**Partitioning Clustering (K-means)**

**K-means**

The k-means algorithm is very simple and basically consists of two steps. It is initialized by a random choice of cluster centres, e.g. a random selection of objects in the data set or random values within the range for each variable. The steps are:

1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

In this tutorial we want to show how to use K-means in R through few examples (codes are provided in blue colour).

**IRIS Dataset (Our first simple k-means code)**

The dataset is the Iris dataset; this dataset contains data on flowers from three different species of Iris (i.e. three classes/groups): setosa, versicolor and virginica. Each observation contains 4 variables, the petal width, petal length, sepal width and sepal length. This dataset is labelled since it contains the species of the flower (i.e. we know the desired output). Let’s see if the unsupervised k-means algorithm can detect the species on its own (i.e. without utilising the desired output knowledge)! Iris data is considered as a classic benchmark dataset.

library(datasets)

We can show the iris data with this command, just type "iris" for show the all data

head(iris)

+/- Output

## 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

Or we can use command "names" for show the label/column names

names(iris)

+/- Output

## [1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"

## [5] "Species"

In this example, we assign the data from column 1-4 (features) to variable x, and the class to variable y

x = iris[,-5]

y = iris$Species

Create kmeans model with this command: (You need to put the number how many cluster you want, in this case we can use 3 because we already now in iris data we have 3 classes)

kc <- kmeans(x,3)

type "kc" for show summary

Kc

+/- Output

## K-means clustering with 3 clusters of sizes 62, 38, 50

##

## Cluster means:

## Sepal.Length Sepal.Width Petal.Length Petal.Width

## 1 5.902 2.748 4.394 1.434

## 2 6.850 3.074 5.742 2.071

## 3 5.006 3.428 1.462 0.246

##

## Clustering vector:

## [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

## [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

## [71] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2

## [106] 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2

## [141] 2 2 1 2 2 2 1 2 2 1

##

## Within cluster sum of squares by cluster:

## [1] 39.82 23.88 15.15

## (between\_SS / total\_SS = 88.4 %)

##

## Available components:

##

## [1] "cluster" "centers" "totss" "withinss"

## [5] "tot.withinss" "betweenss" "size" "iter"

## [9] "ifault"

After we know the result, we need to know how many error and missing data, so we need to compare the clustering result with the species/classes iris data. We use table for comparison

table(y,kc$cluster)

+/- Output

##

## y 1 2 3

## setosa 0 0 50

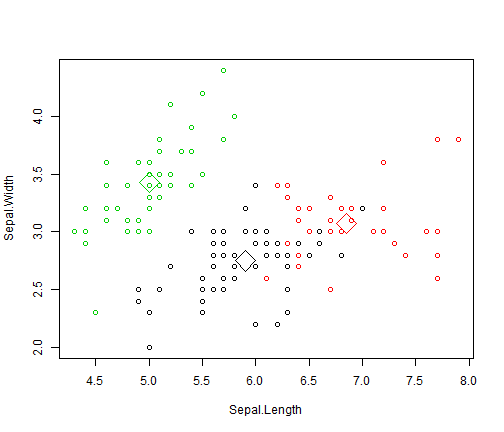
## versicolor 48 2 0

## virginica 14 36 0

For plotting we can use plot function. In this case we plot the Sepal length as x-axis and Sepal Width as y-axis, you may choose different.

plot(x[c("Sepal.Length", "Sepal.Width")], col=kc$cluster)

points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=23, cex=3)



You can see, that although “green” samples are linear separable from the other two classes, this cannot be verified for the other two (“black” & “red”).

The format of the K-means function in R is kmeans(*x*, *centers*) where *x* is a numeric dataset (matrix or data frame) and *centers* is the number of clusters to extract. The function returns the cluster memberships, centroids, sums of squares (within, between, total), and cluster sizes. Since K-means cluster analysis starts with k randomly chosen centroids, a different solution can be obtained each time the function is invoked. Use the set.seed() function to guarantee that the results are reproducible. Additionally, this clustering approach can be sensitive to the initial selection of centroids. The kmeans() function has an nstart option that attempts multiple initial configurations and reports on the best one. For example, adding nstart=25 will generate 25 initial configurations. This approach is often recommended.

