University of Southern Denmark
IMADA
Ricardo Campello

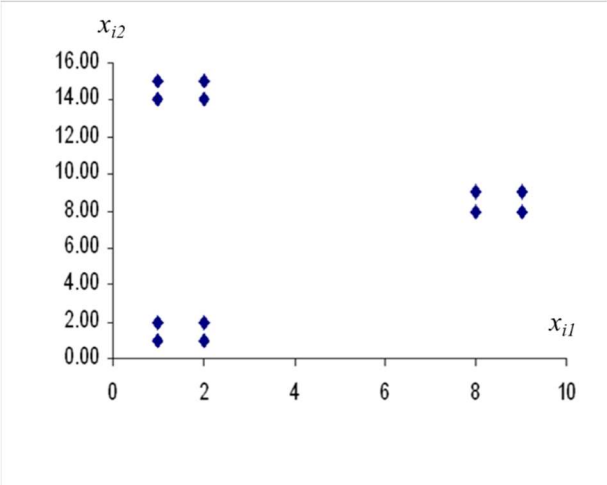
DM583: Data Mining

Exercise 3: Partitioning Clustering and Evaluation (k-Means, SSE and Silhouette)

Exercise 3-1 Standard k-Means

a) Perform k-means in the above dataset starting from prototypes [6 6], [4 6] and [5 10]. You can use R as a calculator to assist e.g. during distance and centroid computations, but otherwise you are asked to do it "manually" in a detailed and step-by-step fashion. Use standard *Euclidean* distance (not squared).

Observation $\mathbf{x}_i$	x_{i1}	x_{i2}
1	1	2
2	2	1
3	1	1
4	2	2
5	8	9
6	9	8
7	9	9
8	8	8
9	1	15
10	2	15
11	1	14
12	2	14



Proposed Solution:

We start by initialising our three prototypes as $\mathbf{v}_1$ = [6 6] (prototype of Cluster 1), $\mathbf{v}_2$ = [4 6] (prototype of Cluster 2), and $\mathbf{v}_3$ = [5 10] (prototype of Cluster 3). Then we compute the Euclidean distance between each data object and each prototype. The detailed steps are omitted here for the sake of compactness, but the result should be as follows (rows stand for data objects and the 1st, 2nd and 3rd columns stand for the prototypes of Cluster 1, 2, and 3, respectively):

```
     centre1  centre2  centre3
1   6.403124 5.000000 8.944272
2   6.403124 5.385165 9.486833
3   7.071068 5.830952 9.848858
4   5.656854 4.472136 8.544004
5   3.605551 5.000000 3.162278
6   3.605551 5.385165 4.472136
7   4.242641 5.830952 4.123106
8   2.828427 4.472136 3.605551
9  10.295630 9.486833 6.403124
10  9.848858 9.219544 5.830952
11  9.433981 8.544004 5.656854
12  8.944272 8.246211 5.000000
```

Data objects {1, 2, 3, 4} are closer to (the prototype of) Cluster 2, whereas objects {5, 7, 9, 10, 11, 12} are closer to Cluster 3, and objects {6, 8} are closer to Cluster 1, so we make these assignments creating the first partition of the data.

We then update the cluster prototypes as their centroids (multivariate means):

```
              x1       x2
centre1 8.500000  8.00000
centre2 1.500000  1.50000
centre3 3.833333 12.66667
```

Now we recompute the distances between all objects and centroids:

```
     centre1    centre2   centre3
1   9.604686  0.7071068 11.036555
2   9.552487  0.7071068 11.809836
3  10.259142  0.7071068 12.005786
4   8.845903  0.7071068 10.823072
5   1.118034  9.9247166  5.550275
6   0.500000  9.9247166  6.962200
7   1.118034 10.6066017  6.335526
8   0.500000  9.1923882  6.256108
9  10.259142 13.5092561  3.670453
10  9.552487 13.5092561  2.967416
11  9.604686 12.5099960  3.131382
12  8.845903 12.5099960  2.266912
```

Data objects {1, 2, 3, 4} are closer to (the prototype of) Cluster 2, whereas objects {9, 10, 11, 12} are closer to Cluster 3, and objects {5, 6, 7, 8} are closer to Cluster 1, so we make these assignments creating the second partition of the data.

