R Notebook

Code

C3283188 - Patrick Jansson

COMP3340 - Data Mining Assignment 3

Reading data from regression dataset of choice

Hide

```
energy <- read.xlsx("../Datasets/Regression Datasets/EnergyEfficiency.xlsx", 1, header = TRUE
)
```

Reading data from classification dataset of choice

Hide

```
iris <- read.csv("../Datasets/Classification Datasets/Iris.csv", stringsAsFactors = TRUE)
iris$Species <- factor(iris$Species, levels = c("Iris-setosa", "Iris-versicolor", "Iris-virgi
nica"))
```

Exercise 1

Hide

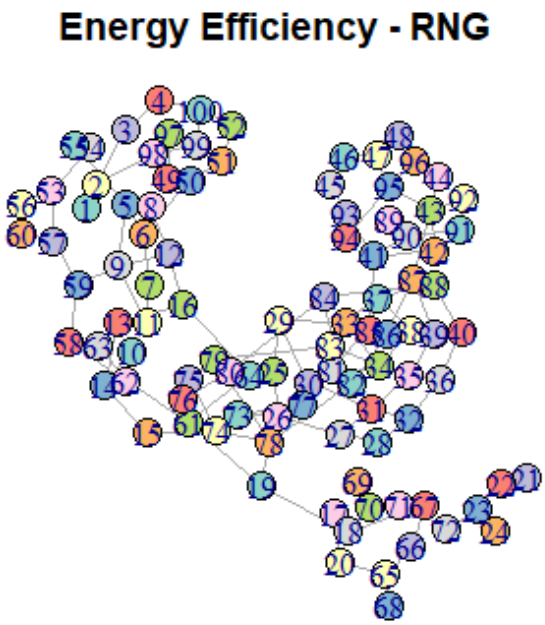
```
pal3 <- brewer.pal(9, "Set3")

energy.preds <- data.matrix(subset(energy[1:100,], select = -c(NA.)))

energy.distance <- as.matrix(Dist(energy.preds, "euclidean"))
```

Hide

```
energy.completeGraph <- generate.complete.graph(1:nrow(energy.preds), energy.distance)
```

Hide

```
energy.rng <- rng(dx=energy.distance)
plot(energy.rng, vertex.color=pal3, vertex.size=12, layout=layout_with_dh, main="Energy Effic
iency - RNG")
```



Exercise 2

Hide

```
churn <- read.delim("../Datasets/Classification Datasets/Customer Churn.txt", header=TRUE, se
p = ",", stringsAsFactors = TRUE)
churn <- churn[complete.cases(churn),]
sample = sample.split(churn, SplitRatio = .75)
```

Training data set

Hide

```
train = subset(churn, sample == TRUE)
```

Test data set

Hide

```
test  = subset(churn, sample == FALSE)
```

Training and test data were split up at a ratio of 3:1 in favor of the training set randomly. Full dataset was constrained to 100 entries to account for computational power.

Hide

```
ctrl.churn <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   repeats = 5)

rfe.churn <- rfe(x=train[1:100,!names(train) %in% c("Churn.", "Phone")], y=train$Churn.[1:100
],
                 sizes = c(3:8, 10, 15),
                 rfeControl = ctrl.churn)

rfe.churn
```

```
Recursive feature selection

Outer resampling method: Cross-Validated (10 fold, repeated 5 times)

Resampling performance over subset size:
```

	Variables <S3: AsIs>	Accuracy <S3: AsIs>	Kappa <S3: AsIs>	AccuracySD <S3: AsIs>	KappaSD <S3: AsIs>	Selected <S3: AsIs>
1	3	0.8433	0.046149	0.07881	0.2660	
2	4	0.8393	0.068626	0.07865	0.2623	
3	5	0.8315	0.015670	0.08530	0.2323	
4	6	0.8457	0.045454	0.07926	0.2504	
5	7	0.8537	0.041699	0.06976	0.2315	*
6	8	0.8493	0.035719	0.06885	0.2338	
7	10	0.8478	0.017508	0.06771	0.1869	
8	15	0.8472	0.007558	0.06739	0.1649	
9	19	0.8286	0.036763	0.09108	0.2693	

9 rows

```
The top 5 variables (out of 7):
   Day.Charge, Day.Mins, Eve.Charge, Eve.Mins, Intl.Mins
```

