Chapter 1

Spectral Clustering for Data Streams

1.1 Introduction

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

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

1.2 Literature Review

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

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

Graph partitioning problems have long been of interest to mathematicians, ever since Euler solved the popular "Seven Bridges of Königsberg" problem in 1736. However it was not until the 1970's that the link between graph partitioning and matrix spectra was observed when Donath and Hoffman (1973) established a lower bound for the number of edges which must be cut in a graph partitioning problem, based on a factor of the eigenvectors of the affinity matrix. Around the same time Fieldler (1973) published work highlighting the importance of the value of the second smallest eigenvector of the affinity matrix when producing a two way graph partition, also known as a bi-partition. As bi-partitioning was studied the question arose, how to step from bi-partitioning problems to n-partitioning problems? The obvious two approaches are to either recursively apply bi-partitioning on the graph or calculate an n-partition directly from the n smallest eigenvectors. This is discussed further in Alpert (1995) where they claim that recursive bi-partitioning, which was more popular at the time, is inferior compared with creating one partition using many eigenvectors. For our purposes we will consider only the direct n-partitioning problem. For a full review of the historical origins of spectral clustering, see the tutorial von Luxburg et al. (2008).

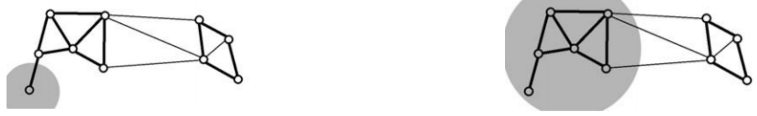
The first paper to bring Spectral Clustering to the Machine Learning community was Shi and Malik (2000) which treated image segmentation as a graph-cut problem. Mathematically, a cut of a graph G is the set of edges S, such that the removal of S from G disconnects G. A graph is connected if there is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

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

The Minimum cut (Wu and Leahy, 1993) is simply the cut which generates the minimum cut weight. However the minimum cut does not always provide the best solution; it tends to favour removing small sections and isolated nodes of the graph. The main proposal from Shi and Malik was to consider the cut function as a fraction of the total edge connections to all nodes in the graph, instead of using the total edge weight connecting the two partitions. This subtle yet important distinction provides us with the highly acclaimed normalised cut (NCut), defined in equation (1.2.2), which is an integral part of spectral clustering algorithms. The main idea is that big clusters will increase assoc(A,V), thus decrease Ncut(A,B). This will encourage splitting the data into fairly evenly sized clusters, and avoid the minimum cut issue of segmented isolated points. Minimising the normalised cut is an NP-complete problem but an approximate solution can be found efficiently by finding the eigenvector with the second smallest eigenvalue. Figure 1.2.1 shows both the minimum cut and normalised cut solutions for a particular graph. The shaded/non-shaded represented the partitioning. The minimum cut isolates one node from the rest of the graph, whilst the normalised cut provides a more balanced and sensible partition.

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

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



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

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

On Spectral Clustering Ng et al. (2001) was one of the first papers to provide theoretical guarantees on performance for spectral clustering algorithms. Unlike previous authors who had only shown empirical results to justify spectral clustering abilities, they prove that their spectral clustering algorithm will produce a reasonable clustering, given certain assumptions that the clusters are well-spaced. This version of the spectral clustering algorithm has been popularly cited throughout the literature and is the one that we shall reference in Section 1.3.

Although spectral clustering has been shown to perform well empirically on simple data sets, computational problems arise as the data set size increases. There have been a number of ways to deal with speed up in the static case, one of the most popular methods is to use Nyström efficiency methods (Williams and Seeger (2001); Fowlkes et al. (2004)). The Nyström method samples the columns of the affinity matrix and approximates the full matrix by using correlations between the sampled columns and the remaining columns. Effectively we can think of this sampling as a dial, which the user has control over. Sampling more columns will provide a better results, but at a cost. Although Nyström methods is an approximation technique to speed up the computation of spectral clustering, the working memory can be high. Another drawback with Nyström is that due to random sampling it is possible to under represent or entirely miss smaller clusters. One alternative to Nyström methods is to perform a permutation on the data to act as a pre-processing step to generate a smaller summary data set. By feeding a smaller set of representative points into the spectral clustering algorithm instead of the whole data set, we can lessen the effect of the computational bottleneck that comes with eigen analysis. Fast Approximate Spectral Clustering Yan et al. (2009) explores the theoretical guarantees on misclustering the data set, given that some permutation has been performed on the data before the spectral clustering. Specifically they exactly quantify the misclustering of data sets, given that the original data is summarised using K-means (KASP) or a Random Project tree (RASP) as preprocessors. RD: Actually their bound might be naff. Should I criticise this? I don't have anything better to offer! This is explored further in Section 1.3.

