Stats 102B - Clustering with Kernel Functions

Miles Chen, PhD

Department of Statistics

Week 8 Wednesday

UCLA

Section 1

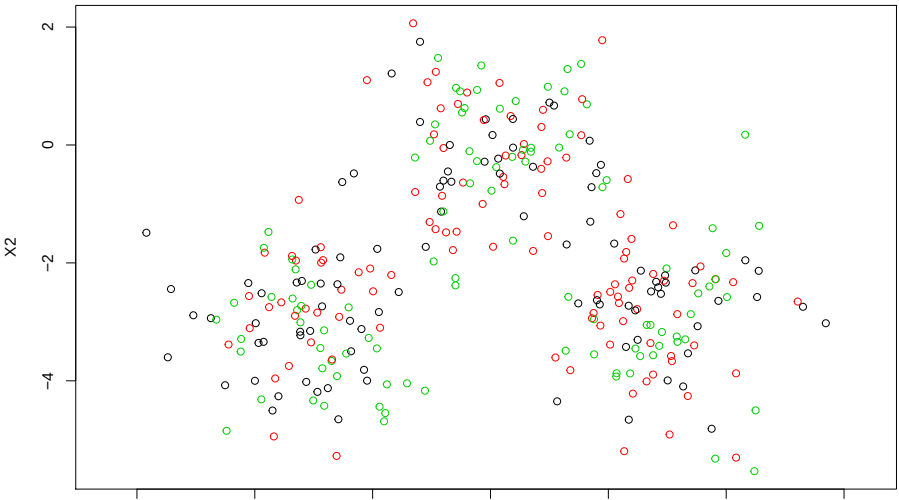
K-means Clustering with Kernel Functions

K-Means Clustering Algorithm

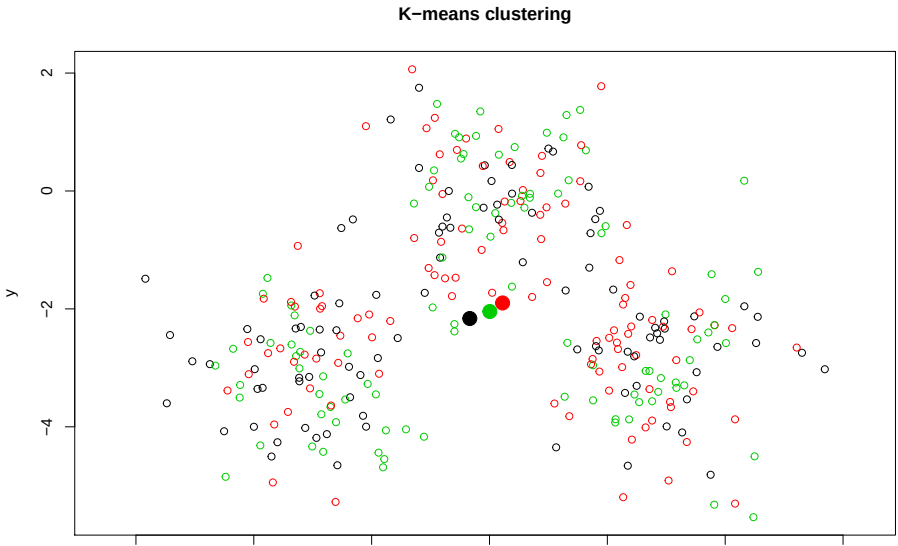
The algorithm can be described as follows:

0. Determine how many (k) clusters you will search for.
1. Randomly assign points in your data to each of the clusters.
2. Once all values have been assigned to a cluster, calculate the centroid of the values in each cluster.
3. Reassign values to clusters by associating values in the data set to the nearest (euclidean distance) centroid.
4. Repeat steps 2 and 3 until convergence. Convergence occurs when no values are reassigned to a new cluster.