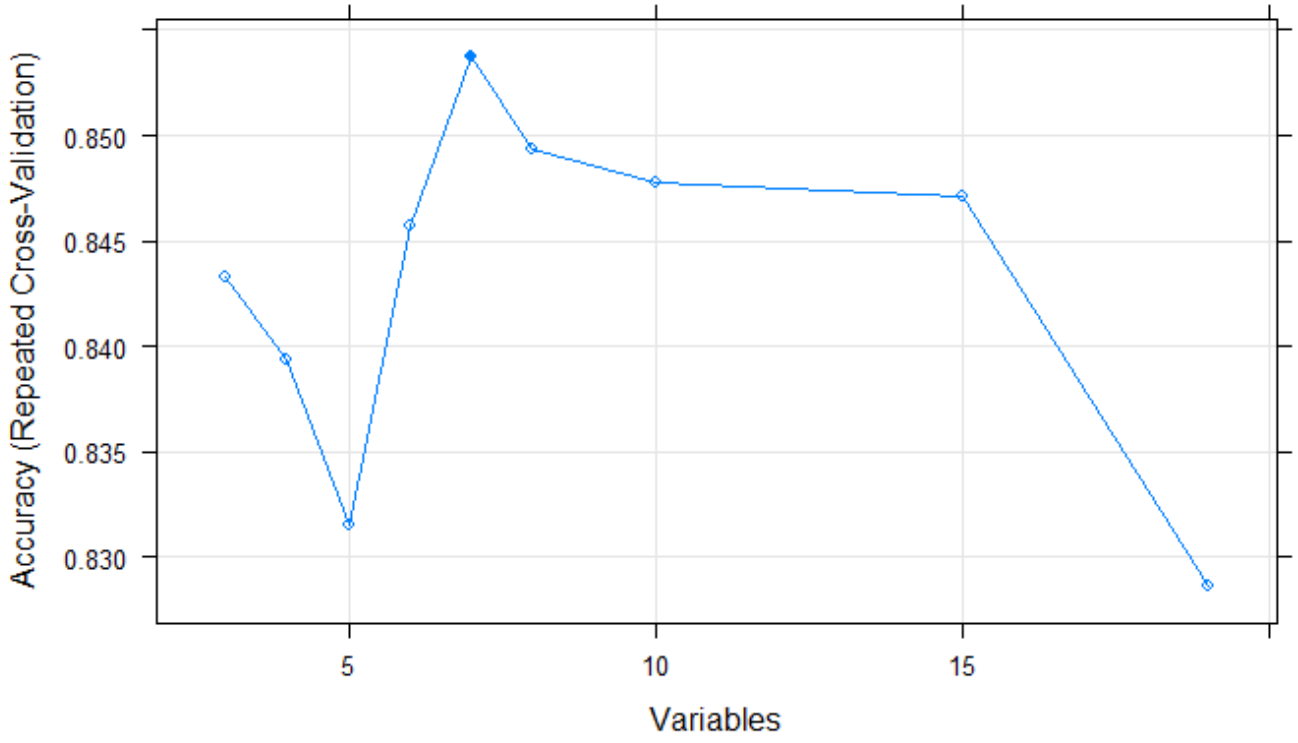
Hide

```
predictors(rfe.churn)
```

```
[1] "Day.Charge"  "Day.Mins"    "Eve.Charge"  "Eve.Mins"    "Intl.Mins"   "Intl.Charge" "Nigh
t.Mins"
```

Hide

```
plot(rfe.churn, type = c("g", "o"))
```



b. Random forest model

Hide

```
rfe.churn$fit
```

```
Call:
 randomForest(x = x, y = y, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 14%
Confusion matrix:
       False. True. class.error
False.     85     2  0.02298851
True.      12     1  0.92307692
```

Now, we predict samples in the test set, and compare them to their actual classes.

Hide

```
churn.preds <-predict(rfe.churn$fit, test[,!names(test) %in% c("Churn")])

conf_matrix.churn <-confusionMatrix(churn.preds,as.factor(test$Churn))
conf_matrix.churn
```

```
Confusion Matrix and Statistics

          Reference
Prediction False. True.
    False.    788   121
    True.      20    23

               Accuracy : 0.8519
                 95% CI : (0.8277, 0.8739)
    No Information Rate : 0.8487
    P-Value [Acc > NIR] : 0.4145

                  Kappa : 0.1896

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9752
            Specificity : 0.1597
         Pos Pred Value : 0.8669
         Neg Pred Value : 0.5349
             Prevalence : 0.8487
         Detection Rate : 0.8277
   Detection Prevalence : 0.9548
      Balanced Accuracy : 0.5675

       'Positive' Class : False.
```

Classifier has an accuracy of 86%. High sensitivity of 97%, however a very low specificity rate of 16%, meaning it has a very low false positive rate.

