Extending linear grouping analysis and robust estimators for very large data sets

by

Justin Harrington

BCom/BSc, University of Auckland, 1996

DipFinMath., Victoria University of Wellington, 1998

MSc, Victoria University of Wellington, 1999

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Statistics)

The University Of British Columbia

(Vancouver, Canada)

May, 2008

Abstract

In this dissertation, we extend the scope of problems where the clustering algorithm Linear Grouping Analysis (LGA) can be applied.

The first extension relates to the linearity requirement inherent within LGA, and proposes a new method for relaxing this requirement by non-linearly transforming the data into a Feature Space, such that in this space the data might then form linear clusters. In order to do this we utilise what is known in the literature as the Kernel Trick. However, one side effect of this transformation is that often the dimension of the transformed space is significantly larger, which can result in more dimensions than there are observations in a given cluster, and cause problems with orthogonal regression. To deal with this situation, we also introduce a new method for calculating the distance of an observation to a cluster when its covariance matrix is rank deficient.

The second extension concerns the combinatorial problem for optimizing a LGA objective function, and adapts an existing algorithm, called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), for use in providing fast, approximate solutions, particularly for the case when data does not fit in memory. We also provide solutions based on BIRCH for two other challenging optimization problems in the field of robust statistics, and demonstrate, via simulation study as well as application on actual data sets, that the BIRCH solution compares favourably to the existing state-of-the-art alternatives, and in many cases finds a more optimal solution.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr Matías Salibián-Barrera. For nearly four years he has been a most excellent adviser, collaborator and colleague (dare I say friend?), and many a time has kept me on track and focused on what is important. I am certainly appreciative of his efforts as well as his subtle, if ever so slightly dark, sense of humor. At this time I would also like to thank him for funding me for most of the last four years.

I would like to thank Professor's Zamar and Welch for agreeing to be members of my supervisory committee and for their insightful and encouraging comments.

My time in Vancouver has been a magical one, and this is mostly due to the wonderful friends I have made while being here. A special thanks goes to Mike (who is a most excellent coffee buddy), and the office staff for a being a source of sanity in a world of idiosyncrasy. To all my friends and colleagues I wish all the best in their future endeavours, and bid them *hasta luego* rather than *adiós*.

I dedicate this thesis to my parents and my wonderful wife. Without the many sacrifices of my parents I would never have had this opportunity, and any achievement of mine is a direct result of the excellent foundation they built for me. And finally, to my wonderful Diana – for her unwavering support and understanding I am truly indebted. The fact that she agreed to marry a investment banker cum professional student still amazes me, and I am very much looking forward to the many, many years we will have together.

Per Angusta Ad Augusta

Through trial to triumph

Co-Authorship Statement

Two articles, which formed the basis of Chapter 3 and Appendix B, were co-authored with Dr Matías Salibián-Barrera. These articles were:

- Harrington, J. and Salibián-Barrera, M. (2007) Finding Approximate Solutions to Combinatorial Problems with Very Large Data Sets using BIRCH, "Special Issue on Statistical Algorithms and Software" of *Computational Statistics and Data Analysis*; and
- Harrington, J. and Salibián-Barrera, M. (2008) birch: Working with very large data sets, *Journal of Statistical Software*.

For both articles, the literature review, theory, development of algorithms, coding in R, and initial drafts of the documents were completed by Justin Harrington. Further editing of the manuscripts for publication and (excellent) supervision was provided by Dr Salibián-Barrera.

Chapter 1

Introduction

Cluster analysis is the study of how to partition data into homogeneous subsets so that the partitioned data share some common characteristic. In two and three dimensions, the human eye is excellent at distinguishing between clusters of data when they are clearly separated. However, when there are more than three dimensions and/or the data is not clearly separated, an algorithm is required which in turn needs a metric of similarity that quantitatively measures the characteristic of interest. An oft cited example of a clustering algorithm is k-means, which partitions data into compact spheres, a characteristic that can be measured with the within-cluster sum of squares. Generically clustering algorithms are combinatorial problems, insofar as they are searching for an optimal partition of the data in order to minimize an objective function. The set of all possible partitions is referred to as the solution space of the problem, and not surprisingly the size of this space increases dramatically with the number of observations, dimensions and clusters.[1]

Linear Grouping Analysis (LGA) is an algorithm for clustering data around hyperplanes that was first introduced in Van Aelst et al. (2006). This algorithm is most appropriate when:

- the variables are related/correlated, which results in clusters with an approximately linear structure; and
- it is not natural to assume that one variable is a "response", and the remainder the "explanatories".

LGA measures the compactness within each cluster via the sum of squared orthogonal distances to hyperplanes formed from the data. However, if the data is not at least approximately linear within a cluster, then clearly this measure is no longer appropriate. In order to measure orthogonal distances, the algorithm relies heavily on the covariance matrix of each cluster to determine its structure, and requires among other things a certain minimum number of observations (relative to the dimension) in order to specify the shape fully.

In this thesis we propose three new extensions to this algorithm. They are:

I - Relax the linearity assumption via transformation of the data set.

II - Allow LGA to perform in the situation where at least one of the covariance matrices does not adequately specify the structure of its cluster.

III - Make LGA scalable with respect to the number of observations.

When the data is not linearly structured, we transform the data into an alternative space, commonly referred to the "feature space", and this is a technique that is frequently used in classification

[1] For example, it was shown in Welch (1982) that k-means for three or more clusters is a NP-hard problem.

problems in the machine learning literature. Often these transformations dramatically increase the dimension of the data set, resulting in a covariance matrix that has insufficient observations to completely specify the new structure of a transformed cluster. In order to deal with this, we introduce a new distance metric, called the "partially orthogonal distance". Together, these are the topics in Chapter 2.

Finally, we introduce an adaption of an algorithm called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) as a means of making LGA scalable by substantially reducing the size of the solution space. In order to do this, we create subsets of the data that are compact and locally similar, and maintain a list of attributes for each subset such that clustering can be done with the subsets instead of the full data set. As it turns out, this approach can be used for a number of different algorithms that also aim to solve combinatorial problems, and we provide details of these in Chapter 3.

In the remainder of this chapter we provide an introduction to the LGA algorithm, and then provide a general introduction to some feature space methods. We then conclude with a review of BIRCH, and the algorithms on which it is applied.

1.1 An Overview of Linear Grouping Analysis

Linear Grouping Analysis (LGA) involves grouping data around hyperplanes, and utilizes orthogonal regression to find hyperplanes that minimize the sum of the squared orthogonal distances of each observation to its representative hyperplane.

First consider the k-means algorithm, which groups the data into k spheres by minimizing the sum-of-squares within clusters. The objective function for this criterion is

$$\sum_{i=1}^{k}\sum_{j=1}^{n} z_{ij}(\mathbf{x}_j - \mathbf{m}_i)^\top(\mathbf{x}_j - \mathbf{m}_i),$$

where $\mathbf{x}_j \in \mathbb{R}^d,\ j = 1, \ldots, n$ is our data, $\mathbf{m}_i$ $(i = 1, \ldots, k)$ are the centres of each sphere, and

$$z_{ij} = \begin{cases} 1 & \text{if observation } j \text{ is part of cluster } i \\ 0 & \text{otherwise.} \end{cases}$$

This is the arrangement of the data with the most compact clusters, where the sum-of-squares for each cluster measures the sum of squared distances of each observation in the cluster to its centre. In order to minimize the objective function, k-means has a two step procedure:

1. Given a clustering of data, for each cluster calculate the sample mean of all the observations currently belonging to the cluster.
2. Calculate the Euclidean distance of each observation to each cluster's centre, and relabel the observation as belonging to the cluster which is closest.

This process is iterated until convergence, and is initialized with a number of different starting configurations. The result with the smallest within cluster sum-of-squares is selected as the best partition.

Contrast this with LGA, which endeavours to find the partition with associated hyperplanes which minimizes the orthogonal distances of each observation to these hyperplanes. In this context, we parametrize a hyperplane by the vector it is orthogonal to ($\mathbf{a}_i$) , and a constant (b_i), and denote this hyperplane $\mathcal{H}(\mathbf{w}_i, b_i) \equiv \mathcal{H}_i$. The orthogonal distance of an observation $\mathbf{x}_j$ to the hyperplane $\mathcal{H}_i$ is given by

$$d_O^2(\mathbf{x}_j, \mathcal{H}_i) = (\mathbf{a}_i^\top \mathbf{x}_j - b)^2,$$

and so the objective function for LGA is given by

$$\sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij}\, d_O^2(\mathbf{x}_j, \mathcal{H}_i).$$

This objective function is referred to as the Residual Orthogonal Sum of Squares (ROSS). In order to minimize this objective, an approach like that of k-means is used:

1. Given an clustering of data, find the hyperplane that best fits the data in each cluster.
2. Calculate the orthogonal distance of each observation to each hyperplane, and assign the observation to the hyperplane that is closest.

Once again, this is iterated to convergence, and is started from a number of different initial configurations. To find the parameters for a hyperplane that best fits a subset of data, we use Orthogonal Regression, as given in the following well known result.

Result 1.1.1 (Orthogonal Regression) Let $\mathbf{y}_1, \ldots \mathbf{y}_j \in \mathbb{R}^d$ be a set of data with sample covariance matrix $\mathbf{S}$. Let the eigenvectors of $\mathbf{S}$ be $\mathbf{e}_1, \ldots, \mathbf{e}_d$ with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots > \lambda_d \geq 0$. Then the solution to

$$\arg\min_{\mathbf{a},b} \sum_{i=1}^{j} (\mathbf{a}^\top \mathbf{y}_i - b)^2$$

subject to the constraint $||\mathbf{a}|| = 1$ is given by

$$\mathbf{a} = \mathbf{e}_d \text{ and } b = \mathbf{e}_d^\top \bar{\mathbf{y}}.$$

Proof of 1.1.1 Adding and subtracting $\mathbf{a}^\top \bar{\mathbf{y}}$ yields

$$\arg\min_{||\mathbf{a}||=1,b} \sum_{i=1}^{n} \left(\mathbf{a}^\top \mathbf{y}_i - b\right)^2 = \arg\min_{||\mathbf{a}||=1,b} \mathbf{a}^\top \mathbf{S}\mathbf{a} + n\left(\mathbf{a}^\top \bar{\mathbf{y}} - b\right)^2.$$

It is clear that the second term is minimized by setting $b = \mathbf{a}^\top \bar{\mathbf{y}}$. To find the minimum of the first term,

$$\arg\min_{||\mathbf{a}||=1} \mathbf{a}^\top \mathbf{S}\mathbf{a} = \arg\min_{||\mathbf{a}||=1} \mathbf{a}^\top \mathbf{E}\mathbf{\Lambda}\mathbf{E}^\top \mathbf{a}$$

where $\mathbf{E}$ is a column matrix containing the eigenvectors of $\mathbf{S}$, and $\mathbf{\Lambda}$ a diagonal matrix with the eigenvalues on the diagonal. Let $\mathbf{w} = \mathbf{E}^\top \mathbf{a}$ and note that $||\mathbf{w}|| = ||\mathbf{a}||$. Then

$$\begin{aligned}\min_{||\mathbf{a}||=1} \mathbf{a}^\top \mathbf{E}\mathbf{\Lambda}\mathbf{E}^\top \mathbf{a} &= \min_{||\mathbf{w}||=1} \mathbf{w}^\top \mathbf{\Lambda}\mathbf{w} \\ &= \min_{||\mathbf{w}||=1} \sum w_i^2 \lambda_i \\ &\geq \min_{||\mathbf{w}||=1} \lambda_d \sum w_i^2 = \lambda_d.\end{aligned}$$

Finally, by substitution, this lower bound is achieved when $\mathbf{a} = \mathbf{e}_d$. □

For those readers familiar with Principal Components Analysis, it is clear that orthogonal regression is looking for the subspace of dimension $(d-1)$ that is closest to the data in the L_2-sense. This corresponds with the basis formed by the first $(d-1)$ principal components, which are the same as the first $(d-1)$ eigenvectors, and are orthogonal to the last principal component - ergo, the result above.

This equivalence between PCA and orthogonal regression gives us a useful relationship between the nature of the covariance matrix and its specification of the structure of the data set. Consider a data set in d-dimensions, but with an orthonormal basis that has less than d vectors describing any meaningful variation in the data – this equates to the number of positive, non-zero eigenvalues being less than d. Clearly in this case, the orthonormal basis described by PCA will not be able to span the full dimension of the data set. This is discussed further in Chapter 2.

LGA is then given by the following algorithm.

Algorithm 1.1.2 (LGA)
LGA is made up of the following five steps:

1. *Scaling of the variables:* each column of the data set is divided by its standard deviation so that it has unit variance.

2. *Generation of starting value:* a starting configuration is generated consisting of a partition of the data set into k mutually exclusive sets of d observations (i.e. an observation can belong at most in one set). For each of these sets, calculate the hyperplane that fits it (with zero error).

3. *Initialization of groups:* for each starting hyperplane, calculate the squared orthogonal distance of all observations to that hyperplane. Then assign each point to the hyperplane which it is closest, and recalculate the hyperplanes based on the new cluster membership.

4. *Iterative refinement:* Recalculate step 3 for each resulting hyperplane until either a) the solution converges (i.e. the membership sets remain the same for two consecutive iterations) or b) the solution fails to converge after an arbitrary number of iterations (e.g. 10).

5. *Resampling:* Repeat steps 2-4 a number of times and select the solution with the smallest total orthogonal sum of squares.

The code to do this is implemented in the R package lga (Harrington, 2008).

For completeness, we close this section with a brief mention of two alternatives to LGA that are both applicable in forming clusters that are linear in shape. Regression based clustering (e.g. Gawrysiak et al. (2001)) is a close relation to LGA insofar as the methodology is broadly built around the same idea:

1. Randomly select starting configurations;
2. Fit a standard linear regression model to the data; and
3. Allocate the observations to the line of smallest residual.

This process is iterated, until some criteria is met. The main limitation of this approach, is that for many data mining applications there is no response/explanatory type of variables, and arbitrary selection of these will lead to different solutions.

Model based clustering (e.g. Banfield and Raftery (1993), Fraley and Raftery (2002)) is a flexible maximum likelihood approach to cluster analysis, and utilizes mixture models to specify the cluster characteristics. Most commonly the underlying distribution is a multivariate normal, but other options are available; for example points clustered uniformly along a straight line (Banfield and Raftery (1992)).

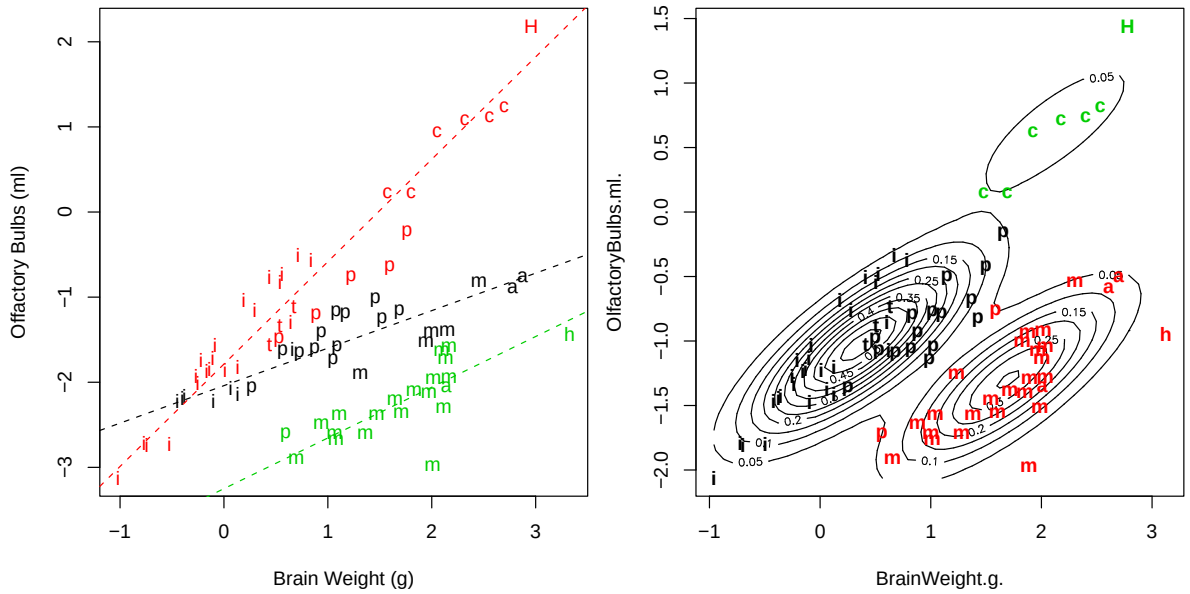
In the case of the multivariate normal, the solution is found using the EM algorithm, and is implemented in R in the package mclust. An example of mclust versus lga is given in Figure 1.1. In this case, the LGA solution is more consistent with the underlying biological reasoning. However, the mclust does form clusters that are elliptical in nature, and perhaps would yield a better solution with the "tweaking" of the covariance structure.

1.1.1 On the Nature of Combinatorial Problems

Broadly speaking we are concerned with combinatorial problems that arise from trying to find a partition of a data set such that a specific objective function is minimized. The nature of the partition may be a "best" subset of the data (in the case of some robust estimators), or completely partitioning the data set into mutually exclusive sets (which is the case in cluster analysis). Either way, these problems are considered to be non-convex since there are many different configurations with local minima (though the meaning of "local" differs depending on context), and as such a simple hill-climbing algorithm such as a Newton-Raphson method is not guaranteed to find the global minimum.

One common method for solving problems such as these involves selecting a random partition/subset of the data set, and then using the data to suggest what an improved configuration might be through an iterative approach of evaluating the objective function, and then sorting the data into partitions or subsets, and this is the approach given in the previous section. Clearly, for a given starting configuration, there is no guarantee that the algorithm will converge to the correct solution, but by selecting a large number of different starts, one can become more confident of finding the global minimum.

The source of this confidence can be shown in the following simple illustration. In the context of LGA, we propose two different starting configurations for a data set with two clusters, given graphically in Figure 1.2. In the figure on the left, clearly this is a "bad" starting configuration,



(a) LGA with $k = 3$.

(b) Mclust with $k = 3$.

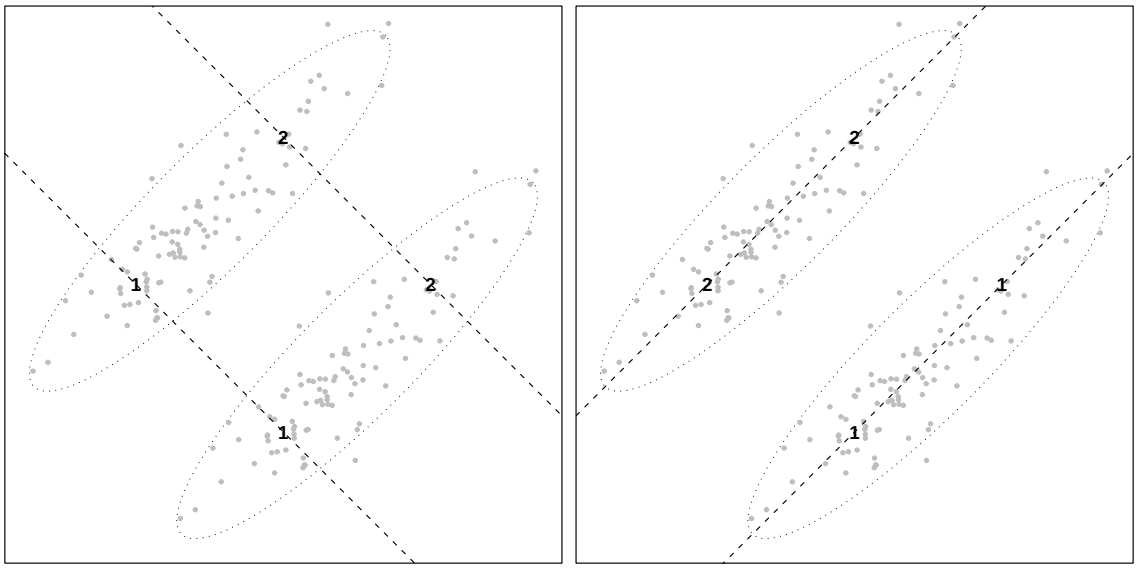
Figure 1.1: An example of LGA versus Mclust using the olfactory bulbs vs. brain weight data set. The data set is the same as the one used in Section 4.1 of Van Aelst et al. (2006). The number of clusters was determined by Mclust using BIC. For reference, the GAP Statistic (Tibshirani et al., 2001) suggests two clusters for LGA.
Key: m = monkeys, i = insectivores, p = primitive primates, c = carnivores, a = apes, h = humans, H = horses.

since the initial points are from different clusters, and the starting hyperplanes do not resemble the "true" hyperplanes. While it is not certain that this starting configuration would not converge to the correct solution, it is off to a bad start.

Now consider Figure 1.2(b), where the initial configuration is a "good" starting point, since the approximately correct structure of the clusters has already been established. In this case, only a few iterations would be required before convergence. Mathematically, we can describe a "good" starting point with the following: let $C_1, \ldots, C_k$ be the true partition of the data set and $S_1, \ldots, S_k$ be the initial starting configuration. Then a partition can be considered a good starting position iff $S_i \subset C_i \; \forall i \in \{1, \ldots, k\}$.

Therefore, the more starting configurations used to initialize the algorithm, the higher the chance of finding a "good" starting point. As it turns out, with LGA it is possible to enumerate the number of starts required so that the probability of obtaining at least one sample with d points from each group is 95% (i.e. there's a 95% chance that at least one starting solution has all the observations correctly clustered). The probability of this occurring is

$$p = \frac{\binom{n_1}{d}}{\binom{n}{d}} \times \frac{\binom{n_2}{d}}{\binom{n-d}{d}} \times \cdots \times \frac{\binom{n_k}{d}}{\binom{n-d\times(k-1)}{d}} \times k! = \frac{\prod_{i=1}^{k}\binom{n_i}{d}}{\prod_{i=0}^{k-1}\binom{n-d\times i}{d}} \times k!$$



(a) Bad starting position (b) Good starting position

Figure 1.2: A demonstration of the importance of starting initialization, with two clusters illustrated by some points and a confidence contour. In both plots we have the same four points; however, on the left the initial partition has points from different clusters together, producing an initial hyperplane that does not reflect the true shape of the clusters. On the right the initial positions are subsets of the true clusters, and the hyperplanes reflect the correct structure.

where n_i is the number of elements in cluster i, and n the total number of observations.[2] Therefore we require m samples such that $1-(1-p)^m = 0.95$. Assuming that each of the clusters are equal sized (i.e. $n_i = \lceil n/k \rceil$ where $\lceil a \rceil$ is the "ceiling" integer part of a), this equates to

$$m = \left\lceil \frac{\log(0.05)}{\log(1-p)} \right\rceil \tag{1.1}$$

where p simplifies to

$$p = \frac{\binom{n_1}{d}^k}{\prod_{i=0}^{k-1} \binom{kn_1 - di}{d}} \times k!.$$

A plot of equation (1.1) for different values of k and d is given in Figure 1.3.

1.2 Extending Linear Grouping Analysis

In this section we provide some background on the material in Chapter 2, which is concerned with performing LGA with non-linear patterns. At first glance, this might seem rather contradictory,

[2]Note that this differs from the result provided in Van Aelst et al. (2006).

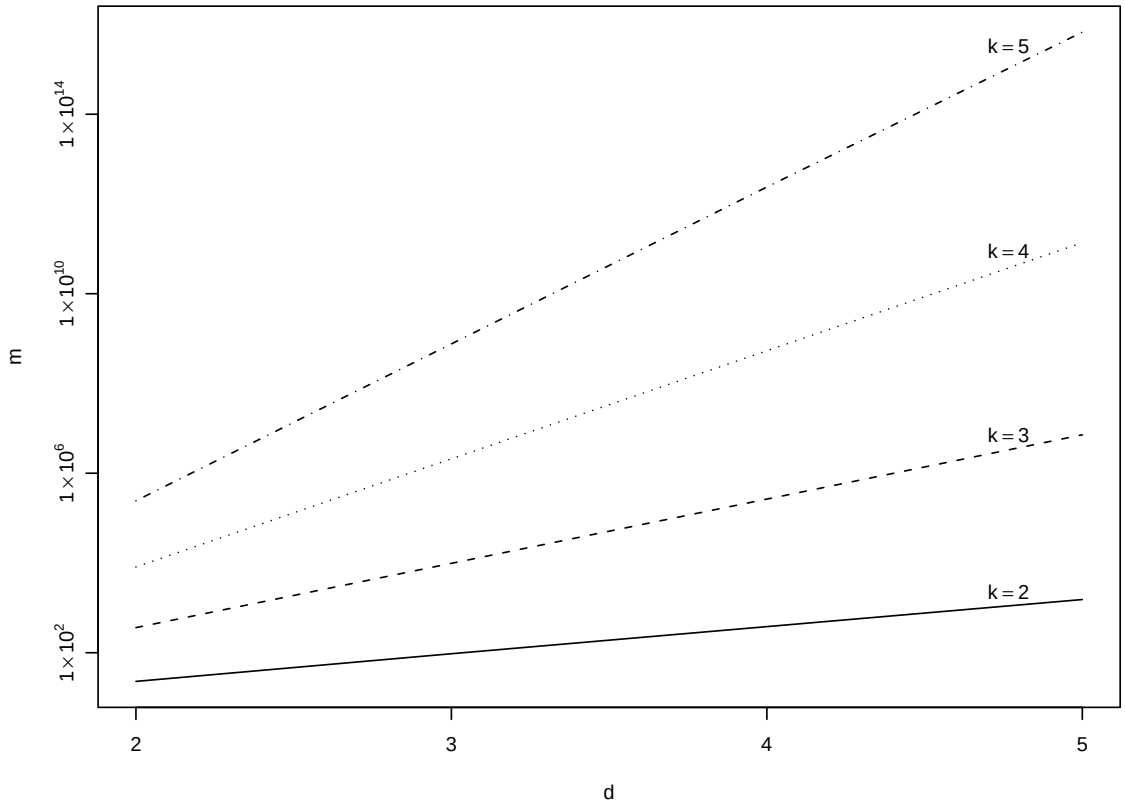


Figure 1.3: The number of samples required to achieve a 95% probability of a "good" initial partition when $n_1 = \ldots = n_k = 1 \times 10^6$.

since by definition the underlying hyperplanes must be "flat", and therefore not appropriate for non-linear patterns. While this is certainly true in Euclidean space, in Chapter 2 we transform the data so that the clusters are linear, and then perform LGA in this transformed space (called the Feature Space).

However, one side effect of transforming the data is that frequently the dimension of the transformed space is dramatically larger than the original dimension of the data, to the point where there are more dimensions than observations, which means that the covariance matrix no longer fully specifies the structure of the data in all dimensions. Rather than reduce the dimension directly via standard algorithms such as Principal Components Analysis, instead we propose a new metric for when there are more dimensions than observations.

What follows is a brief background on some of the other approaches used in dimension reduction for clustering, and a brief discussion of why these approaches would not work in the context of LGA. We then introduce a method for transforming the data, accompanied by an example where it has also been used.

1.2.1 Dimension Reduction and Clustering

Dimension reduction via Principal Components Analysis (PCA) is used in many different situations. One example is when d is large, and n substantial, since the reduction in the overall size of a data set is dramatic if the number of dimensions is reduced. Another common example is where there are many dimensions, which makes it difficult to visualize the data, and a lower-dimensional representation is desired.

PCA has been discussed previously in clustering literature, though the sentiment regarding its effectiveness is mixed. Articles that advocate this approach include Jollife et al. (1982), where principal components analysis is applied on a data set to reduce the number of dimensions from 20 to 10, and an approach similar to k-means applied to the PC scores. In Jolliffe (2002), a section is devoted to describing heuristically when it may be advantageous to apply PCA prior to clustering. One useful case is where some columns are similar (in the Euclidean distance sense), and thus are effectively over-represented with respect to their importance in clustering. Finally, in Ben-Hur and Guyon (2003), applying PCA on DNA microarray data was used with some success.

However, there are also articles that demonstrate where PCA does not work. For example, in Chang (1983) it was shown that in the case of a mixture of two normals, the first few PCs may contain less cluster structure information than other PCs. And in Yeung (2001), applying PCA prior to clustering gene expression data was found empirically not to work well.

In the context of LGA, it can easily be shown that a dimension reduction approach using PCA is unlikely to work. While dimension reduction would work on clusters that are *a priori* known as the true structure of each cluster is then accurate, applying PCA on a whole data set prior to clustering would not take into account the (probably) differing structures, and could even project data from different clusters onto the same space. Clearly, therefore, any dimension reduction should take place after a clustering has been proposed.

1.2.2 Transforming Data into a Feature Space

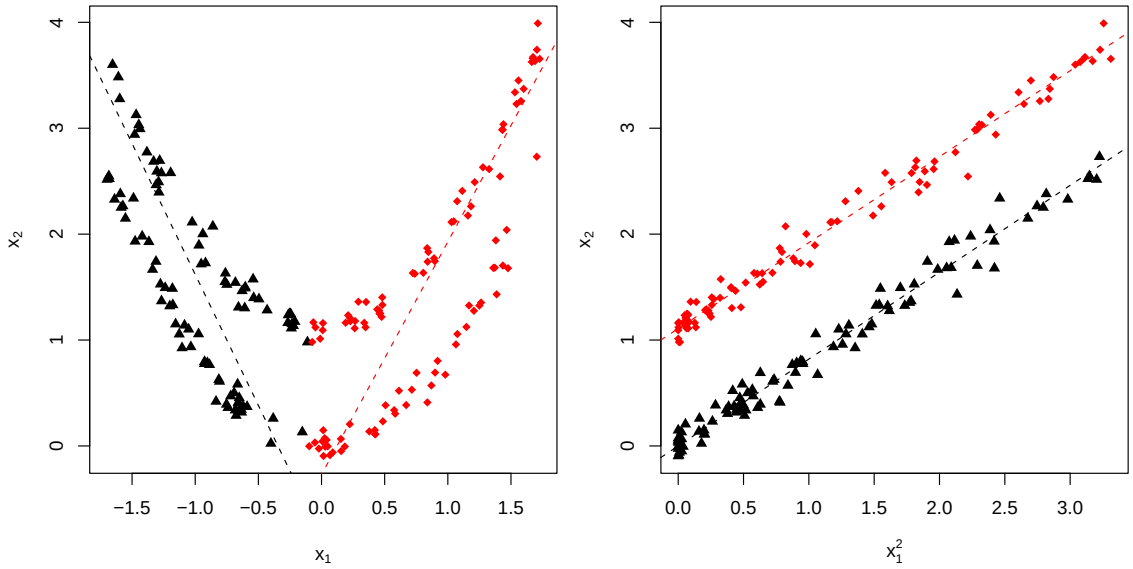
If the clusters of data do not form hyperplanes, then the objective function associated with LGA is not an appropriate measure of similarity within each cluster. Consider the following data set made up of two groups of data, given in Figure 1.4(a). Clearly the data is nonlinear in nature, and the LGA solution does not correctly identify the true groupings. By applying the transformation $\Phi(\mathbf{x}) = (x_1^2,\ x_2)$, however, the data is then linear in this new space, and LGA then identifies the correct grouping.

In this case we explicitly specified a transformation, and then applied LGA to the transformed data. An alternative approach is to calculate the pairwise cross product between all transformed observations, and then through mathematical manipulation work with this cross product instead. This methodology is referred to as a "Kernel Method", and is the approach taken in Chapter 2.

Let $\mathbf{x} \in \mathbb{R}^d$, and consider a nonlinear transformation $\Phi(\mathbf{x})$ such that $\Phi : \mathbb{R}^d \to \mathcal{F}$, what we refer to as the feature space. Using the dot product, called the kernel function and denoted

$$k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^\top \Phi(\mathbf{x}')$$

instead of the transformation directly is commonly called the "kernel trick". The advantage of this approach is that the kernel function can then be either explicitly specified as we did before, or



(a) Original Data (b) Transformed Data

Figure 1.4: A demonstration of transformation to linearity. The symbols and colors correspond to the partition found by LGA using the original data (left), and the transformed data (right). The hyperplane for each partition is also provided.

implicitly defined by stating the kernel function directly. In the latter case, while it may not be possible to specify an equivalent function Φ, a result called "Mercer's Condition" gives the necessary and sufficient conditions under which Φ does exist, and allows the kernel trick to be used.

Some of the more common kernel functions are presented in Table 1.1. It is common to form a matrix of the kernel functions for all observations, with $\mathbf{K} \in \mathbb{R}^{n \times n}$ and $(\mathbf{K})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ for $(i, j = 1, \ldots, n)$. The matrix $\mathbf{K}$ is commonly referred to as the Kernel Matrix.

In order to give a simple demonstration of how the kernel trick is applied, we give a simple example from Girolami (2002) (Feature Space k-means). Recall the k-means objective function, given by

$$\sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} (\mathbf{x}_j - \mathbf{m}_i)^\top (\mathbf{x}_j - \mathbf{m}_i),$$

where $\mathbf{m}_i$ $(i = 1, \ldots, k)$ are the centres of each sphere and z_{ij} denotes the membership of observation j in cluster i. Ignoring the summation temporarily and expanding the brackets yields

$$\mathbf{x}_j^\top \mathbf{x}_j - 2\mathbf{x}_j^\top \mathbf{m}_i + \mathbf{m}_i^\top \mathbf{m}_i,$$

where $n_j = \sum_{j=1}^{n} z_{ij}$ and $\mathbf{m}_i = \sum_{j=1}^{n} z_{ij}\mathbf{x}_i / n_j$ is the sample mean of all the observations in the i-th cluster. Clearly

$$\mathbf{x}_j^\top \mathbf{m}_i = \frac{1}{n_i} \sum_{l} z_{il} \mathbf{x}_l^\top \mathbf{x}_j \qquad\qquad \mathbf{m}_i^\top \mathbf{m}_i = \frac{1}{n_i^2} \sum_{l} \sum_{m} z_{il} z_{im} \mathbf{x}_l^\top \mathbf{x}_m.$$

Name of kernel	$k(\mathbf{x}, \mathbf{z})$
Rational Quadratic*	$1 - \frac{\|\|\mathbf{x} - \mathbf{z}\|\|^2}{\|\|\mathbf{x} - \mathbf{z}\|\|^2 + \theta}$
Polynomial	$\left(\gamma \mathbf{x}^\top \mathbf{z} + \theta\right)^d$
Exponential*	$\exp\left(-\frac{\|\|\mathbf{x} - \mathbf{z}\|\|}{\theta}\right)$
Radial Basis Function (RBF)*	$\exp\left(-\frac{\|\|\mathbf{x} - \mathbf{z}\|\|^2}{\theta}\right)$
Wave*	$\frac{\theta}{\|\|\mathbf{x} - \mathbf{z}\|\|} \sin\left(\frac{\|\|\mathbf{x} - \mathbf{z}\|\|}{\theta}\right)$

Table 1.1: Some commonly used kernels. Source: Genton (2001). The kernel functions that are starred are of the type "isotropic", which means it is only a function of the distance $||\mathbf{x} - \mathbf{z}||$.