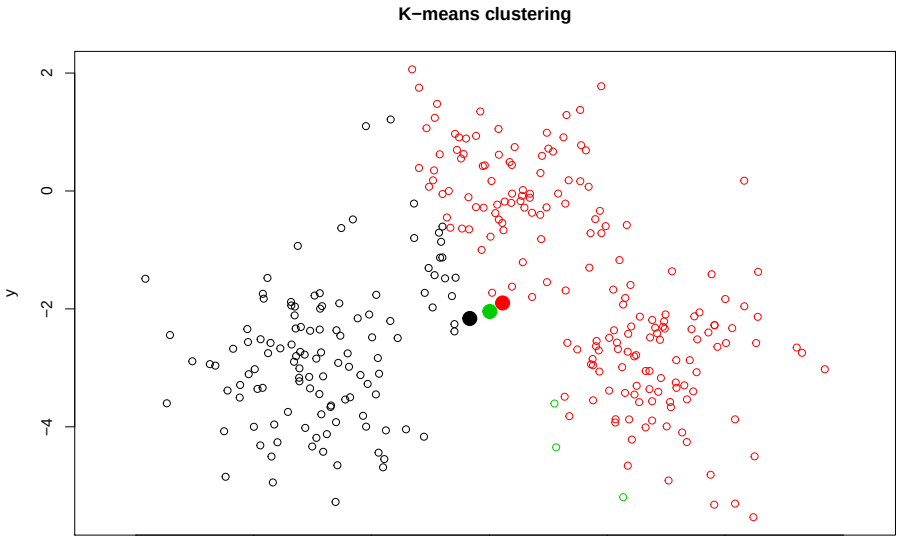
Begin with Random assignments



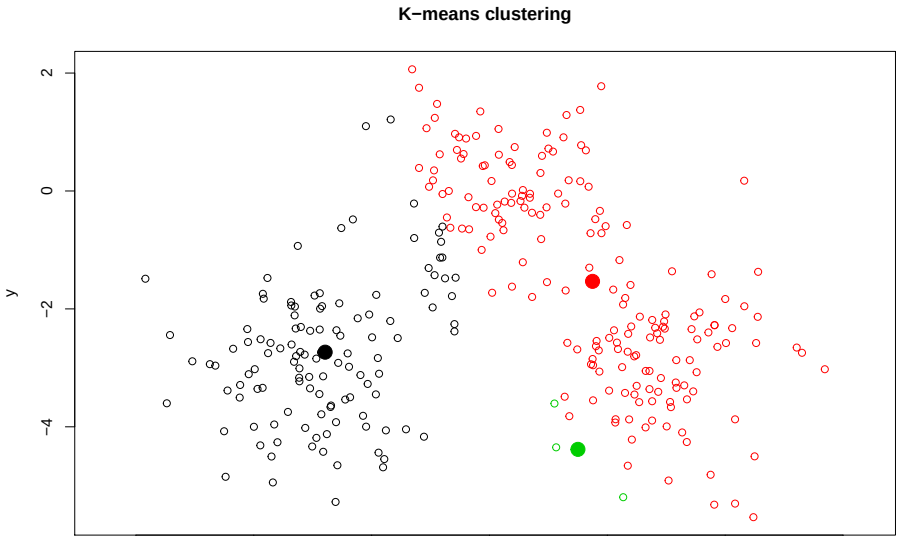
Calculate Centroids



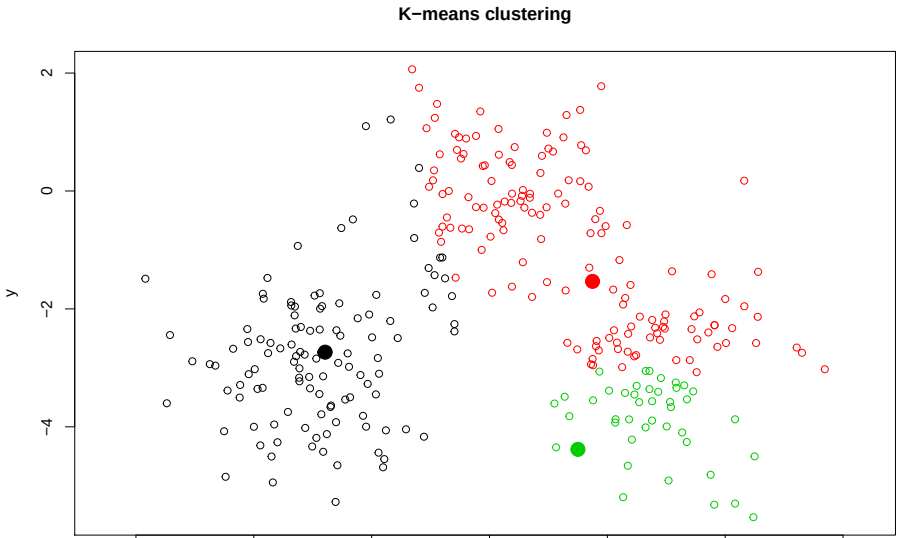
Reassign Points



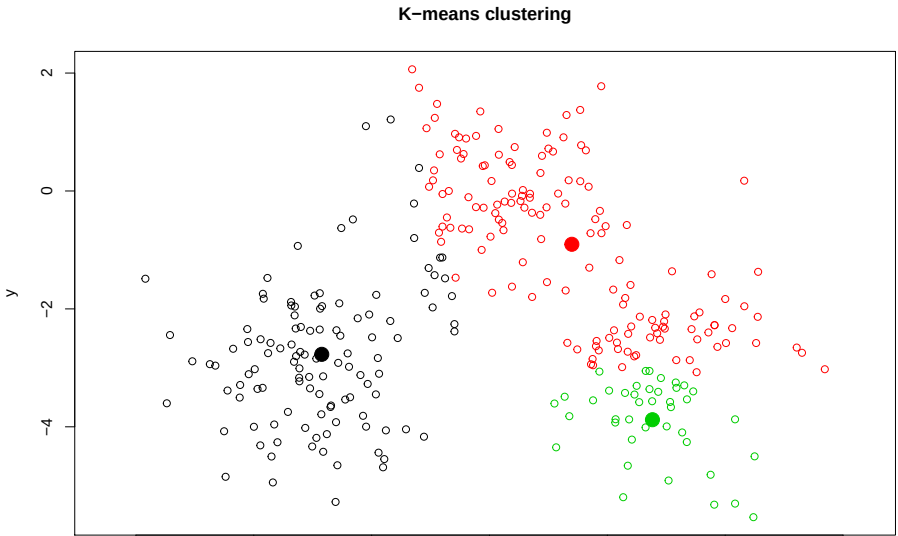
Recalculate Centroids



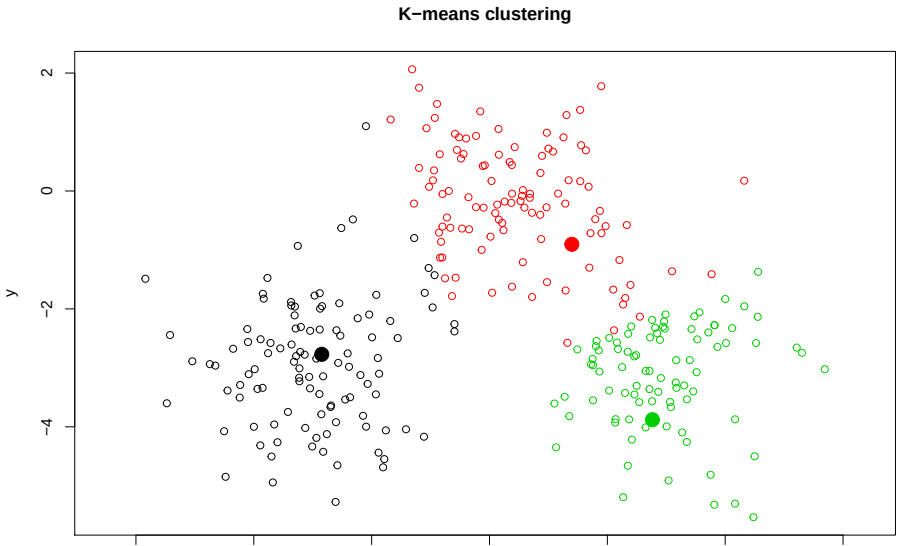
Reassign Points



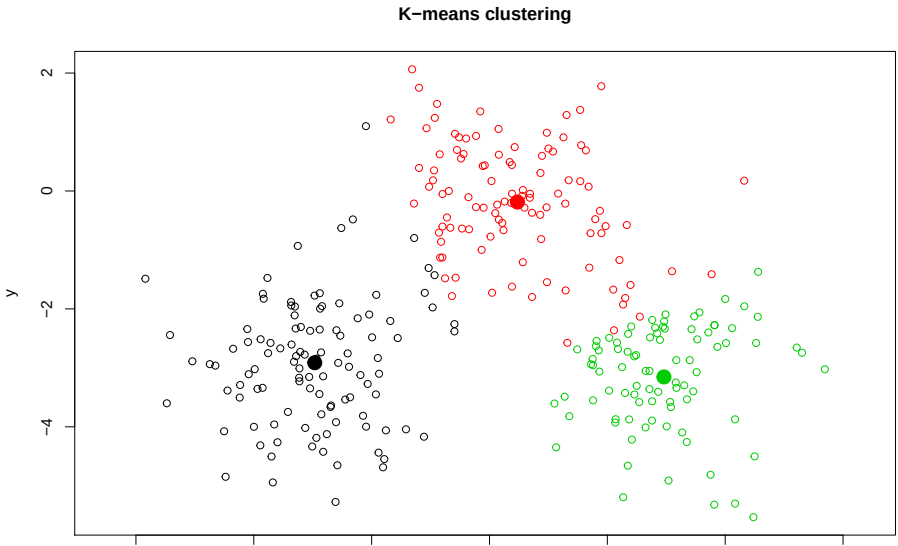
Recalculate Centroids



