Clustering large datasets

D. P. Mercer, Linacre College

October 2003

Abstract

A review of current techniques for clustering large quantitative data sets is presented. It is found that storing summaries of the original data in a tree improves the scalability of traditional methods to large problems. Density and grid-based techniques are introduced as offering similar scalability, whilst offering the user the ability to discover clusters of arbitrary shapes and an estimate of the number of clusters present in the data.

Contents

Chapter 1

Introduction

1.1 Motivation

The constant advance in computing power makes data collection and storage easier and cheaper than ever before. This has led to the creation of vast datasets in science, government and industry, which present challenging statistical problems.

Unfortunately, the ability to analyse these datasets in a reasonable amount of time and at a reasonable cost has not kept pace. Until we can find a way of unlocking these datasets, the information they contain shall continue to be wasted.

Clustering is one such area where the analysis of large datasets is becoming a problem. Clustering is employed in many areas, finding widespread use in finance, chemistry, marketing and astronomy. Within these fields, analysing large datasets via traditional methods has moved from being tedious, to being infeasible. For example, the first dataset from the Sloan Digital Sky Survey contains the positions and brightness of around 53 million unique objects, or 2.3 terra-bytes. The Centre for Computational Drug Discovery at Oxford University has a catalogue of 3.5 billion molecules being screened for their cancer fighting potential.

1.2 Setting the scene

The development of the clustering algorithms currently in popular use was driven by biologists, working in numerical taxonomy[1] in the 1960s before being promptly taken up by statisticians. The algorithms were based on the distance-space, where the similarity between two objects was quantified by the distance between them as calculated by some distance-metric, see Appendix A. Such metrics are still used in many of the new algorithms being presented.

The problem with analysing the new generation of problems with these methods is one of *scalability*. The algorithms above often have computational complexities that are quadratic

[1] For a detailed account see Sneath and Sokal (1963)

in the number of objects n, or worse. The high orders have a devastating effect on the run-time and memory requirements for large applications.

These difficulties have sparked a great deal of research in many different disciplines. Clustering is now a significant research area in Computer Science, Machine Learning and Statistics. The main developments have been the introduction of *density-based* and *grid-based* clustering methods, as well as improvements in *model-based* methods. Clustering algorithms can be classified as being members of five distinct families:

- Partitioning methods;
- Hierarchical methods;
- Model-based methods;
- Density-based methods; and
- Grid-based methods.

The original algorithms of the biologists and statisticians are members of the first two families, namely partitioning and hierarchical methods. A quick overview of each family is provided below.

Partitioning methods

Given n objects, these methods construct k partitions of the data, by assigning objects to groups, with each partition representing a cluster. Generally, each cluster must contain at least one object; and each object may belong to one and only one cluster, although this can be relaxed.

As it is infeasible to test all partitions for even moderate n and k, partitioning algorithms do not consider all partitions and can only find local optima.

Hierarchical methods

Create a hierarchical decomposition of the objects in the dataset by either merging or splitting clusters sequentially. These are referred to as agglomerative and divisive hierarchical methods, respectively.

Such methods are said to provide multi-resolution *clustering, as a range of clusters at different levels of similarity are returned by the algorithm.*

Model-based methods

Formulate a model and fit it to the data by estimating suitable parameters. The models are typically statistical mixture models or neural networks.

Methods involving statistical models are sometimes said to perform a "conceptual clustering", as clusters are given distributions that govern their behaviour and may suggest some real-world meaning. Neural networks were originally motivated by an abstract attempt to model the way that the brain clusters objects.

Density-based methods

Clusters are defined as dense regions in the data space, i.e. a larger than expected number of points in a given subspace.

Care must be taken (by statisticians) to remember that here "density" refers to the physical concept of density rather than a particular statistical distribution.

Grid-based methods

Characterized by the practice of dividing the data space into a finite number of cells to form a grid. All clustering operations are then performed on the cells of this grid.

Of course it is perhaps dangerous to pigeon-hole new algorithms as having to belong to one and only one of the above families. More, they serve to define some of the distinct styles in clustering; each style having its own set of qualities and problems. We shall discuss some "hybrid" algorithms in Chapter 4.

1.3 Definition of the clustering problem

Clustering algorithms map a "measurement" space onto a "meaning" space. We shall denote the measurement and meaning spaces as $\boldsymbol{X}$ and $\boldsymbol{\Omega}$ respectively. They are quite different in nature.

The "measurement" space $\boldsymbol{X}$ can be multidimensional; it can contain any combination of continuous, discrete or categorical random variables.

The "meaning" space $\boldsymbol{\Omega}$ is a finite and discrete set of labels. After observing an object with measurement $x \in \boldsymbol{X}$, we wish to assign a label, ω to that object in order to assign it to a particular cluster, where $\omega \in \boldsymbol{\Omega}$.

Our definition of the clustering problem, taken in part from Fraley and Raftery (1998) is as follows:

Definition 1.3.1 ("The Problem") *On the basis of quantitative measurements on a set of objects, devise a method that assigns the objects to meaningful subclasses.*

Strictly, the above definition restricts us to "unsupervised" methods; that is, where the algorithm must estimate the number of clusters itself. However, we will see that many of the clustering algorithms presently in use require additional information to be supplied by the user. This might be reasonable when it is possible for the user to explore the data visually and supply a reasonable guess as to the number of clusters, however, in large high-dimensional problems this will not be feasible. Therefore algorithms that ask the user to supply the number of clusters are to be frowned upon. Requiring the user to supply the values of any parameters is also bad (but perhaps unavoidable) practice, as this can influence the final clustering. We shall categorize any method that requires information, other than the data, as "supervised".

Our analysis shall focus on how the complexity of the algorithms depends on n. There are two aspects to bear in mind: the *time*-complexity and the *space*-complexity. The time complexity of an algorithm refers to the time it takes to run. The space-complexity refers to the amount of memory required by the algorithm while it is running. Our primary focus shall be on the time-complexity of the algorithms.

1.4 Desirable properties

Whilst remembering that our principal aim is the processing of large datasets, there are some additional properties that our "ideal" algorithm should have. These include:

- taking a single scan of the data in order to produce clusters;
- the ability to discover clusters with arbitrary shapes and sizes;
- being able to produce high quality clusters in the presence of noise;
- the ability to work unsupervised, see Section 1.3
- the ability to deal with a variety of data types, i.e. binary, ordinal and categorical values;
- insensitivity to the order in which observations are processed; and
- scalability in the presence of high-dimensional data.

1.5 Outline of this report

This report concentrates on the problem of clustering large datasets. It is organized as follows.

A review of traditional clustering methods is presented in Chapter 2. The methods presented are supervised, and so not directly applicable to the definition of our problem in Section 1.3, rather, they are intended to describe common clustering methods that are in popular use.

Chapter 3 presents some attempts at adapting the methods in Chapter 2 so that they scale to large datasets. Methods discussed include: the standard "training set" technique, CLARANS, Fractionization, BIRCH and the use of mrkd-Trees.

Chapter 4 introduces a selection of methods, predominantly from the density and grid-based families, that are currently being used to cluster larger applications. These include: DBSCAN, DENCLUE, STING and DBCLASD, as well as the SOON algorithm and a particle filter.

Chapter 5 attempts to tie together the useful aspects of Chapters 2, 3 and 4, before presenting some ideas for future research.

Chapter 2

Traditional methods

As touched upon in Chapter 1, the clustering techniques in popular use belong to the partitioning, hierarchical and more recently, the model-based families. This chapter presents those algorithms in most widespread use and highlights the reasons that they are poor at tackling large datasets.

2.1 Partitioning methods

Given n objects, these methods construct k partitions of the data, with each partition representing a cluster. The general method works as follows: given the number of clusters, an initial partition is made; objects are then moved between partitions in an attempt to improve some objective function[1].

To find a global optimum for the objective function we need to consider all $N(n,k)$ possible partitions, where:

$$N(n,k) = \frac{1}{k!}\sum_{i=1}^{k}(-1)^{(k-i)}\binom{k}{i}i^n \tag{2.1}$$

$N(n,k)$ is one of Stirling's numbers of the second kind, see Jensen (1969). With increasing n this soon becomes infeasible, so that inevitably, partitioning algorithms do not consider all partitions and may thus only find local optima.

2.1.1 k-means

The k-means method is the standard clustering algorithm, described by Hartigan (1975) and still enjoys widespread use. The method partitions the data into k clusters, where the k is supplied by the user. Clusters are described by the p-vector mean of the objects contained in the cluster, referred to as a *centroid*.

The algorithm partitions the objects so as to minimise the within-cluster discrepancies, where a discrepancy is defined as the difference between an object and its centroid. The

[1] Partitioning methods are often referred to as *Iterative Relocation Algorithms* in the early literature. For a more detailed discussion see Gordon (1999).

standard measure of discrepancy used is the L_2 norm. This leads to the following objective function:

$$A = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - c_j\|^2$$

where x_i denotes the measurement on the i^{th} object and C_j represents the j^{th} cluster, with centroid c_j.

The k-means algorithm is as follows:

1. Create k centroids to initialize the algorithm.

2. Assign each of the n objects to the cluster for which it has the smallest L_2 norm.

3. Update the centroids in light of the current membership.

4. For each object i, where $x_i \in C_j$ calculate:

$$h = \arg\min_{r \neq j} \frac{n_r \|x_i - c_r\|}{n_r - 1}$$

   where n_r is the number of objects assigned to cluster r.

   If $\frac{n_h \|x_i - c_h\|}{n_h - 1} < \frac{n_j \|x_i - c_j\|}{n_j - 1}$ then move object i from cluster j to cluster h.

   Update the values of the relevant centroids.

5. If an object has been moved in the last n calls to Step 4, go to Step 3. Otherwise stop.

A great advantage of the k-means algorithm is that: the time complexity is $O(n)$, making it slightly more scalable; summaries for the clusters are available directly from the algorithm; it can work with any L_p norm and it is robust to the order of the data.

Criticizing the method, we would complain about:

- having to provide the number of clusters k. Common practice is to use a hierarchical clustering method to suggest a suitable k. This also helps with point 4, below;

- the method being sensitive to outliers;

- tending to find spherical clusters of equal size;

- having to find initial centroids to start the algorithm. Hartigan and Wong (1979) suggest using actual objects as initial cluster centres. These could be selected randomly. Hence the final set of clusters is going to be dependent on the initial set of clusters; and

- having to convert to the distance-space every time we need to know how to cluster an object. This raises the computational cost.

2.1.2 Fuzzy k-means

The k-means method described above provides a "hard" clustering. Each object is assigned to one cluster only. A "fuzzy" or "soft" clustering produces degrees of membership for each object belonging to each cluster. This allows us to quantify the confidence we have in each particular labelling. The degree to which object i is a member of cluster j is expressed by the proportion, z_{ij} where the proportions are non-negative and must sum to one over all clusters for each object i.

The clustering problem was recast by Bezdek (1981) as a mathematical optimization problem, using Lagrange multipliers. Clustering proceeds by minimising an objective function A in terms of the membership proportions z

$$A = \sum_i \sum_j z_{ij}^r \|x_i - c_j\|^2; r > 1$$

subject to the following constraints:

1. $z_{ij} \in [0, 1]$ for all i, j.
2. $0 < \sum_{i=1}^n z_{ij} < n$
3. $\sum_{j=1}^k z_{ij} = 1$ for all i.

Note that r can be used to control the degree of fuzziness. Large values of r produce fuzzier clusterings.

Given a set of cluster centres c_j, the optimal membership of an object i to cluster j is given by:

$$z_{ij}^* = \frac{\|x_i - c_j\|^{\frac{-2}{r-1}}}{\sum_{h=1}^k \|x_i - c_h\|^{\frac{-2}{r-1}}}$$

The fuzzy k-means algorithm is then as follows:

1. Generate values for the initial membership coefficients z via some method.
2. Calculate the new centre of each cluster as the weighted average of the data using the current membership coefficients.
3. Calculate the membership coefficients using the optimal rule above.
4. Loop between 2 and 3 until some convergence criterion is satisfied.

