portfolio_4

Henry Bourne

2023-02-13

Cluster Analysis

Task 1

First lets load in the data which we will use in this task which will be the iris dataset :

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
##                 
```

First let's perform **agglomerative clustering**. First we must calculate the distances between the datapoints, for your distance measure we will use the euclidian distance (which is equivalent to minkowski with power of 2). We will use this distance measure as it works well when working with low-dimensional data (which we are), is easy to calculate, the data doesn't need to be scaled and the fact that we will be using the euclidian distance to perform k-means shortly.

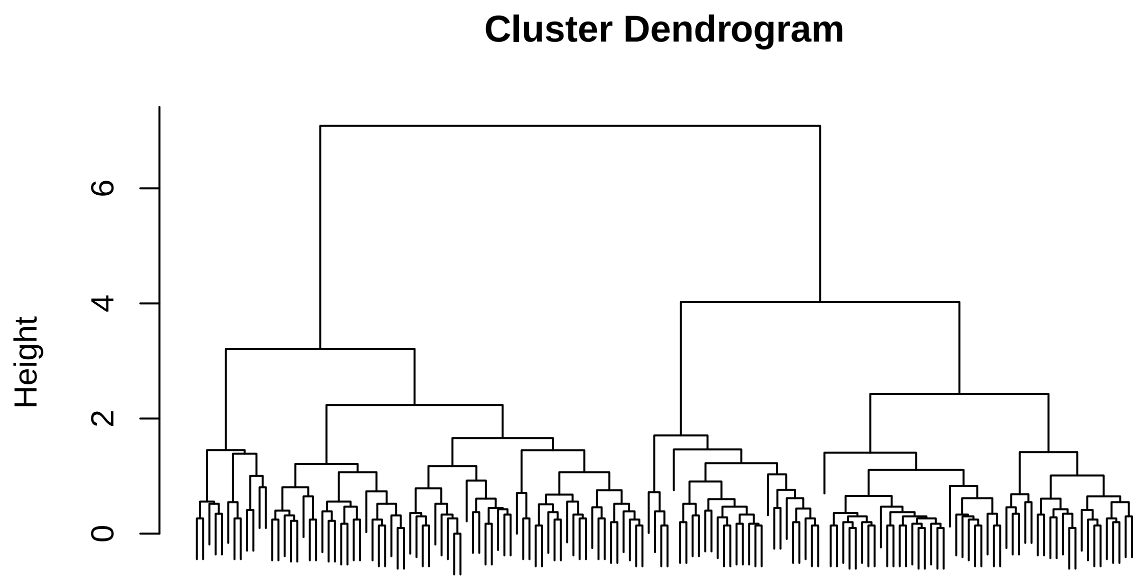
```
dist_iris <- dist(iris[,-5], method = "euclidian")
```

Now we will perform the agglomerative clustering. We will calculate $d_{t,(r,s)}$ using the Lance-Williams recursive formula with complete linkage:

```
agglo_iris <- hclust(dist_iris, method="complete")
```

Now let’s plot the **dendogram** to summarize the agglomerative clustering process:

```
plot(agglo_iris, labels = FALSE, xlab="", sub = "")
```



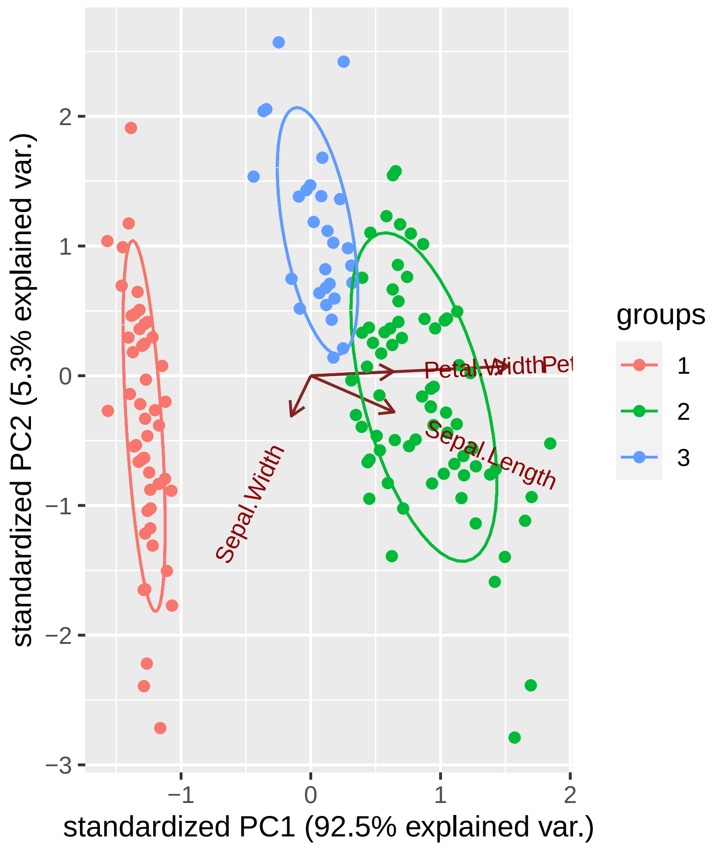
We will use three clusters as we want to use the clustering as a model to help us identify from which of the three species each datapoint belongs, let’s now workout the clustering for K=3:

