ON K-MEANS CLUSTERING USING MAHALANOBIS DISTANCE

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Joshua David Nelson

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Statistics

April 2012

Fargo, North Dakota

North Dakota State University
Graduate School

Title

On K-Means Clustering Using Mahalanobis Distance

By

Joshua David Nelson

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Volodymyr Melnykov
Chair

Rhonda Magel

Seung Won Hyun

James Coykendall

Approved:

5/7/2012	Rhonda Magel
Date	Department Chair

ABSTRACT

A problem that arises quite frequently in statistics is that of identifying groups, or clusters, of data within a population or sample. The most widely used procedure to identify clusters in a set of observations is known as K-Means. The main limitation of this algorithm is that it uses the Euclidean distance metric to assign points to clusters. Hence, this algorithm operates well only if the covariance structures of the clusters are nearly spherical and homogeneous in nature. To remedy this shortfall in the K-Means algorithm the Mahalanobis distance metric was used to capture the variance structure of the clusters. The issue with using Mahalanobis distances is that the accuracy of the distance is sensitive to initialization. If this method serves as a significant improvement over its competitors, then it will provide a useful tool for analyzing clusters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION TO CLUSTER ANALYSIS

The process of separating observations into clusters is a very fundamental problem at the heart of statistics. Clusters are characterized by groups of data points which are in "close" proximity to one another. Data clusters are frequently encountered when sampling from a population because there may exist several distinct subpopulations within the population whose responses vary considerably from one another. While it is much easier to visually detect clusters in univariate or bivariate data, the task becomes increasingly difficult as the dimensionality of the data increases. Currently, there is no optimal clustering method for every scenario. It is for this reason that many clustering algorithms exist today.

There are three popular approaches to clustering: hierarchical clustering, partitional clustering, and model-based clustering, each with its own strengths and drawbacks. We will now provide a brief overview of these strategies.

1.1. Hierarchical Clustering

One approach when forming clusters is to use a hierarchical technique. Hierarchical techniques come in two varieties: agglomerative and divisive. Both methods require a means to evaluate the "closeness" between points, and also between clusters at each step. To determine the closeness between individual points, a distance metric is required. A few common choices of distance metric are shown below [1].

Let $x, y \in \mathbb{R}^p$, where $x = (x_1, x_2, \ldots, x_p)^T$ and $y = (y_1, y_2, \ldots, y_p)^T$. Then we define the following distance metrics:

1. **Euclidean Distance**

$$D(x, y) = ||x - y|| = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2},$$

2. **Manhattan Distance**

$$D_1(x, y) = \sum_{i=1}^{p} |x_i - y_i|,$$

3. **Mahalanobis Distance**

$$D_m(x, y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^T \mathbf{\Sigma}^{-1} (x_i - y_i)}.$$

In addition to these measures of distance between individual points, it is necessary to have a distance measure between clusters in order to decide whether or not they should be merged. There are several intercluster distance measures, called linkages, that may be used when merging clusters. Below are some routinely used linkage criteria.

Let A and B be distinct clusters. Then we define the following:

1. **Complete Linkage**

$$max\{D(a, b) | a \in A, b \in B\},$$

2. **Single Linkage**

$$min\{D(a, b) | a \in A, b \in B\},$$

3. **Average Linkage**

$$\frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} D(a, b).$$

Agglomerative clustering takes a bottom-up approach by assigning each of the N points in the data set to its own cluster, then combining the clusters systematically until only one cluster remains. At each step, the two closest clusters are merged together using either a distance or linkage measure, whichever is appropriate. In

contrast, divisive clustering begins by assigning all N points to one cluster and dividing them into smaller clusters until each point is its own cluster [1]. There exist several algorithms for both of these types of clustering methods, but those will not be discussed here.

There are two benefits of employing a hierarchical clustering technique. Firstly, the number of clusters does not need to be specified in order for the algorithm to work. The tree-like structure allows the user to decide a logical stopping point for the algorithm. Another feature of such an approach is the nested nature of the clusters. In this way, there is a convenient, hierarchical ordering of the clusters. A disadvantage of hierarchical strategies is also related to the nested nature of the clusters produced. Since the clusters are nested, there is no way to correct an erroneous step in the algorithm. Therefore, hierarchical clustering is an advantageous approach when it is known a priori that the clusters have a particular structure. The next section will cover a clustering method that uses information about the underlying distributions of the clusters to obtain classifications for data points.

1.2. Model-Based Clustering

The idea behind mixture modeling is represent each individual cluster with its own distribution function. Usually, each distribution, commonly referred to as a mixing component, takes the form of a Gaussian distribution. Suppose there are K clusters in our data set. If our data set is p-dimensional the distribution for k^{th} mixing component takes the form:

$$f_k(x) = \frac{1}{\sqrt{(2\pi)^p \left|\boldsymbol{\Sigma}_k\right|}} \exp\left(-\frac{1}{2}(x-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(x-\boldsymbol{\mu}_k)\right)$$

However, $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K\}$ and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \ldots, \boldsymbol{\Sigma}_K\}$ are unknown, so they must be estimated. The mixture itself is obtained by taking the sum of these mixing components together with their respective weights $\pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$, where

$\sum_{k=1}^{K} \pi_k = 1, 0 < \pi_k \leq 1.$

$$f(\boldsymbol{x}_j|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{k=1}^{K} \pi_k f_k(x|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Using this mixture distribution function, it is easy to construct the likelihood function for our set of observations:

$$L(x) = \prod_{j=1}^{N} f\left(\boldsymbol{x}_j|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$$

In order to form the clusters, each point is assigned a probability of belonging to each of the K clusters called a posterior probability. The posterior probability of observation $\boldsymbol{x}_j$ belonging to cluster C_k is:

$$\widehat{P}\left(C_k|\boldsymbol{x}_j\right) = \frac{\widehat{\pi}_k f\left(\boldsymbol{x}_j, \widehat{\boldsymbol{\mu}}_k, \widehat{\boldsymbol{\Sigma}}_k\right)}{f\left(\boldsymbol{x}_j\right)},$$

where $\widehat{\pi}_k$, $\widehat{\boldsymbol{\mu}}_k$, and $\widehat{\boldsymbol{\Sigma}}_k$ are given by:

$$\widehat{\pi}_k = \frac{1}{N} \sum_{j=1}^{N} \widehat{P}\left(C_k|\boldsymbol{x}_j\right), k = 1, 2, \ldots, K-1,$$

$$\widehat{\boldsymbol{\mu}}_k = \frac{1}{\widehat{\pi}_k N} \sum_{j=1}^{N} \boldsymbol{x}_j \widehat{P}\left(C_k|\boldsymbol{x}_j\right), k = 1, 2, \ldots, K,$$

$$\widehat{\boldsymbol{\Sigma}}_k = \frac{1}{\widehat{\pi}_k N} \sum_{j=1}^{N} \left(\boldsymbol{x}_j - \widehat{\boldsymbol{\mu}}_k\right)\left(\boldsymbol{x}_j - \widehat{\boldsymbol{\mu}}_k\right)^T \widehat{P}\left(C_k|\boldsymbol{x}_j\right), k = 1, 2, \ldots, K.$$

Posterior probabilities and likelihood functions are calculated iteratively, so they are either impossible or very arduous to maximize analytically. Therefore, a numerical method is often employed instead to locate a local maximum in the likelihood function. One such method is the Expectation-Maximization (EM) algorithm [1, 2].