This algorithm shares the usual problems of having to specify the number of clusters k and a vague initialization step. However, its attractive features are that:

- the time complexity of Steps 2 and 3 is only $O(n)$;
- a measure of confidence in each assignment of an object to a cluster is readily available through the membership coefficient; and
- summaries of the clusters are available (i.e. the cluster centres) at the end of the algorithm. They are a weighted average of the data.

2.1.3 k-medoids

The k-medoid method partitions a distance-space into k clusters. A medoid is an object that is selected from the dataset represent a cluster. The algorithm selects k medoids to represent the k clusters. Clusters are then created by assigning each of the remaining objects to the nearest medoid. The most common k-medoid algorithm is the Partitioning Around Medoids (PAM) algorithm of Kaufman and Rousseeuw (1990).

The k-medoid algorithm is as follows:

1. Arbitrarily select k objects from the data as medoids.

2. Consider swapping the pair of objects (i,h); where $i \in$ selected objects and h $\in$ non-selected objects. Denote the swap as $i \leftrightarrow h$. Let $d(x_i, x_h)$ be the distance-measure[2] between two objects i and h.
   Now consider another non-selected object j.

   Calculate T_{ih}, the "total swap contribution" for $i \leftrightarrow h$, as

   $$T_{ih} = \sum_j C_{jih}$$

   where C_{jih} is the contribution to $i \leftrightarrow h$ from object j, defined below.

   There are four possibilities to consider when calculating C_{jih}.

   A If j currently belongs to the cluster defined by medoid i (denote cluster i), consider the distance $d(x_j, x_h)$ between object j and object h.

   If h is further from j than the second best medoid i' is from j, then the contribution from object j to the swap is:

   $$C_{jih} = d(x_j, x_{i'}) - d(x_j, x_i)$$

   The result of $i \leftrightarrow h$ would be that object j now belongs to cluster i'.

   Else, if h is closer to j than i' is to j, the contribution from j to the swap is:

   $$C_{jih} = d(x_j, x_h) - d(x_j, x_i)$$

   The result of $i \leftrightarrow h$ would be that object j now belongs to cluster h.

   B If j currently belongs to cluster k, where $k \neq i$, check the distance between object j and object h.

   If h is further from j than the medoid k is from j, then the contribution from object j to the swap is:

   $$C_{jih} = 0$$

   The result of $i \leftrightarrow h$ would be that object j still belongs to cluster k.

[2]For further details see Appendix A.

Else, if h is closer to j than k is to j, the contribution from j to the swap is:

$$C_{jih} = d(x_j, x_h) - d(x_j, x_k)$$

The result of $i \leftrightarrow h$ would be that object j now belongs to cluster h.

3. Let $(i^*, h^*) = \arg\min_{i,h} T_{ih}$. If $T_{i^*h^*} < 0$ then swap $i^* \leftrightarrow h^*$.

   Now object $h \in$ selected objects and $i \in$ non-selected objects.

   Go to Step 2.

4. Allocate each non-selected object to the cluster defined by the nearest medoid.

The most attractive property of this method is its robustness. The use of medoids to define clusters makes this method very resistant against outliers in the data. It does not have to store a vast amount of information in addition to the original data in memory; all that is required is the label of the selected object.

The k-medoids algorithm has a number of disadvantages, namely that:

- the algorithm needs the number of clusters k to be entered as input, hence it is supervised. This requires a good guess from the user, which might not be available;
- the time complexity per iteration of the algorithm is $O(n^2)$. The damage is done in steps 2 and 3 of the algorithm. For each iteration there are $k(n-k)$ $i \leftrightarrow$ h swaps to consider; calculating each swap involves accessing $(n-k)$ distances, making one iteration $O(k(n-k)^2)$; and
- by using medoids, the algorithm does not provide any way of describing the clusters other than in terms of membership details[3].

Other partitioning methods include the Minimal Spanning Tree (MST) algorithm of Kruskal (1957). The length of each edge in the spanning tree corresponds to the distance between two objects; $n(n-1)/2$ edges are required to connect all the objects. The MST is the spanning tree for which the total sum of edges is smallest. Removing the longest $k-1$ edges from the MST provides a partitioning with maximal cluster separation, see Gordon (1999).

2.2 Hierarchical clustering

Hierarchical clustering on the distance space attempts to partition n objects into nested clusters; it is either agglomerative or divisive. Agglomerative methods are "bottom-up", clustering n objects into k clusters by merging the pair of most similar clusters at each stage. Divisive methods are "top-down"; starting from one cluster they seek to divide the data into k partitions by dividing one of the clusters at each stage.

Almost all hierarchical clustering algorithms are agglomerative, as divisive methods present a formidable computational task. At each stage of a divisive algorithm we should consider

[3]If only distance-space data is available, we cannot hope for much more than this.

each way of splitting the data. There are $2^{n-1} - 1$ ways of splitting the data into two groups on the first pass of the algorithm, see Krzanowski (1988). The number of partitions becomes overwhelming even for modest datasets. However, a well-argued method for divisive hierarchical clustering is presented by Kaufman and Rousseeuw (1990).

2.2.1 Agglomerative methods

Agglomerative methods begin by treating each object as a cluster. The algorithm proceeds by merging the two "nearest" clusters. The distance between two clusters i and h is defined by a metric $D_{i,h}$. There is a plethora of metrics available; those favoured include: single-link, complete-link and the group average. The group average metric defines the distance between the two clusters as the Euclidean distance between the centre of gravities of the points in each cluster. A general class of metrics was given by Lance and Williams. Let $D_{k,ij}$ be the distance between cluster k and the union of cluster i and cluster j, then:

$$D_{k,ij} = \alpha_i D_{k,i} + \alpha_j D_{k,j} + \beta D_{i,j} + \gamma |D_{k,i} - D_{k,j}|$$

The agglomerative method is as follows:

1. Consider each object to be a singleton cluster. The $(n \times n)$ distance matrix represents the distances between all possible pairs of clusters.

2. Find the smallest element in the matrix. This corresponds to the pair of clusters that are most similar. Merge these two clusters, say i and h, together.

3. Calculate the distances between the newly formed cluster and the other remaining clusters using a distance metric. Delete the row and column of cluster i and overwrite the row and column of cluster h with the new values.

   This reduces the order of the distance matrix by 1.

4. If the current number of clusters is greater than k, go to step 2; otherwise stop[4].

The advantage of hierarchical methods are that they are easy to implement computationally, and hence reasonably fast to run; they are able to tackle larger datasets than the k-medoids method; and they are unsupervised, in the sense that we can run the algorithm without having to provide the number of clusters. Their disadvantages are that:

- the algorithm, as shown above has $O(n^3)$ time complexity.

  The damage is being done in Step 2. Even though the order of the distance matrix decreases with each iteration, the cost of Step 2 on iteration k is $O((n-k)^2)$, and we are guaranteed $(n-k)$ iterations before we get to k;

- the clusters produced are heavily dependent on the metric $D_{i,j}$.

  Different metrics can and do produce very different clusters. For instance, the complete-link metric tends to produce spherical clusters, whereas the single-link metric produces elongated clusters; and

[4]It is common practice to keep going until all objects are merged into a single cluster. The resulting tree (called a dendrogram) gives the details of the full range of clusters.

- we are left with a wide variety of nested clusters. We still have to decide which clusters, if any, we are going to choose.

2.2.2 Divisive methods

Kaufman and Rousseeuw (1990) give a lucid description of how we might implement a divisive hierarchical algorithm[5]. No attempt is made to consider all, or even a respectable subset of the $N(n, k)$ possible partitions[6].

The idea of an object being "dissatisfied" with the cluster it is currently in is introduced. The most dissatisfied object corresponds to the one that is most distant from all other objects in the cluster. Let the average distance between object i and the cluster C_j be defined as:

$$D_{i,C_j} = n_j^{-1} \sum_{x_h \in C_j} d(x_i, x_h) \tag{2.2}$$

The most dissatisfied object splits off and forms a new cluster. This is equivalent to splitting the cluster with the largest diameter, i.e. that containing the most distant pair of objects.

Objects currently in the cluster being split decide whether they would be happier moving to the new cluster, or staying where they are. Again, average distance to all objects in the group is used. If an object h is closer to another cluster then the move is made. All objects are considered until none of them want to move.

The algorithm continues by splitting the cluster with the largest diameter, until there are n clusters.

Briefly, the algorithm is as follows.

1. Select the cluster containing the most distant pair of objects. This is the cluster with the largest diameter.

2. Within this cluster, find the object with the largest average distance from the other objects. Remove the object from the cluster, letting it form a new singleton cluster.

3. For object h in the cluster being split, calculate the average distance between it and the current cluster; and the average distance between the object and the new cluster.

   If the distance to the new cluster is less than that to the current cluster, move the object h to the new cluster. Loop over all objects in the cluster.

4. If no objects want to move, but the current number of clusters is greater than k, go to 1. Otherwise stop.

A brief discussion of this methods' drawbacks would include that:

- a naive analysis of the algorithm would show it to be of $O(n^3)$ time complexity, $O(n^2)$ on the Step 1 of the algorithm for each iteration. This is even before the potentially expensive calculations that may take place in Step 3, albeit on a subset of the n original objects;

[5] They, in turn, attribute their algorithm to the proposal of MacNaughton-Smith *et al.* (1964)

[6] See equation (2.1).

- Step 3 implies that the group averages between an object and the new and existing clusters need to be recalculated after an object is moved. This will be costly in terms of the number of calculations and the amount of storage required. Also, the effect outliers have on this process is not clear; and
- the method only searches one of the $N(n, k)$ possible partitions.

2.3 Model-based methods

Model-based methods describe any clustering method where a model can be formulated and fit to the data. The process of selecting a model places a great deal of supervision on the clustering method, suggesting that the user has reasonable of knowledge about the structure of the data. Model-based methods struggle to satisfy the requirements of our problem as set out in Definition 1.3.1.

The types of models commonly used tend to be either based on statistical mixture distributions; or neural networks. The discussion of neural network models has been omitted for brevity.

2.3.1 EM algorithm

It is natural to formulate the problem of clustering a set of objects as a missing data problem. The complete set of data would contain the observations made on the objects as well as a label that allocates the object to the correct group. In our case, the labels are missing; all we have to go on is are the measurements recorded for the objects.

The EM algorithm is the standard way of analysing statistical models that have missing data. It is based on the premise that in some cases, if the complete set of data was available the analysis would be easy. Therefore the EM algorithm proceeds by estimating the missing data (the E-Step) and then estimating the parameters of the model, via maximum likelihood (the M-Step).

This approach requires the collection of objects and their clusters to be represented by a statistical model. The data are considered to be a random sample from a mixture of several probability distributions. These probability distributions define the clusters. Each object is generated by one and only one of these distributions; hence belongs to one and only one cluster[7].

The likelihood of the data has a multinomial form and can be defined as follows:

$$L(\boldsymbol{\psi}) = \prod_{i=1}^{n} f(x_i, \boldsymbol{\psi})$$

where $\boldsymbol{\psi}$ is a set of parameters specifying the current model and $f(x, \boldsymbol{\psi})$ is the p.d.f of the mixture distribution.

[7]The sampling mechanism for creating the data can be thought of as a two stage process. First, pick a distribution from the mixture at random. Second, generate a random observation from this distribution.

