Drawbacks of TCLUST

A flexible Trimmed K-means algorithm in

Robust Cluster Analysis

Pan Charoensuk

V00827791

Stat 454 - Robust Statistics

Project 2

Department of Mathematics and Statistics

University of Victoria

Abstract

Outlying observations could be influential to the standard statistical estimators. Clustering Analysis which relies on the classical estimators is not an exception. One single outlier could form an uninteresting and unmeaningful cluster. Many robust estimators are used to remedy this problem. Simply deleting such outlying observations is one of the techniques we could consider, though one may argue that we would be losing out on data and that the outlying observations that may form a group could be meaningful and describe a strange behavior in our dataset. In which case a mixture model should be considered. With this motivation, we will consider only the case where outlying observations ought to be trimmed. To gain some flexibility and freedom, diverging from the classical clustering algorithm K-means that only restricts the spherical shape of the cluster and that each observation has the same weight, the TCLUST algorithm is invented for such purposes so that we are able to freely make desirable modifications. In exchange, researchers are forced to carefully choose suitable parameters which would be evaluated by the TCLUST algorithm. Drawbacks and problems are presented in this Project, signifying that inappropriate choices of initial parameters could lead to non-sensible results, and that the classical method is meant to be appreciated at times.

Section 1. Introduction to Robust Cluster Analysis

In unsupervised and non-hierarchical clustering, K-means is one of the most popular algorithms used to partition a dataset so that each clustered group shares more prominent similarities than that of other groups. K-means is an iterative, computationally efficient and easy to implement algorithm, but despite its perks, K-means also has several drawbacks in cluster analysis. In Robust Statistics, breakdown points play a crucial role in determining whether an estimator is robust. A breakdown point of an estimator is simply defined as the proportion of a maximum number of outlying observations before an estimator gives an incorrect result. In Section 2, we discuss the breakdown point of K-means and perform a simulation study to discover the breakdown points of the robust estimator of the K-means, and Trimmed K-means. In Section 3, we introduce the TCLUST algorithm, an algorithm that constrains the eigenvalues of the cluster scatter matrices, seen previously in Project 1. Despite its perks on the flexibility over the classical spherical shaped clusters with equally weighted observations, there are some drawbacks in the TCLUST algorithm. We will discuss its disadvantages in Section 4. Lastly, even before executing the clustering algorithms, we need to specify a sensible number of clusters k. We go over various methods to choose the most appropriate candidates k for the dataset, and Section 5 gives an example of choosing k with a real dataset taken from Kaggle.

Section 2. Breakdown point and Robustness of the estimators

Since K-means executes the algorithm iteratively by taking the mean of all observations assigned to the their centroids, K-means also has a breakdown point of 0, which is similar to the classical mean estimator. To robustify the algorithm, one could consider fitting a mixture model, treating an outlying group as another cluster, or one could simply trim out certain regions to remove outlying observations. In this project, we are only concerned with the trimming approach to cluster analysis.

Section 2.1. Example

In this example, we examine the influence of outlying observations on K-means and Trimmed K-means. We generate three mixtures of the bivariate gaussian components with two main clusters and one outlying cluster. The R code provided in the appendix section yields Figure 1. Our goal here is to correctly assign the two main clusters and disregard the outlying cluster. We see that K-means gives a cluster assignment to the outlying group, shown in Figure 1, (a), while Trimmed K-means provides the correct result in (b) with 5 observations deviating from the main clusters. However, as we increase such observations to 6, the algorithm treats them as another group instead (c). In conclusion, we have that the breakdown point for Trimmed K-means is the trimming proportion, α, which is similar to the classical trimmed means.