Hide

```
conf_matrix.churn$byClass["F1"]
```

```
     F1
0.91788
```

The F1 score is a balance between Precision and Recall.

Precision is the ratio of correct predictions in the all the positive predicted observations. Recall is the ratio of correctly predicted positive observation in the whole positive classs.

Hide

```
mcc(churn.preds, test$Churn)
```

```
[1] 0.2328666
```

Matthew's Correlation Coefficient is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between −1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and −1 indicates total disagreement between prediction and observation. To get a good score, a classifier must get a good score in all 4 confusion matrix categories. This classifier does not have a fantastic mcc.

Hide

```
youden<- conf_matrix.churn$byClass["Sensitivity"] + conf_matrix.churn$byClass["Specificity"]
 -1
youden
```

```
Sensitivity
  0.1349697
```

Youden's J statistic has a range through 0 to 1, with 1 meaning all values were predicted correctly. As seen with my number, Youden's J statistic can be negative but it is said to be between 0 and 1 where positives and negatives are the number of real positive and real negative samples.

Exercise 3

- **Input:** A set *X* of *m* examples, linear binary array of ***n*** features and a binary label assigned to each of them, positive integer *k > 0.*
- **Question:** Is there a subset of features S such that:
  - *S ⊆ { 1, …, n }*
  - *|S| = k,*
  - No pair of examples in X have the same values for the features in S ***but have different values for the binary label characteristic.***

Exercise 4

A *string* is a concatenation of symbols of a given *alphabet*. Let Σ be one such alphabet. We define a *pattern* as a string *s* over an *extended alphabet* that now includes the 'wild card' symbol and we write $\Sigma* := \Sigma \cup *$.

Exercise 5

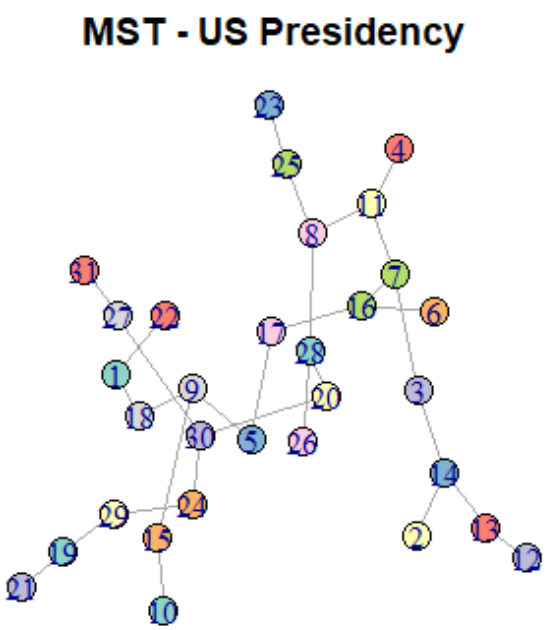
Hide

```
us <- read.csv("../Datasets/Classification Datasets/USPresidency.csv")
us.preds <- data.matrix(subset(us[1:31,], select = -c(Year)))
us.distance <- as.matrix(Dist(us.preds, "euclidean"))
us.completeGraph <- generate.complete.graph(1:nrow(us.preds), us.distance)
us.mst <- generate.mst(us.completeGraph)
```

a.

Hide

```
plot(us.mst$mst.graph, vertex.color=pal3, vertex.size=12, layout=layout_with_dh, main="MST -
 US Presidency")
```



b & d)

