Applying robust covariance matrix on TCLUST algorithm

Songwan Joun, V00729173

Abstract

Clustering is an unsupervised machine learning algorithm used for grouping data points based on their distance metrics. TCLUST provides general trimming approach to robust cluster analysis. However, TCLUST uses random samples to form initial scatter matrices of clusters. Also, this algorithm updates parameters based on eigenvalue-ratio restrictions. Instead, in this report, I will substitute cluster centers and scatter matrices in initialization and maximization step of TCLUST algorithm as robust covariance matrix location estimators and robust covariance matrices. This approach will prevent clusters from being seriously affected by extreme or biased data points. In the simulation study, I compared the performance of clustering by four different robust covariance matrices. The result showed that, while all of the robust covariance estimation methods compared showed fairly good performance, "Donoho-Stahel projection based estimator" showed the minimum misclassification rate.

Keywords: Robust Clustering, Robust covariance matrix, TCLUST, fast-MCD algorithm,

Mathematics Subject Classification 62H30

1. Introduction

Clustering is an unsupervised machine learning algorithm used for grouping data points based on their distance metrics such as Euclidean distance, cosine distance, Mahalanobis distance and other user defined distances. García et al. (2008) mentioned in their paper that, among many other clustering algorithms, K-means is one of the standard methods, which assumes Gaussian-like and same spherical shapes of roughly equal sizes for all clusters. However, K-means does not produce reliable clustering results when there are constellations of groups that depart strongly from this assumption. Moreover, Böhm et al. (2006) stated that K-means and other typical clustering algorithms tend to be sensitive to outliers, because they try to optimize the log-likelihood of a Gaussian, which is equivalent to the Euclidean (or Mahalanobis) distance – either way, an outlier has high impact on the clustering.

To avoid the clustering results from being significantly affected by the outlier, robust approach to clustering algorithm has been studied among various researchers. García et al. (2008) developed a general trimming approach to robust cluster analysis, TCLUST. García et al. (2008) also mentioned that "It may be seen as a classification EM-algorithm where a kind of "concentration" step as in fast-MCD (minimum covariance determinant) algorithm is also applied". In maximization step, they update cluster means and cluster covariance by sample means and scatter matrices. Here, scatter matrices are modified by truncated eigenvalues and spectral decomposition of sample covariance matrices. Also, in the initialization step, TCLUST uses scatter matrices and cluster centers calculated from randomly chosen $k \times (p + 1)$ observations where k is the number of clusters, and p is the dimension of dataset. A detailed algorithm for TCLUST can be found in García et al. (2008).

However, using sample means and scatter matrices of randomly selected points as an initial input can lead to performance degradation of TCLUST algorithm. This is because randomly extracted points may well represent each cluster, but may also be severely biased. Accordingly, early clusters that are severely misformed can have a significant impact on the resulting cluster.

Alternatively, for each cluster, we can think of using robust covariance matrices instead of scatter matrices formed from random initial data points. We can also apply robust covariance matrices in maximization step (updating cluster centers and cluster scatter matrices) of TCLUST algorithm. This replacement will be meaningful because first, robust covariance matrices will prevent clusters from being seriously affected by extreme or biased random data points, and second, instead of giving eigenvalue-ratio constraints or trimming, this approach will give another way of robust clustering.

In this report, I will substitute cluster centers and scatter matrices in initialization and maximization step of TCLUST algorithm as robust covariance matrix location estimators and robust covariance matrices. The performance of four estimators of robust covariance matrices were compared. I will first review one of the robust covariance estimation algorithm, Fast-MCD in Section 2, and do simulation study to compare this algorithm with other three methods on clustering performance in Section 4.

2. Fast-MCD algorithm

When the dimension of a data is larger than two, it is hard to detect outliers. Instead, using robust distance estimator is better in this case because traditional Mahalanobis distance is no longer be able to find proper outliers due to masking effect (Rousseeuw et al., 1999). There are many robust methods for estimating covariance matrices including fast MCD, reweighted MCD, Donoho-Stahel projection based estimator and constrained M estimator. In this report, I will review the idea behind especially for the fast-MCD algorithm, how it is implemented, and see its breakdown point. Those four methods are available in "*covRob()*" function the R package "*robust*".

All of the mathematical derivations and descriptions in section 2 are from the work by Rousseeuw et al. (1999). The MCD algorithm looks for h data points such that their classical covariance matrix has the minimum determinant when total of n data points are given. The location estimator of MCD algorithm is the mean of these h points, and the MCD estimate of scatter is their covariance matrix. It has a breakdown point $\frac{(\mathrm{n-h})}{\mathrm{n}}$ where $\frac{\mathrm{n}}{2} \leq h < n$. However, MCD estimator takes a lot of computation time for large number of observations and high dimensions (p). To solve this computational difficulty, fast-MCD algorithm were developed. The basic ideas are an inequality involving order statistics and determinants, and techniques which we call selective iteration and nested extensions. It has a breakdown point $(n-h+1)/n$.