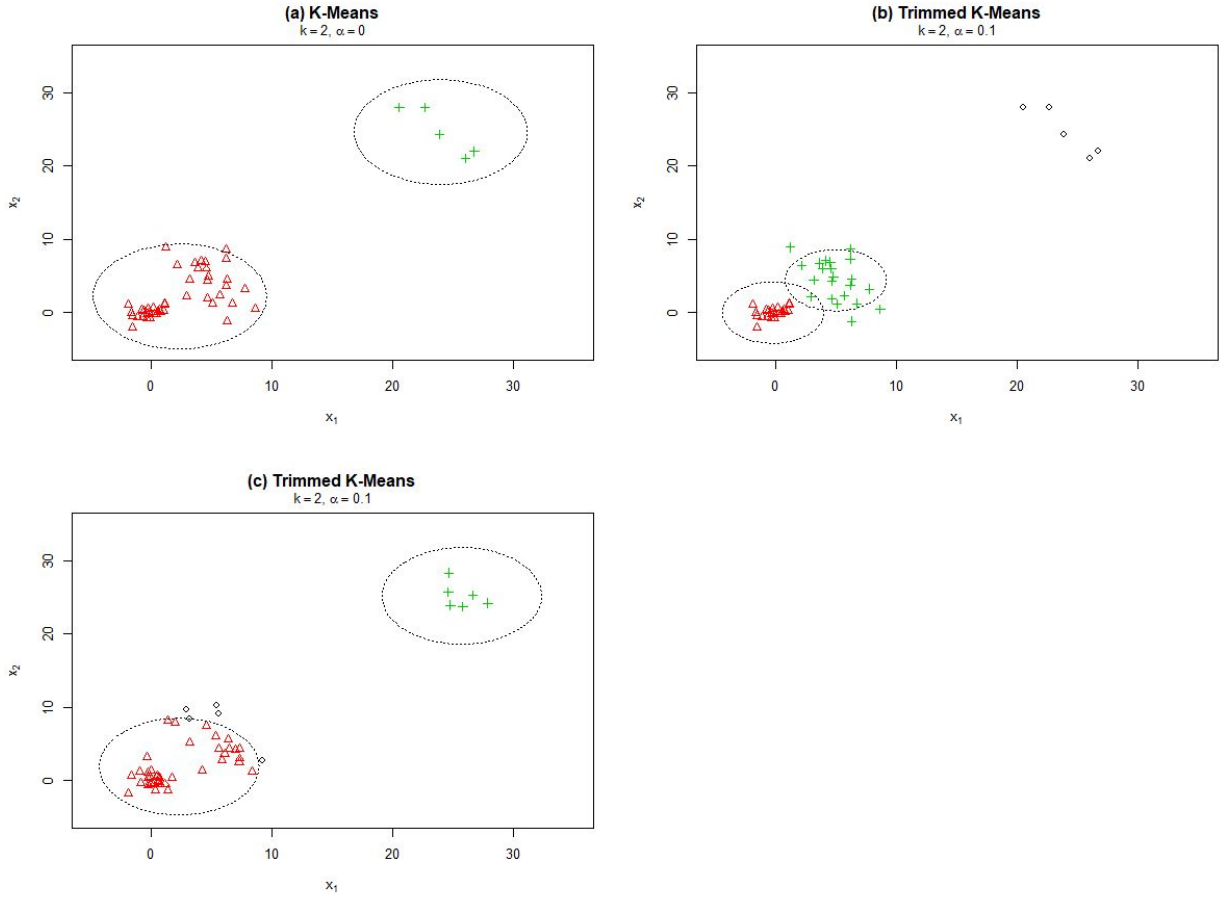

Figure 1: Results of the clustering processes for the bivariate normal generated data using K-means and Trimmed K-means algorithm with number of clusters k = 2 and the trimming proportion α = 0.1. The total number of observations is 49, 22 in each main cluster and 5 in the outlying group in (a) and (b). 50 observations in (c) with 22 in each main cluster and 6 in the outlying group.

Section 3. TCLUST - a flexible Trimmed K-Means

In our Project 1, we introduced the TCLUST algorithm. The TCLUST algorithm has the advantage of being able to adjust the cluster assignment by putting the restriction on scatter matrices which determine the shape of the clusters. TCLUST also allows us to assign appropriate weights onto each observation so that the cluster assignment algorithm yields an optimal result. We have seen that in Project 1, TCLUST

improves the misclassification rate on top of K-means and Trimmed K-means on both simulation studies and a real dataset, Iris. Moreover, as the background noise is introduced, TCLUST gives desirable results over Trimmed K-means, illustrated below in Figure 2.

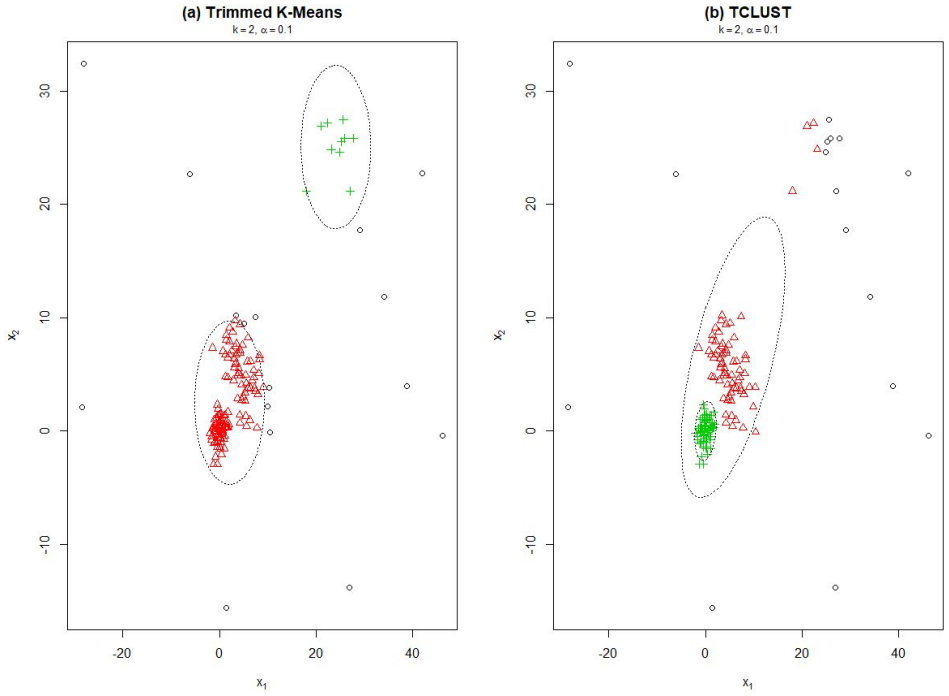

Figure 2: Clustering using Trimmed K-means (a) and TCLUST (b) with background noise with k = 2 and α = 0.1.