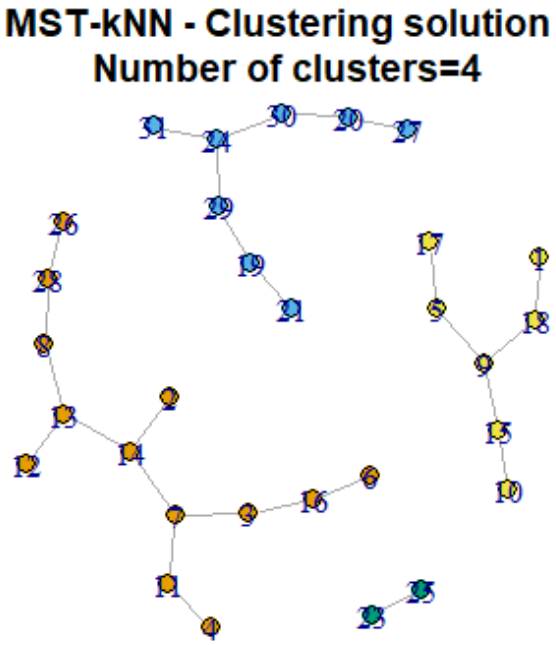
Hide

```
results <- mst.knn(us.distance)
```

```
 Only there is  30 nodes in solutions. Clustering solution only will have these nodes.
```

Hide

```
library("igraph")
plot(results$network, vertex.size=8,
     vertex.color=igraph::clusters(results$network)$membership,
     layout=igraph::layout.fruchterman.reingold(results$network, niter=10000),
     main=paste("MST-kNN - Clustering solution \n Number of clusters=",results$cnumber,sep=""
))
```



c.

- Starting off with a complete weighted *G(V,E,C)* where:
  - *V*: set of *n* vertices in the graph (one for each element)
  - *E*: set of edges. One fro each pair of elements *(i,j).*
  - *C*: set of edges' cost. Represents the distance between *i* and *j*.
- Minimum Spanning Tree (MST)
  - $G_{MST}(V,E_{MST},C_{MST})$
- *k*-Nearest Neighbors (*kNN*)
  - $G_{kNN}(V,E_{kNN},C_{kNN})$
- To calculate the edges in both the MST and the k-NN, we need to produce a partition of the graph vertices and identify a forest (set of subtrees of the complete weighted graph; these are our clusters).
- So we get a new graph:
  - $G_{CLUSTER}(V, E_{CLUSTER},C_{CLUSTER})$, with:
    - $G_{CLUSTER} = E_{MST}\ E_{kNN}$
    - Dynamic/adaptive value of *k*

Exercise 6

Hide

```
us <- read.csv("../Datasets/Classification Datasets/USPresidency.csv", row.names = "Year")
km.res <- kmeans(us, 4, nstart = 1)
print(km.res)
```

```
K-means clustering with 4 clusters of sizes 12, 7, 6, 6

Cluster means:
         Q1        Q2        Q3        Q4        Q5        Q6        Q7        Q8         Q9
Q10       Q11        Q12    Target
1 0.6666667 1.0000000 0.1666667 0.0000000 0.7500000 0.2500000 0.8333333 0.5833333 0.08333333
0.08333333 0.5833333 0.08333333 1.0000000
2 0.8571429 0.7142857 0.2857143 0.7142857 0.5714286 0.5714286 0.2857143 0.0000000 0.14285714
0.57142857 0.0000000 0.57142857 0.0000000
3 0.8333333 0.1666667 0.3333333 1.0000000 0.0000000 0.3333333 0.6666667 0.6666667 0.83333333
0.00000000 0.3333333 0.00000000 0.1666667
4 0.0000000 0.0000000 0.1666667 0.0000000 1.0000000 0.0000000 0.1666667 0.6666667 0.50000000
0.00000000 0.0000000 0.16666667 0.8333333

Clustering vector:
1864 1868 1872 1880 1888 1900 1904 1908 1916 1924 1928 1936 1940 1944 1948 1956 1964 1972 186
0 1876 1884 1892 1896 1912 1920 1932 1952 1960 1968 1976 1980
   4    1    1    3    4    1    1    1    4    1    1    1    1    1    1    1    4    4
3    2    3    4    3    2    3    2    2    2    3    2    2

Within cluster sum of squares by cluster:
[1] 19.083333 14.285714 10.000000  6.166667
 (between_SS / total_SS =  42.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
"size"         "iter"         "ifault"
```

Hide

```
aggregate(us, by=list(cluster=km.res$cluster), mean)
```

cluster <int>	Q1 <dbl>	Q2 <dbl>	Q3 <dbl>	Q4 <dbl>	Q5 <dbl>	Q6 <dbl>	Q7 <dbl>	<
1	0.6666667	1.0000000	0.1666667	0.0000000	0.7500000	0.2500000	0.8333333	0.583
2	0.8571429	0.7142857	0.2857143	0.7142857	0.5714286	0.5714286	0.2857143	0.000
3	0.8333333	0.1666667	0.3333333	1.0000000	0.0000000	0.3333333	0.6666667	0.666
4	0.0000000	0.0000000	0.1666667	0.0000000	1.0000000	0.0000000	0.1666667	0.666