2.1. Concentration step

First, let's define the C-step (concentration step) of the fast-MCD algorithm. Consider a dataset $\mathrm{X_n} = \{\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\}$ of p-variate observations. Let $H_1 \subset \{1, \dots, n\}$ with $|\mathrm{H_1}| = h$, and put $\boldsymbol{T_1} \coloneqq (1/h)\sum_{i \in H_1}(\boldsymbol{x_i} - \boldsymbol{T_1})$ and $\boldsymbol{S_1} \coloneqq (1/h)\sum_{i \in H_1}(\boldsymbol{x_i} - \boldsymbol{T_1})(\boldsymbol{x_i} - \boldsymbol{T_1})'$. If $\det(\mathbf{S_1}) \neq \mathbf{0}$, define the relative distances $\mathrm{d_1}(i) \coloneqq \sqrt{(\boldsymbol{x_i} - \boldsymbol{T_1})'\mathbf{S}_\mathbf{1}^{-1}(\boldsymbol{x_i} - \boldsymbol{T_1})}$ for $\mathrm{i} = 1, \dots, \mathrm{n}$. Now take $\mathrm{H_2}$ such that $\{\mathrm{d_1}(i); i \in H_2\} \coloneqq \{(\mathrm{d_1})_{1:n}, \dots, (\mathrm{d_1})_{h:n}\}$, where $(\mathrm{d_1})_{1:n} \leq (\mathrm{d_1})_{2:n} \leq \cdots \leq (\mathrm{d_1})_{n:n}$ are the ordered distances, and compute $\boldsymbol{T_2}$ and $\boldsymbol{S_2}$ based on $\mathrm{H_2}$. Then, $\det(\mathbf{S_2}) \leq \det(\boldsymbol{S_1})$ with equality if and only if $\boldsymbol{T_2} = \boldsymbol{T_1}$ and $\boldsymbol{S_2} = \boldsymbol{S_1}$.

This theorem is called concentration step because we concentrate on the h observations with smallest distances and $\mathbf{S_2}$ is more concentrated (has a lower determinant) than $\boldsymbol{S_1}$. The idea is that we take many initial choices of $\mathrm{H_1}$, repeat C-steps, and stop if $\det(\mathbf{S_2}) = 0$ or $\det(\mathbf{S_2}) =$

$det(\boldsymbol{S_1})$ hence keeping the solution with the lowest determinant. Algorithm 1 describes C-step in algorithmic terms.

Algorithm 1. Pseudocode Concentration step

```
Given the $h$-subset $\mathrm{H_{old}}$ or the pair $(\boldsymbol{T_{old}}, \boldsymbol{S_{old}})$, perform the following:
    1. Compute the distances $\mathrm{d_{old}(i)}$ for $\mathrm{i} = 1, \dots, \mathrm{n}$.
    2. Sort the distances, which yields a permutation $\pi$ for which $\mathrm{d_{old}}(\pi(1)) \leq \mathrm{d_{old}}(\pi(2)) < \cdots < \mathrm{d_{old}}(\pi(1))$.
    3. Put $\mathrm{H_{new}} := \{\pi(1), \pi(2), \dots, \pi(h)\}$.
    4. Compute $\mathbf{T}_{\mathrm{new}} := \mathrm{ave}(H_{new})$ and $\mathbf{S}_{\mathrm{new}} := \mathrm{cov}(\mathrm{H_{new}})$.
```

2.2. Algorithm construction

(**Creating initial Subsets $\boldsymbol{H_1}$**) The question arises how to set the initial subsets H_1. There are two possible ways to set initial subsets. The first situation is when the dataset is univariate (having $p = 1$). In this case, we can draw H_1 at random. On the other hand, when $p \geq 2$, draw a random $(p + 1)$-subset J, and then compute $\boldsymbol{T_0} := avg(J)$ and $\boldsymbol{S_0} := cov(J)$. Then, compute the distances $d_0^2(i) := (\boldsymbol{x_i} - \boldsymbol{T_0})'\boldsymbol{S_0}^{-1}(\boldsymbol{x_i} - \boldsymbol{T_0})$ for $i = 1, \dots, n$. Sort them into $\mathrm{d_0}(\pi(1)) \leq \mathrm{d_0}(\pi(2)) < \cdots < \mathrm{d_0}(\pi(1))$ and put $H_1 := \{\pi(1), \pi(2), \dots, \pi(h)\}$.

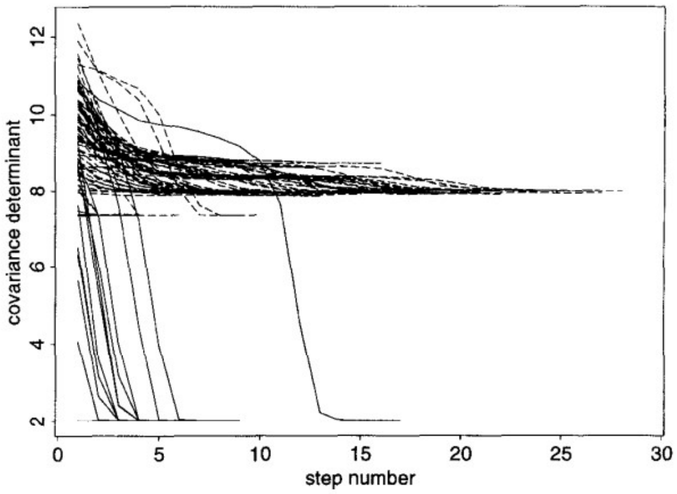

Figure 1. Covariance determinant of subsequent C-steps. Each sequence stops when no further reduction is obtained

(Selective iteration) In each C-step, it contains many calculations such as calculating robust covariance matrix, distances, and determinant. Therefore, reducing the number of C-step would improve the speed of algorithm. Figure 1 is from the Rousseeuw et al. (1999) and for each starting subsample H_1, it shows the value of covariance determinant for each step in concentration step. The dashed line stands for non-robust results and solid line stands for a robust result. From the figure, we can see that after two iterations, many subsamples that will lead to the global optimum already have a rather small determinant. Therefore, two concentration steps would produce enough minimization of determinant for MCD estimator.