Here, we are still interested in clustering the two main groups and disregarding the background noise along with an outlying group. Figure 2, (a), shows that while the background noise has been trimmed, the outlying group becomes a cluster and the two main groups become another. TCLUST gives us a desirable result in (b), as we allow more flexibility on the weights and the shape of the clusters over the classical Trimmed K-means which only assumes the equal sized spherical shape of clusters.

TCLUST also takes the whole data structure into account in order to decide which parts of the sample should be discarded. This will allow us to get better results than having researchers determine the regions which are to be trimmed prior to the execution of the algorithm, or simply trimming the data “symmetrically”. We call such a trimming procedure, “impartial trimming”.

Section 4. Drawbacks and Problems

We discussed previously in Project 1 that K-Means relies heavily on four assumptions: knowing the appropriate number of clusters *k*, SSE is the right objective to minimize, every cluster has the same shape and every observation has the same weight. TCLUST satisfies most of these assumptions as we are able to alter the shape of the clusters and assign different weights for each observation. With more freedom and flexibility, we are tasked with creating more intricate strategies for making decisions. Choosing the inappropriate type and the inappropriate upper bound of constraints would lead to undesirable results. This is especially harder in the case where we are given a multidimensional data with dimension of 3 and greater, as it is currently impossible to get a clear visualization of the full data clusters.

Firstly, we would like to know what is a sensible choice of *k*. Notice that the trimming proportion α is dependent on *k*. Iteratively trimming off the data could lead to trimming off outlying clusters, reducing *k*, and vice versa. The function “ctlcurves” in TCLUST gives a visual representation on the values of *k* which maximize the objective function at different trimming levels. Our objective function here is the log-likelihood function which is evaluated at different partition scenarios given the random generated

points. Garc´ıa-Escudero et al. (2012) defines this function rigorously and provides the mathematical derivations. Therefore the number of cluster *k* plays an important role in achieving the optimal clustering. There are various techniques and algorithms to determine the choice of *k*. To assist in verifying whether the "ctlcurves" function gives the best choice of *k*, the function "NbClust" (available in the "NbClust" package) performs many different algorithms on the dataset and gives a descriptive results on which *k* the majority of the algorithms have chosen. We give an example of when the function "NbClust" assists in making a decision for choosing the best *k* in Figure 4. Note that "NbClust" is only able to perform the analysis on the K-means algorithm, hence we will only consider the case where α is 0. Consider the bivariate gaussian data seen in Figure 1 with two main clusters and one outlying cluster. The total number of clusters here is three. Figure 3 illustrates various values of *k* at different α. Since we want to maximize the objective function at α = 0, our intuitive choice would be *k* = 4. This will lead to an overfitted problem as we have one excess cluster. Figure 4 correctly identifies the number of clusters using the majority rule. Therefore we have that the number of cluster *k* is 3.

The R code to Figure 4 is provided in the appendix. We will not, however, go into details of the function "NbClust". Charrad et al. (2014) discusses further usage and