Finally, applying the transformation Φ by substituting $\mathbf{x}_i$ with $\Phi(\mathbf{x}_i)$, and denoting the center of the i-th cluster in the feature space by $\mathbf{m}_i^\Phi$, yields the following representation in the feature space

$$\sum_{i=1}^{k}\sum_{j=1}^{n} z_{ij}(\Phi(\mathbf{x}_j) - \mathbf{m}_i^\Phi)^\top(\Phi(\mathbf{x}_j) - \mathbf{m}_i^\Phi) =$$

$$\sum_{i=1}^{k}\sum_{j=1}^{n} z_{ij}\left\{\Phi(\mathbf{x}_j)^\top\Phi(\mathbf{x}_j) - \frac{2}{n_i}\sum_l z_{il}\Phi(\mathbf{x}_l)^\top\Phi(\mathbf{x}_j) + \frac{1}{n_i^2}\sum_l\sum_m z_{il}z_{im}\Phi(\mathbf{x}_l)^\top\Phi(\mathbf{x}_m)\right\},$$

and applying the kernel trick

$$= \sum_{i=1}^{k}\sum_{j=1}^{n} z_{ij}\left\{k(\mathbf{x}_j, \mathbf{x}_j) - \frac{2}{n_i}\sum_l z_{il}k(\mathbf{x}_l, \mathbf{x}_j) + \frac{1}{n_i^2}\sum_l\sum_m z_{il}z_{im}k(\mathbf{x}_l, \mathbf{x}_m)\right\}. \tag{1.2}$$

In other words, instead of applying the transformation to $\mathbf{x}$ and then performing k-means, we can perform k-means in the feature space by optimizing equation (1.2) directly. An approach that is similar to this, but relevant for LGA instead, is taken in Chapter 2.

Kernel methods have been used elsewhere in a clustering context in addition to Girolami (2002). Spectral clustering (Ng et al., 2001) has a novel approach of examining the kernel matrix (in this case the RBF kernel is used) to search for a block diagonal structure that typifies cluster designs. For example, consider two clusters where all the points within a cluster are very compact, and

the two clusters are very far apart. Then any isotropic kernel matrix would appear to be a block diagonal with a matrix of all ones on the block diagonal, and zero elsewhere.

Support Vector clustering (Ben-Hur et al., 2001) takes a dramatically different approach, and is more consistent with the approach used in Support Vector Machines than the more standard clustering algorithms that are a combinatorial problem on an objective function. In this article they transform the data into the feature space, and then form the smallest sphere that encloses the complete data set. This sphere is then mapped back into the Euclidean space, where it forms a set of contour boundaries which enclose clusters of data. Points within each of these contour boundaries are then considered to be clusters. This approach does not require the specification of the number of clusters, and the examples provided in the paper suggest that this approach is highly sensitive to tuning parameters.

1.3 Finding Approximate Solutions to Combinatorial Problems with Very Large Data Sets using BIRCH

In Chapter 3 we review an algorithm that allows for the optimization of a class of combinatorial problems to be done in an approximate way, and then extend it to a number of combinatorial problems relevant to statistics. The algorithm is called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies, Zhang et al., 1997), and has the advantage that it scales very well with respect to the number of observations. In this section, we give some motivation for BIRCH, including alternative implementations of this algorithm, and then review other, similar, algorithms. We conclude with a review of the robust methods and clustering algorithms that are extended in Chapter 3.

Interested readers should refer to Appendix B for a description of the implementation of BIRCH in R, which has been submitted to the Journal of Statistical Software.

1.3.1 Motivation

One approach for solving non-convex optimization problems, such as LGA, is to explore the solution space in order to find the configuration that yields the global minimum of a given objective function. We do this by starting at a number of different subsets of the data set which allows the structure of the data to suggest where to go. This type of local search is more effective than just performing random searches over the solution space, and is the approach used in Section 1.1 for optimising the LGA objective function.

However, as the number of dimensions and/or the number of observations increases, then clearly the solution space is increasing in size, and so too the number of subsets required in order to adequately cover this space. The algorithm BIRCH is an effective method of reducing the size of the solution space, making optimization much easier.

BIRCH works by pre-processing a data set and grouping observations together into locally compact, similar subclusters. Associated with each subcluster is a vector of summary statistics, called clustering features (CFs). In practise, these clustering features are formed using an algorithm that, heuristically, examines each observation of a data set in isolation, finds the closest subcluster, and adds it to the subcluster, updating the CFs in the process. If the observation is not close to any

subcluster, then it forms it own subcluster.

It can be shown that for a certain class of algorithm, an approximately equivalent optimization over an objective function is possible using the CFs instead of the underlying data. This is very advantageous, since the solution space with respect to subclusters is several orders of magnitude smaller than that of the full data set. In this way, we can optimize more efficiently.

The nature of the approximation in the process lies in the fact that we are using subclusters, rather than the underlying observations, when selecting subsets of the data. For example, consider Orthogonal Regression, given in Result 1.1.1, and assume that we have the following tuple as our CF:

$$CF_J = \left(\sum_{i \in \mathcal{L}_J} 1, \sum_{i \in \mathcal{L}_J} \mathbf{X}_i, \sum_{i \in \mathcal{L}_J} \mathbf{X}_i \mathbf{X}_i^\top \right) = (n_J, LS_J, SS_J);$$

i.e. the number of observations, the sum of those observations, and the sum of squares of those observations.

Recall, that the orthogonal regression solution is given by the eigenvectors of the covariance matrix and the mean of the underlying data. It can be shown that these statistics can be calculated from the CFs directly – i.e. the calculation of the orthogonal regression using the CFs is identical to the calculation using the underlying observations making up those subclusters. In this sense, the CFs can be considered sufficient statistics for calculating orthogonal regression.

Now consider the LGA algorithm, which utilizes orthogonal regression for calculating distances. In particular, LGA is trying to find the clusters of data with the minimum residual orthogonal sum of squares to their closest hyperplane. Now consider the equivalent with BIRCH subclusters, where LGA attempts to find the partition of subclusters that minimize the ROSS. As demonstrated above, we can calculate the ROSS exactly, but if the "true" partition of the data requires splitting observations within one subcluster, then the global minimum (based on the underlying observations) cannot be achieved.

Clearly, as the number of subclusters increases - equivalently, the number of observations in each subcluster decreases - then the need to split subclusters in order to find the global minimum is less of an issue as the subclusters are more "granular". However, a direct side effect of the increased number of subclusters is the increased size of the solution space, which means there is a trade-off between accuracy and optimization efficiency. This is discussed at length in Chapter 3.

Finally, one additional benefit of this approach involves the practical consideration of the limitations of the statistical software, R (R Development Core Team, 2007). R requires that a data set be held in memory for an analysis, which may not be possible if the data set is very large. As mentioned before, BIRCH only pre-processes observations one at a time, and produces an object much smaller than the original data set. Therefore, as long as the BIRCH object can fit into memory (which is always the case as long as the parameters for building the object are set correctly), it has no limitation beyond that of the operating system for the size of data sets it can deal with.

1.3.2 Literature Review

The algorithm BIRCH was first introduced in Zhang et al. (1997) in a Computer Science subject area journal, and focused mostly on operational details such as a self-sizing mechanism for fitting within a certain sized memory. In terms of clustering, this article used a hierarchical clustering

paradigm on the subclusters for providing good quality initialization to a further algorithm, and briefly discusses other clustering methods such as CLARANS (Ng and Han, 2002) and k-means.

A direct extension of BIRCH is given in Chiu et al. (2001), which makes BIRCH applicable to data of mixed type attributes. This is done through a new clustering feature, given by

$$CF_J = (n_J, LS_J, SS_J, NB_J)$$

where the additional element, NB_J, contains a table for each categorical attribute, and a summary of counts for each categorical attribute in subcluster J.

In the broader family of algorithms that are applicable to large data sets, there are many examples that are variations of BIRCH. Two of these are STING (Wang et al., 1998) and CURE (Guha et al., 2001). The STING (Statistical Information Grid-based) algorithm divides the spatial area into rectangular cells, and employs a hierarchical structure such that differing levels of cells correspond with different resolutions of the data. Summary statistics for each cell are maintained, and are used much in the same way as the CFs in BIRCH. However, the actual contents of the CFs were different, and more orientated towards SQL-type queries.

In CURE (Clustering Using Representatives), a hierarchical clustering algorithm is also used. However, its process of forming the hierarchy is somewhat different. CURE begins by drawing a random sample of points out of the data set, which then become representative points. These points are then "shrunk" by a fraction α towards the centroid of the set of data, and then the closest points are merged together. This process is repeated, yielding at the end multiple representative points for each cluster.

Finally, BIRCH is discussed in the context of real-time data in Guha et al. (2003), as an algorithm that is capable of receiving data in increments without needing to hold the "old" data in memory. This feature is referred to as a data stream, and the article goes on to describe various characteristics that define this type of data.

1.3.3 Algorithms Extended

The applicability of the BIRCH algorithm is defined by a) the specification of clustering features that can be updated by data arriving singly, and b) the ability of an underlying objective function to be specified in terms of the clustering features instead of the underlying observations. If these two criteria are met, then BIRCH can be used to reduce the solution space of the combinatorial problem, and enable analysis of data sets that would otherwise be too large to fit in memory.

Clearly a number of combinatorial algorithms could be adapted for use with BIRCH. In Chapter 3, we demonstrate this approach with the following algorithms: Least Trimmed Squares (LTS, Rousseeuw and Leroy, 1987), Minimum Covariance Determinant (MCD, *ibid*) and Robust Linear Grouping Analysis (RLGA, García-Escudero et al., 2007).

LTS is a robust regression estimator, and is concerned with finding the subset of $h = \alpha n, \alpha \in (0.5, 1]$ observations that minimize the squared residuals such that

$$\sum_{i=1}^{h} r^2_{(i)} \tag{1.3}$$

where $r_i^2 = (y_i - \mathbf{x}_i^\top \hat{\boldsymbol{\beta}})^2$ and $r^2_{(i)}$ denotes the i-th order statistic of this residual. The mechanism

for optimizing this equation is called fast-LTS, was first introduced in Rousseeuw and van Driessen (2006), and is given by the following algorithm:

Algorithm 1.3.1 (fast-LTS)
For a number of different, randomly selected subsamples of size d:

1. Use the random subsample to compute the ordinary least squares (OLS) regression estimator $\hat{\boldsymbol{\beta}}$.
2. Iterate the following steps until convergence, or a specified number of iterations has occurred:
   (a) Let $H \leftarrow \{$ the h smallest squared residuals, $r^2 = (\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}})^\top(\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}})\}$.
   (b) Calculate the OLS estimator $\hat{\boldsymbol{\beta}}$ based on the observations in H.

   These steps are commonly referred to as "concentration steps"

Select the solution with the smallest LTS, as given by equation (1.3).

MCD is a robust estimator for location and dispersion, and is concerned with finding the subset of $h = \alpha n, \alpha \in (0.5, 1]$ observations that have the sample covariance with smallest determinant, as given by

$$\arg\min_{J \in \mathcal{H}} |S_J| \tag{1.4}$$

where $\mathcal{H}$ is the solution space of all subsets of size h,

$$S_J = \frac{1}{h-1}\sum_{i \in J}(\mathbf{X}_i - \bar{\mathbf{X}}_J)(\mathbf{X}_i - \bar{\mathbf{X}}_J)^\top \qquad \text{and} \qquad \bar{\mathbf{X}}_J = \frac{1}{h}\sum_{i \in J}\mathbf{X}_i.$$

The algorithm for minimizing this objective function is called fast-MCD (Rousseeuw and van Driessen, 1999), and is given by the following algorithm.

Algorithm 1.3.2 (fast-MCD)
For a number of different, randomly selected subsamples of size $(d+1)$:

1. Use the initial subsample to calculate an initial estimate of the sample mean and covariance.
2. Iterate the following steps until convergence, or a specified number of iterations has occurred:
   (a) Let $H \leftarrow \{$ the h points with smallest Mahalanobis distances: $(\mathbf{X} - \hat{\boldsymbol{\mu}})^\top \mathbf{S}^{-1}(\mathbf{X} - \hat{\boldsymbol{\mu}})\}$.
   (b) Calculate the new $\hat{\boldsymbol{\mu}}$, $\mathbf{S}$ based on the observations in H.

   These steps are commonly referred to as "concentration steps"

Select the solution with the smallest MCD, as given by equation (1.4).

Clearly, both LTS and MCD have much in common in their approach to optimizing an objective function. As it turns out, these algorithms are also both amended in the same fashion if the data set

is large. For large n, a nested approach to optimization is used, whereby a number of subsamples of size $n_s << n$ are taken, and two concentration steps are performed from a variety of random starts within each subsample. For each subsample the best 10 solutions (with respect to the objective function) are stored. Then the subsamples are pooled together, and the best solutions from the previous stage are used to initialize a number of concentration steps. Once again, the best 10 solutions are retained, and used to seed concentration steps on the whole data set.

The last algorithm extended for use with BIRCH is Robust LGA (RLGA) (García-Escudero et al., 2007), which is the robust equivalent of LGA. The goal of RLGA is to find the $h = \alpha n$ subset with the minimum residual orthogonal sum-of-squares to the respective hyperplanes, and has a very similar algorithm as given for LGA in Algorithm 1.1.2, with the following amendments:

3a. *Select the αn subset:* Order the orthogonal residual of each observation to its closest hyperplane. Select the smallest αn observations, and use just those observations to recalculate the hyperplanes using orthogonal regression.

5. *Resampling:* Repeat steps 2-4 a number of times and select the solution with the smallest trimmed orthogonal sum of squares.

This algorithm is implemented in the R package lga.

Bibliography

J. D. Banfield and A. E. Raftery. Ice flow identification in satellite images using mathematical morphology and clustering about principal curves. *Journal of the American Statistical Association*, 87(417):7–16, 1992.

J. D. Banfield and A. E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49(3):803–821, 1993.

A. Ben-Hur and I. Guyon. *Detecting Stable Clusters Using Principal Component Analysis*, pages 159–182. Methods in Molecular Biology. Humana Press, Mar. 2003.

A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

W. C. Chang. On using principal components before separating a mixture of two multivariate normal distributions. *Applied Statistics*, 32(3):267–275, 1983.

T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris. A robust and scalable clustering algorithm for mixed type attributes in large database environment. *Proceedings of the seventh ACM SIGKDD*, 2001.

C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97:611–631, 2002.

L. García-Escudero, A. Gordaliza, R. San Martín, S. Van Aelst, and R. H. Zamar. Robust linear clustering. *Unpublished Manuscript*, 2007.

P. Gawrysiak, H. Rybinski, and M. Okoniewski. Regression - yet another clustering method. In *International Intelligent Information Systems Conference*, Advances in Soft Computing Journal. Springer-Verlag, 2001.

M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, pages 299–312, 2001.

M. Girolami. Mercer Kernel Based Clustering in Feature Space. *IEEE Transactions on Neural Networks*, 13(3):780–784, 2002.

S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. *Information Systems*, 26(1):35–58, 2001.

S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data stream: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, May/June 2003.

J. Harrington. *lga: Tools for linear grouping analysis (LGA)*, 2008. R package version 1.1-0, and available on CRAN.

I. Jollife, B. Jones, and B. Morgan. Utilising clusters: A case-study involving the elderly. *Journal of Royal Statistical Society. Series A*, 145(2):224–236, 1982.

I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 14, 2001.

R. T. Ng and J. Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

P. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection*. Wiley, 1987.

P. J. Rousseeuw and K. van Driessen. Computing LTS regression for large data sets. *Data Mining and Knowledge Discovery*, 12:29–45, 2006.

P. J. Rousseeuw and K. van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, August 1999.

R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *J. R. Statist Soc. B*, 63(2):411–423, 2001.

S. Van Aelst, X. Wang, R. H. Zamar, and R. Zhu. Linear Grouping Using Orthogonal Regression. *Computational Statistics & Data Analysis*, 50(5):1287–1312, Mar. 2006.

W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proc. 24th Int. Conf. Very Large Data Bases*, 1998.

W. J. Welch. Algorithmic complexity: three NP- hard problems in computational statistics. *Journal of Statistical Computation and Simulation*, 15(1):17–25, 1982.

K. Yeung. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17 (9):763–774, 2001.

T. Zhang, R. Ramakrishnan, and M. Livny. *BIRCH: A New Data Clustering Algorithm and Its Applications*, volume 1 of *Data Mining and Knowledge Discovery*, pages 141 – 182. Springer Netherlands, June 1997.

Chapter 2

Extending Linear Grouping Analysis*

2.1 Introduction

Linear Grouping Analysis (LGA, Van Aelst et al., 2006) involves grouping data around hyperplanes, and utilizes orthogonal regression to select hyperplanes that minimize the sum of the squared orthogonal distances of each observation to its representative hyperplane. This has the advantage that it treats all variables in a symmetric way, and does not require the identification of "response" or "explanatory" variables (Zamar, 1989).

In this article we briefly review LGA, and then introduce two new methods for extending its applicability. The first method introduces a variant of orthogonal regression, called *partially orthogonal regression*, which is motivated by high dimensional data which can produce clusters with covariance matrices that are rank deficient. The second method relaxes the linearity requirement of LGA by utilizing a machine learning technique of transforming the data set into a *feature space* and then performing LGA in this space. We then conclude with some examples, and a brief discussion including promising directions for future research.

2.2 Linear Grouping Criterion

The goal of LGA is to find k hyperplanes, parametrized by $\mathbf{w}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$ and denoted $\mathcal{H}(\mathbf{w}_i, b_i) \equiv \mathcal{H}_i \, (i = 1, \ldots, k)$ such that $\mathcal{H}_i = \{\mathbf{z} : \mathbf{w}_i^\top \mathbf{z} = b_i\}$, with the smallest residual orthogonal sum of squares (ROSS). With our data given by $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, a membership matrix $\mathbf{Z}$ with binary elements $(\mathbf{Z})_{ij} = z_{ij} \in \{0, 1\}, i = 1, \ldots, k, j = 1, \ldots, n$, where a '1' means that observation j is a member of cluster i, and a set of hyperplanes $\mathcal{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_k\}$, the objective function for LGA is given by

$$\mathcal{R}(\mathbf{x}_1, \ldots, \mathbf{x}_n | \mathcal{H}, \mathbf{Z}) = \mathcal{R}(\mathcal{X} | \mathcal{H}, \mathbf{Z}) = \sum_{i=1}^{k} \sum_{j=1}^{n} z_{ij} \, d_O^2(\mathbf{x}_j, \mathcal{H}_i), \tag{2.1}$$

where

$$d_O^2(\mathbf{x}_j, \mathcal{H}_i) = (\mathbf{w}_i^\top \mathbf{x}_j - b_i)^2$$

is the square orthogonal distance of $\mathbf{x}_j$ to the hyperplane $\mathcal{H}_i$, and $\mathcal{X}$ denotes our set of observations. To optimize equation (2.1) with respect to the set of hyperplanes $\mathcal{H}$ and the membership matrix $\mathbf{Z}$, Van Aelst et al. (2006) utilizes a two-step approach. For an initial partition given by membership matrix $\mathbf{Z}_0$, do the following:

*A version of this chapter will be submitted for publication. Harrington, J. A. and Salibian-Barrera, M. Extending Linear Grouping Analysis. Full proofs for every result, lemma and theorem are given in Appendix A.

Step I - For each cluster, as defined by the membership matrix $\mathbf{Z}_j$, we calculate the best fitting hyperplane. I.e.

$$\mathcal{H}_j = \arg\min_{\mathcal{H}} \mathcal{R}(\mathcal{X}|\mathcal{H}, \mathbf{Z}_j).$$

Step II - Then, given the hyperplane for each cluster, we can calculate the distance of every observation to each hyperplane, and assign each observation to the closest. This is used to update the membership matrix. I.e.

$$\mathbf{Z}_{j+1} = \arg\min_{\mathbf{Z}} \mathcal{R}(\mathcal{X}|\mathcal{H}_j, \mathbf{Z}).$$

This procedure continues until convergence – equivalently until $\mathbf{Z}_{j+1} = \mathbf{Z}_j$ – or $j = N$, where N is a user-specified maximum number of iterations. Note that Step I could be decomposed into

$$\min_{\mathcal{H}} \mathcal{R}(\mathcal{X}|\mathcal{H}, \mathbf{Z}_j) = \min_{\mathcal{H}} \left(\sum_{i=1}^{k} \mathcal{R}(\mathbf{x}_i|\mathcal{H}_i) \right),$$

where $\mathbf{x}_i$ are the observations in partition i (as determined by $\mathbf{Z}_j$), and are mutually exclusive sets.

$$\Rightarrow \min_{\mathcal{H}} \mathcal{R}(\mathcal{X}|\mathcal{H}, \mathbf{Z}_j) = \sum_{i=1}^{k} \min_{\mathcal{H}_i} \mathcal{R}(\mathbf{x}_i|\mathcal{H}_i).$$

This means that to find the set of hyperplanes that minimizes Step I, it is sufficient to find the hyperplane that minimizes each cluster. In order to find the best fitting hyperplane in Step I, we use the following, well known, result.

Result 2.2.1 (Orthogonal Regression) Let $\mathbf{x}_1, \ldots \mathbf{x}_n \in \mathbb{R}^d$ have sample covariance matrix $\mathbf{S}$. Let the eigenvectors of $\mathbf{S}$ be $\mathbf{e}_1, \ldots, \mathbf{e}_d$ with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots > \lambda_d \geq 0$ and $\text{Rank}(\mathbf{S}) \geq (d-1)$. The solution to

$$\min_{\mathbf{a},b} \sum_{i=1}^{n} (\mathbf{a}^\top \mathbf{x}_i - b)^2$$

subject to the constraint $||\mathbf{a}|| = 1$ is given by

$$\mathbf{a} = \mathbf{e}_d \text{ and } b = \mathbf{e}_d^\top \bar{\mathbf{x}}.$$

Note that in this definition restrictions are given with respect to the smallest eigenvalues, as it is explicitly stated that the smallest eigenvalue must be unique (i.e. have multiplicity one). This is because if more than one of the smallest eigenvalues are equal, then the eigenvectors associated with these eigenvalues are no longer unique. In the context of orthogonal regression this situation is most likely to occur when there are fewer observations in a cluster than dimensions (more specifically, $n < d-1$), or if one column is linearly dependent on the others, as this would result in multiple zero-valued eigenvalues and a covariance matrix rank less than $(d-1)$.

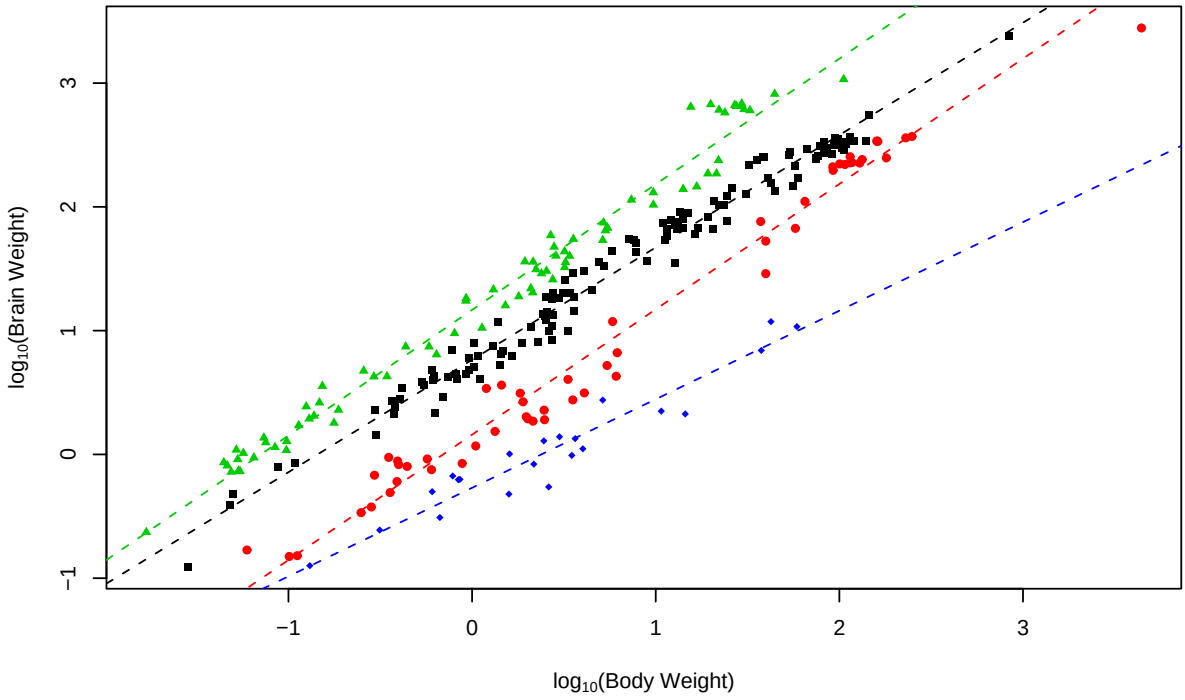


Figure 2.1: An example of LGA applied to the allometry data set from Crile and Quiring (1940), with $k = 4$. The different colours and symbols correspond to the clusters found, and the lines are the hyperplanes associated with each cluster.

It is clear that orthogonal regression is very closely related to principal components analysis, which is looking for the vector which specifies the maximal variance, and is given by the eigenvector associated with the largest eigenvalue. As it turns out the solution is essentially the hyperplane formed by the basis of $(d-1)$ principal components, which is the hyperplane orthogonal to the last principal component.

The LGA algorithm is initialized from a number of different random starting partitions to facilitate the search for a global minimum. These are given by hyperplanes determined by k random subsamples of size d. The number of starts is selected such that the probability of a "good" starting configuration is 95%, where a good start is defined as the configuration that is a subset of the actual partitioning. The logic behind this approach is that, given a starting configuration with hyperplanes resembling the "true" solution, Step II will then produce the correct membership matrix. For example, consider clustering n observations into k clusters, with the correct partition given by $C_1, \ldots, C_k$. Then a good starting configuration would comprise of $I_1, \ldots, I_k$ where $|I_j| = d$, and $I_j \subset C_j \forall j \in \{1, \ldots, k\}$.

Example. We reproduce an example from Van Aelst et al. (2006) using an allometry data set, obtained from Crile and Quiring (1940). This data set studies the relationship between the brain and body weight for $n = 282$ vertebrates, and a logarithmic (base 10) transformation has been applied to each column. Setting $k = 4$, a plot of the clusters is given in Figure 2.1. The R package lga was used to find the clusters.

2.3 Partially Orthogonal Regression

In the discussion associated with Result 2.2.1 above, the importance of the multiplicity of the smallest eigenvalue of the sample covariance matrix was highlighted. One situation in which repeated eigenvalues occur is when the rank of the covariance matrix for a cluster is less than $(d-1)$ – which is often caused by small clusters and/or high dimensions. This is a situation that occurs repeatedly in Section 2.4, where we apply transformations that increase the dimension considerably.

One approach for dealing with clustering when the rank of the sample covariance matrix for any cluster is less than $(d-1)$ would be via dimension reduction. Methods of dimension reduction have been discussed previously in clustering literature. For example, in Jollife et al. (1982) principal components analysis is applied on a data set to reduce the number of dimensions from 20 to 10, and an approach similar to k-means applied to the PC scores. In Jolliffe (2002), a section is devoted to describing heuristically when it may be advantageous to apply PCA prior to clustering. However, in Chang (1983), it was shown that, at least in the case of a mixture of two normals, the first few PCs may contain less cluster structure information than other PCs. And in Yeung (2001), applying PCA prior to clustering gene expression data was found not to work well empirically.

In the context of LGA, it can easily be shown that a dimension reduction approach using PCA is unlikely to work. While dimension reduction would work on clusters whose membership is given *a priori*, since the actual structure of each cluster is then known, applying PCA on a whole data set prior to clustering would not take into account the (probably) differing structures, and, at worst, project data from different clusters onto the same space. Clearly, any dimension reduction should take place after a partition has been proposed.

Unfortunately, since LGA relies heavily on the use of eigenvectors via orthogonal regression, we cannot simply ignore dimension problems that result from the covariance matrix of any cluster having rank less than $(d-1)$, as the last eigenvector is not unique, and this is the vector used as per Result 2.2.1 for the calculation of the orthogonal distance. Therefore, in this section we introduce a new measure of distance, called *Partially Orthogonal Regression* (POD), as an alternative to orthogonal regression for when this is the case.

Definition 2.3.1 Let $\mathbf{C} \in \mathbb{R}^{d\times d}$ be a sample covariance matrix of rank $1 < \nu < (d-1)$, with $(\mathbf{e}_i, \lambda_i),\ i = 1, \ldots, d$ the eigenvector/value pairs such that $\lambda_1 \geq \ldots \geq \lambda_\nu > \lambda_{\nu+1} = \ldots = \lambda_d = 0$. Let $\boldsymbol{\mu} \in \mathbb{R}^d$ be a fixed point and let $\mathbf{E} = (\mathbf{e}_1|\ldots|\mathbf{e}_\nu)$ be the basis of a hyperplane $\mathcal{H}$ going through $\boldsymbol{\mu}$, consisting of the eigenvectors of $\mathbf{C}$ with positive eigenvalues.

Then the Partially Orthogonal Distance (POD) of a point $\mathbf{y} \in \mathbb{R}^d$ to $\mathcal{H}$ is given by

$$d_P^2(\mathbf{y}; \mathbf{E}, \boldsymbol{\mu}) = \mathbf{z}^\top(\mathbf{P}_1 + \mathbf{P}_2/\lambda_\nu)\mathbf{z}, \tag{2.2}$$

where $\mathbf{z} = \mathbf{y} - \boldsymbol{\mu},\ \mathbf{P}_1 = \mathbf{I} - \mathbf{E}\mathbf{E}^\top,\ \mathbf{P}_2 = \mathbf{e}_\nu\mathbf{e}_\nu^\top$,

$$\mathbf{E} = \begin{pmatrix} | & | & & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_\nu \\ | & | & & | \end{pmatrix}.$$

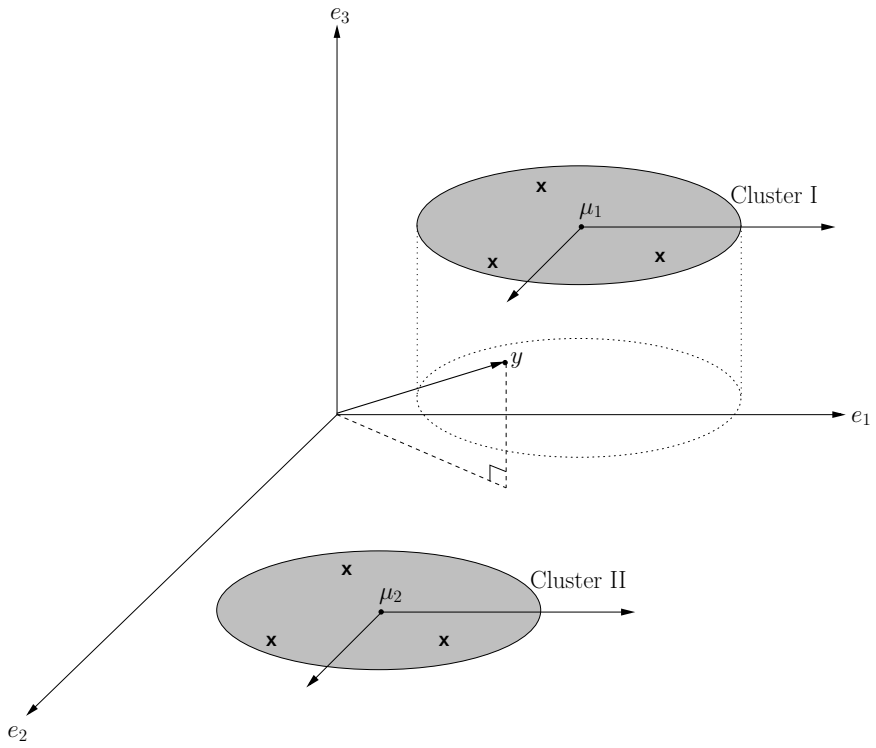


Figure 2.2: Introducing Partially Orthogonal Regression. We have two clusters, centred at $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ with identical covariance matrix $\mathbf{C}$ with eigenvalues $\lambda_1 > \lambda_2 > \lambda_3 = 0$. Note that the axes are the eigenvectors of $\mathbf{C}$, and these axes are shown inside the cluster to emphasize their structure. An observation which are we trying to cluster is given by $\mathbf{y}$, and is not part of either cluster.

It can be easily shown that equation (2.2) can be broken into two separate distances, comprising:

$$d_P^2(\mathbf{y}; \mathbf{E}, \boldsymbol{\mu}) = d_E^2(\mathbf{y}, \mathbf{p}) + d_S^2(\mathbf{p}, \tilde{\mathbf{y}}), \tag{2.3}$$

where $d_E^2(\mathbf{y}, \mathbf{p}) = \mathbf{z}^\top \mathbf{P}_1 \mathbf{z}$ is the Euclidean distance of $\mathbf{y}$ to its projection $\mathbf{p}$ on the basis $\mathbf{E}$ going through $\boldsymbol{\mu}$, and $d_S^2(\mathbf{p}, \tilde{\mathbf{y}}) = \mathbf{z}^\top \mathbf{P}_2 / \lambda_\nu \, \mathbf{z}$ is a scaled distance of $\mathbf{y}$ to the hyperplane orthogonal to the vector $\mathbf{e}_\nu$ and going through $\boldsymbol{\mu}$.