4 rows | 1-9 of 14 columns

Hide

```
NA
```

a. An inter-rater reliability method is the degree of agreement among independent observers who rate, code, or assess the same phenomenon. Some examples of these are, if there is only two raters, the **Scott's Pi** or the **Cohen's Kappa**. In the case of *more than two raters* the **Fleiss' Kappa**, which is based on Scott's Pi, is recommended.

b. Scott's Pi or Cohen's Kappa as there are only two raters and as such, Fleiss' Kappa would not be recommended.

c.

Exercise 7

a. Lazy Classification: Lazy learners simply store the training data and wait until a testing data appear. When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting. Class Imbalance: A frequent problem in labelled datasets, such as binary and multiclass tasks. Can affect predictive performance of most ML algorithms. Often refers to the class distribution is not equal or close to equal, an is instead biased or skewed.

b. Churn Dataset

Hide

```
conf_matrix.churn <-confusionMatrix(churn.preds,as.factor(test$Churn))
conf_matrix.churn
```

```
Confusion Matrix and Statistics

          Reference
Prediction False. True.
    False.    788   121
    True.      20    23

               Accuracy : 0.8519
                 95% CI : (0.8277, 0.8739)
    No Information Rate : 0.8487
    P-Value [Acc > NIR] : 0.4145

                  Kappa : 0.1896

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9752
            Specificity : 0.1597
         Pos Pred Value : 0.8669
         Neg Pred Value : 0.5349
             Prevalence : 0.8487
         Detection Rate : 0.8277
   Detection Prevalence : 0.9548
      Balanced Accuracy : 0.5675

       'Positive' Class : False.
```

High accuracy and very high sensitivity, showing the confusion matrix is quite good at knowing when the true is meant to be true. Low specificity however, meaning it's not vey accuracy regarding false's being false.

Iris Dataset

Hide

```
iris <- read.csv("../Datasets/Classification Datasets/Iris.csv", stringsAsFactors = TRUE)
iris <- iris[complete.cases(iris),]
sample.iris = sample.split(iris, SplitRatio = .75)
train.iris = subset(iris, sample.iris == TRUE)
test.iris  = subset(iris, sample.iris == FALSE)

ctrl.iris <- rfeControl(functions = rfFuncs,
                   method = "repeatedcv",
                   repeats = 5)

rfe.iris <- rfe(x=train.iris[1:100,!names(train.iris) %in% c("Species", "Id")], y=train.iris
$Species[1:100],
                 sizes = c(5, 10, 15, 20),
                 rfeControl = ctrl.iris)

iris.preds <-predict(rfe.iris$fit, test.iris[,!names(test.iris) %in% c("Species")])

conf_matrix.iris <-confusionMatrix(iris.preds,as.factor(test.iris$Species))
conf_matrix.iris
```

```
Confusion Matrix and Statistics

                 Reference
Prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              17               0              0
  Iris-versicolor           0              17              2
  Iris-virginica            0               0             14

Overall Statistics

               Accuracy : 0.96
                 95% CI : (0.8629, 0.9951)
    No Information Rate : 0.34
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9399

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Iris-setosa Class: Iris-versicolor Class: Iris-virginica
Sensitivity                        1.00                1.0000               0.8750
Specificity                        1.00                0.9394               1.0000
Pos Pred Value                     1.00                0.8947               1.0000
Neg Pred Value                     1.00                1.0000               0.9444
Prevalence                         0.34                0.3400               0.3200
Detection Rate                     0.34                0.3400               0.2800
Detection Prevalence               0.34                0.3800               0.2800
Balanced Accuracy                  1.00                0.9697               0.9375
```

Very high accuracy, very high overall sensitivity and specificity. This confusion matrix shows how we can achieve very good results.