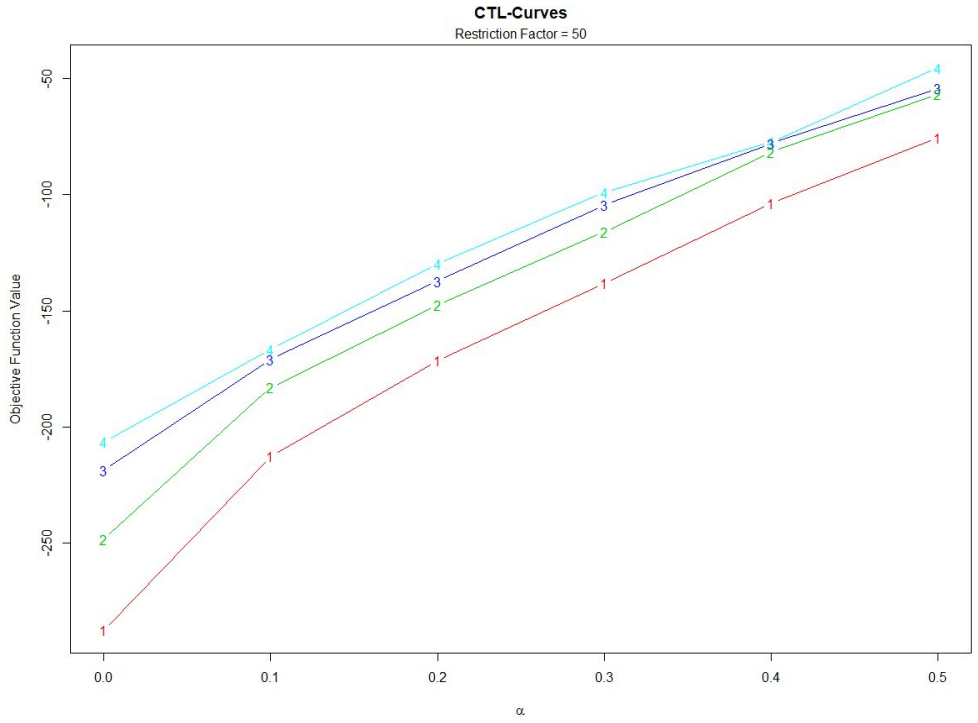

Figure 3: Classification Trimmed Likelihood Curves on the generated bivariate gaussian data seen in Figure 1, illustrating which *k* maximizes the objective function at different trimming level α.

R output from NbClust function

```
**********************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 14 proposed 3 as the best number of clusters

                 ***** Conclusion *****

* According to the majority rule, the best number of clusters is  3


**********************************************************************
```

Figure 4: An R output for the bivariate gaussian data seen in Figure 1, evaluated by the NbClust function using different algorithms.

gives descriptive mathematical procedures. The last technique that we will use to verify whether our choice of *k* is sensible, is the function “DiscrFact”, included in the TCLUST

package, where we could look for the doubtful cluster assignments for the algorithm. A usage of this function is demonstrated in the "**cars**" dataset in Section 5.

Secondly, the results obtained from the Iris data application in the Project 1 have shown some inconsistency with the TCLUST algorithm as we increase the upper bound on the ratio of the scatter matrices' eigenvalues. Since TCLUST is an iterative algorithm, it relies heavily on the randomly generated initial points. In this case, as we allow more flexibility onto the constraint, the randomly generated initial points have a higher chance of being in the same clusters. For example, one cluster could be vastly bigger than the other, leading to a rare case of having all of the generated initial points being in the big cluster. This leads to a high misclassification rate on the prediction compared to the actual data. Table 1 shows misclassification rates on the Iris data with different algorithms performed in Project 1. Here, in the "TCLUST" function, we set n.start = 50. So we will have fifty initializations for the clustering procedures. When the upper bound increases to 10, misclassification rates become inconsistent, contrary to when the constraint is more strict with an upper bound of 5. To alleviate this problem, we increase the number of initializations to three-hundred, n.start = 300. This greatly improves on the diversity of the algorithm so that for the first iteration, we hope to have the initializations being evenly distributed to the initial cluster assignment. Table 2 shows an improvement on the misclassification rates for when the upper bound is 10 as the results now are consistent in each run of the algorithm as we set n.start = 300. The R code to Table 1 and Table 2 is provided in the appendix.

Clustering on Iris data with n.start = 50

	K-means	TkMeans	TCLUST(upp_bound=5)	TCLUST(upp_bound=10)
Run 1	0.1066667	0.14	0.09333333	0.08666667
Run 2	0.1066667	0.14	0.09333333	0.64666667
Run 3	0.1066667	0.14	0.09333333	0.64000000
Run 4	0.1066667	0.14	0.09333333	0.08666667
Run 5	0.1066667	0.14	0.09333333	0.64666667
Run 6	0.1066667	0.14	0.09333333	0.08666667
Run 7	0.1066667	0.14	0.09333333	0.64000000
Run 8	0.1066667	0.14	0.09333333	0.64666667
Run 9	0.1066667	0.14	0.09333333	0.08666667
Run 10	0.1066667	0.14	0.09333333	0.08666667

Table 1: Misclassification rates for various algorithms performed on the Iris dataset with four variables, Sepal Length, Sepal Width, Petal Length and Petal Width. K-means, Trimmed k-means, TCLUST with upper bound of 5 and TCLUST with upper bound of 10 are evaluated 10 times for consistency. Here, TCLUST takes on 50 intializations.

Clustering on Iris data with n.start = 300

	K-means	TkMeans	TCLUST(upp_bound=5)	TCLUST(upp_bound=10)
Run 1	0.1066667	0.14	0.09333333	0.08666667
Run 2	0.1066667	0.14	0.09333333	0.08666667
Run 3	0.1066667	0.14	0.09333333	0.08666667
Run 4	0.1066667	0.14	0.09333333	0.08666667
Run 5	0.1066667	0.14	0.09333333	0.08666667
Run 6	0.1066667	0.14	0.09333333	0.08666667
Run 7	0.1066667	0.14	0.09333333	0.08666667
Run 8	0.1066667	0.14	0.09333333	0.08666667
Run 9	0.1066667	0.14	0.09333333	0.08666667
Run 10	0.1066667	0.14	0.09333333	0.08666667

Table 2: Misclassification rates for various algorithms performed on the full Iris similarly to Table 1. Here, TCLUST takes on 300 intializations, allowing more diversity for the clustering procedures to avoid most of the initializations being in the same initial cluster.