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

Zelnik-manor et al. (2004) present a method for automatically choosing the true number of clusters using the eigenvectors to inform their choice. More commonly the eigenvalues are used to estimate the number of clusters, but if the clusters are not clearly separated identifying the number of clusters from eigenvalues alone is not trivial. We shall assume that the true number of clusters is known.

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

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

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

An incremental update algorithm is proposed in Ning et al. (2007) and Ning et al. (2010) which can deal with both additional data points joining the network, and similarity weights changing between existing data points. The algorithm updates the eigenvectors and eigenvalues directly without performing a full eigen-decomposition. The addition of a new data point is treated as a series of n weight changes, where is n is the number of currently observed data points. However the authors recommend a full re-clustering in batch to minimise cumulative errors. There are some issues with update method, mainly that the updating of eigenvectors means that the orthogonality property may be lost. Also if the spatial neighbourhoods of often changing vertices are large it can still be computationally difficult as the eigenvector update step involves the inversion of a matrix. Finally the authors recommend a full spectral re-clustering occasionally to prevent the accumulation of errors in the eigenvectors, this is not feasible in the streaming setting. Generally this method is not suitable for data streaming, as the size of the Laplacian can grown unbounded for an infinite data stream. However since it is the most well-known relevant "online" algorithm existing, we will compare performance against it in Section 1.5.

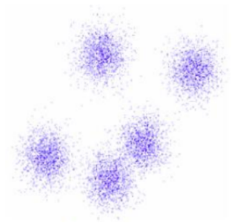
Other incremental spectral clustering algorithms include Kong et al. (2011), Langone et al. (2014) and Dhanjal et al. (2011) which approximates the eigen decomposition of the Laplacian incrementally but still requires regular full re-clustering.

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

1.3 Spectral Clustering

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

The goal of clustering algorithms is to partition data ($X = x_1, x_2, \ldots, x_n$) into k disjoint classes such that each x_i belongs to one and only one class. Clusters come in all shapes and sizes, they can be spherical and compact as in Figure 1.3.1a or connected but not visually compact as in Figure 1.3.1b. Data which is compact may be simple to cluster as the gaps between clusters are easy for simple clustering algorithms like k-means to identify. Connected but non-compact data sets are can be much more challenging than compact data sets, and can cause some simple clustering algorithms to fail. Spectral clustering can provide good quality segmentation on even these difficult cases, however its performance comes at the cost of computational complexity. We shall use the Jordan-Weiss (NJW) spectral clustering framework Ng et al. (2001) which is described in Algorithm 1.


(a) Compact clusters

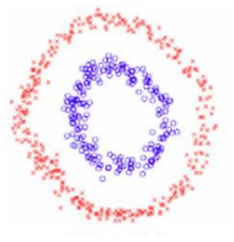

(b) Non spherical clusters

Figure 1.3.1: Examples of clusters

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

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

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

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

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

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

Using this construction will give a sparse affinity matrix instead of a fully connected graph, which will help lower the computational complexity. In Section 1.5, we use a fully connected graph for all experiments.RD: May change to ϵ neighbourhood as Ning is proving slow with fully connected The degree of each vertex v_i is defined as, $d_i = \sum_{j=1}^{N} A_{ij}$, the sum of the rows of the affinity matrix. The degree matrix, D, is then defined as a diagonal matrix with i^{th} diagonal element equal to d_i. We use the Normalised Laplacian defined in equation (1.3.3), which relates to an approximation of minimising the Normalised Cut, discussed in Section 1.2. RD: In Algorithm 1 - bold all of the vectors