Example

Hide

```
expected_value <- factor(c(1,1,1,0,1,1,0,1,0,1))
predicted_value <- factor(c(1,0,1,1,0,1,1,0,0,1))

example <- confusionMatrix(data=predicted_value, reference = expected_value)
example
```

```
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 1 3
         1 2 4

               Accuracy : 0.5
                 95% CI : (0.1871, 0.8129)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.9527

                  Kappa : -0.087

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.3333
            Specificity : 0.5714
         Pos Pred Value : 0.2500
         Neg Pred Value : 0.6667
             Prevalence : 0.3000
         Detection Rate : 0.1000
   Detection Prevalence : 0.4000
      Balanced Accuracy : 0.4524

       'Positive' Class : 0
```

This confusion matrix shows us how not to do it. Low sensitivity, specificity and accuracy all mean the predicted values were very off.

Matthew’s Correlation Coefficient is in essence a correlation coefficient between the observed and predicted binary classifications; it returns a value between −1 and +1. A coefficient of +1 represents a perfect prediction, 0 no better than random prediction and −1 indicates total disagreement between prediction and observation. To get a good score, a classifier must get a good score in all 4 confusion matrix categories. 1. Churn Dataset

Hide

```
mcc(churn.preds, test$Churn)
```

```
[1] 0.2328666
```

This dataset doesn’t have a very good mcc

2. Iris Dataset

Hide

```
mcc(iris.preds, test.iris$Species)
```

```
[1] 0.9421748
```

This dataset has a very good mcc

3. Example Dataset

Hide

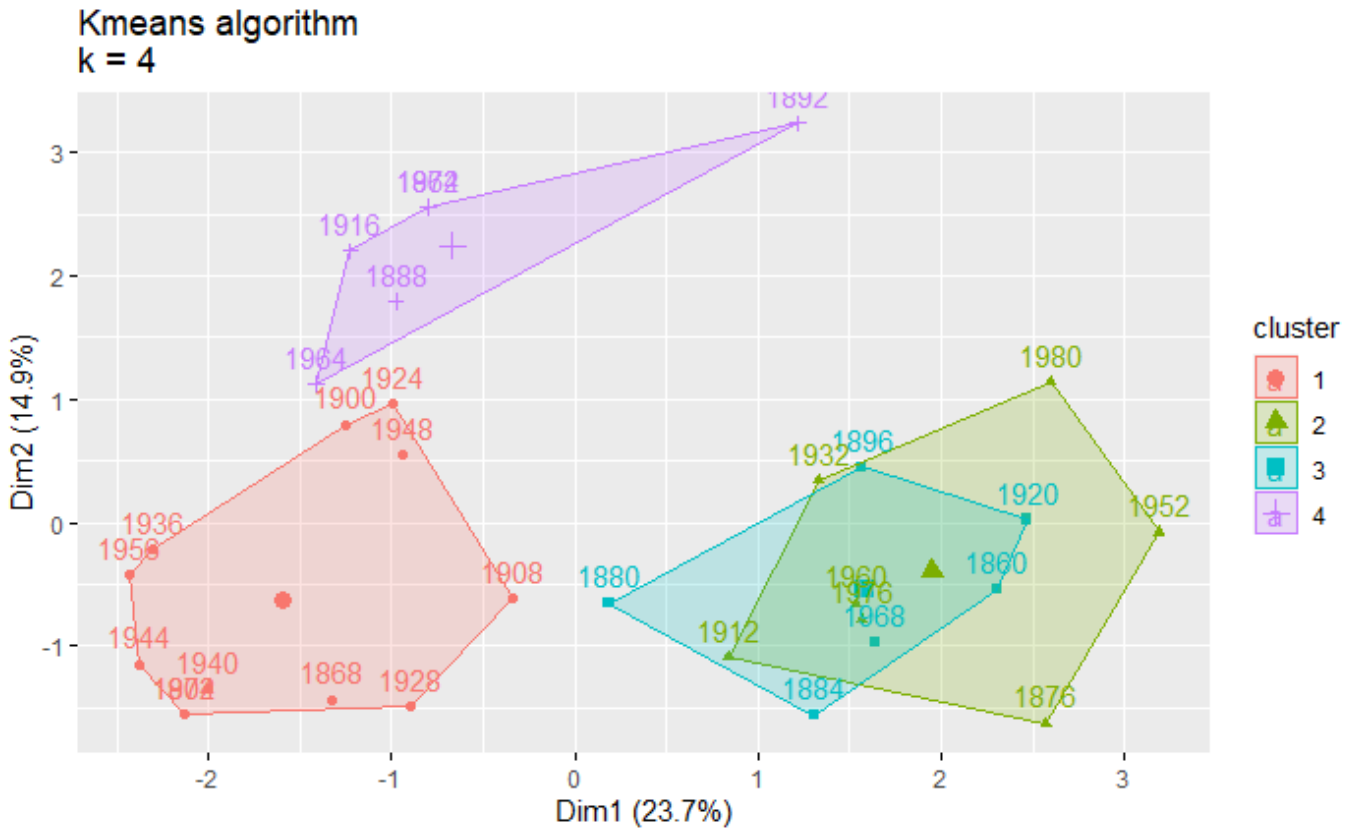
```
mcc(predicted_value, expected_value)
```

```
[1] -0.08908708
```

