

V. Shankar, M. Kleijnen, S. Ramanathan, R. Rizley, S. Holland, and S. Morrissey. Mobile shopper marketing: Key issues, current insights, and future research avenues. *Journal of Interactive Marketing*, 34:37–48, 2016.

C. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

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

H. Shinnou and M. Sasaki. Spectral clustering for a large data set by reducing the similarity matrix size. *In Proceedings of the Sixth International Language Resources and Evaluation)*, 201–204, 2008.

T. Silkina. *Application of Distributed Acoustic Sensing to Flow Regime Classification.* Master's thesis, Norwegian University of Science and Technology, 2014.

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

Q. Song, J. Ni, and G. Wang. A fast clustering-based feature subset selection algorithm for high-dimensional data. *IEEE Transactions on Knowledge and Data Engineering*, 25(1): 1–14, 2013.

D. A. Spielman and S.-H. Teng. Spectral partioning works: planar graphs and finite element meshes. In *Foundations of Computer Science*, 96–105, 1996.

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

H. L. Taylor, S. C. Banks, and J. F. McCoy. Deconvolution with the norm. *Geophysics*, 44 (1):39–52, 1979.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, 267–288, 1996.

K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. *Computer Vision*, 255–261, 1999.

J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

U. University of Reading. Pets 2001 data sets. `http://www.cvg.rdg.ac.uk/slides/pets.html`, 2001.

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

J. van der Horst, H. Den Boer, P. In 't Panhuis, B. Wyker, R. Kusters, D. Mustafina, L. Groen, N. Bulushi, R. Mjeni, K. F. Awan, S. M. Rajhi, M. M. Molenaar, A. Reynolds, R. Paleja, D. Randell, R. Bartlett, and K. Green. Fibre optic sensing for improved wellbore production surveillance. In *International Petroleum Technology Conference*, 2014.

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

D. Wagner and F. Wagner. Between min cut and graph bisection. In *Proceedings of the 18th*

International Symposium on Mathematical Foundations of Computer Science, 744–750, 1993.

G. Warnell and R. Chellappa. Compressive sensing in visual tracking. In H. El-Alfy, editor, *Recent Developments in Video Surveillance*, chapter 1, 205–224. InTech, 2011.

C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 682–688, 2001.

Z. Wu and R. Leahy. 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, 1993.

D. Yan, L. Huang, and M. I. Jordan. Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 907–916, 2009.

H. Yang, M. R. Lyu, and I. King. Efficient online learning for multitask feature selection. *ACM Transactions on Knowledge Discovery from Data*, 7(2):1–27, 2013.

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

T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, 1996.