COMP3340_A3

Leala Darby

08/11/2020

First import the required libraries:

```
library("reticulate")# for incorporating Python code
library("png")
library("grid")
#library("RWeka")
library("scatterplot3d")
use_virtualenv("r-reticulate")
#virtualenv_install("r-reticulate", packages=c("scikit-learn"))
```

Exercise 1

The regression dataset chosen is the Yacht Hydrodynamics dataset sourced from the UCI Machine Learning Repository on the 7th of November, with the .data file downloaded and saved as yacht_hydrodynamics.data in the working directory. The response variable will be removed. There are 6 remaining features, and 308 samples in total.

```
import pandas as pd
import numpy as np
import math
import networkx as nx
yacht_data = pd.read_csv("yacht_hydrodynamics.data", delim_whitespace=True)
# Remove the response variable
yacht_data = yacht_data.drop("7", axis = 1)
# # Define node names by row
i_names = [str(i) for i in yacht_data.index.values]
yacht_data.head()
```

```
##      1      2     3     4     5      6
## 0 -2.3  0.568  4.78  3.99  3.17  0.125
## 1 -2.3  0.568  4.78  3.99  3.17  0.150
## 2 -2.3  0.568  4.78  3.99  3.17  0.175
## 3 -2.3  0.568  4.78  3.99  3.17  0.200
## 4 -2.3  0.568  4.78  3.99  3.17  0.225
```

The variables are continuous and of different scales. Before defining the Euclidean Distance between points, the values will be standardised using a z-score transformation.

```
# normalise the data
norm_yacht_data = (yacht_data - yacht_data.mean())/yacht_data.std()
def euclid_distance(s1, s2):
    return math.sqrt(np.sum((s1-s2) ** 2))
# Distance matrix for samples:
samples_edist_m = [[euclid_distance(norm_yacht_data.iloc[y],
  norm_yacht_data.iloc[x]) for y in range(len(i_names))] for x in range(len(i_names))]
round(pd.DataFrame(samples_edist_m, columns = i_names, index = i_names), 3).head()
```

```
##        0      1      2      3      4  ...    303    304    305    306    307
## 0  0.000  0.248  0.495  0.743  0.991  ...  3.636  3.793  3.959  4.133  4.315
## 1  0.248  0.000  0.248  0.495  0.743  ...  3.489  3.636  3.793  3.959  4.133
## 2  0.495  0.248  0.000  0.248  0.495  ...  3.355  3.489  3.636  3.793  3.959
## 3  0.743  0.495  0.248  0.000  0.248  ...  3.234  3.355  3.489  3.636  3.793
## 4  0.991  0.743  0.495  0.248  0.000  ...  3.128  3.234  3.355  3.489  3.636
##
## [5 rows x 308 columns]
```

Defining an `Edge` class to be used throughout relevant exercises:

```
class Edge:
    def __init__(self, s, w, e):
        self.start = s
        self.weight = w
        self.end = e
    def get_edge(self):
        return (str(self.start) + ’-’ + str(self.weight) + ’-’ + str(self.end))
    def get_nx_edge(self):
        return ((self.start, self.end, {’label’: str(self.weight)}))
```

Next, defining code to compute the RNG.

```
# Function to generate Relative Neighbourhood Graph
def relative_neighbourhood_graph(G):
    V = [i for i in range(len(G))]
    RNG = []
    for u in V: # start point
        for v in V: # end point
            dist = G[u][v]
            if (dist != 0):
                for r in V: # third point
                    if ((r != u) & (r != v)):
                        d_r_u = G[u][r]
                        d_r_v = G[v][r]
                        if ((d_r_u < dist) & (d_r_v < dist)):
                            break
                else:
                    RNG.append([u,v])
    return RNG
# Function to generate .gml for Relative Neighbourhood Graph
def get_RNG(matrix, names, output_file):
    E = relative_neighbourhood_graph(matrix)
    V = [i for i in range(len(matrix))]
    RNG = []
    node_labels = [names[n] for n in V]
    for e in range(1, len(E)):
        RNG.append(Edge(node_labels[E[e][0]], matrix[E[e][0]][E[e][1]], node_labels[E[e][1]]))
    G = nx.Graph()
    G.add_nodes_from(node_labels)
    G.add_edges_from([e.get_nx_edge() for e in RNG])
    nx.write_gml(G, output_file)
# Generate Relative Neighbourhood Graph for samples:
get_RNG(samples_edist_m, i_names, 'graph_1.gml')
```