This dataset has a quite good mcc ## Exercise 8

Hide

```
p3 <- fviz_cluster(km.res,  data = us) + ggtitle("Kmeans algorithm\nk = 4")
p3
```



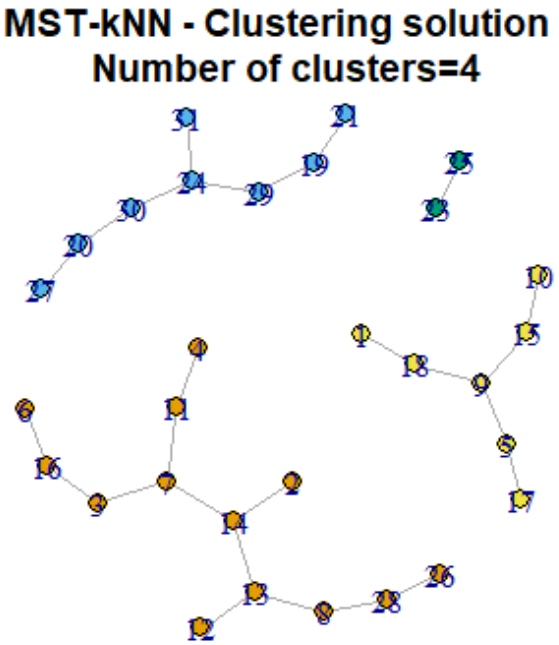
Hide

```
results <- mst.knn(us.distance)
```

```
 Only there is  30 nodes in solutions. Clustering solution only will have these nodes.
```

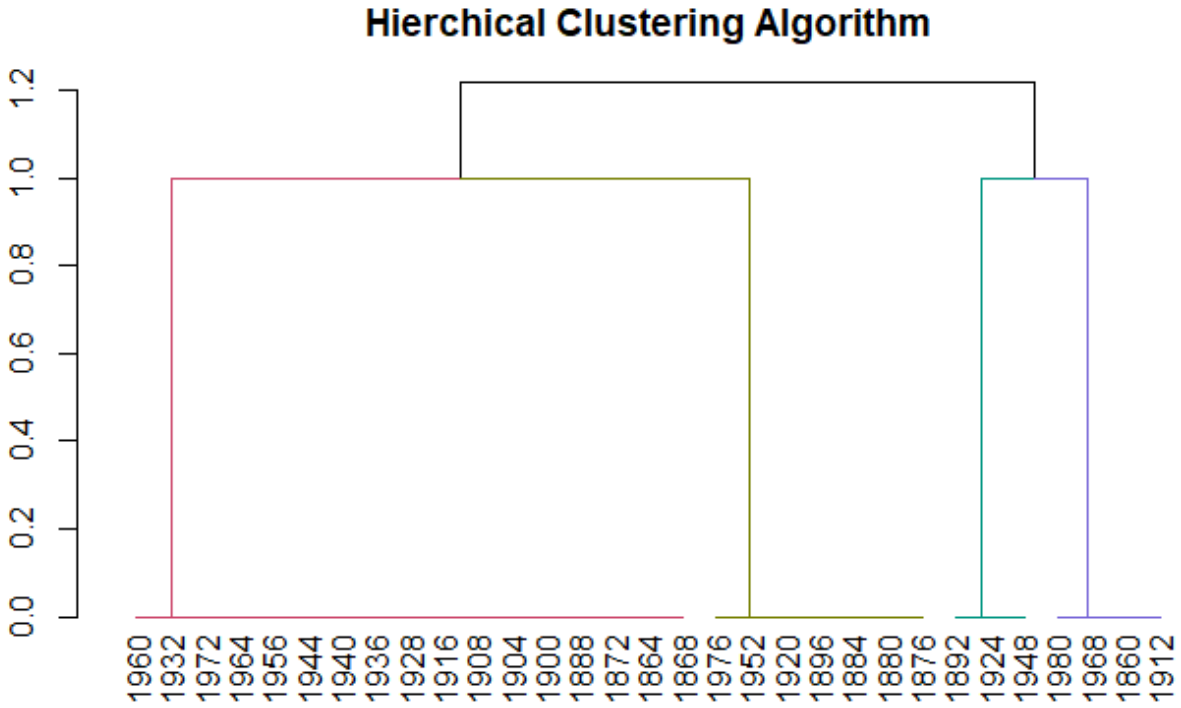
Hide

```
library("igraph")
plot(results$network, vertex.size=8,
     vertex.color=igraph::clusters(results$network)$membership,
     layout=igraph::layout.fruchterman.reingold(results$network, niter=10000),
     main=paste("MST-kNN - Clustering solution \n Number of clusters=",results$cnumber,sep=""
))
```