```
clust_agglo_iris <- cutree(agglo_iris, k=3); clust_agglo_iris
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 2 3 2 2
##  [75] 2 2 2 2 2 3 3 3 3 2 3 2 2 2 3 3 3 2 3 3 3 3 3 2 3 3 2 2 2 2 2 2 3 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2
```

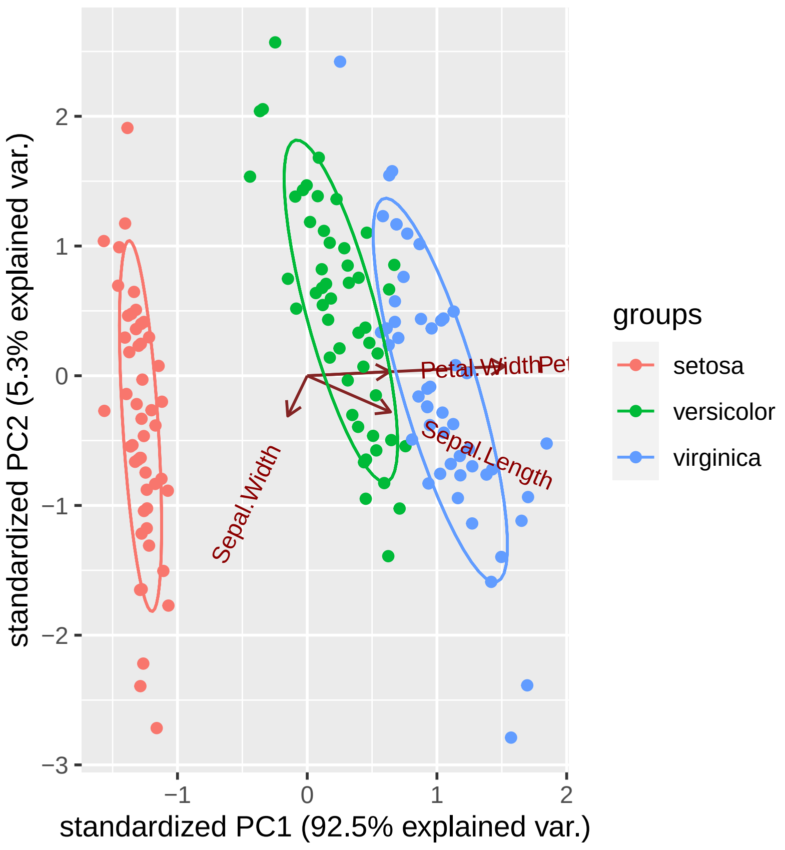
Let’s now perform PCA and plot a 2-dimensional reduction of our data to visualize our clusters, note that for PCA we will use the covariance matrix as the data doesn’t need to be scaled and this way we preserve variance:

```
iris_pca <- prcomp(~ . -Species, iris)
ggbiplot(iris_pca, ellipse=TRUE, groups = as.factor(clust_agglo_iris))
```



We can compare this with the true labels:

```
ggbiplot(iris_pca, ellipse=TRUE, groups = iris$Species)
```



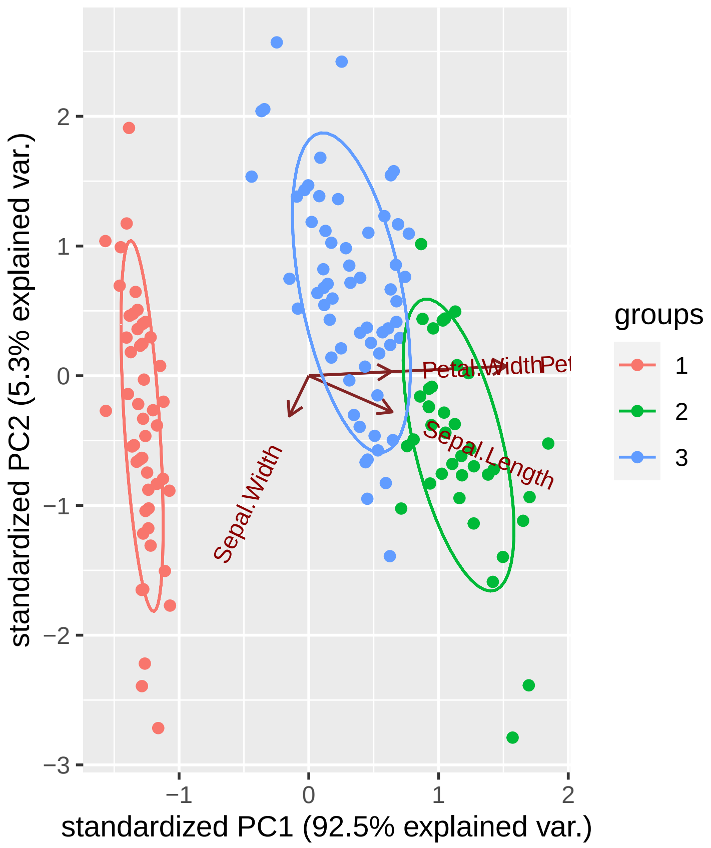
Comparing we see that the clustering has done mostly a good job, however, it is wrongly clustering many datapoints that are versicolor as virginica. Applying alggomerative clustering here works fine as the data is scaled well and the dataset is small, however, we are not particularly interested in many of the added benefits of using hierarchichal clustering here. We are mainly interested in categorizing the datapoints into the 3 species we aren't so interested in clustering for varying levels of granularity, we don't have any categorical variables and there aren't many outliers.

Let's now perform **K-means clustering**, we will use the Euclidean norm as k-means minimizes within-cluster variance which is identical to the sum of squared euclidean distances from the center. We will perform k-means clustering using LLoyds algorithm and with 3 clusters (as we are again trying to identify which of the three species each datapoint belongs to):