The output file graph_1.gml was processed using yEd Graph Editor to produce the following visualisation in `Figure 1`:

```
graph_1 <- readPNG("graph_1.png")
grid.raster(graph_1)
```

Exercise 2

- “The classification problem could be of binary type” - does that mean it must be? I would like to use the Iris dataset, which has >2 classes in the response. The classification dataset to be used is the iris dataset from sklearn.datasets. It has four continuous features, a response variable with three classes, and 150 samples in total.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
iris_data = pd.DataFrame(data= np.c_[iris['target'], iris['data']],
                     columns= ['CLASS'] + iris['feature_names'])
print(iris_data.head())
```

```
##    CLASS  sepal length (cm)  ...  petal length (cm)  petal width (cm)
## 0    0.0                5.1  ...                1.4               0.2
## 1    0.0                4.9  ...                1.4               0.2
## 2    0.0                4.7  ...                1.3               0.2
## 3    0.0                4.6  ...                1.5               0.2
## 4    0.0                5.0  ...                1.4               0.2
##
## [5 rows x 5 columns]
```

Of the 150 samples, 80% (120 rows) will be denoted the training set and 20% (30 rows) will be set aside for testing.

```
iris_train, iris_test = train_test_split(iris_data, test_size = 0.2)
print(iris_train.head())
```

```
##      CLASS  sepal length (cm)  ...  petal length (cm)  petal width (cm)
## 89     1.0                5.5  ...                4.0               1.3
```

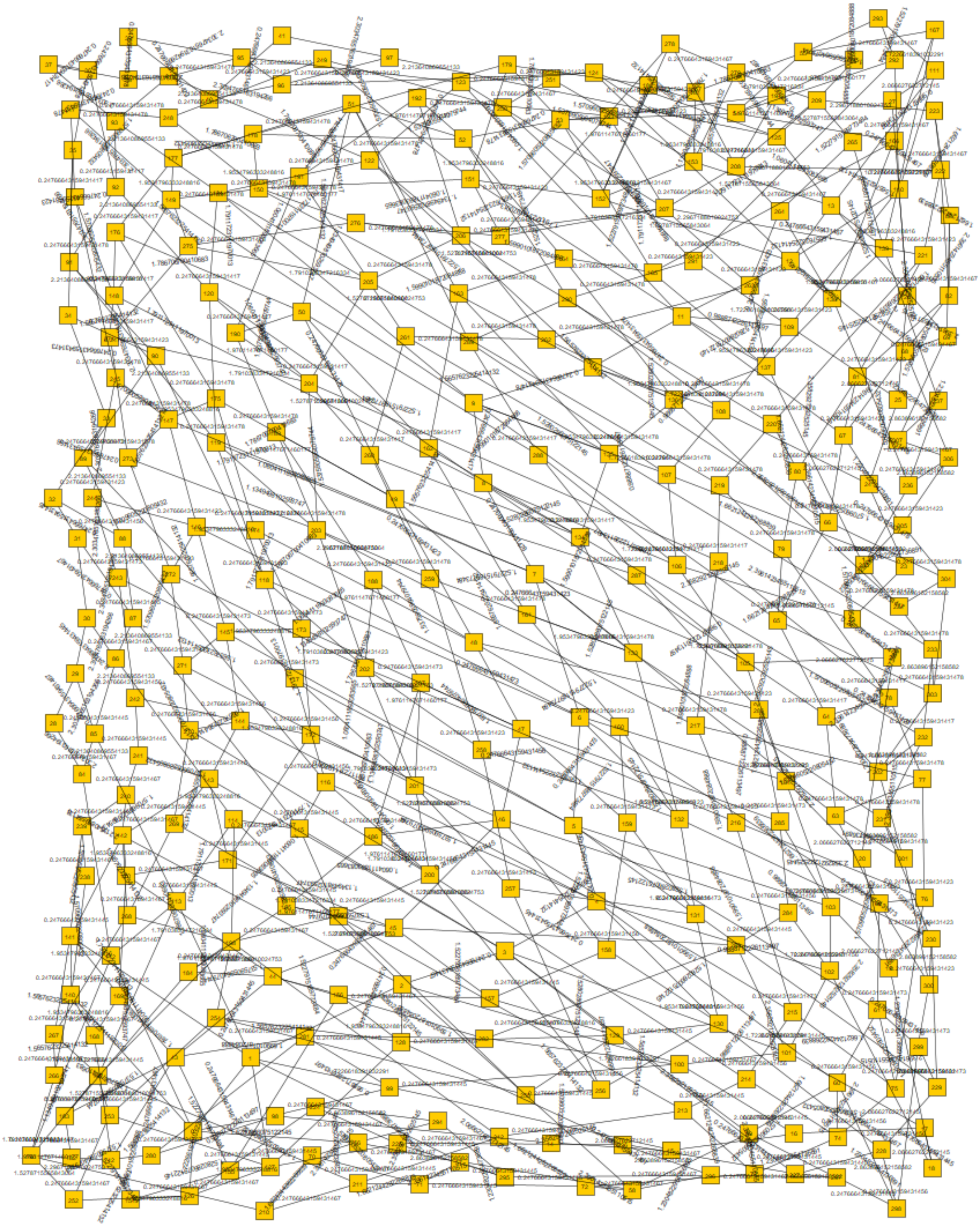


Figure 1: Relative Neighbourhood Graph for samples