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

Algorithm 1 NJW spectral clustering algorithm

```
Input: Data set $S = x_1, \dots, x_n$, number of clusters $k$
Output: $k$-way partition of the input data
1: Construct the affinity matrix $A$ by the following Gaussian kernel function:
       $$a_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), i, j = 1, \dots, n.$$
2: Compute the normalised affinity matrix $L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where $D$ is the diagonal matrix
   with $D_{ii} = \sum_{j=1}^{n} a_{ij}$.
3: Compute the $k$ eigenvectors of $L$, $v_1, v_2, \dots, v_k$, associated with the $k$ largest eigenvalues,
   and form the matrix $X = [v_1, v_2, \dots, v_k]$.
4: Renormalize each row of $X$ to form a new matrix $Y$.
5: Partition the $n$ rows into $k$ clusters via a general cluster algorithm, such as the k-means
   algorithm.
6: Assign the original point $x_i$ to the cluster $k \iff$ the corresponding row $i$ of the matrix
   $Y$ is assigned to the cluster $k$.
```

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

Algorithm 2 KASP

```
Input: $n$ data points $\{xi\}_{i=1}^{n}$, number of representative points $m$, number of overall clusters $k$
Output: $m$-way partition of the input data
1: Perform k-means with $m$ clusters on $x_1, \dots, x_n$
2: Compute the cluster centroids $y1, \dots, y_k$ as the $m$ representative points.
3: Build a correspondence table to associate each $x_i$ with the nearest cluster centroids $y_j$ .
4: Run a spectral clustering algorithm on $y_1, \dots, y_k$ to obtain an $k$-way cluster membership for each of $y_i$.
5: Recover the cluster membership for each $x_i$ by looking up the cluster membership of the corresponding centroid $y_j$ in the correspondence table.
```

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

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

1.4 Micro-cluster based spectral clustering in data streams

In this Section we discuss online streaming methods, introduce the streaming algorithm Clustream, and state the combined online Spectral clustering algorithm.

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

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

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

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

Clustream Aggarwal et al. (2003) offers a framework which allows quick and easy updates and the ability to perform sophisticated clustering algorithms. The main idea is to separate the clustering process in two stages, a micro clustering stage and a macro clustering stage. The micro clustering stage continuously updates statistical summaries of the data stream, and the macro clustering is more computationally intensive and run in batch or on a user request.

The micro-clustering stage is a way of maintaining an active, evolving representative summary of the data, without storing the absolute values of the data points. Micro-clusters are defined as a temporal extension of the cluster feature vector first described in Zhang et al. (1996). The data stream is summarised by many small clusters, which are initially generated by k-means. The online phase stores q microclusters in memory, where q is an input parameter. We take an initial training set, and perform k-means but choose the value of k to be much larger than the expected number of true macro clusters k. The aim here is to create a fine scale summary of the data. The value of q should be chosen to be as large as computationally comfortable. The larger k is, the finer scale that the summaries will be. It is vital to ensure that the micro-cluster well represent the underlying data set or else the macro clustering will under perform. These q clusters are our first micro clusters. Over time, we will update these micro clusters, adding new data points to them, merging them and removing old micro clusters, although the number of micro clusters should stay fixed throughout.

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

Assume that we have a data stream S which consists of d-dimensional data x_i arriving in sequence. $S = \{x_1, x_2, x_3, \ldots x_i, \ldots,\}$ $, x_i \in \mathbb{R}^d$. Each microcluster has five components. Each micro-cluster M_j for $(j \in 1 \ldots, q)$ is stored as a $(2 \cdot d + 3)$ tuple $(\boldsymbol{CF1^x}, \boldsymbol{CF2^x}, n, CF1^t, CF2^t)$. The definitions are given in equation (1.4.1). $CF1^x$ is the sum of all observed data in that micro cluster, $CF2^x$ is the sum of the squares of the data and n is the number of elements assigned to that micro-cluster. $CF1^t$ and $CF2^t$ refer to the sum of the time stamps, and the sum of squared time stamps respectively. Note that both $CF1^x$ and $CF2^x$ are d-dimensional vectors. RD: Bold all the vectors