As each object has been generated by one and only one of these distributions, the joint density of x_i and z_i, $f(x_i, z_i; \boldsymbol{\psi})$ can be written:

$$\prod_{j=1}^{k} \left(\pi_j f_j(x_i)\right)^{z_{ij}}$$

where π_j represents the prior probability that object i came from the component density $f_j(\cdot)$; the component densities typically depend on additional parameters. The z_{ij} are 0-1 indicator variables, indicating whether object i was generated by $f_j(\cdot)$ or not and $z_i = (z_{1i}, \ldots, z_{ki})$ is the vector containing the k indicator variables for object i, $\sum_{j=1}^{k} z_{ji} = 1 \forall i = 1, \ldots n$. These indicator variable represents our missing labels and so are our primary interest. This hard clustering is relaxed so that the z_{ij} can express a soft clustering. This is actually important in removing bias from the estimates, Bryant and Williamson (1978).

The EM algorithm, on the t^{th} iteration is as follows:

1. E-Step: Calculate the expected value of the indicator variables based on the current model and data.

$$z_{ij}^{(t)} = E_{\psi}(z_{ij}|x) = \Pr_{\psi}(z_{ij} = 1|x) = \frac{f_j(x_i)\pi_j^{(t)}}{\sum_{g=1}^{k} f_g(x_i)\pi_g}$$

2. M-Step: Estimate the prior probabilities π, equivalent to the proportion of objects coming from each cluster.

$$\pi_j^{(t+1)} = \sum_{i=1}^{n} z_{ij}^{(t)}/n$$

Within $\boldsymbol{\psi}$ there are the parameters that define the statistical distributions that comprise the model. These parameters also need updating in the M-Step of the algorithm. In the simplest case these updates are weighted averages, using the z as weights. However, if we keep these updates in a separate step (Step 2) we can see the similarity with the fuzzy k-means method.

1. Generate values for the initial membership coefficients z via some method.
2. Calculate the parameters $\boldsymbol{\psi}$ defining the current statistical model as the weighted average of the data using the current membership coefficients.
3. Calculate the membership coefficients using the EM algorithm above.
4. Loop between Steps 2 and 3 until some convergence criterion is satisfied.

The main advantage of the EM algorithm is that it can fit an otherwise intractable statistical model to the data. These will be the subject of a more general discussion elsewhere in the report. The disadvantages of the method are that:

- the algorithm scans the entire dataset for every iteration;
- by imposing a statistical model on the data, we are relying on a large amount of prior knowledge that may or may not be available. This goes against the definition of the ideal method as described in Chapter 1;

- the EM algorithm has a linear rate of convergence and will get very slow for complex models and large datasets. There is a large literature on techniques to speed the algorithm up including: the SEM, Aitken's acceleration method and Moore's kd-tree; and
- the algorithm is dependent on its starting point.

2.4 Notes for clustering large datasets

Firstly, we notice that if we want to reduce our algorithm's computational complexity we need to stop working on the distance space. Methods that work on the distance-space[8] seem to have more acute scalability problems than their vector-space counterparts. These problems arise from the nature of the space. Calculating and storing the relationship between all possible pairs of n objects is $O(n^2)$. Calculating the distance between two objects can be expensive when the measurements are of high dimension. Also, it is often hard for distance-based methods to find suitable ways of summarizing a cluster within the algorithm. There is no natural choice for the "centre" of a cluster and finding one via some ad-hoc method adds to the computational cost. Vector-space methods enjoy some advantages over distance-space methods; compare the complexity of k-means to k-medoids. The clusters formed have natural summary representations; it is straight-forward to find the geometrical centre of a set of objects in p-dimensional space. For a more detailed discussion see Ganti *et al.* (1999).

Secondly, if we impose a statistical model on the vector-space, then we can use statistics estimated from objects in the cluster to represent it. The ability of vector-space models to calculate "reliable" representations of each cluster can be used to improve storage and calculation costs. Using a mechanism defined a priori adds an undesirable level of supervision to the algorithm, but perhaps statistical summaries can be employed in a helpful way.

Chapter 3 presents some methods that attempt to improve upon partitioning, hierarchical and model-based methods.

[8] k-medoids and the two hierarchical algorithms

Chapter 3

Improving traditional methods

3.1 Overview

An obvious way of clustering larger datasets is to try and extend existing methods so that they can cope with a larger number of objects. The focus is on clustering large numbers of objects rather than a small number of objects in high dimensions.

Kaufman and Rousseeuw (1990) suggested the CLARA[1] algorithm for tackling large applications. CLARA extends their k-medoids approach for a large number of objects. It works by clustering a sample from the dataset and then assigns all objects in the dataset to these clusters. The algorithm, briefly, is as follows:

1. Draw a sample from the n objects and cluster it into k groups.

2. Assign each object in the dataset to the nearest group.

3. Store the average distance between the objects and their respective groups.

4. Repeat the process five times, selecting the clustering with the smallest average distance.

While providing a means to assign a large number of objects to groups, CLARA is clearly not ideal. Let M be the maximum number of objects that our clustering method can process in a reasonable time. In cases where $n >> M$, clustering a small sample from the data will often result in some groups present in the data being missed entirely (this is certain in the case where there are $k > M$ groups present in the data).

Although methods are organized underneath "family" headings, this merely reflects the intentions of the authors in their original papers; the methods introduced in this section will extend any clustering algorithm that classifies n objects into k distinct clusters. Ways of selecting the value of k are also suggested.

[1] Clustering LARge Applications

3.2 Partitioning methods

3.2.1 CLARANS

This algorithm was developed by Ng and Han (1994) as a way of improving the CLARA method of Kaufman and Rousseeuw (1990). CLARANS stands for "Clustering Large Applications based on a RANdomized Search". The authors claim that it provides better clusters with a smaller number of searches[2].

Some terminology. A solution, S, to the k-medoids problem is just a set of k unique objects from the dataset. Two solutions are said to be "neighbours" if they have $k-1$ out of the k representative objects in common. Thus each solution S has $k(n-k)$ neighbours. Within the set of all possible solutions, there must be some global minimum representing the best k representative objects.

You will recall from Section 3.1 that CLARA searched within a sample of objects to find the k objects that minimised some measure of average within-cluster distance. This is done using the PAM method of Kaufman and Rousseeuw (1990), which checks every neighbour of a particular solution to see if it can be improved.

Instead of exhaustively searching a random subset of objects, CLARANS proceeds by searching a random subset of the neighbours of a particular solution, S. Thus the search for the best representation is not confined to a local area of the data.

The CLARANS algorithm is governed by two parameters: $\text{MAX}_{\text{neigh}}$, the maximum number of neighbours of S to assess; and MAX_{sol}, the number of local solutions to obtain.

The CLARANS algorithm is as follows:

1. Set S to be an arbitrary set of k representative objects. Set $i = 1$.

2. Set $j = 1$.

3. Consider a neighbour R of S at random. Calculate the total swap contribution (as defined in Section 2.1.3) of the two neighbours.

4. If R has a lower cost, set $R = S$ and go to Step 2.

   Otherwise increment j by one. If $j \leq \text{MAX}_{\text{neigh}}$ go to Step 3.

5. When $j > \text{MAX}_{\text{neigh}}$, compare the cost of S with the best solution found so far. If the cost of S is less, record this cost and the representation.

   Increment i by one. If $i > \text{MAX}_{\text{sol}}$ stop, otherwise go to Step 1.

In the authors' experiments, CLARANS was shown to be more efficient than both PAM and CLARA in terms of run-time. However, these experiments only tested CLARANS on datasets containing up to 100 objects. While such performance is not to be unexpected, we should test CLARANS on some larger problems.

[2]By "searches" they mean the comparison of k alternative objects as representatives of the clusters.

3.3 Hierarchical methods

3.3.1 Fractionization and refractionization

Fractionization was first suggested by Cutting *et al.* (1992) as a way of adapting any hierarchical clustering method so that it could tackle large datasets. Their idea was to split the data into "manageable" subsets (called fractions) and then apply the hierarchical method to each fraction. The clusters resulting from the fractions are then clustered into k groups by the same clustering method. The number of groups, k to be estimated must be supplied in advance.

Fractionization

In the original specification of the algorithm, k is the required number of groups to be estimated, supplied a priori. Let n be the number of objects in the data and M be the maximum number of objects that our clustering procedure can handle in a reasonable time[3].

The fractionization algorithm is as follows:

1. Split the data up into fractions of size M.

2. Cluster each fraction into αM clusters, where $\alpha < 1$. Summarize each new cluster by its mean. Refer to these cluster means as *meta-observations* and treat them as if they were data.

   Note that the original algorithm only retains the mean of the new clusters as the meta-observation. Tantrum et al. *(2002) suggest summarizing each meta-observation by its set of sufficient statistics, in what the call "model-based fractionization".*

3. If the number of *meta-observations* is greater than M, then our clustering method still cannot process them. Go to Step 1, treating the *meta-observations* as if they were data.

   So the algorithm repeats this loop until it has less than M meta-observations.

4. Cluster the *meta-observations* into k clusters using the clustering method of your choice.

5. Classify each individual as belonging to the cluster with the nearest mean.

   Note that if the sufficient statistics have been retained, more sophisticated allocation rules may be used.

The saving comes when we cluster each fraction; the computational effort required to do this is $O(M^2)$ and so is independent of n. On the i^{th} iteration there are $\alpha^{i-1}n/M$ fractions to be processed by the clustering method. Hence total running time is linear in n (the cost incurred by Step 5) and decreasing in α.

[3]Obviously $n \gg M$, otherwise we might be tempted to persevere with our chosen clustering method.

There are, however, a number of problems. The first is the old problem of specifying the number of clusters in advance. The second involves the formation of the *meta-observations*.

The *meta-observations* are summaries, calculated from the objects deemed to form a suitable cluster in Step 2. If this cluster contains objects from different groups, then the value of the *meta-observation* will deviate from that of one expected from a "true" group present in the data. Once the *meta-observation* is formed this error can never be corrected.

Tantrum *et al.* (2002) suggested *refractionization* as a way of combating these two problems. They made the observation that these errors would be less likely if the fractions were more "pure", i.e. predominantly containing objects from only one of the groups. Forming pure fractions would be easy if you knew which objects belonged to which groups.

Refractionization

Given that forming pure fractions is trivial if you have the group labels, refractionization is the repeated application of the fractionization algorithm that processes fractions based on the clusters resulting from the previous fractionization iteration.

The refractionization algorithm is as follows:

1. Split the data up into fractions of size M.

2. Cluster each fraction into αM clusters, where $\alpha < 1$. Summarize each new cluster by its sufficient statistics. Refer to these as *meta-observations*.

3. If the number of *meta-observations* is greater than M then our clustering method still cannot process them. Go to Step 1, treating the *meta-observations'* means as if they were data.

4. Cluster the *meta-observations* into k clusters.

5. Create the fractions for the next iteration as the clusters are formed in Step 4. As soon as a cluster has M objects in it, consider it to be a fraction and remove it from the process.

6. Classify each individual as belonging to a cluster based on its sufficient statistics.

Tantrum *et al.* (2002) gave some indication of the limitations of the refractionization method. Letting n_g be the true number of groups in the data, n_f be the number of fractions after splitting the data and n_m be the number of *meta-observations* after clustering each fraction (as in Step 2 of the algorithm), they note that:

- If $n_g \leq n_m$ then fractionization will work and refractionization is unnecessary. This is because the fractions are more likely to be pure. In the case where $n_g \geq n_m$, some *meta-observations* must be based on clusters that contain two or more groups. Hence they are impure; and

- refractionization will not recover the true groups where $n_g > n_f n_m$. There will be a fraction that contains more than n_m groups, leading to impurity.

In order for refractionization to provide an estimate of the number of clusters, the value of k is determined by the "approximate weight of evidence" (AWE) criterion in Step 4. This requires the user to supply a statistical model for the data. Estimate the number of clusters by maximizing the AWE:

$$\hat{k} = \arg\max_{k}(2L(k) - 2r(3/2 + \log n)) \tag{3.1}$$

where $L(k)$ is the log-likelihood of the k cluster model and r is the number of parameters. The algorithm stops when the number of clusters k stabilizes and changes in cluster composition are sufficiently small.