We then update the cluster prototypes as their centroids (multivariate means):

```
         x1   x2
centre1 8.5  8.5
centre2 1.5  1.5
centre3 1.5 14.5
```

Now we recompute the distances between all objects and centroids:

```
      centre1    centre2    centre3
1   9.9247166  0.7071068 12.5099960
2   9.9247166  0.7071068 13.5092561
3  10.6066017  0.7071068 13.5092561
4   9.1923882  0.7071068 12.5099960
5   0.7071068  9.9247166  8.5146932
6   0.7071068  9.9247166  9.9247166
7   0.7071068 10.6066017  9.3005376
8   0.7071068  9.1923882  9.1923882
9   9.9247166 13.5092561  0.7071068
10  9.1923882 13.5092561  0.7071068
11  9.3005376 12.5099960  0.7071068
12  8.5146932 12.5099960  0.7071068
```

Clearly, the closest centroid to each observation remains unchanged, which means the algorithm has converged. The final solution (partition) is {1, 2, 3, 4}, {9, 10, 11, 12}, {5, 6, 7, 8}, which correspond to the visually natural clusters in the figure.

b) Would you expect the clustering result (partition) to change if you were to use *squared Euclidean* distance instead? Discuss it conceptually, justifying your answer on a theoretical basis.

Proposed Solution:

Since squared Euclidean is just a monotone transformation of the standard (non-squared) Euclidean distance, both measures will produce the same assignment of observations to clusters based on the closest centroid, which means that the final clustering result will not change, i.e., it will necessarily be the same.

Important Note/Disclaimer: *The standard k-Means algorithm is derived as the locally optimal solution to an optimization problem defined based on the squared Euclidean distance. It can be shown that this does not necessarily correspond to the optimal solution to an equivalent problem that could be defined instead in terms of the standard, non-squared Euclidean distance. This means that running the k-Means algorithm with standard Euclidean distance is still in practice only guaranteed to be solving the original optimization problem, which is defined in terms of squared Euclidean distance.*

c) Repeat item (a), now using function `kmeans()` from package `stats` in R with argument `centers` to explicitly provide the specified initial prototypes as input.

Proposed Solution:

```
# Read Dataset:
toy_data <- read.csv("Kmeans_Exercise_Toy_Dataset.csv")
# Define Data Frame with the 3 Initial Cluster Prototypes:
centre1 <- data.frame(x1 = 6, x2 = 6)
centre2 <- data.frame(x1 = 4, x2 = 6)
centre3 <- data.frame(x1 = 5, x2 = 10)
centres <- rbind(centre1,centre2,centre3)
rownames(centres) <- c("centre1","centre2","centre3")
# Run k-Means from the Specified Initial Prototypes:
cl <- kmeans(toy_data, centers = centres)
# These are the Resulting Cluster Centers:
cl$centers
# These are the Resulting Cluster Labels (Partition):
cl$cluster
```

Exercise 3-2 k-Means with Recursive Centroid Updates

Suppose that at a certain iteration during execution of k-Means, the cluster prototypes of all k clusters have just been updated. Specifically for the ith cluster (C_i), its prototype has just been updated as the centroid ($\mathbf{v}_i$) of the $|C_i|$ observations currently belonging to that cluster ($|C_i|$ denotes the ith cluster's size or cardinality). Now, in the next step of the algorithm, the distances between every observation in the dataset and such updated centroids will be recomputed. Let's suppose that observations $\mathbf{x}_j$, $\mathbf{x}_l$, and $\mathbf{x}_m$ will no longer be closer to $\mathbf{v}_i$, they will become closer to the updated centroid of another cluster, whereas observations $\mathbf{x}_p$ and $\mathbf{x}_q$, which are currently part of other clusters, will have $\mathbf{v}_i$ as their closest centroid. Assuming that these are the only 5 observations to be moved from/to the ith cluster in the next step, derive an equation to update its centroid $\mathbf{v}_i$ from its current value, using only these 5 observations as well as the cluster size $|C_i|$ prior to such an update.

Proposed Solution:

The current centroid is the multivariate mean of the observations in the current cluster C_i, i.e.: $\mathbf{v}_i = (\Sigma_{\mathbf{x} \in Ci}\ \mathbf{x})/|C_i|$. The new centroid in the next step will add to the mean computation the 2 observations joining the cluster, while removing the 3 observations leaving the cluster, i.e., $\mathbf{v}_i^{new} = (\Sigma_{\mathbf{x} \in Ci}\ \mathbf{x} + \mathbf{x}_p + \mathbf{x}_q - \mathbf{x}_j - \mathbf{x}_l - \mathbf{x}_m) / (|C_i| + 2 - 3)$, which can be readily rewritten recursively as $\mathbf{v}_i^{new} = (|C_i| \cdot \mathbf{v}_i + \mathbf{x}_p + \mathbf{x}_q - \mathbf{x}_j - \mathbf{x}_l - \mathbf{x}_m) / (|C_i| + 2 - 3)$.