**Example 2 (Variation from the previous):**

Since the initial cluster assignments are random, let us set the seed to ensure reproducibility.

library(datasets)

head(iris)

set.seed(20)

irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)

irisCluster

In the above code, rather than taking four variables as inputs, we have selected only two (3rd & 4th).

K-means clustering with 3 clusters of sizes 46, 54, 50

Cluster means:

Petal.Length Petal.Width

1 5.626087 2.047826

2 4.292593 1.359259

3 1.462000 0.246000

Clustering vector:

[1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

[35] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[69] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1

[103] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1

[137] 1 1 2 1 1 1 1 1 1 1 1 1 1 1

Within cluster sum of squares by cluster:

[1] 15.16348 14.22741 2.02200

(between\_SS / total\_SS = 94.3 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size" "iter"

[9] "ifault"

Since we know that there are 3 species involved, we can ask the algorithm to group the data into 3 clusters, and since the starting assignments are random, we specify nstart = 20. This means that R-code will try 20 different random starting assignments and then select the one with the lowest within cluster variation. We can see the cluster centroids, the clusters that each data point was assigned to, and the within cluster variation. Let us compare the clusters with the species.

table(irisCluster$cluster, iris$Species)

setosa versicolor virginica

1 0 2 44

2 0 48 6

3 50 0 0

As we can see, the data belonging to the setosa species got grouped into cluster 3, versicolor into cluster 2, and virginica into cluster 1. The algorithm wrongly classified two data points belonging to versicolor and six data points belonging to virginica.

**IRIS Dataset (k-means code that includes also some tools for “finding” the proper number of clusters)**

**NbClust** is an R Package for determining the relevant number of clusters in a dataset. It provides 30 indices which determine the number of clusters in a data set and it offers also the best clustering scheme from different results to the user. In addition, it provides a function to perform k-means and hierarchical clustering with different distance measures and aggregation methods. Any combination of validation indices and clustering methods can be requested in a single function call. This enables the user to simultaneously evaluate several clustering schemes while varying the number of clusters, to help determining the most appropriate number of clusters for the data set of interest. So, you need to install the related package in you RStudio environment.

install.packages("NbClust")

|  |
| --- |
|  |

You can then load it in memory as

library(NbClust)

To load the Iris dataset in R type at the console

data(iris)

To see the dataset column headings and datatype

str(iris)

head(iris)

We need to remove the column ‘Species’ from this dataset because if we don’t, then we will get an error when executing the NbClust method

iris$Species=NULL

Now apply the NbClust method as given but first set the seed function to any value so that your result is reproducible.

set.seed(26)

clusterNo=NbClust(iris,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")

“index=all” means test for all 30 indices with the given parameters. You will then get a brief summary of the results as shown below

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Among all indices:

\* 2 proposed 2 as the best number of clusters

\* 13 proposed 3 as the best number of clusters

\* 5 proposed 4 as the best number of clusters

\* 1 proposed 6 as the best number of clusters

\* 2 proposed 10 as the best number of clusters

\*\*\*\*\* Conclusion \*\*\*\*\*

\*According to the majority rule, the best number of clusters is 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Try also

clusterNo=NbClust(iris,distance="manhattan", min.nc=2,max.nc=15,method="kmeans",index="all")

and / or

clusterNo=NbClust(iris,distance="maximum", min.nc=2,max.nc=15,method="kmeans",index="all")

**Elbow Method**

We can also plot the Within Cluster Sum of Squares and the number of clusters to find the location of a bend or a knee in the plot which is considered as an indicator of the appropriate number of clusters.