```
## 97     1.0                6.2  ...                4.3               1.3
## 133    2.0                6.3  ...                5.1               1.5
## 16     0.0                5.4  ...                1.3               0.4
## 48     0.0                5.3  ...                1.5               0.2
##
## [5 rows x 5 columns]
```

Next perform entropy-based Fayyad-Irani discretisation (https://www.ijcai.org/Proceedings/93-2/Papers/022.pdf) using `RWeka::Discretize`, removing the ambiguous features that result. Following the discretisation step, no features are removed the dataset.

```
# py$iris_train$CLASS <- as.factor(py$iris_train$CLASS)
# train_data <- Discretize(CLASS ~ ., data = py$iris_train)
# # Remove any features that are identical after discretisation
# train_data <- train_data[!duplicated(lapply(train_data, summary)) & !duplicated(lapply(train_data, su
#                                                                   fromLast = TRUE)]
# head(train_data)
```

Aside: Save the cutpoints defined here for use on the test data.

```
# cutpoints <- list()
# feature <- character()
# for (i in seq(1, length(names(train_data))-1)){ # for each feature
#   feature[i] <- names(train_data)[i]
#   t <- table(train_data[,feature[i]], train_data$CLASS)
#   for (j in seq(1, length(levels(train_data[,names(train_data)[i]])))){ # for each bin of a feature
#     print(names(which.max(t[j,]))) # the label to assign to the bin
#     print(levels(train_data[,names(train_data)[i]])) # the values of the bin itself
#   }
# }