3.3.2 BIRCH

The BIRCH[4] algorithm proposed by Zhang *et al.* (1996), works in a similar manner to the fractionization algorithm of Cutting *et al.* (1992). Objects in the dataset are arranged into sub-clusters, known as "cluster-features". These cluster-features are then clustered into k groups, using a traditional hierarchical clustering procedure.

A cluster-feature, (CF) represents a set of summary statistics $T(\boldsymbol{x})$ on a subset of the data, e.g.

$$T(\boldsymbol{x}) = \left(n, \sum_{j \in \mathrm{CF}} x_j, \sum_{j \in \mathrm{CF}} x_j x_j^T\right) \tag{3.2}$$

These summary statistics are infact the sufficient statistics for the Gaussian mixture model, although the authors do not make any explicit assumptions about the data.

BIRCH makes use of a tree structure to create and store the cluster-features, referred to as a CF-tree. The tree is built dynamically, one object at a time. The dimensions and size of the tree are determined by two parameters, B and ϵ.

A CF-tree consists of *leaf* and *non-leaf* nodes. A non-leaf node has at most B children, where the children represent cluster features CF_i for $i = 1, \ldots, B$. The non-leaf node represents a cluster made up of the sub-clusters of its children. A leaf node contains at most L entries, where each entry is a cluster-feature CF_j, for $j = 1, \ldots, L$. Leaf nodes represent clusters formed from the sum of its entries.

The parameter ϵ is a tolerance placed on the diameter of a cluster-feature. If the diameter of a cluster-feature exceeds ϵ, then the cluster-feature must be partitioned. Note that the size of a CF-tree is inversely proportional to ϵ.

The BIRCH method is quite simple and is as follows:

1. Scan the data and build the CF tree.
2. If the memory buffer is exceeded during the construction of the tree, treat the leaves of the current tree as if they were objects. Raise the tolerance, ϵ and build another (smaller) tree from the cluster-features and the remaining objects. Loop between Step 1 and Step 2 until the tree fits into main memory.
3. Use the hierarchical clustering method to group the cluster-features into k groups.

[4]Balanced Iterative Reducing using Cluster Hierarchies

4. Allocate objects to the nearest cluster.

Steps 2, 3 and 4 above are relatively straight-forward, whereas Step 1 is rather vague.

A CF tree is built dynamically, one object is inserted at a time. The new object is placed in the most similar leaf node; and then in the most similar entry in that leaf node[5]. If the diameter of the most similar cluster feature exceeds ϵ, then the leaf is split. The two most dissimilar objects are sent to new cluster features and the old points are allocated to the most similar again.

After insertion of a new object, all relevant cluster-features are recalculated and passed up towards the root. If the CF-tree will not fit in main memory then Step 2 of the algorithm is carried out, see above.

3.4 Model-based methods

3.4.1 Mrkd-trees: an implementation of EM algorithm

The EM algorithm, as described in Chapter 2 is notoriously slow. One of causes of this is that it scans the entire dataset on each iteration. Moore (1999) suggested a method using what he calls an mrkd-tree that reduces the number of times the data is accessed. The acronym *mrkd* stands for "multiple-resolution k-dimension", where k is the number of dimensions in the data.

Definition of mrkd-tree structure

An mrkd-tree is a binary tree consisting of nodes that contain certain pieces of information. The tree is built by a recursive partitioning of the dataset, in such a way that the partitions adapt to the local density of the data. Each node can be considered to own a particular subset of points from the data. Different partitions can own a different number of points. Nodes corresponding to dense regions will own many points, whereas nodes corresponding to sparse region will own very few and will often be singletons.

A node can be referred to as either a leaf or a non-leaf node. A non-leaf node has two "children", which in turn are nodes that own the two disjoint sub-partitions of their parent's data points. Leaf nodes are nodes with no "children".

Every node in the tree stores the following information:

- the bounds of the hyper-rectangle that contains all the objects owned by the node; and
- a set of statistics summarizing the data owned by the node.

If a node is a non-leaf node it also contains the following:

- the value on which the partition of the data is made; and

[5]Presumably "most similar" is defined as being smallest change in diameter after inserting the point, although Zhang's paper is not explicit.

- the dimension to which this value refers.

These values are used when traversing the tree to search for data in a particular region of the sample space.

The tree is constructed using a top-down recursive procedure. Given any node that owns a set of points.

1. determine the bounding hyper-rectangle for the set of points.
2. find the widest dimension of the bounding hyper-rectangle.
3. if the widest dimension is greater than some threshold, ϵ_d, then declare the node a leaf and record the points that is owns. Go to Step 4.

   Otherwise partition the points either side of the centre of the widest dimension. Call this centre the split-value and store it in the node, along with the split dimension.
4. if the node is a leaf, stop, otherwise repeat the procedure for the children.

The EM algorithm

In order to complete an iteration of the EM algorithm, for a Gaussian model, we simply need the set of sufficient statistics:

$$\left\{ \sum_{i=1}^{n} z_{ij}, \sum_{i=1}^{n} z_{ij} x_i, \sum_{i=1}^{n} z_{ij} x_i x_i^T \right\} \tag{3.3}$$

for $j = 1, \ldots, k$ and x and z are defined as in Section 2.3.1.

The EM algorithm is then:

1. E-Step: Calculate the expected value of the indicator variables based on the current model and data.

   $$z_{ij}^{(t)} = E_\psi(z_{ij}|x) = \Pr_{\boldsymbol{\psi}}(z_{ij} = 1|x) = \frac{f_j(x_i)\pi_j^{(t)}}{\sum_{g=1}^{k} f_g(x_i)\pi_g^{(t)}}$$

   where $\boldsymbol{\psi}$ is the set of parameters defining the model.
2. M-Step: Estimate the parameter estimates based on the set of sufficient statistics

   $$\begin{aligned} \pi_j^{(t+1)} &= \sum_{i=1}^{n} z_{ij}^{(t)}/n \\ \mu_j^{(t+1)} &= \sum_{i=1}^{n} z_{ij}^{(t)} x_i / \sum_{i=1}^{n} z_{ij}^{(t)} \\ \Sigma_j^{(t+1)} &= \sum_{i=1}^{n} z_{ij}^{(t)} x_i x_i^T / \sum_{i=1}^{n} z_{ij}^{(t)} x_i - \mu_j^{(t+1)} \mu_j^{(t+1)T} \end{aligned} \tag{3.4}$$

   for $j = 1, \ldots, k$.

The EM algorithm can be invoked on an mrkd-tree by calling some function $m(\cdot)$ on the root node; where $m(\cdot)$ is a function that returns the set of sufficient statistics for a given node.

If $m(\cdot)$ is called on a leaf node, r, then we calculate:

$$\bar{z}_j = \Pr(x \in C_j | \bar{x}, \boldsymbol{\psi}) = \frac{f(\bar{x}|x \in C_j, \boldsymbol{\psi}) \Pr(x \in C_j|\psi)}{\sum_{h=1}^{k} f(\bar{x}|x \in C_h, \boldsymbol{\psi}) \Pr(x \in C_h|\psi)} \tag{3.5}$$

where $\bar{x}$ is the centroid of the points owned by the node and C_j refers to cluster j for $j = 1, \ldots, k$. We return the approximation to the sufficient statistic:

$$\begin{aligned} \sum_{i=1}^{n_r} z_{ij} &\approx \bar{z}_j \times n_r \\ \sum_{i=1}^{n_r} z_{ij} x_i &\approx \bar{z}_j \times n_r \times \bar{x} \\ \sum_{i=1}^{n_r} z_{ij} x_i x_i^T &\approx \bar{z}_j \times n_r \times \bar{x} \times S_r \end{aligned} \tag{3.6}$$

for $j = 1, \ldots, k$, where n_r is the number of points owned by node r and S_r is the sample covariance of the data owned by node r.

If $m(\cdot)$ is called on a non-leaf node, we simple call the function recursively on its children. The returned values of the sufficient statistics are added together as we go.

Note that with a tree constructed using $\epsilon_b = 0$, (i.e. it has n leaves, each containing a single data point), there will be no computational improvement.

An iteration of the EM algorithm traverses the entire tree and therefore scans the entire dataset, doing $O(nk^2)$ work per iteration. However, we can improve in this if we can "prune" the tree above some of the leaves. By pruning at a non-leaf node we are stating that all the data-points owned by the node are suitably similar; we can evaluate the node as if it were a leaf, without searching its descendants.

The idea is that if the points owned by the leaf are very similar there will be little variation in the z_{ij}, encouraging us to assume that $\sum_{i=1}^{n} z_{ij} x_i \approx \bar{z}_j \sum_{i=1}^{n} x_i$ [6]. In order to do this we need to calculate $z_j^{\min} = \min_{x_i \in \text{Node}(r)} z_{ij}$ and $z_j^{\max} = \max_{x_i \in \text{Node}(r)} z_{ij}$ at each node, for all j. These are the lower and upper bounds on the z_{ij} for the node.

It turns out that this is not trivial; $z_j^{\min}$ and $z_j^{\max}$ depend on the mean and covariances of all the component densities, not just that of the j^{th}. Moore (1999) solved this via computational geometry, using the Mahalanobis distance to work out lower and upper bounds on the value of $f_j(x)$ within the hyper-rectangle[7]. The details are not given here.

Moore (1999) suggests the following pruning criterion: prune at node if $z_j^{\min} - z_j^{\max} < \tau z_j^{\text{total}}$, where $z_j^{\text{total}} = \sum_{i=1}^{n} z_{ij}$ is the total weight given to the j^{th} cluster over the entire

[6] Constructing the tree so that leaves contain relatively few points will help to ensure this approximation is reasonable.

[7] The largest possible value of the Mahalanobis distance within the hyper-rectangle corresponds with the smallest value of $f_j(x)$, and vice versa

dataset and τ is a small positive constant. Note that this criterion is favoured over the more obvious $z_j^{\min} - z_j^{\max} < \epsilon$, as it was noticed that clusters with larger total weight over the entire dataset could tolerate much looser bounds.

Moore (1999) applied this problem to a range of simulated Gaussian mixtures and reports eight-fold to 1000-fold speedups over the vanilla EM algorithm.

In Chapter 4 we will review some methods that do not rely on existing clustering techniques to cluster large datasets.

Chapter 4

Current methods

The development of clustering techniques has typically proceeded along heuristics lines. This chapter contains some influential methods from the density, grid and model-based schools. It concludes with an interesting hybrid.

4.1 Density-based methods

4.1.1 DBSCAN

DBSCAN[1] is a density-based algorithm for spatial data presented by Ester *et al.* (1996); it is very popular in the family of density-based clustering methods. The algorithm's most attractive features include: an ability to identify clusters with arbitrary shapes; the ability to cluster in the presence of noise; and a low order dependence.

Density-based describe clusters as regions of the sample space with a high density of points, compared to sparse regions. The key idea here is of each point in the data having its own "neighbourhood" in the sample space[2]. We are then interested in the "density" of points in that neighbourhood.

An informal definition of a cluster might be as follows: for a point in a cluster, the neighbourhood of that point–with a given radius–must contain a minimum number of points. This is equivalent to the density in the neighbourhood of the point exceeding some threshold.

Definition 4.1.1 *Let $\boldsymbol{x}$ be data, $\boldsymbol{x} \in \boldsymbol{X}$. The ϵ-neighbourhood.*

The ϵ-neighbourhood of an arbitrary point p, denoted $N_\epsilon(p)$ is defined as the set:

$$N_\epsilon(p) = \{x \in \boldsymbol{x} | d(p, x) < \epsilon\} \tag{4.1}$$

[1] "Density Based Spatial Clustering of Applications with Noise"

[2] For a point $x \in \mathbb{R}^p$ the "neighbourhood" refers to a set of points that in turn describe a hyper-rectangle in p dimensions.