Exercise 3-3 Parallel / Distributed k-Means

a) Suppose that we have a data collection $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ with N observations distributed across s local processing/storage sites with independent (separated, non-shared) memory, each of which holding exclusive access to (and only to) a subset of the data. For instance, assuming $s = 3$ sites and a dataset with $N = 9$ observations, Site 1 could have access only to observations $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$, Site 2 could have access only to observations $\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7\}$, and Site 3 could have access only to observations $\{\mathbf{x}_8, \mathbf{x}_9\}$. We want to perform exact k-Means (for a given choice of k) on the whole dataset $\mathbf{X}$ and obtain the same result (no approximations) as if the algorithm had simultaneous and unrestricted access to any data observation, which is not the case. For simplicity, we assume that we have a central processing node that can communicate *summary cluster statistics* to and from the local processing/storage sites. However, for privacy and/or data transmission bandwidth constraints, data observations are NOT allowed to be transmitted across the network (neither between two local sites nor between a local site and the central processing node). Describe an algorithm that solves the problem.

Proposed Solution:

We initialize the k cluster prototypes $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$ in the central node. Then, we distribute all those same k prototypes to all s local sites. Locally in the jth site, we can compute the distances between each observation in the subset stored in that site and the cluster prototypes, so we can locally partition the subset of observations into the k clusters in question (noting that the assignment of each observation to the cluster corresponding to its closest prototype depends only on the observation itself and the prototypes, it does not depend on other observations). This is performed within every site independently. Each site then transmits to the central node the following two local summary statistics about each of the k clusters: the local cardinality (size) and the local multivariate sum of the observations within the cluster. Let N_{ij} be the number of observations assigned to the ith cluster in the jth site and let $\mathbf{a}_{ij}$ be the multivariate sum of those observations. In the central node, we can then compute the exact updated prototype (centroid) of the ith cluster as $\mathbf{v}_i = (\mathbf{a}_{i1} + \ldots + \mathbf{a}_{is}) / (N_{i1} + \ldots + N_{is})$. Once all k prototypes are updated in the same way, they are redistributed to the local sites for another iteration of the algorithm, and the process above is then just repeated for multiple iterations of k-Means.

b) Run one complete iteration of the algorithm described in the previous item (with standard Euclidean distance, $k = 3$, and initial prototypes given by $\mathbf{v}_1 = [6\ \ 6]$, $\mathbf{v}_2 = [4\ \ 6]$ and $\mathbf{v}_3 = [5\ \ 10]$) on the following dataset, where it is assumed that the observations are distributed across 2 separate processing/storage sites (A and B):

Object $\mathbf{x}_i$	x_{i1}	x_{i2}	Processor/Site
1	1	2	A
2	2	1	B
3	1	1	A
4	2	2	B
5	8	9	A
6	9	8	B
7	9	9	B
8	8	8	A
9	1	15	B
10	2	15	A
11	1	14	B
12	2	14	A

Proposed Solution:

The initial prototypes are distributed to both sites, A and B, and within each site the corresponding distances between the locally stored observations and each prototype are computed locally. The distances are as follows:

```
     centre1  centre2  centre3 site
1   6.403124 5.000000 8.944272    A
```

2	6.403124	5.385165	9.486833	B
3	7.071068	5.830952	9.848858	A
4	5.656854	4.472136	8.544004	B
5	3.605551	5.000000	3.162278	A
6	3.605551	5.385165	4.472136	B
7	4.242641	5.830952	4.123106	B
8	2.828427	4.472136	3.605551	A
9	10.295630	9.486833	6.403124	B
10	9.848858	9.219544	5.830952	A
11	9.433981	8.544004	5.656854	B
12	8.944272	8.246211	5.000000	A

In site A, observations {1, 3} are closer to (the prototype of) Cluster 2, whereas observations {5, 10, 12} are closer to Cluster 3, and observation {8} is closer to Cluster 1, so we make these assignments creating the first partition of the data locally within Site A.