# feature <- character()
# cutpoint <- numeric()
# for (i in seq(1, length(names(train_data))-1)){
#   feature[i] <- names(train_data)[i]
#
#   cutpoint[i] <- as.numeric(regmatches(levels(train_data[, i])[2], gregexpr("-?\\d.\\d+", levels(trai
# }
# ( cutpoints <- data.frame(feature, cutpoint) )
```

Genuinely don't know how to go on with this question. The chi-squared test might not even remove any features. and if I have three classes at the end, what sort of distance measure could I define? Might need to go and find a binary dataset after all.

Exercise 3

Formal mathematical definition of the k-Feature Set Problem:
Input: A set $X = \{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ of m examples having $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}, t^{(i)}\} \in \{0,1\}^{n+1}\ \forall\ i$, and an integer $k > 0$.

Question: Does there exist a feature set S, where $S \subseteq \{1, ..., n\}$, $|S| = k$, and for all pairs of examples $i \neq j$: if $t^{(i)} \neq t^{(j)}$ $\exists$ $I \in S$ such that $x_I^{(i)} \neq x_I^{(j)}$?

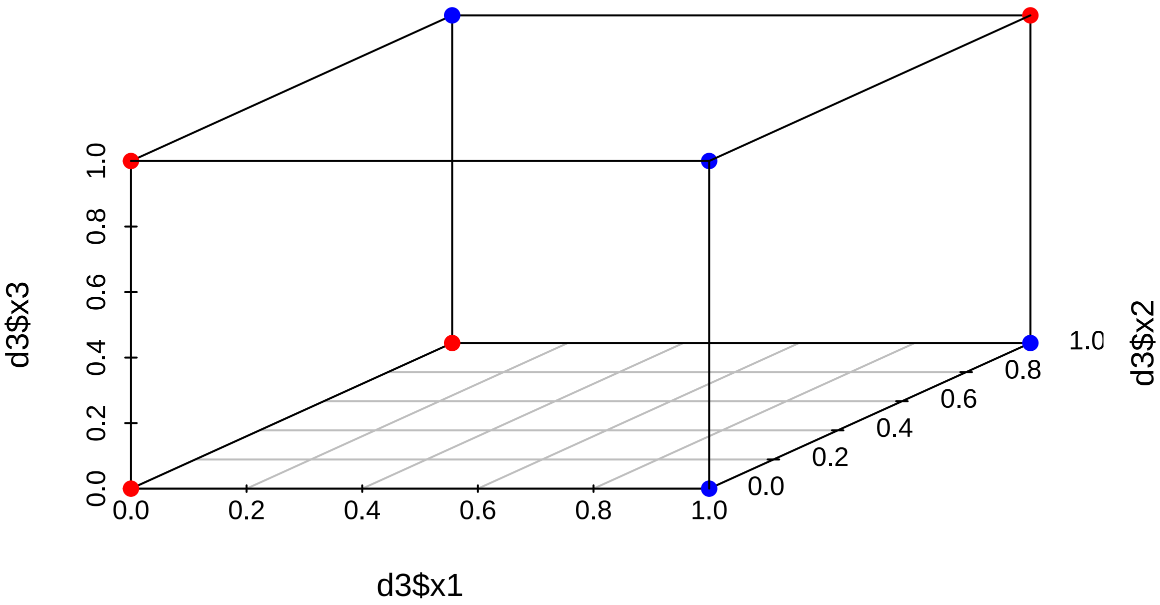
An example of the problem is as follows, where variables `x1`, `x2`, `x3`, `x4` represent characteristics of a student and `y` represents their grade on a test.

```
d3 <- data.frame(x1 = c(1,0,1,1,1,0,0,0),
                 x2 = c(0,1,1,0,1,1,0,0),
                 x3 = c(1,1,0,0,1,0,1,0),
                 x4 = c(0,1,0,1,0,1,0,1),
                 y  = as.factor(c(rep("pass", 4), rep("fail", 4))))
d3
```

```
##   x1 x2 x3 x4    y
## 1  1  0  1  0 pass
## 2  0  1  1  1 pass
## 3  1  1  0  0 pass
## 4  1  0  0  1 pass
## 5  1  1  1  0 fail
## 6  0  1  0  1 fail
## 7  0  0  1  0 fail
## 8  0  0  0  1 fail
```

It can be determined that this table has a 3-Feature set $F = \{x1, x2, x3\}$ and no 2-Feature sets. Additionally, the points are not linearly separable, as is demonstrated by the following 3-dimensional scatterplot (points of the "fail" class are plotted in red, while those in the "pass" class are plotted in blue):

