Applications of convex hulls to outlier detection

Reece Hewitt

January 2024

This piece of work is a result of my own work and I have complied with the Department's guidance on multiple submission and on the use of AI tools. Material from the work of others not involved in the project has been acknowledged, quotations and paraphrases suitably indicated, and all uses of AI tools have been declared.

Contents

Many thanks go to my project supervisor Andrew Wade, he was of great help in furthering my knowledge within the area, providing feedback and directing me to potential useful resources.

Chapter 1

Introduction

In real world data, outliers occur for many different reasons and having methods to be able to discover them without losing valuable data is an area of particular interest for statistical applications. There is several methods available to detect them and through this dissertation we aim to present and contrast a few of those. We focus around methods involving convex hulls, these methods have applications which have more versatility than some more traditional methods as we will discuss later.

We begin by introducing the ideas of what a convex hull is alongside some useful computational tools, alongside exploring what we mean by outliers. We then progress onto methods to find convex hulls and discuss computational speeds of each. Then we seek to take these algorithms and make use of them for the purpose of outlier detection, then we compare this against a more common method for outlier detection, before discussing some issues with our convex hull methods. Finally, we introduce a different application of convex hulls that helps remedy our previous issues and show how we can expand the ideas to apply to more circumstances.

Code used for the various outlier methods discussed will be available on my GitHub username ReeceHewitt

Chapter 2

Basic definitions

Within this chapter we aim to provide the basic foundations from which we will develop our ideas of outlier detection from, to do this we must discuss three areas in turn. We begin with gaining an understanding of convex sets and hulls, then providing some foundations for computation and comparing algorithms and finally trying to develop our ideas of what outliers are.

2.1 Convexity

Before we can begin understanding what a convex hull is, we must first begin with a definition of a *convex set*:

Definition 2.1.1 *A set S is convex if for any two points a and b within S, all points along the straight line connecting a and b are also contained within the set.*

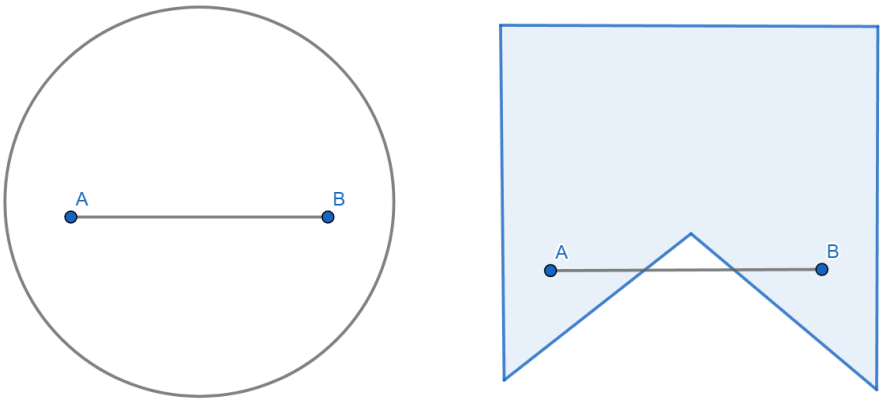

Figure 2.1: Left:Convex Right:Non-convex

Now we have our definition of a convex set we can now define what we mean by a *convex hull*:

Definition 2.1.2 *The convex hull of a set of points A is the smallest convex set that contains all of A.*

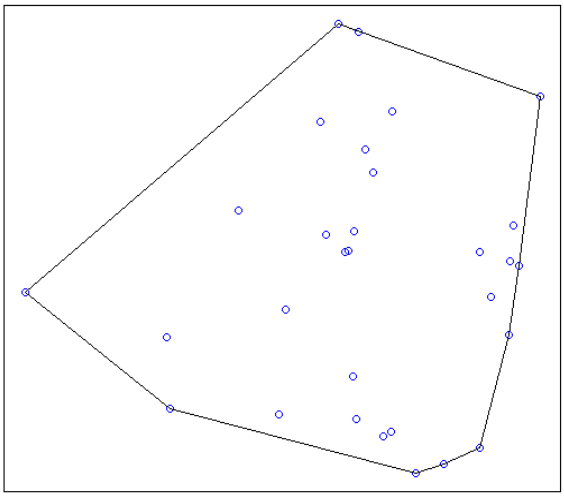

Figure 2.2: An example of a convex hull of a multivariate normal data set

2.2 Computational definitions

Throughout the rest of this dissertation we will be making comparisons between various computational algorithms, a standardised approach is required to easily quantify the speed of various algorithms, one measure we will make use of for this is big $O()$ notation. The following definition is from [1]

Definition 2.2.1 *The function f is said to be of $O(g)$, if there is a constant $c > 0$ and a number n_0 such that $f(n) \leq cg(n)$ for all $n > n_0$.*

Another method we will make use of for the purpose of comparison is the CPU computation time for any of our given functions. This method has the obvious benefit of being a very tangible comparison factor however it does not allow for comparisons easily between different users as different PC components will have an impact on the times produced. For tracking our CPU computation time we will make use of the inbuilt function in R called system.time() which works as shown below.

```
 system.time(convex_peeling(new_data),100)
user  system elapsed
0.03    0.00    0.03
```

We simply provide it with a given function to execute and a number of times to repeat the function which it then averages over and provides us with a time that we can use to compare.

2.3 Outliers

A sensible place to start is by discussing what we mean by an outlier, a variety of definitions have been suggested. One such definition is "Outliers are extreme values that differ from most other data points in a dataset" as suggested by [2], this idea of extreme values is important and is a concept we will discuss more in the following section when we attempt to find our convex hulls. Another interesting suggestion as to what we mean by outliers is "an observation that deviates so strongly from the other sample values that it is unlikely to have been generated by the same mechanism" from [7]. This idea of outliers being from some alternate mechanism is particularly of interest as it helps explain a lot of real world outliers we see, often outliers are a result of clerical error during data collection which we will aim to demonstrate in this toy example.

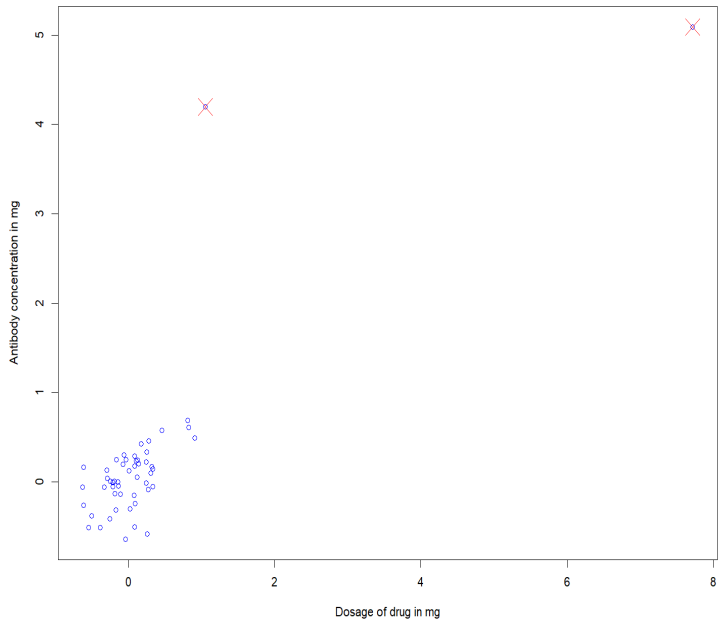

Figure 2.3: Image showing potential clerical errors highlighted by red crosses

In figure 2.3 we can clearly see two points that have potential to be outliers as indicated by the red cross marks, they do not remotely match the rest of the distribution suggesting there is some outside factor causing this. In this case the outliers can be explained by simply a misplaced decimal point resulting in a severely skewed values that if not dealt with would have severe impact on the population statistics of this example.