1.3. Partitional Clustering

Partitional clustering is another good approach when the number of clusters, K, is known. There are two partitioning algorithms that operate in similar ways: K-Means and K-Medoids. K-Medoids will be discussed briefly; however, K-Means is our primary interest in this paper, so it will receive more attention in this section. The K-Medoids algorithm is as follows:

1. Select K initial medoids at random from the N points in the data set.

2. Assign each point in the data set to the closest of the chosen K initial medoids. Here, a commonly used distance metric is the Euclidean distance. These K sets of points are the new medoids.

3. For each medoid c_k, and for each non-medoid point P, swap c_k and P, then compute the total cost of the new configuration, where the cost is given by $C = \sum_{k=1}^{K} \sum_{j=1}^{n_k} ||x_{kj} - c_k||$.

4. Choose the configuration whose cost is the minimum of all configurations.

5. Repeat steps 2 to 5 until there is no change in the medoids.

The K-Medoids algorithm is flexible in that medoids can be swapped at each step. The cost can be computed using any of the distances mentioned in the previous section, and ultimately the choice of distance will affect the shape of the clusters [3, 4, 5].

A staple of cluster analysis, and one of the most commonly used algorithms today, is the K-Means algorithm. The K-Means algorithm was developed by Stuart Lloyd in 1957 in his later published paper regarding Pulse-Code Modulation and researched further by James MacQueen in his 1967 analysis of the algorithm [6]. The algorithm does not make any explicit assumptions about the distribution of

the clusters; however, K-Means operates under the assumption that the number of clusters K is fixed or known a priori, and assigns points into clusters in a way that tries to minimize the cost function given by:

$$\boxed{C = \sum_{k=1}^{K} \sum_{x_j \in C_k} ||x_j - \widehat{\boldsymbol{\mu}}_k||^2} \tag{1}$$

where C_k represents the set of data points assigned to the k^{th} cluster, and $\widehat{\boldsymbol{\mu}}_k$ is the mean vector for the k^{th} cluster [7, 8]. The algorithm proposed by Lloyd works in the following way:

1. Select K points at random from the N points in the data set. These will be the initial cluster centers.

2. Assign each point in the data set to the nearest cluster center.

3. Recalculate the mean vector for each cluster using the points assigned in the previous step.

4. Repeat steps #2 and #3 until the clusters do not change.

One way to visualize how the algorithm works is to create a Voronoi diagram. Let $S = \{x_1, x_2, \ldots, x_N\}$ be a set of points existing in a space X containing some measure of distance. Then, the region $C_k \subseteq X$ is the set of points in X which are closest to x_j. In the diagram below, the cluster centers are shown. Any other points in the data set that fall within the dotted boundaries are assigned to the corresponding cluster. Within each region, the centroid of the points is calculated, and the diagram is redrawn for the new cluster centers.

A notable advantage of K-Means is that the algorithm is non-deterministic. The algorithm also works very fast due to its simplicity. This feature allows K-Means to be run numerous times until a satisfactory cluster configuration is reached. However,

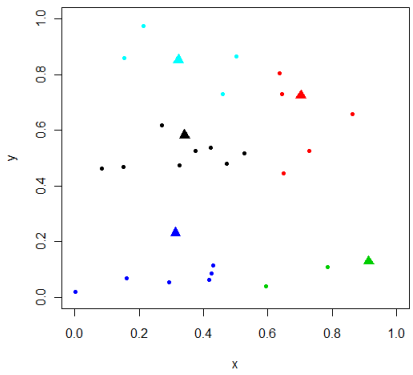

Figure 1. An illustration of how the K-Means algorithm would partition the points into clusters during a single iteration.

it is easy to see that such an approach requires more memory and computational time. Regardless, it is customary to rerun the algorithm several times in order to overcome the drawback of picking initial cluster centers at random, and many implementations of K-Means have a feature allowing the user to specify the number of iterations to be run. Due to the stochastic nature of the algorithm, much attention is paid to how K-Means is initialized. Since K-Means only locates a local minimum rather than a global minimum for the cost function, bad initializations can lead the algorithm to poor classifications. However, this shortfall can be mitigated to some extent by initializing the clusters in a more intelligent manner. Several approaches have been proposed, and some of them will be discussed below.

- The Forgy Approach (FA) was proposed by Forgy in 1965. The FA method initializes the K-Means algorithm by choosing K observations at random from the data set and designates them as initial cluster centers. The remaining $N-K$ points in the data set are then assigned to the closest cluster center to form the initial partitioning [9].

- The MacQueen Approach (MA) is a slight variation of the Forgy Approach devised by MacQueen in 1967. The initial K cluster centers are chosen at random

from the data set. In order to assign the remaining points into clusters, the MA method takes the first observation among the remaining $N-K$ unassigned points and assigns it to the cluster whose center is nearest. Next, the cluster centers are recalculated by taking the centroid of the points assigned to each cluster. This process is repeated with each remaining observation until all N points are assigned to a cluster [10].

- Kaufman and Rousseeuw developed another sequential initialization technique in 1990. Their method begins by taking the observation located closest to the center of the data set and using this point as the first initial cluster center. Then, the second cluster center is chosen in a fashion such that the total within sum of squares is reduced the most. This process is repeated until K cluster centers are chosen [11].

The biggest disadvantage of the K-Means algorithm is that it is ill-equipped to handle non-spherical clusters. This arises due to the distance metric chosen in the cost function, most commonly the Euclidean distance. This choice of metric allows the algorithm to complete computations quickly and is a very intuitive measure of distance; however, it is not always suitable for clustering. In many cases, a data set may contain clusters whose eigenvalues and eigenvectors differ substantially. A broader approach to clustering should be able to handle these types of situations without any explicit distributional assumptions about the data, as in model-based clustering methods. We will explore another option in the case where the clusters to be classified have a more elongated, ellipsoidal structure.