```
clust_kmeans_iris <- kmeans(iris[,-5], centers=3, algorithm = "Lloyd")
```

Again we will use PCA to produce a two dimensional reduction of our data to visualize the clusters:

```
ggbiplot(iris_pca, ellipse=TRUE, groups = as.factor(clust_kmeans_iris$cluster))
```

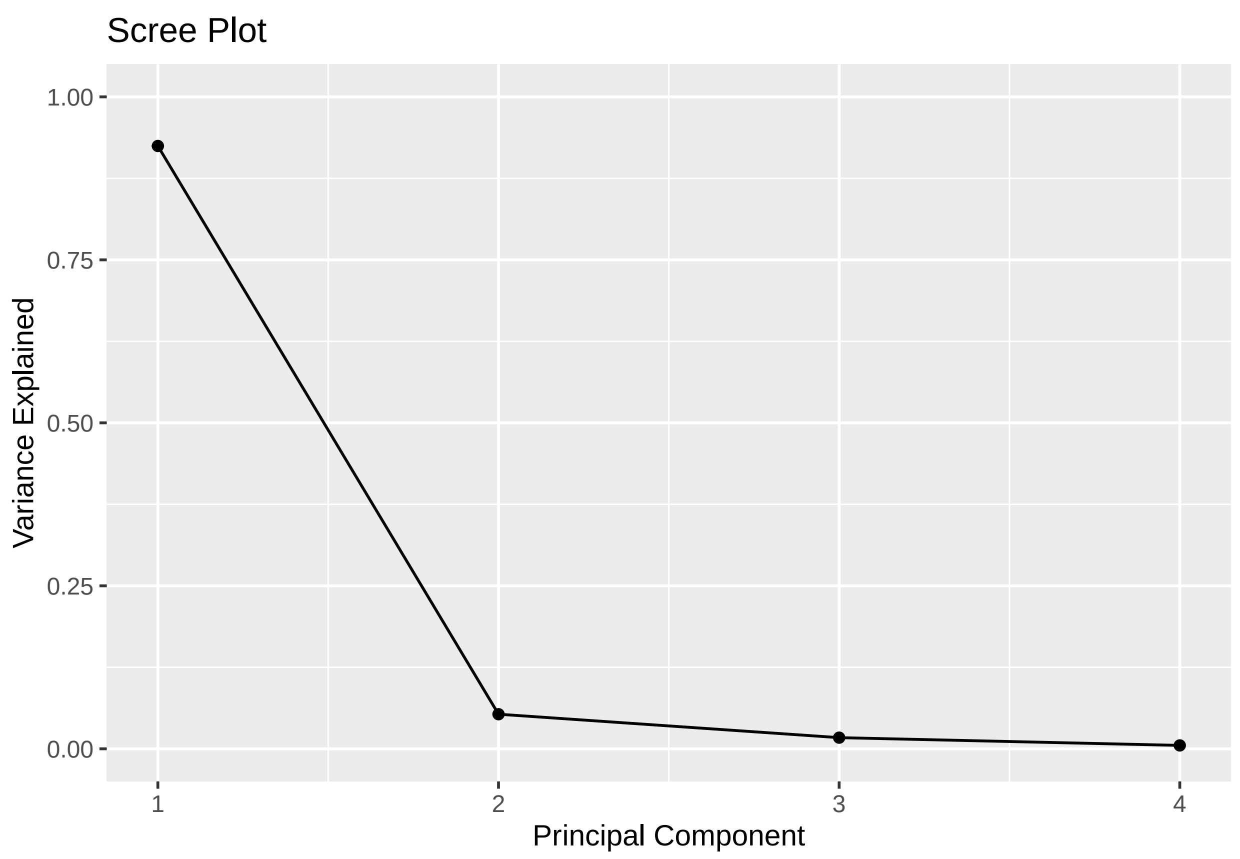


Looking at the above plot it appears k-means has done a very good job in clustering and appears to do slightly better in correctly clustering versicolor and virginica, although this time identifies more datapoints as versicolour. As we know the initial number of clusters, the data is scaled well, there aren't any categorical variables, we can assume spherical density, data isn't very high dimensional and we know the number of clusters we need k-means clustering is very appropriate as a choice of clustering method here.

Let's now perform k-means clustering on a dimensionality reduction of our dataset, we will use PCA to obtain this reduction. We've already carried out PCA so let's now produce a scree plot to help us choose the number of principal components:

```
var_prcnt = iris_pca$sdev^2 / sum(iris_pca$sdev^2)