First, calculate the within sum of squares (withinss) of different numbers of clusters:

k = 2:10

set.seed(42)

WSS = sapply(k, function(k) {kmeans(iris[1:4], centers=k)$tot.withinss})

You can then use a line plot to plot the within sum of squares with a different number of k

plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")

If you wish to have consistency with the variable names, and replicate example 1, here is the equivalent code:

library(NbClust)

library(datasets)

head(iris)

names(iris)

x = iris[,-5]

y = iris$Species

#set.seed(26)

clusterNo=NbClust(x,distance="euclidean", min.nc=2,max.nc=15,method="kmeans",index="all")

#clusterNo=NbClust(iris,distance="manhattan", min.nc=2,max.nc=15,method="kmeans",index="all")

#clusterNo=NbClust(iris,distance="maximum", min.nc=2,max.nc=15,method="kmeans",index="all")

k = 2:10

#set.seed(42)

WSS = sapply(k, function(k) {kmeans(x, centers=k)$tot.withinss})

plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")

**WINE DATASET**

The wine data set contains the results of a chemical analysis of wines grown in a specific area of Italy. Three types of wine are represented in the 178 samples, with the results of 13 chemical analyses recorded for each sample. The Type variable has been transformed into a categorical variable.

install.packages('rattle.data')

library(rattle.data)

library(NbClust)

data(wine, package='rattle.data')

head(wine)

## Type Alcohol Malic Ash Alcalinity Magnesium Phenols Flavanoids

## 1 1 14.23 1.71 2.43 15.6 127 2.80 3.06

## 2 1 13.20 1.78 2.14 11.2 100 2.65 2.76

## 3 1 13.16 2.36 2.67 18.6 101 2.80 3.24

## 4 1 14.37 1.95 2.50 16.8 113 3.85 3.49

## 5 1 13.24 2.59 2.87 21.0 118 2.80 2.69

## 6 1 14.20 1.76 2.45 15.2 112 3.27 3.39

## Nonflavanoids Proanthocyanins Color Hue Dilution Proline

## 1 0.28 2.29 5.64 1.04 3.92 1065

## 2 0.26 1.28 4.38 1.05 3.40 1050

## 3 0.30 2.81 5.68 1.03 3.17 1185

## 4 0.24 2.18 7.80 0.86 3.45 1480

## 5 0.39 1.82 4.32 1.04 2.93 735

## 6 0.34 1.97 6.75 1.05 2.85 1450

**Explore and Pre-processing Data**

Let's see the structure of wine data set

str(wine)

#> 'data.frame':  178 obs. of  14 variables:

#> $ Type           : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...

#> $ Alcohol        : num  14.2 13.2 13.2 14.4 13.2 ...

#> $ Malic          : num  1.71 1.78 2.36 1.95 2.59 1.76 1.87 2.15 1.64 1.35 ...

#> $ Ash            : num  2.43 2.14 2.67 2.5 2.87 2.45 2.45 2.61 2.17 2.27 ...

#> $ Alcalinity     : num  15.6 11.2 18.6 16.8 21 15.2 14.6 17.6 14 16 ...

#> $ Magnesium      : int  127 100 101 113 118 112 96 121 97 98 ...

#> $ Phenols        : num  2.8 2.65 2.8 3.85 2.8 3.27 2.5 2.6 2.8 2.98 ...

#> $ Flavanoids     : num  3.06 2.76 3.24 3.49 2.69 3.39 2.52 2.51 2.98 3.15 ...

#> $ Nonflavanoids  : num  0.28 0.26 0.3 0.24 0.39 0.34 0.3 0.31 0.29 0.22 ...

#> $ Proanthocyanins: num  2.29 1.28 2.81 2.18 1.82 1.97 1.98 1.25 1.98 1.85 ...

#> $ Color          : num  5.64 4.38 5.68 7.8 4.32 6.75 5.25 5.05 5.2 7.22 ...