2. K-MEANS USING MAHALANOBIS DISTANCES

Improving accuracy of the K-Means algorithm means that two problems need to be addressed: the initialization procedure should be altered to select points close to cluster centers, and a distance metric must be used which is better equipped to handle non-spherical clusters [12]. To handle the first issue, points which have more neighbors will be favored in the initialization step. This will help the algorithm to choose points which are near the centers of clusters where the density is the highest. Next, the K-Means algorithm will be adapted to use the Mahalanobis distance metric in place of the Euclidean distance metric. The Mahalanobis distance metric will allow K-Means to identify and correctly classify non-homogeneous, non-spherical clusters.

It is easy to see how the Mahalanobis distance is a generalization of the Euclidean distance. The Euclidean distance D can be written as:

$$D(x,y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2} = \sqrt{(x-y)^T(x-y)} = \sqrt{(x-y)^T I_{(p\times p)}^{-1}(x-y)}.$$

Hence, the Euclidean distance tacitly assumes that $\mathbf{\Sigma} = I_{(p\times p)}$. By allowing the covariance matrix to take on a more general form

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp}, \end{pmatrix}$$

the clusters can take a larger variety of shapes.

Intuitively, we can think of the Mahalanobis distance from a point to its respective cluster center as its Euclidean distance divided by the square root of the variance

in the direction of the point. The Mahalanobis distance metric is preferable to the Euclidean distance metric because it allows for some flexibility in the structure of the clusters and takes into account variances and covariances amongst the variables. Two different initialization procedures will be developed using the Mahalanobis distance metric. The first will involve using Euclidean distances to generate initial clusters as in the traditional K-Means algorithm.

Mahalanobis K-Means Initialization without Stretching:

1. Pick K points at random from the data set to be the initial clusters.

2. Calculate the Euclidean distance from each point in the data set to each cluster center.

3. Form the initial clusters by assigning each point to the cluster center whose distance is the least of the k distances.

It can be shown that if $X \sim \mathbf{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ then $D_m^2 \sim \chi_p^2$. Since the estimates $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Sigma}}$ available for each cluster rather than the true cluster parameters, $D_m^2 \sim \chi_p^2$ asymptotically. Using this information, we can develop another algorithm to classify observations into clusters. We introduce a novel approach to initialization by "stretching" the clusters, which will improve the cluster covariance matrix estimates.

Mahalanobis K-Means Initialization with Stretching:

1. Form initial clusters of size w by locating a point near the mode of a cluster and taking the $w-1$ nearest points to it.

2. Using these w points, estimate $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Sigma}}$ for the cluster.

3. Find the 95% confidence ellipsoid for this cluster and pick up any additional points falling within the ellipsoid.

4. Update $\widehat{\boldsymbol{\mu}}$ and $\widehat{\boldsymbol{\Sigma}}$ using these additional points.

5. Repeat steps 3 and 4 until no additional points are picked up by the confidence ellipsoid.

6. Next, remove these points from the data set and steps 1-5 until K initial clusters are formed.

With these initialization procedures in mind, we introduce the two versions of Mahalanobis K-Means denoted *MK-Means1* (without stretching) and *MK-Means2* (with stretching).

MK-Means1 Algorithm

1. Pick K points at random from the data set to be the initial clusters.

2. Calculate the Euclidean distance from each point in the data set to each cluster center.

3. Form the initial clusters by assigning each point to the cluster center whose distance is the least of the k distances.

4. Next, calculate the Mahalanobis distance from each cluster center to each of the N data points and assign each point to the nearest cluster center.

5. Recalculate $\widehat{\boldsymbol{\mu}}_k$ and $\widehat{\boldsymbol{\Sigma}}_k$ for $k = 1, \ldots, K$ and repeat step 5 until the clusters do not change.

MK-Means2 Algorithm

1. Locate the nearest w points to each point in the data set and calculate the sum of these distances. Sort these summed distances from least to greatest.

2. Generate a random variable R from a multinomial distribution with weights $cn^2, c\left(n-1\right)^2, \ldots, c\left(1\right)^2$, where $c = \left(\frac{1}{2}\sum_{j=1}^{n} j\right)^{-1}$.

3. Select the R^{th} element of the summed distance list and designate it as a cluster center, then remove this observation as well as its $w-1$ closest neighbors so that they may are not eligible as candidates for the next cluster center. Repeat steps 1 through 3 until K cluster centers are chosen.

4. For each cluster center, take the $w-1$ closest points and form the initial clusters. Use these points to calculate the initial estimates of $\widehat{\boldsymbol{\mu}}_k$ and $\widehat{\boldsymbol{\Sigma}}_k$ for $k = 1, \ldots, K$.

5. Using the estimates of $\widehat{\boldsymbol{\mu}}_k$ and $\widehat{\boldsymbol{\Sigma}}_k$, locate all points in the data set satisfying $(x_j - \widehat{\boldsymbol{\mu}}_k)^T \widehat{\boldsymbol{\Sigma}}_k^{-1} (x_j - \widehat{\boldsymbol{\mu}}_k) \leq \chi_p^2(0.05)$, where $j = 1, \ldots, n$ and $k = 1, \ldots, K$ and include them into their corresponding cluster.

6. Update the cluster means and variances $\widehat{\boldsymbol{\mu}}_k$ and $\widehat{\boldsymbol{\Sigma}}_k$ for each k and repeat steps 5 and 6 until the clusters do not change.

7. Next, calculate the Mahalanobis distance from each cluster center to each of the remaining $N-K$ data points and assign each point to the nearest cluster center.

8. Recalculate $\widehat{\boldsymbol{\mu}}_k$ and $\widehat{\boldsymbol{\Sigma}}_k$ for $k = 1, \ldots, K$ relying on the new partitioning and repeat steps 7 and 8 until the clusters do not change.

The second algorithm is a bit more complicated in the initialization step. It requires the construction of confidence ellipsoids, which will be used to pick up more points in the cluster. However, the advantage is that the "stretching" of the original spheres will likely pick up more observations along the eigenvector corresponding to the largest eigenvalue of the cluster, which will allow the algorithm to better approximate an initial covariance matrix for each cluster. This step is crucial because poor estimates of cluster covariance matrices can lead to inaccurate distance computations, affecting the final shape of the clusters as in K-Means. Figure 2 illustrates how a

cluster that is initially spherical expands to become more elliptical and better reflects the shape of the cluster after the stretching process.