qplot(c(1:4), var_prcnt) +
  geom_line() +
  xlab("Principal Component") +
  ylab("Variance Explained") +
  ggtitle("Scree Plot") +
  ylim(0, 1)
```

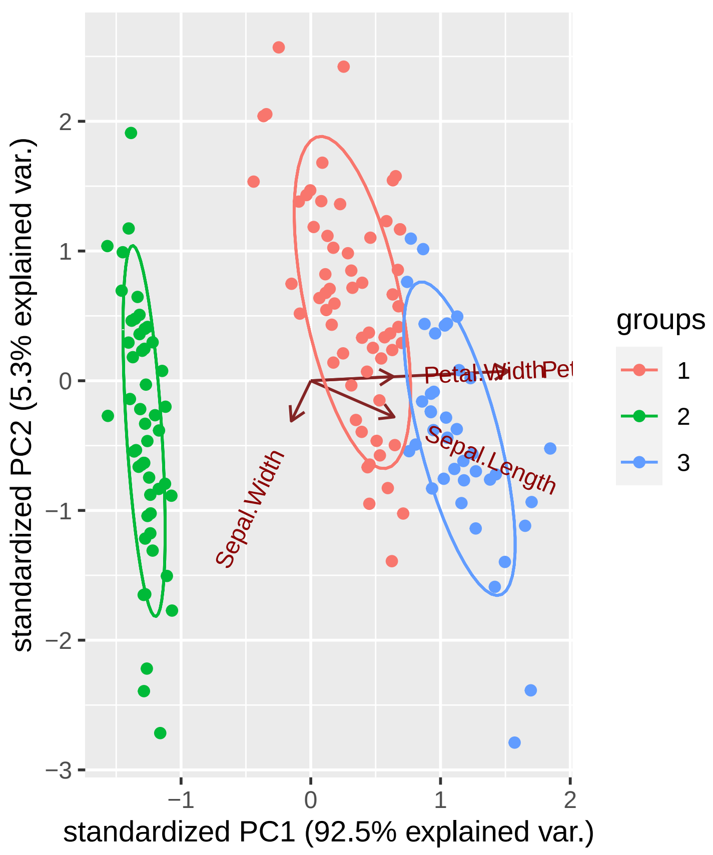
```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```



The first principal component explains a very large percentage of the variance, so let's simply perform k-means clustering using the data rotated by just the first principal component:

```
iris_rotated <- as.matrix(iris[,-5]) %*% iris_pca$rotation[,1]
clust_kmeans_iris_pca <- kmeans(iris_rotated, centers=3, algorithm = "Lloyd")
```

```
ggbiplot(iris_pca, ellipse=TRUE, groups = as.factor(clust_kmeans_iris_pca$cluster))
```



It seems visually that this performs slightly better than when we did k-means on the non reduced dataset, computing and comparing the number of miss-classified datapoints we can confirm this:

```
species.true <- as.integer(iris$Species)

species.kmeans <- replace(clust_kmeans_iris$cluster, clust_kmeans_iris$cluster==2, 4)
species.kmeans <- replace(species.kmeans, species.kmeans==1, 5)
species.kmeans <- replace(species.kmeans, species.kmeans==3, 6)
species.kmeans <- replace(species.kmeans, species.kmeans==4, 3)
species.kmeans <- replace(species.kmeans, species.kmeans==5, 1)
species.kmeans <- replace(species.kmeans, species.kmeans==6, 2)

species.kmeans_pca <- replace(clust_kmeans_iris_pca$cluster, clust_kmeans_iris_pca$cluster==1, 4)
species.kmeans_pca <- replace(species.kmeans_pca, species.kmeans_pca==3, 5)
species.kmeans_pca <- replace(species.kmeans_pca, species.kmeans_pca==4, 1)
species.kmeans_pca <- replace(species.kmeans_pca, species.kmeans_pca==5, 3)

sum(ifelse(species.kmeans - species.true != 0, 1, 0))
```

```
## [1] 16
```

```
sum(ifelse(species.kmeans_pca - species.true != 0, 1, 0))
```

```
## [1] 112
```