However, sometimes the outliers can be indicative of an entirely separate mechanism for example when we have several clusters of points within our dataset it can potentially suggest there is potentially multiple areas of interest within one set of data. In these cases we may have reason to believe that only one cluster is important to us or we may wish to separate out these clusters for individual study, both of these ideas we will return to later in section 5.

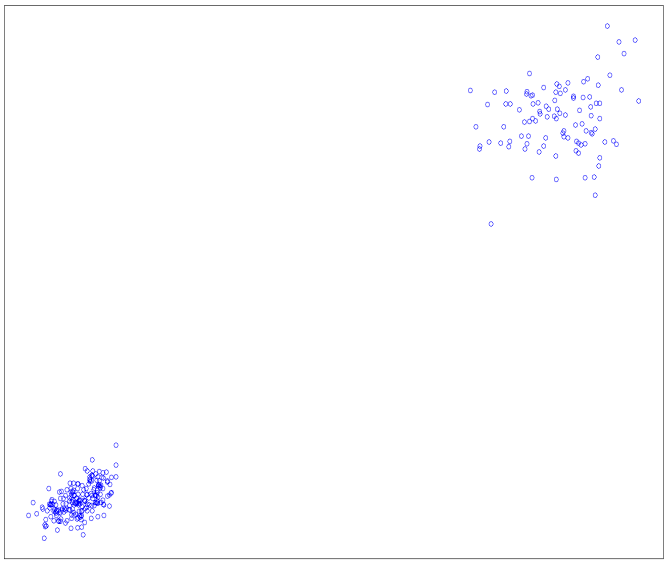

Figure 2.4: An example of a dataset with two clear separate clusters that may both warrant study

Now we have an essence of what outliers are we need to develop some methods for discovering outliers, but before we can begin explaining those methods we must first develop some methods for finding convex hulls.

Chapter 3

Algorithms to find convex hulls

3.1 Motivation

Now we have an understanding of what a convex hull is the next logical step is to establish some methods through which we can find a convex hull for some given set of points. During this chapter I present a variety of methods to find convex hulls we will also discuss the varying computational complexities, benefits and negatives of all the algorithms. The general algorithms and some definitions used in this chapter have been taken from [3], influence was also taken from [4].

3.2 Simple Brute force algorithm

We start with our simplest algorithm and we begin with a definition:

Definition 3.2.1 *A point p in a convex set S is considered extreme if no two points $a, b \in S$, such that p is on the straight line connecting a and b.*

The vertices of a convex hull C of a set S are the extreme points of S, so we want to be able to find these extreme points. Luckily, its relatively simple to test whether a given point p is extreme, we test if p falls within any triangle formed by vertices of S. If p is within any triangle then it is not an extreme point this is demonstrated in figure 3.1. There are $O(n^3)$ triangles for a set with n points and we can check whether a given point is in those triangles in linear time, we must test all n points so our basic algorithm to determine extreme points has complexity of $O(n^4)$.

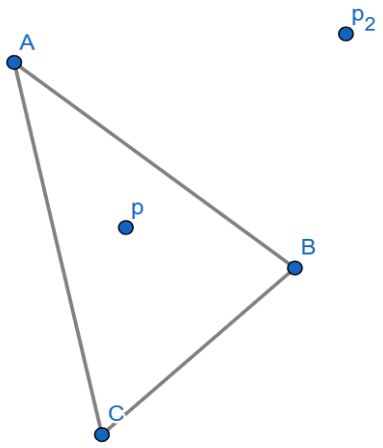

Figure 3.1: Image demonstrating our determination of extreme points with p being not extreme and p_2 being extreme

We do however encounter an issue when we connect our extreme points in the order we find them to create our convex hull as shown in figure 3.2 . This random ordering of extreme points is clearly not useful as the result is not actually our convex hull, therefore we now wish to also find a method to order our extreme points in a sensible order to produce our hull.

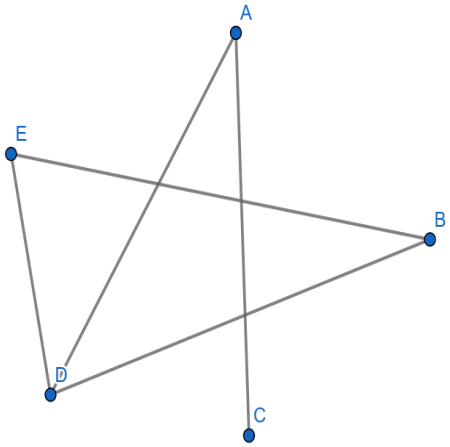

Figure 3.2: Image showing the disorder of connections between extreme points

Luckily this ordering process is relatively simple, we begin by finding the centroid of all our hull points defined as follows.

Definition 3.2.2 *The centroid C of a set of points $P = \{p1, p2, ..., pn\}$ is found by $1/n \sum_{i=1}^{n} p_i$*

Once we have found this centroid has been found we transform all hull points into polar coordinates with relation to our centroid C, now we simply order the points based on size of polar angle giving us a finished convex hull. This process does however take some time converting to polar takes $O(n)$ time especially when factored with the time complexity of finding extreme points. However, we are typically converting a much smaller number of points into polar coordinates than we consider when testing for extreme points so this has small impact on the overall computational time of this algorithm.

There is a more computationally optimised method to find the ordering of points we need not consider the actual size of the polar angle instead we consider the signed area of the triangle formed between pairs of points and our centroid. For two points p_1, p_2 and our centroid C the point p_2 forms a smaller angle with the x axis if and only if the the triangle C, p_2, p_1 has positive signed area. We can use paired comparisons of our points to find the correct ordering of our hull points. Through the use of this method our final computational complexity of our basic algorithm is $O(n^4)$, this is very slow and for attempting to find hulls of large sets is not very appropriate, however let us see if we can adapt our ideas to find a more optimum algorithm

3.3 Graham's scan

As for the simple method we seek the extreme points of our set of points, we go about finding these extreme points starting where we finished with the previous method with ordering. We begin by finding the centroid of our points, we could do this by averaging all n points though this takes $O(kn)$ time for n points and k dimensions. Luckily, Graham [5] showed that a centroid calculated from the average of any three non-collinear points is sufficient for our purposes. We can very quickly test points to determine if they are collinear, for example by determining the gradient between one point A_1 and some other point B_1 and comparing this with the gradient between some third point C and A , if the gradients are equivalent then the points are collinear and we can test a different C_1.

Now we have found our centroid point we convert all points to polar with respect to this point and order them as we discussed previously, alternatively we can use the signed area method for a more efficient solution. We have a special case to handle when we have points that have identical polar angle, logically, we choose to remove the points with smallest distance from our centroid leaving only the outermost point as this is the only point with chance of being an extreme point. For example in figure 3.4 assume C is our centroid and D, E, F are points in our set we would in this case choose to remove D and E from consideration based on distance. When calculating the distance between points we do not need to consider the square rooted value instead we can consider the absolute value, through this we can save a small amount of computation time when choosing to remove these points.