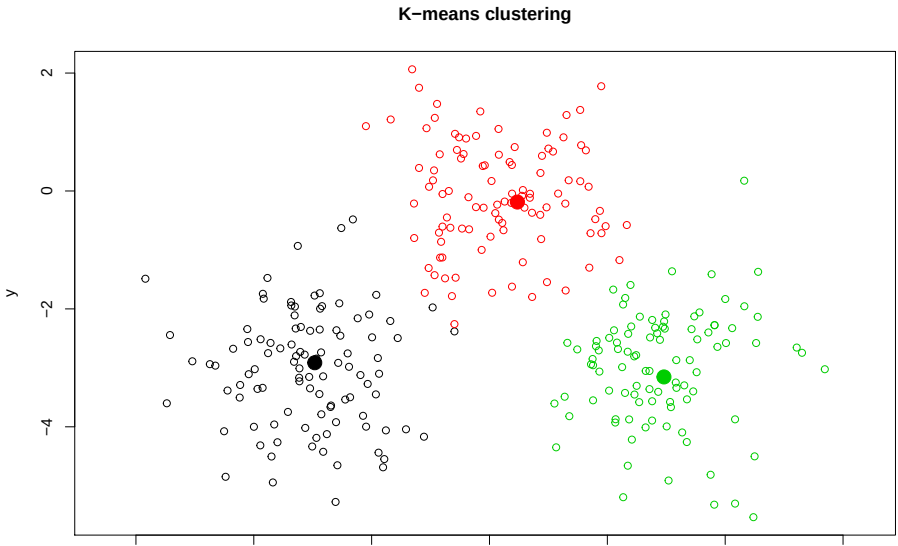
Reassign Points



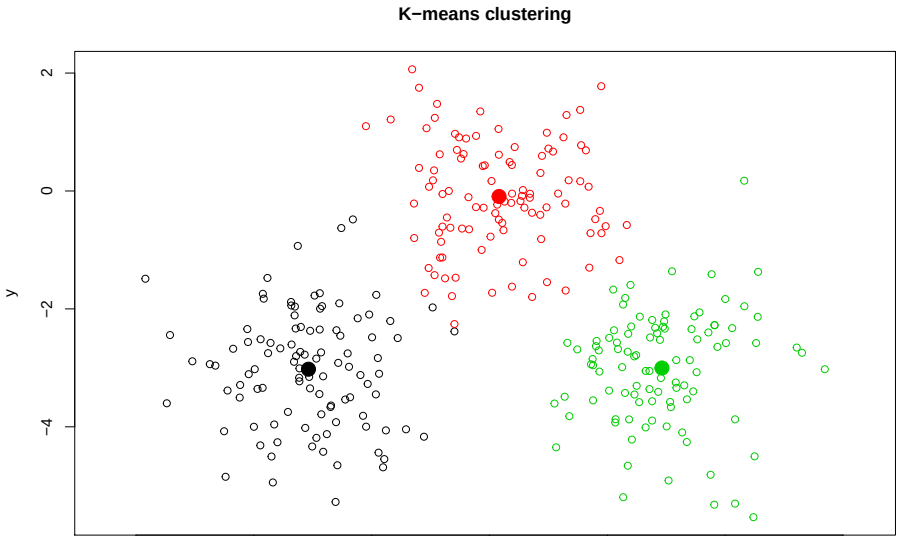
Recalculate Centroids



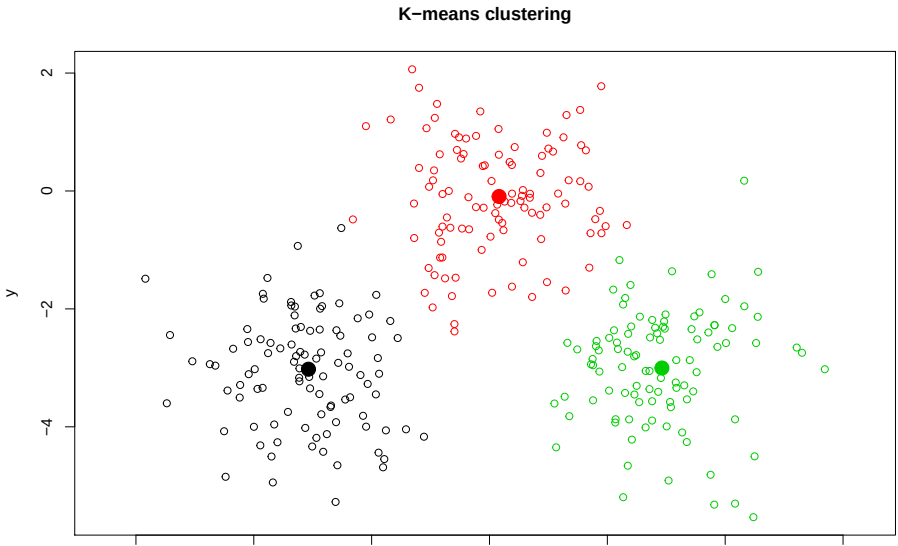
Reassign Points



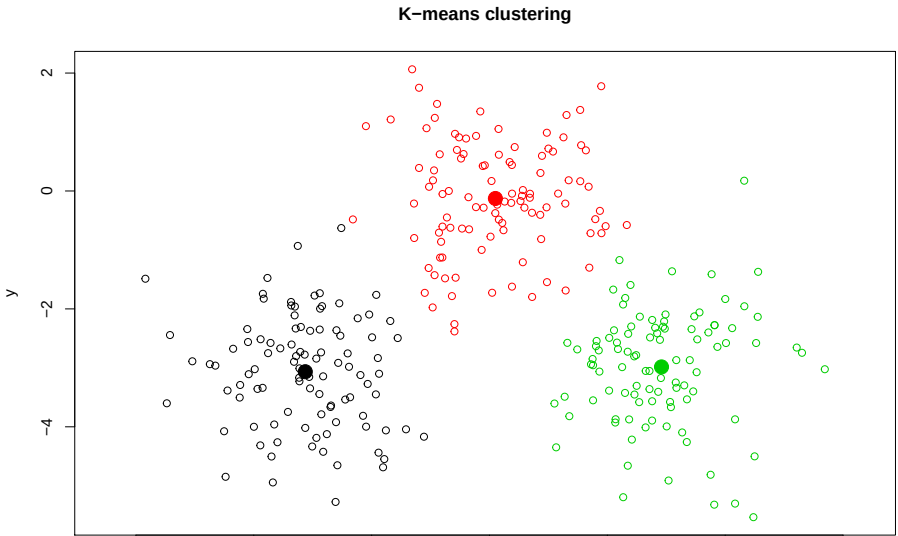
Recalculate Centroids



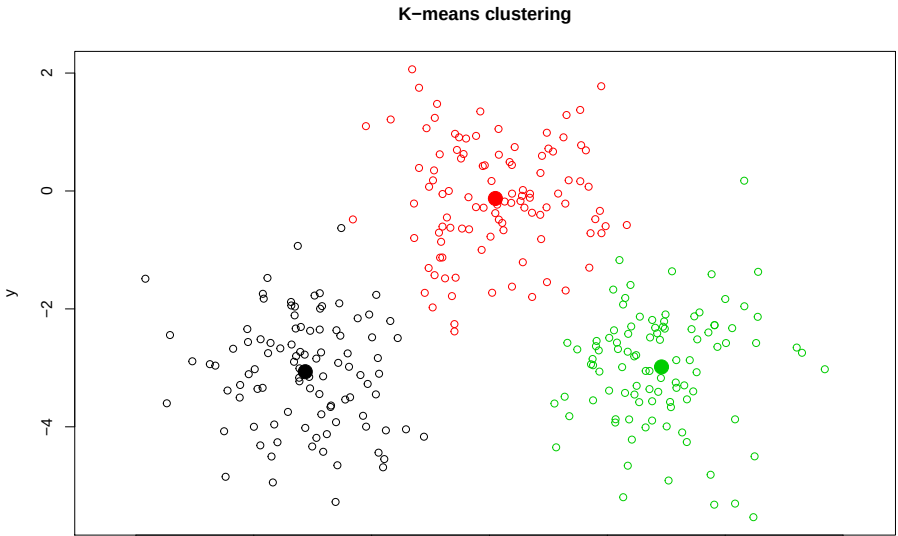
Reassign Points



Recalculate Centroids



Reassign Points (no changes: converged)