Hide

```
clusters <- hclust(dist(us[, 3:4]), method = 'average')
suppressPackageStartupMessages(library(dendextend))
avg_dend_obj <- as.dendrogram(clusters)
avg_col_dend <- color_branches(avg_dend_obj, h = 0.5)
plot(avg_col_dend, main="Hierchical Clustering Algorithm")
```



Exercise 9

a. Did not complete

b. Association rules are given in the form as below:

$$A => B[Support, Confidence]$$

The part before => is referred to as *if (Antecedent)* and the part after => is referred to as *then (Consequent).* For a Rule A=>B, Support is given by:

$$Support(A => B) = \frac{frequency(A, B)}{N}$$

For a rule A=>B Confidence shows the percentage in which B is bought with A.

$$Confidence(A => B) = \frac{P(A \cap B)}{P(A)} = \frac{frequency(A, B)}{frequency(A)}$$

Support and Confidence measure how interesting the rule is. It is set by the minimum support and minimum confidence thresholds.If a rule A=>B[Support, Confidence] satisfies min_sup and min_confidence then it is a strong rule. When you apply Association Rule Mining on a given set of transactions T your goal will be to find all rules with:

1. Support greater than or equal to min_support
2. Confidence greater than or equal to min_confidence
   So finding association rules for the US Presidency dataset using the support threshold to 60% would prune rules where the support fails to meet those thresholds.

Exercise 10

Did not complete

Exercise 11

Half working

Hide

```
library(rpart,quietly = TRUE)
```

```
Attaching package: ‘rpart’

The following object is masked from ‘package:dendextend’:

    prune

The following object is masked from ‘package:cccd’:

    prune
```

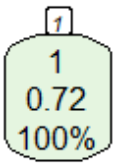
Hide

```
library(caret,quietly = TRUE)
library(rpart.plot,quietly = TRUE)
library(rattle)
```

```
Loading required package: bitops

Attaching package: ‘bitops’

The following object is masked from ‘package:Matrix’:

    %&%

Rattle: A free graphical interface for data science with R.
Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
Type 'rattle()' to shake, rattle, and roll your data.

Attaching package: ‘rattle’

The following object is masked from ‘package:randomForest’:

    importance
```

Hide

```
us <- read.csv("../Datasets/Classification Datasets/USPresidency.csv")
us.preds <- data.matrix(subset(us[1:31,], select = -c(Year)))


set.seed(12345)
train <- sample(1:nrow(us.preds),size = ceiling(0.80*nrow(us.preds)),replace = FALSE)
us_train <- us.preds[train,]
us_test <- us.preds[-train,]

penalty.matrix <- matrix(c(0,1,10,0), byrow=TRUE, nrow=2)

us_train <- as.data.frame(us_train)

tree <- rpart(Q1~.,
data=us_train,
parms = list(loss = penalty.matrix),
method = "class")

rpart.plot(tree, nn=TRUE)
```



Exercise 12

a. **Cross-validation**
   A resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation
b. **Bootstrapping**
   A type of resampling where large numbers of smaller samples of the same size are repeatedly drawn, with replacement, from a single original sample.
c. **Imputation**
   The process of replacing missing data with substituted values. When substituting for a data point, it is

known as “unit imputation”; when substituting for a component of a data point, it is known as “item imputation”.