Each micro-cluster M_j will have

$$\begin{aligned}
CF1^x &= \sum_{x_i \in j} x_i \ . \\
CF2^x &= \sum_{x_i \in j} (x_i)^2 \ , \\
CF1^t &= \sum_{x_i \in j} t_i \ , \\
CF2^t &= \sum_{x_i \in j} (t_i)^2 \ , \\
n &= \sum_{x_i \in j} 1 \ , \qquad (1.4.1)
\end{aligned}$$

When a new data point x_{t-1} is assigned to microcluster j, the following update ((1.4.2))is applied.

$$\begin{aligned}
CF1^x &= CF1^x + x_i \ , \\
CF2^x &= CF2^x + (x_i)^2 \ , \\
CF1^t &= CF1^t + t_i \ , \\
CF2^t &= CF2^t + (t_i)^2 \ , \\
n &= n + 1 \ , \qquad (1.4.2)
\end{aligned}$$

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

$$\text{Centre of microcluster j} = \bar{M_j} = \frac{CF1^x}{n} \tag{1.4.3}$$

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

$$M^* = \underset{j \in 1:m}{\arg\min} \|X_i - \bar{M_j}\|^2 \tag{1.4.4}$$

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

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 $\bar{M}_j, (1 \leq j \leq m)$ are treated as representative points for the data stream S, and a standard clustering algorithm can be used to determine clusters. The nature of this algorithm allows the user to get close to online streaming and perform spectral clustering on a summary of the whole of the data set. The Clustream algorithm is given in Algorithm 3.

Algorithm 3 Clustream RD: tidy this!

```
Input: $q$
Output: -
  Initialise the microclusters kmeans($x_1, \dots x_{init}, q$) and equations (1.4.1)
  for each new data point $x_i$ do
    Find the closest microcluster to $x_i$, $M^*$ using equation (1.4.4)
    if $x_i$ belongs to  then
      absorb $x_i$ into microcluster $M^*$ using equations (1.4.2)
    else
      Use $x_i$ to start it's own new microcluster using equations (1.4.1)
      if any microcluster is suitably old then
        Remove it
      else
        Merge the two closest microclusters
      end if
    end if
  end for
```

Clustream has proved popular, since the paper was first published in 2003 it has been cited over 1400 times. Empirically there is much evidence to show that micro-clustering is an effective way to maintain information about your data set. However there has been little theoretical work to explain why this micro-clustering method seems to work so well. Our intention is to a develop method for applying spectral clustering to data streams. In order to achieve this we adapt a micro-clustering type approach to quickly update a summary of the data. When an overall clustering is required, spectral clustering is performed using the centres of the micro-clusters as the input data. The micro-clusters act as a way of summarising the constantly arriving data steam whilst allowing updates to occur in a non intrusive, non-computationally difficult manner, with limited storage requirements.

RD: Need to introduce using spectral clustering as the macroclustering algorithm in Clustream. Does this need it's own algorithm? Would be very similar to clustream. Maybe not?

Our online spectral clustering algorithm is given in Algorithm 4

```
Algorithm 4 Online Spectral Clustering
Input: $q$,
Output: Outputs
```

RD: Write my algorithm here

In Section 1.5 we analyse the performance of Online Spectral Clustering.

RD: The end of this section needs wrapping up. The next bit will be used if I use the two different methods of feeding microclusters into the spectral clustering algorithm.

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

1. Calculate the centre of each microcluster $\bar{M}_j$ and use it as an object to be clustered by the macro clustering algorithm.
2. Do the same as before, but weighting each microcluster centre$\bar{M}_j$ proportionally to n (the number of points assigned to that microcluster), so that microclusters with more objects will have more influence on the final clustering.

1.5 Experimentation

The set up:

- three algorithms
- a number of data streams - some static, some evolving, some simulated, some real
- performance metrics - Purity and V-measure

The three streaming algorithms must keep track of the data streams, and in batch must deliver a spectral clustering of the data. In our experiments, the batch size is 10, i.e, we request a spectral clustering every 10 time steps.

1.5.1 The algorithms

Windowed Spectral

This is a base line. Window size ???