Note that the shape of the neighbourhood is determined by a distance function, $d(\cdot, \cdot)$, similar to those met in Chapter 2.

Definition 4.1.1 immediately suggests an approach: define a cluster by requiring that there are a minimum number of points, m in the ϵ-neighbourhood of each of its constituent points. Unfortunately this approach fails, as clusters contain two types of points: "core" points in the middle of the cluster and "boundary" points at the edge. We would expect boundary points to have fewer points in their ϵ-neighbourhood than a core point.

To proceed we must define notions of "reachability" and "connectivity".

Definition 4.1.2 *Directly density-reachable.*

An arbitrary point p is directly density-reachable from a point q, w.r.t ϵ and m if:

$$p \in N_\epsilon(q)$$

$$\|N_\epsilon(q)\| \geq m \tag{4.2}$$

where $\|\cdot\|$ represents the cardinality of a set and (4.2) is known as the "core point" condition.

Definition 4.1.3 *Density-reachable.*

An arbitrary point p is density-reachable from a point q, (w.r.t ϵ,m) if there is a chain of points $p_1, \ldots, p_n$–where $p_1 = p$ and $p_n = q$–such that p_{i+1} is directly density-reachable from p_i.

In order for two boundary points in a cluster C to be density-reachable, there must be a core point in C from with both points are density reachable. This introduces the idea of *connectivity.*

Definition 4.1.4 *Density-connectivity.*

A point p is density-connected to a point q, if there is a point r such that p and q are density reachable from r w.r.t ϵ and m.

We can now define a cluster. Informally, a cluster is defined to be a set of density-connected points which is maximal with respect to density-reachability.

Definition 4.1.5 *A Cluster*

A Cluster C, w.r.t ϵ and m is a non-empty subset of $\boldsymbol{x}$ satisfying:

1. *Maximality. $\forall r, s$: if $x_r \in C$ and x_s is density-reachable from x_r, w.r.t ϵ and m, then $x_s \in C$.*
2. *Connectivity. $\forall r, s \in C$: x_r is density-connected to x_s w.r.t ϵ and m.*

An undesirable feature of this set up is that any cluster defined by ϵ and m must have at least m points. This is because any point in the cluster will be density-connected to itself via some point and this point must satisfy the core-point condition (4.2).

The DBSCAN algorithm is as follows:

1. Select an arbitrary point x.

2. Find all points that are density-reachable from x.

   If x is a core point then we have formed a cluster. If x is a boundary point then no points are density-reachable from x.

3. Repeat for the next point in the data.

If two clusters of different density are sufficiently close to one another, they may have been merged erroneously by the algorithm. Two clusters having at least the same density will be separated if $D(C_i, C_j) > \epsilon$, where $D(C_i, C_j) = \min\{d(x_r, x_s) | r \in C_i, s \in C_j\}$. A recursive run of DBSCAN, with an increased ϵ, on each of the clusters will be necessary.

The parameters ϵ and m need to be supplied. The authors suggest that they are estimated by a heuristic approach given in their paper. Briefly, the distance from all points to their k^{th} nearest-neighbour is calculated[3], sorted and plotted. The graph will often contain a kink indicating the desired distance. ϵ is set to this value.

The DBSCAN algorithm has a number of desirable properties. Namely that:

- it can find clusters of an arbitrary shape;
- it has low order time complexity, $O(n \cdot \log n)$, in terms of n. Calculation of an ϵ-neighbourhood is $O(\log n)$, as the data can be stored in an R*-tree, Beckmann *et al.* (1990). We have n neighbourhoods to calculate for the data;
- the number of clusters is not explicitly supplied by the user. The number of clusters will depend on the value chosen for ϵ (although this is estimated from the data) and the choice of distance function; and
- the algorithm can cluster objects in the presence of noise.

Noise is defined formally as follows:

Definition 4.1.6 *Noise*

Let $C_1, \ldots, C_k$ be clusters in the data, w.r.t ϵ and m. Then noise is defined as the set of points in $\boldsymbol{x}$ not belonging to any cluster:

$$Noise = \{x \in \boldsymbol{x} | \forall j : x \notin C_j\} \tag{4.3}$$

for $j = 1, \ldots, k$.

All points that are not density-reachable from any other point in the data are labelled noise; these points are removed after all points have been considered and are not included in the later recursive calls of DBSCAN.

A major and unintended drawback is that clusters must have a minimum of m points, by virtue of the definitions of density-connectivity. This makes it impossible for DBSCAN to find very small clusters within large datasets.

The OPTICS algorithm was proposed by Ankerst *et al.* (1999) as a multi-resolution extension to DBSCAN. In DBSCAN, clusters with very high density can be completely

[3]The authors use $k = 4$.

contained in clusters of a lower density, due to the use of a global ϵ parameter. Running DBSCAN a number of times, over a set of decreasing[4] ϵ parameter will produces an ordering of density-based clusters. The complexity of OPTICS is clearly $O(N(\epsilon) \cdot n \cdot \log n)$, where $N(\epsilon)$ is the number of parameters in the set.

4.1.2 DENCLUE

DENCLUE[5] uses mathematical functions known as *influence* function to model the impact of a object within that object's neighbourhood. The density of the data space is then calculated as the sum of the influence functions over all objects. Clusters (called "density-attractors") are then defined as the local maxima of the overall density function.

Definition 4.1.7 *Influence function.*

The influence function of an object $x \in \boldsymbol{x}$ is a function $f_B^x : \mathbb{R}^p \to \mathbb{R}_0^+$, denoted equivalently by:

$$f_B^x(y) = f_B(y, x) \tag{4.4}$$

Note that influence functions usually rely on a distance function $d(x, y)$; popular influence functions include: the *Square Wave*, where $f_{sq}(y, x) = 1$ if $d(x, y) < \sigma$, 0 otherwise; and the *Gaussian*, $f_G(y, x) = \exp(-d(x, y)^2/2\sigma^2)$, $\sigma > 0$ in both cases.

Definition 4.1.8 *Density function.*

The density function is defined as the sum of the influence functions of all the objects in the data:

$$f_B^D(y) = \sum_{i=1}^{n} f_B^{x_i}(y) \tag{4.5}$$

Definition 4.1.9 *Density-attractor.*

A point $x^* \in \mathbb{R}^p$ is called a density-attractor for a given influence function if and only if x^* is a local maximum of the density function f_B^D.

There are two types of a cluster.

Definition 4.1.10 *Centre-defined cluster.*

A Centre-defined cluster C with respect to (σ,ϵ) and a density-attractor x^* is a subset of objects for which $x \in C$ are density-attracted to x^* and $f_B^D(x^*) \geq \epsilon$.

Note that objects attracted by a point x^* for which $f_B^D(x^*) \leq \epsilon$ are labelled as outliers.

Definition 4.1.11 *Arbitrary cluster.*

An arbitrary cluster C with respect to (σ,ϵ) and a set of density-attractors, A, is a subset of objects where:

[4] Starting with the smallest ϵ ensures that the high density clusters are create first.

[5] "DENsity-based CLUstEring"

1. *$\forall x \in C$ there exists $\{x^*\} \in A : f_B^D(x^*) \geq \epsilon$, i.e. x is directly attracted to something; and*

2. *$\forall x_1^*, \forall x_2^* \in A$ there exists a path $P \subset \mathbb{R}^p$ from x_1^* to $\forall x_2^*$ with $f_B^D(p) \geq \epsilon$, $\forall p \in P$.*

Notice part (2) of the above definition is similar to the definitions of *connectivity* in 4.1.2 and 4.1.4.

The DENCLUE algorithm works by finding an efficient way to calculate the density function and density-attractors. To speed calculation when calculating the density for a point y, only objects that are close to y are considered. Other points can be omitted from the calculation with significant error; for more details see Hinneburg and Keim (1998).

Definition 4.1.12 *Local density function.*

Let the function $Near(y) = \{x \in \boldsymbol{x} : d(x, y) \leq \sigma_{Near}\}$. Then the local density function $\hat{f}^D(y)$ is:

$$\hat{f}^D(y) = \sum_{x_i \in Near(y)} f_B^{x_i}(y) \tag{4.6}$$

The DENCLUE algorithm works as follows:

1. Find the minimal bounding hyper-rectangle of the data and divide into p-dimensional hyper-cubes of length 2σ. Only cubes that contain data need to be determined. Refer to them as being "populated", denoting the set of populated cubes as C_{pop}.

2. Store statistics for all cubes in C_{pop} (i.e. the cube Means $\bar{c}$, $c \in C_{\text{pop}}$).

3. Identify pairs of "connected" cubes as being those for which $d(\bar{c}_i, \bar{c}_j) \leq 4\sigma$.

   As calculating this would be $O(C_{pop}^2)$, we introduce a further threshold ϵ_h to create a set of highly-populated cubes $C_{h.pop} = \{c \in C_{pop} | n_c \geq \epsilon_h\}$, where n_c is the number of objects in cube c.

   In general, $\|C_{h.pop}\| << \|C_{pop}\|$. Connecting highly populated cubes to their neighbours (including populated neighbours) is $O(\|C_{h.pop}\| \cdot \|C_{pop}\|)$. The authors suggest $\epsilon_h = \epsilon/2p$.

4. Calculate the overall density and determine the density-attractors using a hill-climbing procedure on $\hat{f}^D$ and $\bigtriangledown \hat{f}^D$.

5. Assign objects to clusters defined by the density attractors.

The complexity of DENCLUE is linear in n, for scanning the data and calculating cube statistics. All extra complexity has an order dependent on the cardinality of the cubes sets. Any outliers are considered to be noise and are removed from the process, allowing clusters of arbitrary shape can be found in the presence of noise.

More importantly, the use of influence functions allows DENCLUE to generalize a set of density-based methods. Using the square-wave function with $\sigma = \epsilon_{\text{DBSCAN}}$ and $\epsilon = m_{\text{DBSCAN}}$ gives the DBSCAN algorithm.

4.2 Grid-based methods

4.2.1 STING

Wang *et al.* (1997) present a multi-resolution summary of a dataset, known as a "STatistical INformation Grid", that can be used to answer database queries efficiently. Querying the density of cells in the grid provides a grid-based clustering method similar to DBSCAN.

The data space is recursively divided into rectangular cells. All non-leaf cells are partitioned to form child cells. Sufficient statistics for the objects bounded by the hyper-rectangle of the cell are stored. Once the bottom layer of cells have been determined, the statistics can be determined in a bottom-up fashion. Wang *et al.* (1997) store the following at each cell:

- the number of objects;
- the mean;
- the standard deviation;
- the minimal and maximal values of objects; and
- a statistical distribution label, e.g. "Normal", "Uniform" or "None".

Only the number of objects and the mean of the points would be required for the clustering method. Trying to merge cells with different distribution labels seems particularly messy.

The STING grid answers queries as follows. Given any starting level of cells, each cell is given a probability–with a confidence region–of being "relevant" to the query. Irrelevant cells can be ignored, while relevant cells are followed down a layer until we reach the bottom of the grid. At the bottom of the grid we have a list of relevant cells containing information with which to answer the query.

To perform a clustering, a secondary stage is required: the neighbours of all relevant cells are tested for their relevancy. The algorithm proceeds as follows:

1. Start at the root cell.
2. For all cells in the current layer, calculate the probability and confidence region that the cell is relevant and label it accordingly.