In order to motivate why this approach is appropriate when there are fewer observations than dimensions in any cluster, we give an illustration in $\mathbb{R}^3$, where only one eigenvalue is zero. Although orthogonal regression could still be applied here, we utilize this simple illustration to give an intuition into this approach without needing to imagine higher dimensions where any intuition might be lost. Our example is sketched in Figure 2.2.

We have two clusters (I and II) with centres $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbb{R}^3$ and both clusters have covariance matrix $\mathbf{C} \in \mathbb{R}^{3\times 3}$ with eigenvalues $\lambda_1 > \lambda_2 > \lambda_3 = 0$. A situation where this might occur is where each of the clusters has been built with only three observations, as indicated in the figure. Now let us consider a new observation, $\mathbf{y} \in \mathbb{R}^3$, which is not part of either cluster, and ponder the question of which cluster it should belong to, keeping in mind that this approach needs to generalize to where more than one eigenvalue is equal to zero (i.e. $\lambda_3 = 0$ has multiplicity greater than one).

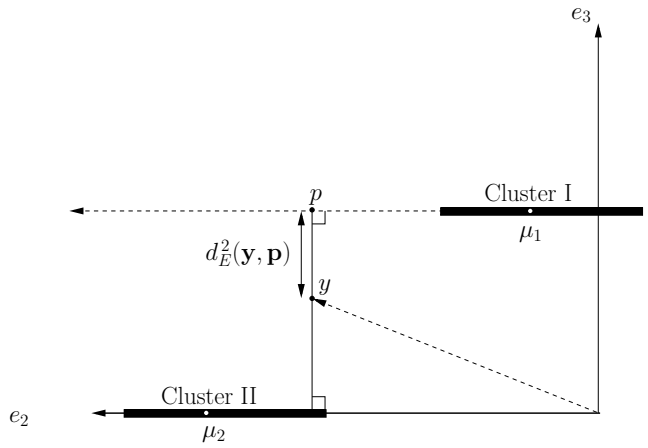


Figure 2.3: Looking at the $\mathbf{e}_2 - \mathbf{e}_3$ plane. Note that the thickness in the clusters in the direction of $\mathbf{e}_3$ is for illustrative purposes only.

Two new alternatives for measuring the distance of $\mathbf{y}$ to each cluster are:

Distance I – The orthogonal distance of $\mathbf{y}$ to the hyperplane given by the basis made up of the eigenvalues with only positive eigenvalues; or

Distance II – The orthogonal distance of $\mathbf{y}$ to the hyperplane orthogonal to the eigenvector with the smallest positive eigenvalue.

Next we shall discuss briefly each of the distances above, show that these distances correspond with the elements identified in equation (2.3), and finally justify why the inclusion of both distances is logical.

Distance I – The orthogonal distance of y to the hyperplane given by the basis made up of the eigenvalues with only positive eigenvalues: In our illustration, the basis with positive eigenvalues is made up of $(\mathbf{e}_1, \mathbf{e}_2)$, and is orthogonal to $\mathbf{e}_3$. Therefore, in this situation the orthogonal distance to a hyperplane defined by this basis is equivalent to the orthogonal distance as it would be calculated using Result 2.2.1 (standard orthogonal regression). This distance is illustrated in Figure 2.3, and is denoted $d_E^2(\mathbf{y}, \mathbf{p})$ in equation (2.3).

However, there are two problems with using orthogonal regression directly. Firstly, since our desired application includes the situation where the multiplicity of the smallest eigenvalue is greater than one, the smallest eigenvector may not be unique, and therefore the vector $\mathbf{e}_3$ cannot be used for calculation. Thus, instead of calculating distances using orthogonal vectors (as in Result 2.2.1), we would use the projection of $\mathbf{y}$ onto the basis for which the eigenvectors are well defined (in our example, $\mathbf{e}_1$ and $\mathbf{e}_2$), and then calculate the distance of our observation to this projection.

Furthermore, with this distance we are measuring how close our observation is to the hyperplane orthogonal to $\mathbf{e}_3$ (which extend to infinity in the directions $\mathbf{e}_1$ and $\mathbf{e}_2$), without considering how close $\mathbf{y}$ is to the linear structure shown in Figure 2.3. That is, $\mathbf{y}$ may be close to the hyperplane

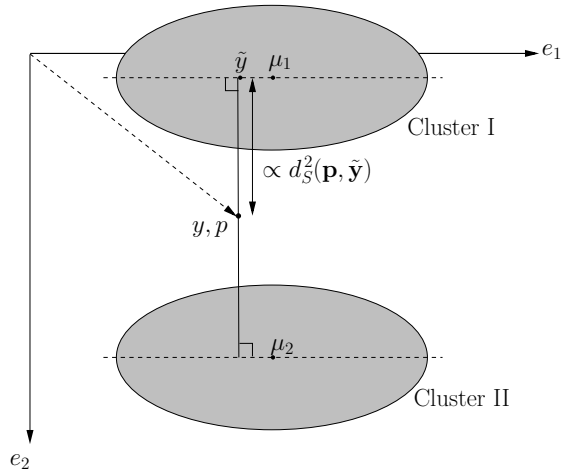


Figure 2.4: A "bird's-eye-view" looking down onto the $\mathbf{e}_1 - \mathbf{e}_2$ plane.

orthogonal to $\mathbf{e}_3$ for one cluster, but close to the data associated with another cluster in all other dimensions.

Distance II – The orthogonal distance of y to the hyperplane orthogonal to the eigenvector with the smallest positive eigenvalue: In our illustration, the eigenvector associated with the smallest positive eigenvalue is $\mathbf{e}_2$, and the distance of $\mathbf{y}$ to this hyperplane is proportional to $d_S^2(\mathbf{p}, \tilde{\mathbf{y}})$, as seen in Figure 2.4. Clearly this is equivalent to performing orthogonal regression by ignoring the contribution of $\mathbf{e}_3$, and is equivalent to the orthogonal distance in the $\mathbf{e}_1 - \mathbf{e}_2$ plane.

While the linear structure of the data is visible in this plane, there are three possible faults with this approach.

- Firstly, if the two clusters were different only in the $\mathbf{e}_3$ direction, then this would not be a useful measure as it would be equal for both clusters.
- Secondly, we are discarding any information present in $\mathbf{e}_3$ that might be relevant in deciding which cluster the observation belongs to (as seen in Figure 2.3).
- Finally, when calculating distance we are not considering the structure of the data in a Mahanalobian-sense; i.e. the distance in the direction of $\mathbf{e}_2$ should be expressed in terms of how spread out the data is in that direction, in the same way that Mahalanobis distance considers distance to the centre of a data set in terms of the confidence contours of the data's covariance structure.

In order to reflect the structure of the cluster, we scale the Euclidean distance of $\mathbf{p}$ to $\tilde{\mathbf{y}}$ by the smallest positive eigenvalue, since this distance is solely a function of the matching eigenvector, addressing the third fault.

These arguments show that there is some value in each approach, and so we combine them into our new measure of distance, as given in equation (2.3). In order to generalize our illustration to

d dimensions, let us assume that we now have ν positive eigenvalues – in practise we select ν such that $c\%$ of the total variation in the data is explained i.e.

$$\arg\min_{\nu} \frac{\sum_{i=1}^{\nu} \lambda_i}{\sum_{j=1}^{d} \lambda_j} > c. \tag{2.4}$$

Consider the basis of interest – in our illustration we have only $\lambda_3 = 0$ which defines the basis (with any meaningful variation) as $\mathbf{e}_1 - \mathbf{e}_2$; in d dimensions this corresponds with the basis $\mathbf{E} = (\mathbf{e}_1|\ldots|\mathbf{e}_\nu)$. Then Distance I corresponds to the distance between $\mathbf{y}$ and its projection, $\mathbf{p}$, on the basis $\mathbf{E}$ and going through $\boldsymbol{\mu}$ – this gives our result for $d_E^2(\mathbf{y}, \mathbf{p})$.

Finally, recall Distance II, which measures how far away $\mathbf{y}$ is to the clusters in the $\mathbf{e}_1 - \mathbf{e}_2$ plane, as shown in Figure 2.4. Given that in this basis $\mathbf{y}$ is equivalent to $\mathbf{p}$ (its orthogonal projection on to the basis), the distance from $\mathbf{y}$ to a cluster corresponds to the distance of $\mathbf{p}$ to $\tilde{\mathbf{y}}$, where $\tilde{\mathbf{y}}$ is the projection of $\mathbf{y}$ onto the hyperplane orthogonal to $\mathbf{e}_\nu$. Given that we wish to express this distance with respect to the structure of the cluster, we scale this distance by the eigenvalue λ_ν, as the distance in this direction is proportional only to $\mathbf{e}_\nu$, and no other eigenvector. This is the distance denoted $d_S^2(\mathbf{p}, \tilde{\mathbf{y}})$. By then adding $d_E^2(\mathbf{y}, \mathbf{p})$ and $d_S^2(\mathbf{p}, \tilde{\mathbf{y}})$, we then have Definition 2.3.1.

Remark. Consider Definition 2.3.1, Figures 2.2 – 2.4, and assuming $\lambda_\nu = 1$. Then $d_S^2(\mathbf{p}, \tilde{\mathbf{y}}) = d_E^2(\mathbf{p}, \tilde{\mathbf{y}})$, and by Pythagoras Theorem

$$d_P^2(\mathbf{y}; \mathbf{E}, \boldsymbol{\mu}) = ||\mathbf{y} - \mathbf{p}||^2 + ||\mathbf{p} - \tilde{\mathbf{y}}||^2 = ||\mathbf{y} - \tilde{\mathbf{y}}||^2 = d_E^2(\mathbf{y}, \tilde{\mathbf{y}})$$

i.e. the Partially Orthogonal Distance can be decomposed into two orthogonal distances, and is equivalent to the Euclidean distance from $\mathbf{y}$ to $\tilde{\mathbf{y}}$.

In order to perform LGA with POD, we propose the following algorithm.

Algorithm 2.3.2 (LGA-POD)
LGA with partially orthogonal distances is made up of the following five steps:

1. *Generation of starting value:* a starting configuration is generated consisting of a partition of the data set into k mutually exclusive sets of $nsamp$ observations (i.e. an observation can belong at most in one set). For each of these sets, calculate the number of eigenvectors that explain 99.99% of the data using equation (2.4). Call this ν.

2. *Initialization of groups:* for each starting hyperplane with corresponding ν, calculate the partially orthogonal distance of all observations to that hyperplane. Then assign each point to the hyperplane which it is closest.

3. *Iterative refinement:* Recalculate step 3 for each resulting hyperplane until either a) the solution converges (i.e. the membership sets remain the same for two consecutive iterations) or b) the solution fails to converge after an arbitrary number of iterations (e.g. 10).

4. *Resampling:* Repeat steps 2-4 a number of times and select the solution with the smallest partially orthogonal sum of squares.

The rationale behind this algorithm, in particular the setting of ν, is similar to that given in Section 2.2, insofar as we randomly select a number of different starting configurations with the hope

of getting one that is a subset of the true partitioning. As before, this is intuitively advantageous, since the hyperplanes will already be the appropriate shape, but now also the number of eigenvectors (ν) will be set correctly.

The setting of the *nsamp* parameter requires some care. It is desirable to set the value as small as possible, as this makes it much easier to find a "good" starting configuration. However, setting *nsamp* too low will exacerbate any rank problems further, as the number of eigenvectors with positive eigenvalues corresponds directly with the initial number of observations in the build set.

Example. We simulate two rank deficient clusters by simply adding a constant column to each. This is comparable with the data set illustrated in Figure 2.2. In this case we have generated two clusters of size 30 from a multivariate normal $\mathbf{X}_i = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), i = 1, 2$, with $\boldsymbol{\mu} = (0\ 0)^\top$ and $\boldsymbol{\Sigma}$ a diagonal matrix with 2 on the diagonal. Then for the first cluster we set $\mathbf{Y}_1 = (\mathbf{X}_1|\mathbf{0})$ and for the second $\mathbf{Y}_2 = (\mathbf{X}_2|\mathbf{5})$. Clearly there is no variation in the third dimension, and the covariance matrix of either cluster will be of rank two. Applying the algorithm to this data set yields a perfect clustering, and a plot is given in Figure 2.5(a). Note that in the first two dimensions the clusters are not separable.

Example. We simulate two clusters with more dimensions than observations, and therefore have covariance matrices that are less than full rank. Once again using the multivariate normal distribution, we generate 15 observations from a $\mathcal{N}_{20}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ and 15 observations from a $\mathcal{N}_{20}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}_1 = (0, \ldots, 0)^\top$ and $\boldsymbol{\mu}_2 = (3, 3, 3, 0, \ldots, 0)^\top$. The covariance matrix is given by a diagonal matrix with elements $(4, 2, 0.1, \ldots, 0.1)$. Applying the algorithm to this data set yields a perfect clustering, and a plot is given in Figure 2.5(b).

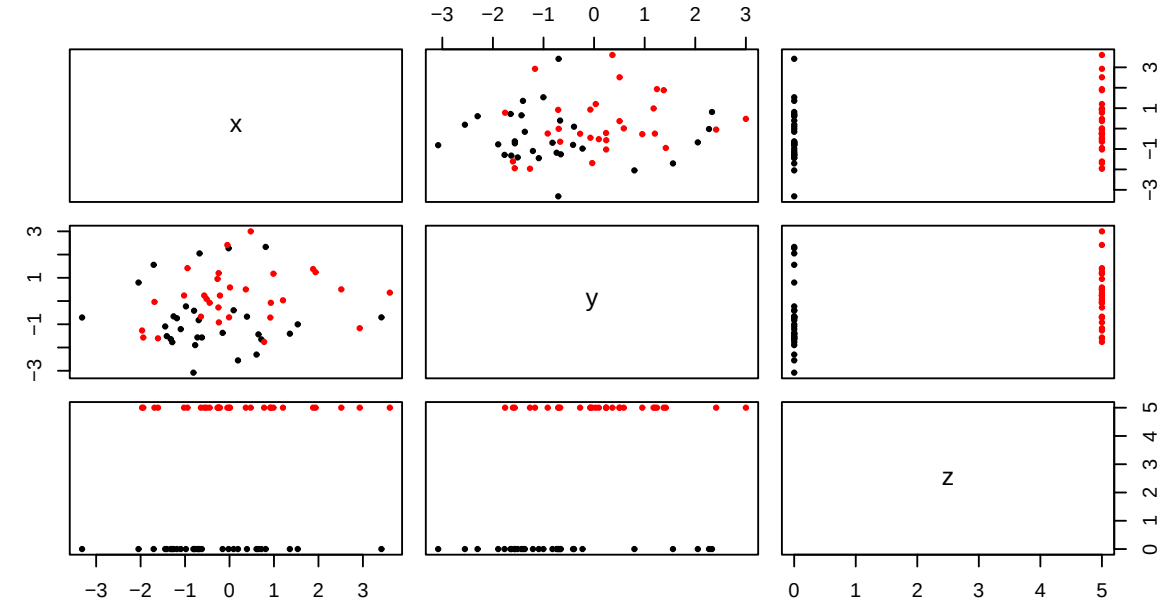
2.4 Linear Grouping in the Feature Space

In Section 2.2 we introduced a criterion for clustering around hyperplanes in Euclidean space. Clearly, the underlying assumption is that the data in these clusters are aligned in a linear fashion. In this section we relax this assumption, by transforming the data into a *Feature Space* where the data may have a linear structure, and performing LGA within this space. We denote the transformation

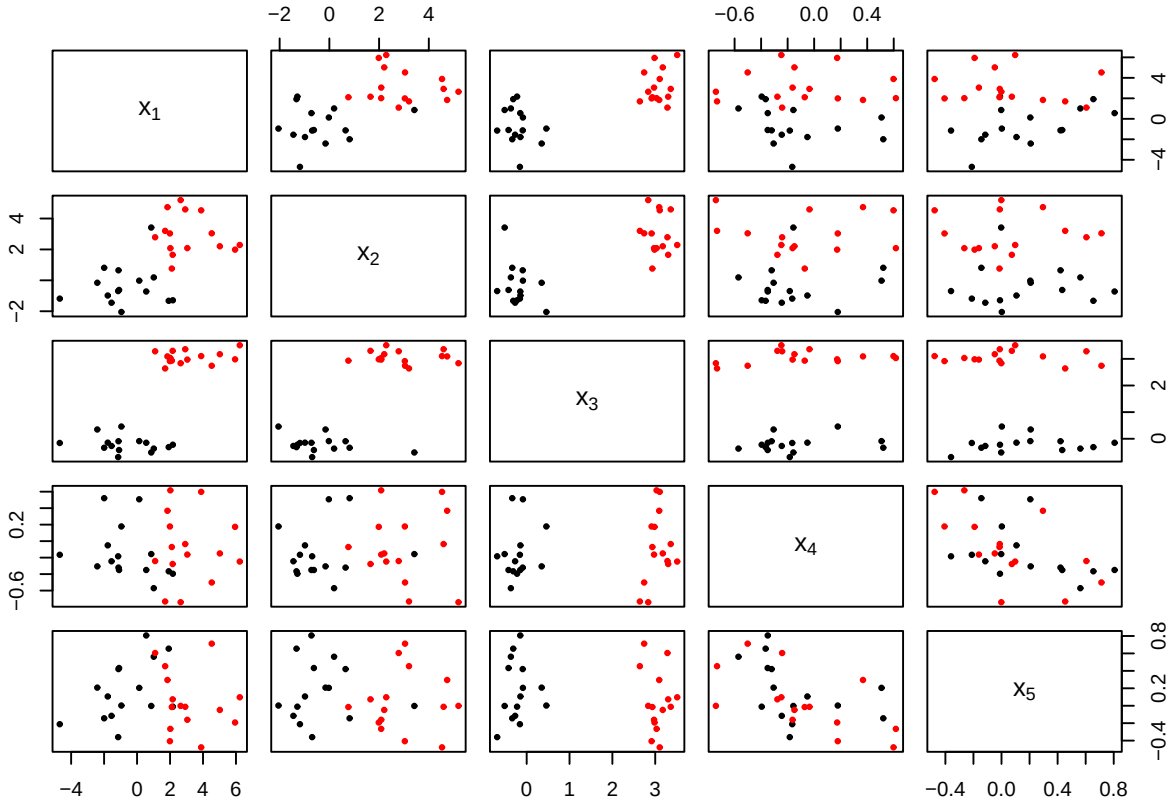
$$\Phi : \mathbb{R}^d \to \mathcal{F}$$

with $\mathcal{F} \in \mathbb{R}^F$, and refer to the corresponding clustering algorithm in the feature space as kLGA. Since it can be shown that the LGA objective function (equation (2.1)) only depends on the cross-products of the observations, rather than working with the transformed data directly, we instead manipulate the cross-product $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^\top \Phi(\mathbf{x}')$, called the kernel function, utilizing what is commonly called the *kernel trick* in machine learning literature. The advantage of this is flexibility and generality; if we wish to specify the transformation directly, it is trivial to calculate the kernel function i.e. the transformation is *explicitly* defined. However, it is also possible to specify the kernel function directly, with the transformation being *implicitly* defined. In the latter case, while it may not be possible to describe the actual transformation, a result called *Mercer's Theorem* provides necessary and sufficient conditions for the kernel function such that the existence of the transformation is guaranteed. Interested readers should refer to Schölkopf and Smola (2002) for further details.

There are various examples of unsupervised learning methods using feature spaces and kernel func-



(a) $\mathbf{Y}_1 = (\mathbf{X}_1|\mathbf{0}),\ \mathbf{Y}_2 = (\mathbf{X}_2|\mathbf{5}), \mathbf{X}_i = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$



(b) $\mathcal{N}_{20}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}), \boldsymbol{\mu}_1 = (0, \ldots, 0)^\top, \boldsymbol{\mu}_2 = (3, 3, 3, 0, \ldots, 0)^\top$

Figure 2.5: Examples of LGA-POD on data sets whose clusters have covariance matrix of less than full rank.

tion transformation in machine learning literature, though they are not as well known as the supervised learning methods (e.g. SVM classifiers (Schölkopf and Smola, 2002), SV Regression (*ibid*), and Least-Squares SVM (Suykens et al., 2002)). Examples of unsupervised methods include Kernel PCA (Schölkopf et al., 1998), Kernel Feature Analysis (Smola et al., 2002), and those with specific application in clustering such as Spectral Clustering (Ng et al., 2001), a feature space version of k-means (Girolami, 2002) and Support Vector Clustering (Ben-Hur et al., 2001). In this article, we have adopted a similar framework to that of (Girolami, 2002), though the details are substantially different.

In order to apply this methodology to LGA, we will transform the observations into the feature space and then express the objective function in terms of dot-products. Assuming that all covariance matrices in the feature space have rank of F or $(F-1)$, the results of Section 2.2 apply immediately. However, when the transformation is only given implicitly by the choice of kernel function, we will need to use the results in Kernel PCA (kPCA, Schölkopf et al., 1998) to find the eigenvectors of the covariance matrix in the feature space. Following this, we look at the case where the covariance matrices have rank less than $(F-1)$, which in turn closely resembles the development of the POD metric. Finally, we will discuss some of the technical challenges associated with this calculation.

2.4.1 Preliminaries and Notation

Applying the transformation in equation (2.1), we are interested in minimizing the objective function

$$\mathcal{R}\left(\Phi(\mathcal{X})|\mathcal{H},\mathbf{Z}\right)=\sum_{i=1}^{k}\sum_{j=1}^{n}z_{ij}\,d_O^2\left(\Phi(\mathbf{x}_j),\mathcal{H}_i\right)$$

with all the parameters defined the same as in equation (2.1). It is trivial to show that if the covariance matrix is F or $(F-1)$, then the best fitting hyperplane through the transformed observations has the same parameters as given in Result 2.2.1. This necessitates the spectral decomposition of the sample covariance matrix of the observations $\Phi(\mathbf{x}_i), i=1,\ldots,n$, given by

$$\mathbf{C}_\Phi=\frac{1}{n}\sum_{i=1}^{n}\left(\Phi(\mathbf{x}_i)-\bar{\Phi}(\mathbf{x}_j)\right)\left(\Phi(\mathbf{x}_i)-\bar{\Phi}(\mathbf{x}_j)\right)^\top=\frac{1}{n}\sum_{i=1}^{n}\tilde{\Phi}(\mathbf{x}_i)\tilde{\Phi}(\mathbf{x}_i)^\top,$$

where $\tilde{\Phi}(\mathbf{x}_i)=\Phi(\mathbf{x}_i)-\bar{\Phi}(\mathbf{x})$ and $\bar{\Phi}(\mathbf{x})$ is the sample mean of the transformed data. Define $\tilde{k}(\mathbf{x}_i,\mathbf{x}_j)=\tilde{\Phi}(\mathbf{x}_i)^\top\tilde{\Phi}(\mathbf{x}_j)$, and let the eigenvalues of $\mathbf{C}_\Phi$ be given by $\lambda_1\geq\lambda_2\geq\ldots\geq\lambda_{F-1}>\lambda_F\geq 0$, with corresponding eigenvectors $\mathbf{e}_i\in\mathbb{R}^F, i=1,\ldots,F$.

For computational ease, we generate the kernel matrix $\mathbf{K}\in\mathbb{R}^{n\times n}$, $(\mathbf{K})_{ij}=k(\mathbf{x}_i,\mathbf{x}_j)$ and the centred kernel matrix $\tilde{\mathbf{K}}\in\mathbb{R}^{n\times n}$, $(\tilde{\mathbf{K}})_{ij}=\tilde{k}(\mathbf{x}_i,\mathbf{x}_j)$. It was first shown in Schölkopf et al. (1998) that $\tilde{\mathbf{K}}$ can be calculated from $\mathbf{K}$ with the following lemma.

Lemma 2.4.1 The centred kernel matrix, $\tilde{\mathbf{K}}$ is given by

$$\tilde{\mathbf{K}}=\mathbf{K}-\frac{1}{n}\mathbf{1}_n\mathbf{K}-\frac{1}{n}\mathbf{K}\mathbf{1}_n+\frac{1}{n^2}(\mathbf{1}_n\mathbf{K})\mathbf{1}_n,$$

where $\mathbf{1}_n$ is a $n\times n$ matrix containing ones.

Let the eigenvectors of $\tilde{\mathbf{K}}$ be given by $\boldsymbol{\alpha}_j \in \mathbb{R}^n$ $(j = 1, \dots n)$, with corresponding eigenvalues $\tilde{\lambda}_j$. Now, we can use the following theorem, from Schölkopf et al. (1998).

Theorem 2.4.2 (Schölkopf et al. (1998)) The eigenvector associated with the j-th largest eigenvalue of $\mathbf{C}_\Phi$, $\mathbf{e}_j \in \mathbb{R}^F$ is given by

$$\mathbf{e}_j = \frac{1}{\sqrt{\tilde{\lambda}_j}} \sum_{i=1}^{n} \alpha_j^i \tilde{\Phi}(\mathbf{x}_i),$$

where α_j^i is the i-th element of the vector $\boldsymbol{\alpha}_j \in \mathbb{R}^n$, the eigenvector associated with $\tilde{\lambda}_j$, the j-th largest eigenvalue of $\tilde{\mathbf{K}}$. Furthermore,

$$\lambda_i = \frac{\tilde{\lambda}_i}{n}.$$

Note that we cannot calculate $\mathbf{e}_j$ directly, since $\tilde{\Phi}(\mathbf{x}_i)$ is unknown. However, we can calculate $\mathbf{e}_j^\top \tilde{\Phi}(\mathbf{x}_k) \propto \sum \alpha_j^i \tilde{\Phi}(\mathbf{x}_i)^\top \tilde{\Phi}(\mathbf{x}_k) = \sum \alpha_j^i \tilde{k}(\mathbf{x}_i, \mathbf{x}_k)$ for example, as the dot-product can be calculated using the centred kernel function in Lemma 2.4.1. And, as will be shown in the next section, this allows us to calculate orthogonal regression in the feature space.

2.4.2 kLGA with Orthogonal Regression

In any clustering paradigm, there is the concept of a build set of observations (in this case used to form the hyperplanes) and then a subsequent step of calculating the distance of all observations to this hyperplane. For convenience, we introduce notation that is more consistent with this approach.

Let S denote the indices of the build set of observations with cardinality $m = |S|$. The kernel matrix of the observations in S is given by $\mathbf{K}^S \in \mathbb{R}^{m \times m}$ with elements $k(\mathbf{x}_i, \mathbf{x}_j)$ $(i, j \in S)$, and applying Lemma 2.4.1 to $\mathbf{K}^S$ yields its corresponding centred kernel matrix $\tilde{\mathbf{K}}^S$. Finally, let $S[i]$ denote the i-th element of S, and

$$\mathbf{k}_j^S = \begin{pmatrix} (\mathbf{K})_{S[1],\,j} \\ (\mathbf{K})_{S[2],\,j} \\ \vdots \\ (\mathbf{K})_{S[m],\,j} \end{pmatrix} \in \mathbb{R}^m,$$

where $(\mathbf{K})_{a,\,b}$ denotes the $(a,\, b)$ element of the kernel matrix $\mathbf{K}$. Let a matrix containing these vectors for all observations be given by

$$\mathbf{K}^F = \left(\mathbf{k}_1^S \vdots \mathbf{k}_2^S \vdots \cdots \vdots \mathbf{k}_n^S \right) \in \mathbb{R}^{m \times n}.$$

We now have the building blocks for performing orthogonal regression in the feature space. Consider a build set (S) of points with m observations in the feature space whose covariance matrix has rank $(F - 1)$ or F where F is the dimension in the feature space. In order to calculate the orthogonal distance in the feature space, we combine the results from Result 2.2.1 (Orthogonal Regression) and Theorem 2.4.2 (kPCA), to get the following distance metric.

Result 2.4.3 The orthogonal distance of observation $\mathbf{y} = \Phi(\mathbf{x})$ to a hyperplane orthogonal to $\mathbf{e}_k$ (with $\lambda_k > 0$ and $k \leq F$) and built on observations with indices in the set S, is given by

$$d_O^2(\mathbf{y}, \mathcal{H}) = (\mathbf{e}_k \cdot \mathbf{y} - \mathbf{e}_k \cdot \bar{\mathbf{y}}_s)^2, \tag{2.5}$$

where

$$\mathbf{e}_k \cdot \bar{\mathbf{y}}_s = \frac{1}{m^2\sqrt{\tilde{\lambda}_k}}\, \boldsymbol{\alpha}_k^\top \left(m\mathbf{K}^S\mathbf{1} - \mathbf{1}^\top\mathbf{K}^S\mathbf{1} \right), \tag{2.6}$$

$$\mathbf{e}_k \cdot \mathbf{y}_i = \frac{1}{\sqrt{\tilde{\lambda}_k}}\, \boldsymbol{\alpha}_k^\top (\mathbf{k}_i^S - \frac{1}{m}\mathbf{1}^\top\mathbf{k}_i^S), \tag{2.7}$$

$\bar{\mathbf{y}}_s$ is the sample mean of the observations in S, $\boldsymbol{\alpha}_k \in \mathbb{R}^n$ is the eigenvector associated with $\tilde{\lambda}_k$, the k-th largest eigenvalue of $\tilde{\mathbf{K}}^S$, and $\mathbf{1}$ is a vector of length m containing all ones.

When the observation $\mathbf{y}_i = \Phi(\mathbf{x}_i)$ in the lemma above is part of the build set, equation (2.5) simplifies to

$$(\mathbf{e}_k \cdot \mathbf{y}_i - \mathbf{e}_k \cdot \bar{\mathbf{y}})^2 = \left(\sum_{j=1}^{m} \alpha_k^j \tilde{K}_{ji}^S \right)^2$$

from Lemma 2.4.1. In order to make this result easier to implement for coding, we can express Result 2.4.3 in such a way that it can be calculated for a matrix $\mathbf{Y}$ (each row being an observation) with the following equation:

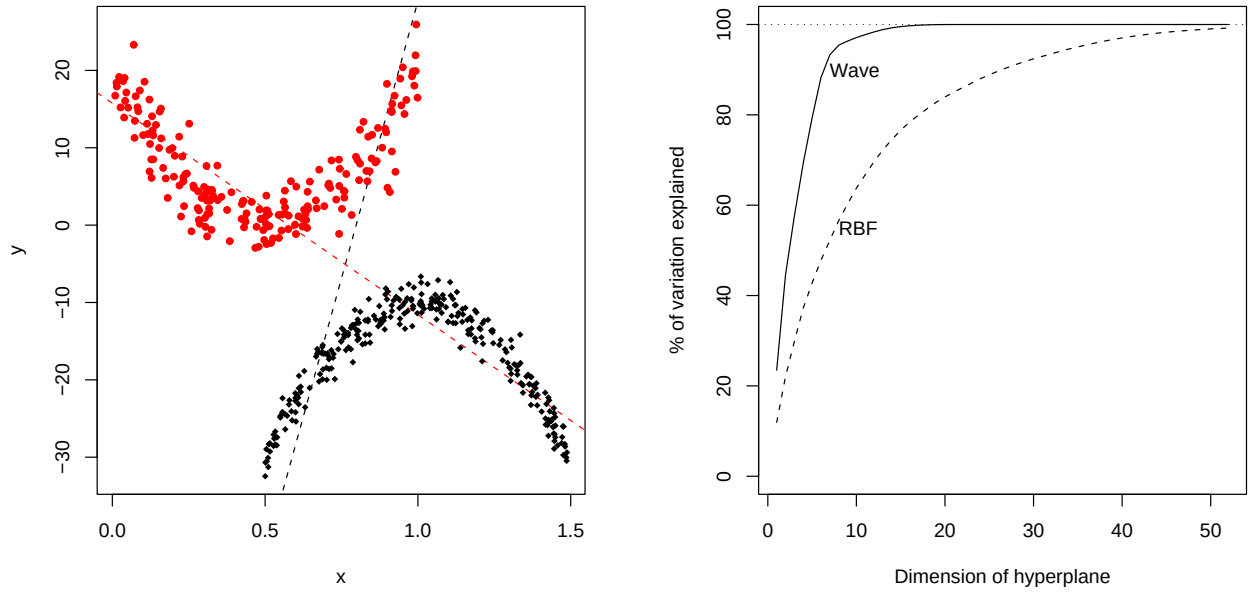
$$d_O^2(\mathbf{Y}, \mathcal{H}) = \begin{pmatrix} d_O^2(\mathbf{y}_1, \mathcal{H}) \\ d_O^2(\mathbf{y}_2, \mathcal{H}) \\ \vdots \\ d_O^2(\mathbf{y}_n, \mathcal{H}) \end{pmatrix} = \frac{1}{\tilde{\lambda}_k} \left((\mathbf{K}^F - \frac{1}{m}\mathbf{1}_m\mathbf{K}^F)^\top \boldsymbol{\alpha}_k - b \right)^2,$$

where $\mathbf{1}_m$ is a $m \times m$ matrix containing all ones, and

$$b = \frac{1}{m^2}\, \boldsymbol{\alpha_k}^\top \left(m\mathbf{K}^S\mathbf{1}_m - \mathbf{1}_m^\top\mathbf{K}^S\mathbf{1}_m \right).$$

This derivation raises some interesting technicalities regarding rank. It was assumed (explicitly and implicitly) in this section that the covariance matrix $\mathbf{C}_\Phi \in \mathbb{R}^{F\times F}$ has rank F or $(F-1)$ in the feature space, the dimension of which is defined by the transformation Φ. However, in the calculation of the eigenvectors of $\mathbf{C}_\Phi$, we use the spectral decomposition of $\tilde{\mathbf{K}}^S \in \mathbb{R}^{m\times m}$, where m may be larger or smaller than F. This means that the number of positive (non-zero) eigenvalues of $\tilde{\mathbf{K}}^S$ determines uniquely the approach that is required.