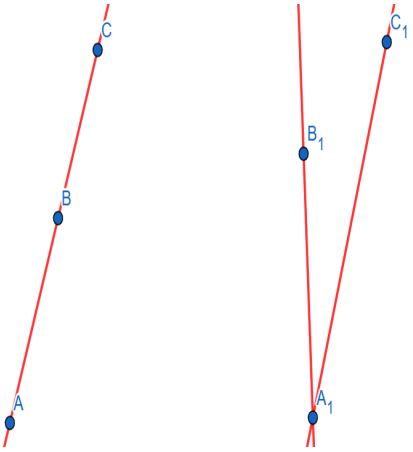

Figure 3.3: Image showing collinearity and non-collinearity

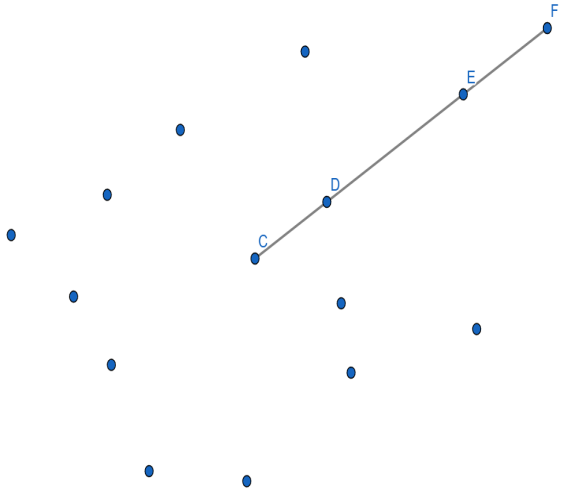

Figure 3.4: Image demonstrating our special case when points of identical polar angle, in this case we remove D and E

Now we have an ordered set of points and have potentially removed some points in the case of identical polar angle, now we begin to identify which of the remaining points are extreme. This is where we make use of the Graham's scan. We first must identify a starting point for our scan we choose to use the lowest and leftmost point as this point is definitely on our hull. We then consider sets of three consecutive ordered points beginning from our start point and the next two points. We label these points p_1, p_2, p_3 for the following demonstration of the algorithm, if the internal angle of $p_1p_2p_3$ is reflex then we define this as a left turn , if the internal angle is not reflex then we define this as a right turn as shown in figure 3.5.

If we find a left turn we progress the scan to the next point and now test p_2, p_3, p_4.

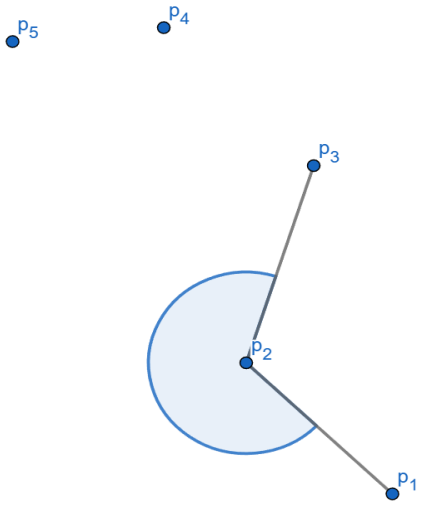

Figure 3.5: Image showing an example of a right turn with internal angle being shown in blue

If we find a right turn we remove p_2 and instead test p_1, p_3, p_4.

We repeat this process of exclusion until we have completed a full cycle of our points and returned to p_1 the points left after our process are our hull points and are ordered as required for our convex hull. By beginning by sorting our points and then performing our scan around the ordered points we have significantly improved our computation time this algorithm has $O(nlog(n))$.

3.4 Jarvis's March

Throughout this section we have focused on finding the extreme points of our given set we begin this subsection by thinking about extreme edges, these are the line segments that form our convex hull. We can define these as:

Definition 3.4.1 *A line segment formed from the connection of two points p_1, p_2 in our set S is extreme if and only if all other points lie only on one side of that line segment.*

For a given set S containing n points we know there will be $\binom{n}{2}$ edges to check against the remaining $n-2$ to determine whether any given edge is extreme. We can complete this checking process in $O(n^3)$ time which is an improvement over our original brute force algorithm but still rather slow, luckily Jarvis suggested some improvements in [6] that improve this further. These improvements revolved around the idea that if we have identified a edge connecting p_1, p_2 is extreme we can be sure that there is another extreme edge connected to p_2, with this fact we can explain Jarvis's march.

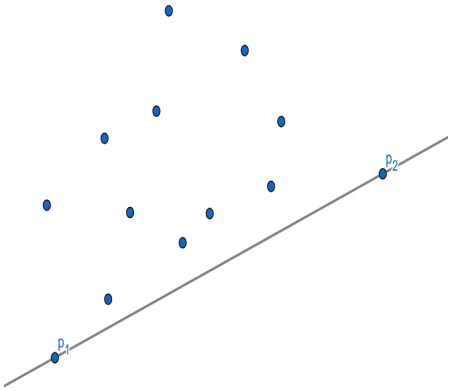

Figure 3.6: Image showing an example of an extreme edge from p_1 to p_2

We begin at the lowest point in our set p_l, we know this point must be a vertex of the hull so will have two extreme edges connected to it. To find the next counterclockwise point on our hull we find the point with smallest polar angle (≥ 0 with respect to p_l this point is the next hull vertex p_2. We now find the next point with smallest polar angle with respect to p_2 and repeat this process until we reach the highest point in our set p_h. Once we have reached this point we now can find the second half of our hull. This process is similar and begins at the highest point but instead of the point with smallest polar angle with respect to p_h and the positive x axis we instead use the negative x axis due to the fact we are now descending our hull. Part of this process is demonstrated in figure 3.7

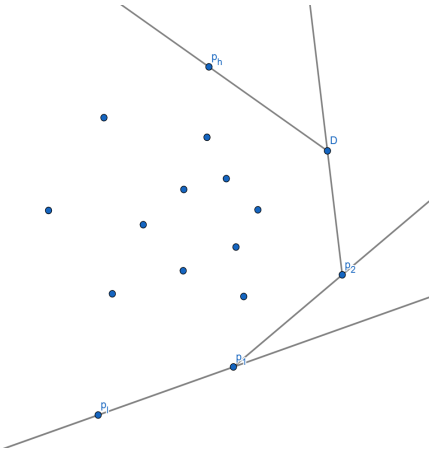

Figure 3.7: Image showing the march from lowest vertex to highest vertex

Using this algorithm we achieve a worst case time of $O(n^2)$ which is an improvement over our initial brute force method and over our checking of every line, however this is still slower than our Graham's scan. Luckily this is our worst case scenario if our hull has h vertices then we know our algorithm has $O(hn)$ time which for small enough h is a considerable improvement over any of our previously discussed algorithms. If we have reason to believe our hull will be a polygon with low number of sides this method is extremely useful to employ. We can see a comparison of our algorithms efficiencies in figure 3.8. Within this section we have focused on 2 dimensional versions of the algorithms, many of them can be extended to n dimensional cases.

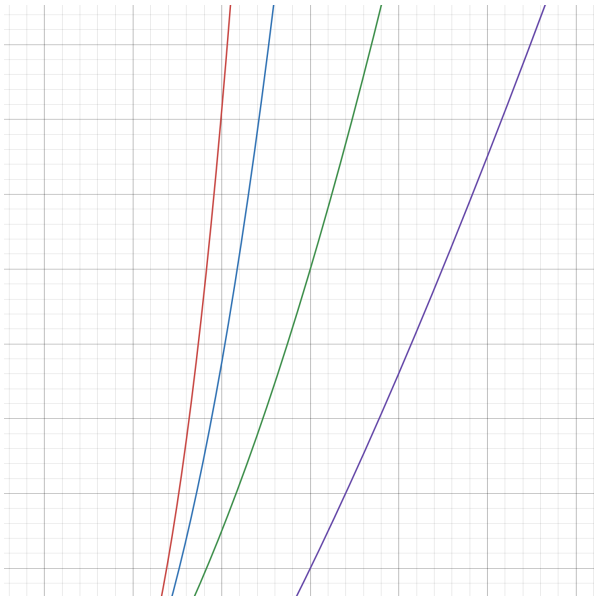

Figure 3.8: Image showing comparison of computational complexities Red: $O(n^4)$ Blue: $O(n^3)$ Green: $O(n^2)$ Purple $O(nlog(n))$

Chapter 4

Outlier detection

4.1 Uni-variate outliers

We start with discussing outlier detection and removal for uni-variate data. Traditionally, we use box plots and interquartile range to identify outliers, however for this dissertation we choose to focus on a different method. This method is called α-peeling and will be explained below.