Section 2

Kernelized K-means

Kernel Functions

A kernel function finds the inner product of two vectors after they have been transformed by some function $\varphi(\mathbf{x})$.

$$K(\mathbf{x_a}, \mathbf{x_b}) = \langle \varphi(\mathbf{x_a}), \varphi(\mathbf{x_b}) \rangle$$

For a linear kernel, the transformation function ϕ is the identity function.

$$K(\mathbf{x_a}, \mathbf{x_b}) = \langle \varphi(\mathbf{x_a}), \varphi(\mathbf{x_b}) \rangle = \mathbf{x_a}^T \mathbf{x_b}$$

Example Kernel Functions

Let's say we have a transformation function ϕ that transforms a 2D vector into a 3D vector.

$$\varphi(\mathbf{x}) = \varphi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix}$$

The equivalent kernel function is:

$$\begin{aligned} K(\mathbf{x_a}, \mathbf{x_b}) &= \langle \varphi(\mathbf{x_a}), \varphi(\mathbf{x_b}) \rangle \\ &= [x_{a1}, x_{a2}, x_{a1}^2 + x_{a2}^2] \cdot [x_{b1}, x_{b2}, x_{b1}^2 + x_{b2}^2] \\ &= x_{a1}x_{b1} + x_{a2}x_{b2} + (x_{a1}^2 + x_{a2}^2)(x_{b1}^2 + x_{b2}^2) \\ &= \mathbf{x_a} \cdot \mathbf{x_b} + ||\mathbf{x_a}||^2 ||\mathbf{x_b}||^2 \end{aligned}$$

Polynomial / Quadratic Kernel

$$K(\mathbf{x_a}, \mathbf{x_b}) = (C + \mathbf{x_a}^T \mathbf{x_b})^\gamma$$

For $\gamma = 2$ and C $= 0$, the transformation function is:

$$\varphi(\mathbf{x}) = \varphi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

$$\begin{aligned} K(\mathbf{x_a}, \mathbf{x_b}) = \langle \varphi(\mathbf{x_a}), \varphi(\mathbf{x_b}) \rangle &= [x_{a1}^2, x_{a2}^2, \sqrt{2}x_{a1}x_{a2}] \cdot [x_{b1}^2, x_{b2}^2, \sqrt{2}x_{b1}x_{b2}] \\ &= (x_{a1}x_{b1})^2 + (x_{a2}x_{b2})^2 + 2x_{a1}x_{a2}x_{b1}x_{b2} \\ &= (x_{a1}x_{b1} + x_{a2}x_{b2})^2 \\ &= (\mathbf{x_a}^T \mathbf{x_b})^2 \end{aligned}$$

Other Kernel Functions

Gaussian Kernel: $K(\mathbf{x_a}, \mathbf{x_b}) = \exp(-\gamma(\mathbf{x_a} - \mathbf{x_b})^T(\mathbf{x_a} - \mathbf{x_b}))$

See: https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Polynomial Kernel: $K(\mathbf{x_a}, \mathbf{x_b}) = (C + \mathbf{x_a}^T\mathbf{x_b})^\gamma$

See: https://en.wikipedia.org/wiki/Polynomial_kernel

All Kernel Functions are inner products

All Kernel functions are inner products of vectors after they have been transformed by the function $\varphi(\mathbf{x})$.

However, we do not even need to think of the transformation itself.

Because $K(\mathbf{x_a}, \mathbf{x_b}) = \varphi(\mathbf{x_a})^T \varphi(\mathbf{x_b})$ is an inner product for some arbitrary transformation, a kernel function can be used as a substitute for an inner product in the original space (i.e. $\mathbf{x_a}^T \mathbf{x_b}$)

Distance to the centroid in K-means

In K-means clustering, we assign a point to a cluster based on its distance to a centroid.

The squared distance of point $\mathbf{x}_n$ to the centroid of cluster k, $\boldsymbol{\mu}_k$ is

$$d_{nk} = (\mathbf{x}_n - \boldsymbol{\mu}_k)^T(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

The centroid of cluster k, $\boldsymbol{\mu}_k$ is

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{j=1}^{N} z_{jk}\mathbf{x}_j$$

Where z_{nk} is 1 if point j belongs to cluster k and 0 if it does not, and $N_k = \sum z_{nk}$ the number of objects assigned to cluster k.

Distance to the centroid in K-means

If we plug in the expression for $\boldsymbol{\mu}_k$ into the formula for distance to a centroid, we get:

$$
\begin{aligned}
d_{nk} &= (\mathbf{x}_n - \boldsymbol{\mu}_k)^T(\mathbf{x}_n - \boldsymbol{\mu}_k) \\
&= \left(\mathbf{x}_n - \frac{1}{N_k}\sum_{m=1}^{N} z_{mk}\mathbf{x}_m\right)^T \left(\mathbf{x}_n - \frac{1}{N_k}\sum_{r=1}^{N} z_{rk}\mathbf{x}_r\right) \\
&= \mathbf{x}_n^T\mathbf{x}_n - \frac{1}{N_k}\sum_{m=1}^{N} z_{mk}\mathbf{x}_m^T\mathbf{x}_n - \frac{1}{N_k}\sum_{r=1}^{N} z_{rk}\mathbf{x}_n^T\mathbf{x}_r + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}\mathbf{x}_m^T\mathbf{x}_r \\
&= \mathbf{x}_n^T\mathbf{x}_n - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk}\mathbf{x}_m^T\mathbf{x}_n + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}\mathbf{x}_m^T\mathbf{x}_r
\end{aligned}
$$

We can combine the $\frac{1}{N_k}\sum_{m=1}^{N} z_{mk}\mathbf{x}_m^T\mathbf{x}_n$ and $\frac{1}{N_k}\sum_{r=1}^{N} z_{rk}\mathbf{x}_n^T\mathbf{x}_r$ because they are both exactly equal to $\boldsymbol{\mu}_k^T\mathbf{x}_n$.

Kernelized distance to the centroid in K-means

We now have an expression for the distance of a point n to the centroid of cluster k which uses only inner products and does not require that we find $\boldsymbol{\mu}_k$ directly:

$$d_{nk} = \mathbf{x}_n^T \mathbf{x}_n - \frac{2}{N_k} \sum_{m=1}^{N} z_{mk} \mathbf{x}_m^T \mathbf{x}_n + \frac{1}{N_k^2} \sum_{m=1}^{N} \sum_{r=1}^{N} z_{mk} z_{rk} \mathbf{x}_m^T \mathbf{x}_r$$

If we replace the inner products with kernel functions, we now have a kernelized distance to centroid

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k} \sum_{m=1}^{N} z_{mk} K(\mathbf{x}_m, \mathbf{x}_n) + \frac{1}{N_k^2} \sum_{m=1}^{N} \sum_{r=1}^{N} z_{mk} z_{rk} K(\mathbf{x}_m, \mathbf{x}_r)$$

By using Kernel functions, we do not actually need to transform our data into the higher dimensional space.