Lastly, we have discovered that all of the aforementioned algorithms do not give reasonable results when performed on high dimensional dataset in some cases. Garc´ıa-Escudero et al. (2012) also mentions that even though TCLUST is an

improvement on the trimming procedures based on K-means algorithm, in the situation where we have a high number of clusters and dimensions, TCLUST algorithm could get computationally inefficient and expensive. This is due to the fact that the impartial trimming approach, which takes the whole data structure into account, has to evaluate each possible partition of the data and choose the best combinations of the main clusters and the outlying clusters until we reach the cluster assignment which maximizes the objective function.

Section 5. Cars dataset

Consider the “cars” dataset, taken from Kaggle. We would like to cluster the cars by their brands, namely US, Japan and Europe by taking seven variables, MPG, cylinders, weight in lbs, year, etc, into account.

We would like to find the appropriate number of clusters *k*. To visualize the candidates of *k*, we will look at the CTL-curves, and use the function “DiscrFact,” instead of the function “NbClust”. Figure 5 illustrates that *k* = 3,4 and 5 are the candidates as they give high values for our objective function. However, there is no significant improvement as we increase *k* from 4 to 5. Hence we will only consider the case where *k* = 3 and 4. In the case where *k* = 4, Figure 6 and 7 contain the plots to assist in visualizing doubtful clustering decisions. Garc´ıa-Escudero et al. (2012) discusses and defines the discriminant factors as the criterion extensively. For the silhouette plot, large (close to 0) discriminant factors, DF, indicates doubtful