Definition 4.1.1 *Alpha peeling is the removal of the upper and lower α fraction of an ordered set of data.*

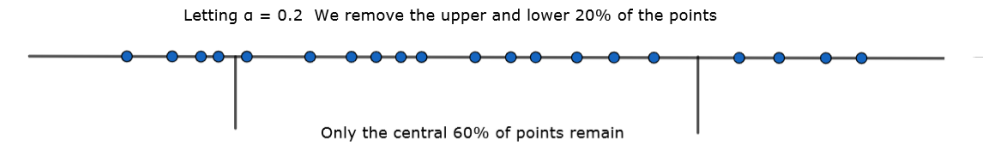

Figure 4.1: Demonstration of α-peeling

Once we have performed our α-trimming on a given data set we can then perform any parameter calculations we wish. This method is particularly useful for finding an accurate mean for a data set with several suspicious points or when there is extreme outlier points which if left in would severely impact parameter calculation.

However, there is some drawbacks from this method, if we have high variance data set by peeling we may lose this high variance in our calculated parameters. As such attempting to simulate from these parameters may not be accurate to the true distribution. This can be somewhat rectified if we test various values of α to find a good balance of outlier removal and leaving enough data to still "tell the true story".

4.2 Multi-variate outliers using α-peeling

Having a new method for handling uni-variate outliers is useful, but the vast majority of real world data we want to investigate is not uni-variate. As such it would be useful to be able to extend our ideas of α-peeling to the multi-variate case, luckily we can do exactly this using convex hulls.

For both the following demonstration of multi-variate α-peeling and for the rest of the dissertation we will focus on the bi-variate case, for the ease of visual demonstration and easier explanation of algorithms. We also make use of a toy distribution of 50 points simulated from a multivariate normal distribution.

To begin we take our data set and find the convex hull of the data. We can find the hull with any previously discussed methods, however for the demonstrations in this dissertation we make use of the built-in function in R *chull*().

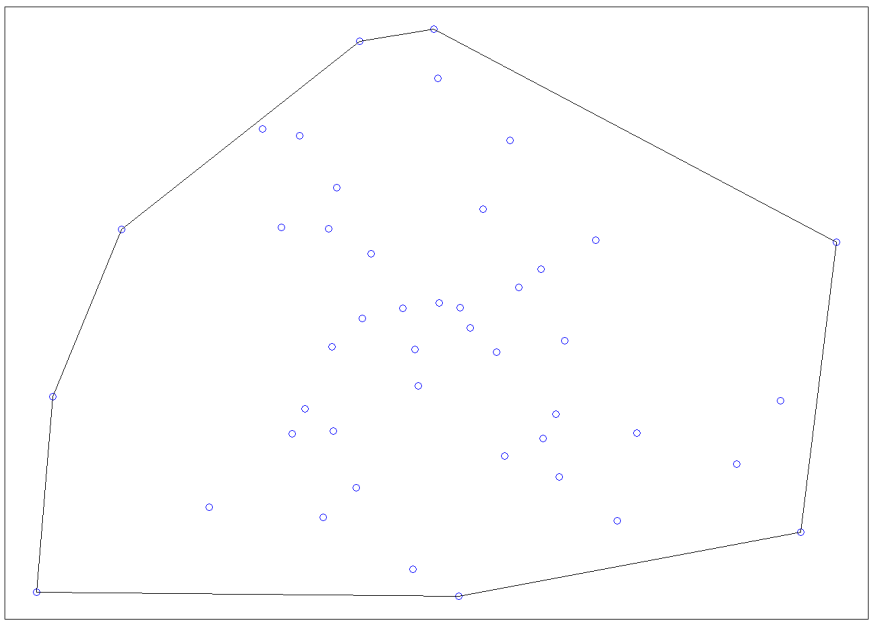

Figure 4.2: Convex hull of 50 point data set

Now we view all the points on the hull as outliers, since as they are on the

hull they are the extreme points for the data. We now remove all points on the hull from the data. Leaving us with the data below.

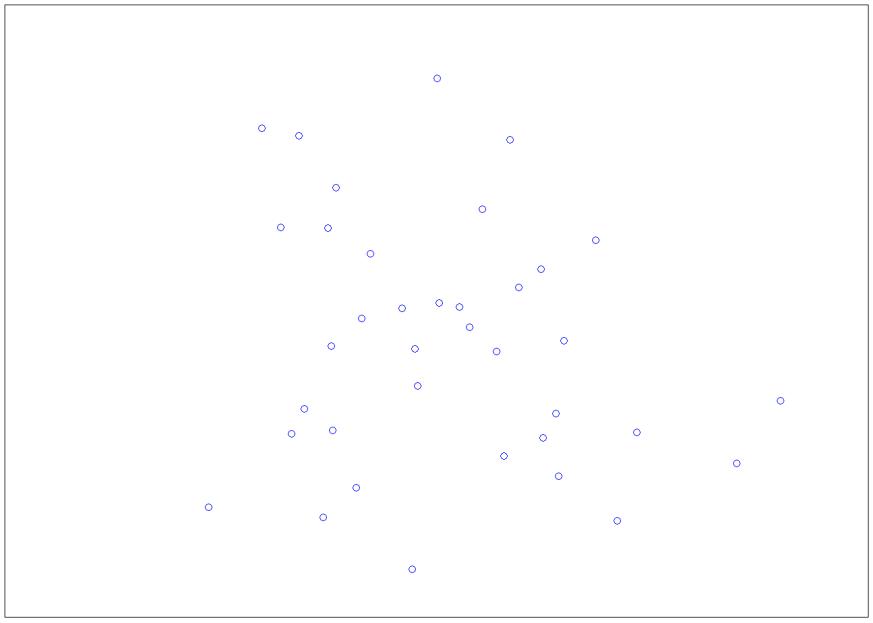

Figure 4.3: First peel of data set

To perform α-peeling we repeat this process of finding the hull and then removing points until we have removed $2\alpha n$ points for a data set with n points. We can also choose to stop this process after any number of peels this gives us a new definition.

Definition 4.2.1 *The depth of a given point p is the number of convex peelings required for p to be on the convex hull of the peeled dataset.*

To demonstrate this concept of depth we now fully peel our data set to a total of depth 6 at which point we only have one point remaining. As shown in figure 4.4.

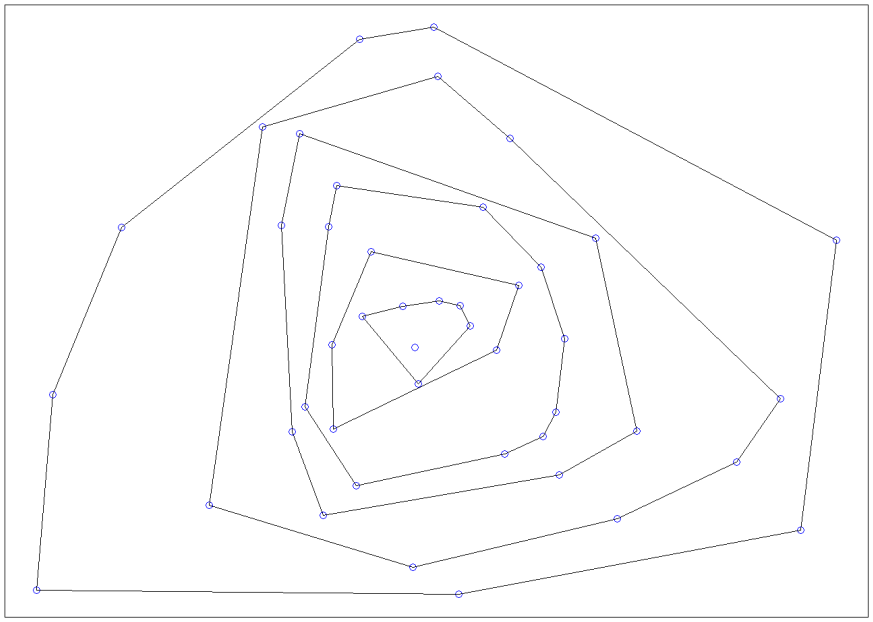

Figure 4.4: Fully peeled data set

4.3 Use of convex hulls for new data points

Once we have peeled our dataset to a level we have determined to be sufficient to remove outliers we can find the convex hull of the remaining data S. Now imagine we receive some new small set (in comparison to the size of the peeled dataset) of data, that we denote D we can make use of the hull we just found to determine whether any of these points are potential new outliers. For a given point p_1 in D if p_1 is inside the hull we can add it to our data S as a new observation, if the point does not fall inside the hull we do not add it to S.