In site B, observations {2, 4} are closer to (the prototype of) Cluster 2, whereas observations {7, 9, 11} are closer to Cluster 3, and observation {6} is closer to Cluster 1, so we make these assignments creating the first partition of the data locally within Site B.

We then compute the summary statistics for each of the three clusters within each site.

In Site A, we have $N_{1A} = 1$, $N_{2A} = 2$, $N_{3A} = 3$, $\mathbf{a}_{1A} = \mathbf{x}_8 = [8\ \ 8]$, $\mathbf{a}_{2A} = \mathbf{x}_1 + \mathbf{x}_3 = [2\ \ 3]$, $\mathbf{a}_{3A} = \mathbf{x}_5 + \mathbf{x}_{10} + \mathbf{x}_{12} = [12$ 38].

In Site B, we have $N_{1B} = 1$, $N_{2B} = 2$, $N_{3B} = 3$, $\mathbf{a}_{1B} = \mathbf{x}_6 = [9\ \ 8]$, $\mathbf{a}_{2B} = \mathbf{x}_2 + \mathbf{x}_4 = [4\ \ 3]$, $\mathbf{a}_{3B} = \mathbf{x}_7 + \mathbf{x}_9 + \mathbf{x}_{11} = [11$ 38].

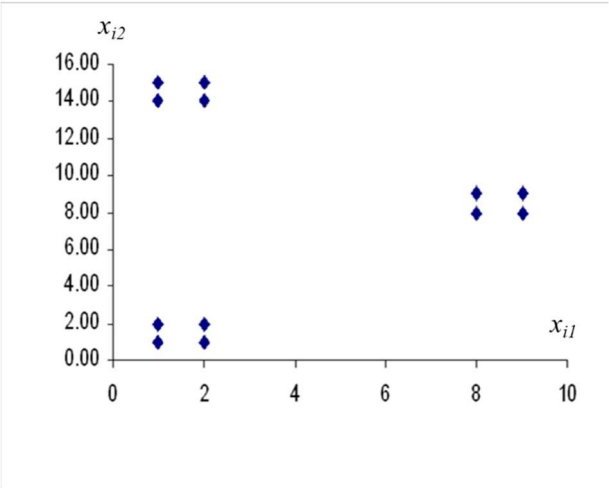
These summary statistics are then received and aggregated in the central node, which updates the cluster prototypes as:

$\mathbf{v}_1 = (\mathbf{a}_{1A} + \mathbf{a}_{1B}) / (N_{1A} + N_{1B}) = ([8\ \ 8] + [9\ \ 8]) / (1 + 1) = [17\ \ 16]/2 = [8.5\ \ 8]$,
$\mathbf{v}_2 = (\mathbf{a}_{2A} + \mathbf{a}_{2B}) / (N_{2A} + N_{2B}) = ([2\ \ 3] + [4\ \ 3]) / (2 + 2) = [6\ \ 6]/4 = [1.5\ \ 1.5]$,
$\mathbf{v}_3 = (\mathbf{a}_{3A} + \mathbf{a}_{3B}) / (N_{3A} + N_{3B}) = ([11\ \ 38] + [12\ \ 38]) / (3 + 3) = [23\ \ 76]/6 = [3.833\ \ 12.666]$.

These updates prototypes are then redistributed to the local sites and the procedure is then iterated until convergence.

Exercise 3-4 Choice of *k* and Initial Prototypes in k-Means with SSE Evaluation

Consider the same toy dataset used in other exercises:

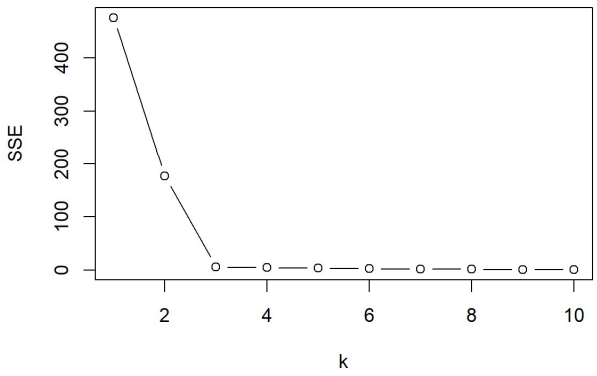


Use the built-in function `kmeans()` from package `stats` in R with argument `centers` set as an integer *k* to provide the specified number of clusters as input. In this setting, by default, the *k* initial cluster prototypes will be selected randomly by the algorithm. In order to minimize the chances that the algorithm gets stuck at local minima due to bad initial prototypes (unlikely in this simple toy example, but very likely in real datasets), you can use argument `nstart` to tell function `kmeans()` to run the algorithm multiple (`nstart`) times, each time from a different random initialization of prototypes, and choose the best clustering result according to the Sum of Squared Errors (SSE) criterion, which is minimized by k-Means for a fixed value of *k*. You can then run the algorithm for k varying within a given range (say, *k* = 1, 2, 3, …, 10) and plot the resulting SSE values as a function of *k*. Does the plot suggest the natural number of clusters in the dataset? Discuss.