In practise, if the transformation Φ is explicitly specified, then the dimension of the feature space can be determined, and it will be known *ex ante* whether the covariance matrix in the feature space is going to have rank F or $(F-1)$ when the number of observations in the build set is taken into consideration. However, if an implicit transformation is used, more care is required. In particular, some kernel functions, such as the Radial Basis Function (RBF), transform the data into an infinite dimensional space, and therefore no sample covariance matrix based on a finite sample will be full



(a) kLGA with an explicit transformation

(b) Eigenvalue analysis of Wave and RBF kernels

Figure 2.6: Plots from example one for kLGA. On the left we have the output from kLGA, with and explicit transformation $\Phi(\mathbf{x}) = (\mathbf{x}_1, \mathbf{x}_1^2, \mathbf{x}_2)$. Also on this plot is the hyperplanes that the (non-feature space) LGA would fit. On the right we look at the percentage of total variation explained, based on the eigenvalues, for the first cluster using two different kernel functions.

rank. Therefore, prior to using Result 2.4.3, careful analysis of the eigenvalues of $\tilde{\mathbf{K}}^S$ should be undertaken.

Example. To introduce kLGA with orthogonal regression, we first use a simple example with an explicit transformation function. For the first cluster, we have 200 independent random bivariate samples from the model $X \sim \text{Unif}(0,1)$, and $Y = 80(X-\frac{1}{2})^2+\epsilon$ where $\epsilon \sim \mathcal{N}(0, 2.8^2)$. The second cluster has 300 independent random bivariate samples from $X \sim \text{Unif}(\frac{1}{2}, \frac{3}{2})$, and $Y = -80(X-1)^2 - 10 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1.5^2)$. The transformation function is given by $\Phi(\mathbf{x}) = (x,\, x^2,\, y)$.

The results from kLGA are given in Figure 2.6(a). As a matter of reference, we also provide an analysis of the eigenvalues for two other kernel functions, being the Wave Kernel and the RBF kernel, given by

$$\text{Wave:} \qquad k(\mathbf{x}, \mathbf{z}) = \begin{cases} \frac{\theta}{||\mathbf{x}-\mathbf{z}||} \sin\left(\frac{||\mathbf{x}-\mathbf{z}||}{\theta}\right) & \text{if } ||\mathbf{x}-\mathbf{z}|| > 0 \\ 1 & \text{if } ||\mathbf{x}-\mathbf{z}|| = 0 \end{cases} \tag{2.8}$$

$$\text{RBF} \qquad k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{z}||^2}{\theta}\right). \tag{2.9}$$

A plot of the total variation explained versus the number of eigenvalues, as given by the equation $\sum_{i=1}^{\nu} \lambda_i / \sum_{j=1}^{n} \lambda_j$, $\nu = 1, \ldots, 50$, is provided in Figure 2.6(b) for each kernel function. Clearly the dimension of transformed data using the implicitly defined transformation is substantially larger than that for our explicitly defined transformation, which has a dimension of three.

2.4.3 kLGA with POD

As mentioned in the previous section, the rank of the covariance matrix in the feature space is as relevant in the feature space as it is in the Euclidean space, which was discussed at length in Section 2.3. As a result, we conclude our foray into the feature space with an extension of the POD methodology for use in the feature space. Recalling Definition 2.3.1, and proceeding along the same lines as Section 2.4.3, we get the following result.

Result 2.4.4 Let there be k eigenvectors, denoted $\mathbf{e}_1, \ldots, \mathbf{e}_k$, with positive (non-zero) eigenvalues. The Partially Orthogonal Distance (POD) of a point $\mathbf{y} = \Phi(\mathbf{x}) \in \mathbb{R}^F$ to the hyperplane with basis $\mathbf{e}_1, \ldots, \mathbf{e}_k$ going through $\bar{\mathbf{y}}_s$ is given by

$$d_P^2(\mathbf{y}, \mathbf{E}, \bar{\mathbf{y}}_s) = \mathbf{z}^\top \mathbf{z} + \frac{1}{\lambda_k} \mathbf{z}^\top \mathbf{e}_k \mathbf{e}_k^\top \mathbf{z} - \mathbf{z}^\top \mathbf{E}\mathbf{E}^\top \mathbf{z},$$

where $\mathbf{z} = \mathbf{y} - \bar{\mathbf{y}}_s$ and

$$\mathbf{E} = \begin{pmatrix} | & | & & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_k \\ | & | & & | \end{pmatrix}.$$

In practise, this equation can be expressed using equations (2.6) and (2.7) for all observations at once, with

$$d_P^2(\mathbf{Y}, \mathcal{H}) = \begin{pmatrix} d_P^2(\mathbf{y}_1, \mathcal{H}) \\ d_P^2(\mathbf{y}_2, \mathcal{H}) \\ \vdots \\ d_P^2(\mathbf{y}_n, \mathcal{H}) \end{pmatrix} = \operatorname{Diag}\left(\tilde{\mathbf{K}}^*\right) + \frac{1}{\tilde{\lambda}_k / m} \left((\mathbf{K}^F - \frac{1}{m}\mathbf{1}_m \mathbf{K}^F - b')^\top \frac{\boldsymbol{\alpha}_k}{\sqrt{\tilde{\lambda}_k}} \right)^2 - \left((\mathbf{K}^F - \frac{1}{m}\mathbf{1}_m \mathbf{K}^F - b')^\top \mathbf{A} \right)^2,$$

where $m = |S|$, $\mathbf{1}_m$ is a $m \times m$ matrix containing all ones,

$$\tilde{\mathbf{K}}^* = \mathbf{K} - \frac{1}{m}\mathbf{1}_m \mathbf{K}^F - \frac{1}{m}\mathbf{K}^F \mathbf{1}_m + (\mathbf{1}_m \mathbf{K}^S)\mathbf{1}_m,$$
$$b' = \frac{1}{m^2} \left(m\mathbf{K}^S \mathbf{1} - \mathbf{1}^\top \mathbf{K}^S \mathbf{1} \right),$$

and

$$\mathbf{A} = \begin{pmatrix} | & | & & | \\ \boldsymbol{\alpha}_1 / \sqrt{\tilde{\lambda}_1} & \boldsymbol{\alpha}_2 / \sqrt{\tilde{\lambda}_2} & \cdots & \boldsymbol{\alpha}_k / \sqrt{\tilde{\lambda}_k} \\ | & | & & | \end{pmatrix}.$$

In order to cluster using this formula, Algorithm 2.3.2 is used, and the parameter *nsamp* is required in order to specify the size of the initial sample. Recall that in the Euclidean space (which is the

context of Section 2.3), *nsamp* should be the smallest value such that the relevant dimensions are included, where the relevant dimensions can be defined as the number of positive eigenvalues determined in equation (2.4). However, in the feature space, there are kernel functions that imply a dimension that is infinitely large, which makes this approach more difficult. In this case, the size of *nsamp* given in equation (2.4) may need to be very large in order to capture the necessary information (see, for example, Figure 2.6(b) for an indication of this with the RBF kernel), which makes the combinatorial problem extremely difficult. In the examples that follow, we have selected a smaller value for ν in the interests of an easier optimization step, and leave the detailed study of this problem to further research.

Example. In this example we generate two bivariate clusters. In the first cluster, $X \sim$Unif$(-\pi, \pi)$ and $Y = \sin(X)/4 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$, while in the second cluster $X \sim$Unif$(-\pi, \pi)$ and $Y = 5\sin(X) + \epsilon$, with ϵ defined the same. Each cluster contains 100 independent observations following the above models.

Using the wave kernel function given in equation (2.8), with $\theta = 10$ we run the kLGA(POD) algorithm, with the results given in Figure 2.7(a).

Example. In our second example, we form two clusters with the following characteristics:

Cluster 1 - 400 independent observations from the model: $X \sim$Unif$(-\pi, 2\pi)$ and $Y = 5\sin(X) + 15 + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.25)$

Cluster 2 - A combination of two models.

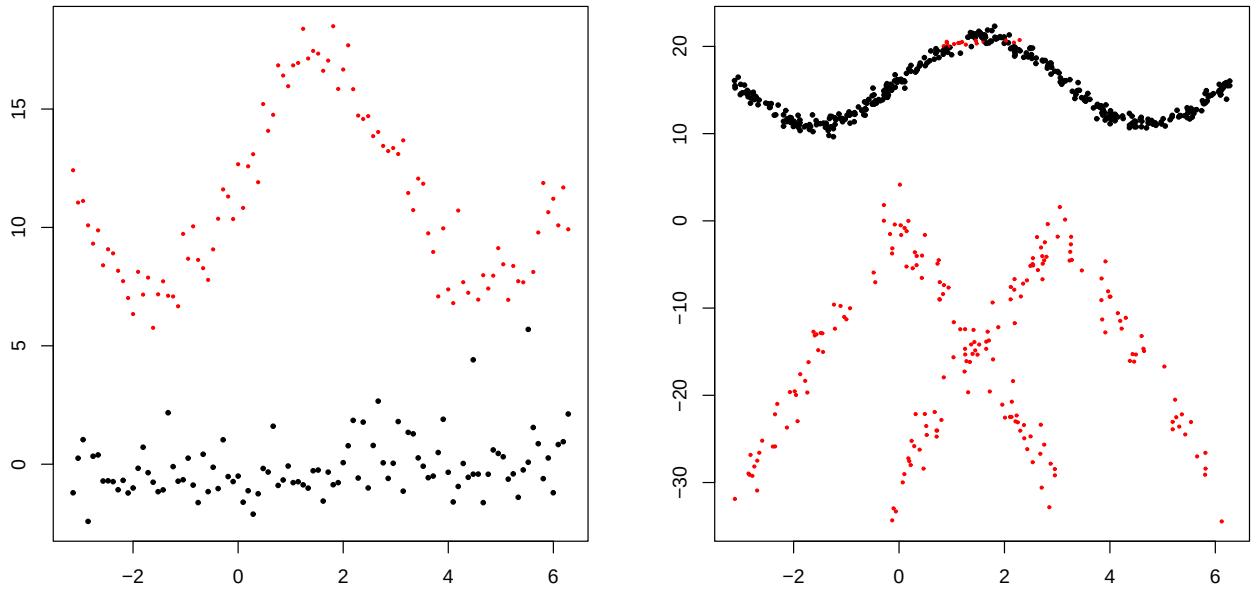
- 100 independent observations from the model: $X \sim$Unif$(-\pi, \pi)$ and $Y = -10|X| + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 4)$.
- 100 independent observations from the model: $X \sim$Unif$(3 - \pi, 3 + \pi)$ and $Y = -10|X - 3| + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 4)$.

We use kLGA(POD) with the RBF Kernel Function (equation (2.9)). The results are given in Figure 2.7(b). Note that there is some misclassification, however this accounts for less than 1% of the whole data set.

2.5 Discussion and Future Research

In this article we have introduced two significant extensions to the LGA algorithm. The first extends LGA to allow the covariance matrix of any cluster to have rank less than $(d-1)$ by introducing a new distance called Partially Orthogonal Regression. The second method extends LGA to the feature space by using the kernel trick, allowing for clustering non-linear patterns and is demonstrated for both cases where covariance matrices have rank F or $(F-1)$, and the rank is less than $(F-1)$.

Much of the discussion was devoted to the number of positive eigenvalues within the feature space, and how it is of crucial importance. Further research in this area is necessary, in particular the relationship between the number of eigenvalues and the number of the starting samples for initializing the algorithms (given by the parameter *nsamp*). In the methods given here we explicitly check that the nature of the eigenvalues is consistent with the algorithm chosen, but have not currently quantified the effects of this approach.



(a) kLGA(POD) with Wave Kernel

(b) kLGA(POD) with RBF Kernel

Figure 2.7: Plots from the examples for kLGA (POD). On the left we have the output from kLGA (POD) using the wave kernel with $\theta = 10$. On the right we have the output from kLGA (POD) using the RBF kernel with $\theta = 200$.

It is clear that further research is required into the selection of the kernel functions, and tuning the parameters of these functions, though this comment can be applied more generally to the area of feature space methodologies as a whole. However, in the area of unsupervised learning, of which these algorithms are part, this is especially difficult since the option of first training the algorithm using a known solution is not relevant. Methods have been proposed elsewhere, such as cross validation, and this would be a logical place to start.

With respect to selection of kernel functions and LGA more specifically, research is required into the nature of the topology of the spaces generated with implicit transformations, and how they relate to the underlying assumptions regarding linear clusters in these spaces. It may be possible to determine in which cases LGA is sensible, and in which feature spaces LGA makes no sense.

One avenue of research that has already proved fruitful, though is not yet at maturity, is the use of the gap statistic (Tibshirani et al., 2001) for the selection of the polynomial degree when using a polynomial kernel. The gap statistic is useful for examining nested models in order to discover the "elbow" in residual sum of squares, and has been used successfully for finding the number of clusters in the context of k-means and LGA. In this case, it is intuitively consistent that, as the degree of the polynomial increases, the ROSS decreases slowly until the "true" degree is attained, at which point it decreases dramatically. Once the degree exceeds the "true" value, once again the ROSS decreases slowly. Since the gap statistic is tuned to find such "elbows", it seems to work well, though further work is required.

Bibliography

A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

W. C. Chang. On using principal components before separating a mixture of two multivariate normal distributions. *Applied Statistics*, 32(3):267–275, 1983.

G. Crile and D. Quiring. *A Record of the Body Weight and Certain Organ and Gland Weights of 3690 Animals*. Ohio Journal of Science, 1940.

M. Girolami. Mercer Kernel Based Clustering in Feature Space. *IEEE Transactions on Neural Networks*, 13(3):780–784, 2002.

I. Jollife, B. Jones, and B. Morgan. Utilising clusters: A case-study involving the elderly. *Journal of Royal Statistical Society. Series A*, 145(2):224–236, 1982.

I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 14, 2001.

B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

A. Smola, O. L. Mangasarian, and B. Schölkopf. Sparse kernel feature analysis. In *Classification, Automation, and New Media*. Springer, 2002.

J. A. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Square Support Vector Machines*. World Scientific, 2002.

R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *J. R. Statist Soc. B*, 63(2):411–423, 2001.

S. Van Aelst, X. Wang, R. H. Zamar, and R. Zhu. Linear Grouping Using Orthogonal Regression. *Computational Statistics & Data Analysis*, 50(5):1287–1312, Mar. 2006.

K. Yeung. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17 (9):763–774, 2001.

R. H. Zamar. Robust estimation in the errors-in-variables model. *Biometrika*, 76(1):149–160, 1989.

Chapter 3

Finding Approximate Solutions to Combinatorial Problems with Very Large Data Sets using BIRCH*

3.1 Introduction

A very large data set is traditionally defined as one that exceeds the size of the main memory on a computer. Once this size is exceeded, paging/swapping is required to access the data which is very inefficient and time consuming.

In this paper we focus on the problem of finding approximate solutions to non-convex optimization problems, particularly those that arise when computing high-breakdown point robust estimators, such as the Minimum Covariance Determinant and the Least Trimmed Squares estimators (Rousseeuw and Leroy, 1987). The optima of these problems are generally found by using some variation of random search algorithms. Recently improved methods based on this approach have been shown to perform better than other heuristic algorithms, like simulated annealing and tabu search (see Salibian-Barrera et al. (2007)). The basic idea of the random subsampling algorithm (Rousseeuw, 1984) is to generate a candidate solution by randomly drawing a subsample from the data. This candidate is then refined by performing local improvements in the objective function to find a local minimum near this starting point. In the robust statistics context this strategy is expected to find a good approximation to the optimum when the random subsample of the data is drawn from the "good" observations (non outliers). In general, a very large number of random starts are needed to ensure the solution space is well represented among these candidates, and furthermore, each starting point requires many refinement iterations.

To fix ideas, consider the MCD estimator, which for a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \in \mathbb{R}^p$ and an integer $n/2 \leq h \leq n$, is defined as the sample mean and covariance matrix of the subsample of size h with sample covariance matrix with the smallest determinant. This is a combinatorial problem (there are $n!/(h!(n-h)!)$ possible subsamples of size h) whose solution can be approximated using the fast-MCD algorithm (Rousseeuw and van Driessen, 1999). This approach randomly draws a large number (N) of subsamples of size $p+1$, which are then enlarged and iteratively refined by selecting points that are close (in the Mahalanobis distance sense) to the starting values. We see that this algorithm accesses the whole data set a very large number of times. When the data set is large (i.e. it is not possible to store it in memory), input/output delays due to paging or swapping make the use of these methods unfeasible.

*A version of this chapter is in the review process for publication. Harrington, J. and Salibian-Barrera, M. (2007) Finding Approximate Solutions to Combinatorial Problems with Very Large Data Sets using BIRCH, "Special Issue on Statistical Algorithms and Software" of Computational Statistics and Data Analysis.

We adopt the following essential criteria for an algorithm to be successful when applied to very large data sets:

- it should only require one read through the data set, after which the data set can be discarded;
- it should only require one line of data to be held in memory at any point in time; and
- it should reduce the effective size of the solution space for combinatorial problems.

We will show that the BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm can be adapted appropriately to find approximate solutions to optimization problems such as those mentioned above. These methods satisfy the conditions listed above. In this paper we illustrate this approach by applying BIRCH to find approximate solutions to the optimization problems that define the MCD and the LTS (Least Trimmed of Squares) (Rousseeuw and Leroy, 1987). Finally, we also adapt BIRCH to the linear grouping algorithm (LGA) (Van Aelst et al., 2006) and its robust version (RLGA) (García-Escudero et al., 2007).

Our presentation will be organized as follows. Section 3.2 briefly describes the BIRCH algorithm. In Section 3.3 our approach is applied to compute approximations to the MCD and LTS estimators, and to run the RLGA algorithm, with very large data sets. We show that the algorithm we propose works very well (compared with the state-of-the-art alternatives) when the data set fits in memory. Moreover, our numerical experiments show that, in this case, our algorithm can be used to very quickly compute a good starting point that when iterated using the whole data set converges to the same solution found by the best algorithms available. Section 3.4 reports the results of two simulation studies that indicate that this new method also works very well in large data sets. Finally, some concluding remarks are included in Section 3.5.

3.2 Data Pre-processing

The data pre-processing algorithm BIRCH was first introduced in Zhang et al. (1996). It groups the data set into compact subclusters that have summary statistics (called Clustering Features (CF)) associated to each of them. These CFs are computed and updated as the subclusters are being constructed. The end result is an "in-memory" summary of the data, where "local" compact subclusters are represented by appropriate summary statistics.

This approach lends itself to robust statistics and clustering, in that both methodologies are concerned with finding areas of homogeneous data that are locally similar and dense. As such, it is a simple extension to consider using compact subclusters instead of individual observations to obtain a fast and good approximation to the solution of the corresponding optimization problems. It is important to note that by choosing the summary statistics appropriately we can evaluate the objective function exactly. In other words, the approximated nature of this approach is only due to the fact that both the random starts and the subsequent iterative refining steps use sets of points (subclusters) instead of individual observations. Further discussion of the influence of this approximation is given in Section 3.2.2.

The advantage of pre-processing the data is that, once completed, the data set can be discarded, and the number of subclusters is much reduced relative to the number of observations, thereby making the combinatorial problem smaller.

It will be shown in Section 3.2.1 that at no point in time is more than one observation required to be held in memory, and combined with the previous points, this means we have met our essential criteria mentioned in Section 3.1.

In the original BIRCH article, the method proposed adapts a global clustering algorithm e.g. CLARANS (Ng and Han, 2002), and selects the clustering for the desired number of groups. These are then used as seeds for an optional cluster refinement, though it is unclear if this is based on the complete data set, or performs further steps using just the subclusters. An extension to k-means on mixed type attributes is given in Chiu et al. (2001). Neither of these articles explicitly considers the application of BIRCH from the perspective of a combinatorial problem and the reduction of the solution space.

Since the introduction of BIRCH, other methods for the formation of tree-like structures and their application have been explored. In Breunig et al. (2001), BIRCH is used as an initial step to "compress" the data set into representative objects. However, in the context of hierarchical clustering, the concern is that the size of each subcluster (as given by its number of observations) is not considered. As such, a post-processing step is proposed, involving going back to the original data set and sampling a number of objects, with the number proportion to the size of the subcluster. These points are subsequently used for clustering.

Other improvements are offered in Nassar et al. (2004) and Tung et al. (2005) regarding how the tree is created. In the first article, the BIRCH algorithm is optimized for data sets with "real-time" data streams by adding an attribute to subclusters that have been amended since the first construction. This allows for incremental updating of only those subclusters that have recently admitted data without needing to rebuild an entire tree. Finally, in the latter article, a dual criteria of spatial proximity as well as orientation is considered in the merging of subclusters; the combination of these attributes would make for a more compact merged subcluster.

The idea of using summary statistics of subsets of the data is also explored in other ways; for example in spatial statistics via grid based methods (e.g. Wang et al., 1998). It was also noted in this article that care must be taken when using BIRCH that the columns are of similar scale when the subclusters are being formed, else the interpretation of Euclidean and Mahalanobis distance becomes difficult, a fact that is implemented in the following algorithms. In the case where the data set does not fit in memory, an approach for scaling is proposed in the next section.

3.2.1 Forming the Subclusters

A simplified algorithm for forming the subclusters is given next. Further details of the complete algorithm, including refinements such as self-sizing, subcluster splits and CF-Trees, can be found in Zhang et al. (1996).

Let $\mathbf{X}_1, \ldots, \mathbf{X}_N$ denote the sample to be processed, where $\mathbf{X}_j \in \mathbb{R}^p$.

Pre-processing:

1. The first observation $\mathbf{X}_1$ forms its own subcluster.
2. Let $\mathcal{T}$ be the set of current subclusters, and let $\bar{\mathbf{X}}_t$, and CF_t, $t \in \mathcal{T}$ be the associated subcluster centres and clustering features (currently left unspecified).
3. For each of the remaining observations $\mathbf{X}_2, \ldots, \mathbf{X}_N$:

(a) Calculate the L_2-norm distance from $\mathbf{X}_j$ to each of the centres $\bar{\mathbf{X}}_t$ of the existing subclusters. Select the closest centre $\bar{\mathbf{X}}_{t_0}$.

(b) If the observation meets the "closeness" and "compactness" criteria described below, then $\mathbf{X}_j$ joins the t_0 subcluster, and both $\bar{\mathbf{X}}_{t_0}$ and CF_{t_0} are updated.

(c) If not, a new subcluster $t+1$ containing $\mathbf{X}_j$ is formed, with $\bar{\mathbf{X}}_{t+1} = \mathbf{X}_j$ and CF_{t+1} the corresponding clustering features, and the list of subclusters is updated: $\mathcal{T} = \mathcal{T} \cup \{\mathbf{X}_j\}$

The clustering features we consider are functions of the subcluster members that are easily updated when a new observation joins (e.g. the sum of the elements of the subcluster). It is clear from the above that only a single observation is held in memory at any point in time.

Note that this algorithm requires the calculation of the distances between $\mathbf{X}_j$ and every subcluster centre. This can be avoided by using a more efficient structure, such as a CF-tree which is similar to a "B+"-tree (Wedekind, 1974). In this approach, each observation is passed to a root node, with a number of non-leaf nodes as children. Each non-leaf node contains the sample mean of all observations beneath it. The observation is then simply passed down this tree to the closest non-leaf node based on L_2-norm distance to the centres, and each centre is updated as the observation is given to it. Finally, when it reaches the appropriate subcluster, it is either absorbed, or else forms its own subcluster. Once a non-leaf node exceeds a (user-specified) number of subclusters, it splits with the subclusters allocated to the split non-leaf nodes based on a proximity measure. A more complete description of this structure is given in Appendix B, which is also available as Harrington and Salibian-Barrera (2008).

The number of subclusters that are formed at the end of this algorithm depends on the criterion used to decide whether a new observation being processed can join an existing subcluster or needs to form a new cluster. Consider a point $\mathbf{X}_j$ currently being processed, and let $\bar{\mathbf{X}}_{t_0}$ be the closest subcluster centre. In order for $\mathbf{X}_j$ to join the t_0 cluster, Zhang et al. (1996) required either of the following criteria:

Parameters:

(i) *closeness* – $\|\mathbf{X}_j - \bar{\mathbf{X}}_{t_0}\| \leq a$ for some pre-defined $a > 0$.

(ii) *compactness* – The trace of the sample covariance matrix of the union of observations in the t_0 cluster *and* $\mathbf{X}_j$ cannot exceed a pre-defined constant $b > 0$.

Although other measures of compactness, e.g. the generalized variance, can be used in (ii), we followed the original BIRCH specification by using the trace. In our algorithm we use both jointly, as we found that any one of these criteria alone was not enough to form subclusters that summarize the data well. The disadvantage of using only the "compactness" criterion is that, when the number of observations within a subcluster is large, a point could be far from the centre of the subcluster, and yet adding it to the subcluster would not noticeably affect the sample covariance matrix of the members of the subcluster. In other words, it may allow for "scatter" outside the otherwise compact body of points within the subcluster.

The disadvantage of only using the "closeness" criterion is apparent when there are few points in a subcluster. Consider an extreme example of three observations on a line. The first two observations have the sample mean exactly bisecting them. When the third point is added, the sample mean of the new subcluster would shift towards the new point, increasing the distance of the first observation to the new centre. As more points are added on this line, this shifted centre would admit more

points as they meet the closeness criteria, whilst moving the initial points further away. We refer to this effect as "travelling". The subcluster centre starts shifting and the subcluster becomes unduly "long". In other words, the compactness of the subcluster is reduced. The combination of these criteria, therefore, ensures that the resulting subclusters are compact.

As mentioned in the proceeding section, the data set should be scaled by dividing each column by its standard deviation prior to pre-processing. This necessitates the calculation of the standard deviations, which can be done either directly if the data set is sufficiently small, or else by using a very large value for the closeness and compactness criteria, yielding a very small tree from which the calculation of these statistics is trivial. The algorithm is then re-run on the same data set with each observation scaled as it is processed. It should be noted that a robust scale estimate would be preferable for scaling the data set; however there is no "updating formula" for such estimators with high breakdown point, and therefore cannot be applied within the BIRCH framework.

3.2.2 Selection of Parameters

The selection of the closeness and compactness parameters is a trade-off between (a) granularity of the data sets and (b) reduction of the effective solution space. Clearly as these parameters increase, the number of subclusters decreases and the "size" of the subclusters increases. The impact on the solution to combinatorial problems is that the solution is "coarser" in that the subclusters reflect less the subtleties of the underlying data set. Therefore, increasing these parameters would yield solutions further away from that found using the original data set, and vice versa decreasing the parameters would give a solution more closely approximating the solution based on the original data set.

However, by decreasing the parameters, the solution space is enlarged due to the increased number of subclusters, which makes for a more difficult combinatorial problem, to the point where any potential gains in optimality of the solution might be lost. Also, an increased number of subclusters requires more memory, which may be a limiting factor.

Clearly, if a refinement step following this algorithm (as suggested in Zhang et al. (1996)) were to take place, then a "coarse" solution would be sufficient. If, however, this were not to be practical because the size of the data set excludes this option, then the selection of the parameters is of more significance. The question of setting of parameters is explored further in the following sections, where the results of our experiments suggest that the optimality of the solution is only slightly affected by the selection of the parameters, and it is irrelevant if refinement takes place.

3.3 Applications

In this section we show how to adapt this approach to compute two robust estimators (the MCD for multivariate location and scatter, the LTS for linear regression) and a robust clustering algorithm: the Robust Linear Grouping Analysis. We will show that the approximate solutions obtained by our approach compare very well with those given by current algorithms.

Consider the CF given by the following tuple:

$$CF_J = \left(\sum_{i \in \mathcal{L}_J} 1, \sum_{i \in \mathcal{L}_J} \mathbf{X}_i, \sum_{i \in \mathcal{L}_J} \mathbf{X}_i \mathbf{X}_i^\top \right) = (n_J, LS_J, SS_J);$$

i.e. the number of observations, the sum of those observations, and the sum of squares of those observations. The set $\mathcal{L}_J$ contains the members of subcluster J. Let N be the number of subclusters, and for all $i \neq j \in \{1, \ldots, N\}$, $\mathcal{L}_i \cap \mathcal{L}_j = \varnothing$ by construction. We can define the addition operator in a natural way:

$$\begin{aligned} CF_I + CF_J &\stackrel{\text{def}}{=} (n_I + n_J, LS_I + LS_J, SS_I + SS_J) \\ &\equiv \left(\sum_{i \in \{\mathcal{L}_I \cup \mathcal{L}_J\}} 1, \sum_{i \in \{\mathcal{L}_I \cup \mathcal{L}_J\}} \mathbf{X}_i, \sum_{i \in \{\mathcal{L}_I \cup \mathcal{L}_J\}} \mathbf{X}_i \mathbf{X}_i^\top \right), \end{aligned} \tag{3.1}$$

which equates exactly to the features associated with the union of the member vectors $\mathcal{L}_I \cup \mathcal{L}_J$. We call these statistics "sufficient", insofar as the evaluation of the objective functions we are interested in on the observations of a subcluster (or a union of subclusters) can be done exactly using only the CF's. Thus, as mentioned before, we do not need to keep the data set in memory.

3.3.1 Minimum Covariance Determinant

The Minimum Covariance Determinant (MCD) (Rousseeuw, 1984, p 877) is a robust measure of location and dispersion. It is calculated by finding the subset of data of size $h = \alpha n$, $\alpha \in (\frac{1}{2}, 1]$ whose sample covariance matrix has smallest determinant i.e. if $\mathcal{H}$ is the solution space of all possible subsets of size h, then the MCD is

$$\arg\min_{J \in \mathcal{H}} |S_J|$$

where

$$S_J = \frac{1}{h-1} \sum_{i \in J} (\mathbf{X}_i - \bar{\mathbf{X}}_J)(\mathbf{X}_i - \bar{\mathbf{X}}_J)^\top \qquad \text{and} \qquad \bar{\mathbf{X}}_J = \frac{1}{h} \sum_{i \in J} \mathbf{X}_i$$

The location and dispersion estimate is then given by the sample mean and sample covariance of this subset.

We compared the performance of our algorithm with the fast-MCD (Rousseeuw and van Driessen, 1999), which randomly selects $(p+1)$ sized subsets of data, calculates its sample mean ($\hat{\boldsymbol{\mu}}$) and covariance matrix ($\hat{\mathbf{S}}$), and then updates them using the following "concentration" steps.

Concentration Step:

- Let $H \leftarrow \{$ the h points with smallest Mahalanobis distances: $(\mathbf{X} - \hat{\boldsymbol{\mu}})^\top \mathbf{S}^{-1} (\mathbf{X} - \hat{\boldsymbol{\mu}})\}$
- Calculate the new $\hat{\boldsymbol{\mu}}$, $\mathbf{S}$ based on the observations in H.

It can be shown that these concentration steps always decrease the objective function above, and are iterated until convergence or until some upper bound on iterations has occurred. The outputs from this algorithm are estimates for $\hat{\boldsymbol{\mu}}$, $\mathbf{S}$ and a vector of the points that make up the "best" h-subset.

As this is a local search algorithm, a large number of random starting configurations is required in order to adequately cover the solution space. It is easy to show that this number grows very rapidly as a function of the dimension p and number of observations of the data.

Our new algorithm, MCD-BIRCH, uses the subclusters derived in Section 3.2, and then attempts to find the optimal size-h set H using the following algorithm.

MCD-BIRCH:

1. Run the pre-processing algorithm.
2. For each subcluster, calculate its covariance matrix, and the corresponding rank. Retain an index of those subclusters whose covariance are of full rank.
3. Randomly select a sufficient number of subclusters of full rank to adequately cover the solution space. For each of these subclusters (denoted by H):
   (a) Calculate the Mahalanobis distance of all remaining subcluster centres to the centre of H, using its centre ($\hat{\boldsymbol{\mu}}_H$) and covariance matrix ($\hat{\mathbf{S}}_H$) (which are computed using the CF's of H).
   (b) Order the distances, and select the smallest number of subclusters with closest centres such that the sum of the number of observations in the union of these subclusters exceeds $h = \alpha n$.
   (c) Calculate $\hat{\boldsymbol{\mu}}_H$ and $\hat{\mathbf{S}}_H$ using the combined CF of this union of subclusters (from equation (3.1)) to form a new seed.
   (d) Let H be the union of these subclusters, and update its CF.
   (e) Iterate from (a) until convergence or until a fixed number of iterations is exceeded.
4. Select the configuration with smallest determinant of the combined sample covariance matrix.

Remark. Note that in Step 2 we only consider "full rank clusters" because we need to compute the Mahalanobis distances (which requires the inverse of the covariance matrix). This restriction is not of much concern because clusters that are not full rank may not be representative of the shape as they usually contain very few observations.

Clearly, the merged CF can be considered sufficient statistics for the calculation of the MCD objective function. Consider a set of CFs $\mathcal{A}$ with complete membership vector $\mathcal{L} = \cup_{i\in\mathcal{A}}\mathcal{L}_i$. Then

$$\begin{aligned} S_{\mathcal{A}} &= \frac{1}{n_{\mathcal{A}} - 1} SS_{\mathcal{A}} - \frac{1}{n_{\mathcal{A}}^2} LS_{\mathcal{A}} LS_{\mathcal{A}}^\top \\ &\equiv \frac{1}{|\mathcal{L}| - 1} \sum_{i\in\mathcal{L}} (X_i - \bar{X}_{\mathcal{L}})(X_i - \bar{X}_{\mathcal{L}})^\top \end{aligned}$$

This is also a local search algorithm, but rather than searching over a solution space of dimension $\binom{n}{h}$, the search is over a much smaller (but coarser) solution space. However, despite this coarseness, the following examples will show that the approximation appears to be very good. Furthermore, if the data set can fit in memory, concentration steps using the individual observations can be applied to the resulting $\hat{\boldsymbol{\mu}}$ and $\mathbf{S}$ estimates until convergence, as described for fast-MCD.

Example. The Digitized Palomar Sky Survey (DPOSS) data has been used in articles related to robustness (e.g. Rousseeuw and van Driessen (1999), Rousseeuw and van Driessen (2006), etc) as

well as clustering (e.g. Rocke and Dai (2003)). It contains characteristics from celestial objects, and has 132,402 observations in 33 dimensions. For further information see Odewahn et al. (1998).

In this example, we run the fast-MCD algorithm and the MCD-BIRCH algorithm on the "Aperture" and "Combined stellar function" columns for each of the F, J and N bands, yielding $p = 6$. The fast-MCD algorithm is implemented in R as `covMcd` from the library `robustbase`[1], and the MCD-BIRCH algorithm as `covMcd.birch` in the library `birch`[2].