We can determine whether a point is inside the hull based on whether it falls within any triangle formed from hull vertices in a similar process to our determination of extreme points earlier. This process allows us to very quickly add new observations without having to repeat the peeling again. However we do want the number of points we consider to be small, as if the new dataset is too large our convex peeling may no longer be appropriate to the dataset. In the case where we have a large amount of new data we should repeat the peeling process.

4.4 A more traditional approach to outlier detection

We have now developed our α-peeling to the multivariate case, it would be useful to be able to compare our method with another method for outlier detection. For this we will make use of the Mahalanobis distance as discussed in [7]. A sensible place to begin then is to define what the Mahalanobis distance is:

Definition 4.4.1 *The Mahalanobis distance for a dataset with p variables and n observations is defined as*

$$d_M^2 = (x - m)^T \Sigma (x - m)$$

where x is the matrix representing our data, Σ is the covariance matrix for our data and m is the mean of our data

This distance metric differs from the normal euclidean distance as it is influenced by the shape of the covariance matrix, this fact makes it useful for outlier detection when we have a non-uniformly shaped dataset. Under the assumption that our dataset is distributed by a multi-variate normal distribution we can make use of the fact that the Mahalanobis distance is distributed by the χ^2 distribution approximately. From this fact we can now demonstrate how to detect outliers.

We first have to make an assumption that our data in our set S is normally distributed, then we calculate the mean and covariance matrix of our data, for this we use built in functions of R. From here we can calculate the Mahalanobis distance for our set of points S as described in 4.4.1. Once we have obtained our list of distances we can compare them against a $\chi^2_{\alpha,df}$ distribution where *df* is the degrees of freedom, α is a value between 0 and 1 typically we select 0.95 or 0.975. Then if the value of any given distance is greater than the χ^2 value we identify it as an outlier and choose to remove the corresponding point. The resulting plot from this process acting on a toy dataset from a multivariate normal distribution with outliers artificially added is figure 4.5.

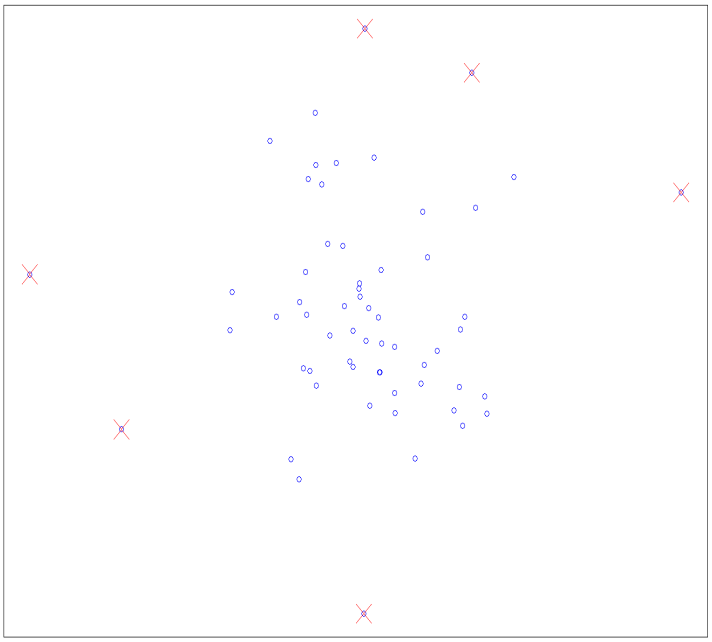

Figure 4.5: Plot showing potential outliers identified by red crosses

We do have the issue of having to have normality in our dataset for this method to work properly, this does potentially limit the datasets we can apply this method to, luckily we can test whether our set is roughly normally distributed using a QQ plot. The following explanation of how to get a QQ plot is taken directly from [7].

1. Sort the values of d_i yielding ordered values $d_(i)$, i = 1,...,n.
2. Compute the quantiles $q_i = \chi^2_{q,1-(i-0.5)/n}$, i.e. the quantiles of the χ^2_q distribution with a probability mass of $(i-0.5)/n$ to their left hand side.
3. Plot $d_(i)$ (vertical) versus q_i (horizontal).
4. Compare the plotted points to a straight line through the origin with slope equal to 1. Deviations from this line indicate deviations from a multivariate normal distribution .

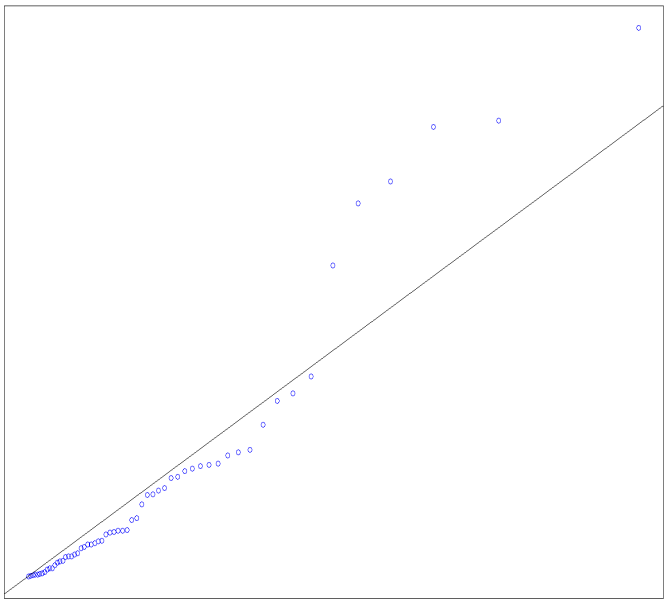

Figure 4.6: QQ plot for our data in figure 4.4.1

We can see from figure 4.6 that our data is following the line expected of a multivariate distribution however there are some points that are not following the expected trend these are resulting from the outliers we added into our data set.

4.5 Comparison of our outlier detection methods

In this subsection we will compare both of our outlier detection methods developed so far against each other. We will use a dataset generated from the following code in R to compare.

```
set.seed(100)
data <- mvrnorm(200000,rep(0,2),matrix(c(1,0,0,1),2,2))
add_data <- mvrnorm(10000,rep(0,2),matrix(c(6,0.2,0.2,7),2,2))
new_data <- rbind(data,add_data)
```

We artificially insert outliers using the add_data and use a large seeded dataset for replicability and also so the time to run our algorithm is long

enough for comparison.

First we will compare the computation time of the two methods with a large enough peeling depth to remove an approximately similar number of points.

```
system.time(peeling_depth(new_data,4,plot = FALSE),100000)
   user  system elapsed
   0.03    0.00    0.03
system.time(mahalanobis_outliers(new_data,0.95),100000)
   user  system elapsed
   0.01    0.00    0.01
```

From the code we can see that both methods have similar execution times with our Mahalanobis based method being slightly more efficient in terms of time taken to run, however there is very small differences and if we continued for enough samples or different data we may find scenarios where our peeling method is faster. We also must take into account that we are peeling to depth 4 if we adjusted the depth this speed could be far slower or much faster. For example peeling to depth 15 results in the time taken shown below.

```
system.time(peeling_depth(new_data,15,plot = FALSE),100000)
   user  system elapsed
   0.19    0.02    0.21
```

We now reduce the data set size by a factor of 100 for ease of viewing in the following comparison graphs.