#> $ Hue            : num  1.04 1.05 1.03 0.86 1.04 1.05 1.02 1.06 1.08 1.01 ...

#> $ Dilution       : num  3.92 3.4 3.17 3.45 2.93 2.85 3.58 3.58 2.85 3.55 ...

#> $ Proline        : int  1065 1050 1185 1480 735 1450 1290 1295 1045 1045 ...

Wine data set contains 1 categorical variables (label) and 13 numerical variables. But these numerical variables is not scaled, we can use **scale** function for scaling and centering data and then assign it as training data.

data.train <- scale(wine[-1])

Data is already centered and scaled.

summary(data.train)

#>   Alcohol             Malic

#> Min.   :-2.42739   Min.   :-1.4290

#> 1st Qu.:-0.78603   1st Qu.:-0.6569

#> Median : 0.06083   Median :-0.4219

#> Mean   : 0.00000   Mean   : 0.0000

#> 3rd Qu.: 0.83378   3rd Qu.: 0.6679

#> Max.   : 2.25341   Max.   : 3.1004

#>      Ash             Alcalinity

#> Min.   :-3.66881   Min.   :-2.663505

#> 1st Qu.:-0.57051   1st Qu.:-0.687199

#> Median :-0.02375   Median : 0.001514

#> Mean   : 0.00000   Mean   : 0.000000

#> 3rd Qu.: 0.69615   3rd Qu.: 0.600395

#> Max.   : 3.14745   Max.   : 3.145637

#>   Magnesium          Phenols

#> Min.   :-2.0824   Min.   :-2.10132

#> 1st Qu.:-0.8221   1st Qu.:-0.88298

#> Median :-0.1219   Median : 0.09569

#> Mean   : 0.0000   Mean   : 0.00000

#> 3rd Qu.: 0.5082   3rd Qu.: 0.80672

#> Max.   : 4.3591   Max.   : 2.53237

#>   Flavanoids      Nonflavanoids

#> Min.   :-1.6912   Min.   :-1.8630

#> 1st Qu.:-0.8252   1st Qu.:-0.7381

#> Median : 0.1059   Median :-0.1756

#> Mean   : 0.0000   Mean   : 0.0000

#> 3rd Qu.: 0.8467   3rd Qu.: 0.6078

#> Max.   : 3.0542   Max.   : 2.3956

#> Proanthocyanins        Color

#> Min.   :-2.06321   Min.   :-1.6297

#> 1st Qu.:-0.59560   1st Qu.:-0.7929

#> Median :-0.06272   Median :-0.1588

#> Mean   : 0.00000   Mean   : 0.0000

#> 3rd Qu.: 0.62741   3rd Qu.: 0.4926

#> Max.   : 3.47527   Max.   : 3.4258

#>      Hue              Dilution

#> Min.   :-2.08884   Min.   :-1.8897

#> 1st Qu.:-0.76540   1st Qu.:-0.9496

#> Median : 0.03303   Median : 0.2371

#> Mean   : 0.00000   Mean   : 0.0000

#> 3rd Qu.: 0.71116   3rd Qu.: 0.7864

#> Max.   : 3.29241   Max.   : 1.9554

#>    Proline

#> Min.   :-1.4890

#> 1st Qu.:-0.7824

#> Median :-0.2331

#> Mean   : 0.0000

#> 3rd Qu.: 0.7561

#> Max.   : 2.9631

**Model Fitting**

We can use NbClust function to determine what the best number of clusteres k is for K-Means

set.seed(1234)

nc <- NbClust(data.train,

              min.nc=2, max.nc=15,

              method="kmeans")

table(nc$Best.n[1,])