Note: You can repeat this exercise for any real dataset of your choice (e.g., `Iris`, etc.) **[OPTIONAL].**

Proposed Solution:

```
# Read dataset:
toy_data <- read.csv("Kmeans_Exercise_Toy_Dataset.csv")
SSE <- rep(0, 10) # Vector of length 10 initialized with zeros
# Loop through k = 1, 2, 3, …, 10:
for (k in 1:10){
   # Run k-means with k clusters randomly initialized 10 times:
   km.out <- kmeans(x = toy_data, centers = k, nstart = 10)
   SSE[k] <- km.out$tot.withinss # Store best resulting SSE for each k
}
# Plot SSE as a function of k:
plot(1:10, SSE, xlab="k", ylab="SSE", type = "b")
```

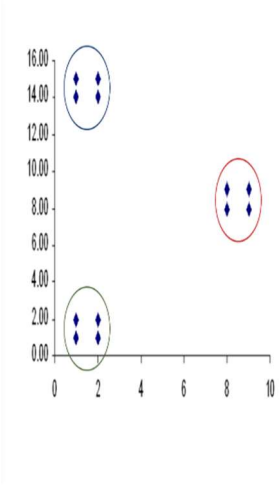


The plot shows a distinct "knee" (or "elbow") at *k* = 3, which is the natural number of clusters in the dataset. In this case, since clusters are quite compact and well-separated, this knee/elbow is quite noticeable, as expected from visualization of the dataset in the figure.

Exercise 3-5 Silhouette Width Criterion (SWC)

a) Consider again the same toy dataset used in other exercises, and the natural clustering solution as indicated by circles in the figure below. Compute the *individual contribution* of the bottom right observation in the blue (top left) cluster – i.e., the 12th observation ([2 14]) – to both the original Silhouette Width Criterion (SWC) as well as to its simplified version, the Simplified SWC (SSWC), using Euclidean distance. You can use R as a calculator to assist e.g. during distance and centroid computations, but otherwise you are asked to do it "manually" in a detailed and step-by-step fashion.

Observation $\mathbf{x}_i$	x_{i1}	x_{i2}
1	1	2
2	2	1
3	1	1
4	2	2
5	8	9
6	9	8
7	9	9
8	8	8
9	1	15
10	2	15
11	1	14
12	2	14



Proposed Solution:

We start by computing the centroids of the three clusters in the candidate solution, which are visually trivial from the data as $\mathbf{v}_1$ = [8.5 8.5] (red cluster), $\mathbf{v}_2$ = [1.5 1.5] (green cluster), and $\mathbf{v}_3$ = [1.5 14.5] (blue cluster). Then, we compute the Euclidean distance from the particular data object of interest (12th observation) to the other observations (for SWC computation) as well as to the cluster centroids (for SSWC computation). The detailed steps are omitted here for the sake of compactness, but the result should be as follows:

```
        1         2         3         4         5
12.041595 13.000000 13.038405 12.000000  7.810250
        6         7         8         9        10
 9.219544  8.602325  8.485281  1.414214  1.000000
       11        12
 1.000000  0.000000

  centre1    centre2    centre3
 8.5146932 12.5099960  0.7071068
```

Starting with the original SWC. The average distance of the observation of interest (12th) to the other observations in the same (blue) cluster, i.e., observations 9, 10 and 11, is a_{12} = (1.414214 + 1.0 + 1.0) / 3 = 1.138071. The average distance from the 12th observation to the observations in the green cluster (obs. 1 to 4) is (12.041595+13.0+13.038405+12.0)/4 = 12.52, which is larger than the average distance to the observations in the red cluster (obs. 5 to 8), namely, (7.810250+9.219544+8.602325+8.485281)/4 = 8.52935, so we can take the average distance to the closest neighboring cluster (red) as the smallest of those, i.e., b_{12} = 8.52935. From these, we can compute the original silhouette of the 12th observation as $s_{12} = (b_{12} - a_{12}) / \max\{b_{12}, a_{12}\}$ = 0.86657.