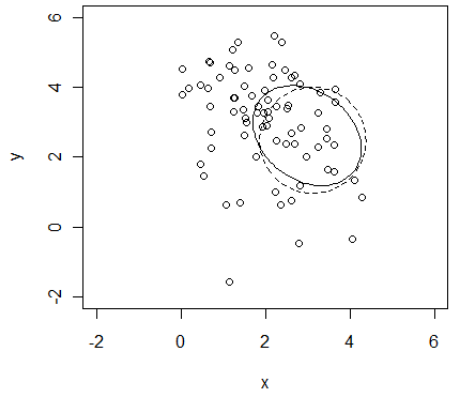

Figure 2. The smaller of the two rings is the 95% confidence ellipsoid of the spherical cluster, and the larger of the two rings is the 95% confidence ellipsoid of the cluster after the stretching step.

The initialization shown in the figure does not capture the entirety of the cluster; however, this is not a major issue. It does capture the general shape of the cluster which is the more important aspect in terms of calculating Mahalanobis distances. While more computationally expensive, initializing spherical clusters in this manner will help to ensure that initial cluster centers are chosen in regions with a greater concentration of points with high probability. Typically, several runs of the algorithm are used to obtain different cluster configurations, as in K-Means. The selection process of initial cluster centers is strengthened by the removal of neighboring points because the likelihood of selecting two points from the same cluster is reduced greatly by removing a high density subset of the cluster. However, the possibility of this occurring increases if the clusters are not homogeneous, i.e. if the points in the data set are distributed very unevenly among clusters. In the case where clusters are non-homogeneous, the value of w may be substantially lower than the number of points

n_k in cluster k, and care must be taken to ensure that a suitable realization for the initial cluster centers is found.

Another issue that arises due to bad initializations is cluster absorption. If an outlying point P is chosen by chance to be a cluster center in the initialization step, the estimate of the covariance matrix will be inflated. This is because the cluster will be formed by taking the nearest $w-1$ points to P, which will likely be further away than if P were chosen closer to a high density region of points. If two clusters are close enough together, it could be the case that the initial cluster around P will pick up several points from each cluster. This is a problematic feature of the algorithm because an inflated covariance matrix will cause the cluster to expand further and potentially absorb two or more clusters.

In the next section, we will conduct a simulation study to compare the accuracy of K-Means and the two Mahalanobis K-Means algorithms that have been developed. The three algorithms will also be compared side by side using an actual data set. To assess the accuracy of the two algorithms, it is necessary to have a decision criterion for which run of the algorithm should be used. For both algorithms, equation 1 on page 6 will be used to determine the best run.

3. SIMULATION STUDY

The K-Means algorithm and the Mahalanobis K-Means algorithm were compared side by side in a simulation study. The study took into account various parameter settings for the clusters generated to test the performance of each algorithm. For each setting, 1000 data sets were generated with known classification. The algorithms ignored this information and proceeded to classify the observations into clusters. To compare the three algorithms, the cluster classifications obtained by each were compared against the known cluster classifications, and the proportion of correctly classified observations was calculated for each algorithm.

Simulations were implemented using R statistical software using the MixSim [13] package. This package allows us to simulate mixture models with multivariate normal component distributions from which observations are drawn. We are then able to run the three algorithms which will decide cluster membership for the points. Resulting clusters for each of the three algorithms can then be compared to the actual cluster membership of the points.

There were several parameters studied including the number of clusters, clusters' mixing proportions and overlap between clusters. The maximum pairwise overlap, denoted by Ω_{max}, is the probability of misclassification. Thus, the higher Ω_{max} is, the harder it is to classify points into clusters. An example of the code used to generate a bivariate data set along with a plot of the points is shown below.

```
> A <- MixSim(MaxOmega = 0.005, K = 5, p = 2, PiLow = 0.05)
> n <- 500
> PI <- A$Pi
> mu <- A$Mu
> s <- A$S
> sim <- simdataset(n, PI, mu, s)
```

```
> plot(sim$X[,1], sim$X[,2], col = sim$id, xlab = "x", ylab = "y")
```

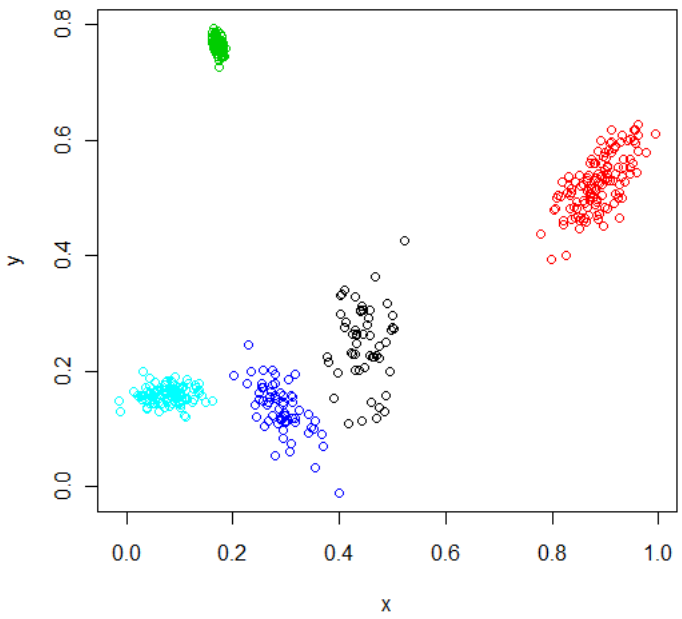

Figure 3. The graph shows 500 bivariate observations divided into five clusters with minimal overlap ($\Omega_{max} = 0.005$).

The primary focus of the simulation study was to compare the performance of each algorithm when the number of clusters, the dimensionality and overlap of the clusters change. However, the mixing proportions were left variable to emulate the uneven cluster representation that might be seen in actual data. A list of the cluster parameters used in the study is shown in table 3.

For each data set generated, each algorithm was run 10 times and the iteration with the smallest value of the cost function was chosen for each algorithm. The proportion of correctly classified observations was then calculated for each method. The distribution of this proportion is highly skewed for these algorithms, so the median will be used to minimize the impact of extreme outliers. For box plots of the algorithms on different settings, see the Appendix. Data given in the following tables

show the median values of the proportion of correctly classified observations. The K-Means column shows the median proportion with ordinary K-Means, the MK-Means1 column gives the median proportion using Mahalanobis K-Means with no covariance matrix stretching in the initialization step, and finally the MK-Means2 column gives the median proportion using Mahalanobis K-Means with the proposed stretching in the initialization step.

Figures 4-7 show the distributions of the proportion of correctly classified observations for each of the different parameter settings used in the simulation study. The boxplots are not stratified by the values of Ω_{max} as this value would be unknown in general.