Incremental Spectral Clustering

Recap Ning algorithm Discuss the need to re-cluster regularly (how can we cope with this in a streaming setting?) Settings for experiments: Empirically we found (like the authors) that 2 iterations of the update step is enough to stabilise the estimates of the eigenvalues and eigenvectors and use that setting.

Spectral Clustream

Choice of the number of microclusters?

1.5.2 The data sets

Simulated Data

Simple simulated data sets, Gaussian. May include PCA plots?

Data set	# Dim	# Clusters
A	2	3
B	3	5
C	2	2
D	3	4
E	3	6
F	5	8

Table 1.5.1: Simulated Data sets

If using, also introduce norm, norm overlap, t, cassini, spirals, norm jump.

Real Data

Texture data

UCI Pendigits

We investigate the performance of Clustream spectral clustering and Clustream K-means on real data. The data set is taken from the UCI Pendigits data set. The data set consists of hand drawn digits of the numbers 0-9. There are 250 samples taken from 44 writers. The data was collected using a pressure sensitive tablet. There are 16 features, each relating to the co-ordinate information taken from the input tablet. We restrict our analysis to pairwise comparison of digits. For example we attempt to cluster the digits 0 and 1, and treat the data as if it is arriving in a constant data stream.

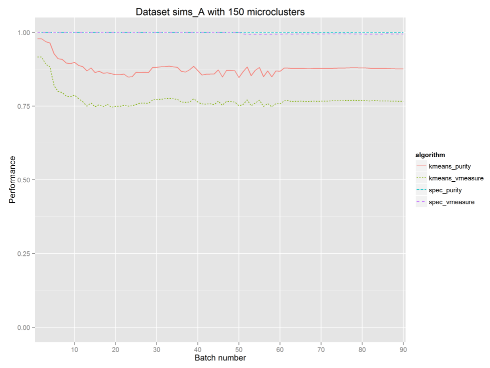
1.5.3 Performance

- Introduce Purity
- Introduce V-Measure

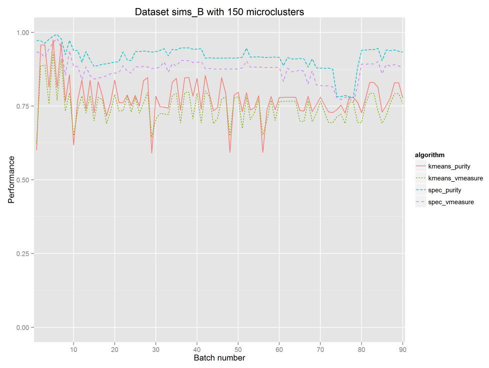
1.5.4 Results

Clustream Macroclustering - K-means vs Spectral

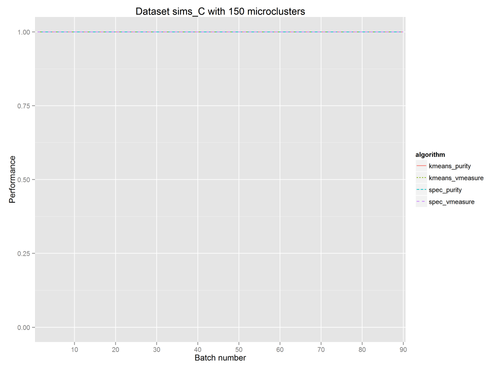
Figure 1.5.1 shows performance of clustream K-means and clustream spectral on the six simulated streams introduced above. We ran clustream for the duration of the stream with a 150 microclusters. The initial microclusters were generated using 500 initial data points with kmeans. Every 50 time steps, a macro clustering stage was performed, both k-mean and spectral clustering. At each batch time point, the performance and V-measure were calculated.