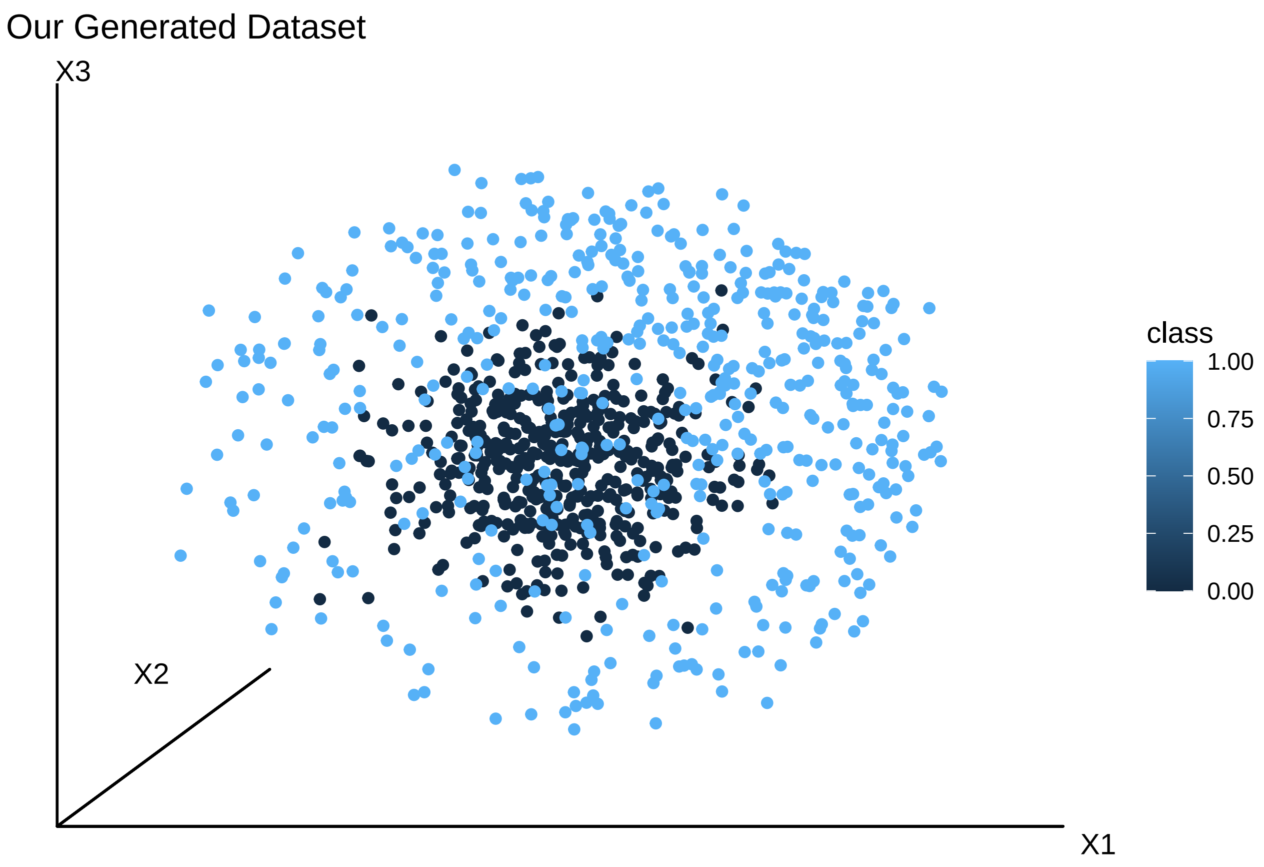
Task 2

For this task we will use the following dataset (also used in portfolio 3):

```r
library(Rfast)
```

```
## Loading required package: Rcpp

## Loading required package: RcppZiggurat
```

```r
set.seed(321)
n <- 1000
d <- 3
C0 <- rmvnorm(n/2, rep(0,d), diag(rep(0.06, d)))
C1 <- rvmf(n/2, mu = c(1, 1, 1) / sqrt(10), k = 1) + rmvnorm(n/2, rep(0,d), diag(rep(0.005, d)))
D <- data.frame(rbind(C0, C1), "class"= c(rep(0,n/2), rep(1,n/2) ) )

library("gg3D")
ggplot(D, aes(x=X1, y=X2, z=X3, color=class)) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
            angle=c(0,0,0),
            hjust=c(0,2,2),
            vjust=c(2,2,-2)) +
  ggtitle("Our Generated Dataset")
```



This data although spherical is not linearly seperable, we will perform both kmeans clustering and spectral clustering and compare their performance on this dataset. First let's perform kmeans clustering, where we set K=2 (as there are two classes) and we will use LLoyds algorithm:

```r
clust_kmeans_D <- kmeans(D[,-4], centers=2, algorithm = "Lloyd", iter.max = 100)
```

Now let's perform spectral clustering:

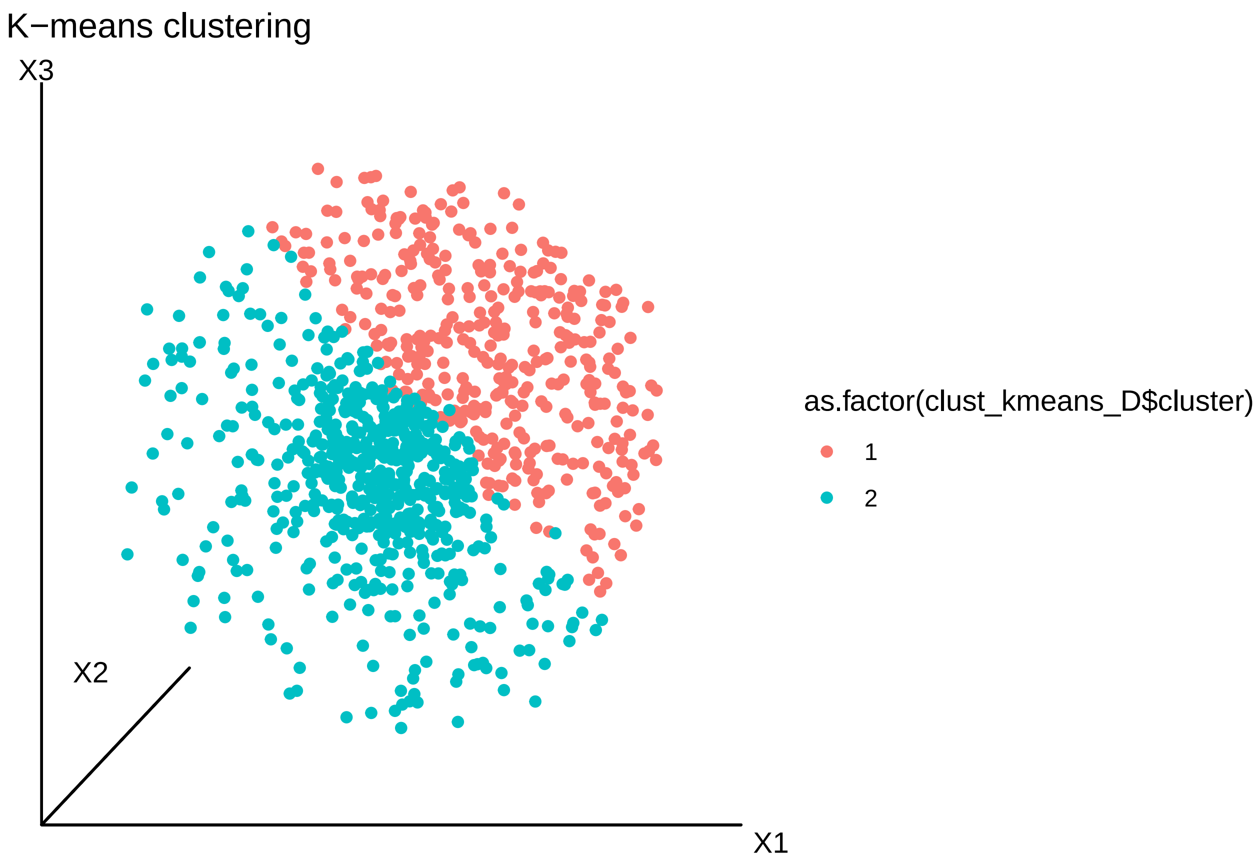
```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:scales':
##
##     alpha

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
clust_spectral_D <- specc(as.matrix(D[,-4]), centers=2)
```