The most striking result from the simulation study is how poorly the "no-stretching" version of Mahalanobis K-Means works with respect to both of the other algorithms. The vast difference between the median proportions with and without stretching provide strong evidence that stretching the cluster covariance matrices in the initialization step is crucial to the algorithm's success. This difference appears even more drastic when the data increases in dimensionality. While the ordinary K-Means algorithm and MK-Means2 appear to perform similarly on bivariate and 5-variate data, MK-Means1 suffers considerably. Taking the additional steps to ensure that cluster covariance matrices are scaled and rotated properly greatly improves the utility of the Mahalanobis distance. In the absence of this stretching step, the resulting estimates of $\mathbf{\Sigma}$ are far less accurate than using the identity matrix for $\mathbf{\Sigma}$ as ordinary K-Means does. However, it does appear that the Mahalanobis K-Means with stretching provides a substantial improvement over ordinary K-Means.

Comparing ordinary K-Means to Mahalanobis K-Means with stretching yields interesting results. First, the two algorithms perform very well with only 5 clusters present in the data set; however, there is a slight drop off in the accuracy of MK-

Means2 as cluster overlap increases, whereas K-Means only drops off in the 5 cluster case when overlap increases. Since MK-Means2 picks up initial cluster centers in areas of high point density it may be the case that in high overlap scenarios, MK-Means2 picks up initial points in regions where there is significant cluster overlap. This can cause MK-Means2 to mistakenly classify the two overlapping clusters as one cluster. Therefore, the MK-Means2 algorithm appears to work optimally on data sets where clusters are more well-separated.

The most attractive feature of MK-Means2 is that it is better equipped to handle data sets with more clusters. As the number of clusters in the data set increases, K-Means drops off substantially, whereas MK-Means2 remains very accurate. In the 5 cluster case, K-Means and MK-Means2 are both very effective at identifying clusters in data. However, if the data set contains 10 clusters, the median proportion of correctly classified observations for K-Means drops by 4.0-11.6%, with maximum cluster overlap and dimensionality held constant. In contrast, the accuracy of MK-Means2 remains virtually identical for these scenarios.

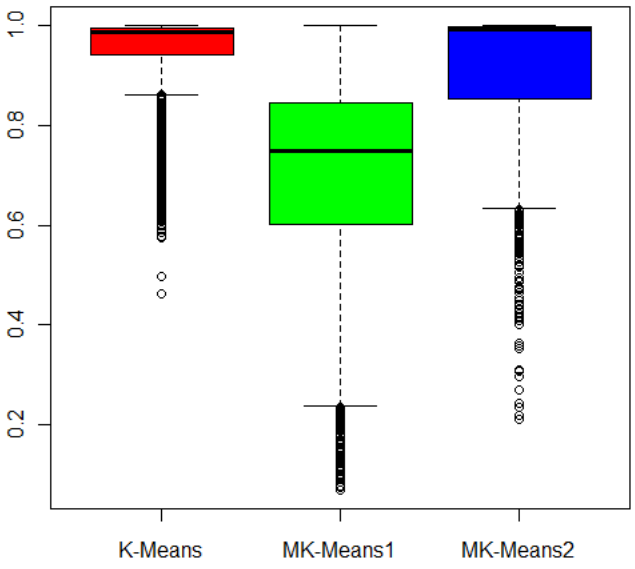

Figure 4. Box plots for K-Means, MK-Means1, and MK-Means2 for Bivariate Case with 5 Clusters.

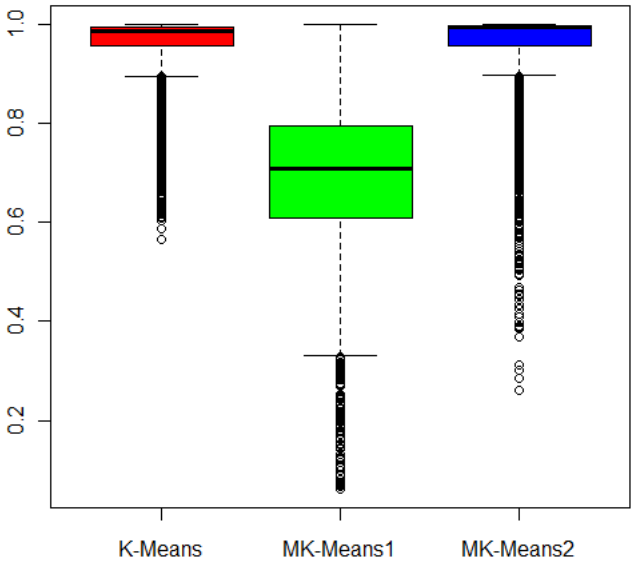

Figure 5. Box plots for K-Means, MK-Means1, and MK-Means2 for 5-variate Case with 5 Clusters.

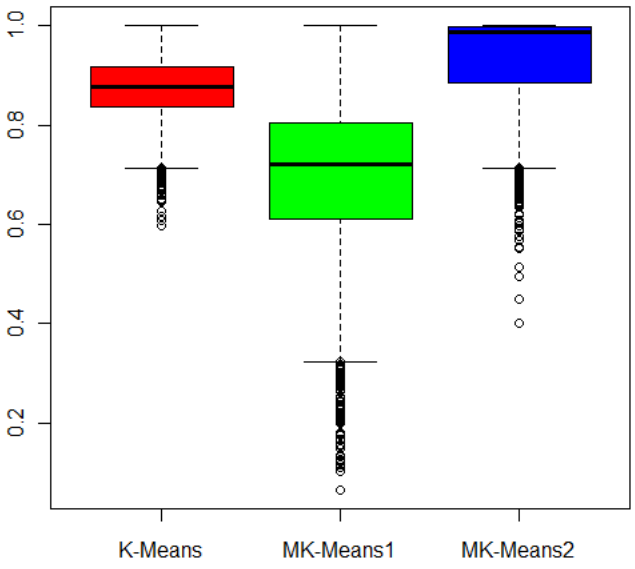

Figure 6. Box plots for K-Means, MK-Means1, and MK-Means2 for Bivariate Case with 10 Clusters.