   "Relevancy" is whether the density of a cell is likely to be greater than the query density.
3. If the current layer is the bottom layer, proceed. Otherwise repeat Step 2 for the children of the relevant cells in current layer.
4. For all relevant cells, examine all cells within a given distance from the current cell.
5. Put all relevant cells that have not been already discovered in a queue and repeat the Step 4 until the queue is empty. You now have one cluster.
6. Repeat Step 4 and 5 until all relevant cells have been assessed. Then stop.

Calculating probabilities and intervals for any cell in Step 2 will be $O(1)$. The total number of cells in the grid will be $O(B)$, where there are B cells on the bottom layer of the grid. Hence total calculations involving all relevant cells will be $O(B)$. The only dependence on n will be in creating the grid initially.

Clustering via a STING grid can find clusters of arbitrary shape, but these clusters tend to be slightly crude due to their rectangular nature. Outliers are handled, as cells containing them will be dismissed as irrelevant when travelling through the grid.

It is not clear how well this method will scale for larger dimensions; the original paper only looks at two dimensions.

4.3 Model-based methods

4.3.1 Particle filters

A particle filters is a sequential method that allows Monte Carlo techniques to be applied to dynamic state-space systems. For linear Gaussian models analysis may proceed via the Kalman filter. For non-linear non-Gaussian cases the Kalman filter does not extend simply. Particle filters have been proposed as approximate estimation methods. A comprehensive review of particle filters may be found in Doucet *et al.* (2001).

The particle filter estimates a quantity of interest–usually the posterior distribution of the unknown parameters–with a set of N weighted particles. If the state-space of the model at time t is denoted as s_t, then particle i is simply a realization of a particular state, $s_t^{(i)}$. Each particle is assigned a weight $w_t^{(i)}$.

A particle filter is therefore the set of particles and weights, $\{s_t^{(i)}, w_t^{(i)}\}_{i=1}^N$. Any function of interest, say $g(s_t)$, can be approximated by:

$$\mathrm{E}(g(S_t)|\boldsymbol{x}_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)} g(s_t^{(i)}) \tag{4.7}$$

where $\boldsymbol{x}_{1:t}$ is the data observed up to time t. When we receive a new observation, x_{t+1}, the set of particles and weights are updated sequentially.

Chopin (2002) and then Fearnhead (2002) noted that the clustering problem can be formulated as a sequential problem if you view the data as being observed in a particular order. For an existing dataset a random order would need to be imposed, but it is easy to envisage many clustering problems that occur in real time and therefore do have a natural order.

Adapting our notation from Chapter 2, let $\boldsymbol{\psi}_t$ represent the unknown parameter at time t. This parameter can be partitioned as $\boldsymbol{\psi}_t = \{k_t, \boldsymbol{\theta}_t, \boldsymbol{z}_t\}$, where: k_t is the number of clusters in the model; $\boldsymbol{\theta}_t$ represents the unknown parameters of the k_t component densities; and $\boldsymbol{z}_t$ is the 0-1 classification matrix assigning objects to clusters. Note that for clusters represented by Gaussian distributions, $\boldsymbol{\theta}_t$ can be further partitioned as $\boldsymbol{\theta}_t = \{\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\}$, the location vectors and covariance matrices of the component densities respectively.

Let $\boldsymbol{\psi}_t$ be the state of the model at time t. Then $\{\boldsymbol{\psi}_t^{(i)}\}_{i=1}^N$ is a set of N representative particles at time t. A particle $\boldsymbol{\psi}_t^{(i)}$ is a particular instance of our probabilistic model, containing:

- the number clusters;
- the locations and covariances of the component densities; and
- the assignments of objects to clusters. Note that from these assignments the prior probabilities of group membership can be ascertained.

The particle filter method is as follows. Each new observation signifies the start of the next iteration. The new observation x_t is placed in each prospective cluster of each particle $\{\boldsymbol{\psi}_t^{(i)}\}_{i=1}^N$, to create a new particle. Within each existing particle $\boldsymbol{\psi}_t^{(i)}$ there are $k_t^{(i)}+1$ possible assignments, (the particle can join one of the $k_t^{(i)}$ clusters, or from a new singleton group), creating $k_t^{(i)}+1$ new particles.

The weight assigned to each new particle should reflect the likelihood of the data observed so far, arising from the particular model represented by the particle. This likelihood can be factorized as the product of the likelihood of the existing "parent" particle and the likelihood of the new object being a member of the cluster to which it was assigned.

$$\begin{aligned} w_t^{(i)} &= f(\boldsymbol{x}_{1:t}|\boldsymbol{\psi}_t^{(i)}) \\ &= f(\boldsymbol{x}_{1:t}|\boldsymbol{\theta}_t^{(i)}, \boldsymbol{z}_t^{(i)}) \\ &\approx f(x_t|\boldsymbol{\theta}_{t-1}^{(i)}, \boldsymbol{z}_{t-1}^{(i)}) f(\boldsymbol{x}_{1:t-1}|\boldsymbol{\theta}_{t-1}^{(i)}, \boldsymbol{z}_{t-1}^{(i)}) \\ &= f(x_t|\boldsymbol{\theta}_{t-1}^{(i)}, \boldsymbol{z}_{t-1}^{(i)}) w_{t-1}^{(i)} \\ &= f(x_t|\boldsymbol{\theta}_{t-1}^{(i)}) \Pr(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}^{(i)}) w_{t-1}^{(i)} \end{aligned} \tag{4.8}$$

In all, a total of $M = \sum_{i=1}^N (k_t^{(i)}+1)$ new particles are created. This means that the number of particles created by the algorithm will grow exponentially with each new observation. In order to stop things getting out of control, the number of particles is reduced to N before the next object is observed. This is done via resampling.

If the M particles represent the density of interest, we can create the required sample of N particles by sampling from the M available. This is equivalent to a multinomial sampling of the particles $\{\boldsymbol{\psi}_t^{(j)}\}_{j=1}^M$, such that $(K_1, \ldots, K_M)$ has a multinomial distribution, $\mathrm{Mu}(N; w_t^{(1)}, \ldots, w_t^{(M)})$, where K_j is the number of times particle $\boldsymbol{\psi}_t^{(j)}$ is resampled, see Gordon *et al.* (1993). Many improvements to this resampling method exist, see Liu and Chen (1998), Carpenter *et al.* (1999), Chen and Liu (2000) and Fearnhead and Clifford (2003).

Briefly, the method of Fearnhead and Clifford (2003) works using the following idea. Retain all particles with (normalized) weight greater than $1/c$, (where c is the solution of $\sum_{j=1}^M \min(cw_n^{(j)}, 1) = N$). Resample the remaining particles from those that are not retained, giving them each weight $1/c$.

The particle filter algorithm is as follows:

1. Initialize algorithm by creating N particles $\{\boldsymbol{\psi}_0^{(i)}\}_{i=1}^N$, assigning them all equal weight N^{-1}.

2. For a new object x_t, create M new particles by inserting the object into each cluster of each particle. Update all parameters contained in the particle.

3. Assign each particle a weight via (4.8).

4. Resample N particles from the M new particles.

5. If there are still objects to process, go to Step 2. Otherwise Stop.

Each iteration of the particle filter processes a new object. The advantage is that the complexity of each iteration is $O(N)$; it likely that $N << n$ for large datasets. Hence, for batch problems such as ours the total complexity will be $O(nN)$, so linear in n.

4.3.2 SOON

The SOON[6] approach, presented by Frigui and Rhouma (2001), uses a neural network to organize a set of objects into k stable and structured clusters. The value of k is found in an unsupervised manner.

Their method is based on the "Self Organizing Map" (SOM) method of Kohonen[7], which places a large number of spatially correlated representational objects in a one or two dimensional space. Representatives that are close together are considered similar. Each time an observation is presented, the closest representative is found and the values of all representatives are updated. The data are presented until convergence. For further details and original references see Ripley (1996).

With the SOON method, each object in the data, O_j, is represented as an "Integrate and Fire" oscillator, characterized by a phase ψ_j and state y_j, where:

$$y_j = f(\psi_j) \quad = \quad \frac{1}{b}\log\left\{1 + (e^b - 1)\psi_j\right\} \tag{4.9}$$

where $0 \leq \psi_j \leq 1$ and $0 \leq y_j \leq 1$ for $j = 1, \ldots, n$; $f(\cdot)$ is smooth, monotonically increasing with $f(0) = 0$ and $f(1) = 1$; b is a constant determining how $f(\cdot)$ is concave down (usually $b = 2$).

Whenever an oscillator's state reaches the threshold ($y_j = 1$), it "fires", with the following consequences:

- the oscillators phase and state, ψ_j and y_j are reset to zero; and
- the phase of all the other oscillators change by an amount $\epsilon_j(\psi_h)$, for $h = 1, \ldots, n; h \neq j$.

Changing the phase of the other oscillators has the effect of either "exciting" or "inhibiting" them. An oscillator is excited by having its phase increased; it is inhibited by decreasing its phase. The precise amount of the change is determined by the *coupling* function

[6] Self organizing oscillator networks

[7] All though their paper does not make direct reference to this.

$\epsilon_j(\psi_h)$, which in turn depends on the dissimilarity between the two oscillators (equivalently objects) in question. A typical coupling function would be as follows:

$$\begin{aligned} C_{\mathrm{E}}\left[1-(\frac{d_{ij}}{d_0})^2\right] \qquad & \text{if } d_{ij} \leq d_0 \\ C_{\mathrm{I}}\left[(\frac{d_{ij}-1}{d_0-1})^2-1\right] \qquad & \text{if } d_{ij} > d_0 \end{aligned} \tag{4.10}$$

where $d_{ij} = d(x_i, x_j)$ is short-hand for the measure of dissimilarity between two objects i and j, and d_0 is a threshold dissimilarity that determines the cutoff for what is deem "similar". d_0 can be viewed as a *resolution* parameter, as it affects the number of groups created. C_{E} and C_{I} are the maximum excitory and inhibitory couplings permitted.

The firing of an oscillator tends to excite a few oscillators, whilst inhibiting many others. Thus an oscillator receives much more inhibition than excitement; hence is it common that $C_{\mathrm{E}} > C_{\mathrm{I}}$, although this is not necessary for the method to work[8].

Once a set of oscillators are synchronized, the way that members of the set interact with other oscillators not in the set must be made uniform. This is so that the set remains synchronized.

If $S = \{O_{s_1}, \ldots, O_{s_r}\}$ represents as set of synchronized oscillators, we must set $\epsilon_j(\psi_{s_1}) = \epsilon_j(\psi_{s_2}) = \ldots = \epsilon_j(\psi_{s_r})$. This is achieved by making the dissimilarities between an object in S and another object i, not in S, equal for all objects in S (i.e. $d_{is_1} = d_{is_2} = \ldots = d_{is_r} = \hat{d}_{iS}$.) There are many choices for $\hat{d}_{iS}$; common ones include:

$$\begin{aligned} \hat{d}_{iS} &= \min(d_{is_1}, d_{is_2}, \ldots, d_{is_r}) \\ \hat{d}_{iS} &= \max(d_{is_1}, d_{is_2}, \ldots, d_{is_r}) \\ \hat{d}_{iS} &= \frac{1}{r}\sum_{j=1}^{r} d_{is_j} \end{aligned} \tag{4.11}$$

Note that these are just the *single*, *complete* and *average* link functions, common to hierarchical clustering algorithms.

The SOON algorithm is as follows:

1. Initialize the phases ψ_j at random; $j = 1, \ldots, n$.
2. Identify the next oscillator to "fire". Denote this oscillator as O_i

   This is just the oscillator with the largest phase.
3. Bring O_i to threshold and adjust the other phases, $\psi_j \rightarrow \psi_j + (1 - \psi_i)$ for $j = 1, \ldots, n$; $j \neq i$.
4. For oscillators O_j for $j = 1, \ldots, n$; $j \neq i$:
   - Compute y_j using 4.9.
   - Compute the coupling effect using 4.10.