Figure 4.7 shows one comparison plot between our Mahalanobis method and peeling of depth 4, with this depth we can see that both methods remove similar amounts of points and both agree on the major outlying data. Both methods however do still leave a rather large spread of our remaining data if we decided some of those points may also be outliers our peeling method requires very little tuning in order to remove more of those points. We can simply increase the peeling depth as shown in figure 4.8 where we increase the peeling depth to 8 and reduce the spread significantly.

A major issue with our Mahalanobis method is it requires normality for it to function correctly so for small data sets makes it unusable in a variety of scenarios as we will demonstrate later, whereas our peeling method does not have these downsides and can be applied to many scenarios.

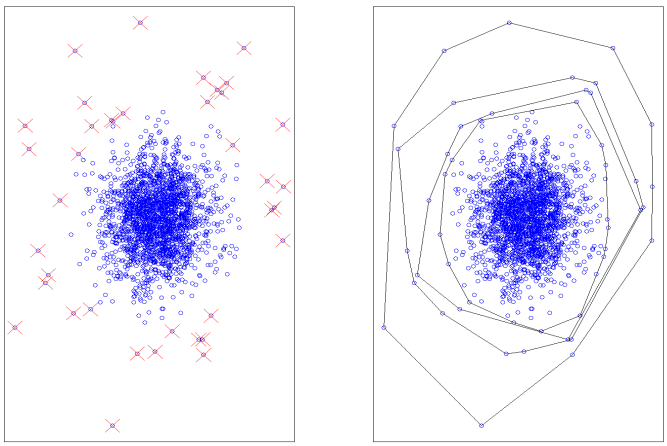

Figure 4.7: Plots comparing the Mahalanobis (Left) and peeling (Right) outlier detection methods

4.6 Problems with traditional peeling

We now discuss some issues with our normal peeling method beginning with the computational complexity to perform α-peeling with this method.

The method described in 4.2 has worst case computational complexity of $O(n^2 log(n))$. This worst case arises when the data is made up of $n/3$ nested triangles as we have to perform the maximum number of peels possible for a dataset of size n. This worst case scenario however can be reduced to $O(n^2)$, figure 4.9 showcases a visual comparison of the $O()$ times. The method to obtain this faster computation time will be explained below.

Our more efficient method of finding the fully peeled dataset as well as finding the depth of a given point requires an application of the Jarvis's march as described in section 3.4.

The following explanation is adapted from [3]:

We begin by introducing a dummy point with coordinates equal to the bottom most and left most point in our set and label this point p_0. We now perform our Jarvis's march and the first point we find on our march we label as Start. We then continue our march until we find ourselves back at our Start vertex and we have now found our depth one hull. We remove all points on this hull except the point directly before Start. This vertex becomes our new p_0 from which we can continue our process until no points remain. This is illustrated in 4.10 below.

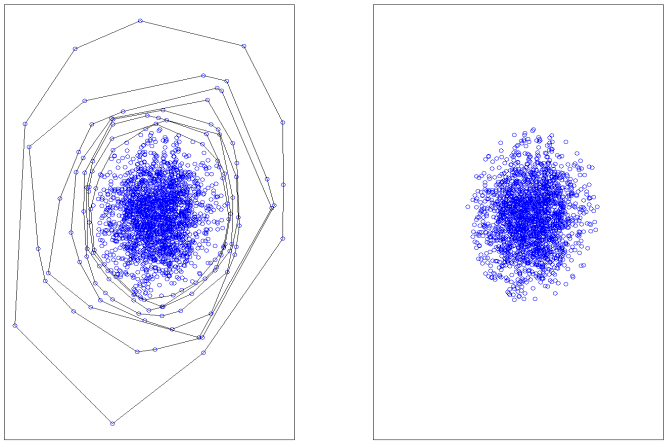

Figure 4.8: Plot showing peeling of depth 8 and the resulting remaining dataset

A general problem we may encounter when performing data analysis on multi-modal data is the effects of different clusters on the calculation of parameters such as the mean. In figure 4.11 we can see the mean is between the two data clusters and as such is not useful in making predictions. If we imagine the top cluster is of interest to us we may attempt to use our peeling method to remove any outliers in the top cluster and remove the bottom cluster entirely.

However we now encounter our second problem with our traditional α-peeling, when we apply our method we get the following result as shown in figure 4.12. Peeling to a high enough depth (in this case depth 10) to remove the bottom cluster results in us losing a large amount of both the shape of the top cluster as well as reducing the total amount of remaining data points in it significantly. This severely impacts our ability to make inference from this data and as such our traditional multi-variate α-peeling is not appropriate. As a result we seek a new method.

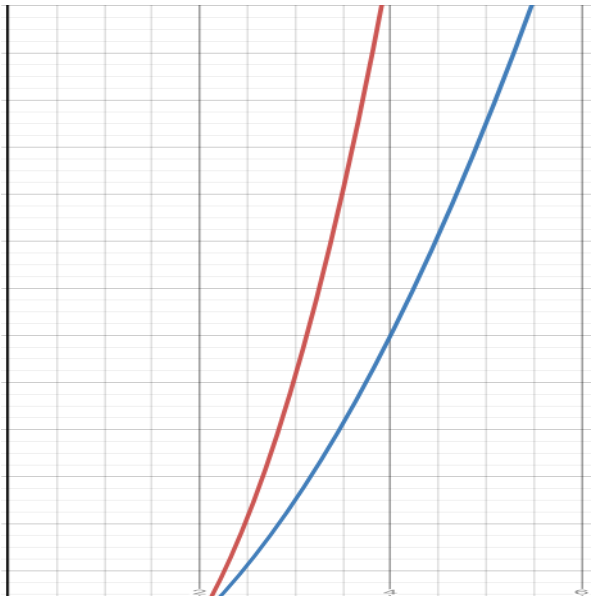

Figure 4.9: Graph showing worst case efficiency of the two methods Red: $O(n^2 log(n))$ Blue: $O(n^2)$

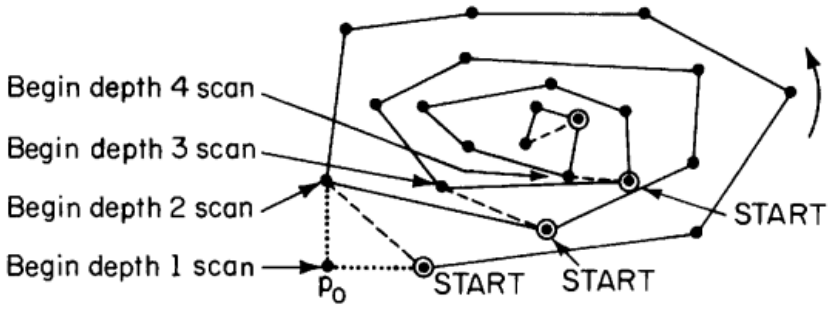

Figure 4.10: Image demonstrating faster peeling from [3]