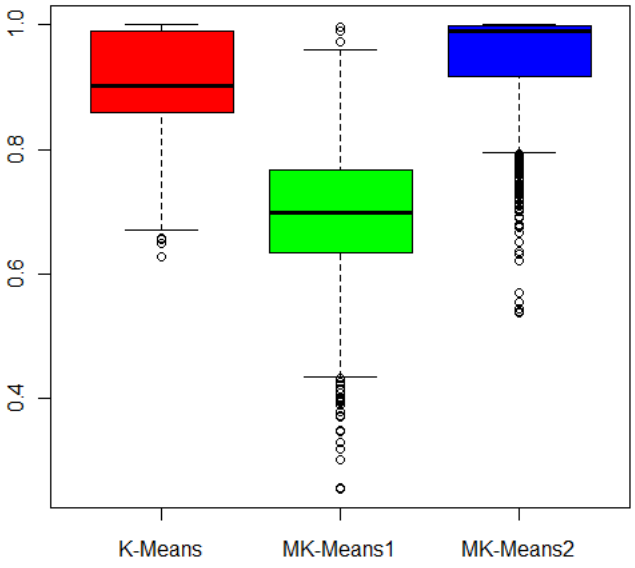

Figure 7. Box plots for K-Means, MK-Means1, and MK-Means2 for 5-variate Case with 10 Clusters.

Symbol	Definition	Settings
Ω_{max}	Maximum amount of pairwise overlap between the clusters.	0.005, 0.01, 0.05
K	Number of clusters in dataset.	5,10
p	Dimensionality of dataset.	2,5
π_{low}	Lowest proportion of observations assigned to a cluster.	0.05
N	Number of total observations.	500
nstart	Number of iterations of algorithm on each data set.	10
w	Number of nearest neighbors used to calculate initial clusters.	25

Table 1. Cluster Parameters for Simulation Study.

Ω_{max}	Number of Clusters	K-Means	MK-Means1	MK-Means2
0.005	5	0.994	0.760	0.998
0.01	5	0.990	0.763	0.996
0.05	5	0.956	0.720	0.932
0.005	10	0.878	0.732	0.996
0.01	10	0.880	0.728	0.996
0.05	10	0.876	0.711	0.943

Table 2. Results for Bivariate Data.

Ω_{max}	Number of Clusters	K-Means	MK-Means1	MK-Means2
0.005	5	0.996	0.732	0.998
0.01	5	0.992	0.720	0.996
0.05	5	0.950	0.684	0.944
0.005	10	0.896	0.708	0.998
0.01	10	0.904	0.714	0.994
0.05	10	0.910	0.673	0.928

Table 3. Results for 5-variate Data.

4. FURTHER APPLICATIONS TO REAL DATA

It has been shown empirically that the modified K-Means algorithm assigns data into clusters better than K-Means under certain conditions. In this section, we apply the modified K-Means algorithm to a familiar data set. This exercise will test the performance of the modified K-Means algorithm on real data. The modified K-Means outperforms K-Means if the clusters are multivariate normal as we've seen in the simulation study; however, it remains to be seen if the algorithm is more effective with messier data. In this section, both algorithms will be used to identify clusters in the Iris data set.

The Iris data set is a very well-known data set consisting of 150 total samples of three species of Iris: Iris setosa, Iris virginica, and Iris versicolor, with 50 samples from each species [14]. Four characteristics of the flowers were measured: petal length, petal width, sepal length, and sepal width. Iris has grown to become a benchmark for testing classification techniques by statisticians. In particular, R.A. Fisher investigated the nature of the data in his research on discriminant analysis [15]. Our goal will be to separate the observations into clusters representing the aforementioned species by using the sepal and petal information collected. Figure 8 shows a scatterplot matrix of the four independent variables in the Iris data set.

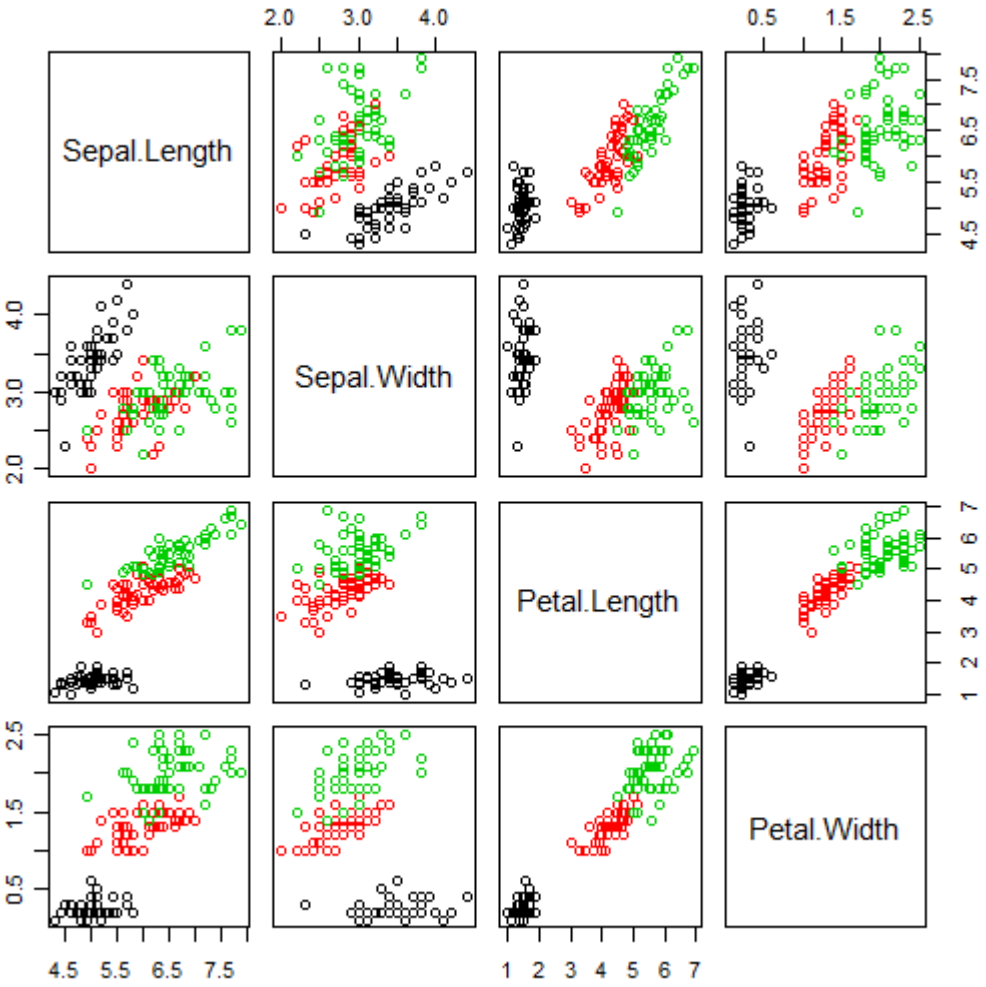

Figure 8. Iris setosa observations are shown in black, Iris versicolor in red, and Iris virginica in green.