[8] all that is necessary is that the excitory coupling causes a positive change in phase and the inhibitory a negative change.

- Adjust the state y_j using a clipping function.
- Compute the new phases $\psi_j = f^{-1}(y_j)$.
- Reset ψ_i and y_i to zero.

5. Identify any oscillators that have synchronized. Reset their phases and update their dissimilarities using 4.11.
6. If all synchronized groups have stabilized, stop. Otherwise go to Step(2).

The SOON algorithm provides a novel way of clustering data. The authors claim that it can cluster large dataset into an unsupervised number of groups. This is perhaps a slight overstatement, as:

- the algorithm works on the dissimilarity matrix, which needs to be stored in memory at $O(n^2)$ cost. Steps 3 and 4 have quadratic time complexity in n per iteration;
- while the user does not have to specify a value for k and the process does estimate its own value, the number of synchronized groups can be controlled in part by the resolution parameter d_0. A small d_0 creates many small clusters, whilst a large d_0 few large clusters; and
- the values of C_{E}, C_{I} and b are crucial in determining the rate of convergence of the groups, as well as which oscillators get the chance to fire.

4.4 Hybrid methods

4.4.1 DBCLASD

DBCLASD stands for "Distribution-Based Clustering Algorithm for CLustering LArge Spatial Datasets", Xu *et al.* (1998). It is a hybrid of the model-based, density-based and grid-based approaches.

The key idea for identifying clusters is that of density-based clustering, that is within the sample space, clusters appear to be more dense than other regions. A distinguishing characteristic of dense regions is that the nearest-neighbour distances for points inside the region are smaller than for points outside the region.

Therefore the probability distribution of the nearest-neighbour distance can be used to describe the characteristics of a cluster formed from a set of points. This distribution can be be used when testing whether a neighbouring point should be included in the cluster or not.

Definition 4.4.1 *Nearest-neighbour of a point and Nearest-neighbour distance.*

Let q be a query point and S be a set of points. The nearest-neighbour of q in S is denoted by $NN_S(q)$ and is the point in the set $S - \{q\}$ with the smallest distance to q. This distance is called the nearest-neighbour distance and is denoted by $NND_S(q)$[9].

[9]Note that this is simply calculated using the usual distance function $d(\cdot, \cdot)$, such that $\mathrm{NND}_S(q) = d(p, q)$, where p is the nearest-neighbour of q.

Definition 4.4.2 *Nearest-neighbour Distance Set of a set of points.*

Let S be a set of points and e_i be the elements of S. Let NNDSet(S) be the set of all the nearest-neighbour distance values for all the $e_i \in S$.

Assume that all points within a cluster are uniformly distributed by a homogeneous Poisson point-process. The probability distribution of the nearest-neighbour distance for the cluster is determined as follows.

Let N points be uniformly distributed over the space R, where R has a volume $V(R)$. The probability that a point will be found in a subset $S \subset R$ is equal to $V(S)/V(R)$.

The probability that the $\mathrm{NND}_R(q)$ is greater than x is equal to the probability that none of the N points is located in the hyper-sphere around q with radius x, denoted $B(q, x)$.

$$\Pr(\mathrm{NND}_R(q) > x) = \big(1 - \frac{V(B(q,x))}{V(R)}\big)^N \tag{4.12}$$

$$\Pr(\mathrm{NND}_R(q) \leq x) = 1 - \big(1 - \frac{V(B(q,x))}{V(R)}\big)^N \tag{4.13}$$

So the probability distribution is parameterized by N and $V(R)$. Counting the number of objects N is easy, but finding the volume, $V(R)$ is not. The DBCLAS algorithm approximates $V(R)$ via a grid-based method.

The sample space is divided into p-dimensional cells, where the length of the edge of a cell is set to be the maximum element of NNDSet(S), where S is a set of points. A cell is said to be "occupied" if it contains one or more points of the set. The approximation of the volume of the set S is the sum of all occupied grid cells.

A χ^2-test is used to determine whether the observed nearest-neighbour distances fit the expected distance distribution. It is now possible to define a cluster.

Definition 4.4.3 *A cluster based on a nearest-neighbour distance distribution.*

Let $\boldsymbol{x}$ be a set of points. A cluster C is a non-empty subset of $\boldsymbol{x}$ with the following properties:

- *NNDSet(C) has the expected distribution within a required level of confidence;*
- *C is maximal. Each extension of C to include a neighbouring point violates the χ^2 condition; and*
- *C is connected. Each pair of points $(a, b) \in C$ has a path of occupied grid cells connecting a to b.*

The DBCLASD algorithm incrementally augments an initial cluster by a neighbouring point, provided the NNDSet($\cdot$) of the resulting cluster still fits the expected distance distribution. The algorithm proceeds as follows:

1. Generate a set of candidate points for inclusion in the cluster, C.

This is achieved via a regional query–using an area determined by m–on all points in C[10].

2. Select a point p that has not already been assigned to a cluster to start a new cluster, C.

3. Generate a set of candidates points for all points in C.

4. Test the set of candidates for inclusion via the χ^2-test, one by one.

   The inclusion test is as follows: augment the current cluster with the candidate point. Use the χ^2-test to check the hypothesis that the nearest neighbour distance set of the cluster still agrees with the expected distribution.

   If a candidate point satisfies this test, include it in the cluster. Update the set of candidates to include the new candidate points due to the point that has just been inserted. Repeat Step 3, testing the next candidate. Stop when the set of candidates is empty.

   If a candidate point is not included, label it "unsuccessful".

5. After all candidate points have been assessed, repeat Step 3, reassessing the set of "unsuccessful" points.

6. Stop after all points in the data have been considered.

The appealing properties of the DBCLASD algorithm are that:

- it can find clusters of arbitrary shape. This is by virtue of the grid system, that allows points in a grid cell to be considered as a cluster as long as they satisfy the statistical assumptions;
- it determines the number of clusters k;
- it has a low order time complexity on n. The set of candidate points for a cluster can be found efficiently using an R*-tree, Beckmann *et al.* (1990); and
- it requires no input from the user, the parameter m is found from the data.

The criticizing the algorithm, we comment that:

- it is incremental. The clustering is based on the points met so far, before considering the data as a whole. As such, the final clustering is dependent on the order of the data;
- the χ^2-test is only valid for clusters with a minimum size of 30. Therefore clusters must be created without testing initially. It will be impossible to find very small clusters in large databases. When DBCLASD creates a new cluster for a new point p, it simply adds the 29 nearest neighbouring points to the cluster.

[10] m is chosen to be such that no point with a $\mathrm{NND}_C(\cdot)$ greater than expected will be found. This is achieved by solving:

$$N \cdot \Pr(\mathrm{NND}_C(p) > m) < 1 \tag{4.14}$$

for m.

Chapter 5

Concluding remarks

5.1 Discussion of algorithms

When devising methods by which to cluster large datasets, it is likely that successful methods will have some or all of the following properties: a low order dependence on n; require a single scan of the data; guaranteed to fit into main memory; and able to estimate k without supervision.

5.1.1 Is it worth trying to extend traditional methods?

Fractionization, BIRCH and the mrkd-tree EM algorithm each have some, but not all of these attributes. All of them achieve their goal of being able to cluster large datasets, with varying levels of quality in their final clusters.

CLARANS is the only method of the four to tackle the large dataset via sampling. It works on a random sample of the information in the data, whereas all the other methods work on a summary of the complete information. Increased sampling, by considering more "neighbours" will cause the algorithm to tend to the k-medoids solution, with a similar computational complexity.

Fractionization achieves a low order dependence on n by using *meta-observations* to summarize small regions of the data. The meta-observations are created with a single scan of the data. The use of a standard clustering algorithm to cluster the meta-observations provides some flexibility for the user.

Drawbacks include the need for the user to supply k and the possibility of contamination in the meta-observations. Both of these problems were addressed by the refractionization method, that iterates the fractionization procedure. In order for refractionization to estimate the number of groups a probabilistic model must be specified; this requires a model-based clustering method to be used as the standard clustering method[1].

BIRCH takes a single scan of the data in building its CF-tree. The cluster features are then clustered using a standard method, thus creating a low order dependence on n.

[1] It should be noted that many model-based methods favour spherical clusters.

The CF-tree adapts to the local density of the data by setting a tolerance, ϵ on the radii of its cluster-features. Further, the tree building process guarantees that the tree will fit into main memory, as the tolerance will be raised successively until and the tree rebuilt until it does. Note that there is an implicit assumption that if a tree containing n_{CF} cluster-features can fit into main memory then n_{CF} will be sufficiently small to be able to be clustered by some standard clustering method (i.e. $n_{\mathrm{CF}} \leq M$; this is by no means certain.). The tree would have to be built until there are M or fewer cluster-features. This may have a detrimental effect on the quality of the clusters produced, as the formation of the cluster-features will no longer solely depend on the density of the data.

Other problems with BIRCH include the need to supply k and the possible "contamination" of cluster-features (cluster-features that are comprised of objects from different groups). The second of these problems is similar to that encountered in fractionization. It is likely that the amount of contamination will be less in the cluster-features as they were allowed to adapt to the local density during formation. Contamination, however small could be corrected by calling the BIRCH method iteratively, in a manner similar to refractionization. This may also provide an opportunity to estimate the value of k.

Analysis of the use of mrkd-trees is complicated, as its performance is interwoven with that of the EM algorithm[2]. Pelleg and Moore (2000) demonstrate how the mrkd-tree can be used to speed up the k-means algorithm. Regardless of the method, the mrkd-tree structure seems sensible; it is straight-forward to build, and its use of sufficient statistics to summarize data is efficient when storing information from the data in a compressed form. The tree need only be built once, requiring a single scan of the data and therefore creating a low order dependence on n.

A drawback is that the mrkd-tree is not guaranteed to fit into main memory. However it would be easy to adapt the building process to something similar to that used to build the CF-tree in the BIRCH algorithm. For instance, the tolerance on the widest dimension, ϵ_d could be increased, with leaf nodes treated as data and reinserted until the tree could fit in main memory.

All these methods (excluding CLARANS) accelerate existing methods by working with summaries of the actual data. The data is compressed into a set of statistics. This has the effect of cutting the order dependence on n to being that which is required to read the data into memory.

5.1.2 What is being offered by the new methods?

The methods presented in Chapter 4 go a long way towards satisfying our "wish list" in Chapter 1. In particular, the majority of methods: have very low order dependence on n, some methods are linear in n; detect clusters of arbitrary shape; estimate the number of groups k; and can filter out outliers.

Within Chapter 4 there are perhaps only three distinct methods.

1. The "density algorithms", (DBSCAN, DENCLUE, DBCLASD and STING);

[2]The EM algorithm finds a local maximum in the likelihood, but convergence is notoriously slow and hard to ascertain.

All these algorithms work via ideas of dense areas and clusters defined by the connectivity between these areas. Any differences arise though the definition of these areas. DBSCAN defines an area to be a neighbourhood determined by an object, whereas DENCLUE, DBCLASD and STING use a hyper-rectangle whose boundaries are independent of the objects it contains.

2. The particle filter; and

3. The oscillator network.

The density algorithms and the particle filter all achieved a low order dependence on n by taking a single scan of the data, though in quite different ways. The density algorithms continued the theme of Chapter 3. They use a single scan of the data as a preprocessing stage to form a summary of the data in memory; clustering is then carried out using the summary rather than the actual points. The particle filter reads an object at a time and incorporates the information by updating each one of its N particles; the actual observation can then be discarded. Both methods achieve scalability by breaking the link between the calculations and the data. The scalability of the SOON algorithm is very poor in comparison, $O(n^2)$ per iteration.