Now for the Simplified Silhouette (SSWC). The average distance of the observation of interest (12th) to the centroid of its (blue) cluster is a_{12} = 0.7071068. The average distance from the 12th observation to the centroid of the green cluster is 12.5099960, which is larger than the average distance to the centroid of the red cluster (8.5146932), so we can take the distance to the closest neighboring cluster (red) as the smallest of those, i.e., b_{12} = 8. 5146932. From these, we can compute the simplified silhouette of the 12th observation as $s_{12} = (b_{12} - a_{12}) / \max\{b_{12}, a_{12}\}$ = 0.9169545.

b) If you would write a code in R that repeats the computations in item (a) for all observations and averages the results, you would be able to compute the SWC and SSWC values for the candidate clustering solution in the figure. While writing your own code from scratch is left as optional, in this item you are asked to freely use R (packages, functions, programming) to compute (at least) the original SWC criterion for this solution.

Proposed Solution:

```
SWC <- function(clusterLabels, dataPoints){
  require(cluster)
  sil <- silhouette(x = clusterLabels, dist = dist(dataPoints))
  return(mean(sil[,3]))
}

# Read dataset:
toy_data <- read.csv("Kmeans_Exercise_Toy_Dataset.csv")

clus_labels = c(1,1,1,1,2,2,2,2,3,3,3,3) # Candidate Solution
SWC(clusterLabels = clus_labels, dataPoints = toy_data)
```

The result is **0.8793842**

c) Now, merge the two clusters on the left (blue and green circles) into a single cluster and repeat item (b). Compare with the results in item (b). What conclusions can you draw about the quality of the two candidate clustering solutions assessed according to SWC?

Proposed Solution:

```
clus_labels = c(1,1,1,1,2,2,2,2,1,1,1,1) # Candidate Solution
SWC(clusterLabels = clus_labels, dataPoints = toy_data)
```

The result is **0.4075317**, which represents a drastic reduction. This strongly suggests that merging the two clusters in question is not suitable according to the SWC criterion, and the initial solution (without the merger) is much better.

d) Repeat Exercise 3.4, now replacing SSE with SWC when choosing the number of clusters, noticing that: (i) you will no longer look for a knee/elbow but rather for a maximum peak at the plot of the SWC as a function of k; (ii) SWC is not defined for $k = 1$, so you need to start from $k = 2$ instead.

Note: You can repeat this exercise for any real dataset of your choice (e.g., `Iris`, etc.) **[OPTIONAL].**

Hint: You can use the following given function to compute SWC for a given dataset `dataPoints` and its candidate clustering solution `clusterLabels`, which is based on the built-in `silhouette()` function from package `cluster` (it requires this R package to be installed):

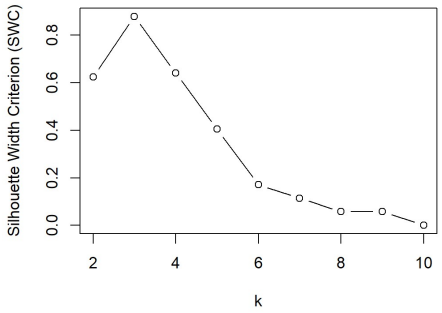
```
SWC <- function(clusterLabels, dataPoints){
   require(cluster)
   sil <- silhouette(x = clusterLabels, dist = dist(dataPoints))
return(mean(sil[,3]))
}
```

Proposed Solution:

```
SWC <- function(clusterLabels, dataPoints){
  require(cluster)
  sil <- silhouette(x = clusterLabels, dist = dist(dataPoints))
  return(mean(sil[,3]))
}

# Read dataset:
toy_data <- read.csv("Kmeans_Exercise_Toy_Dataset.csv")

Silhouette <- rep(0, 10)
for (k in 2:10){
  km.out <- kmeans(x = toy_data, centers = k, nstart = 20)
  Silhouette[k] <- SWC(clusterLabels = km.out$cluster, dataPoints = toy_data)
}
plot(2:10, Silhouette[2:10], xlab="k", ylab="Silhouette Width Criterion (SWC)", type = "b")
```



Clearly, the plot suggests that the optimal number of clusters for k-Means is $k = 3$, according to the SWC criterion, which matches our visual intuition.