Kernelized K-means Clustering

We can now use this Kernelized distance metric in the K-means clustering algorithm

0) Determine how many (k) clusters you will search for.
1) Randomly assign points in your data to each of the clusters.
2) Reassign values to clusters by associating values in the data set to the nearest centroid using the Kernelized distance.
3) Repeat steps 2 until convergence. Convergence occurs when no values are reassigned to a new cluster.

Note that we no longer need to find the actual centroids themselves. The Kernelized distance will find the distances to centroids directly.

Example

To illustrate this, I will use a very simple dataset. I will do Kernelized K-means clustering twice.

In the first run, I will transform the 2D data into the higher dimensional space with the function $\varphi()$. I will then perform standard K-means clustering using Euclidean distance in 3 dimensional space.

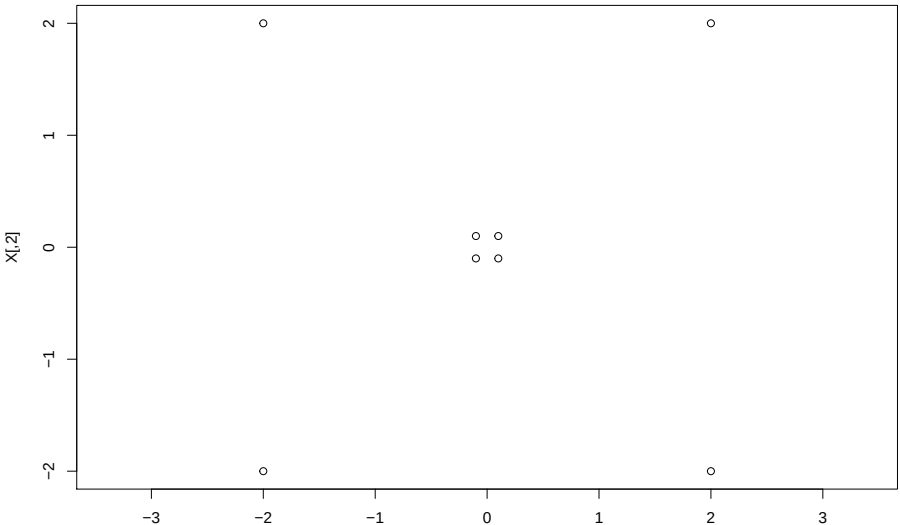
In the second run, I will perform Kernelized K-means clustering directly on the 2D data using the Kernelized distance metric.

Example

Our very simple data. 8 points. 4 near the origin. 4 that are far from the origin.

```
X <- matrix(c(
  0.1,  0.1,
  0.1, -0.1,
 -0.1,  0.1,
 -0.1, -0.1,
    2,  2,
    2, -2,
   -2, -2,
   -2,  2),
  byrow = TRUE,
  ncol = 2)
```

```r
plot(X, asp = 1)
```

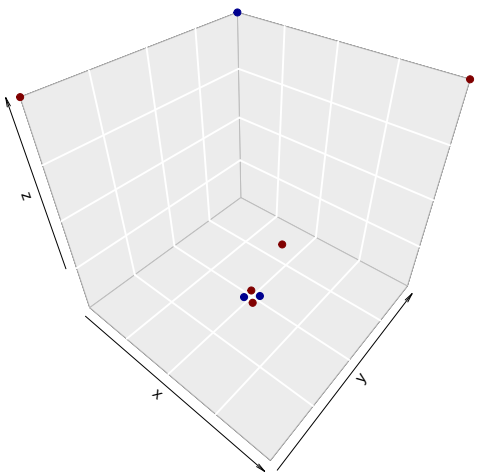


We transform a 2D coordinate $(x1, x2)$ to the 3d coordinate: $(x1, x2, x1^2 + x2^2)$

```
phi <- function(x){ return(c(x[1], x[2], x[1]^2 + x[2]^2)) }
transformed <- t(apply(X, 1, FUN = phi))
set.seed(3); assignments <- sample(c(1,2), nrow(X), replace = TRUE)
df <- data.frame(transformed, assignments); print(df)
```

```
##      X1   X2   X3 assignments
## 1  0.1  0.1 0.02           1
## 2  0.1 -0.1 0.02           2
## 3 -0.1  0.1 0.02           2
## 4 -0.1 -0.1 0.02           1
## 5  2.0  2.0 8.00           2
## 6  2.0 -2.0 8.00           2
## 7 -2.0 -2.0 8.00           2
## 8 -2.0  2.0 8.00           1
```

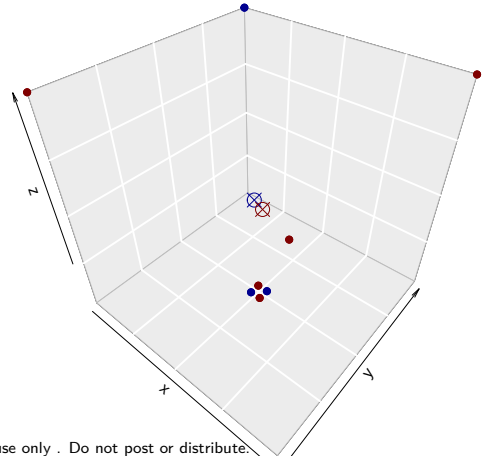
```r
library(plot3D)
scatter3D(df[,1], df[,2], df[,3], pch = 19, colvar=df[,4], bty = "g", colkey =
```



```r
# find the means of the clusters
library(dplyr)
means <- df %>% group_by(assignments) %>% summarise(x1 = mean(X1), x2 = mean(X
means <- as.matrix(means[,-1]); print(means)
```

```
##              x1         x2    x3
## [1,] -0.6666667  0.6666667 2.680
## [2,]  0.4000000 -0.4000000 4.808
```

```r
library(plot3D)
scatter3D(df[,1], df[,2], df[,3], pch = 19, colvar=df[,4], bty = "g", colkey =
points3D(means[,1], means[,2], means[,3], pch = 13, cex = 2, add = TRUE, colva
```