(Nested extensions) The computation time not only depends on concentration step, but it also depends on the number of sample size n because for each n, the distance $d(\pi)$ are calculated. To fasten the algorithm, when $n \geq 1{,}500$, nested calculation is introduced. For example, if $n = 1{,}500$, non-overlapping 5 subsets of 300 samples that are sum up to n are formed. To construct each subsample, the algorithm draws 1,500 observations one by one without replacement. The first 300 samples are put in the first subset, and so on. When $n \leq 600$, we just perform selective iteration described in the previous paragraph. When $600 \leq n \leq 1{,}500$, at most four subsamples of 300 or more are formed. Whenever $n > 600$, two C-steps are performed within each subset. For every subset, the best 10 solutions are stored ($\boldsymbol{T_{sub}}, \boldsymbol{S_{sub}}$), take two C-steps, and then pooled together ($\boldsymbol{T_{merged}}, \boldsymbol{S_{merged}}$). Finally, each of these 10 solutions is extended to the full dataset in the same way, and the best solution is produced ($\boldsymbol{T_{full}}, \boldsymbol{S_{full}}$). The detailed steps are described in Algorithm 2.

2.3 Fast MCD pseudocode

The pseudocode for fast-MCD algorithm is described in Algorithm 2.

Algorithm 2. Pseudocode for fast-MCD algorithm

```
1. The default $h$ is $[(n-h+1)/2]$, but the user may choose any integer $h$ with $[(n-h+1)/2] \leq h \leq n$.
2. If $h = n$, then the MCD location estimate $\boldsymbol{T}$ is the average of the whole dataset, and the MCD scatter estimate $\boldsymbol{S}$ is its covariance matrix. Report these and stop.
3. If $p = 1$ (univariate data), compute the MCD estimate $(\boldsymbol{T}, \boldsymbol{S})$ by the exact algorithm of Rousseeuw and Leroy (1987, pp. 171-172); then stop.
4. From here on, $h < n$ and $p \geq 2$. If $n$ is small (say, $n \leq 600$), then
    - Repeat (say) 500 times:
      *Construct an initial $h$-subset $H_1$, starting from a random $(p+1)$-subset;
      *Carry out two C-steps;
    - For the 10 results with lowest $\det(\boldsymbol{S_3})$:
      *Carry out C-steps until convergence;
    - Report the solution $(\boldsymbol{T}, \boldsymbol{S})$ with lowest $\det(\boldsymbol{S})$.
5. If $n$ is larger (say, $n > 600$), then
    - construct up to five disjoint random subsets of size $n_{sub}$ according to "nested extensions" in Section 2.2 (Say, five subsets of size $n_{sub} = 300$);
    - Inside each subset, repeat $500/5 = 100$ times:
      * Construct an initial subset $H_1$ of size $h_{sub} = [n_{sub}(h/n)]$;
      * Carry out two C-steps, using $n_{sub}$ and $h_{sub}$;
      * keep the 10 best results $(\boldsymbol{T_{sub}}, \boldsymbol{S_{sub}})$;
    - Pool the subsets, yielding the merged set (say, of size $n_{merged} = 1{,}500$);
    - In the merged set, repeat for each of the 50 solutions $(\boldsymbol{T_{sub}}, \boldsymbol{S_{sub}})$;
      * Carry out two C-steps, using $n_{merged}$ and $h_{merged} = [n_{merged}(h/n)]$;
      * keep the 10 best results $(\boldsymbol{T_{merged}}, \boldsymbol{S_{merged}})$;
    - In the full dataset, repeat for the $m_{full}$ best result:
      * Take several C-steps, using $n$ and h;
      * keep the best final results $(\boldsymbol{T_{full}}, \boldsymbol{S_{full}})$;

      Here, $m_{full}$ and the number of C-steps depend on how large the dataset is.
```

3. Modified TCLUST Algorithm

The proposed algorithm is a modification of TCLUST algorithm by García et al. (2008). I replaced scatter matrices and cluster centers by a robust covariance matrices and robust covariance matrix location estimators.

3.1. Initialization

For the initial j^{th} cluster weights w_j^0, it is chosen at random. For the parameter p as a dimension of dataset, k as a number of clusters, and q as an arbitrary integer number, we randomly choose $k \times (p+q)$ points. Here, the number q should be chosen to be able to calculate the robust covariance matrix. Generally, q should be larger or equal to 2 and as q increases, the calculated robust covariance matrix better represents each cluster shapes. For each cluster, by using those selected data points, we can calculate the initial robust covariance matrices