Using fast-MCD the log of the determinant of the resulting covariance matrix was -30.71 with the number of random starting subsamples set at 500. This compares favourably with MCD-BIRCH with a log-determinant of -30.69 using just 100 random starting subclusters. Furthermore, the two best $h-$subsets (with $\alpha = 0.51$) have 99.7% of members in common. Finally, performing a refinement by using the solution from MCD-BIRCH as the starting seed for one sequence of concentration steps (to convergence) on the full data set yields a log(MCD) of -30.71, and the best h-subsets have 99.9% of observations in common.

Example. This data was taken during a Nuclear Physics experiment using an accelerated ion beam incident on a hydrogen gas target at the DRAGON facility at TRIUMF, Canada's National Nuclear and Particle Physics Laboratory (Engela et al. (2005) and Hutcheon et al. (2003)).[3] Some of the ions combine with the hydrogen atoms in a nuclear fusion reaction, resulting in an excited product nucleus being formed which 'decays' within a billionth of a second to a stable energy configuration. This decay process involves the emission of one or more high energy photons, gamma rays, which are subsequently detected using a set of sensitive instruments surrounding the gas target. These reactions are measured for the purpose of understanding their role in stars in the field of Nuclear Astrophysics.

Because of the discrete quantum nature of energy levels in an excited nucleus, the gamma rays in question have discrete energies and can occur in various combinations of one another (but all adding up to the same total energy). When more than one gamma ray is detected the values of the two highest energies are retained. Plots of the data set are given in Figures 3.1 & 3.2, and there are 209,037 observations and two dimensions.[4]

In this case, the fast-MCD algorithm achieved a log(MCD) of -7.531 using 500 starting subsamples, and the log(MCD) using MCD-BIRCH was -7.524 using 100 starting subclusters. There was a 98.6% intersection rate between the two best $h-$subsets. Performing an additional refinement on the full data set using the solution from MCD-BIRCH yielded a log(MCD) of -7.532, and has 99.9% of observations in common with fast-MCD. Contours of 95% confidence ellipsoids derived from the asymptotic χ^2 approximation using the classical, fast-MCD and MCD-BIRCH estimates of location and dispersion are given in Figure 3.1, and based on this plot it is clear that the fast-MCD and MCD-BIRCH algorithms are producing near-identical results.

[1] Original code by Rousseeuw and Van Driessen, adapted by Valentin Todorov. R package version 0.2-8.

[2] The package `birch` is available on CRAN, and also contains the algorithms for LTS-BIRCH, RLGA-BIRCH and others. The pre-processing code is written in C++, and the remaining code written in native R. Version 1.1-2 of `birch` was used for this article, and is discussed in detail in Harrington and Salibian-Barrera (2008) (a copy of this article is given in the appendices)

[3] Many thanks to Dr C. Ruiz from TRIUMF for providing this data.

[4] It should be noted that with this example, as well as the ones that follow, the estimates calculated are not necessarily those that would be chosen if "real" analysis (in the statistical sense) were to be done. Rather, the data sets are used for illustration purposes only.

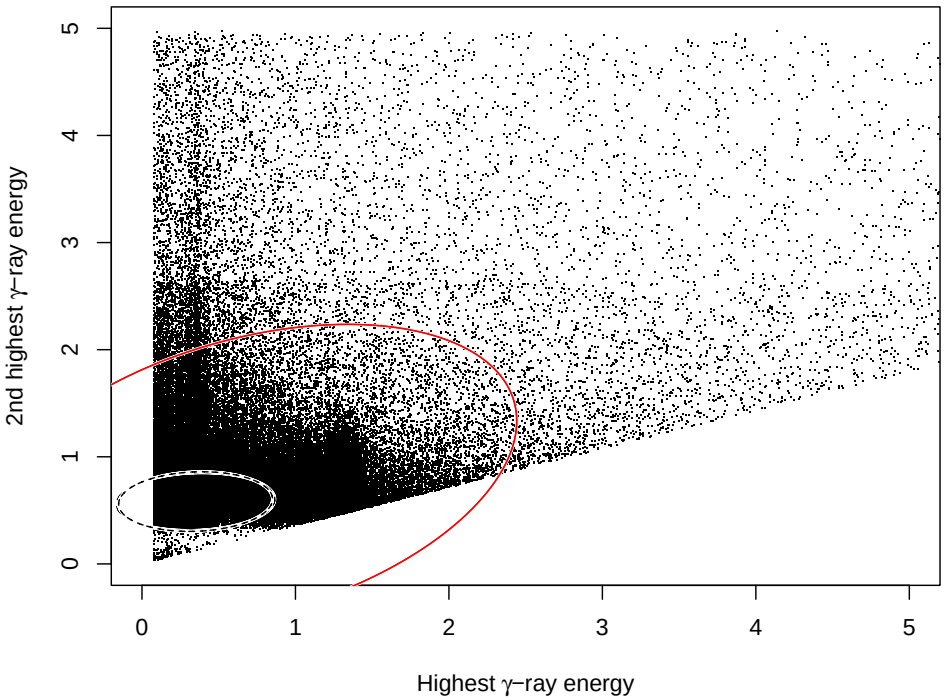


Figure 3.1: TRIUMF Data Set: In this plot is the data set with the contours based on 95% confidence ellipsoids derived from the asymptotic χ^2 approximation using the fast-MCD, MCD-BIRCH and classical estimates. The classical confidence contour is the red line. The x-axis has been trimmed for clarity.

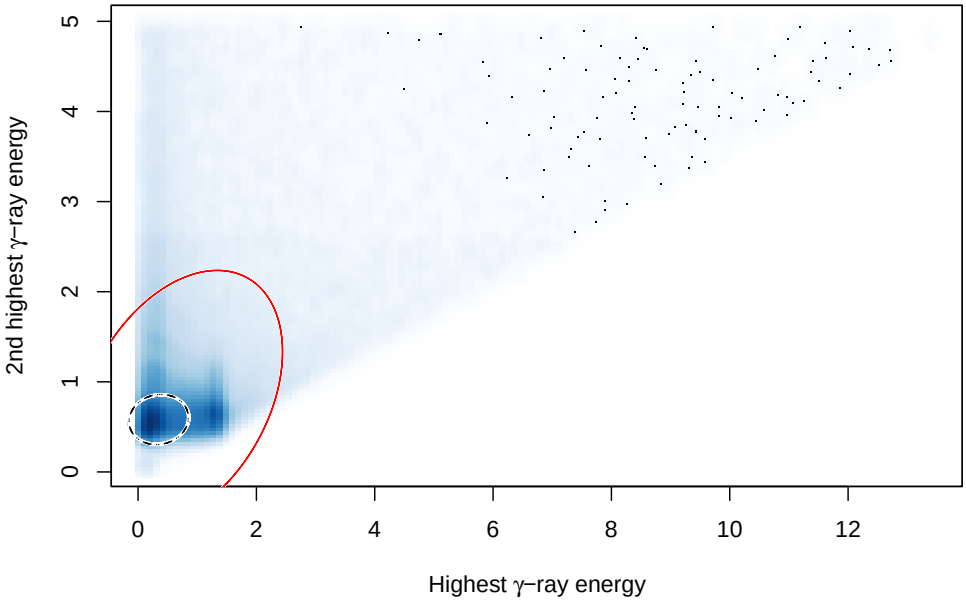


Figure 3.2: TRIUMF Data Set: a smoothed color density representation of the scatter plot using `smoothScatter` from the Bioconductor package `geneplotter`. The contours from Figure 3.1 are also provided. The axes have not been trimmed.

			Compactness						
			2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Closeness	2^{-0}	log(MCD)	−29.75	−29.35	−29.87	−30.56	−30.62	−30.67	−30.69
		N	7	10	37	160	569	1902	6570
		h	67858	68157	67628	66304	66262	66204	66205
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71
	2^{-1}	log(MCD)	−29.86	−30.07	−29.87	−30.56	−30.62	−30.67	−30.69
		N	13	16	37	160	569	1902	6570
		h	67654	67173	67628	66304	66262	66204	66202
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71
	2^{-2}	log(MCD)	−30.08	−30.08	−30.48	−30.61	−30.62	−30.67	−30.69
		N	49	49	51	154	569	1902	6570
		h	66421	66421	66374	66220	66262	66205	66202
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71
	2^{-3}	log(MCD)	−30.52	−30.52	−30.52	−30.47	−30.62	−30.67	−30.69
		N	137	137	137	163	576	1902	6570
		h	66272	66272	66272	66437	66260	66205	66202
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71
	2^{-4}	log(MCD)	−30.61	−30.61	−30.61	−30.64	−30.57	−30.67	−30.69
		N	409	409	409	417	625	1905	6570
		h	66243	66243	66243	66203	66347	66205	66205
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71
	2^{-5}	log(MCD)	−30.64	−30.64	−30.64	−30.64	−30.66	−30.68	−30.69
		N	1012	1012	1012	1012	1012	1937	6574
		h	66229	66229	66229	66229	66203	66201	66201
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71
	2^{-6}	log(MCD)	−30.68	−30.68	−30.68	−30.68	−30.68	−30.68	−30.69
		N	2943	2943	2943	2943	2943	2924	6589
		h	66201	66201	66201	66201	66201	66208	66201
		log(MCD)-R	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71	−30.71

Table 3.1: The effect of user selected parameters on number of subclusters and performance. N is the number of subclusters resulting from the selection of parameters, h is the number of actual observations in the h-subset, and performance is measured by the log of the MCD. The log of the MCD after refinement on the best solution from MCD-BIRCH is given by log(MCD)-R.

Selection of Parameters for MCD-BIRCH

As mentioned in Section 3.2.2, the selection of the user-parameters (compactness and closeness) is left to the user without much in the way of guidance other than balancing the size of the combinatorial problem with coarseness of the solution. In all applications in this article, the selection of these parameters was based entirely on achieving a certain number of subclusters. In most cases approximately 5000 subclusters was the level chosen, based on the proximity of the BIRCH solution to the algorithm it was benchmarking. In order to quantify the effect of this selection, we examine here the effect on the log-MCD for the DPOSS data set in Table 3.1 for different values of compactness and closeness.

Based on this analysis it would appear that the selection of parameters within certain "reasonable" bounds has little effect on the eventual efficacy of the algorithm. In particular, if further refinement on the solution is taken using concentration steps on the complete data, then the accuracy in the selection of the parameters is not crucial.

The philosophy of this article has been to treat the setting of these parameters much in the same way one would treat the setting of a bandwidth parameter in a smoothing operation. That is, there is no necessarily right or wrong answer, but rather something that suits the needs of the problem at hand.

3.3.2 Least Trimmed Squares

Consider a linear model of the type

$$Y = \mathbf{X}\boldsymbol{\beta} + \epsilon$$

with squared residuals given by r_i^2. Let $r_{(i)}^2$ be the ordered squared residuals. The Least Trimmed Squares (LTS) (Rousseeuw and Leroy, 1987) estimator is a robust regression estimator that can be tuned to have high-breakdown point, independently of the dimension p of the covariates $\mathbf{X}$. As the name suggests, the LTS estimator minimizes the trimmed sum of the squared residuals. For a given choice of $n/2 \leq h \leq n$, the LTS estimator minimizes

$$\sum_{i=1}^{h} r_{(i)}^2 \,,$$

where $r_{(i)}^2$ denotes the i-th order statistic of the squared residuals.

The so called "fast-LTS" algorithm in Rousseeuw and van Driessen (2006) uses a similar strategy as the fast-MCD algorithm described above. In this case the algorithm uses the initial random subsample to compute the ordinary least squares (OLS) regression estimator $\hat{\boldsymbol{\beta}}$. This initial estimator is then improved using the following "concentration" steps.

Concentration Step:

- Let $H \leftarrow \{$ the h smallest squared residuals, $r^2 = (\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}})^\top(\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}})\}$
- Calculate the OLS estimator $\hat{\boldsymbol{\beta}}$ based on the observations in H.

These steps, called concentration steps, are iterated until convergence or until a predetermined maximum number of iterations is exceeded. The output from this algorithm is the estimate for $\hat{\boldsymbol{\beta}}$ and a vector of the points that make up the best h-subset.

The BIRCH equivalent to LTS (LTS-BIRCH) uses the pre-processing from Section 3.2. However, this time the augmented matrix $\mathbf{Z} = (\mathbf{X}|\mathbf{Y})$ is preprocessed and that the elements of the corresponding CF's can be partitioned as follows:

$$LS_J = \begin{pmatrix} \sum_{i\in\mathcal{L}_J} \mathbf{X}_i \\ \sum_{i\in\mathcal{L}_J} Y_i \end{pmatrix} \qquad SS_J = \begin{pmatrix} \sum_{i\in\mathcal{L}_J} \mathbf{X}_i\mathbf{X}_i^\top & \sum_{i\in\mathcal{L}_J} \mathbf{X}_i Y_i \\ \sum_{i\in\mathcal{L}_J} \mathbf{X}_i Y_i & \sum_{i\in\mathcal{L}_J} Y_i^2 \end{pmatrix}$$

Note that $LS_J \in \mathbb{R}^{(p+1)}$ and $SS_J \in \mathbb{R}^{(p+1)\times(p+1)}$. It can then be shown by elementary matrix algebra that $\hat{\boldsymbol{\beta}}$ and $\sum_{i\in\mathcal{L}} r_i^2$ (the within-cluster sum of squared residuals) can be calculated based entirely on the CFs, and therefore they are sufficient statistics for the purposes of calculating the LTS objective function for the observations in the J subcluster.

The algorithm is as follows.

LTS-BIRCH:

1. Run the pre-processing algorithm.
2. For each subcluster, calculate the rank of the sum-of-squares matrix. Retain an index of those subclusters whose sum-of-squares are full rank.
3. Randomly select a sufficient number of subclusters of full rank to adequately cover the solution space. For each of these clusters (denoted by H):

(a) DPOSS

fast-LTS –
Model:
$\texttt{MAperF} = 1.728 - 0.957 \times \texttt{csfF}$
log(RMS)= -7.698

LTS-BIRCH –
Model:
$\texttt{MAperF} = 1.712 - 0.938 \times \texttt{csfF}$
log(RMS) = -7.696
log(RMS) with refinement = -7.698

(b) TRIUMF

fast-LTS –
Model:
$Y = 0.595X$
RMS = 0.03921

LTS-BIRCH –
Model:
$Y = 0.597X$
RMS = 0.03926
RMS with refinement = 0.03921

Table 3.2: The fitted regression lines for each approach on each data set, and the Mean Squared Residuals (with logarithm applied for DPOSS). The refinement step applies one concentration step on the whole data set using the solution from LTS-BIRCH.

(a) Calculate the OLS estimator $\hat{\boldsymbol{\beta}}$ based on H.

(b) Using $\hat{\boldsymbol{\beta}}$, calculate the sum of squared residuals (SSR) for all other subclusters.

(c) Order these SSR, and select the smallest number of subclusters with smallest SSR's such that the sum of the number of observations in the union of these subclusters exceeds $h = \alpha n$.

(d) Let H be the union of these subclusters, and update its CF.

(e) Iterate until convergence or until a fixed number of iterations is exceeded.

4. Select the configuration with smallest sum of squared residuals.

Example. Revisiting the DPOSS data, the LTS estimate is calculated using the `ltsReg` command from the package `robustbase`, and then using the equivalent BIRCH approach available in the `birch` package. Following the same approach as in Rousseeuw and van Driessen (1999), the data set consists of just those observations classed as stars, and the relationship between the attributes "csfF" and "MAperF". The plot of the data, with fitted lines, is given in Figure 3.3, and the fitted models in Table 3.2(a). The fast-MCD algorithm used 5000 random starting subsamples, and the LTS-BIRCH algorithm 100 random starting subclusters. For a comparison of processing times for each algorithm, see Section 3.4.1.

Example. Returning to the TRIUMF Data Set, we perform LTS without an intercept term. A plot of the fits is given in Figure 3.4, with results in Table 3.2(b). The two sets have 97.7% of points in common. Note that the fast-LTS and LTS-BIRCH lines are sufficiently similar that they obscure each other.

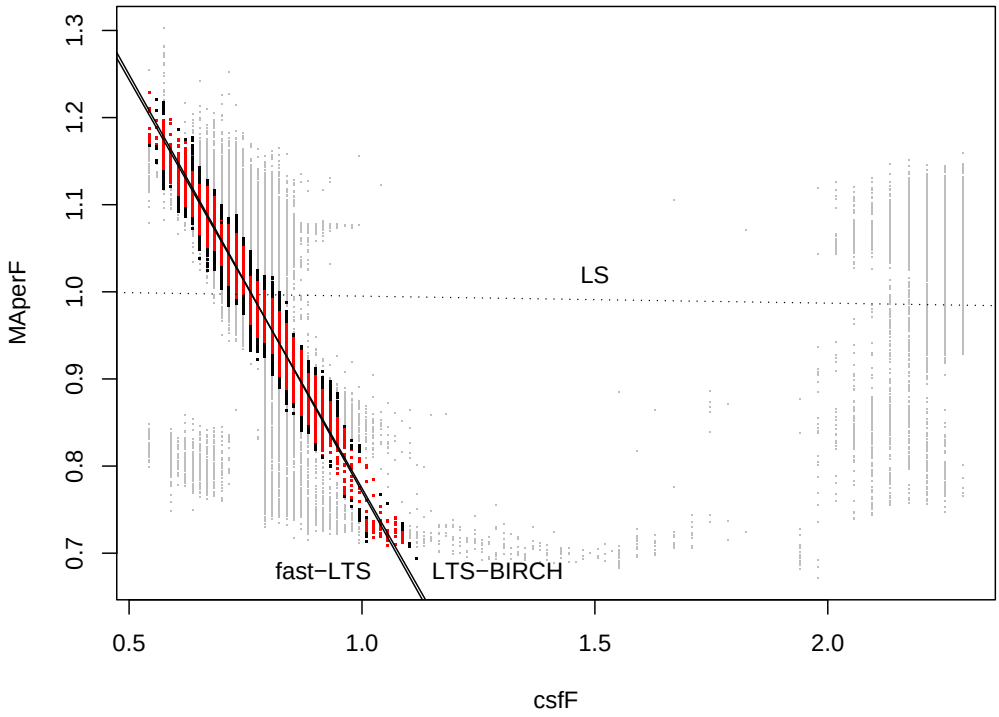


Figure 3.3: DPOSS Data Set. The lines are the resulting regression lines for fast-LTS, LTS-BIRCH and classical LS. The red data points are for points in both LTS $h-$subsets, and the darker points are for those that are only in one of the sets.

3.3.3 Robust Linear Grouping Analysis

Linear Grouping Analysis (LGA) (Van Aelst et al., 2006) is a clustering algorithm for data sets in which the interest lies in detecting linear structures, such as hyperplanes. The main advantage of this algorithm over others with similar goals (e.g. Gawrysiak et al., 2001) is that it does not require an explanatory/response relationship between the variables (it is not model based), but rather finds hyperplanes using orthogonal regression.

Assuming that we are searching for k hyperplanes, and given a membership vector C_i for the i-th cluster, the optimization problem associated with LGA is

$$\min_{C_1,\ldots,C_k} \sum_{i=1}^{k} \sum_{j \in C_i} d(x_j, \mathcal{H}_i)^2 \,, \tag{3.2}$$

with $d(x_j, \mathcal{H}_i)$ the orthogonal distance of x_j to hyperplane $\mathcal{H}_i$. The minimization is taken over all possible disjoint partitions of $\{1, \ldots, n\}$ into k sets $C_1, \ldots, C_k$. The proposed algorithm to find a solution to (3.2) starts from an initial configuration (obtained drawing random subsamples from the data) and then forming clusters by assigning points to their closest hyperplane (using orthogonal distances). The current configuration is then updated using the new clusters, and this procedure iterated until convergence. After repeating this for many starting configuration, the one with the smallest final residual orthogonal sum of squares is selected as the approximated solution to (3.2).

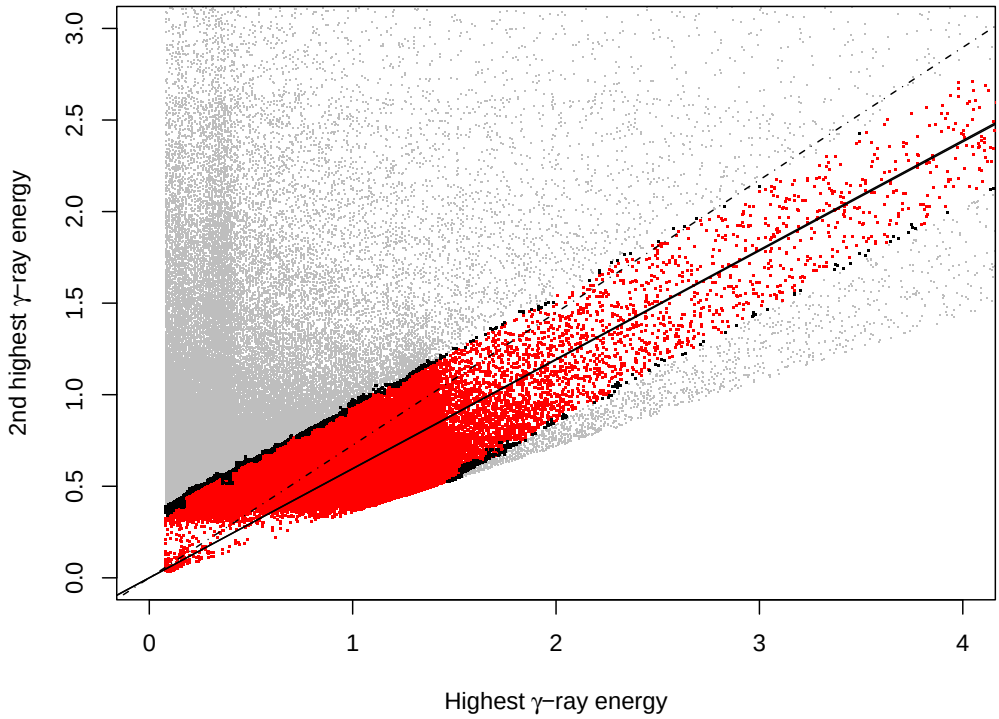


Figure 3.4: TRIUMF Data Set. The lines are the resulting regression lines for fast-LTS, LTS-BIRCH and OLS (dashed). The red points are those in both LTS $h-$subsets, and the darker points are those that are only in one of the sets. The x-axis has been trimmed for clarity.

Recently, a robust version of this algorithm has been proposed in García-Escudero et al. (2007). For a given integer $n/2 \leq h \leq n$ (that determines the robustness properties of the solution), we need to solve

$$\min_{H \in \mathcal{H}} \min_{C_1^H, \ldots, C_k^H} \sum_{i=1}^{k} \sum_{j \in C_i^H} d(x_j, \mathcal{H}_i)^2 \,,$$

where $\mathcal{H}$ is the set of all subsets of $\{1, \ldots, n\}$ of size h, and, as before, $C_1^H, \ldots, C_k^H$ form a disjoint partition of size k of H, for each $H \in \mathcal{H}$. In terms of finding an approximate solution to this problem, the proposed algorithm is very similar to that of LGA.

- Let $H \leftarrow \{$ the h smallest orthogonal distances $d(x_i)$, $i = 1, \ldots, n\}$, where $d(x_i)$ denotes the orthogonal distance of x_i to the closest hyperplane.
- Update the hyperplanes using only the observations in H.

These steps are iterated until convergence or a pre-determined upper bound on the number of iterations is attained.

Our new algorithm, RLGA-BIRCH, instead uses the subclusters derived in Section 3.2, and is given by:

1. For each subcluster, calculate its covariance matrix, and the corresponding rank. Retain an index of those subclusters whose covariance matrix are of full rank.

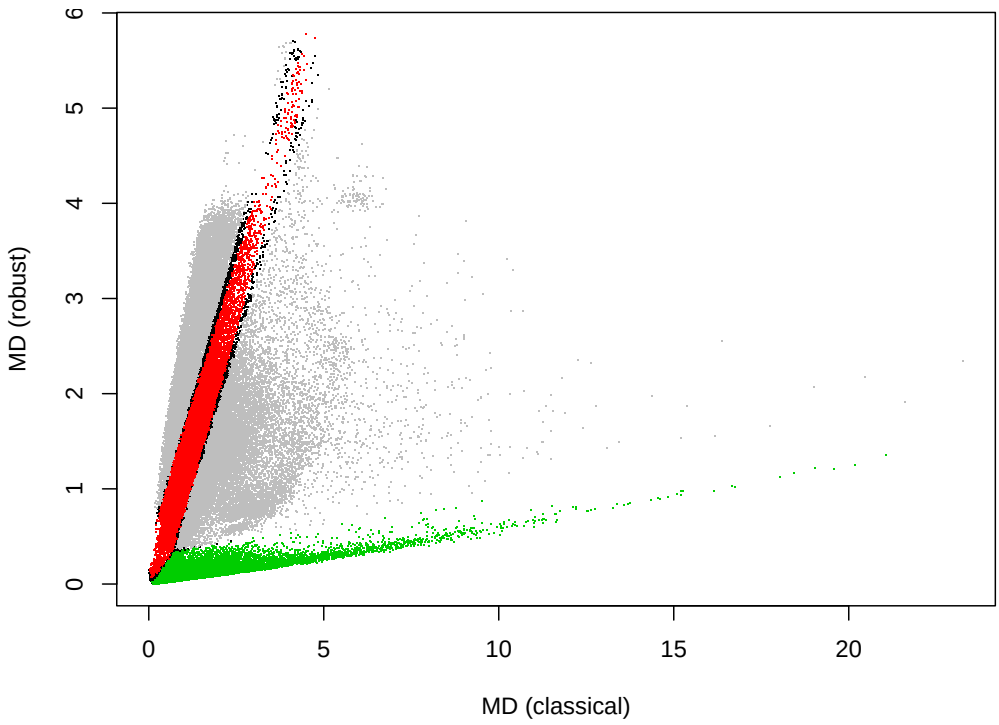


Figure 3.5: DPOSS Data Set: The red and green points represent the output where both RLGA and RLGA-BIRCH agree. The darker points are those where the two algorithms are not in agreement.

2. Randomly select a subset of k subclusters of full rank. Use each for a seed to:
   (a) Calculate the hyperplanes of each seed, using the covariance estimate derived from the seed's CF.
   (b) Calculate the average residual orthogonal sum-of-squares for each subcluster, to every hyperplane. Retain just the distance to the closest hyperplane.
   (c) Order the distances, and select the smallest subset of the closest subclusters such that the sum of the number of observations within these subclusters just exceeds h.
   (d) For each hyperplane, generate a fit based on all subclusters that belong to it and are in the "best" subset. Use these as the seeds for the next iteration.
   (e) Iterate until convergence or until a predetermined maximum number of iterations is exceeded.
3. Select the configuration with smallest residual orthogonal sum-of-squares.

Example. DPOSS Data Set. We construct an outlier map (Rousseeuw and Van Zomeren, 1990) by plotting the Mahalanobis distances using standard estimates against Mahalanobis distances using robust distances, given in Figure 3.5. We then apply RLGA and RLGA-BIRCH to this data set as a means of robustly identifying the two groups.

The residual orthogonal sum-of-squares using the original algorithm is 870.0 versus the BIRCH approach with 873.0. The two best h-subsets have 98% of observations in common. Finally, one

refinement step performed using the BIRCH solution to initialize the algorithm, gives a best h-subset with residual orthogonal sum-of-squares of 870.0, and 99.8% of observations in common with the original algorithm's solution.

3.4 Simulation Studies

3.4.1 Simulation Study with LTS

The example in Section 3.3.2, suggests that LTS-BIRCH compares very well with the best algorithms available to compute LTS when the data fit in memory. In this section we report the results of a simulation study that confirms this observation. We also provide a simple example where LTS-BIRCH does significantly better than the fast-LTS using much fewer starting candidates. The reason for this lies in the much smaller (though coarser) solution space belonging to LTS-BIRCH, and is easier to explore thoroughly via a random search algorithm.

Let the level of contamination be denoted $c\%$. Each data set has $[n(1-c)]$ observations generated from the null model $Y = \mathbf{X}\boldsymbol{\beta}_0 + \epsilon$, $\mathbf{X} \sim \mathcal{N}_p(0, \mathbf{I})$, $\epsilon \sim \mathcal{N}(0,1)$ with $\boldsymbol{\beta}_0 = (1, \ldots, 1)^\top$. We added $[nc]$ of contamination following a linear model with $\boldsymbol{\beta} = (-1, 1)^\top$ when $p = 2$, and $\boldsymbol{\beta} = (5.5, -23.5, 1, \ldots, 1)^\top$ when $p = 20$, and covariates $X_c \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{\mu} = (5, 0, \ldots, 0)$ and $\boldsymbol{\Sigma}$ a diagonal matrix of 1.1.

In order to make a fair comparison, we introduce two additional methods referred to as LTS-BIRCH-R and fast-LTS-BIRCH. LTS-BIRCH-R takes the best solution from LTS-BIRCH, and uses it as a seed for up to twenty concentration steps, or less if convergence of the "best" subset is achieved, using the whole data set. This allows for a more realistic comparison in objective function with fast-LTS.

It is difficult to make meaningful comparisons in the processing times due to differences in language used for coding each algorithm. The fast-LTS algorithm is written in FORTRAN and somewhat mature. The tree building part of LTS-BIRCH is written in C++, which was chosen for ease in development rather than speed, while the LTS part of the algorithm is coded in R. Finally, the concentration steps are performed in R.

In order to get a rough upper bound on the timing of the R part of these algorithms, we propose the following: build the CF-Tree in C++, and then perform LTS using fast-LTS on just the centres of the subclusters. We call this method fast-LTS-BIRCH. While the results with respect to objective function could be very different to that of LTS-BIRCH, the processing time of fast-LTS-BIRCH will approximate the time of LTS-BIRCH were it to be coded in C/FORTRAN, as (a) the number of starting samples and steps to convergence are approximately the same; and (b) is an upper bound since fast-LTS performs many more concentration steps on different starting candidates than LTS-BIRCH-R.

The metrics used to measure the effectiveness of each algorithm are:

(a) the percentage difference in the "best h mean squared residual" compared to the residuals obtained with $\boldsymbol{\beta}_0$, i.e.

$$\frac{\bar{r}_i - \bar{r}_{i,0}}{\bar{r}_{i,0}}$$

	$n = 30,000$	$n = 100,000$	$n = 250,000$	$n = 1,000,000$
$p = 2,\ c = 0\%$	0.05	0.1	0.1	0.15
$p = 2,\ c = 15\%$	0.1	0.1	0.1	0.3
$p = 2,\ c = 35\%$	0.1	0.1	0.1	0.3
$p = 20,\ c = 35\%$	22	25	27	30

Table 3.3: The parameters used for creating the CF-tree using BIRCH for the LTS simulation study. The parameters are closeness and compactness, and in each simulation the parameters are equal.

where $\bar{r}_i$ is the average of the h-subset of squared-residuals from the given model (fast-LTS, MCD-BIRCH, etc.), and $\bar{r}_{i,0}$ is the average of the h smallest squared "residuals" $Y_i - \boldsymbol{\beta}_0 \mathbf{X}_i$.

(b) the percentage of "outliers" (contaminated observations) that are included in the best h-subset.

In each case, we generated 100 samples, and the outputs from fast-LTS, LTS-BIRCH and LTS-BIRCH-R retained,[5] for the following combinations of sample size n, number of covariates p, and proportion of contamination c:

- $p = 2,\ 20$
- $n = 30,000,\ 100,000,\ 250,000,\ 1,000,000$
- $c = 0\%, 15\%,\ 35\%.$

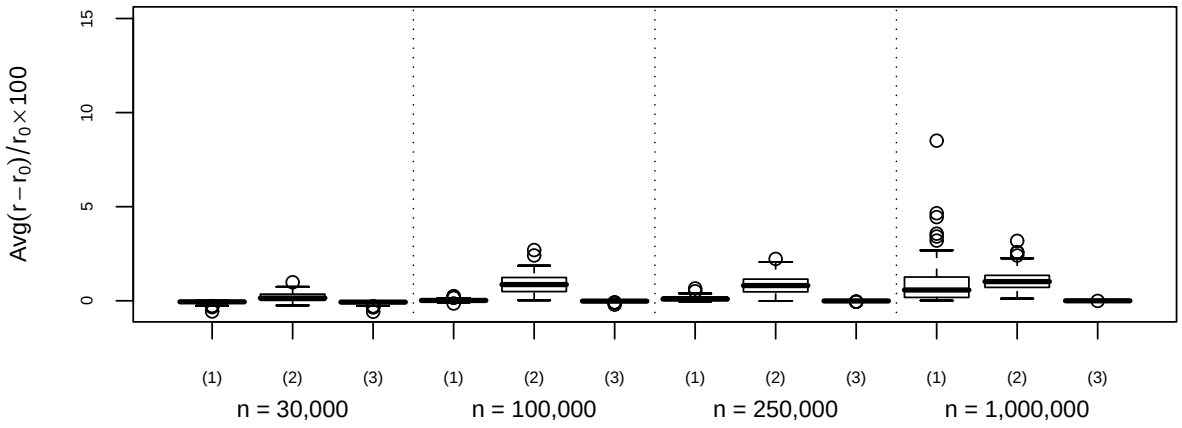
The parameters used are given in Table 3.3, and were set in order to achieve approximately 3,000-4,000 subclusters. The algorithm fast-LTS used 500 subsamples for starting seeds, and LTS-BIRCH and LTS-BIRCH-R used 100. The level α was set at 0.5. The processing time was measured on a Intel Xeon CPU 3.00GHz linux box.

Box plots of the first metric are given in Figures 3.6 and bar plots of the second metric in Figure 3.7. The processing times are given in Table 3.4.