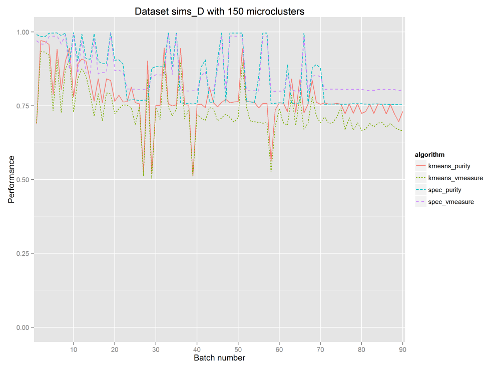
(a) Data set A



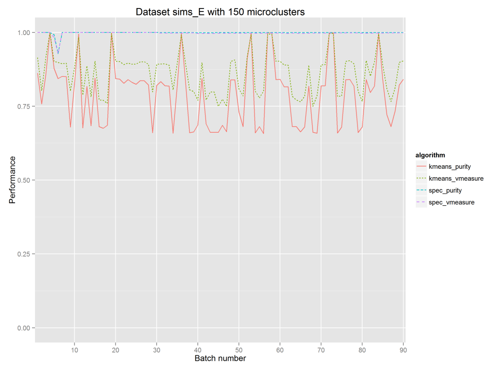
(b) Data set B



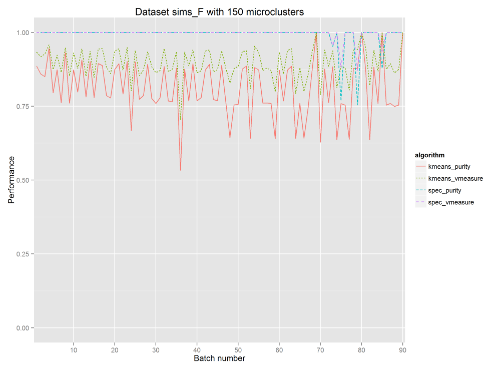
(c) Detest C



(d) Data set D



(e) Data set E



(f) Data set F

Figure 1.5.1: Performance of Clustream k-means and Clustream Spectral

In the simulated data set we generally see spectral out performing k-means. Some data sets seem easier than others to cluster, for example the performance for Data set C is near perfect for algorithms.

We also ran this on the UCI Pendigits data streams, again with 150 microcluster. These streams are considerably shorter, and also more difficult to cluster. Performance varies across the different pairwise settings. Digits 3 and 6 seems easy for both to separate. Generally k-means and spectral have similar performance, both consistently giving better values for purity than V-measure, which is to be expected (?).

Clustream and evolving data sets - jump

We are also interested in evolving data sets. In this simple simulated data set, one cluster instantly jumps from one location to another. Figure **??** shows performance of Clustream k-means and Clustream spectral on this jump detest.

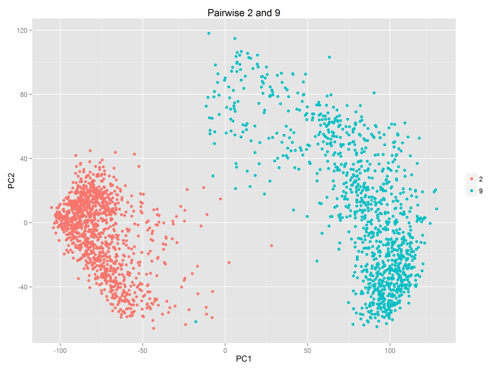
Spectral Clustream vs Ning (without re-clustering)

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

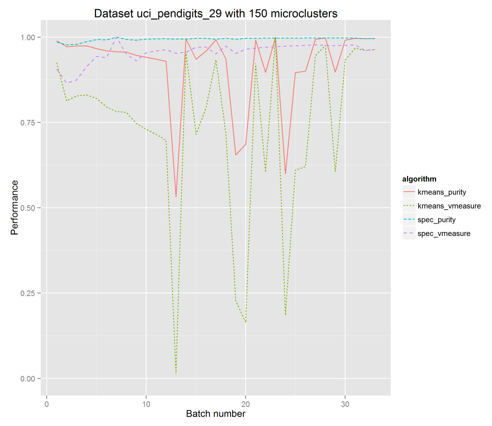
In Figure 1.5.4 I run both Ning and Clustream on the “Norm overlap” data set. We do not perform re-clustering for the Ning algorithm at all, so we can view how bad performance may get over time. This was run over a fairly short stream of just 200 data points. Cluster performance was requested every 10 time steps, resulting in 20 performance points.