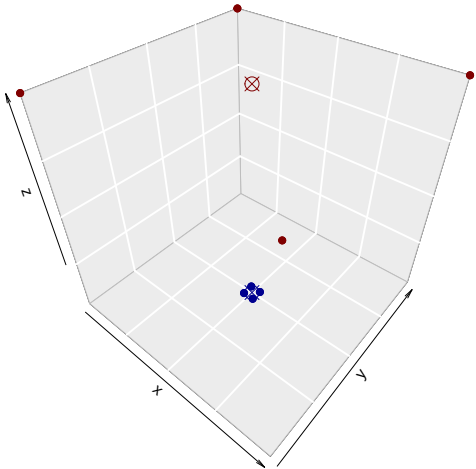


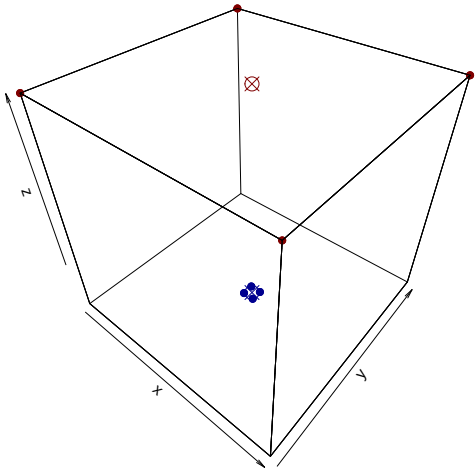
```r
# Actual 3D plot
library(plot3Drgl)
```

```
## Loading required package: rgl
```

```r
scatter3D(df[,1], df[,2], df[,3], pch = 19,
          colvar=df[,4], bty = "f", colkey = FALSE, plot = FALSE)
points3D(means[,1], means[,2], means[,3], pch = 13,
         cex = 2, add = TRUE, colvar=c(1,2), colkey = FALSE, plot = FALSE)
plotrgl()
```

```
## the centroids:

##              x1         x2    x3
## [1,] -0.6666667  0.6666667 2.680
## [2,]  0.4000000 -0.4000000 4.808

##     X1   X2   X3 dist_to_cent_a dist_to_cent_b assignments
## 1  0.1  0.1 0.02       7.984489       23.26494           1
## 2  0.1 -0.1 0.02       8.251156       23.10494           1
## 3 -0.1  0.1 0.02       7.717822       23.42494           1
## 4 -0.1 -0.1 0.02       7.984489       23.26494           1
## 5  2.0  2.0 8.00      37.191289       18.50886           2
## 6  2.0 -2.0 8.00      42.524622       15.30886           2
## 7 -2.0 -2.0 8.00      37.191289       18.50886           2
## 8 -2.0  2.0 8.00      31.857956       21.70886           2
```





```
## the centroids:

##      x1 x2   x3
## [1,]  0  0 0.02
## [2,]  0  0 8.00
```

	X1	X2	X3	dist_to_cent_a	dist_to_cent_b	assignments
1	0.1	0.1	0.02	0.0200	63.7004	1
2	0.1	-0.1	0.02	0.0200	63.7004	1
3	-0.1	0.1	0.02	0.0200	63.7004	1
4	-0.1	-0.1	0.02	0.0200	63.7004	1
5	2.0	2.0	8.00	71.6804	8.0000	2
6	2.0	-2.0	8.00	71.6804	8.0000	2
7	-2.0	-2.0	8.00	71.6804	8.0000	2
8	-2.0	2.0	8.00	71.6804	8.0000	2

Kernelized K-Means Clustering without transforming the data

With kernel functions, it is not necessary to actually transform the data into the higher dimensional space. The Kernel distance to a cluster centroid can be found directly with the equation we found earlier:

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk}K(\mathbf{x}_m, \mathbf{x}_n) + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}K(\mathbf{x}_m, \mathbf{x}_r)$$

For this transformation, the kernel function is:

$$K(\mathbf{x_a}, \mathbf{x_b}) = \mathbf{x_a} \cdot \mathbf{x_b} + ||\mathbf{x_a}||^2||\mathbf{x_b}||^2$$

```r
# Same data (untransformed) and same initial assignments
set.seed(3); assignments <- sample(c(1,2), nrow(X), replace = TRUE)
df <- data.frame(X, assignments); print(df)
```

```
##      X1   X2 assignments
## 1  0.1  0.1           1
## 2  0.1 -0.1           2
## 3 -0.1  0.1           2
## 4 -0.1 -0.1           1
## 5  2.0  2.0           2
## 6  2.0 -2.0           2
## 7 -2.0 -2.0           2
## 8 -2.0  2.0           1
```

```r
z_a <- as.integer(assignments == 1) # create vector z_a: 1 if point is in cluster a,
z_b <- as.integer(assignments == 2) # vector z_b
```

In the equation,

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk} K(\mathbf{x}_n, \mathbf{x}_m) + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk} z_{rk} K(\mathbf{x}_m, \mathbf{x}_r)$$

$K(\mathbf{x}_m, \mathbf{x}_r)$ requires us to find the Kernel value of every possible combination between point m and point r.

Recall our Kernel function: $K(\mathbf{x_a}, \mathbf{x_b}) = \mathbf{x_a} \cdot \mathbf{x_b} + ||\mathbf{x_a}||^2 ||\mathbf{x_b}||^2$

```
N = nrow(X)
Ke = matrix(0,ncol=N,nrow=N)
for(m in 1:N){
  for(r in 1:N){
    Ke[m,r] = X[m,] %*% X[r,] + norm(X[m,], "2")^2 * norm(X[r,], "2")^2
  }
}
```

```r
print(Ke) # kernel values for all possible pairs of points
```

```
##          [,1]    [,2]    [,3]    [,4]  [,5]  [,6]  [,7]  [,8]
## [1,]  0.0204  0.0004  0.0004 -0.0196  0.56  0.16 -0.24  0.16
## [2,]  0.0004  0.0204 -0.0196  0.0004  0.16  0.56  0.16 -0.24
## [3,]  0.0004 -0.0196  0.0204  0.0004  0.16 -0.24  0.16  0.56
## [4,] -0.0196  0.0004  0.0004  0.0204 -0.24  0.16  0.56  0.16
## [5,]  0.5600  0.1600  0.1600 -0.2400 72.00 64.00 56.00 64.00
## [6,]  0.1600  0.5600 -0.2400  0.1600 64.00 72.00 64.00 56.00
## [7,] -0.2400  0.1600  0.1600  0.5600 56.00 64.00 72.00 64.00
## [8,]  0.1600 -0.2400  0.5600  0.1600 64.00 56.00 64.00 72.00
```

$$\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}K(\mathbf{x}_m, \mathbf{x}_r)$$

```
double_sum_a = 0
for(m in 1:N){
  for(r in 1:N){
    double_sum_a = double_sum_a + z_a[m] * z_a[r] * Ke[m,r]
  }
}
double_sum_a
```

```
## [1] 72.6416
```

```
sum( (z_a %*% t(z_a)) * Ke ) # can also be found this way
```

```
## [1] 72.6416
```

We find the vector of distances from each point to the centroid of cluster A:

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk}K(\mathbf{x}_m, \mathbf{x}_n) + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}K(\mathbf{x}_m, \mathbf{x}_r)$$

```
distances_to_a <- rep(0,N); N_a <- sum(z_a)
for(n in 1:N){
  distances_to_a[n] = Ke[n,n] - 2 / N_a * sum(z_a * Ke[,n]) +
    1/N_a^2 * double_sum_a
}
# values match distances to centroids in transformed
# space on slide 35
distances_to_a
```

```
## [1]  7.984489  8.251156  7.717822  7.984489 37.191289 42.524622 37.191289
## [8] 31.857956
```

We find the vector of distances from each point to the centroid of cluster B:

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k}\sum_{m=1}^{N} z_{mk}K(\mathbf{x}_m, \mathbf{x}_n) + \frac{1}{N_k^2}\sum_{m=1}^{N}\sum_{r=1}^{N} z_{mk}z_{rk}K(\mathbf{x}_m, \mathbf{x}_r)$$

```
distances_to_b <- rep(0,N); N_b <- sum(z_b);
double_sum_b <- sum( (z_b %*% t(z_b)) * Ke )
for(n in 1:N){
  distances_to_b[n] = Ke[n,n] - 2 / N_b * sum(z_b * Ke[n,]) +
    N_b^(-2) * double_sum_b
}
distances_to_b # values match distances on slide 35
```

```
## [1] 23.26494 23.10494 23.42494 23.26494 18.50886 15.30886 18.50886 21.7088
```

```
distances <- cbind(distances_to_a, distances_to_b)
distances
```

```
##      distances_to_a distances_to_b
## [1,]       7.984489       23.26494
## [2,]       8.251156       23.10494
## [3,]       7.717822       23.42494
## [4,]       7.984489       23.26494
## [5,]      37.191289       18.50886
## [6,]      42.524622       15.30886
## [7,]      37.191289       18.50886
## [8,]      31.857956       21.70886
```

```
assignments <- apply(distances, 1, which.min); assignments
```

```
## [1] 1 1 1 1 2 2 2 2
```

```
df <- data.frame(X, assignments)
z_a <- as.integer(assignments == 1) # create vector z_a: 1 if point is in clu
z_b <- as.integer(assignments == 2) # vector z_b
double_sum_a <- sum( (z_a %*% t(z_a)) * Ke )
distances_to_a <- rep(0,N); N_a <- sum(z_a)
for(n in 1:N){
  distances_to_a[n] = Ke[n,n] - 2 / N_a * sum(z_a * Ke[n,]) +
    1/N_a^2 * double_sum_a
}
distances_to_b <- rep(0,N); N_b <- sum(z_b)
double_sum_b <- sum( (z_b %*% t(z_b)) * Ke )
for(n in 1:N){
  distances_to_b[n] = Ke[n,n] - 2 / N_b * sum(z_b * Ke[n,]) +
    N_b^(-2) * double_sum_b
}
df <- data.frame(df[,1:3], distances_to_a, distances_to_b)
```

```
print(df)
```

```
##     X1   X2 assignments distances_to_a distances_to_b
## 1  0.1  0.1           1         0.0200        63.7004
## 2  0.1 -0.1           1         0.0200        63.7004
## 3 -0.1  0.1           1         0.0200        63.7004
## 4 -0.1 -0.1           1         0.0200        63.7004
## 5  2.0  2.0           2        71.6804         8.0000
## 6  2.0 -2.0           2        71.6804         8.0000
## 7 -2.0 -2.0           2        71.6804         8.0000
## 8 -2.0  2.0           2        71.6804         8.0000
```

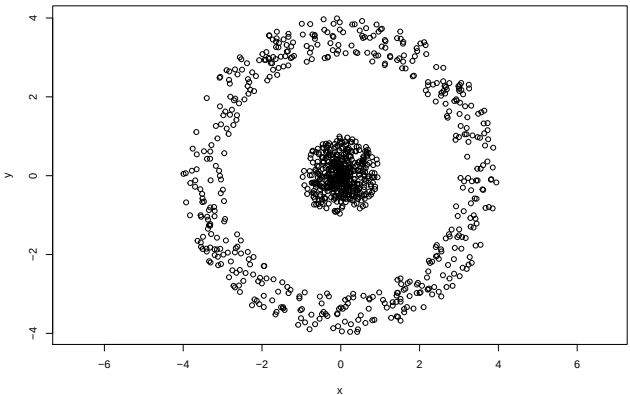
Section 3

Another Example

Artificial Example

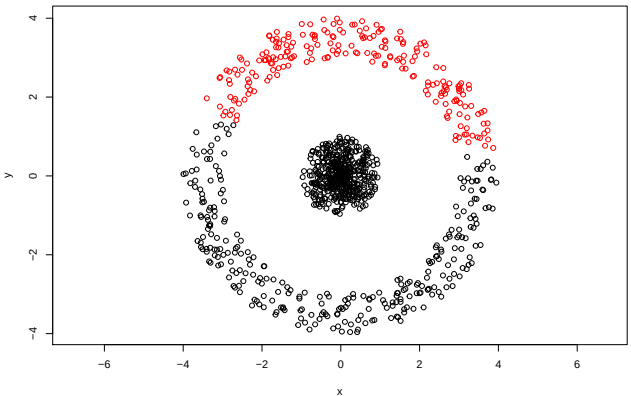
```
theta <- runif(500, 0,2*pi)
r2 <- runif(500, 3,4)
r1 <- runif(500, 0,1)
x1 <- r1*cos(theta)
y1 <- r1*sin(theta)
x2 <- r2*cos(theta)
y2 <- r2*sin(theta)
X <- cbind(x = c(x1,x2),y = c(y1,y2))
```

```r
plot(X, asp = 1)
```



Results of K-means Clustering

```
results <- kmeans(X, 2)
plot(X, col = results$cluster, asp = 1)
```

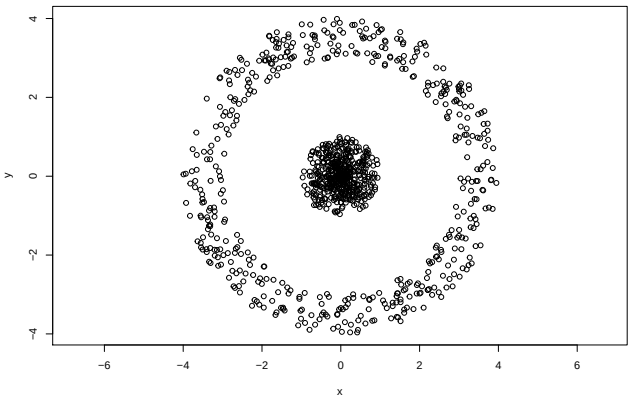


K-means clustering fails

For this silly data, K-means clustering fails because it uses Euclidean distance as a measure of similarity

Kernel K-means

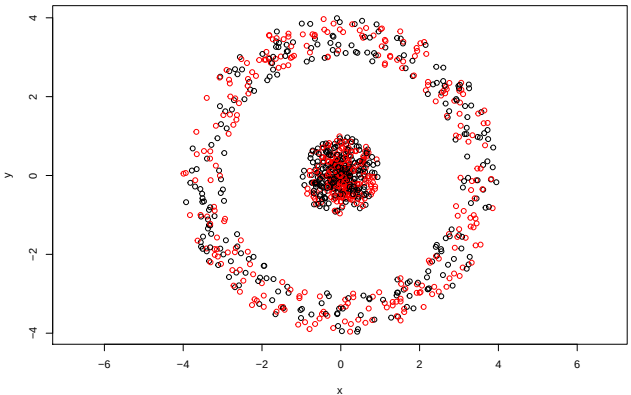
```
plot(X, asp = 1)
```



Random Assignments and Calculations

```
set.seed(1)
new_assignments <- sample(c(1,2), nrow(X), replace = TRUE)
N = nrow(X); gamma = 1
Ke = matrix(0,ncol=N,nrow=N)
for(m in 1:N){
  for(r in 1:N){
    Ke[m,r] = exp(-gamma*sum((X[m,]-X[r,])^2))  # gaussian Kernel
  }
}
```