```
scatterplot3d(x = d3$x1, y=d3$x2, z=d3$x3, color = c("red", "blue")[d3$y], pch=19)
```



It is clear from inspecting `Figure X` that no single plane could be drawn that separates these points by colour. Therefore, the 3-Feature set $F = \{x1, x2, x3\}$ is not linearly separable.

Exercise 4

Formal mathematical definition of the l-Pattern Identification Problem:
Input: A finite alphabet Σ, two disjoint sets $Good, Bad \subseteq \ \Sigma^n$ of strings (where a string is a concatenation of symbols from that alphabet) and an integer $l > 0$.
Question: Is there a set P of patterns (where a pattern is a string s over an extended alphabet $\Sigma_* := \Sigma \cup \{*\}$) having $|P| \leq l$ and $P \rightarrow (Good, Bad)$?

A toy example of the problem is as follows, with factors contributing to a student achieving a passing or failing `grade` on a piano examination. The features consistute `level` the level of difficulty of the exam, `daily_practice` the amount of daily practice performed (>1hr, 1-2hrs, 2+hrs), `fitness` the regularity of the student's exercise, and `lesson_attendance` the student's regularity of lesson attendance. These are all measured on a scale with 3 levels - Low (L), Medium (M) and High (H).

```
d4 <-  data.frame(level = c("L", "L", "L", "M", "L", "M", "H", "H"),
                  daily_practice = c("M", "H", "L", "H", "L", "L", "L", "M"),
                  fitness = c("M", "L", "M", "L", "M", "L", "H", "H"),
                  lesson_attendance = c("H", "M", "H", "L", "H", "L", "L", "L"),
                  grade = as.factor(c(rep("pass", 4), rep("fail", 4))))
d4
```

```
##   level daily_practice fitness lesson_attendance grade
## 1     L              M       M                 H  pass
## 2     L              H       L                 M  pass
## 3     L              L       M                 H  pass
## 4     M              H       L                 L  pass
## 5     L              L       M                 H  fail
## 6     M              L       L                 L  fail
## 7     H              L       H                 L  fail
## 8     H              M       H                 L  fail
```

The following 4-pattern solution is presented:
For pass,
$L * MH$
$* HL *$
For fail,
$* L * L$
$H * * L$
These two sets of patterns uniquely identify samples corresponding to their respective classes of grade, and each cover all existing samples in each group.
Examining the data shows no 3-pattern solutions.

Exercise 5

The Iris dataset from sklearn.datasets will be used for this task. The target cluster will be removed for the purpose of computing the MST.

```
from operator import itemgetter
from sklearn import datasets
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
iris_data = pd.DataFrame(data= np.c_[iris['target'], iris['data']],
                     columns= ['CLASS'] + iris['feature_names'])
iris_features = iris_data.drop("CLASS", axis = 1) # remove the target from a copy of the data
i_names = [str(i) for i in iris_features.index.values] # Define node names by row
print(iris_features.head())
```

```
##    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
## 0                5.1               3.5                1.4               0.2
## 1                4.9               3.0                1.4               0.2
## 2                4.7               3.2                1.3               0.2
## 3                4.6               3.1                1.5               0.2
## 4                5.0               3.6                1.4               0.2
```

The remaining dataframe has 150 samples and four features.

The data will be normalised using a z-score transformation, and a distance matrix defined using the Euclidean distance measure.

```
# Normalise the data
norm_iris_data = (iris_features - iris_features.mean())/iris_features.std()
def euclid_distance(s1, s2):
    return math.sqrt(np.sum((s1-s2) ** 2))
# Distance matrix for samples:
```

```
iris_samples_edist_m = [[euclid_distance(norm_iris_data.iloc[y],
  norm_iris_data.iloc[x]) for y in range(len(i_names))] for x in range(len(i_names))]
round(pd.DataFrame(iris_samples_edist_m, columns = i_names, index = i_names), 3)
```