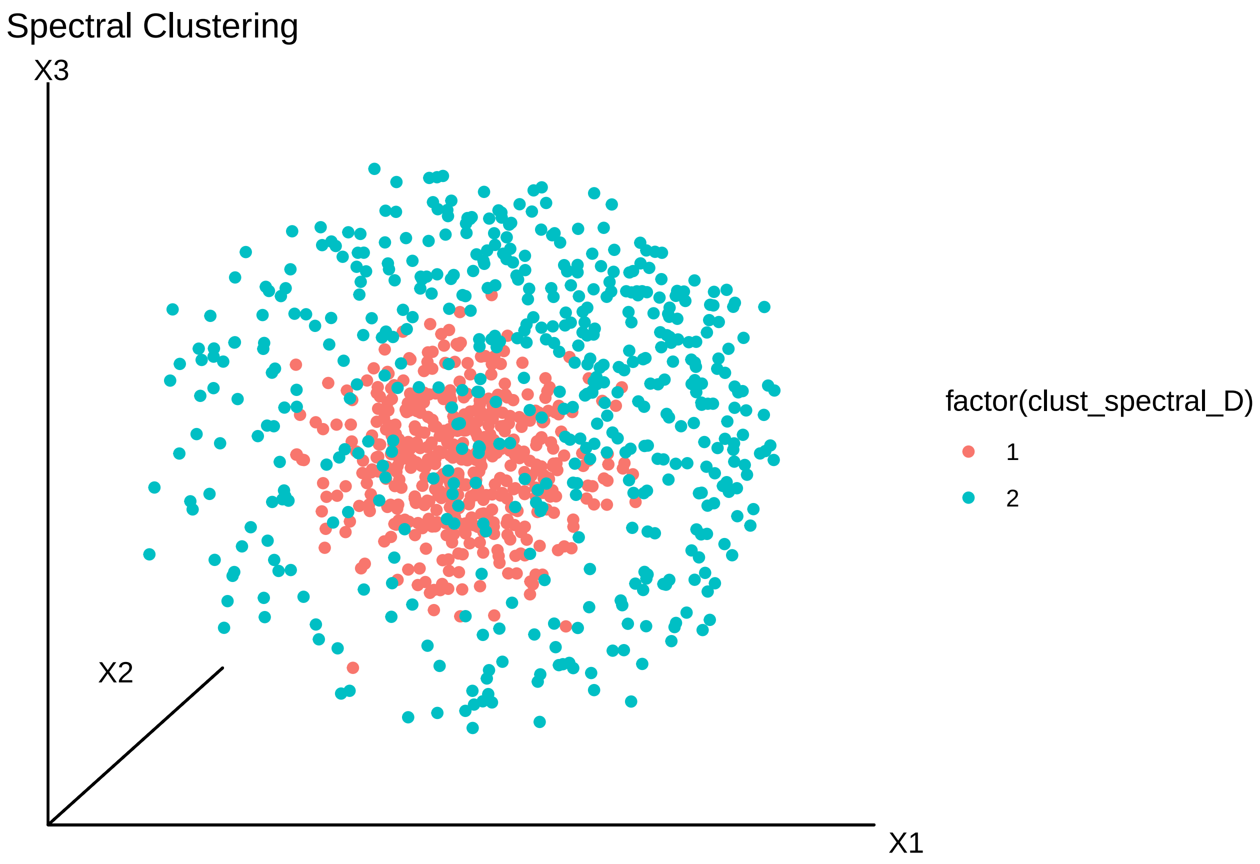
Let's now plot our data coloured by cluster:

```r
ggplot(D, aes(x=X1, y=X2, z=X3, colour=as.factor(clust_kmeans_D$cluster))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
            angle=c(0,0,0),
            hjust=c(0,2,2),
            vjust=c(2,2,-2)) +
  ggtitle("K-means clustering")
```



We see that using kmeans clustering has led to poor clusters, where the clusters don't at all represent the two classes. Let's now do the same for our clusters obtained using spectral clustering:

```
ggplot(D, aes(x=X1, y=X2, z=X3, colour=factor(clust_spectral_D))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
          angle=c(0,0,0),
          hjust=c(0,2,2),
          vjust=c(2,2,-2)) +
  ggtitle("Spectral Clustering")
```