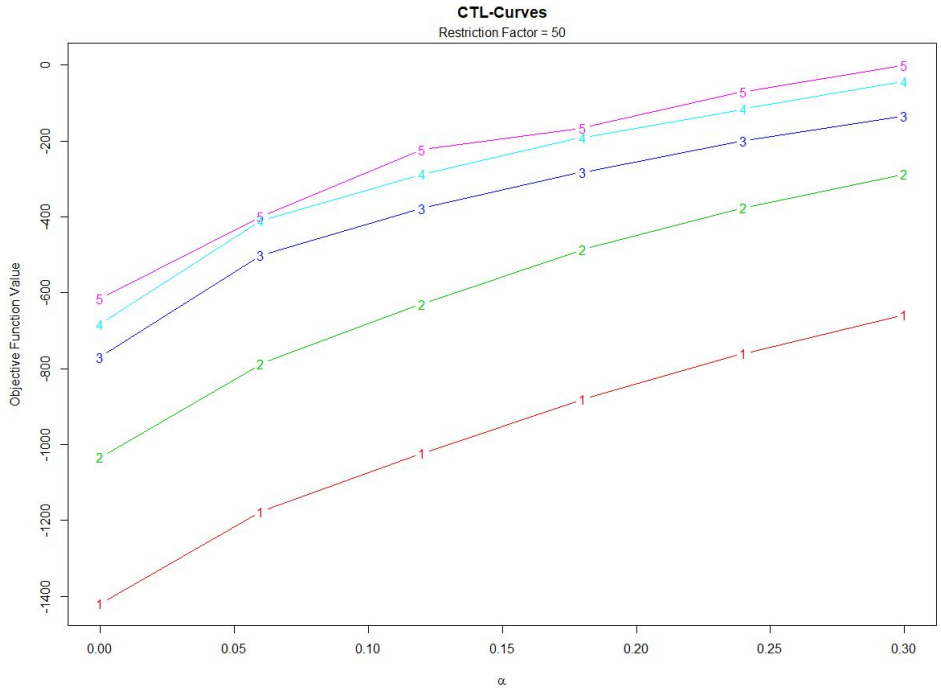

Figure 5: CTL Curves of Car dataset to determine the sensible choice of *k*

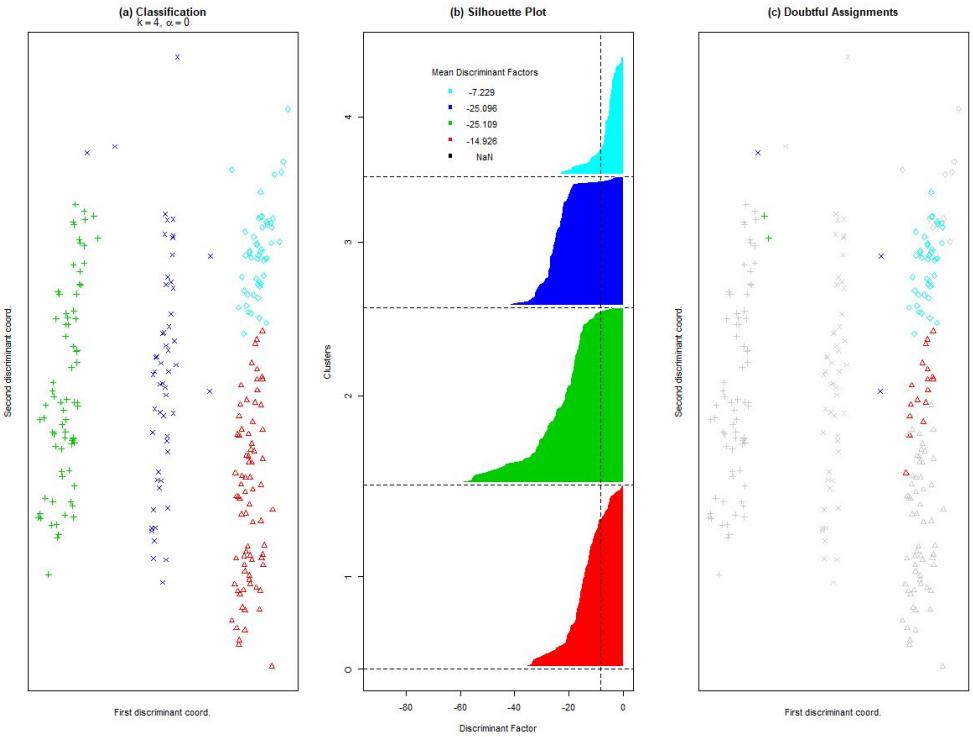

Figure 6: Results from the function “DiscrFact” function to help determine the doubtful cluster assignments with *k* = 4.

assignments or trimming decisions. Cluster 4 in Figure 6, (b), has a width close to zero meaning that observations are assigned into Cluster 4 doubtfully-and Figure 6, (c), reinforces this result as we see that a significant amount of the observations (in red and light blue) are doubtfully assigned. As we reduce *k* to 3, we see much better results, shown in Figure 7. Figure 7, (b), shows an improvement as the widths for all of the clusters are small, meaning that we do not have many doubtful cluster assignments. Figure 7, (c), supports this result as we only see two doubtful assignments (in red). Therefore, we have that *k* = 3 is the most appropriate choice.

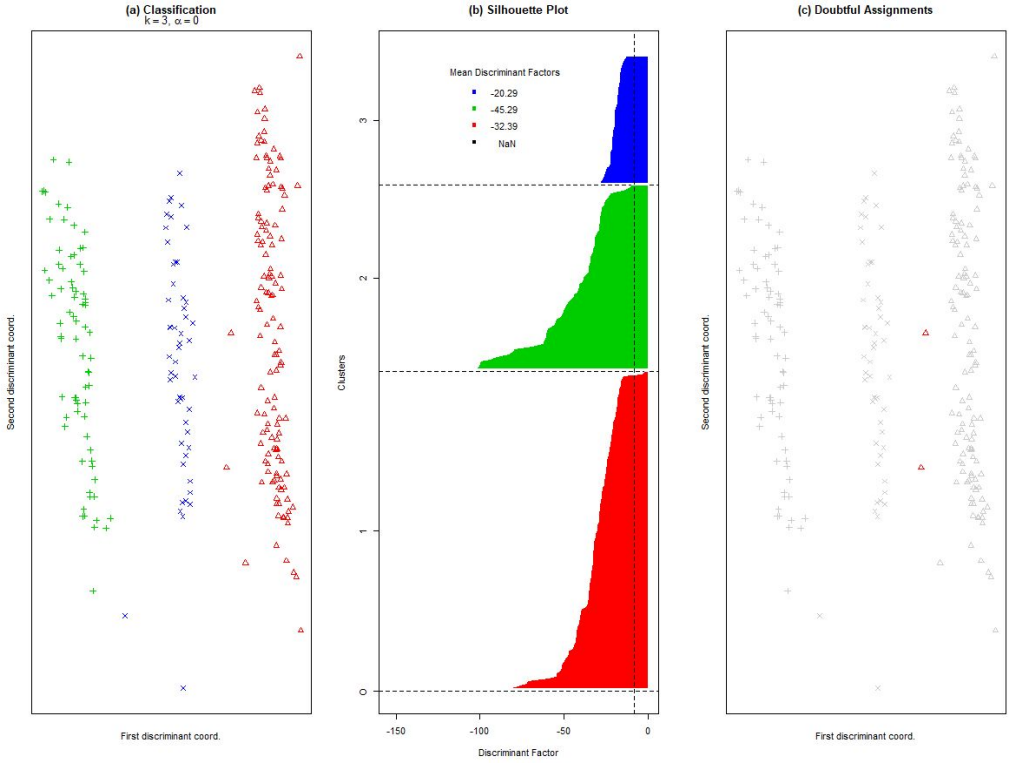

Figure 7: Results from the function "DiscrFact" function to help determine the doubtful cluster assignments with *k* = 3. (b) shows an improvement on the doubtful decisions over Figure 6 as DF in Silhouette plot are not close to 0. Moreover, (c) barely has any red triangles, the doubtful assignments

Section 6. Conclusion

K-means is a widely used clustering algorithm in unsupervised learning. However, the mean estimator which K-means relies on, is not a robust estimator. To robustify the mean, we trimmed out the most unusual and outlying data relative to the majority. To robustify even further, we alter the shape of the clusters and the weight on each observation by putting constraints onto the scatter matrices, using TCLUST algorithm. At the price of freedom, we need to choose the initial parameters carefully. There is no absolute answer to which parameters are optimal and often researchers have to make such decision given the tasks and problems at hand. Increasing the number of initializations gives reasonable results more consistently, with the cost of some computational speed. This problem quickly magnifies when we are dealing with a large number of clusters k in a high dimensional case. Hence TCLUST algorithm should not always be the only pivotal algorithm. Other non-hierarchical algorithms such as classical K-means should also be considered as it is not as computationally expensive as TCLUST. Other techniques such as performing Principal Component Analysis (PCA), Independent Component Analysis (ICA), Cross Validation (CV), etc, on our dataset would also prove useful to enhance the clustering procedures and ensure that our results are desirable.