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

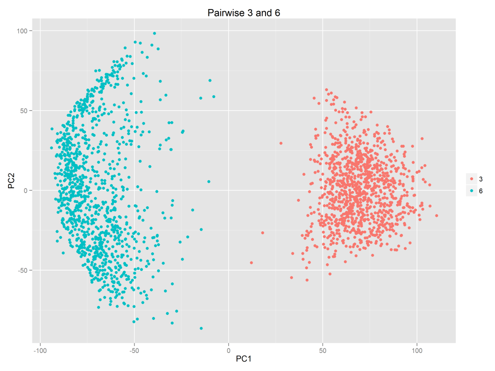
We can also see that Spectral Clustream is generally performing much better in terms of



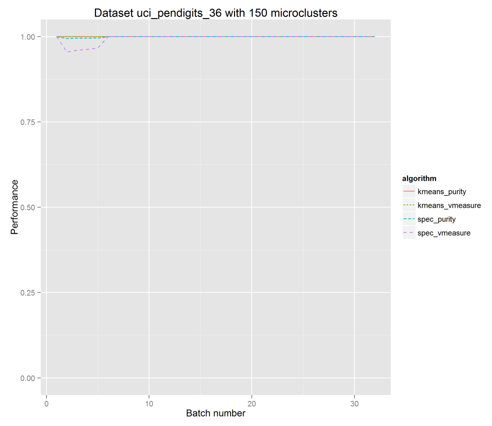
(a) PCA of digits 2 and 9



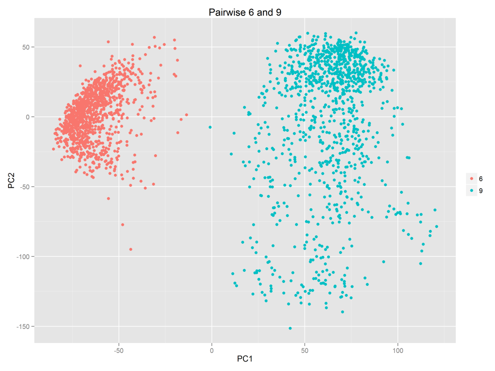
(b) Performance for digits 2 and 9



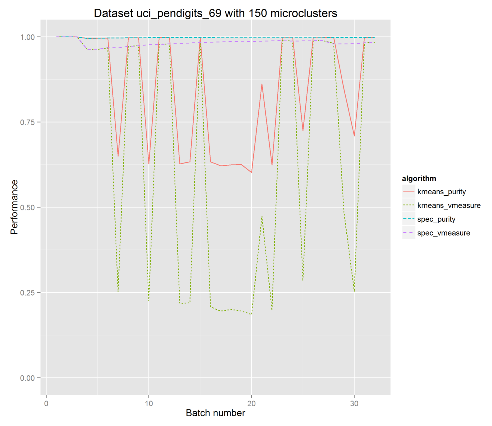
(c) PCA of digits 3 and 6



(d) Performance for digits 3 and 6



(e) PCA of digits 6 and 9



(f) Performance for digits 6 and 9

Figure 1.5.2: Pendigits Pairwise - kmeans vs spectral

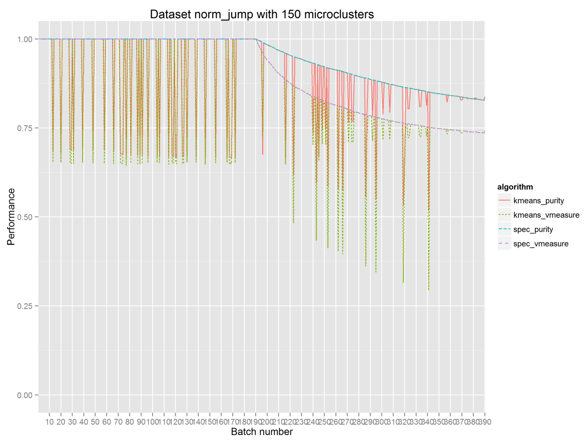


Figure 1.5.3: Clustream - jump data set

1 both purity and V-measure. Would be good to see something similar on real data.