0 2 3 8 13 14 15

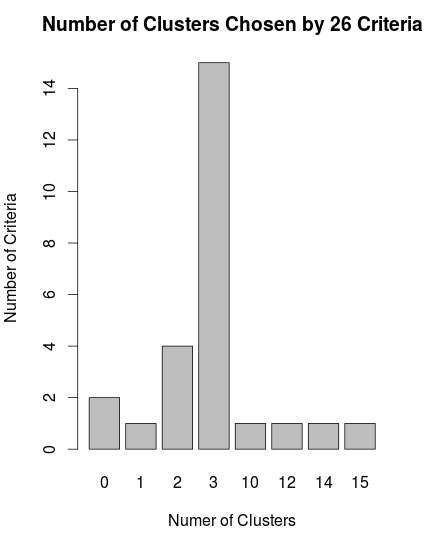
2 3 14 1 2 1 1

barplot(table(nc$Best.n[1,]), # provide bar charts####

        xlab="Numer of Clusters",

        ylab="Number of Criteria",

        main="Number of Clusters Chosen by 30 Criteria")



According to the graph, we can find the best number of clusters is 3. Beside NbClust function which provides 30 indices for determining the number of clusters and proposes the best clustering scheme, we can draw the sum of square error (SSE) scree plot and look for a bend or elbow in this graph to determine appropriate k.

wss <- 0

for (i in 1:15){

  wss[i] <-

    sum(kmeans(data.train, centers=i)$withinss)

}

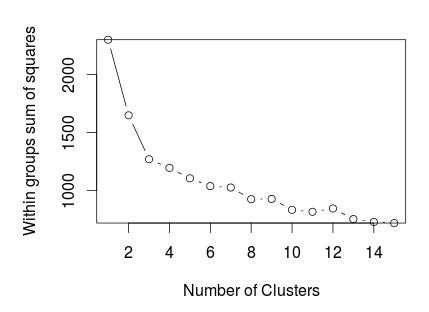
plot(1:15,

  wss,

  type="b", ### "b" for both####

  xlab="Number of Clusters",

  ylab="Within groups sum of squares")



Both two methods suggest k=3 is best choice for us. It's reasonable if we take notice that the original data set also contains 3 classes.

**Fit the model**

We now fit wine data to K-Means with k = 3

set.seed(1234)

fit.km <- kmeans(data.train, 3)

Then interpret the result

fit.km

#> K-means clustering with 3 clusters of sizes 51, 65, 62

#>

#> Cluster means:

#>      Alcohol      Malic        Ash Alcalinity

#> 1  0.1644436  0.8690954  0.1863726  0.5228924

#> 2 -0.9234669 -0.3929331 -0.4931257  0.1701220

#> 3  0.8328826 -0.3029551  0.3636801 -0.6084749

#>     Magnesium     Phenols  Flavanoids Nonflavanoids

#> 1 -0.07526047 -0.97657548 -1.21182921    0.72402116

#> 2 -0.49032869 -0.07576891  0.02075402   -0.03343924

#> 3  0.57596208  0.88274724  0.97506900   -0.56050853

#>   Proanthocyanins      Color        Hue   Dilution

#> 1     -0.77751312  0.9388902 -1.1615122 -1.2887761

#> 2      0.05810161 -0.8993770  0.4605046  0.2700025

#> 3      0.57865427  0.1705823  0.4726504  0.7770551

#>      Proline

#> 1 -0.4059428

#> 2 -0.7517257

#> 3  1.1220202

#>

#> Clustering vector:

#>   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

#>  [26] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

#>  [51] 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 3 2

#>  [76] 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2

#> [101] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 3 2 2 2

#> [126] 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

#> [151] 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

#> [176] 1 1 1

#>

#> Within cluster sum of squares by cluster:

#> [1] 326.3537 558.6971 385.6983

#>  (between\_SS / total\_SS =  44.8 %)

#>

#> Available components:

#>

#> [1] "cluster"      "centers"      "totss"

#> [4] "withinss"     "tot.withinss" "betweenss"

#> [7] "size"         "iter"         "ifault"