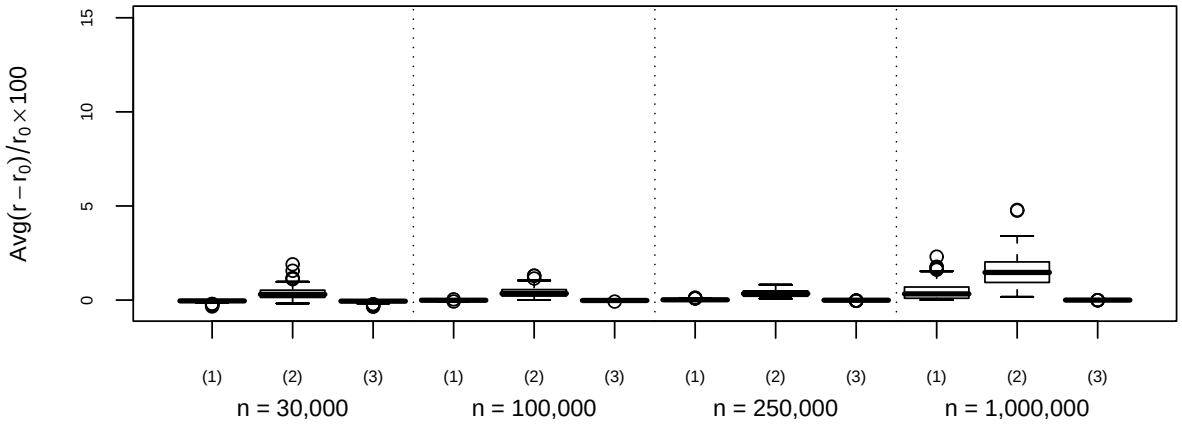
Firstly, with the parameters $p = 2$, $c = 0\%$ and $p = 2$, $c = 15\%$ (Figures 3.6(a), (b)), it is clear that all methods produce residuals very close to that of the null model, and the performance bar plot (omitted) reinforces this, with both methods having less than 5% contaminated observations in the best h-subset for all simulations. However, for $p = 2, c = 35\%$ it becomes apparent that LTS-BIRCH is beginning to outperform fast-LTS in both residuals (Figure 3.6(c)) as well as percent contamination (Figure 3.7(e)). Finally, for $p = 20$, $c = 35\%$ LTS-BIRCH clearly is still finding the correct solution, whereas fast-LTS is not (Figures 3.7(d) & 3.7(f)).

In terms of processing time, it is not surprising that in all cases fast-LTS was faster than LTS-BIRCH and LTS-BIRCH-R, but as identified previously, it is not reasonable to make direct comparisons due to differences in computing language. Instead, a more reasonable comparison in processing time can be gained with fast-LTS and fast-LTS-BIRCH, which are clearly much closer. Given that there is a lot of scope for making this algorithm more efficient, primarily through choice of the

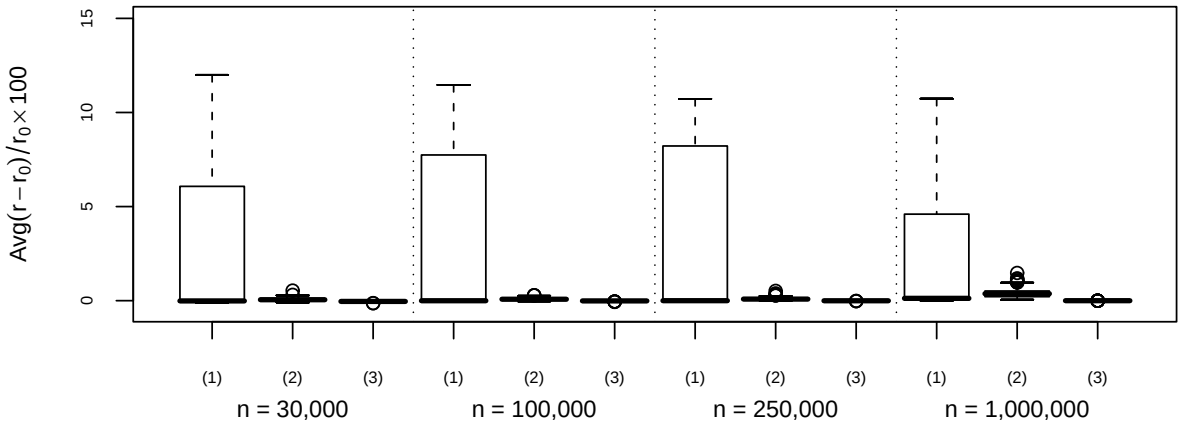
[5]For fast-LTS-BIRCH, just the processing time is kept.



(a) $p = 2,\ c = 0\%$

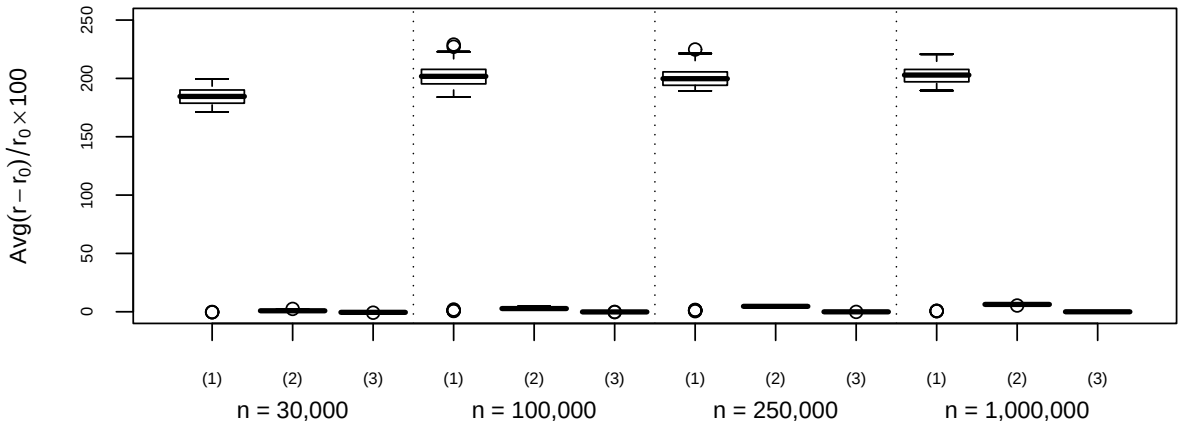


(b) $p = 2,\ c = 15\%$



(c) $p = 2,\ c = 35\%$

Figure 3.6: Box plots of the mean difference between the "true" LTS residual and the one calculated using different methods. The methods are: (1) fast-LTS; (2) LTS-BIRCH; (3) LTS-BIRCH-R. Each data set (given by n) is simulated 100 times.



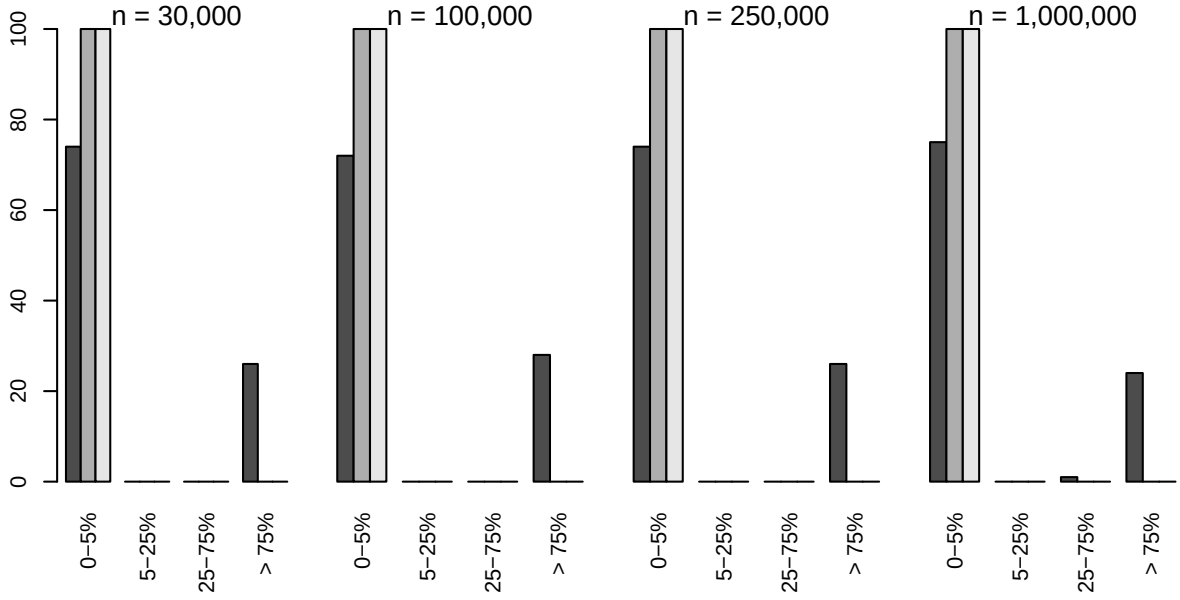
(d) $p = 20,\ c = 35\%$

Figure 3.6 *continued*: Box plots of the mean difference between the "true" LTS residual and the one calculated using different methods. The methods are: (1) fast-LTS; (2) LTS-BIRCH; (3) LTS-BIRCH-R. Each data set (given by n) is simulated 100 times. Note that the y-axis in this case has a different scale.

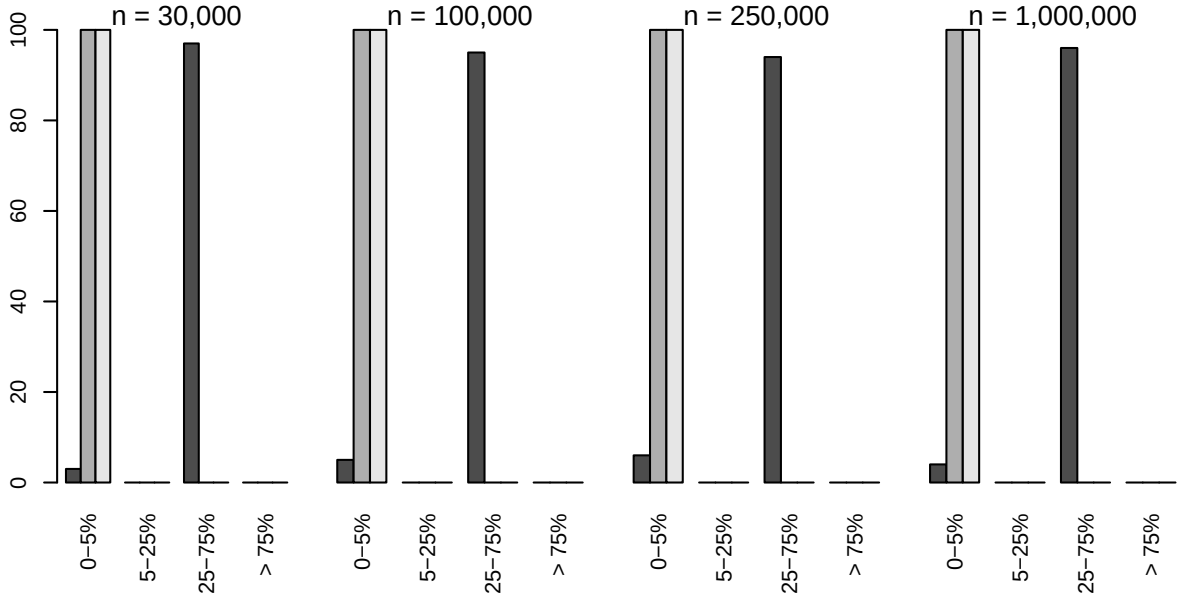
	$p = 2,\ c = 0\%$				$p = 2,\ c = 15\%$			
n	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
30,000	0.8	21.0	39.6	3.2	0.8	12.5	13.1	2.2
100,000	2.1	17.3	32.8	4.5	2.0	19.1	21.5	4.8
250,000	4.6	25.6	49.2	9.2	4.4	29.6	36.7	9.7
1,000,000	10.6	41.9	85.4	27.4	11.7	26.7	55.7	19.5

	$p = 2,\ c = 35\%$				$p = 20,\ c = 35\%$			
n	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
30,000	0.7	11.8	12.1	2.3	2.4	55.2	57.0	4.3
100,000	1.9	18.0	19.4	4.7	4.2	57.0	65.1	7.7
250,000	4.1	28.2	32.8	9.7	10.3	59.7	83.3	14.3
1,000,000	11.5	26.6	52.5	19.4	40.3	95.0	225.9	49.5

Table 3.4: The median total processing time for each LTS algorithm in seconds. Key: (1) - fast-LTS; (2) - LTS-BIRCH; (3) - LTS-BIRCH-R; (4) fast-LTS-BIRCH.



(e) $p = 2,\ c = 35\%$



(f) $p = 20,\ c = 35\%$

Figure 3.7: Bar plots of the percentage of contaminated observations in the "best" subset for LTS. The data set is generated 100 times and the percent contamination in the $h-$subset is retained. The darkest shade bars are for fast-LTS, the mid-shade for LTS-BIRCH, and the lightest bars for LTS-BIRCH-R.

language in which it is coded, it is expected that with further development this algorithm will be much faster.

It should be noted that, regardless of whether or not it is faster than fast-LTS, LTS-BIRCH-R does very well in all scenarios. In those situations where the fast-LTS solution has a slightly lower objective function than LTS-BIRCH (in particular, the few visible in Figures 3.6(a), (b)), concentration steps on the whole data set using one solution from the LTS-BIRCH solution as a seed yields at worst identical results as the fast-MCD solution, and in most cases a better solution.

3.4.2 Simulation Study with MCD

In this section we confirm the results found in Section 3.3.1, demonstrating via a simulation study how the number of dimensions, the level of contamination, and number of observations affect the performance of fast-MCD and MCD-BIRCH.

Our simulation involved selecting different levels of n, p and contamination ($c\%$), and generating 100 data sets. We then calculated the MCD for each data set using the fast-MCD algorithm as implemented in the R package `robustbase`, followed by MCD-BIRCH, and MCD-BIRCH with a refinement step consisting of a sequence of concentration steps performed using the best MCD-BIRCH solution as the initial seed. This last method is denoted MCD-BIRCH-R, and is a more direct comparison of MCD, as the fast-MCD has a number of concentration steps on its best solutions. Once again we also provide the timings for fast-MCD-BIRCH for comparison with fast-MCD, with the former method carrying out fast-MCD on just the centres of the subclusters as an estimate of the upper bound for processing MCD-BIRCH were it to be coded in FORTRAN/C++.

Each data set had $[n(1-c)]$ observations generated from the null model $\mathbf{X} \sim \mathcal{N}_p(\boldsymbol{\mu}_0, \mathbf{I})$ with $\boldsymbol{\mu}_0 = (0, \ldots, 0)^\top$. We then added $[nc]$ of contaminated observations with model $\mathbf{X} \sim \mathcal{N}_p(\boldsymbol{\mu}, \mathbf{I})$ with $\boldsymbol{\mu} = (10, \ldots, 10)^\top$. This is the same simulation study as that performed in (Rousseeuw and van Driessen, 1999).

The metrics used to measure the effectiveness of each algorithm are:

(a) the logarithm of the determinant of the sample covariance of the best h-subset to the asymptotic estimate i.e.

$$\log\left(|S|\right)$$

where $|S|$ is the MCD estimate from the method used; and

(b) the percentage of "outliers" (contaminated observations) that are included in the best h-subset.

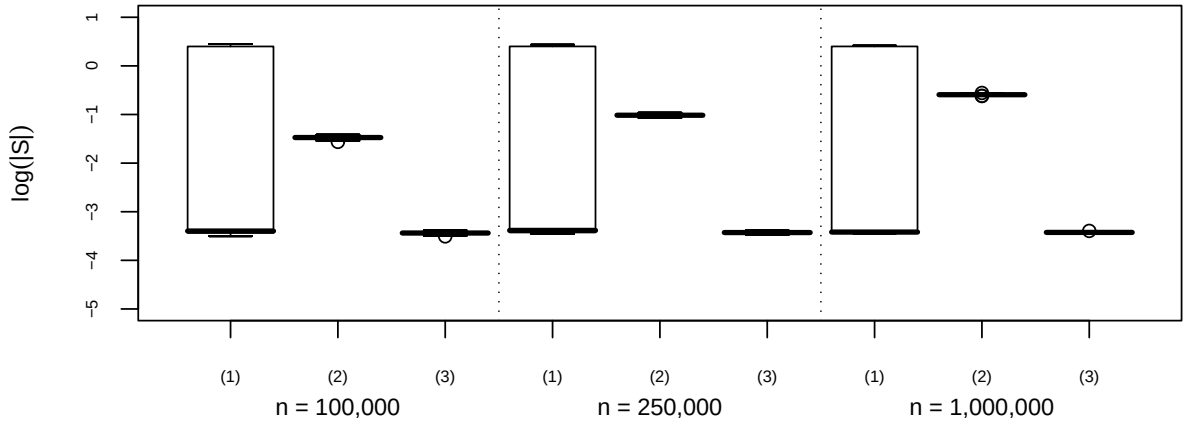
The logarithm is applied because of the sometime large differences in MCD estimates for different algorithms and simulations.

The parameters used for creating the BIRCH trees are given in Table 3.5. Plots of each metric and method for $p = 20$ are given in Figures 3.8 – 3.9, and similarly in Figures 3.10 – 3.11 for $p = 30$. Finally, the processing time was measured on a Intel Xeon CPU 3.00GHz linux box, and are given in Table 3.6.

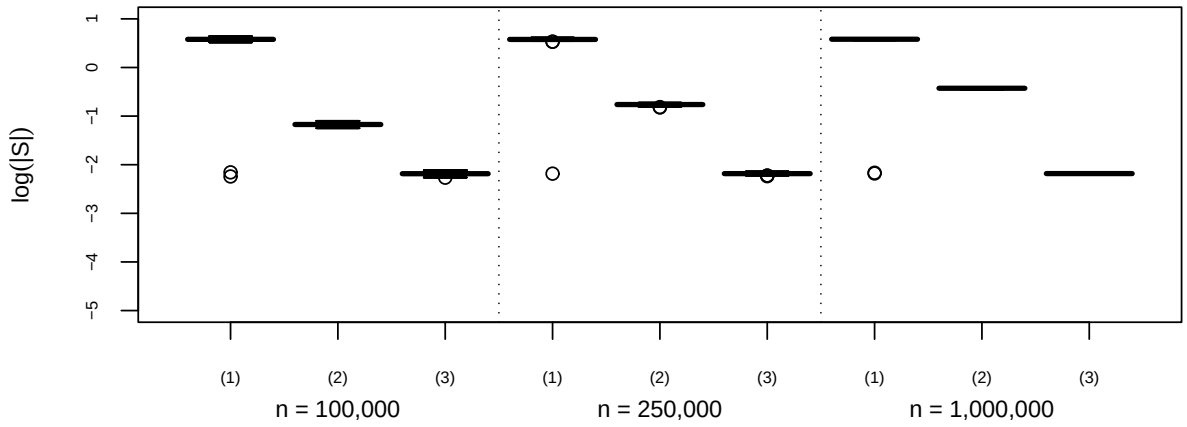
In the set of plots for $p = 20$, it is clear that the number of observations, n, does not significantly affect the performance of the fast-MCD or MCD-BIRCH-R methods, and in all cases the fast-MCD median MCD is larger than the median MCD-BIRCH-R estimate. What is interesting is that as

	$n = 100,000$	$n = 250,000$	$n = 1,000,000$
$p = 20,\ c = 30\%$	23	26	30
$p = 20,\ c = 40\%$	23	26	30
$p = 30,\ c = 30\%$	40	45	50
$p = 30,\ c = 40\%$	40	45	50

Table 3.5: The parameters used for creating the CF-tree using BIRCH for the MCD simulation study. The parameters are closeness and compactness, and in each simulation the parameters are equal.

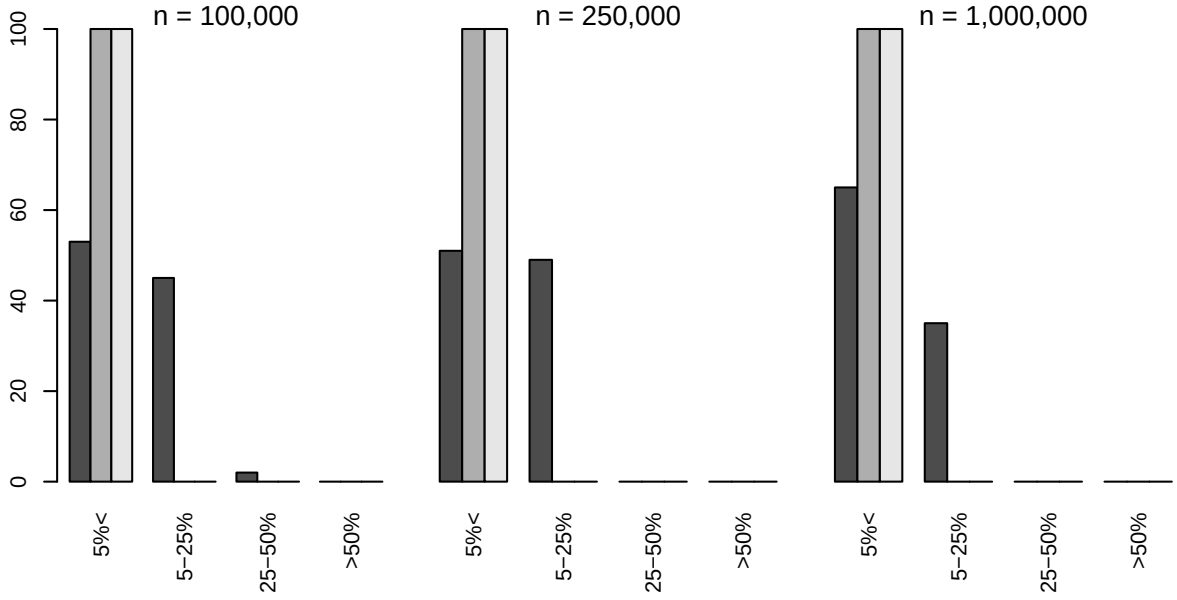


(a) $p = 20,\ c = 30\%$

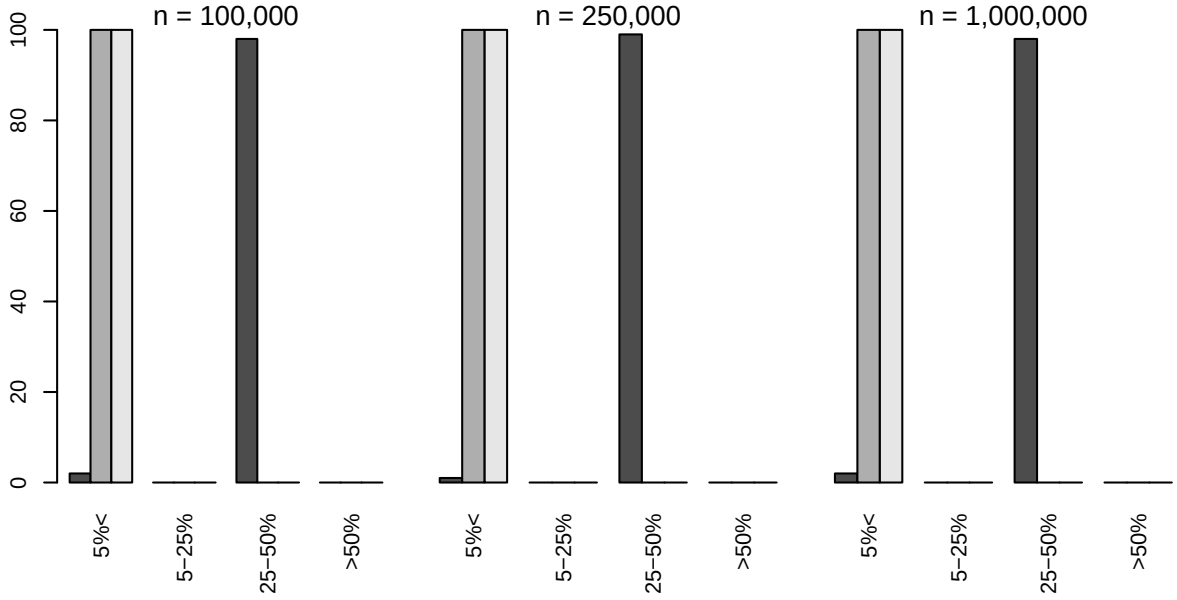


(b) $p = 20,\ c = 40\%$

Figure 3.8: Box plots of the logarithm MCD estimate, calculated using three different methods. The methods are: (1) fast-MCD; (2) MCD-BIRCH; (3) MCD-BIRCH-R. Each data set (given by n) is simulated 100 times.

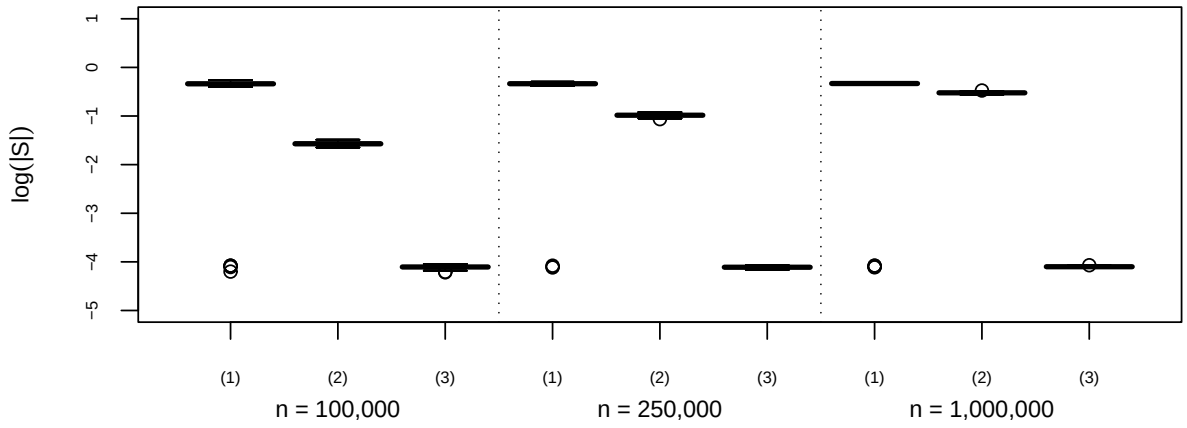


(a) $p = 20,\ c = 30\%$

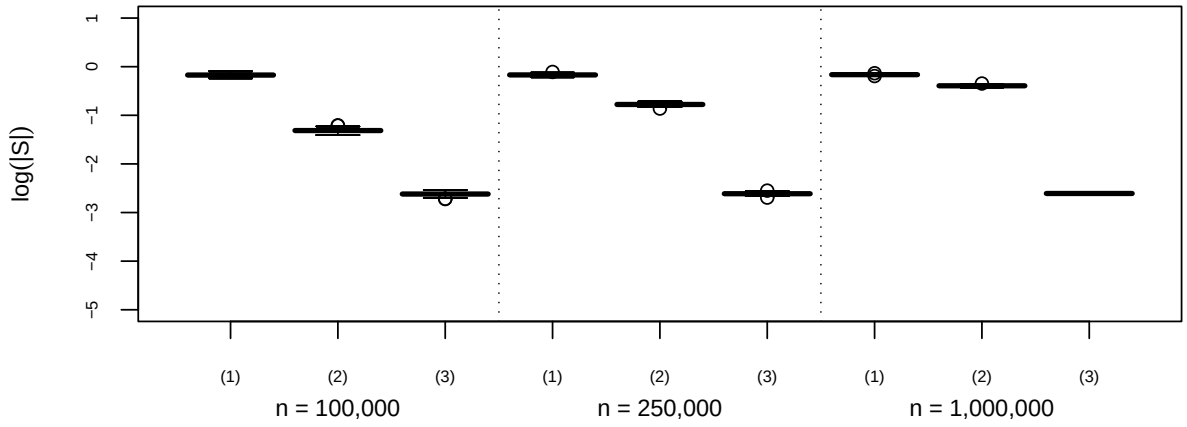


(b) $p = 20,\ c = 40\%$

Figure 3.9: Bar plots of the percentage of contaminated observations in the "best" subset for MCD. The data set is generated 100 times and the percent contamination in the $h-$subset is retained. The darkest shade bars are for fast-MCD, the mid-shade for MCD-BIRCH, and the lightest bars for MCD-BIRCH-R.



(a) $p = 30,\ c = 30\%$



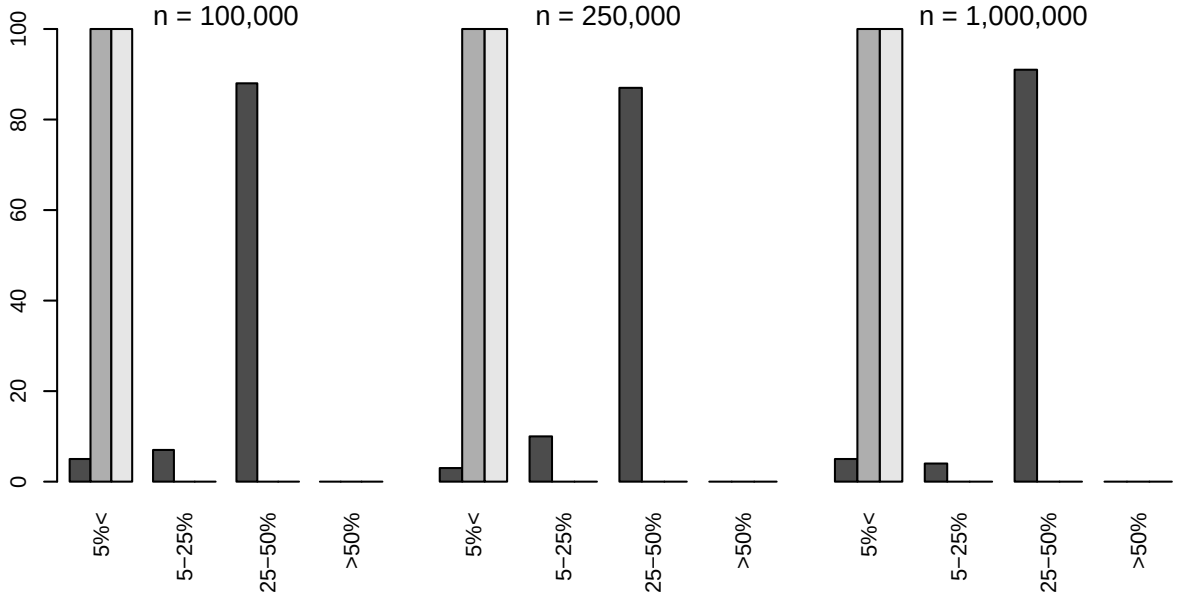
(b) $p = 30,\ c = 40\%$

Figure 3.10: Box plots of the logarithm of the MCD estimate, calculated using three different methods. The methods are: (1) fast-MCD; (2) MCD-BIRCH; (3) MCD-BIRCH-R. Each data set (given by n) is simulated 100 times.

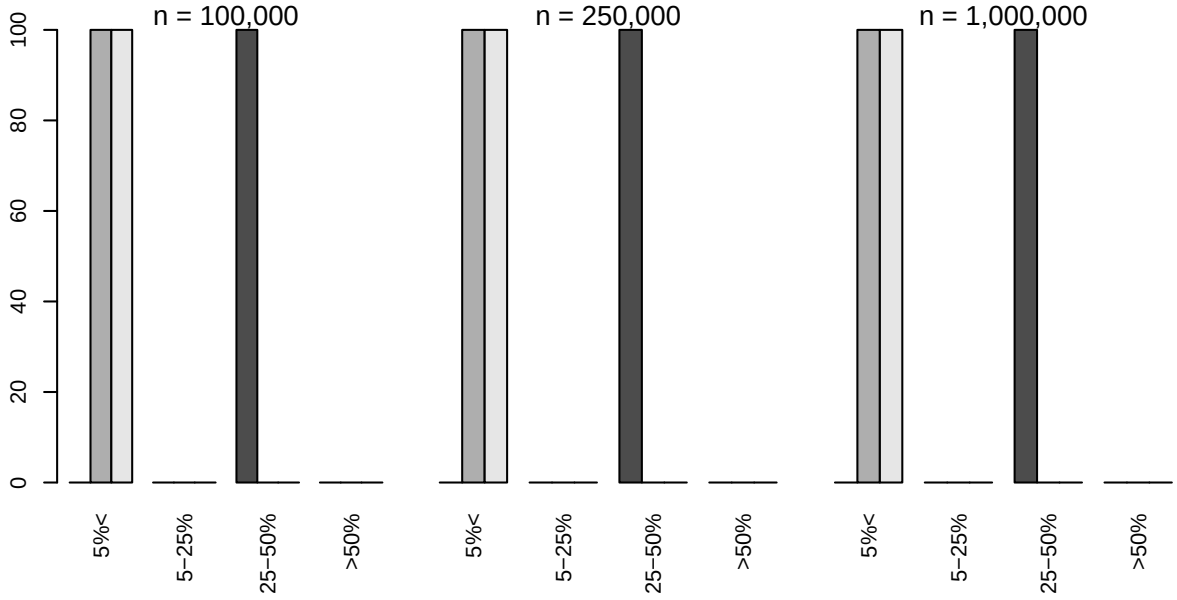
n increases, so too does the MCD-BIRCH estimate, but in the bar plots in Figure 3.9 the MCD-BIRCH method still performs very well, with no contaminated observations in the best h-subset. This will be discussed shortly.

In the set of plots for $p = 30$, it is clear that the MCD-BIRCH-R method is consistently producing the most optimal result, whereas fast-MCD only finds an equivalent result a few times. For the remainder of the simulations, fast-MCD and MCD-BIRCH are producing similar, less optimal results (relative to MCD-BIRCH-R).

In both the box plots of MCD, it is clear that as n increases, so too does the MCD for MCD-BIRCH, and yet the performances given in the bar plots remain perfect. While, at first glance, this may appear contradictory, however upon further investigation it appears that this is an artifact of the methodology for selecting the radius and compactness attributes i.e. choosing parameters such that we have approximately 3000-4000 subclusters. Therefore, although the number of subclusters has remained the same, the magnitude of the data reduction, equivalently the increase in data



(a) $p = 30,\ c = 30\%$



(b) $p = 30,\ c = 40\%$

Figure 3.11: Bar plots of the percentage of contaminated observations in the "best" subset for MCD. The data set is generated 100 times and the percent contamination in the $h-$subset is retained. The darkest shade bars are for fast-MCD, the mid-shade for MCD-BIRCH, and the lightest bars for MCD-BIRCH-R.