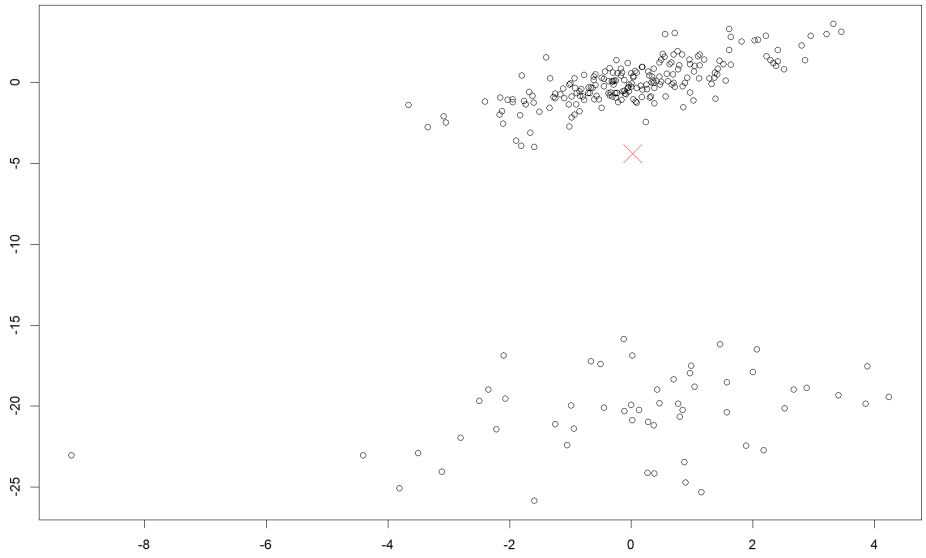

Figure 4.11: A bimodal dataset with the mean displayed as a red cross

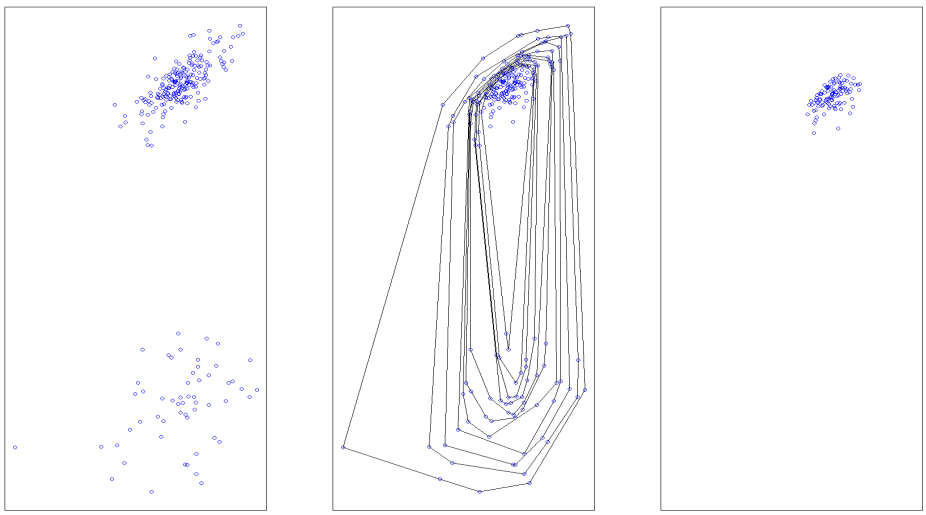

Figure 4.12: Our bimodal dataset being peeled using our normal method to a depth of 10

Chapter 5

Resistant Peeling algorithm

5.1 Motivation

In the last chapter we developed some basic methods to perform outlier detection resulting in a technique to handle multi-variate data. However we then discovered some problems that arise as a result of applying this technique to certain types of dataset specifically those with multiple clusters. We would like our new technique to handle traditional peeling scenarios well, ideally be computationally quick and be able to deal with these multi-modal datasets. Luckily, a method exists that can better handle multi-modal data more readily named the resistant peeling method, through this chapter we present and explain this new method.

The method is adapted from [8] but all code and visual demonstrations are my own, we work with toy datasets for testing and explanation. To obtain the datasets we combine two or more bi-variate normal distributions datasets to result in one larger multi-modal dataset. For example to produce the bimodal dataset we used as an example in the previous chapter and for the explanation of our method in this chapter we combine two multivariate normal datasets:

one of 200 points with mean $\begin{pmatrix}0\\2\end{pmatrix}$ and variance $\begin{pmatrix}2 & 1.6\\1.6 & 2\end{pmatrix}$.

and one of 55 points with mean $\begin{pmatrix}0\\-20\end{pmatrix}$ and variance $\begin{pmatrix}6 & 1.8\\1.8 & 6\end{pmatrix}$.

The code to produce the data and combine them is shown below

```
set.seed(100)
data2 <- mvrnorm(200,rep(0,2),matrix(c(2,1.6,1.6,2),2,2))
data3 <- mvrnorm(55,c(0,-20),matrix(c(6,1.8,1.8,6),2,2))
bimodal <- rbind(data2,data3)
```

5.2 The resistant peeling method

We begin our resistant peeling in a familiar way, with finding the convex hull of our dataset.

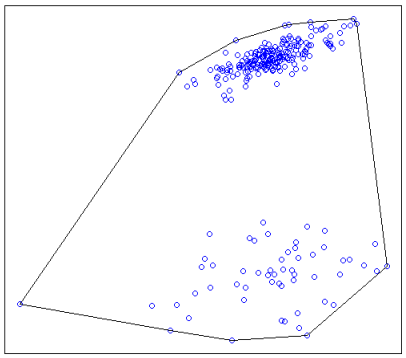

Figure 5.1: Convex hull of our example data

We now think about each vertex on the hull in turn, for each vertex we complete the next set of steps (for the rest of this explanation the plots are generated for the leftmost vertex).

First we calculate the distance between the vertex and all other points in the data set and then order these distances (using euclidean distance).

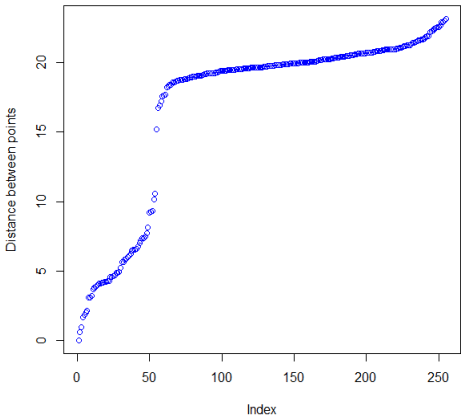

Figure 5.2: Ordered distance plot

Next we calculate the gaps between these ordered distance points, we can view this as the difference in the y axis for each two adjacent points in figure 5.2. Now we find the index for the maximum gap and we use this to identify what points to remove.

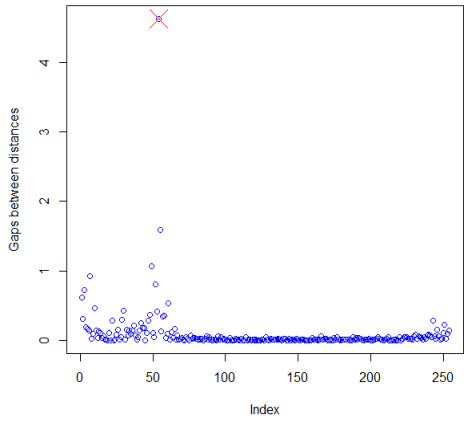

Figure 5.3: Gaps plot with the maximum gap identified by a red cross

To help explain this removal process we wish to identify whether some set of points are interior or exterior points. In this process we let exterior points be those points that are not part of the large cluster we are interested in or are points we view as outliers of the large cluster. Conversely, interior points are points that are part of the large cluster.

To allow our algorithm to classify points as interior or exterior we check the index of where the maximum gap in distances occurs with respect to the total number of data points. If this maximum gap occurs before half of the data points, all points corresponding to the gaps before this maximum gap are considered exterior points and are removed from the dataset. If the maximum gap occurs at or after half of the total number of points then only the hull vertex is considered a exterior point.

In this example the maximum gap occurs at 54 which is prior to index 128, the halfway index, so all points that gave the gaps and distances for indexes 1-54 are exterior and need to be removed.

Once we have repeated this process for all hull vertices we have a data set that has been peeled with resistant peeling and our algorithm has been completed .

5.3 Comparison with other outlier detection methods

Now we present the actual effect of our new method and compare it with our previous method on our toy dataset. We select an $\alpha = 0.12$ removing around 61 data points using each method.