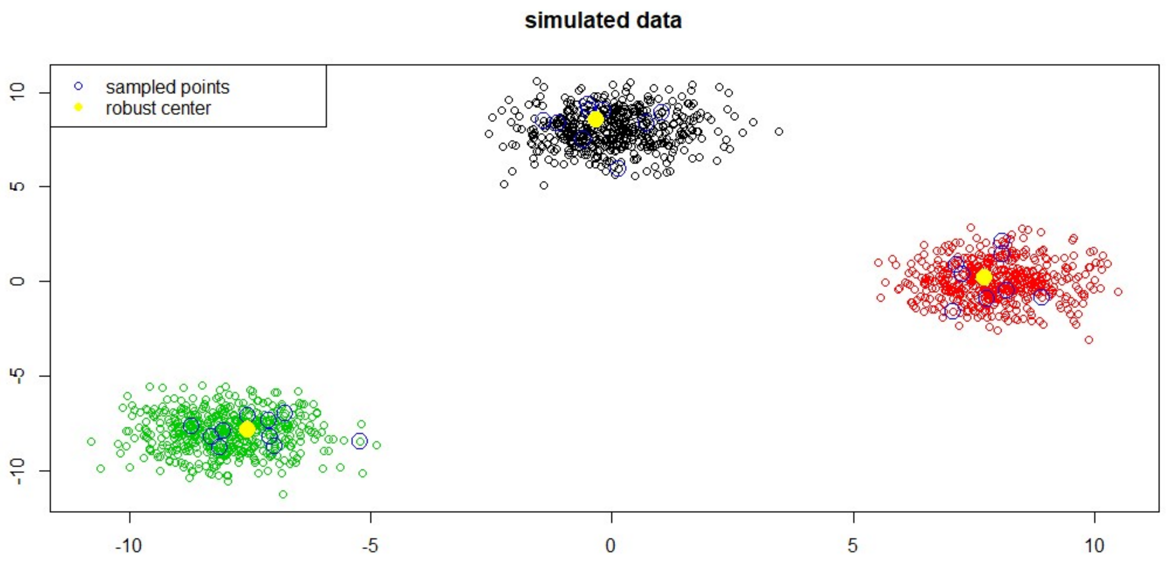

Figure 2. Scatter plot of simulated dataset when $k = 3$, $p = 3$, and $q = 6$. Robust location estimators for each clusters are plotted as yellow circles.

$\boldsymbol{S_j^0}, \mathrm{j} = 1, \dots, \mathrm{k}$. Also, we can calculate robust cluster location estimator $\boldsymbol{\mu_j^0}, \mathrm{j} = 1, \dots, \mathrm{k}$ using estimated covariance matrices.

It is important to have approximately $(p+q)$ observations for each cluster because otherwise, random initial points will poorly represent each cluster. For this, we can use *K*-means algorithm on only $k \times (p+q)$ randomly chosen points. We can calculate cluster assignment of $k \times$

$(p+q)$ points for k clusters, and accept the cluster assignment if the assignments are approximately balanced (each cluster has roughly $(p+q)$ observations) between clusters. In R, this can be done by *"kmeans(x=random_points, centers=k)$cluster"*.

For example, Figure 2 shows the initial robust location estimators when $k=3,\ p=3$, and $q=6$. For each cluster, 8, 10, and 9 points were randomly chosen. This result was accepted by the algorithm because it is approximately balanced to $(p+q)=9$ points for each cluster. We can see that those selected points (blue points) properly represents each cluster. We can calculate robust covariance matrices $\boldsymbol{S}_j^{\boldsymbol{0}}, \mathrm{j}=1,\dots,\mathrm{k}$ based on those sampled points (blue points) and calculate the robust location parameter of each cluster ($\boldsymbol{\mu}_j^{\boldsymbol{0}}, \mathrm{j}=1,\dots,\mathrm{k}$, yellow points) from the robust covariance matrices.

3.2. EM algorithm with concentration step

From the expectation step to maximization step, those steps are repeated until convergence. (i.e., $\theta^{l+1}=\theta^{l}$) or a maximum number of iterations is reached. In the expectation step, the goal is to calculate posterior probabilities for each observation $\mathbf{x}_{\mathrm{i}}$ and assign observations to a cluster which provides maximum posterior probability. Let the distance metric from an observation $\boldsymbol{x}_i$ to the center of cluster j ($\boldsymbol{\mu}_j^{\boldsymbol{l}}$) at l^{th} iteration be $D_j(\boldsymbol{x}_i;\theta^l)=w_j^l\phi\left(\boldsymbol{x}_i;\boldsymbol{\mu}_j^{\boldsymbol{l}},\boldsymbol{S}_j^{\boldsymbol{l}}\right)$. If $D_j(\boldsymbol{x}_i;\theta^l)$ small, the distance of the $\boldsymbol{x}_i$ to $\boldsymbol{\mu}_j^{\boldsymbol{l}}$ is large. The posterior probabilities are defined as $\frac{D_j(x_i;\theta^l)}{\sum_{j=1}^{k}D_j(x_i;\theta^l)}$ for $j=1,\dots,k$ where θ^l is the set of cluster parameters in the iteration *l*.

After we calculate posterior probabilities for each observation, we can trim [nα] observations.

Let $D(\boldsymbol{x}_i;\theta^l)$ be the maximum of $D_j(\boldsymbol{x}_i;\theta^l)$ across all clusters for observation $\boldsymbol{x}_i$. Then, $D(\boldsymbol{x}_i;\theta^l)$ is defined as $D(\boldsymbol{x}_i;\theta^l) = max\{D_1(\boldsymbol{x}_i;\theta^l), \ldots, D_k(\boldsymbol{x}_i;\theta^l)\}$. We can calculate $D(\boldsymbol{x}_i;\theta^l)$ for all of the observations and sort it from smallest to largest. Then we trim the smallest [nα] observations as possible outliers. For the non-outlying observations, we can assign them to the clusters which provides the largest posterior probabilities (i.e. assign $\boldsymbol{x}_i$ to cluster *j* if $D_j(\boldsymbol{x}_i;\theta^l)$=$D(\boldsymbol{x}_i;\theta^l)$). As a result, we will now have the partition $R_0, R_1, \ldots, R_k$ for every observation. Here, R_0 is the partition for outlying observations with smallest $[n\alpha]$ $D(\boldsymbol{x}_i;\theta^l)$'s.