References

Charrad, M. Ghazzali, N. Boiteau, V. Niknafs, A. (2014). “NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set”. *Journal of Statistical Software,* **61**(6), DOI: 10.18637/jss.v061.i06

García-Escudero, L.A., Mayo-Iscar, A. Fritz, H. (2012). “tclust: An R Package for a Trimming Approach to Cluster Analysis”. *Journal of Statistical Software*, **47**(12), DOI: 10.18637/jss.v047.i12

From website (https://www.kaggle.com/abineshkumark/carsdata, April 5th, 2019).

Appendix

R codes:

```
##############################################################################
################################## SIMULATIONS ###############################
##############################################################################


sigma1 <- diag (2)
sigma2 <- diag (2) * 8 - 2
sigma3 <- diag (2) * 1000
sigma4 <- diag(2) * 10-4
mixt <- rbind(rmvnorm (22, mean = c (0.0, 0), sigma = sigma1),
              rmvnorm (22, mean = c (5,5), sigma = sigma2),
              rmvnorm (5, mean = c (25,25), sigma = sigma4))


mixt2 <- rbind(rmvnorm (22, mean = c (0.0, 0), sigma = sigma1),
               rmvnorm (22, mean = c (5,5), sigma = sigma2),
               rmvnorm (6, mean = c (25,25), sigma = sigma4))


mixt3 <- rbind(rmvnorm (67, mean = c (0.0, 0), sigma = sigma1),
               rmvnorm (67, mean = c (5,5), sigma = sigma2),
               rmvnorm (10, mean = c (25,25), sigma = sigma4))

# Determining number of clusters k
plot(ctlcurves(mixt, k = 1:4, alpha = seq(0, 0.5, by =  0.1), restr.fact = 50))
res <- NbClust(data = mixt, distance = "euclidean", min.nc = 2, max.nc = 5, method = "kmeans", index = "all")

# Discovering the breakdown point for trimmed k-means
mixt_k <- tclust(mixt, k = 2, alpha = 0, restr.fact = 1, equal.weights = TRUE)
plot(mixt_k, main = "(a) K-Means", xlim = c(-5,35), ylim = c(-5,35),
     xlab = bquote(x[1]), ylab = bquote(x[2]))
mixt_tk <- tclust(mixt, k=2, alpha = 0.1, restr.fact = 1, equal.weights = TRUE)
plot(mixt_tk, main = "(b) Trimmed K-Means", xlim = c(-5,35), ylim = c(-5,35),
     xlab = bquote(x[1]), ylab = bquote(x[2]))
mixt2_tk <- tclust(mixt2, k=2, alpha = 0.1, restr.fact = 1, equal.weights = TRUE)
plot(mixt2_tk, main = "(c) Trimmed K-Means", xlim = c(-5,35), ylim = c(-5,35),
     xlab = bquote(x[1]), ylab = bquote(x[2]))


# Background noise is added

mixt3_tk = tclust(mixt3, k=2, alpha = 0.1, restr.fact = 1, equal.weights = TRUE)
plot(mixt3_tk, main = "(a) Trimmed K-Means",
     xlab = bquote(x[1]), ylab = bquote(x[2]))

mixt3_tclust = tclust(mixt3, k = 2, alpha = 0.1, restr.fact = 50, equal.weights = FALSE)
plot(mixt3_tclust, main = "(b) TCLUST", xlab = bquote(x[1]), ylab = bquote(x[2]))

# TCLUST correctly clusters the data while trimmed k-means does not.
```

```
################################################################################
############################## MULTIDIMENSIONAL IRIS ###########################
################################################################################

iris
set.seed(5)
library(tclust)
library(NbClust)
library(mvtnorm)


# plotting CTL-Curves to determine k and alpha
plot(ctlcurves(iris[,1:4], k = 1:5, alpha = seq(0,0.3, by = 0.03)))
#no significant improvement when increase k from 3 to 4,5. Choose k = 3

# defining true clusters
true_iris = data.frame(cbind(iris[,1:4], true_clust = c(rep(2,50), rep(1,50), rep(3,50))))
names(true_iris) = c("x1","x2","x3","x4","true_clust")

# 10 simulations
num_sim <- 10
misclass_table1 <- matrix(NA, nrow = num_sim, ncol = 4)
for(i in 1:num_sim){
  set.seed(i)
  #kmeans 3 clusters
  iris_kmeans <- tclust(iris[,1:4], k = 3, alpha = 0, equal.weights = TRUE, restr.fact = 1)
  #calculate misclassification rate
  k_means_mr=sum(iris_kmeans$cluster!=true_iris$true_clust)/length(true_iris$true_clust)


  #trimmed kmeans
  iris_tkmeans <- tclust(iris[,1:4], k = 3, alpha = 0.03, equal.weights = TRUE, restr.fact = 1)
  #calculate misclassification rate