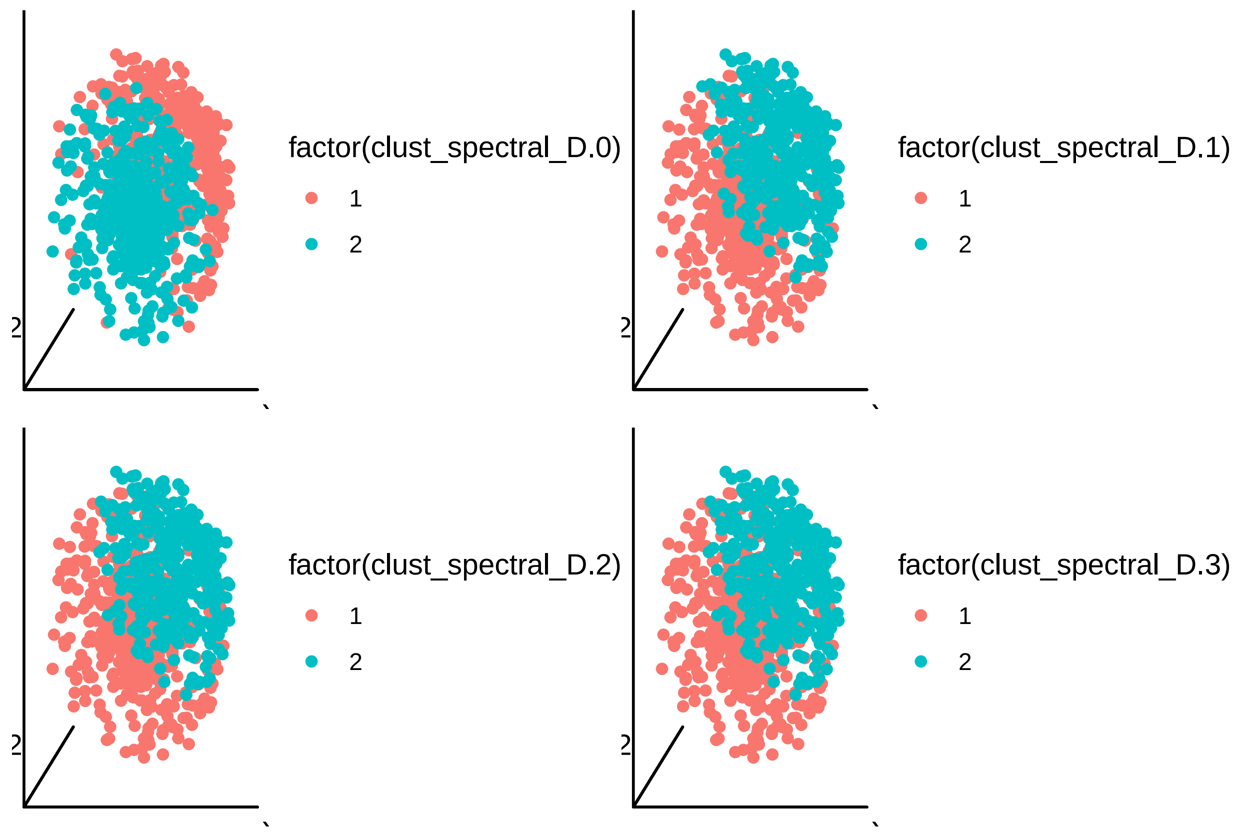
The clusters we have obtained here are much much more representative of the two classes, although definitely not perfect.

Now we will investigate how varying the similarity measure affects the clustering performance. We can change the similarity measure being used by adjusting the hyperparameter for the RBF kernel remembering that if $s_{il} = \exp(-d_{il}^2/c)$ the matrix $[s_{il}]$ is the Gram matrix associated to the RBF kernel ($k(x, x') = \exp(-||x - x'||^2/c^2)$):

```
clust_spectral_D.0 <- specc(as.matrix(D[,-4]), centers=2, kernel="rbfdot", kpar=list("sigma"=0.1))
clust_spectral_D.1 <- specc(as.matrix(D[,-4]), centers=2, kernel="rbfdot", kpar=list("sigma"=0.2))
clust_spectral_D.2 <- specc(as.matrix(D[,-4]), centers=2, kernel="rbfdot", kpar=list("sigma"=0.5))
clust_spectral_D.3 <- specc(as.matrix(D[,-4]), centers=2, kernel="rbfdot", kpar=list("sigma"=1))
```

Let's now plot our clusters:

```
library(patchwork)
plt0 <- ggplot(D, aes(x=X1, y=X2, z=X3, colour=factor(clust_spectral_D.0))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
            angle=c(0,0,0),
            hjust=c(0,2,2),
            vjust=c(2,2,-2))

plt1 <- ggplot(D, aes(x=X1, y=X2, z=X3, colour=factor(clust_spectral_D.1))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
          angle=c(0,0,0),
          hjust=c(0,2,2),
          vjust=c(2,2,-2))

plt2 <- ggplot(D, aes(x=X1, y=X2, z=X3, colour=factor(clust_spectral_D.2))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
          angle=c(0,0,0),
          hjust=c(0,2,2),
          vjust=c(2,2,-2))

plt3 <- ggplot(D, aes(x=X1, y=X2, z=X3, colour=factor(clust_spectral_D.3))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
          angle=c(0,0,0),
          hjust=c(0,2,2),
          vjust=c(2,2,-2))

plt0+plt1+plt2+plt3
```



We see that by adjusting only hyperparameters of the similarity measure the results we obtain can be altered quite a bit, if we were to change the similarity measure itself entirely then we could expect even more differing results.

Note that we performed spectral clustering using M=2, this works well for our data as we want to partition it into two as we want to assign each datapoint to one of the two classes in the dataset.