In the maximization step, the parameters $\theta^l = \left(w_1^l, \ldots, w_k^l, \boldsymbol{\mu}_1^{\boldsymbol{l}}, \ldots, \boldsymbol{\mu}_k^{\boldsymbol{l}}, \boldsymbol{S}_1^{\boldsymbol{l}}, \ldots, \boldsymbol{S}_k^{\boldsymbol{l}}\right)$ are updated, based on non-discarded observations and their cluster assignments. Let $n_j = \#R_j$ be the number of observations in cluster *j*. Then the weights are updated by $w_j^{l+1} = n_j/[n(1-\alpha)]$. In other words, we update weights proportional to the number of observations in cluster *j*. Also, with assigned points for each cluster, we can calculate the robust covariance matrices $\boldsymbol{S}_j^{\boldsymbol{0}}, \mathrm{j} = 1, \ldots, \mathrm{k}$. and robust location estimator $\boldsymbol{\mu}_j^{\boldsymbol{0}}, \mathrm{j} = 1, \ldots, \mathrm{k}$ using estimated covariance matrices.

4. Simulation study

To evaluate the performance of clustering by four different robust covariance matrices, I set five different experimental conditions from (M1) to (M5) from García et al. (2008). They mentioned that "Several data sets of size $\mathrm{n} = 500$ have been generated. Each data set consists of three simulated p-dimensional normally distributed clusters with centers $\boldsymbol{\mu_1} = (0, 8, 0, \ldots, 0)'$, $\boldsymbol{\mu_2} = (8, 0, 0, \ldots, 0)'$ and $\boldsymbol{\mu_3} = (-8, -8, 0, \ldots, 0)'$ and covariance matrices"

$$\boldsymbol{\Sigma_1} = diag(1, a, 1, \ldots, 1)'$$

$$\boldsymbol{\Sigma_2} = diag(b, c, 1, \ldots, 1)'$$

and

$$\boldsymbol{\Sigma_1} = \left(\begin{array}{cc|c} d & e & \\ e & f & \mathbf{0} \\ \hline \multicolumn{2}{c}{\mathbf{0}} & \boldsymbol{I} \end{array}\right)$$

and the following experimental conditions.

(M1) (a, b, c, d, e, f) = (1, 1, 1, 1, 0, 1): Spherical equally scattered groups.

(M2) (a, b, c, d, e, f) = (5, 1, 5, 1, 0, 5): Not spherical, same covariance matrices for the groups.

(M3) (a, b, c, d, e, f) = (5, 5, 1, 3, − 2, 3): Different covariance matrices but the same scale

(M4) (a, b, c, d, e, f) = (1, 20, 5, 15, − 10, 15): Groups with different scales.

(M5) (a, b, c, d, e, f) = (1, 45, 30, 15, − 10, 15): Groups with different scales and severe overlap.

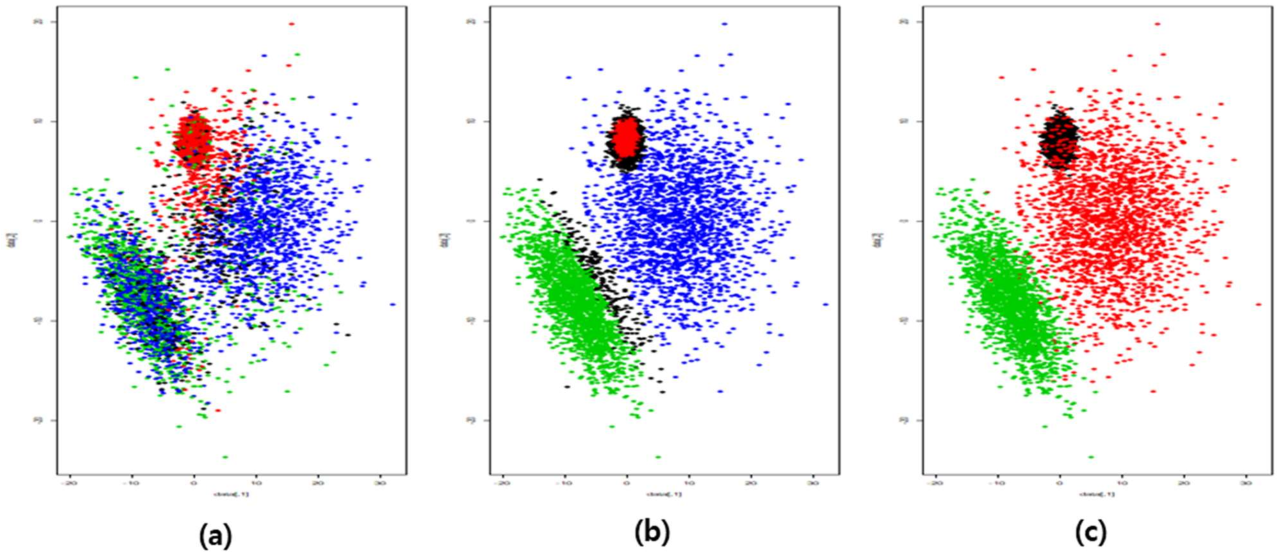

Figure 3. Scatter plots for (a): initial cluster assignment, (b): cluster assignment at iteration 9 and (c): the simulated truth. In (b) black points are trimmed points. Simulation schema (M5) was used with $k = 3$, $p = 6$, $\alpha = 0.1$ and $q = 10$.

In Figure 3, (a) represents the initial cluster assignment, and (b) represents the cluster assignment at iteration 9. The simulation schema (M5) was used with the number of clusters $k = 3$, the data dimension $p = 6$, trimming level $\alpha = 0.1$ and additional $q = 10$ points for calculating robust covariance matrices for each cluster. Compared to the simulated truth (c), we can see that the plot (b) well clustered observations. Most trimming points lied in bridge region (region between clusters). Each clustering results from iteration 1 to iteration 9 can be found in Figure 4. We can see that in the early stage of iteration, the cluster assignments are not accurate and the algorithm trimmed points within clusters. However, as iteration goes, the algorithm gets improved by assigning data points to the right cluster and trimming points in bridge regions.