Or, we can check specific information

fit.km$centers

Alcohol Malic Ash Alcalinity Magnesium Phenols Flavanoids Nonflavanoids

1 0.83 -0.30 0.36 -0.61 0.576 0.883 0.975 -0.561

2 -0.92 -0.39 -0.49 0.17 -0.490 -0.076 0.021 -0.033

3 0.16 0.87 0.19 0.52 -0.075 -0.977 -1.212 0.724

Proanthocyanins Color Hue Dilution Proline

1 0.579 0.17 0.47 0.78 1.12

2 0.058 -0.90 0.46 0.27 -0.75

3 -0.778 0.94 -1.16 -1.29 -0.41

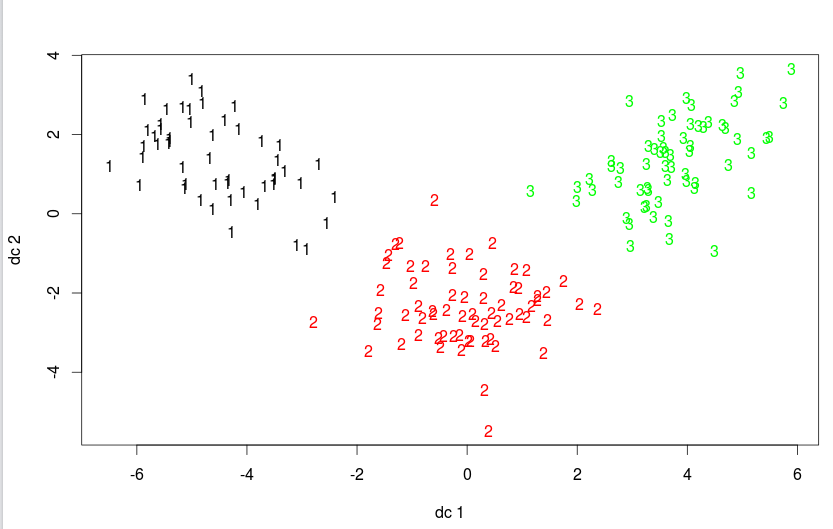
Or

fit.km$size

The result shows information about cluster means, clustering vector, sum of square by cluster and available components. Let's do some visualization to see how data set is clustered. First, we use plotcluster function from fpc package to draw discriminant projection plot

library(fpc)

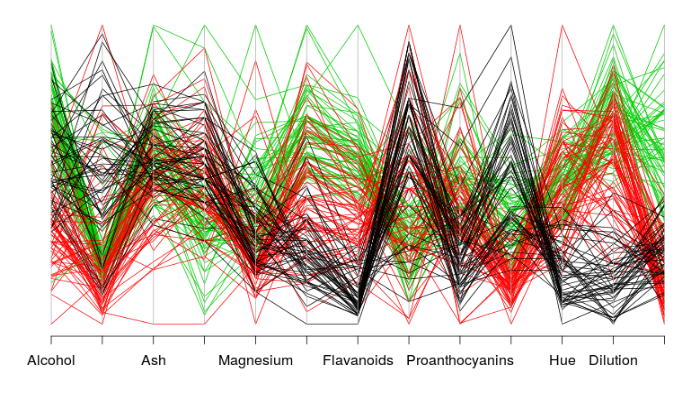
plotcluster(data.train, fit.km$cluster)



We can see the data is clustered very well, there are no collapse between clusters. Next, we draw parallel coordinates plot to see how variables contributed in each cluster

library(MASS)

parcoord(data.train, fit.km$cluster)



We can extract some insights from above graph as black cluster contains wine with low flavanoids value, low proanthocyanins value, low hue value. Or green cluster contains wine which has dilution value higher than wine in red cluster.