Looking at the figure, there appears to be considerable separation between Iris setosa and the other two species; however, Iris versicolor and Iris virginica are overlapped heavily in all of the bivariate scatterplots. In lieu of a clustering algorithm, visual detection of Iris versicolor and Iris virginica as two separate clusters is nearly impossible in two dimensions. Using the three algorithms to separate the observations into clusters yields more promising results. The proportion of correctly classified

observations for each algorithm is given in table 4. This exercise illustrates the ability of the MK-Means2 algorithm to separate the Iris virginica and Iris versicolor clusters despite their close proximity simply by utilizing their respective covariance structures.

Method	Proportion Correct
K-Means	0.893
MK-Means1	0.947
MK-Means2	0.967

Table 4. Results for Iris Data Classification.

5. CONCLUSION

The K-Means algorithm remains a fixture in modeling clusters. It has the advantage of being computationally fast and accurate in cases where there are relatively few clusters. However, it lacks the capability to handle non-spherical clusters and as the number of clusters increases, K-Means is far less useful for cluster classification. Therefore, an alternative approach has been developed to handle data sets with numerous clusters. Modifying the K-Means algorithm by using Mahalanobis distances rather than Euclidean distances allows the algorithm to take clusters' variance structures into account.

Two different initialization procedures were developed in order to maximize the classification accuracy of Mahalanobis K-Means. The first initialization method proposed uses Euclidean distances to initialize spherically shaped clusters in regions with a high concentration of data points. The second initialization method is an extension of the first method and uses the covariance matrix estimate $\widehat{\mathbf{\Sigma}}$ for each cluster to generate 95% confidence ellipsoids which then capture more points. The mean vector and covariance matrix are then updated, and this process is repeated until no additional points are captured by the confidence ellipsoids. The Mahalanobis K-Means algorithms associated with these two initialization methods were named MK-Means1 and MK-Means2, respectively.

Performance of both Mahalanobis K-Means algorithms and the regular K-Means algorithm was evaluated in a simulation study. In general, the performance of MK-Means1 was very poor because the cluster covariance matrices were poorly estimated in the initialization step. As for MK-Means2 and K-Means, both performed very well in the five cluster case. However, MK-Means2 vastly outperformed K-Means in the ten cluster case with the difference being the most pronounced in data sets with lower cluster overlap. MK-Means1 and MK-Means2 both outperformed regular K-Means

in correctly classifying observations in the Iris data set, but MK-Means2 was again the best algorithm of all three.

REFERENCES

[1] A. C. Rencher, *Methods of Multivariate Analysis*. Wiley series in probability and mathematical statistics. Probability and mathematical statistics, J. Wiley, 2002.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[3] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Syst. Appl.*, vol. 36, pp. 3336–3341, Mar. 2009.

[4] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, Part 2, pp. 3336 – 3341, 2009.

[5] I. Lee and J. Yang, "2.27 - common clustering algorithms," in *Comprehensive Chemometrics* (E. in Chief:Stephen D. Brown, R. Tauler, , and B. Walczak, eds.), pp. 577 – 618, Oxford: Elsevier, 2009.

[6] S. P. Lloyd, "Least squares quantization in pcm.," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–136, 1982.

[7] H. Steinhaus, "Sur la division des corp materiels en parties," *Bull. Acad. Polon. Sci*, vol. 1, pp. 801–804, 1956.

[8] R. C. de Amorim and B. Mirkin, "Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering," *Pattern Recognition*, vol. 45, no. 3, pp. 1061 – 1075, 2012.

[9] J. Pea, J. Lozano, and P. Larraaga, "An empirical comparison of four initialization methods for the k-means algorithm," *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1027 – 1040, 1999.

[10] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[11] L. Kaufman and P. Rousseeuw, *Finding Groups in Data An Introduction to Cluster Analysis*. New York: Wiley Interscience, 1990.

[12] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, pp. 264–323, Sept. 1999.

[13] V. Melnykov, W.-C. Chen, and R. Maitra, *MixSim: Simulating Data to Study Performance of Clustering Algorithms.*, 2010. R package version 1.0-2.

[14] E. Anderson, "The species problem in iris," *Annals of the Missouri Botanical Garden*, vol. 23, pp. 457–509, 1936.

[15] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals Eugen.*, vol. 7, pp. 179–188, 1936.