	$p = 20$, $c = 30\%$				$p = 30$, $c = 30\%$			
n	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
100,000	6.12	31.18	39.66	7.74	9.92	39.86	50.36	10.38
250,000	16.22	44.31	72.92	14.82	29.88	57.50	109.78	19.45
1,000,000	69.02	91.47	245.42	50.75	113.72	116.82	364.35	65.76

	$p = 20$, $c = 40\%$				$p = 30$, $c = 40\%$			
n	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
100,000	8.29	50.83	60.90	8.28	16.37	56.31	74.50	12.52
250,000	15.83	52.78	75.82	14.38	26.96	65.81	105.17	21.19
1,000,000	64.17	96.95	203.91	48.72	109.50	115.75	314.15	65.41

Table 3.6: The median total processing time for each MCD algorithm in seconds. Key: (1) - fast-MCD; (2) - MCD-BIRCH; (3) - MCD-BIRCH-R; (4) fast-MCD-BIRCH.

granularity, is getting higher. Looking at the estimate of centre and dispersion derived from the MCD-BIRCH algorithm turns out to be nearly identical with the "known" result in all cases. This phenomenon deserves more investigation, and will be left for further research.

Based on the results in Table 3.6, fast-MCD has a shorter processing time than MCD-BIRCH and MCD-BIRCH-R in all cases. It should also be reiterated that fast-MCD is entirely written in FORTRAN, and therefore the fast-MCD-BIRCH algorithm provides a more direct comparison in timings. In this case, therefore, the BIRCH algorithm is significantly faster than the fast-MCD algorithm when $n = 250,000$ and $1,000,000$ regardless of p, and certainly provides a direction for further optimization.

3.5 Conclusion

Computing robust estimators with good robustness properties generally requires finding the solution to highly complex optimization problems. The current state-of-the-art algorithms to find approximate solutions to these problems involve the local refinement of a potentially large number of random starting points. These random initial candidates are generally constructed by drawing random subsamples from the data, and the iterative refinement steps involve calculations over the whole data set. Since these approximating algorithms typically need to access the entire data set a very large number to times, they become unfeasible in practise when the data do not fit in the computer's memory.

In this paper we show how the BIRCH algorithm can be adapted to calculate approximate solutions to problems in this class. We illustrate our approach by applying it to find approximate solutions to the optimization problems that define the LTS and MCD estimators, and also to the RLGA algorithm. Our approach is compared with the fast-LTS and fast-MCD algorithms on real large data sets that fit in memory. We show that, even for very large data sets that fit in memory, our proposal is able to find approximate LTS and MCD estimators that compare very well with those returned by the fast-LTS and fast-MCD algorithms. Furthermore, in this case (large data sets that fit in the computer's memory), our approximate solution can easily be iteratively refined (as it is

done in both fast-LTS and fast-MCD). In some cases, this procedure converges to the same solution as fast-LTS and fast-MCD (or an extremely close one); but in some other cases our approach is able to find a better solution (in terms of value of the objective function) than that returned by the fast- algorithms. Note that our algorithm can find this good starting point much faster than both the fast-LTS and fast-MCD algorithms.

Finally, the simulation studies indicate that our method performs comparably well to fast-LTS and fast-MCD in simple situations (large data sets with a small number of covariates and small proportion of outliers) and does much better than fast-LTS and fast-MCD in more challenging situations without requiring extra computational time. These findings seem to confirm that when attempting to find a good approximated LTS or MCD estimator for data sets that do not fit in memory, our approach provides a computationally feasible and reliable algorithm. Furthermore, when additional concentration steps are performed on the whole data set using the single best result from LTS-BIRCH and MCD-BIRCH, the resulting estimators always exceed those from fast-LTS and fast-MCD.

To the best of our knowledge there is currently no other reliable algorithm in the literature that can be used for these problems without having to access the whole data set a large number of times.

Bibliography

M. M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. Data bubbles: Quality preserving performance boosting for hierarchical clustering. *ACM SIGMOD*, 2001.

T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris. A robust and scalable clustering algorithm for mixed type attributes in large database environment. *Proceedings of the seventh ACM SIGKDD*, 2001.

S. Engela, D. Hutcheon, S. Bishop, L. Buchmann, J. Caggiano, M. Chatterjee, A. Chen, J. D'Auria, D. Gigliotti, U. Greife, D. Hunter, A. Hussein, C. Jewett, A. Laird, M. Lamey, W. Liu, A. Olin, D. Ottewell, J. Pearson, C. Ruiz, G. Ruprecht, M. Trinczek, C. Vockenhuber, and C. Wrede. Commissioning the DRAGON facility at ISAC. *Nuclear Instruments and Methods in Physics Research*, A553, 2005.

L. García-Escudero, A. Gordaliza, R. San Martín, S. Van Aelst, and R. H. Zamar. Robust linear clustering. *Unpublished Manuscript*, 2007.

P. Gawrysiak, H. Rybinski, and M. Okoniewski. Regression - yet another clustering method. In *International Intelligent Information Systems Conference*, Advances in Soft Computing Journal. Springer-Verlag, 2001.

J. Harrington and M. Salibian-Barrera. birch: Working with very large data sets. URL `http://www.stat.ubc.ca/ harringt/birch/birch-jss.pdf`. Submitted to Journal of Statistical Software, 2008.

D. Hutcheon, S. Bishop, L. Buchmann, M. Chatterjee, A. Chen, J. D'Auria, S. Engel, D. Gigliotti, U. Greife, D. Hunter, A. Hussein, C. Jewett, N. Khan, M. Lamey, A. Laird, W. Liu, A. Olin, D. Ottewell, J. Rogers, G. Roy, H. Sprenger, and C. Wrede. The DRAGON facility for nuclear astrophysics at TRIUMF-ISAC. *Nuclear Physics*, A718:515–517, 2003.

S. Nassar, J. Sander, and C. Cheng. Incremental and effective data summarization for dynamic hierarchical clustering. *ACM SIGMOD*, 2004.

R. T. Ng and J. Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

S. Odewahn, S. Djorgovski, R. Brunner, and R. Gal. Data from the Digitized Palomar Sky Survey. Technical report, Dept. of Astronomy, California Institute or Technology, Pasadena, CA 91125, 1998.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

D. M. Rocke and J. Dai. Sampling and subsampling for cluster analysis in data mining: With applications to sky survey data. *Data Mining and Knowledge Discovery*, 7:215–232, 2003.

P. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection*. Wiley, 1987.

P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

P. J. Rousseeuw and K. van Driessen. Computing LTS regression for large data sets. *Data Mining and Knowledge Discovery*, 12:29–45, 2006.

P. J. Rousseeuw and K. van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, August 1999.

P. J. Rousseeuw and B. C. Van Zomeren. Unmasking multivariate outliers and leverage points. *Journal of the American Statistical Association*, 85(411):633–639, 1990.

M. Salibian-Barrera, G. Willems, and R. Zamar. Computation of tau-estimates for linear regression. Unpublished Manuscript, 2007.

A. K. H. Tung, X. Xu, and B. C. Ooi. CURLER: finding and visualizing nonlinear correlation clusters. *ACM SIGMOD*, 2005.

S. Van Aelst, X. Wang, R. H. Zamar, and R. Zhu. Linear Grouping Using Orthogonal Regression. *Computational Statistics & Data Analysis*, 50(5):1287–1312, Mar. 2006.

W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proc. 24th Int. Conf. Very Large Data Bases*, 1998.

H. Wedekind. On the selection of access paths in a data base system. In J. Klimbie and K. Koffeman, editors, *Data Base Management*. Amsterdam: North-Holland, 1974.

T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference*, Proceedings of the 1996 ACM SIGMOD international conference, 1996.

Chapter 4

Conclusion

This chapter contains an outline of the results obtained in the thesis, the problems that we encountered, and the directions we foresee for future work.

Chapter 2 - Extending linear grouping analysis:

- We reviewed the Linear Grouping criterion (Van Aelst et al. (2006)), as implemented in the R package `lga`, and discussed two requirements for LGA to work. The first requirement was with respect to the smallest eigenvalues of the covariance matrix of a cluster. In particular, the multiplicity of the smallest eigenvalue had to be one in order for the associated eigenvector to be uniquely defined and orthogonal regression to be applicable. One common situation where the multiplicity of the smallest eigenvalue would be greater than one was in the case where there were fewer observations than dimensions within a cluster, a situation that was relevant to following sections. The second requirement was that the cluster must be linear in shape.

- We then introduced a substitute measure, called Partially Orthogonal Regression, for the situation where the first requirement had been violated due to more dimensions than observations. In particular, we considered two clusters that had multiplicity issues, and provided a measure for determining to which cluster an observation (which was not part of either cluster) belonged. It was made up of two distance measures, the first measuring distance in the basis where the eigenvalues were positive, and the second measuring how far away an observation was to a cluster in the dimensions where the eigenvalues were zero.

- We then relaxed the second requirement by introducing linear grouping within a feature space, via an explicit or implicit transformation to linearity. In order to do this, we leveraged off Kernel PCA methodology and the Kernel Trick. One side effect of these transformations, however, was that typically the dimension of the space increased dramatically (especially with implicit transformation), such that the number of observations was less than the number of dimensions. Therefore, a POD approach in the feature space was provided along with an orthogonal regression approach in the feature space.

- It remains for further study what the best tactic would be for the selection of kernel functions, as well as their tuning, though this is a problem prevalent in a lot of scholarship associated with the use of Kernel Methods. A good, motivating, example would also be of some value.

Chapter 3 - Finding Approximate Solutions to Combinatorial Problems with Very Large Data Sets using BIRCH:

- We reviewed the algorithm BIRCH, as originally discussed in Zhang et al. (1997), which was first used as an approximate method for clustering data in the case where the data set did

not fit in the available physical memory. This approach created a summary of the data set comprised of compact subclusters of observations, each with associated statistics that allowed for the calculation of an objective function of combinations of the subclusters, and without needing to refer back to the original data set.

- The nature of combinatorial problems were then discussed more generally, in particular how the data set needed to be referred to a number of times during optimization, and how the size of the solution space meant that the problem gets more difficult as the dimensions and/or number of observations increased. The application of BIRCH was then generalized to a particular class of combinatorial problem where the objective function could be calculated in terms of the clustering features, which was a much simpler optimization problem than optimizing the objective function on the whole data set.

- We then extended BIRCH to three challenging optimization problems to do with robust statistics, being the Minimum Covariance Determinant estimate for location and dispersion, Least Trimmed Squares estimate for building linear models, and RLGA - the robust equivalent to LGA, and showed its efficacy on real data sets, and for the first two methods, simulation studies as well. In all cases, the BIRCH approach yielded at least equivalent, and in many cases improved, solutions to the original algorithms.

- While we demonstrated how the selection of parameters for BIRCH did not materially affect the results for each algorithm, especially in the cases where optional refinement steps occurred, further research in the selection of parameters might provide additional insight into further improvement. Lastly, other examples of difficult optimization problems could benefit from an implementation within the BIRCH framework, if practical.

Bibliography

S. Van Aelst, X. Wang, R. H. Zamar, and R. Zhu. Linear Grouping Using Orthogonal Regression. *Computational Statistics & Data Analysis*, 50(5):1287–1312, Mar. 2006.

T. Zhang, R. Ramakrishnan, and M. Livny. *BIRCH: A New Data Clustering Algorithm and Its Applications*, volume 1 of *Data Mining and Knowledge Discovery*, pages 141 – 182. Springer Netherlands, June 1997.

Appendix A

Proofs and Derivations for Chapter 2

The proof of Result 2.2.1

Orthogonal regression is the technique in which we select a hyperplane such that the orthogonal distances between the observations and the hyperplane are minimized. We define the hyperplane by $\mathcal{H}(\mathbf{a}, b) = \{\mathbf{y} : \mathbf{a}^\top \mathbf{y} = b\}$. The orthogonal distance between an observation $\mathbf{y}$ and hyperplane $\mathcal{H}$ is then given by

$$d_O^2(\mathbf{y}, \mathcal{H}) = (\mathbf{a}^\top \mathbf{y} - b)^2$$

We have observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Then minimizing the squared orthogonal distance yields

$$\begin{aligned}
\min_{\substack{||\mathbf{a}||=1,\\ b}} \sum_{i=1}^{n} \left(\mathbf{a}^\top \mathbf{x}_i - b\right)^2 &= \min_{\substack{||\mathbf{a}||=1,\\ b}} \sum_{i=1}^{n} \left(\mathbf{a}^\top \mathbf{x}_i - \mathbf{a}^\top \bar{\mathbf{x}} + \mathbf{a}^\top \bar{\mathbf{x}} - b\right)^2 \\
&= \min_{\substack{||\mathbf{a}||=1,\\ b}} \sum_{i=1}^{n} \left\{ \left(\mathbf{a}^\top \mathbf{x}_i - \mathbf{a}^\top \bar{\mathbf{x}}\right)^2 - 2\left(\mathbf{a}^\top \mathbf{x}_i - \mathbf{a}^\top \bar{\mathbf{x}}\right)\left(\mathbf{a}^\top \bar{\mathbf{x}} - b\right) \right. \\
&\qquad \left. + \left(\mathbf{a}^\top \bar{\mathbf{x}} - b\right)^2 \right\} \\
&= \min_{\substack{||\mathbf{a}||=1,\\ b}} \sum_{i=1}^{n} \left(\mathbf{a}^\top \mathbf{x}_i - \mathbf{a}^\top \bar{\mathbf{x}}\right)^2 + n\left(\mathbf{a}^\top \bar{\mathbf{x}} - b\right)^2 \\
&= \min_{\substack{||\mathbf{a}||=1,\\ b}} \mathbf{a}^\top \mathbf{S}\mathbf{a} + n\left(\mathbf{a}^\top \bar{\mathbf{x}} - b\right)^2.
\end{aligned}$$

It is clear that the second term is minimized by setting $b = \mathbf{a}^\top \bar{\mathbf{x}}$. To find the minimum of the first term,

$$\min_{||\mathbf{a}||=1} \mathbf{a}^\top \mathbf{S}\mathbf{a} = \min_{||\mathbf{a}||=1} \mathbf{a}^\top \mathbf{E}\mathbf{\Lambda}\mathbf{E}^\top \mathbf{a}$$

where $\mathbf{E} \in \mathbb{R}^{d\times d}$ is a matrix with the eigenvectors of $\mathbf{S}$ as columns, and $\mathbf{\Lambda} \in \mathbb{R}^{d\times d}$ a diagonal matrix with the corresponding eigenvalues on the diagonal. Let $\mathbf{w} = \mathbf{E}^\top \mathbf{a}$. Then

$$\begin{aligned}
\min_{||\mathbf{a}||=1} \mathbf{a}^\top \mathbf{E}\mathbf{\Lambda}\mathbf{E}^\top \mathbf{a} &= \min_{||\mathbf{w}||=1} \mathbf{w}^\top \mathbf{\Lambda}\mathbf{w} \qquad \text{as } ||\mathbf{w}|| = ||\mathbf{a}|| \\
&= \min_{||\mathbf{w}||=1} \sum_{i=1}^{d} w_i^2 \lambda_i \geq \min_{||\mathbf{w}||=1} \lambda_d \sum_{i=1}^{d} w_i^2 = \lambda_d.
\end{aligned}$$

Finally, by substitution, this lower bound is achieved when $\mathbf{a} = \mathbf{e}_d$.

■

The rationale for Definition 2.3.1

Using Equation (2.3) as the starting point, we need to calculate

$$d_P^2(\mathbf{y}; \mathbf{E}, \boldsymbol{\mu}) = d_E^2(\mathbf{y}, \mathbf{p}) + d_S^2(\mathbf{p}, \tilde{\mathbf{y}})$$

Start with $d_E^2(\mathbf{y}, \mathbf{p})$ – it can be shown that the distance between $\mathbf{y}$ and its projection $\mathbf{p}$ on the basis $\mathbf{E} = (\mathbf{e}_1 | \dots | \mathbf{e}_\nu)$ going through $\boldsymbol{\mu}$ is equal to the distance between $(\mathbf{y} - \boldsymbol{\mu}) = \mathbf{z}$ to the basis $\mathbf{E}$, going through the origin. Let $\mathbf{p}^*$ be the projection of $\mathbf{z}$ onto $\mathbf{E}$ (going through the origin), given by

$$\mathbf{p}^* = \sum_{i=1}^{\nu} (\mathbf{e}_i \cdot \mathbf{z}) \mathbf{e}_i = \mathbf{E}\mathbf{E}^\top \mathbf{z}. \tag{A.1}$$

Therefore,

$$\mathbf{z} - \mathbf{p}^* = \mathbf{z} - \mathbf{E}\mathbf{E}^\top \mathbf{z} = (\mathbf{I} - \mathbf{E}\mathbf{E}^\top)\mathbf{z}.$$

Let $\mathbf{P}_1 = \mathbf{I} - \mathbf{E}\mathbf{E}^\top$, and note that $\mathbf{P}_1$ is idempotent and symmetric. Then

$$\begin{aligned} d_E^2(\mathbf{y}, \mathbf{p}) = d_E^2(\mathbf{z}, \mathbf{p}^*) &= (\mathbf{z} - \mathbf{p}^*)^\top (\mathbf{z} - \mathbf{p}^*) \\ &= \mathbf{z}^\top \mathbf{P}_1^\top \mathbf{P}_1 \mathbf{z} = \mathbf{z}^\top \mathbf{P}_1 \mathbf{z}. \end{aligned}$$

Now for $d_S^2(\mathbf{p}, \tilde{\mathbf{y}})$ – this can be thought of as the distance between $\mathbf{p}$ and its projection $\tilde{\mathbf{y}}$ on the basis $\tilde{\mathbf{E}} = (\mathbf{e}_1 | \dots | \mathbf{e}_{\nu-1})$ going through $\boldsymbol{\mu}$, which is equal to the distance between $\mathbf{p}^*$ and its projection $\tilde{\mathbf{y}}^*$ on the basis $\tilde{\mathbf{E}} = (\mathbf{e}_1 | \dots | \mathbf{e}_{\nu-1})$ going through the origin. Therefore,

$$\tilde{\mathbf{y}}^* = \tilde{\mathbf{E}}\tilde{\mathbf{E}}^\top \mathbf{p}^*$$

and

$$\mathbf{p}^* - \tilde{\mathbf{y}}^* = \sum_{i=1}^{\nu} (\mathbf{e}_i \cdot \mathbf{z}) \mathbf{e}_i - \sum_{i=1}^{\nu-1} (\mathbf{e}_i \cdot \mathbf{z}) \mathbf{e}_i = \mathbf{e}_\nu \mathbf{e}_\nu^\top \mathbf{z} \tag{A.2}$$

by substituting in the definition of $\mathbf{p}^*$ from equation (A.1). Finally, in the article, we defined the scaled distance $d_S^2(\mathbf{a}, \mathbf{b})$ in a manner similar to a Mahalanobis distance, and stated that the direction which needs to be scaled is given by λ_ν. Instead of simply applying this directly, we instead scale the direction with the inverse of the whole covariance matrix, and show that this is indeed the case.

$$d_S^2(\mathbf{p}, \tilde{\mathbf{y}}) = d_S^2(\mathbf{p}^*, \tilde{\mathbf{y}}^*) = (\mathbf{p}^* - \tilde{\mathbf{y}}^*)^\top \mathbf{C}^{-1} (\mathbf{p}^* - \tilde{\mathbf{y}}^*)$$

where $\mathbf{C} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^\top$ is the sample covariance in the basis $\mathbf{E}$. Then, from equation (A.2)

$$= \mathbf{z}^\top \mathbf{e}_\nu \mathbf{e}_\nu^\top \mathbf{E}\boldsymbol{\Lambda}^{-1}\mathbf{E}^\top \mathbf{e}_\nu \mathbf{e}_\nu^\top \mathbf{z} = \frac{1}{\lambda_\nu} \mathbf{z}^\top \mathbf{e}_\nu \mathbf{e}_\nu^\top \mathbf{z}$$

■

The proof of Lemma 2.4.1

Starting with the definition of $(\tilde{\mathbf{K}})_{ij} \equiv \tilde{\mathbf{K}}_{ij} = \tilde{k}(\mathbf{x}_i, \mathbf{x}_j)$ (omitting the bracket for convenience)

$$\begin{aligned}
\tilde{\mathbf{K}}_{ij} &= \tilde{\Phi}(\mathbf{x}_i)^\top \tilde{\Phi}(\mathbf{x}_j) \\
&= \left(\Phi(\mathbf{x}_i) - \frac{1}{n}\sum_k \Phi(\mathbf{x}_k)\right) \cdot \left(\Phi(\mathbf{x}_j) - \frac{1}{n}\sum_l \Phi(\mathbf{x}_l)\right) \\
&= \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) - \Phi(\mathbf{x}_i)\frac{1}{n}\sum_l \Phi(\mathbf{x}_l) - \frac{1}{n}\sum_k \Phi(\mathbf{x}_k)\cdot\Phi(\mathbf{x}_j) + \frac{1}{n^2}\sum_k \Phi(\mathbf{x}_k)\sum_l \Phi(\mathbf{x}_l) \\
&= \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) - \frac{1}{n}\sum_l \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_l) - \frac{1}{n}\sum_k \Phi(\mathbf{x}_k)\cdot\Phi(\mathbf{x}_j) + \frac{1}{n^2}\sum_k\sum_l \Phi(\mathbf{x}_k)\cdot\Phi(\mathbf{x}_l) \\
&= \mathbf{K}_{ij} - \frac{1}{n}\sum_l \mathbf{K}_{il} - \frac{1}{n}\sum_k \mathbf{K}_{kj} + \frac{1}{n^2}\sum_k\sum_l \mathbf{K}_{kl} \qquad \text{(A.3)}
\end{aligned}$$

Expressing this as matrix operations yields the result.

■

The proof of Result 2.4.2

We are interested in finding the solution to

$$\lambda\mathbf{e} = \mathbf{C}_\Phi\mathbf{e} \qquad \text{(A.4)}$$

where

$$\mathbf{C}_\Phi = \frac{1}{n}\sum_{i=1}^n \tilde{\Phi}(\mathbf{x}_i)\tilde{\Phi}(\mathbf{x}_i)^\top. \qquad \text{(A.5)}$$

Substituting equation (A.5) into (A.4) yields

$$\lambda\mathbf{e} = \sum_{i=1}^n \left(\tilde{\Phi}(\mathbf{x}_i)^\top\mathbf{e}\right)\tilde{\Phi}(\mathbf{x}_i).$$

This means that all solutions of $\mathbf{e}$ lie in the span of $\tilde{\Phi}(\mathbf{x}_i)$. Therefore

$$\mathbf{e} = \sum_{i=1}^n \alpha_i\tilde{\Phi}(\mathbf{x}_i) \qquad \text{(A.6)}$$

for some scalars $\alpha_1, \ldots, \alpha_n$. Substituting equations (A.5) & (A.6) into equation (A.4), and multiplying by $\tilde{\Phi}(\mathbf{x}_k)$ yields

$$\lambda\tilde{\Phi}(\mathbf{x}_k) \cdot \left(\sum_{i=1}^{n} \alpha_i \tilde{\Phi}(\mathbf{x}_i)\right) = \tilde{\Phi}(\mathbf{x}_k) \cdot \left(\frac{1}{n}\sum_{j=1}^{n} \tilde{\Phi}(\mathbf{x}_j)\tilde{\Phi}(\mathbf{x}_j)^\top\right)\left(\sum_{i=1}^{n} \alpha_i \tilde{\Phi}(\mathbf{x}_i)\right)$$

$$\lambda \sum_{i=1}^{n} \alpha_i \tilde{\Phi}(\mathbf{x}_k) \cdot \tilde{\Phi}(\mathbf{x}_i) = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n} \alpha_i \left(\tilde{\Phi}(\mathbf{x}_k) \cdot \tilde{\Phi}(\mathbf{x}_j)\right)\left(\tilde{\Phi}(\mathbf{x}_j) \cdot \tilde{\Phi}(\mathbf{x}_i)\right)$$

$$\lambda \sum_{i=1}^{n} \alpha_i \tilde{K}_{ki} = \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{n} \alpha_i \tilde{K}_{kj}\tilde{K}_{ji}$$

Generalizing this yields

$$n\lambda\tilde{\mathbf{K}}\boldsymbol{\alpha} = \tilde{\mathbf{K}}\tilde{\mathbf{K}}\boldsymbol{\alpha}. \tag{A.7}$$

It can be shown (though currently omitted) that the solutions λ and $\boldsymbol{\alpha}$ for the equation

$$n\lambda\boldsymbol{\alpha} = \tilde{\mathbf{K}}\boldsymbol{\alpha}$$

are the same as those for the equation (A.7). Finally, to make these eigenvectors orthonormal, we calculate the norm

$$\mathbf{e}_i \cdot \mathbf{e}_i = \sum_{j=1}^{n}\sum_{k=1}^{n} \alpha_i^j \alpha_i^k \tilde{\Phi}(\mathbf{x}_j)^\top \tilde{\Phi}(\mathbf{x}_k)$$

where α_i^j is the j-th element of $\boldsymbol{\alpha}_i$

$$\begin{aligned} &= \sum_{j=1}^{n}\sum_{k=1}^{n} \alpha_i^j \alpha_i^k \tilde{K}_{jk} \\ &= \boldsymbol{\alpha}_i \cdot \tilde{\mathbf{K}}\boldsymbol{\alpha}_i = n\lambda_i \boldsymbol{\alpha}_i \cdot \boldsymbol{\alpha}_i \\ &= n\lambda_i \end{aligned}$$

Therefore, for $\mathbf{e}_i$ to be orthonormal, divide $\boldsymbol{\alpha}_i$ by $\sqrt{n\lambda_i}$. Obviously, if $\mathbf{e}_i \cdot \mathbf{e}_j = 0$ then so too will be $n\sqrt{\lambda_i\lambda_j}\,\mathbf{e}_i \cdot \mathbf{e}_j$ (i.e. they remain orthogonal).

■

The proof of Result 2.4.3

From Result 2.4.2, we have

$$\mathbf{e}_k = \frac{1}{\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \alpha_k^j \tilde{\Phi}(\mathbf{x}_{S[j]})$$

and $\mathbf{x}_{S[j]}$ is the observation associated with the j-th element of the set S. Then the constant term, $\mathbf{e}_k \cdot \bar{\mathbf{y}}_s$, is given by:

$$\begin{aligned}
\mathbf{e}_k \cdot \bar{\mathbf{y}} &= \frac{1}{\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \alpha_k^j \underbrace{\left(\Phi(\mathbf{x}_{S[j]}) - \bar{\Phi}\right)}_{\tilde{\Phi}(\mathbf{x}_{S[j]})} \cdot \overbrace{\left(\frac{1}{m}\sum_{i\in S} \Phi(\mathbf{x}_i)\right)}^{\bar{\mathbf{y}}_s} \\
&= \frac{1}{\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \sum_{i\in S} \alpha_k^j \left(\frac{1}{m}\Phi(\mathbf{x}_{S[j]}) \cdot \Phi(\mathbf{x}_i) - \frac{1}{m^2}\sum_{l\in S} \Phi(\mathbf{x}_l) \cdot \Phi(\mathbf{x}_i)\right) \\
&= \frac{1}{m^2\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \sum_{i=1}^{m} \alpha_k^j \left(m K_{ji}^S - \sum_{l=1}^{m} K_{li}^S\right) \\
&= \frac{1}{m^2\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \alpha_k^j \left(m \sum_{i=1}^{m} K_{ji}^S - \sum_{i=1}^{m}\sum_{l=1}^{m} K_{li}^S\right)
\end{aligned}$$

Let $\mathbf{k}_i^S$ be a vector, with $\mathbf{k}_i^S = \left((\mathbf{K})_{S[1],i}\ (\mathbf{K})_{S[2],i}\ \ldots\ (\mathbf{K})_{S[m],i}\right)^\top$ (as defined in Section 2.4.2 on pg. 30) and $\mathbf{1}$ is a vector of length m containing all ones. Then

$$\mathbf{e}_k \cdot \bar{\mathbf{y}} = \frac{1}{m^2\sqrt{\tilde{\lambda}_k}} \boldsymbol{\alpha}_k^\top \left(m\mathbf{K}^S\mathbf{1} - \mathbf{1}^\top\mathbf{K}^S\mathbf{1}\right) \tag{A.8}$$

Similarly, the first term is given by

$$\begin{aligned}
\mathbf{e}_k \cdot \mathbf{y}_i &= \frac{1}{\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \alpha_k^j \left(\Phi(\mathbf{x}_{S[j]}) - \bar{\Phi}\right) \cdot \Phi(\mathbf{x}_i) \\
&= \frac{1}{\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \alpha_k^j \left(\Phi(\mathbf{x}_{S[j]}) \cdot \Phi(\mathbf{x}_i) - \frac{1}{m}\sum_{l\in S} \Phi(\mathbf{x}_l) \cdot \Phi(\mathbf{x}_i)\right) \\
&= \frac{1}{\sqrt{\tilde{\lambda}_k}} \sum_{j=1}^{m} \alpha_k^j \left(K_{S[j]i} - \frac{1}{m}\sum_{l\in S} K_{li}\right)
\end{aligned}$$

and writing this in vector notation

$$= \frac{1}{\sqrt{\tilde{\lambda}_k}} \boldsymbol{\alpha}_k^\top \left(\mathbf{k}_i^S - \frac{1}{m}\mathbf{1}^\top\mathbf{k}_i^S\right) \tag{A.9}$$

■

The proof of Result 2.4.4

Starting with Definition 2.3.1, and expanding the bracket

$$\begin{aligned} d_P^2(\mathbf{y}; \mathbf{E}, \bar{\mathbf{y}}_s) &= \mathbf{z}^\top (\mathbf{P}_1 + \mathbf{P}_2/\lambda_k)\mathbf{z} \\ &= \mathbf{z}^\top \mathbf{z} + \frac{1}{\lambda_k}\mathbf{z}^\top \mathbf{e}_k \mathbf{e}_k^\top \mathbf{z} - \mathbf{z}^\top \mathbf{E}\mathbf{E}^\top \mathbf{z} \end{aligned} \tag{A.10}$$

where $\mathbf{z} = \mathbf{y} - \bar{\mathbf{y}}_s$. In order to express this in vector notation let $\mathbf{z}_i = \mathbf{y}_i - \bar{\mathbf{y}}_s$, and in the remainder of this proof we tackle each of the elements of this equation.

In general,

$$\begin{aligned} \mathbf{z}_i^\top \mathbf{z}_j &= (\mathbf{y}_i - \bar{\mathbf{y}}_s)^\top (\mathbf{y}_j - \bar{\mathbf{y}}_s) \\ &= \left(\Phi(\mathbf{x}_i) - \frac{1}{n_s} \sum_{k \in S} \Phi(\mathbf{x}_k) \right) \cdot \left(\Phi(\mathbf{x}_j) - \frac{1}{n_s} \sum_{l \in S} \Phi(\mathbf{x}_l) \right) \\ &= \mathbf{K}_{ij} - \frac{1}{m} \sum_{l \in S} \mathbf{K}_{il} - \frac{1}{m} \sum_{k \in S} \mathbf{K}_{kj} + \frac{1}{m^2} \sum_{k \in S} \sum_{l \in S} \mathbf{K}_{kl} \end{aligned}$$

from equation (A.3). Expressing this in matrix form for all $\mathbf{z}_i^\top \mathbf{z}_j, (i, j \in \{1, \ldots, n\})$ yields

$$\tilde{\mathbf{K}}^* = \mathbf{K} - \frac{1}{m}\mathbf{1}_m \cdot \mathbf{K}^F - \frac{1}{m}\mathbf{K}^F \cdot \mathbf{1}_m + \frac{1}{m^2}(\mathbf{1}_m \cdot \mathbf{K}^S) \cdot \mathbf{1}_m,$$

and the diagonal elements of this matrix give the required result.

The second element of equation (A.10) can be found by applying Result 2.4.3, and recognizing that $\lambda_k = \tilde{\lambda}_k/m$ from Result 2.4.2. Finally, the third element can be found by generalizing this equation for all values of $\boldsymbol{\alpha}_i, i \in 1, \ldots, k$.

■

Appendix B

birch: Working with Very Large Data Sets*

B.1 Introduction

Dealing with large data sets in R can be difficult and time consuming, especially when the size of the data set is larger than the available computer memory. The first difficulty arises because when the size of a data set exceeds available physical memory, paging/swapping is required, which typically slows down the computer performance dramatically. An additional problem becomes apparent when one needs to find good approximate solutions to combinatorial problems for very large data sets. More specifically, we are concerned with computing the Minimum Covariance Determinant estimator (MCD, Rousseeuw and Leroy, 1987) for multivariate observations, the Least Trimmed Squares regression estimator (LTS, *ibid*), the clustering provided by the Linear Grouping Analysis algorithm (LGA, Van Aelst et al., 2006), and its robust version (RLGA, García-Escudero et al., 2007). The existing approximating algorithms for these problems require multiple passes over the entire data set, and thus the effect of paging and swapping is greatly exacerbated, to the point of making them unfeasible. Furthermore, note that due to the large size of the solution space, these algorithms typically consider a number of data-driven random starts that are iteratively improved. Obtaining good approximate solutions requires a very large number of random starting points. The number of random starts increases rapidly with the sample size, so that even for data sets that fit in memory, these algorithms may not be able to find a good approximate solution in a reasonable amount of time.

One strategy to work with large data sets is provided by a pre-processing algorithm called BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies), first introduced in Zhang et al. (1996), which summarizes the data using a tree with a relatively small number of subclusters and associated "sufficient statistics". These can be used to easily calculate sample statistics on subsets of the data without having to access the data set. This tree is built with one pass of the data, and only a single observation needs to be held in memory at any time.

The focus of this article will be describing the implementation of the package **birch** in R, referring the reader to Harrington and Salibian-Barrera (2007) (herein referred to as the original article) for a more in depth discussion of the motivation and advantages of this approach. This package also provides methods for these BIRCH trees that compute very good approximated MCD estimators, LTS Regression estimators, and LGA and RLGA clusterings. Additional methods are also described for performing basic operations on trees such as subsetting, plotting and basic summary statistics. Finally, we provide some examples using a commonly analyzed data set, which will demonstrate

*A version of this chapter has been submitted for publication. Harrington, J. and Salibian-Barrera, M. (2008) birch: Working with Very Large Data Sets, Journal of Statistical Software.

the syntax and the efficacy of this approach.