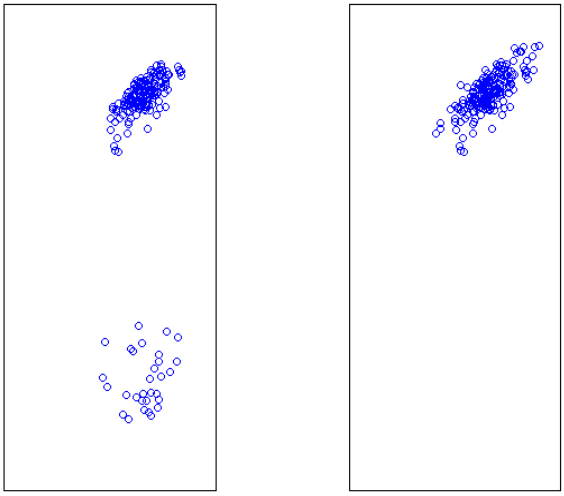

Figure 5.4: Left: Dataset traditionally peeled Right: Dataset with resistant peeling

We can very clearly see the improvement of our resistant peeling algorithm over the traditional method. We have retained significantly more of the top data whilst also fully removing the lower cluster. Therefore if we seek to use α-peeling for outlier detection for multi-modal data our new algorithm is superior as we have to use much larger α values to achieve the same removal of the bottom cluster.

We can also make a comparison between both methods when we peel until we leave only the top cluster.

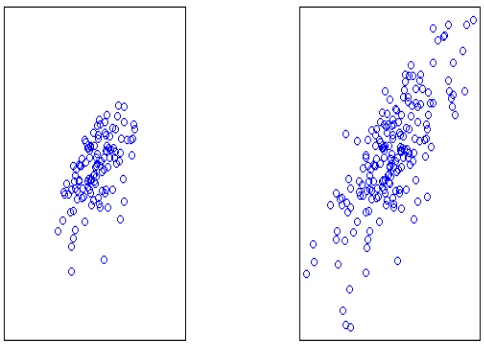

Figure 5.5: Left: Depth 10 normal peeling Right: Resistant peeling

This comparison really exemplifies how useful our new method is we retain a lot about the shape of the distribution meaning any variance calculations we do will be far more relevant to the real distribution. It also gives us more actual data points meaning our calculations will also be more accurate as a result of the larger sample size.

We can also make a comparison between resistant peeling and our other outlier detection method, the Mahalnobis distance calculation. We show this comparison in figure 5.6. We can clearly see that the Mahalanobis method fails at detecting all of the cluster as an outlier and only removes some of the points, whereas as we expect our resistant peeling method fully removes the top cluster leaving only the relatively unaffected bottom cluster. This example demonstrates the major weakness of the Mahalanobis method, when the mean of the data set is between the clusters the method struggles to figure out which points are the issue. This is to be expected though as the bimodal dataset is not multivariate normal so it does not make much sense to use the method for these purposes.

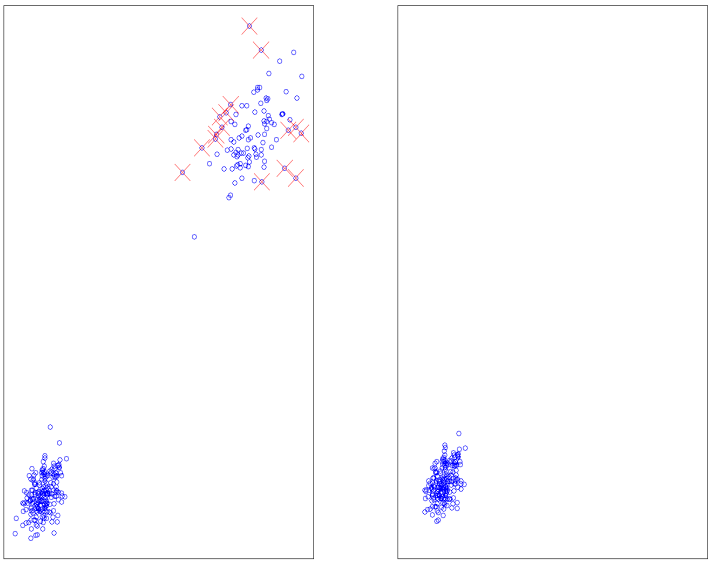

Figure 5.6: Comparison plot between Left: Mahlanobis outlier detection Right: Resistant peeling

5.4 A further application

Imagine instead of removing the smaller cluster let us imagine we instead seek to investigate this smaller cluster, luckily we can simply adapt our resistant peeling method to allow for this. The process is similar to our normal resistant peeling but instead of removing points if the maximum gap occurs before half way the dataset we instead keep those points and remove points if the maximum gap occurs after the halfway point through the dataset.

The result of this is shown in figure 5.7. This inverse peeling makes our method even more versatile and allows for investigation of whichever part of the data we are interested in .

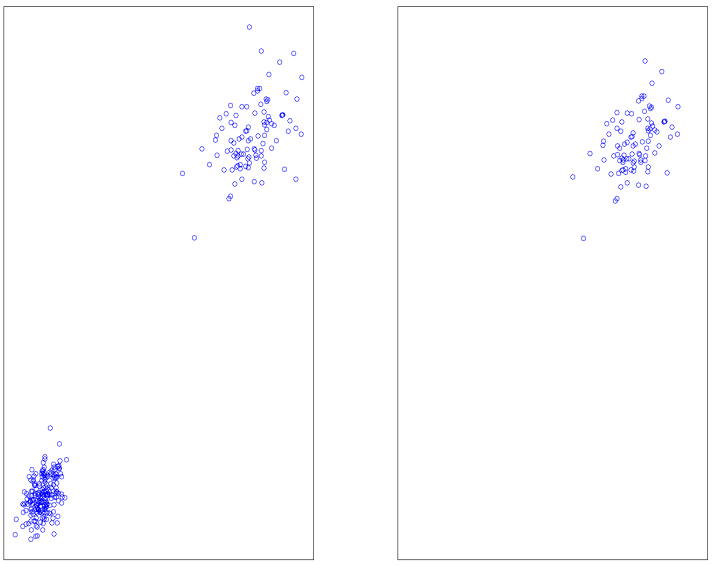

Figure 5.7: A comparison plot showing the removal of the large cluster

Chapter 6

Conclusion

Through this dissertation we have been able to explain a variety of methods to find convex hulls and then in turn have used our understanding of convex hulls to be able to perform outlier detection. From this point there is several potential areas of interest to continue to work on.

One area is extending our ideas of our traditional peeling to n dimensions rather than the 2 dimensions we have focused on during this dissertation, this would require a small effort to adapt the current code to be able to readily handle the *chull*() function for n dimensions, aswell as adjusting the plotting for higher dimensions.

Similarly, we would seek to expand our resistant peeling to higher dimensions, this would require more effort as we have to calculate distances from each plane of an n-dimensional plane to each data point in set, once a robust process to perform this for any given dimension has been developed the remaining steps in turn should be simple to calculate, though presentation of high dimensional data and keeping the readability of any given graphs is difficult.

We also would like to be able to adapt the inverse peeling method to be able to work with tri-modal data or even fully multi-modal data sets. This may involve multiple applications of our peeling algorithm to leave the data we are interested in

Bibliography

[1] https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

[2] https://www.scribbr.com/statistics/outliers/

[3] Preparata and Shamos, *Computational Geometry*,

[4] GeeksforGeeks, https://www.geeksforgeeks.org/convex-hull-algorithm/

[5] R.L Graham *An efficient algorithm for determining the convex hull of a finite planar set*

[6] R. A. Jarvis, On the identification of the convex hull of a finite set of points in the plane,

[7] Tahani Coolen-Maturi Statistical Modelling II 2022/23

[8] Porzio and Ragozini, *Peeling Multivariate Data Sets: A New Approach*