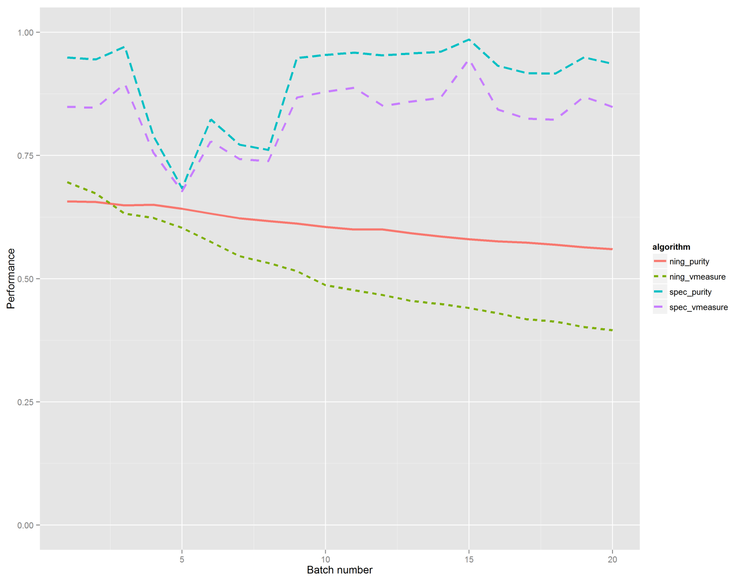
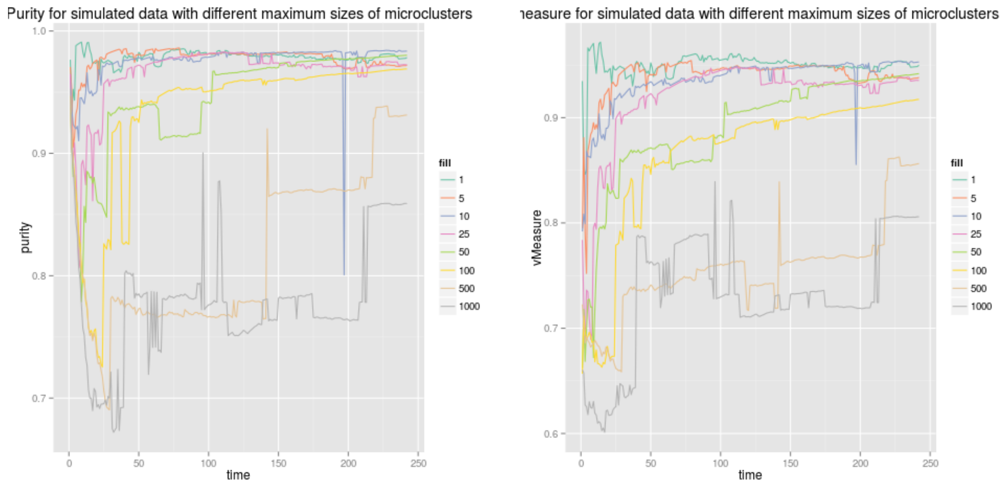


Figure 1.5.4: Ning (no re clustering) vs Spectral Clustream

1.5.5 Optional Extras

Number of microclusters

Give some insight into choosing the number of microclusters q. How does performance change based on the value of q. Aggarwal et al. (2003) suggests using the biggest value of q computationally possible for you. I suspect diminishing returns when increasing q. Can I say something about this? Would be nice to have a plot like Figure 1.5.5.



(a) Purity (b) V-measure

Figure 1.5.5: Effect on performance of the number of microclusters q chosen to capture a data stream

RD: These figures actually show an old method of where microclusters have a fill capacity which we no longer use

Using the microclusters

As highlighted in Section 1.4, there are two ways we could incorporate the microcluster centres into the macro clustering. In the experimentation I have only used method 1. Would be nice to add in method 2 for comparison.

1.6 Conclusion

Bibliography

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

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

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

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

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

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

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

Fowlkes, C., Belongie, S., Chung, F., and Malik, J. (2004). Spectral grouping using the Nyström method. *IEEE transactions on pattern analysis and machine intelligence*, 26(2):214–25.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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