APPENDIX

```
testfunc <- function(){

require(MixSim)

c1 <- matrix(ncol = 4)


## Initialization of K Clusters ##

###############################################################

A <- MixSim(MaxOmega = 0.05, K = 10, p = 2, PiLow = 0.05)

n <- 500

PI <- A$Pi

mu <- A$Mu

s <- A$S

sim <- simdataset(n, PI, mu, s)

p <- 2


## Single Iteration

#################################################################

nstart <- 10

itermat <- c(NA, NA)

indmat <- matrix(rep(NA, n), ncol = n)

ind1mat <- matrix(rep(NA, n), ncol = n)

ind2mat <- matrix(rep(NA, n), ncol = n)


## Functions

#############
```

```
## Growing Clusters
###################

cluster <- function(data, sigmahat, muhat, alpha){
n <- dim(data)[1]
p <- dim(data)[2]

data0 <- sweep(data, 2, muhat, FUN = "-")
m0 <- diag(data0 %*% solve(sigmahat, tol = 1e-50) %*% t(data0))
ind <- which(m0 < qchisq(alpha, df = p, lower.tail = F))

muhat <- colMeans(data[ind,])
sigmahat <- var(data[ind,])




outputs <- list(ind, muhat, sigmahat)
return (outputs)
}

## Cluster Initialization
#########################

clusterinit <- function(data, alpha, w){

## Create a Distance Matrix
```

```
d <- as.matrix(dist(data, method = "euclidean"))
n <- dim(data)[1] - length(which(is.na(d)[1,] == TRUE))
p <- dim(data)[2]

# Need to remove NA values from dist.list so it does not select an NA point
# use which(is.na(data[,1])).



rand1 <- rmultinom(1, 1, (n:1)^2 / sum((1:n)/2))
randn <- sum((1:n)*rand1)
dist.list <- apply(d, 1, dist.n, w)
dist.id <- order(dist.list)[randn]
r <- sort(d[dist.id,])[w]

sphcluster <- as.matrix(data[which(d[dist.id,] < r),])

muhat <- as.vector(colMeans(sphcluster))
sigmahat <- var(sphcluster)

all.est <- list()

all.est[[1]] <- list(which(d[dist.id,] < r), muhat, sigmahat)

all.est[[2]] <- cluster(data, sigmahat, muhat, alpha) # 1st iteration

iter <- 2
```

```
while(identical(all.est[[iter]][[1]], all.est[[(iter-1)]][[1]]) != TRUE){

sigmahat <- unlist(all.est[[iter]][[3]])

muhat <- unlist(all.est[[iter]][[2]])

iter <- iter + 1

est <- cluster(data, sigmahat, muhat, alpha)

all.est[[iter]] <- est



}

return(all.est)

}

## Mahalanobis K-Means
######################

mkmeans <- function(data, clusters){

b <- list()
```

```
clusters <- clusters[as.vector(unlist((lapply(1:length(clusters),
function(i) if(length(clusters[[i]]) > p)
b[[i]] <- i else b[[i]] <- NULL))))]

clusters <- clusters[as.vector(unlist((lapply(1:length(clusters),
function(i) if(det(var(data[clusters[[i]],])) >= 1e-50)
b[[i]] <- i else b[[i]] <- NULL))))]

k <- length(clusters)

distmatrix <- matrix(unlist(lapply(1:k,
function(j) mahalanobis(data, apply(data[clusters[[j]],], 2, mean),
solve(var(data[clusters[[j]],]), tol = 1e-50), inverted = TRUE))),
ncol = k)

ind <- apply(distmatrix, 1, which.min)

clusters <- lapply(1:k, function(i) which(ind == i))

clusters <- lapply(1:k, function(i) unique(clusters[[i]]))

clusters <- clusters[as.vector(unlist((lapply(1:length(clusters),
function(i) if(length(clusters[[i]]) > p) b[[i]] <- i
else b[[i]] <- NULL))))]

return(clusters)
```

```
}

## Regular K-Means Initialization
#################################

kinit <- function(data, k){
centers <- sample(1:dim(data)[1], k)
d <- as.matrix(dist(data, method = "euclidean"))
dcenters <- d[,centers]
ind <- apply(dcenters, 1, which.min)
clusters <- lapply(1:k, function(i) which(ind == i))
clusters <- lapply(1:k, function(i) unique(clusters[[i]]))

return(clusters)
}

## Within Sums of Squares Functions: Mahalanobis and Euclidean
#############################################################

SSm <- function(data, cluster){
mu0 <- apply(data[cluster,], 2, mean)
data0 <- sweep(data[cluster,], 2, mu0, FUN = "-")
s <- var(data[cluster,])
d <- sqrt(diag(data0 %*% solve(s, tol = 1e-50) %*% t(data0)))
ss <- sum(d)
```

```
return(ss)
}

SS <- function(data, cluster){
mu0 <- apply(data[cluster,], 2, mean)
data0 <- sweep(data[cluster,], 2, mu0, FUN = "-")
d <- sqrt(diag(data0 %*% t(data0)))
ss <- sum(d)

return(ss)
}

rowreplace <- function(x, list, values){
x[list,] <- values
x

}

for(m in 1:nstart){

clusters <- list()
i <- 2
n <- dim(sim$X)[1]
k <- 10
alpha <- 0.05
w <- 25
```

```
data <- sim$X
d <- as.matrix(dist(data, method = "euclidean"))

cl <- clusterinit(data, alpha, w)
clusters[[1]] <- unlist(cl[[length(cl)]][[1]])
data <- rowreplace(data, clusters[[1]], NA)



while(i <= k && length(which(is.na(data[,1]) == F)) > w){

cl <- clusterinit(data, alpha, w)
clusters[[i]] <- cl[[length(cl)]][[1]]
data <- rowreplace(data, clusters[[i]], NA)
i <- i + 1



}







mcl <- list()

mcl[[1]] <- clusters
iter <- 2
```

```
repeat{
mcl[[iter]] <- mkmeans(sim$X, mcl[[(iter-1)]])

if(identical(mcl[[iter]], mcl[[(iter-1)]]) == FALSE){

iter <- iter + 1
}

else {
break
}

}



ind <- rep(NA, dim(sim$X)[1])
a <- mcl[[length(mcl)]]



for(i in 1:length(a)){

for(j in 1:length(a[[i]])){

ind[a[[i]][[j]]] <- i
}
}
```

```
## Alternate Initialization
############################################################
clusters <- kinit(sim$X, k)

mcl1 <- list()

mcl1[[1]] <- clusters
iter <- 2

repeat{
mcl1[[iter]] <- mkmeans(sim$X, mcl1[[(iter-1)]])

if(identical(mcl1[[iter]], mcl1[[(iter-1)]]) == FALSE){

iter <- iter + 1
}

else {
break
}

}
```

```
ind1 <- rep(NA, dim(sim$X)[1])
a1 <- mcl1[[length(mcl1)]]


for(i in 1:length(a1)){

for(j in 1:length(a1[[i]])){

ind1[a1[[i]][[j]]] <- i
}
}

## Regular K-Means
###################

kmeans <- kmeans(sim$X, k)

## Results
###############
SS1 <- 0
for(i in 1:length(a)){
SSt <- SS(sim$X, a[[i]])
SS1 <- SS1 + SSt
}
```

```
WSS1 <- 0
for(i in 1:length(a)){
SSc <- SSm(sim$X, a[[i]])
WSS1 <- WSS1 + SSc
}

WSS2 <- 0
for(i in 1:length(a1)){
SSc1 <- SSm(sim$X, a1[[i]])
WSS2 <- WSS2 + SSc1
}

WSS3 <- kmeans$tot.withinss

if(m == 1)
indmat[m,] <- ind
ind1mat[m,] <- ind1
ind2mat[m,] <- kmeans$cluster



if(m > 1)
indmat <- rbind(indmat, ind)
ind1mat <- rbind(ind1mat, ind1)
ind2mat <- rbind(ind2mat, kmeans$cluster)

itermat <- rbind(itermat, c(SS1, WSS1, WSS2, WSS3))
```

```
}
itermat <- itermat[-which(is.na(itermat[,1]) == TRUE),]

a10 <- ClassProp(indmat[which.min(itermat[,1]),], sim$id)
## MK-Means1
#a11 <- ClassProp(indmat[which.min(itermat[,2]),], sim$id)
## My Algorithm using Mahalanobis WSS to determine best clusters
a12 <- ClassProp(ind1mat[which.min(itermat[,3]),], sim$id)
## MK-Means2
a13 <- ClassProp(ind2mat[which.min(itermat[,4]),], sim$id)
## K-Means



c1 <- rbind(c1, c(a10, NA, a12, a13))

c1 <- c1[-which(is.na(c1[,1]) == TRUE),]

return(c1)

}
```

Algorithm comparison code for $\Omega_{max} = 0.05, K = 10, p = 2$. This code will produce 10 runs of each algorithm for each data set, with the best cluster configuration chosen to represent the optimal assignments for each algorithm.