All methods provide the user with an estimation of the number of groups k. Density algorithms look for neighbouring dense regions and use maximal connectivity conditions when defining clusters. The particle filter creates a new cluster for an object when the underlying model suggests a poor fit within the current set. The SOON algorithm will stop once the oscillators have synchronized into a steady-state.

Whilst not having to provide k might seem like progress, these methods are far from being unsupervised. The final clusters–and therefore the estimate for the number of groups–will depend on the parameters supplied by the user when starting the algorithm. The particle filter will create new clusters to fit its underlying statistical model. The synchronization of oscillators in the SOON method will depend on the coupling functions used. The authors of DBCLASD and STING claim unparameterized and hence unsupervised methods, however, STING makes use of distance functions and DBCLASD requires clusters to contain at least 30 objects.

A great improvement is the ability of the density algorithms to find arbitrary shaped clusters, at least in a low number of dimensions. This ability stems from the use of a grid structure to quantize the data space into well defined ares and the notion of connectivity between neighbouring areas. The particle filter is dependent on the underlying statistical model, so tends to find the usual spherical clusters. For similar reasons, SOON tends to find the spherical clusters due reliance on the usual distance functions.

Outliers and noise are well handled by the density algorithms and partially handled by the particle filter. The density algorithms ignore cells in the grid that are sparsely populated. This happy synergy allows clusters to be created in the presence of noise, whilst speeding up the run-time. For the particle filter, particles with clusters containing outliers will have a small weight and therefore be more unlikely to be resampled. However, there is no way of getting rid of them entirely and they might cause bias by getting forced into an unwilling cluster at some point.

The particle filter's use of statistical distributions to define clusters makes it hard to compare with the density based algorithms. Compared with other model-based methods,

	Running time $O(\cdot)$	Estimate k	Arbitrary shapes	Handle noise	One scan of data	Will stop
k-means	n					
k-medoids	n^2			•		
Agglo.	n^3					•
Divisive	n^2					•
EM Alg.	n					
Fract.	n				•	•
Refract.	n	•				
BIRCH	n			•	•	•
mrkd-EM	$n \cdot \log n$				•	
DBSCAN	$n \cdot \log n$	•	•	•	•	•
DENCLUE	n	•	•	•	•	•
DBCLASD	$n \cdot \log n$	•	•	•	•	•
STING	n	•	•	•	•	•
P. Filter	n	•			•	•
SOON	n^2	•				

Table 5.1: Running time and properties of clustering algorithms.

such as the EM Algorithm, the particle filter has addressed the scalability problem well through its single scan of the data. Designed for use in online systems, perhaps many of its best attributes are not highlighted on large "batch" problems.

The SOON algorithm suffers from the long complexities that beset other methods based on neural networks. It is the only method of those discussed in Chapters 3 and 4 that works directly on the distance space. Applying the method to summaries of the data is likely to improve matters.

5.2 What lessons can be learnt?

The most successful clustering methods store summary statistics in trees. Building a tree only requires a single scan of the data. Insertion of new object into an existing tree is usually straight-forward. By constraining the amount of memory available in the tree building process, it is possible for the tree to adapt to fit into main memory.

If trees are to be used in clustering applications, we would want to be sure that they are being built in an optimal way. There are some differences in the way trees are constructed: the mrkd-tree of Moore (1999) is a binary tree with data partitioned at the widest dimension; the STING grid splits each dimension in two, causing 2^p partitions of the space at each node. Sufficient statistics are not always stored at the each node.

Definition of clusters via the connection of dense regions has provided ways to estimate the number of groups k. Coupled with the quantization of the data space, this has allowed methods to: find clusters of arbitrary shape; identify outliers as sparse regions; and offer further computational speed-ups through ignoring sparse/empty regions of the data space.

5.3 Ideas for future work

Large datasets

An optimally efficient tree-based data structure should be ascertained for clustering problems. The use of statistical distributions to define the density of quantized regions should be investigated. Multi-resolution clustering techniques (i.e. the ability to detect clusters within a cluster) need to be formalized. Methods should also be able to include different data types.

Online systems

The ability to cluster data arriving in a constant stream is also very important. The use of tree-based data structures within these systems should be explored as they are likely to be very effective. The importance of outlier detection, as well as cluster detection should stressed.

Appendix A

A.1 Two types of measurement space

We have defined the data presented to a clustering algorithm as a sample from the measurement space $\boldsymbol{X}$. In practice, the measurement space is of two distinct types. We shall refer to these as the distance-space and the vector-space. We shall assume throughout that measurements provided to an algorithm are from either a distance-space or a vector-space.

In a distance-space, each component of the data is the distance between two objects. Hence each point in distance-space gives information on the relationship *between* two objects rather than any single object itself.

A myriad of metrics are available for calculating the similarity between objects. For quantitative data, a popular metric is the Minkowski-distance:

$$d(x_r, x_s) = \Big\{ \sum_{k=1}^{p} \big\| x_{rk} - x_{sk} \big\|^q \Big\}^{1/q} : q \geq 1, x_i \in \mathbb{R}^p$$

Setting $q = 2$ gives Euclidean distance.

All that is required of the "distance" measure is that objects that are similar have relatively small distance between them, compared to objects that are deemed dissimilar. It is this flexibility that allows the clustering of binary and qualitative data via distance based methods.

In many cases the "distance" space has the following formal properties:

$$\begin{aligned} d(x_r, x_s) &\geq 0 \\ d(x_r, x_r) &= 0 \\ d(x_r, x_s) &= d(x_s, x_r) \end{aligned} \tag{A.1}$$

and, the *triangle inequality* holds:

$$d(x_r, x_s) \leq d(x_r, x_t) + d(x_t, x_s)$$

Given n objects, calculating the distances between all pairs of objects will take $O(n^2)$ time and storage. Thus distance-space methods are not immediately scalable to large datasets.

The vector-space consists of measurements recorded on single objects. No direct attempt is made to summarize the relationship between objects in the dataset prior to analysis[1]. Input to the clustering algorithm for an object r is simply the p-vector x_r, containing the measurements taken on that object.

Our clustering algorithms will always work on a matrix $\boldsymbol{x} \in \boldsymbol{X}$, pertaining to either a distance or vector-space. In the distance-space, $\boldsymbol{x}$ will be $(n \times n)$; row and columns correspond to the same entities. For the vector-space, $\boldsymbol{x}$ will be $(n \times p)$; rows and columns correspond to different entities; normally the rows represent the objects and columns the measurements made on them.

[1]However, if a statistical model in imposed on the geometric space (i.e. the Data), this defines the relationship between the different objects.

Bibliography

Ankerst, M., Breunig, M., Kriegel, H. and Sander, J. (1999) Optics: Ordering points to identify the clustering structure. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA* (Eds A. Delis, C. Faloutsos and S. Ghandeharizadeh), pp. 49–60, Philadelphia. ACM Press.

Beckmann, N., Kriegel, H., Schneider, R. and Seeger, B. (1990) The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990* (Eds H. Garcia-Molina and H. H. V. Jagadish), pp. 322–331. ACM Press.

Bezdek, D. (1981) *Pattern recognition with fuzzy objective function algorithms.* Plenum Press, New York, NY.

Bryant, P. and Williamson, J. (1978) Asymptotic behaviour of classification maximum likelihood estimates. *Biometrika*, **65**, 273–281.

Carpenter, J., Clifford, P. and Fearnhead, P. (1999) An improved particle filter for non-linear problems. *IEE proceedings - Radar, Sonar and Navigation*, **146**, 2–7.

Chen, R. and Liu, J. (2000) Mixture Kalman filters. *Journal of the Royal Statistical Society, Series B.*, **63**, 493–508.

Chopin, N. (2002) A sequential particle filter method for static models. *Biometrika*, **89**, 539–552.

Cutting, D., Karger, D., Pedersen, J. and Tukey, J. (1992) Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Copenhagen, Denmark* (Eds N. Belkin, P. Ingwersen and A. Pejtersen), pp. 318–329. ACM Press.

Doucet, A., de Freitas, N. and Gordon, N. (2001) *Sequential Monte Carlo in Practice.* Springer-Verlag, New York, NY.

Ester, M., Kriegel, H., Sander, J. and Xu, X. (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knowledge Discovery and Data Mining* (Eds E. Simoudis, J. Han and U. Fayyad), pp. 226–231. AAAI Press.

Fearnhead, P. (2002) Particle filters for mixture models with an unknown number of components. Technical report, University of Lancaster, Department of Mathematics and Statistics, Lancaster University, Lancaster, LA1 4YF.

Fearnhead, P. and Clifford, P. (2003) Online inference for well-log data. *To appear in the Journal of the Royal Statistical Society, Series B.*

Fraley, C. and Raftery, A. (1998) How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, (41), 578–588.

Frigui, H. and Rhouma, M. (2001) Self-organization of pulse-coupled oscillators with application to clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**(2), 180–195.

Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A. and French, J. (1999) Clustering large datasets in arbitrary metric spaces. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia*, pp. 502–511. IEEE Computer Society.

Gordon, A. (1999) *Classification*. Chapman and Hall, London.

Gordon, N., Salmond, D. and Smith, A. (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings on Radar and Signal Processing*, **140**, 107–133.

Hartigan, J. (1975) *Clustering algorithms*. John Wiley and Sons, New York, NY.

Hartigan, J. and Wong, M. (1979) Algorithm AS136: A k-means clustering algorithm. *Applied Statistics*, **28**, 100–108.

Hinneburg, A. and Keim, D. (1998) An efficient approach to clustering large multimedia databases with noise. In *Proceedings of the 4th ACM SIGKDD*, pp. 58–65. ACM Press.

Jensen, R. (1969) A dynamic programming algorithm for cluster analysis. *Operations research*, **17**, 1034–1057.

Kaufman, L. and Rousseeuw, P. (1990) *Finding groups in data*. Wiley, New York, NY.

Kruskal, T. (1957) On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. of the American Mathematical Society*, **7**, 48–50.

Krzanowski, W. (1988) *Principles of Multivariate Analysis*. Clarendon Press, Oxford.

Liu, J. and Chen, R. (1998) Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, **93**, 1032–1044.

MacNaughton-Smith, P., Williams, T., Dale, M. and Mockett, L. (1964) Dissimilarity analysis: A new technique of hierarchical sub-division. *Nature*, (202), 1034–1035.

Moore, A. (1999) Very fast mixture-model-based clustering using multiresolution kd-trees. *Advances in Neural Information Processing Systems*, **10**, 543–549.

Ng, R. and Han, J. (1994) Efficient and effective clustering methods for spatial data mining. In *In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile.* (Eds J. Bocca, M. Jarke and C. Zaniolo), pp. 144–155.

Pelleg, D. and Moore, A. (2000) Accelerating exact k-means with geometric reasoning. Technical Report CMU-CS-00-105, Carnegie Mellon University, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Ripley, B. (1996) *Pattern recognition and neural networks*. Cambridge Universtiy Press, Cambridge.

Sneath, P. and Sokal, R. (1963) *Principles of Numerical Taxonomy*. W.H. Freeman.

Tantrum, J., Murua, A. and Stuetzle, W. (2002) Model-based clustering of large datasets through fractionization and refractionization. In *Proceedings of ACM SIG KDD Conference 2002, Edmonton, Alberta, Canada*, pp. 183–190. ACM Press.

Wang, W., Yang, J. and Muntz, R. (1997) STING: A statistical information grid approach to spatial data mining. In *Twenty-Third International Conference on Very Large Data Bases*, pp. 186–195, Athens, Greece. Morgan Kaufmann.

Xu, X., Ester, M., Kriegel, H. and Sander, J. (1998) A distribution-based clustering algorithm for mining in large spatial databases. In *In 14th International Conference on Data Engineering, Orlando, FL*, pp. 324–311. IEEE Computer Society Press.

Zhang, T., Ramakrishnan, R. and Livny, M. (1996) BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 103–114. ACM Press.