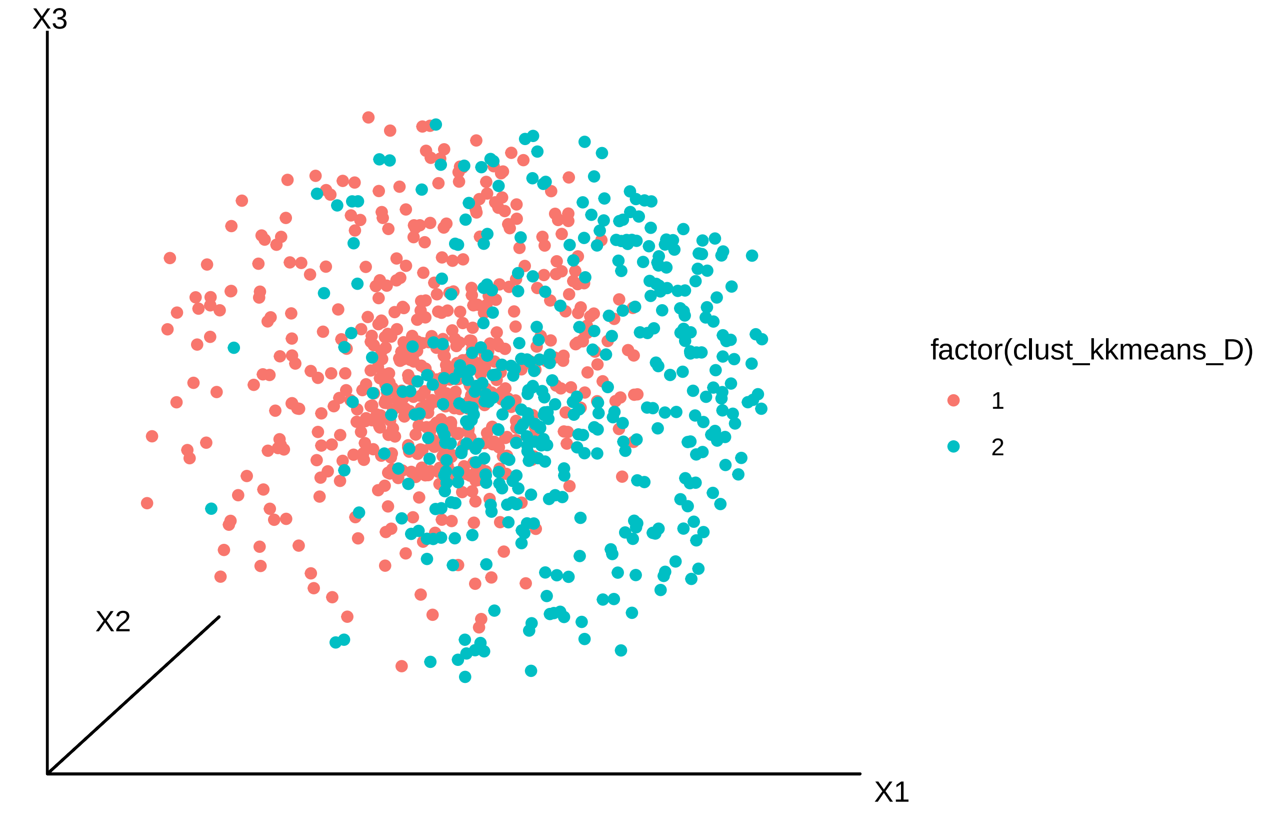
Task 3

Let X be the matrix of our observations, B be the matrix of cluster assignments of each datapoint and D be the diagonal matrix with the proportion of datapoints assigned to each cluster along the diagonal. We can then rewrite our objective function in vectorized format as follows: $\min_B ||X - XBDB^T||_F^2$ (the norm here is the frobenius norm). This is the same as the following objective function: $\max_B \text{trace}(X^T XBDB^T)$. Note that we only have $X^T X$ in our objective function which means we don't have to evaluate the transformation of X and we can simply use the kernel function $k(.,.)$ to compute $\phi(X)^T \phi(X)$.

Now we can go ahead and perform kernel k-means clustering, we will use the rbf kernel as this is a good general purpose kernel that should make the resulting induced feature space linearly separable. We will set K=2 as we would like to partition the data into two based on similarity to try and recover the class labels of the data. We will also use the median heuristic as the bandwidth parameter which is a very popular way to set the bandwidth due to showing good empirical performance (aswell as a little theory backing up the use of the median heuristic).

```r
library(ZVCV)
median_heuristic <- medianTune(as.matrix(D[,-4]))
clust_kkmeans_D <- kkmeans(as.matrix(D[,-4]), 2, kernel="rbfdot", kpar=list("sigma"= median_heuristic )

ggplot(D, aes(x=X1, y=X2, z=X3, colour=factor(clust_kkmeans_D))) +
  theme_void() +
  axes_3D(theta= 0, phi =0) +
  stat_3D(theta=0, phi=0) +
  labs_3D(theta=-13, phi=10,  labs=c("X1", "X2", "X3"),
          angle=c(0,0,0),
          hjust=c(0,2,2),
          vjust=c(2,2,-2)) +
  ggtitle("Kernel K-means Clustering")
```

Kernel K−means Clustering



From the above plot we visually can see that kernel k-means does a pretty good job here with the clustering. We note that the performance appears very similar, however, to that of spectral clustering from the previous task. The reason for this being that kernel k-means and spectral clustering are equivalent (as we alluded to in task 2), the difference here being that we set the bandwidth parameter to the median heuristic. Applying this clustering technique is appropriate in this case as it allows us to cluster non-linearly separable data using a slight variation on an extremely well tried and tested technique.