Plot of Random Assignments

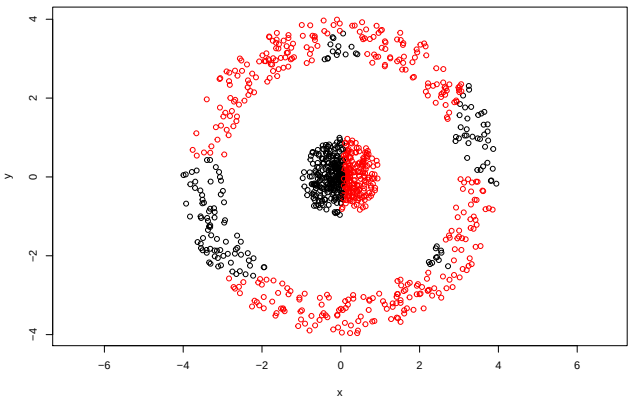


Iteration 1

```
assignments <- new_assignments
z_a <- as.numeric(assignments == 1)
z_b <- as.numeric(assignments == 2)

double_sum_a <- sum( (z_a %*% t(z_a)) * Ke )
double_sum_b <- sum( (z_b %*% t(z_b)) * Ke )

distances_to_a <- rep(0,N); N_a <- sum(z_a)
distances_to_b <- rep(0,N); N_b <- sum(z_b)
for(n in 1:N){
  distances_to_a[n] = Ke[n,n] - 2 / N_a * sum(z_a * Ke[n,]) + 1/N_a^2 * double_sum_a
  distances_to_b[n] = Ke[n,n] - 2 / N_b * sum(z_b * Ke[n,]) + 1/N_b^2 * double_sum_b
}

distances <- cbind(distances_to_a, distances_to_b)
new_assignments <- apply(distances, 1, which.min)

if(all(assignments == new_assignments)){print("converged")} else {print('not converged')}
plot(X, col = new_assignments, asp = 1)
```

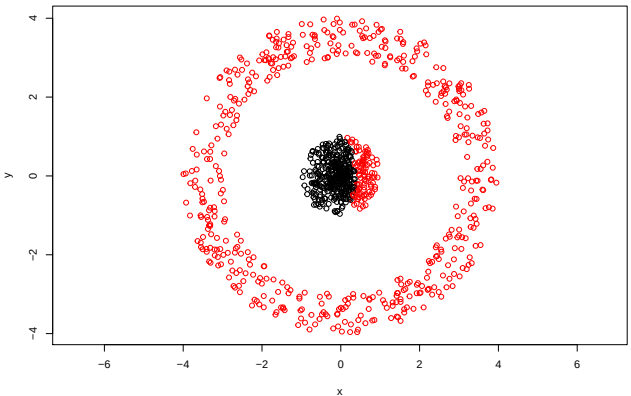
Iteration 1

```
## [1] "not converged"
```



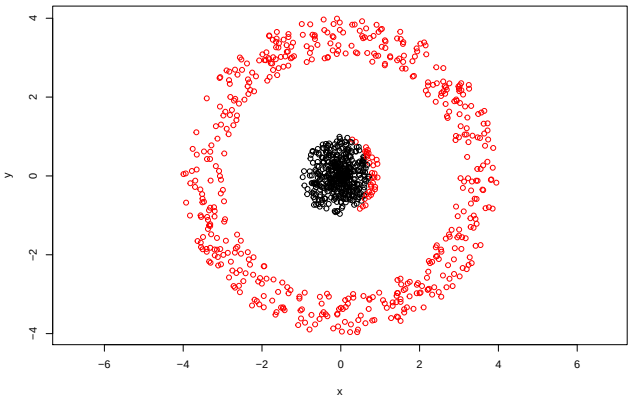
Iteration 2

```
## [1] "not converged"
```



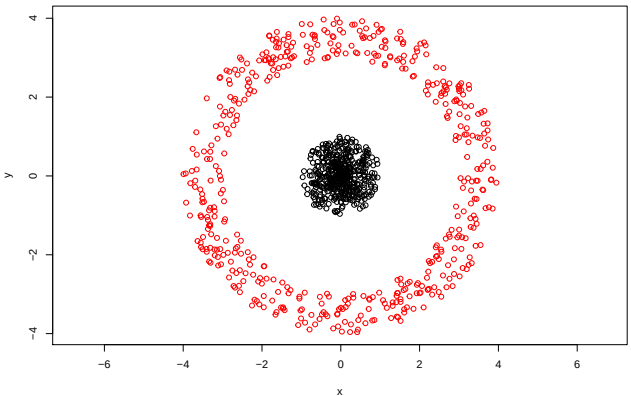
Iteration 3

```
## [1] "not converged"
```



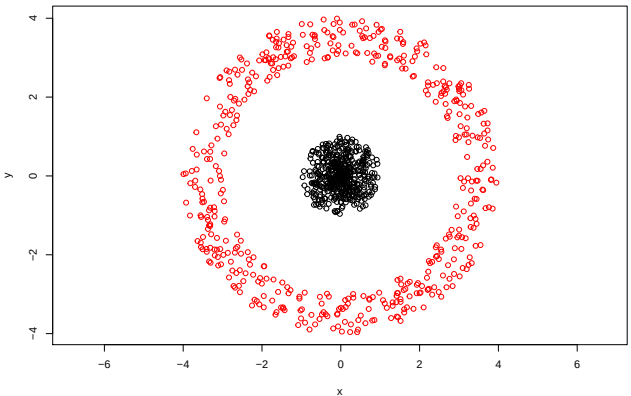
Iteration 4

```
## [1] "not converged"
```



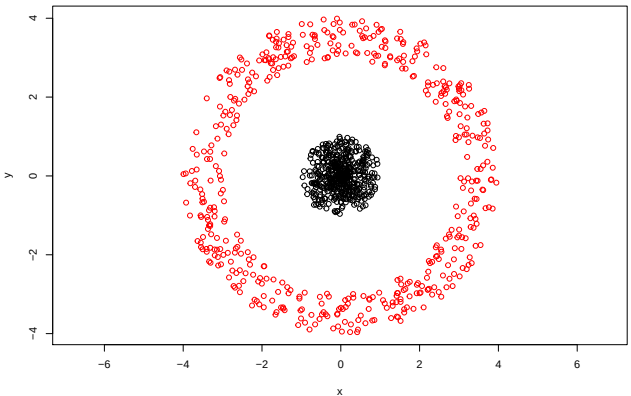
Iteration 5

```
## [1] "converged"
```



Iteration 6

```
## [1] "converged"
```



How does this work?

The most amazing thing about Kernelized K-means clustering is that the location of the centroid which exists in the higher dimensional space does not even need to be calculated.

So the actual location of the centroid is 'undefined' (or requires the use of phi), but the distance (from a point to the centroid) is expressed as an inner product that can be calculated. It's helpful to think of inner products as a measure of similarity.

A weird analogy: We start clustering actors together by how 'similar' they are. I cluster: Chris Evans (Captain America), Chris Hemsworth (Thor), Jason Momoa (Aquaman) together. In another cluster we have: Joseph Gordon Levitt (Inception, 500 Days of Summer), Jesse Eisenberg (Social Network, Zombie land), Adam Scott (Parks and Rec, Big Little Lies).

Let's say I ask you to put Martin Freeman (The Hobbit, Watson from Sherlock) into a cluster.

The 'centroid' of the two cluster is not defined nor may not be known. However, by being able to measure the similarity (find inner product) of our 'test case' Martin Freeman to the other actors that have been clustered, we can get a measure of 'distance' the actor has to each the existing clusters.