B.1.1 BIRCH

A simplified description of this algorithm is given next, with the reader referred to Zhang et al. (1996) for further details of the complete algorithm, including refinements such as self-sizing and subcluster splits. Let $\mathbf{x}_1, \ldots, \mathbf{x}_N$ denote the sample to be processed, where $\mathbf{x}_j \in \mathcal{R}^d$. The algorithm creates subsets of the observations, with $\mathcal{L}_J$ denoting the index of observations in the J-th subcluster. In this implementation, the clustering features (CF) are given by the tuple:

$$CF_J = \left(\sum_{i \in \mathcal{L}_J} 1, \sum_{i \in \mathcal{L}_J} \mathbf{x}_i, \sum_{i \in \mathcal{L}_J} \mathbf{x}_i \mathbf{x}_i^\top \right) = (n_J, LS_J, SS_J)\,,$$

i.e. the number of observations in the leaf, the sum of those observations, and their "sum of squares". It was shown in the original article that in each of the clustering and robust algorithms below the above clustering features are sufficient statistics for the respective objective functions.

The algorithm processes the observations sequentially as follows:

1. The first observation $\mathbf{x}_1$ forms its own subcluster.
2. Let $\mathcal{T}$ be the current set of subclusters, and let $\bar{\mathbf{x}}_t$, and CF_t, $t \in \mathcal{T}$ be the associated subcluster centres and clustering features.
3. For each of the remaining observations $\mathbf{x}_2, \ldots, \mathbf{x}_N$:
   (a) Calculate the distance from $\mathbf{x}_j$ to each of the centres $\bar{\mathbf{x}}_t$ of the existing subclusters. Select the closest centre $\bar{\mathbf{x}}_{t_0}$.
   (b) If the observation meets the "closeness" and "compactness" criteria described below, then $\mathbf{x}_j$ joins the t_0 subcluster, and both $\bar{\mathbf{x}}_{t_0}$ and CF_{t_0} are updated.
   (c) If not, a new subcluster $t+1$ containing $\mathbf{x}_j$ is formed, with $\bar{\mathbf{x}}_{t+1} = \mathbf{x}_j$ and CF_{t+1} the corresponding clustering features, and the list of subclusters is updated: $\mathcal{T} = \mathcal{T} \cup \{\mathbf{x}_j\}$

Once this algorithm is complete, we have a "list" of compact subclusters that are locally similar, with associated summary statistics of the underlying observations making up each subcluster.

Note that this algorithm requires the calculation of the distance between $\mathbf{x}_j$ and every subcluster centre. This can be avoided by using more efficient structures, such as a CF-tree, which is similar to a "B+"-tree (Wedekind, 1974). In this approach, each observation is passed to a root node, with a number of non-leaf nodes as children. These non-leaf nodes contain the sample mean of all observations beneath them. The observation is then simply passed down this tree to the closest non-leaf node, and each centre is updated as the observation is given to it. Finally, when it reaches the appropriate subcluster, it is either absorbed, or else forms its own subcluster. Once a non-leaf node exceeds a (user-specified) number of subclusters, it splits with the subclusters allocated to the split non-leaf nodes based on a proximity measure. A graph of this structure is given in Figure B.1.

The number of subclusters that are formed at the end of this algorithm, denoted as N_L, depends on the criteria used to decide whether a new observation being processed can join an existing subcluster or needs to form a new cluster. Consider a point $\mathbf{x}_j$ currently being processed, and let $\bar{\mathbf{x}}_{t_0}$ be the closest subcluster centre. In order for $\mathbf{x}_j$ to join the t_0 cluster we require:

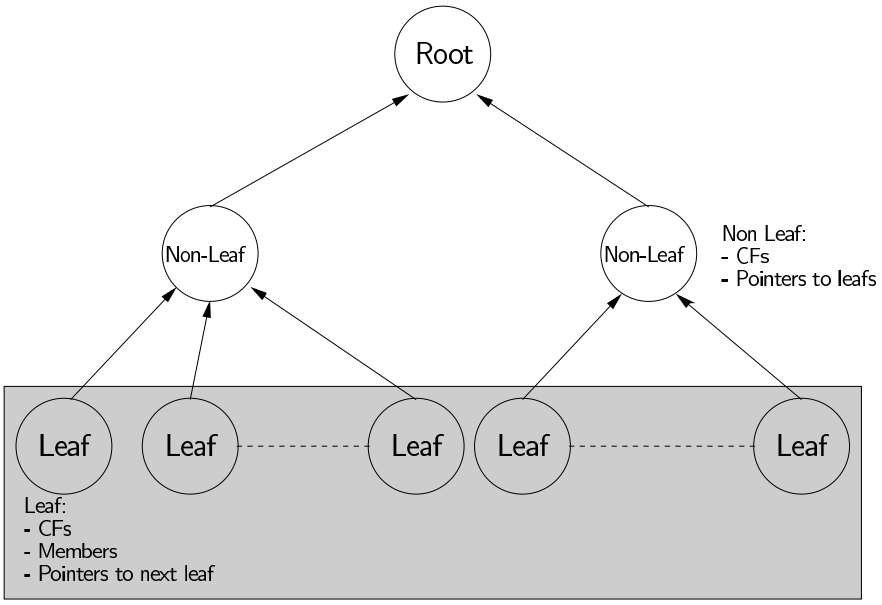


Figure B.1: The tree structure used for pre-processing in **birch**. The area shaded in grey is the data returned as the subclusters.

(i) *closeness*: $\|\mathbf{x}_j - \bar{\mathbf{x}}_{t_0}\| \leq a$ for some pre-defined $a > 0$.

(ii) *compactness*: The trace of the sample covariance matrix of the observations in the t_0 cluster *and* $\mathbf{x}_j$ cannot exceed a pre-defined constant $b > 0$.

Using both criteria is a departure from the approach in Zhang et al. (1996) which proposes using only one of the criteria; for a discussion of this, see the original paper.[1]

It should be noted that for combinatorial problems, such as k-means, etc., BIRCH is a useful approximation when the full data set cannot fit into memory, or is sufficiently large to make random-search algorithms unfeasible. The properties of this approximation will naturally depend on the level of coarseness of the tree – equivalently, the ratio of number of subclusters to number of observations in the underlying data set. However any other non-combinatorial algorithm, such as ordinary least squares, that can use the CF-tuple summary statistics, are doubly advantageous, since in these cases it is possible to calculate the solution exactly from the CF summaries.

B.2 Package birch: Implementation in R

The package **birch** is available on CRAN[2] and written in C/C++ and R (R Development Core Team, 2007). The command for creating a BIRCH tree is given by the syntax

[1] In the implementation of this package, the user is required to set both criteria. However, if only one is desired, then the other criterion can be set to a very high number, thereby not affecting the outcome.

[2] This article describes version 1.1-2 of the package.

```
birchOut <-  birch (x, radius, compact = radius, keeptree=FALSE, ...)
```

where `x` is either an R matrix, a text file name, or any "connection" (in the R sense) to the original data set that is compatible with `read.table`. The arguments `radius` and `compact` specify the "closeness" and "compactness" criteria, respectively. If `x` is referring to a file name, then the `...` argument allows the user to furnish additional information regarding the kind of data set, and is consistent with the syntax for `read.table` (e.g. `sep=","` for comma-separated variable files). The advantage of creating the tree directly from a file or a connection is that, at no time, is the complete data set held in memory.[3]

If the argument `keeptree` is set to `TRUE`, then the CF-tree created is kept in memory. The following commands

```
birch.addToTree(x, birchObject, ...)
birchOut <- birch.getTree(birchObject)
birch.killTree(birchObject)
```

can be used to add data to the tree, retrieve the data from the tree, and remove the tree from memory. Clearly this is advantageous when additional data is to be added or when dealing with real-time data streams, though the user should be aware that, until such time as the tree is removed from memory, there are effectively two copies of the subclusters and their associated CFs and member lists in memory.

The code for creating a BIRCH tree is primarily written in C/C++, with the R API used for fetching the data. All other code (e.g. `covMcd.birch`, `lts.birch`, etc.) is written entirely in R. The structure of the object produced by `birch` (herein referred to as a `birch` object) is given in Table B.1.

B.2.1 Generic Internal Methods

Various generic methods have been implemented for use on a `birch` object. They are:

```
print, summary, dim, length, [ ], cbind, plot, pairs, points, lines
```

The `print` method produces a brief description of the object, in particular the number of subclusters, the attributes used for creating the object, and the dimensions of the original data set. The `summary` method produces the column means and covariance of the whole data set, and can be assigned to an object (comprised of a list with names `means` and `cov`) if desired. The method `dim` produces a vector of length three, containing the tuple: Number of observations in the underlying data set; number of dimensions of the data set; and the number of subclusters in the object, while the method `length` simply produces the number of subclusters in the object. This last method is useful when building a tree to a certain number of subclusters, as is often the case (motivation for this approach is given in the original article).

Subsets of a `birch` object can be created using the standard R notation `[i, j]`, with `i` specifying which subclusters to retain, and `j` which columns. However, care should be used when subsetting by column, as

[3] By design, R will process the data in batches of at most 100,000 observations.

Data:

Name	*Description*	*Dimension*
`sumXi`	A matrix containing the sum of the observations, $\sum \mathbf{x}_i$, in each subcluster	$N_L \times d$
`sumXisq`	An array containing the sum of squares of the observations, $\sum \mathbf{x}_i \mathbf{x}_i^\top$, in each subcluster	$d \times d \times N_L$
`N`	A vector containing the number of observations in each subcluster	$N_L \times 1$
`members`	A list containing which observations are in each subcluster.	N_L

Attributes:

Name	*Description*
`xcolnames`	The column names of the original data set.
`xdim`	A vector containing the number of observations in the original data set, the number of columns in the original data set, and the number of subclusters.
`compact`	The value for compact used when forming the tree.
`radius`	The value for radius used when forming the tree.
`internal`	A pointer to the tree held in memory (if applicable).

Table B.1: The structure of a `birch` object. N_L is the number of subclusters in the object.

```
birch(z, 0.1)[,1:2]
```

will not yield the same result as

```
birch(z[,1:2], 0.1)
```

as the Euclidean distances and trace criteria mentioned previously will produce differing results when calculating `birch` on the whole data set, versus on a subset of the columns. Furthermore, data that is close (in the Euclidean sense) in d dimensions, may not be close in $e > d$ dimensions, particularly when $e - d$ is large. In order to reflect this, the `compact` and `radius` attributes will be set to `NA` in the former case. The method `cbind` is useful in the case where a constant is to be put in front of each observation of the data set, and updates the `sumXi` and `sumXisq` data. However, once again, care should be used with this command, as

```
birch(cbind(1,z), 0.1)
```

will not yield the same result as

```
cbind(1, birch(z, 0.1))
```

for the same reasons as those given when subsetting by column.[4] Once again, in the latter case, the `compact` and `radius` attributes will be set to `NA`.

We have two plotting methods for `birch` objects: `plot` for two-dimensional data; and `pairs` for data with more than two dimensions. Each method has an optional argument, `centers`, with the

[4]The Euclidean distances will be the same, but the trace of the covariance matrix will be different.

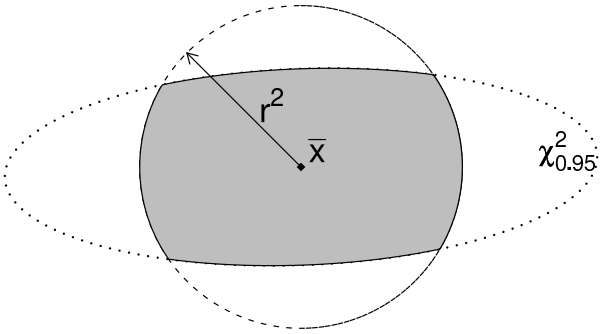


Figure B.2: A new plot method. The shaded area represents the ellipse that is produced when `centers=FALSE`.

default set to `FALSE`. In the default operation, the methods produce 95% ellipses[5] based on the covariance matrix for each subcluster with $N > d$, centred at its sample mean, and is very effective at visualizing the data when there are a reasonable number of subclusters. However, in order to reflect the fact that each observation within a subcluster is within a certain Euclidean distance of the centre of the subcluster, as specified by the `radius` attribute, the ellipses are truncated at this distance. A graphical representation of this is given in Figure B.2. For subclusters with $N \leq d$ observations, only the centre of the subcluster is plotted.

When the number of subclusters is large this plot becomes less informative, and setting the argument `centers = TRUE` produces a plot of just the sample means of each subcluster. These methods accept the usual plotting arguments, such as `col` argument for assigning a different colour to each subcluster.

B.2.2 Clustering

Even when the data set fits in memory, clustering with BIRCH can be highly advantageous for large data sets, as clustering with subclusters rather than the full dimensional data set is a much simpler combinatorial problem. In this case, the above approximate solution can easily be refined to provide a solution that is in many cases better than the one found by attempting direct optimization over the full data set. For an extended discussion on this, refer to the original article.

Two clustering methods are provided: `kmeans.birch` and `lga.birch` (`rlga.birch` is discussed in the following section).[6] Each of these algorithms take a `birch` object as input, and in the simplest case the number of clusters to find. Their syntax is given by

```
kmeans.birch (birchObject, centers, nstart = 1)
lga.birch (birchObject, k, nsamp=100)
```

where `birchObject` is a `birch` object. For k-means, `centers` is either (a) the number of clusters to

[5]The ellipses are produced using the package **ellipse**, which uses a multivariate normal reference distribution.
[6]Non-BIRCH versions of `lga` and `rlga` are available on CRAN in the package **lga**.

find, or (b) a matrix of k rows and d columns containing the initial seed to use, and `nstart` is the number of random starts to use. For LGA, `k` specifies the number of clusters to find, and `nsamp` the number of random starting positions. Both methods return a list containing the clusters found, in terms of subclusters and the actual observations making up these subclusters, as well as the value of the objective function: the Residual Sum of Squares (RSS) in the case of `kmeans.birch`, and the Residual Orthogonal Sum of Squares (ROSS) for `lga.birch`.

An alternative approach for k-means is provided via the command `dist.birch`, which calculates the pairwise distance between the centres of the subclusters and produces a `dist` object, which can be used in conjunction with the `hclust` and `cuttree` commands to provide an initial seed for the `kmeans` algorithm. An example of this is given in the help files, and in Section B.3.

B.2.3 Robust Statistics

Three methods are implemented in this package for robust statistics: `covMcd.birch`, `ltsReg.birch` and `rlga.birch`. These methods are the main focus of the original article, and so will not be discussed in detail here.[7]

The syntax for all three methods is given by

```
covMcd.birch(birchObject, alpha=0.5, nsamp=100)
ltsReg.birch(birchObject, alpha=0.5, intercept=FALSE, nsamp=100)
rlga.birch(birchObject, k, alpha=0.5, nsamp=100)
```

where `alpha` is the size of the h-subset (between 0.5 and 1) as a percentage of the whole data set, and `nsamp` is the number of starting positions for the random search. For the LTS command, the argument `intercept` adds a unit vector to the data set via the `cbind` mechanism given in Section B.2.1.[8] The values returned from each algorithm are given in Table B.2.

Two methods are also provided for optional "refinement steps" for MCD and LTS when the data set fits in memory. As mentioned previously (and elsewhere – see the original article), the solution using BIRCH is approximate, in particular because of the "coarseness" of the underlying tree. If a more accurate solution is required and the data set fits in memory, then an additional step can be taken whereby the solution from the BIRCH algorithm is used to seed additional iterations (local refinements) using the whole data set. The syntax to do this is given by

```
covMcdBirch.refinement (covOut, x, alpha=0.5)
ltsBirch.refinement (ltsOut, x, y, alpha=0.5, intercept=FALSE)
```

where `covOut` and `ltsOut` are the outputs from `covMcd.birch` and `lts.birch` respectively. If the data set was not loaded into R because it was loaded directly from a file or connection, then these methods are not applicable.

[7] Further information on each of the underlying algorithms can be found in Rousseeuw and van Driessen (1999), Rousseeuw and van Driessen (2006), and García-Escudero et al. (2007) (for RLGA). The non-BIRCH versions of the first two methods can be found in **robustbase**, and in **lga** for RLGA.

[8] Reiterating the comment in Section B.2.1, the `cbind` mechanism is only approximate w.r.t. arguments. A (better) alternative is to form a `birch` object on the data set with the unit vector added as the first column. However, at times this is not possible, and this mechanism is then applicable.

MCD:	
Name	*Description*
`zbar`	estimate of location.
`Sz`	estimate of covariance.
`Det`	the Minimum Covariance Determinant.
`best`	a list containing the index of subclusters making up the h-subset and the index of **actual** observations making up the h-subset.
LTS:	
Name	*Description*
`best`	a list containing the index of subclusters making up the h-subset and the index of **actual** observations making up the h-subset
`raw.coefficients`	the fitted LTS regression line
`Resids`	A list containing the sum of squared residuals for the best subset, as well as the sum of squared residuals for the whole data set (based on the LTS regression equation).
RLGA:	
Name	*Description*
`clust`	a list containing the clustering in terms of the subclusters and the clustering in terms of the **actual** observations
`ROSS`	the residual sum of squares of orthogonal distances to the fitted hyperplanes based on the best data set.

Table B.2: The outputs from each of the robust algorithms.

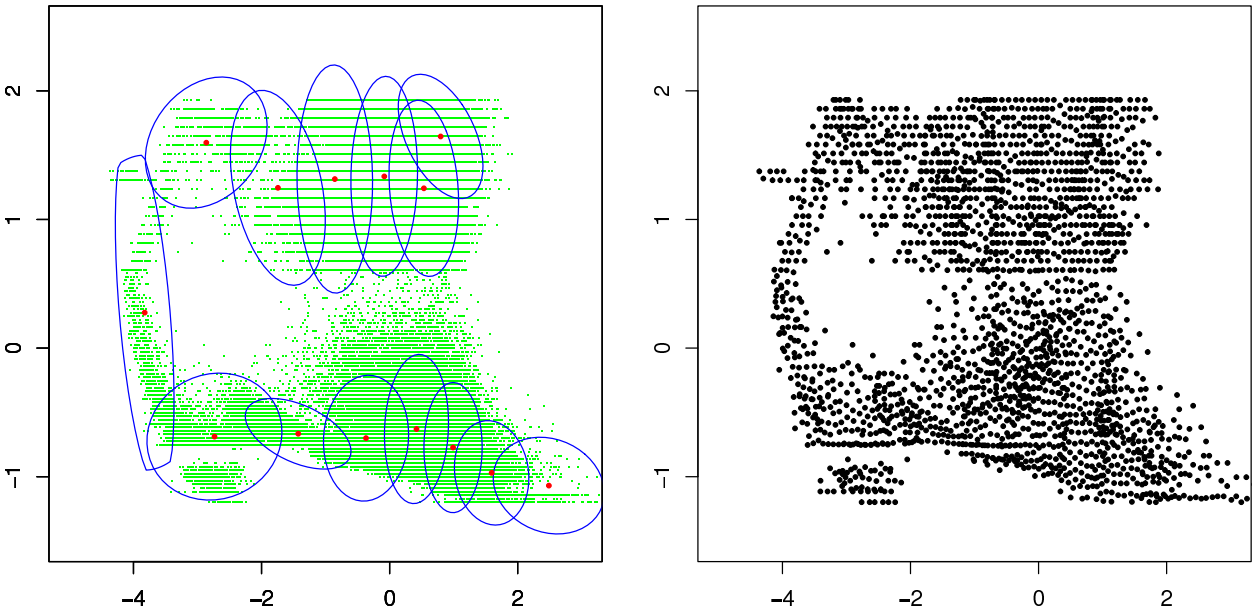
B.3 Example

We illustrate the capabilities of the package by analyzing the data set commonly referred to as the DPOSS data set.

The Digitized Palomar Sky Survey (DPOSS) data has been used in articles related to robustness (e.g. Rousseeuw and van Driessen (1999), Rousseeuw and van Driessen (2006), etc.) as well as clustering (e.g. Rocke and Dai (2003)). It contains characteristics from celestial objects, and has 132,402 observations in 33 dimensions. For further information see Odewahn et al. (1998).

As a first step, we take a subset of the data set by selecting the "Aperture" and "Combined stellar function" variables (scaled such that the sample mean is zero, and variance one) for just the F band, yielding $p = 2$ and $n = 132,402$. This data is in the file `"dposs_2d.csv"`, and in order to demonstrate the functionality, it is loaded directly from a local file in the first instance, and a web server in the second case. We then produce two trees – the first with radius set to 1.5 and compactness equal to 5, and a second with both radius and compactness set to 0.05. These attributes were set for the purposes of visualization. Plots of these are given, along with the original data set, in Figure B.3.

```
R> library(birch)
```



(a) Radius = 1.5, Compactness = 5, Nleafs = 14

(b) Radius = Compactness = 0.005, Nleafs = 2,283

Figure B.3: The `birch` plot method, applied on `birch` objects of differing size. The first plot contains a plot of the ellipses (blue), centres (red), and the underlying data (green). The second plot just contains the centres.

```
R> mytree2 <- birch("dposs_2d.csv", compact=5, radius=1.5, sep=",",
+                   header=TRUE)
R> length(mytree2)
[1] 14

R> ## the object z is a matrix containing the data in dposs_2d.csv
R> plot(z, col='green', pch=".")
R> lines(mytree2, col='blue');
R> points(mytree2, col='red')

R> mytree <- birch("http://www.stat.ubc.ca/~harringt/dposs_2d.csv",
+                  0.005, sep=",", header=TRUE)
R> length(mytree)
[1] 2283
R> plot(mytree, centers=TRUE)
```

We then perform the MCD on a different subset of the DPOSS data set, this time selecting the "Aperture" and "Combined stellar function" for each of the F, J and N bands and just those observations classified as stars yielding $p = 6$ and $n = 55,716$. The full DPOSS data set is stored in the R object `dposs`. The following code forms a `birch` object, and calculates the MCD and a refinement.

```
R> ## Create data set
R> z <- scale(dposs[dposs[,"Class"] == 1,
+                   c("MAperF","csfF","MAperJ","csfJ","MAperN","csfN")])

R> ## Create Birch Object
R> mytree <- birch(z, 0.1)
R> length(mytree)
[1] 6192

R> ## Calculate MCD
R> set.seed(1234)
R> abc <- covMcd.birch(mytree, alpha=0.5, nsamp=100)
R> summary(abc, digits=3)
CovMcd using birch
MCD =  7.566695e-10
Estimate of center
[1] -0.130 -0.291 -0.121 -0.270 -0.133 -0.540
Estimate of dispersion
        [,1]     [,2]    [,3]     [,4]    [,5]     [,6]
[1,]  0.4580 -0.06658  0.4380 -0.09068  0.4768 -0.02701   # etc...

R> log(abc$Det)
[1] -21.00209
R> abcRef <- covMcdBirch.refinement(abc, z, alpha=0.5)
R> ## Once again, summary works with this object (not shown here)

R> log(abcRef$Det)
[1] -21.24701
```

For reference, the equivalent result using `covMcd` from the package **robustbase** yields a log(MCD) of -21.24578, which is slightly higher (i.e. less optimal) than the one found by refining the BIRCH approximate solution.

The LTS algorithm assumes that the object provided is made up of the exploratory and response variables, much in the same way as `lsfit`.

```
R> x <- z[,2]
R> y <- z[,1]

R> mytree3 <- birch(cbind(x,y), 0.002); length(mytree3)
[1] 1818

R> set.seed(1234)
R> abc<- lts.birch(mytree3, intercept=TRUE, alpha=0.5, nsamp=100)
R> summary(abc)
LTS using birch
LTS =  1591.565
```

```
Coefficients
[1] -1.470226 -4.232881

R> abcRef <- ltsBirch.refinement(abc, x, y, alpha=0.5, intercept=TRUE)
R> summary(abcRef)
LTS using birch
LTS =  1495.513
Coefficients
       x1        x2
-1.501868 -4.331047
```

For reference, the equivalent results using `ltsReg` from the package **robustbase** yields a RSS of 1495.893, which again is slightly worse than the result produced by refining the BIRCH approximate solution.

In order to demonstrate RLGA we construct an outlier map (Rousseeuw and Van Zomeren, 1990) by plotting the Mahalanobis distances using standard estimates against Mahalanobis distances using robust distances. We then apply RLGA-BIRCH to this data set as a means of robustly identifying the two groups.

```
R> library(robustbase)
R> set.seed(1234)
R> z <- scale(dposs[, c("MAperF","csfF","MAperJ","csfJ","MAperN","csfN")])
R> a <- covMcd(z)

R> ## create the data set
R> robdist <- mahalanobis(z, colMeans(z[a$best,]), cov(z[a$best,]))
R> classdist <- mahalanobis(z, colMeans(z), cov(z))
R> x <- scale(cbind(classdist, robdist), center=FALSE)

R> mytree <- birch(x, 0.001, 0.001); length(mytree)
[1] 2042
R> set.seed(1234)
R> b <- rlga.birch(mytree, k=2, alpha=0.75)
```

In order to plot this result in the first two dimensions, we use the following code, with the result given in Figure B.4

```
R> plot(mytree, centers=TRUE, col=b$clust$sub+1)
```

Finally, we provide a small demonstration of the implementation of `kmeans.birch`. Using the tree formed for the RLGA example, we first use `kmeans.birch` directly.

```
R> abc <- kmeans.birch(mytree, 2, nstart = 100)
R> abc$RSS
[1] 145512.2
```

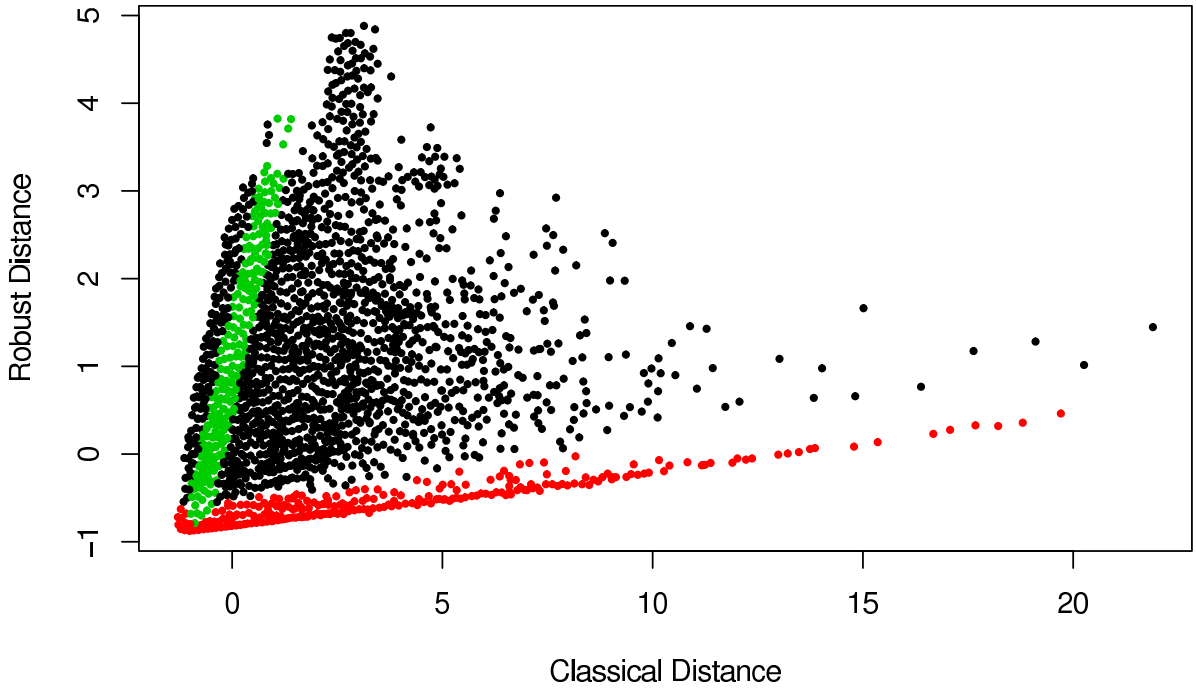


Figure B.4: The results from RLGA.

```
R> ## Add a refinement step (using the k-means algorithm directly)
R> centers <- rbind(summary(mytree[abc$clust$sub == 1,])$mean,
+                   summary(mytree[abc$clust$sub == 2,])$mean)
R> ghi <- kmeans(x, centers=centers)
R> sum(ghi$withinss)
[1] 145429.4
```

Then, running the non-BIRCH version for comparison.

```
R> ## Run the non-BIRCH version for comparison
R> def <- kmeans(x, 2)
R> sum(def$withinss)
[1] 145429.4

R> ## A crosstab of the clustering outputs for kmeans and kmeans.birch
R> table(def$cluster, abc$clust$obs)
     1     2
1   650 87162
2 44177   413

R> ## A crosstab of the clustering outputs for kmeans and
R> ## kmeans.birch with refinement step
R> table(def$cluster, ghi$clust)
```

```
      1     2
1     0 44590
2 87812     0
```

i.e. the result from `kmeans.birch` with a refinement step is identical to that provided by `kmeans` directly.

Following this, we use the command `dist.birch` with `hclust` (the hierarchical clustering algorithm) in order to suggest starting centers, and then complete one iteration with these centers as seeds.

```
R> hc <- hclust(dist.birch(mytree), method="ward")
R> ## Find which "clusterings" correspond with two groups
R> memb <- cutree(hc, k = 2)

R> ## Form the centers based on these
R> centers <- rbind(summary(mytree[memb == 1,])$mean,
+                   summary(mytree[memb == 2,])$mean)

R> ghi <- kmeans.birch(mytree, centers)
R> ghi$RSS
[1] 145533.9

R> ## A crosstab of the clustering
R> table(ghi$clust$obs, def$clust)
    1     2
1 87639  1089
2   173 43501
```

B.4 Conclusion

In this article we have introduced the package **birch**, and described its implementation in R. While most of the justification for this algorithm is provided elsewhere, in particular in Harrington and Salibian-Barrera (2007), the benefits mentioned previously are two-fold: firstly, in the case of very large data sets, a compact summary of the data can be calculated without needing to load the complete data set into memory, and secondly that for combinatorial problems, the effective solution space is significantly reduced, and an approximate solution can easily be computed using the clustering features.

The first benefit is clearly provided by functionality in the `birch` command for building a tree by loading a data set directly from text file, URL or connection, and at no time is the whole data set held in memory. Clearly, therefore, the only limiting factor in terms of size is that the tree doesn't get too large, which in turn is controlled directly through the selection of the compactness and radius criteria. To our knowledge, this is the only approach that has this feature that is capable of solving combinatorial problems like the ones given here.

In demonstrating the second benefit, we refer to a summary of the results in Section B.3, given in Table B.3. The first thing that is apparent is that the BIRCH results are performing very

Minimum Covariance Determinant	Dimension of combinatorial problem	Obj. Function
Non-BIRCH	$55,716 \times 6$	-21.246
BIRCH without refinement	$6,192 \times 6$	-21.002
BIRCH with refinement	$6,192 \times 6$	-21.247
Least Trimmed Squares	Dimension of combinatorial problem	Obj. Function
Non-BIRCH	$55,716 \times 2$	1,495.893
BIRCH without refinement	$1,818 \times 2$	1,591.565
BIRCH with refinement	$1,818 \times 2$	1,495.513
k-means	Dimension of combinatorial problem	Obj. Function
Non-BIRCH	$132,402 \times 2$	145,429.4
BIRCH without refinement	$2,024 \times 2$	145,512.2
BIRCH with refinement	$2,024 \times 2$	145,429.4

Table B.3: A selection of results in the Example section.

well compared with the non-BIRCH version even without the refinement steps, in spite of the fact that this is intended to be an "approximate" solution. Indeed, examples can be found – and some are given in the original article – where the nature of the combinatorial problem and reduced solution space allows BIRCH to achieve results that are superior to the non-BIRCH algorithms. Furthermore, when refinement is applied, the results perform as well as the non-BIRCH version. While these results are sufficient to suggest the merit of this approach, when combined with the fact that these methods are easily scaled with respect to the number of observations, and to a lesser extent the dimension, it is clear that where large data sets are involved, the BIRCH algorithm is an approach worth considering.

Bibliography

L. García-Escudero, A. Gordaliza, R. San Martín, S. Van Aelst, and R. H. Zamar. Robust linear clustering. *Unpublished Manuscript*, 2007.

J. Harrington and M. Salibian-Barrera. Finding approximate solutions to combinatorial problems with very large data sets using BIRCH. URL `http://www.stat.ubc.ca/ harringt/birch/birch.pdf`. Submitted to Special Issue on Statistical Algorithms and Software of Computational Statistics and Data Analysis, 2007.

S. Odewahn, S. Djorgovski, R. Brunner, and R. Gal. Data from the Digitized Palomar Sky Survey. Technical report, Dept. of Astronomy, California Institute or Technology, Pasadena, CA 91125, 1998.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

D. M. Rocke and J. Dai. Sampling and subsampling for cluster analysis in data mining: With applications to sky survey data. *Data Mining and Knowledge Discovery*, 7:215–232, 2003.

P. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection*. Wiley, 1987.

P. J. Rousseeuw and K. van Driessen. Computing LTS regression for large data sets. *Data Mining and Knowledge Discovery*, 12:29–45, 2006.

P. J. Rousseeuw and K. van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, August 1999.

P. J. Rousseeuw and B. C. Van Zomeren. Unmasking multivariate outliers and leverage points. *Journal of the American Statistical Association*, 85(411):633–639, 1990.

S. Van Aelst, X. Wang, R. H. Zamar, and R. Zhu. Linear Grouping Using Orthogonal Regression. *Computational Statistics & Data Analysis*, 50(5):1287–1312, Mar. 2006.

H. Wedekind. On the selection of access paths in a data base system. In J. Klimbie and K. Koffeman, editors, *Data Base Management*. Amsterdam: North-Holland, 1974.

T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference*, Proceedings of the 1996 ACM SIGMOD international conference, 1996.