  tk_means_mr=sum(iris_tkmeans$cluster!=true_iris$true_clust)/length(true_iris$true_clust)

  #tclust
  ## eigen, restr.fact = 5
  iris_tclust.5 <- tclust(iris[,1:4], k = 3, alpha = 0, equal.weights = FALSE, restr.fact = 5)
  #calculate misclassification rate
  tclust.5_mr=sum(iris_tclust.5$cluster!=true_iris$true_clust)/length(true_iris$true_clust)

  ## eigen, restr.fact = 10
  iris_tclust.10 <- tclust(iris[,1:4], k = 3, alpha = 0, equal.weights = FALSE, restr.fact = 10)
  #calculate misclassification rate
  tclust.10_mr=sum(iris_tclust.10$cluster!=true_iris$true_clust)/length(true_iris$true_clust)

  ## misclassification rate table
  misclass_table1[i,1] <- k_means_mr
  misclass_table1[i,2] <- tk_means_mr
  misclass_table1[i,3] <- tclust.5_mr
  misclass_table1[i,4] <- tclust.10_mr

}

colnames(misclass_table1) <- c("K-means", "TkMeans", "TCLUST(upp_bound=5)", "TCLUST(upp_bound=10)")
rownames(misclass_table1) <- c("Run 1", "Run 2", "Run 3", "Run 4", "Run 5", "Run 6", "Run 7", "Run 8",
                               "Run 9", "Run 10")
misclass_table1
```

```r
# TkMeans and TCLUST gives inconsistent results. We will now increase nstart to 300 for
# TCLUST with upper bound = 10

misclass_table2 <- matrix(NA, nrow = num_sim, ncol = 4)
for(i in 1:num_sim){
  set.seed(i)
  #kmeans 3 clusters
  iris_kmeans <- tclust(iris[,1:4], k = 3, alpha = 0, equal.weights = TRUE, restr.fact = 1)
  #calculate misclassification rate
  k_means_mr=sum(iris_kmeans$cluster!=true_iris$true_clust)/length(true_iris$true_clust)


  #trimmed kmeans
  iris_tkmeans <- tclust(iris[,1:4], k = 3, alpha = 0.03, equal.weights = TRUE, restr.fact = 1)
  #calculate misclassification rate
  tk_means_mr=sum(iris_tkmeans$cluster!=true_iris$true_clust)/length(true_iris$true_clust)

  #tclust
  ## eigen, restr.fact = 5
  iris_tclust.5 <- tclust(iris[,1:4], k = 3, alpha = 0, equal.weights = FALSE, restr.fact = 5)
  #calculate misclassification rate
  tclust.5_mr=sum(iris_tclust.5$cluster!=true_iris$true_clust)/length(true_iris$true_clust)

  ## eigen, restr.fact = 10 with nstart = 300
  iris_tclust.10 <- tclust(iris[,1:4], k = 3, alpha = 0, equal.weights = FALSE, restr.fact = 10,
                           nstart = 300)
  #calculate misclassification rate
  tclust.10_mr=sum(iris_tclust.10$cluster!=true_iris$true_clust)/length(true_iris$true_clust)

  ## misclassification rate table
  misclass_table2[i,1] <- k_means_mr
  misclass_table2[i,2] <- tk_means_mr
  misclass_table2[i,3] <- tclust.5_mr
  misclass_table2[i,4] <- tclust.10_mr

}

colnames(misclass_table2) <- c("K-means", "TkMeans", "TCLUST(upp_bound=5)", "TCLUST(upp_bound=10)")
rownames(misclass_table2) <- c("Run 1", "Run 2", "Run 3", "Run 4", "Run 5", "Run 6", "Run 7", "Run 8",
                               "Run 9", "Run 10")
misclass_table2 # results TCLUST upper bound = 10 are now consistent
```

```
##############################################################################################
################################### CARS DATASET #############################################
##############################################################################################

## Discriminant Factors to determine the appropriate k

# cars dataset is taken from Kaggle, https://www.kaggle.com/abineshkumark/carsdata
cars <- read.csv("cars.csv")
variables <- cars[,1:7]
colSums(is.na(variables))

# remove NA's in the variables.
variables <- variables[complete.cases(variables), ]
variables <- scale(variables)

cars <- cars[complete.cases(cars), ]
colSums(is.na(variables))

library(tclust)
par(mfrow = c(1,1))
plot(ctlcurves(variables, k = 1:5, alpha = seq(0, 0.3, by =  0.06), restr.fact = 50))

# TCLUST, discriminant factors function

x1 <- tclust(variables, k = 4, alpha = 0)
plot(DiscrFact(x1, threshold = 0.0001), enum.plots = TRUE)


x2 <- tclust(variables, k = 3, alpha = 0)
plot(DiscrFact(x2, threshold = 0.0001), enum.plots = TRUE)
```