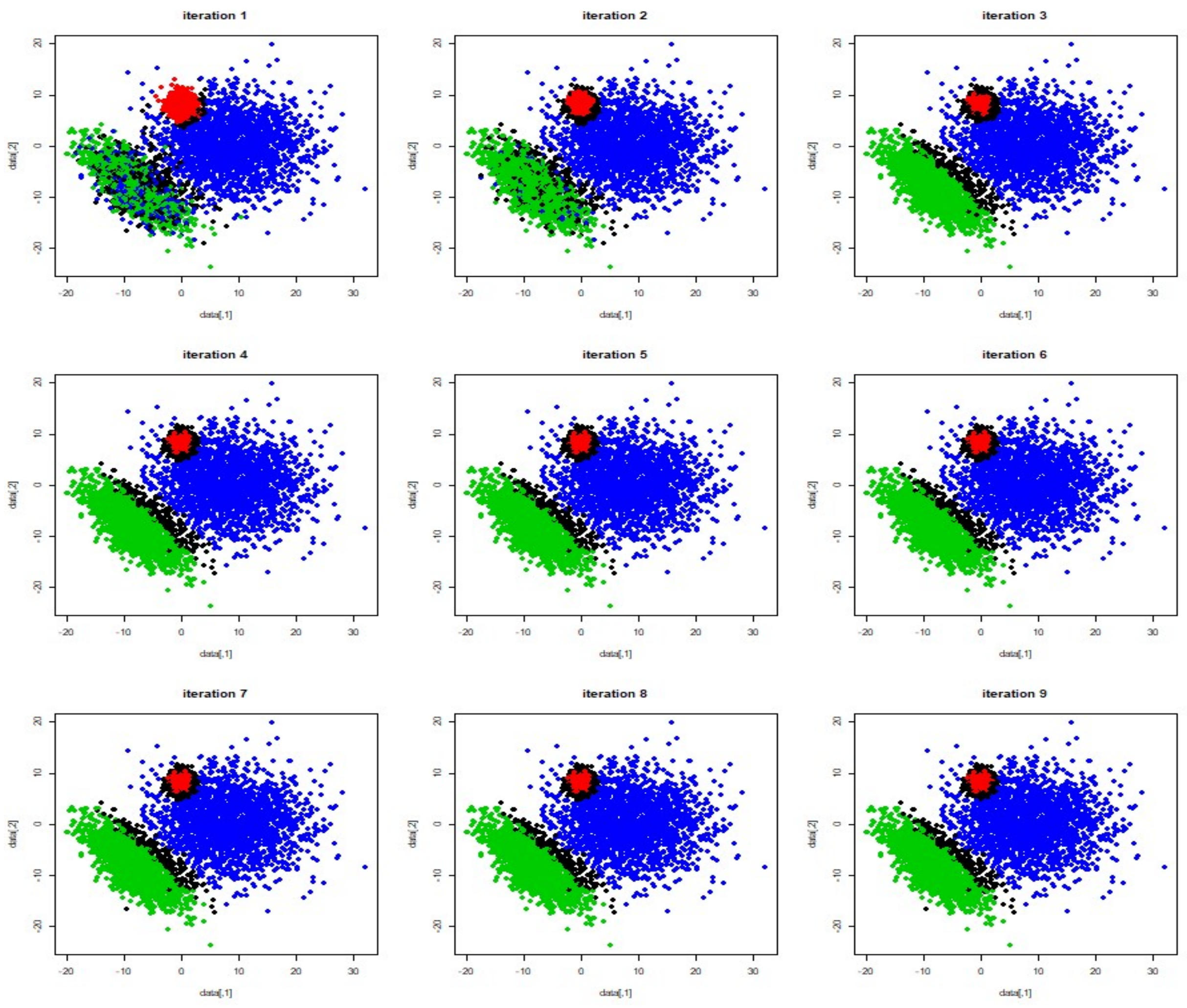

Figure 4. M5 Clustering result when $k = 3$, $p = 6$, $\alpha = 0.1$ and $q = 10$ from iteration 1 to iteration 9. The scatterplot only shows the first two dimensions of the dataset.

Table 1. Misclassification rate for each of the robust covariance estimators. "Mcd", "Weighed", "Donostah", and "M" stands for "fast-MCD" algorithm, "Reweighted MCD", "Donoho-Stahel projection based estimator", and "constrained M estimator" each.

		$\alpha = 0$				$\alpha = 0.1$			
Dimension	**model**	**Mcd**	**Weighted**	**Donostah**	**M**	**Mcd**	**Weighted**	**Donostah**	**M**
$p = 2$	**M1**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M3	0.0002	0.0002	0.0002	0.0002	0.0000	0.0000	0.0000	0.0000
	M4	0.0008	0.0008	0.0008	0.0008	0.0000	0.0000	0.0000	0.0000
	M5	0.0165	0.0165	0.0165	0.0165	0.0054	0.0054	0.0054	0.0054
$p = 6$	**M1**	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	M3	0.0003	0.0003	0.0003	0.0003	0.0000	0.0000	0.0000	0.0000
	M4	0.0017	0.0018	0.0013	0.0018	0.0002	0.0002	0.0002	0.0002
	M5	0.0265	0.0262	0.0258	0.0265	0.0120	0.0117	0.0122	0.0119

Table 1 shows misclassification rate for each of the robust covariance estimators. I compared four different robust covariance estimation methods: "fast-MCD" algorithm, "Reweighted MCD", "Donoho-Stahel projection based estimator", and "constrained M estimator" with non-trimmed, and 10% trimming version. Also, the misclassification rate when using lower dimension dataset ($p = 2$) and higher dimension dataset ($p = 6$) were compared. For all simulation settings from (M1) – (M5), $k = 3,\ q = 10$ were used. Overall, we can see that misclassification rate is lower in trimmed settings than non-trimmed settings. That means, the algorithm trimmed datasets that are easy to misclassify. From the Figure 4, we can see that regions between clusters are trimmed.