```
##          0      1      2      3      4  ...    145    146    147    148    149
## 0    0.000  1.172  0.843  1.100  0.259  ...  4.156  4.062  3.793  3.813  3.324
## 1    1.172  0.000  0.522  0.433  1.382  ...  4.117  3.648  3.734  4.004  3.203
## 2    0.843  0.522  0.000  0.283  0.988  ...  4.303  3.960  3.923  4.059  3.369
## 3    1.100  0.433  0.283  0.000  1.246  ...  4.297  3.875  3.910  4.084  3.329
## 4    0.259  1.382  0.988  1.246  0.000  ...  4.282  4.239  3.923  3.878  3.446
## ..     ...    ...    ...    ...    ...  ...    ...    ...    ...    ...    ...
## 145  4.156  4.117  4.303  4.297  4.282  ...  0.000  1.356  0.462  1.104  1.169
## 146  4.062  3.648  3.960  3.875  4.239  ...  1.356  0.000  1.185  2.146  1.253
## 147  3.793  3.734  3.923  3.910  3.923  ...  0.462  1.185  0.000  1.068  0.773
## 148  3.813  4.004  4.059  4.084  3.878  ...  1.104  2.146  1.068  0.000  1.197
## 149  3.324  3.203  3.369  3.329  3.446  ...  1.169  1.253  0.773  1.197  0.000
##
## [150 rows x 150 columns]
```

5. a)

```
class Edge:
    def __init__(self, s, w, e):
        self.start = s
        self.weight = w
        self.end = e
    def get_edge(self):
        return (str(self.start) + ’-’ + str(self.weight) + ’-’ + str(self.end))
    def get_nx_edge(self):
        return ((self.start, self.end, {’label’: str(self.weight)}))

def minimum_spanning_tree_Prims(G):
    V = [i for i in range(len(G))] # nodes in the graph
    W = [] # nodes in the MST
    adj_weights = [0] + [float(’inf’)] * (len(G) - 1) # the [0] initalises the first node into the MST
    u = [-1] * len(G)
    def closestNode(weights, added_nodes): # find the nearest node in the ’adjacent weights’ list
        m = float(’inf’) # minimum edge distance
        v = -1 # node to return
        for i in range(len(weights)):
            if ((i not in added_nodes) & (weights[i] < m)):
                m = weights[i]
                v = i
        return v

    while (set(W) != set(V)):
        v = closestNode(adj_weights, W)
        if (v != -1):
            W.append(v)
        for i in range(len(G)): # for each node:
            if ((i not in W) & (G[v][i] < adj_weights[i])):
                u[i] = v
                adj_weights[i] = G[v][i]
    return W, u

def get_mst(matrix, names, output_file):
    W, u = minimum_spanning_tree_Prims(matrix)
    node_labels = [names[n] for n in W]
    F = []
    for f in range(1, len(matrix)): # unordered
        F.append(Edge(names[u[f]], matrix[f][u[f]], names[f]))
    G = nx.Graph()
    G.add_nodes_from(node_labels)
    G.add_edges_from([f.get_nx_edge() for f in F])
    nx.write_gml(G, output_file)
    return G
five_a_result = get_mst(iris_samples_edist_m, i_names, ’graph_5a.gml’)
```

The output file graph_5a.gml was processed using yEd Graph Editor to produce the following visualisation in Figure XXXX:

```
graph_5a <- readPNG("graph_5a.png")
grid.raster(graph_5a)
```

5. b)

```
def k_NNG(G, K):
    P = [i for i in range(len(G))]
    kNNG = []
    for p in P:
        dist = []
        for q in P:
            if (p != q):
                dist.append([p, G[p][q], q])
        sorted_dist = sorted(dist, key = itemgetter(1))
        max_dist = sorted_dist[K-1][1]
        for d in sorted_dist:
            if d[1] <= max_dist:
                kNNG.append(d)
    return kNNG