**Evaluation**

Because the original data set wine also has 3 classes, it is reasonable if we compare these classes with 3 clusters fitted by K-Means

confuseTable.km <- table(wine$Type, fit.km$cluster)

confuseTable.km

#>    1  2  3

#> 1  0  0 59

#> 2  3 65  3

#> 3 48  0 0

We can see only 6 sample is missed. Let's use **randIndex** from **flexclust** to compare these two parititions – one from data set and one from result of clustering method.

library(flexclust)

randIndex(confuseTable.km)

#>      ARI

#> 0.897495

The adjusted Rand index provides a measure of the agreement between two partitions, adjusted for chance. It ranges from -1 (no agreement) to 1 (perfect agreement). Agreement between the wine varietal type and the cluster solution is 0.9. Not bad—shall we have some wine?

A few words, related to data pre-processing steps:

**Why normalize or scale the data?**

There can be instances found in data frame where values for one feature could be in the range 1-100, and values for other feature in the range 1-10000000. In scenarios like these, owing to mere greater numeric range, the impact on response variables by the feature having greater numeric range could be more than the one having less numeric range, and this could, in turn, impact prediction accuracy. The objective is to improve predictive accuracy and not allow a particular feature impact the prediction due to large numeric value range. Thus, we may need to normalize or scale values under different features such that they fall under common range. Take a look at following example:

# Age vector

age <- c(25, 35, 50)

# Salary vector

salary <- c(200000, 1200000, 2000000)

# Data frame created using age and salary

df <- data.frame( "Age" = age, "Salary" = salary, stringsAsFactors = FALSE)

#Age  Salary

 # 25  200000

 # 35 1200000

 # 50 2000000

Pay attention on how values for age and salary vary in different ranges.

**Min-Max Normalization**

Data frame could be normalized using Min-Max normalization technique which is specified by the following formula to be applied on each value of features to be normalized. This technique is traditionally used with K-Nearest Neighbors (KNN) classification problems.

(X - min(X))/(max(X) - min(X))

This is equivalent to the following R function

normalize <- function(x) {

    return ((x - min(x)) / (max(x) - min(x)))

  }

In order to apply the normalize function on each of the features of the above data frame (df), the following code could be used. Pay attention to usage of lapply function (check R documentation for further info).

dfNorm <- as.data.frame(lapply(df, normalize))

# One could also use sequence such as df[1:2]

dfNorm <- as.data.frame(lapply(df[1:2], normalize))

In case, one wish to specify a set of features such as salary, following formula could be used:

# Note df[2]

dfNorm <- as.data.frame(lapply(df[2], normalize))

# or df["Salary"]

dfNorm <- as.data.frame(lapply(df["Salary"], normalize))

**Z-Score Standardization**

The disadvantage with min-max normalization technique is that it tends to bring data towards the mean. If there is a need for outliers to get weighted more than the other values, z-score standardization technique suits better. In order to achieve z-score standardization, one could use **R’s built-in scale()** function. Take a look at following example where scale function is applied on “df” data frame mentioned above.

dfNormZ <- as.data.frame( scale(df[1:2] ))

The output is given as:

         Age      Salary

1 -0.9271726 -1.03490978

2 -0.1324532  0.07392213

3  1.0596259  0.96098765

**Confusion matrix function**

**From caret package**

example

confusionMatrix(test\_pred, testing$V1 )

Confusion Matrix and Statistics

Reference

Prediction 1 2 3

1 15 0 0

2 0 22 0

3 0 2 14

Overall Statistics

Accuracy : 0.9623

95% CI : (0.8702, 0.9954)

No Information Rate : 0.4528

P-Value [Acc > NIR] : 1.208e-15

Kappa : 0.9421

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: 1 Class: 2 Class: 3

Sensitivity 1.000 0.9167 1.0000

Specificity 1.000 1.0000 0.9487

Pos Pred Value 1.000 1.0000 0.8750

Neg Pred Value 1.000 0.9355 1.0000

Prevalence 0.283 0.4528 0.2642

Detection Rate 0.283 0.4151 0.2642

Detection Prevalence 0.283 0.4151 0.3019

Balanced Accuracy 1.000 0.9583 0.9744