Also, we can see that if we assume similar cluster shapes between clusters such as M1(spherical equally scattered groups) and M2 (not spherical, same covariance matrices for the groups) the modified TCLUST algorithm had perfect classification. As the clusters has different shapes and overlap, the proposed algorithm started to misclassify. Especially in M5 (groups with different scales and severe overlap), it has the worst classification rate. Overall, "Donoho-Stahel projection based estimator" showed the smallest misclassification rate, however, all other methods worked well, too.

5. Conclusion

In this report, I have discussed alternative approach to existing TCLUST algorithm. First, we have looked at the mathematics behind the "fast-MCD" algorithm, which is one of the robust covariance estimation methods we compared. In "fast-MCD", it has a breakdown point $(n-h+1)/n$ which is higher than the traditional "MCD" algorithm $(n-h)/n$. By replacing scatter matrices from TCLUST algorithm to robust covariance matrix from "fast-MCD", we can have another powerful clustering method that are hardly effected by outliers and better initial start than traditional TCLUST algorithm. Same idea of replacing scatter matrix in TCLUST to robust covariance matrix applies for other three algorithms: "Reweighted MCD", "Donoho-Stahel projection based estimator", and "constrained M estimator". In the simulation study, while all of the robust covariance estimation methods compared showed fairly good performance, "Donoho-Stahel projection based estimator" showed the minimum misclassification rate.

References

- Böhm, C., Faloutsos, C., Pan, J. Y., & Plant, C. (2006, August). Robust information-theoretic clustering. *In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 65-75). ACM.
- Celeux, G., & Govaert, G. (1992). A classification EM algorithm for clustering and two stochastic versions. *Computational statistics & Data analysis*, *14*(3), 315-332.
- Fritz, H., GarcíA-Escudero, L. A., & Mayo-Iscar, A. (2013). A fast algorithm for robust constrained clustering. *Computational Statistics & Data Analysis*, *61*, 124-136.

- García-Escudero, L. A., Gordaliza, A., Matrán, C., & Mayo-Iscar, A. (2008). A general trimming approach to robust cluster analysis. *The Annals of Statistics*, *36*(3), 1324-1345.
- Leroy, A. M., & Rousseeuw, P. J. (1987). Robust regression and outlier detection. *Wiley Series in Probability and Mathematical Statistics, New York: Wiley*, 1987.
- Rousseeuw, P. J., & Driessen, K. V. (1999). A fast algorithm for the minimum covariance determinant estimator. *Technometrics, 41*(3), 212-223.

Supplementary document

1. Rcode

```r
rm(list=ls())
library(mvtnorm);library(robustbase);library(robust)
##########
# define function
##########
add <- function(x) Reduce("+", x)

formmatrix = function(a,b,c,d,e,f,p){
  m1 = c(0,8,rep(0, p-2))
  m2 = c(8,0,rep(0, p-2))
  m3 = c(-8,-8,rep(0, p-2))

  sig1 = diag(c(1,a, rep(1, p-2)))
  sig2 = diag(c(b,c, rep(1, p-2)))
  sig3 = matrix(c(d,e,e,f),2,2,byrow=T)
  sig3 = cbind(sig3, matrix(rep(0,2*(p-2)),2))
  sig3 = rbind(sig3, cbind(t(matrix(rep(0,2*(p-2)),2)), diag(rep(1, p-2))))

  return(list(m1=m1, m2=m2, m3=m3, sig1=sig1, sig2=sig2, sig3=sig3))
}

formdata = function(n, sim1){
  dat1 = rmvnorm(n = n, mean = sim1$m1, sigma = sim1$sig1)
  dat2 = rmvnorm(n = n, mean = sim1$m2, sigma = sim1$sig2)
  dat3 = rmvnorm(n = n, mean = sim1$m3, sigma = sim1$sig3)

  dat1 = cbind(dat1, rep(1,n))
  dat2 = cbind(dat2, rep(2,n))
  dat3 = cbind(dat3, rep(3,n))

  data = rbind(dat1, dat2, dat3)
  return(data)
}

##########
# simulation study
##########
for(xx in 1:4){
  set.seed(1234)
  par(mfrow=c(1,1))
  k = 3 #number of cluster
  p=6 #dimension #########################
  alpha = 0.1#trimming level
  q = 10 ######################
  numsim = 2000
  estim=c("mcd", "weighted", "donostah", "M")[xx]
  sim1 = formmatrix(1,1,1,1,0,1, p)
  sim2 = formmatrix(5,1,5,1,0,5, p)
  sim3 = formmatrix(5,5,1,3,-2,3, p)
  sim4 = formmatrix(1,20,5,15,-10,15, p)
  sim5 = formmatrix(1, 45, 30, 15, -10, 15, p)
  sim=sim1
  simnum=1

  ##########
  # plot data
  ##########
  data2 = formdata(numsim, sim)
  data = data2[,-(p+1)]
  n = dim(data)[1]
```

```r
##########
# Algorithm
##########
#1. Initialization
# Randomly choose weights w
rnum = runif(k)
w = rnum/sum(rnum)

for(ii in 1:100){
  # randomly select init.num =  k*(p+1)
  init.num =  k*(p+q)
  idx = sample(1:dim(data)[1], init.num, replace = F)
  init.pt = data[idx,]

  # validate with kmeans
  km = kmeans(init.pt, k)
  if(sum(table(km$cluster)>=rep((p+3),k))==k){
    print("Properly sampled initial points")
    break;
  }
}

init.idx3 = lapply(1:k, function(x){which(x==km$cluster)})

#(Version 1) calculate initial sample cov: tmat
tmat = lapply(init.idx3, function(x){covClassic(init.pt[x,])})

#(Version 2) use robust cov matrix for initial S
smat = lapply(init.idx3, function(x){covRob(init.pt[x,], estim = estim)})

#visualize
model = lapply(init.idx3, function(x){
  fit.models(list(Robust="covRob", Classical="covClassic"),
             data=init.pt[x,])
})

#calculate initial mean mu
mu = sapply(smat, function(x){x$center})

#### Calculate distance metric (with w(weight), mu, smat)
D = vector("list", k)
for(ii in 1:k){
  D[[ii]] = w[ii]*dmvnorm(data, mean = mu[,ii], sigma = smat[[ii]]$cov)
}

D.nrom = add(D)
post.p = sapply(D, function(x){x/D.nrom}) #data will be assigned to max post.p
max.p = apply(post.p, 1, max)
new.assign = unlist(apply(post.p, 1, which.max))

#trim [n*alpha] observations with smallest max.p
if(floor(n*alpha)!=0){
  idx.trim = order(max.p)[1:floor(n*alpha)]
  new.assign[idx.trim] = 0
}

#points(data, col=factor(new.assign))
old_assign = new.assign

##########
#Loop until convergence / max.iter
##########
max.iter=9
png(filename = paste0("C://Users//User//Dropbox//2019.Spring//STAT556_1_Robust//05_Project2//image//",
                      simnum, "_", p,"_",q,"_",estim,"_",alpha,"_2.png"), width = 900, height = 900)
par(mfrow=c(3,3))
```

```r
for(nn in 1:max.iter){
  #update w
  w = table(new.assign)/length(new.assign)

  #update smat
  smat = lapply(1:k, function(x){covRob(data[new.assign==x,])})
  #covRob(data[new.assign==3,])

  #update mu
  #mu = sapply(1:k, function(x){colMeans(data[new.assign==x,])})
  mu = sapply(smat, function(x){x$center})

  #visualize
  model = lapply(1:k, function(x){
    fit.models(list(Robust="covRob", Classical="covClassic"),
               data=data[new.assign==x,])
  })

  #### Calculate distance metric (with w(weight), mu, smat)
  D = vector("list", k)
  for(ii in 1:k){
    D[[ii]] = w[ii]*dmvnorm(data, mean = mu[,ii], sigma = smat[[ii]]$cov)
  }

  D.nrom = add(D)

  post.p = sapply(D, function(x){x/D.nrom}) #data will be assigned to max post.p

  max.p = apply(post.p, 1, max)
  new.assign = apply(post.p, 1, which.max)

  #trim [n*alpha] observations with smallest max.p
  if(floor(n*alpha)!=0){
    idx.trim = order(max.p)[1:floor(n*alpha)]
    new.assign[idx.trim] = 0
  }
  plot(data, col=factor(new.assign), pch=19, main=paste0("iteration ", nn))

}
dev.off()

##########
# Plot results
##########
png(filename = paste0("C://Users//User//Dropbox//2019.Spring//STAT556_1_Robust//05_Project2//image//",
                      simnum, "_", p,"_",q,"_",estim,"_",alpha,".png"), width = 900, height = 900)
par(mfrow=c(1,3))
plot(data, col=factor(old_assign), pch=19, main="Initial assignment")
plot(data, col=factor(new.assign), pch=19, main="At 9th iteration")
plot(data, col=factor(data2[,p+1]), pch=19, main="Truth")
dev.off()

#Misclassification rate
new.assign2 = new.assign
true = data2[,p+1]

plot(data[new.assign==1,], pch=19, main="At 9th iteration", xlim=c(-100,100), ylim=c(-100,100))
points(data[data2[,p+1]==1,],col="blue", pch=1, main="At 9th iteration", xlim=c(-100,100), ylim=c(-
100,100))

new.assign2[new.assign2==1] = "a"
true[true==1] = "a"

plot(data[new.assign==2,], pch=19, main="At 9th iteration", xlim=c(-100,100), ylim=c(-100,100))
points(data[data2[,p+1]==3,],col="blue", pch=1, main="At 9th iteration", xlim=c(-100,100), ylim=c(-
100,100))
```

```
  new.assign2[new.assign2==2] = "b"
  true[true==3] = "b"

  plot(data[new.assign==3,], pch=19, main="At 9th iteration", xlim=c(-100,100), ylim=c(-100,100))
  points(data[data2[,p+1]==2,],col="blue", pch=1, main="At 9th iteration", xlim=c(-100,100), ylim=c(-
100,100))

  new.assign2[new.assign2==3] = "c"
  true[true==2] = "c"

  if(alpha!=0){
    par(mfrow=c(1,2))
    plot(data[-idx.trim,], col=factor(new.assign2[-idx.trim]), pch=19, main="At 9th iteration")
    plot(data, col=factor(true), pch=19, main="Truth")

    MR = sum(new.assign2[-idx.trim]!=(true[-idx.trim]))/length(true[-idx.trim])

  }else{
    par(mfrow=c(1,2))
    plot(data, col=factor(new.assign2), pch=19, main="At 9th iteration")
    plot(data, col=factor(true), pch=19, main="Truth")

    MR = sum(new.assign2!=(true))/length(true)
  }
  MR
  result = c()
  result = rbind(result,c(simnum, p, q, estim, MR))
  #result2= c()
  result2 = rbind(result2, result)
  result2
}
```