def get_k_NNG(k, matrix, names, output_file):
    E = k_NNG(matrix, k)
    V = [i for i in range(len(matrix))]
    kNNG = []
    node_labels = [names[n] for n in V]
    for e in range(len(E)):
        kNNG.append(Edge(node_labels[E[e][0]], matrix[E[e][0]][E[e][2]], node_labels[E[e][2]]))
    G = nx.Graph()
    G.add_nodes_from(node_labels)
    G.add_edges_from([e.get_nx_edge() for e in kNNG])
```

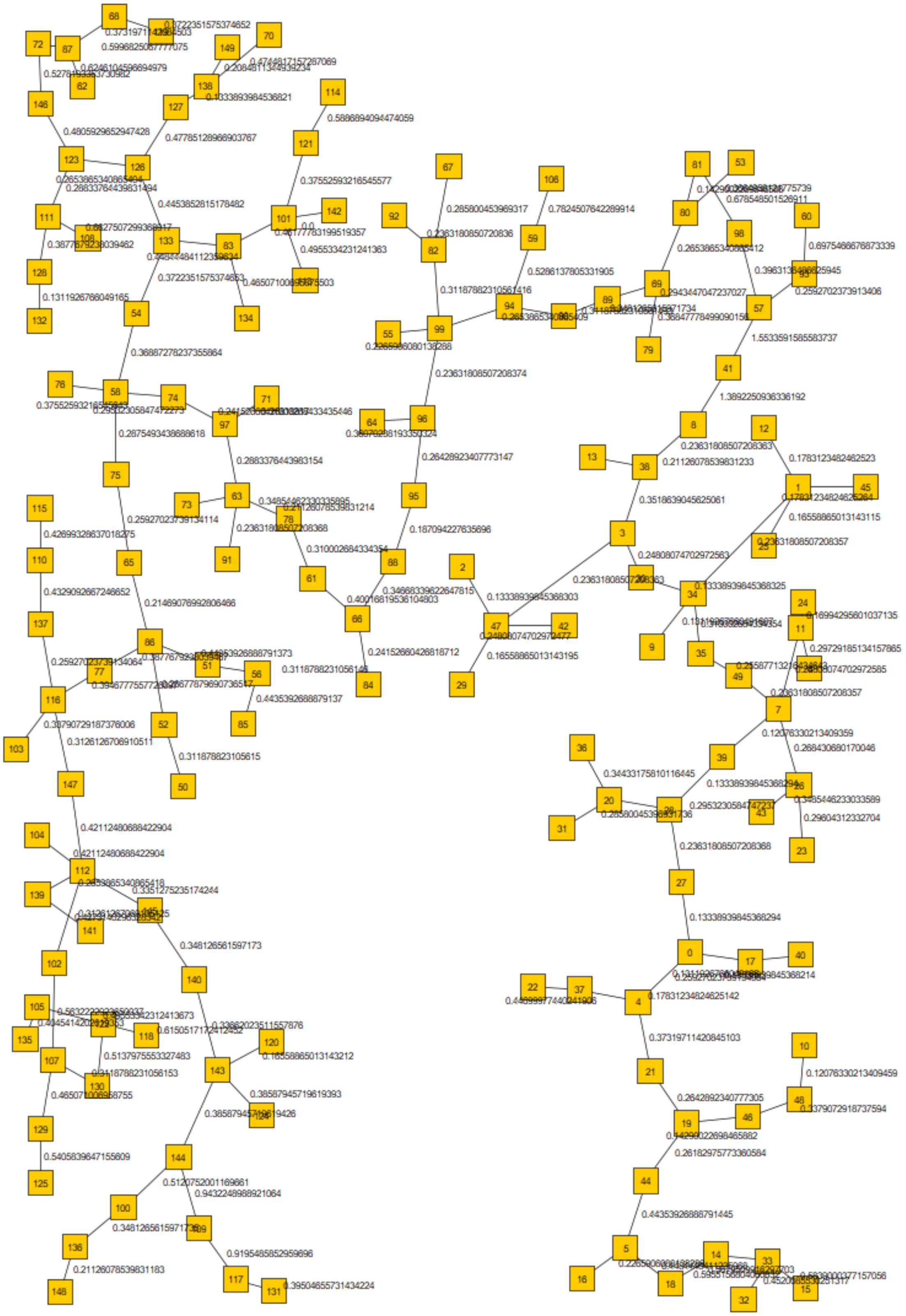


Figure 2: Minimum Spanning Tree for Iris dataset samples

```
    nx.write_gml(G, output_file)
    return G
five_b_result = get_k_NNG(3, iris_samples_edist_m, i_names, 'graph_5b.gml')
```

The output file graph_5b.gml was processed using yEd Graph Editor to produce the following visualisation in `Figure XXXX`:

```
graph_5b <- readPNG("graph_5b.png")
grid.raster(graph_5b)
```

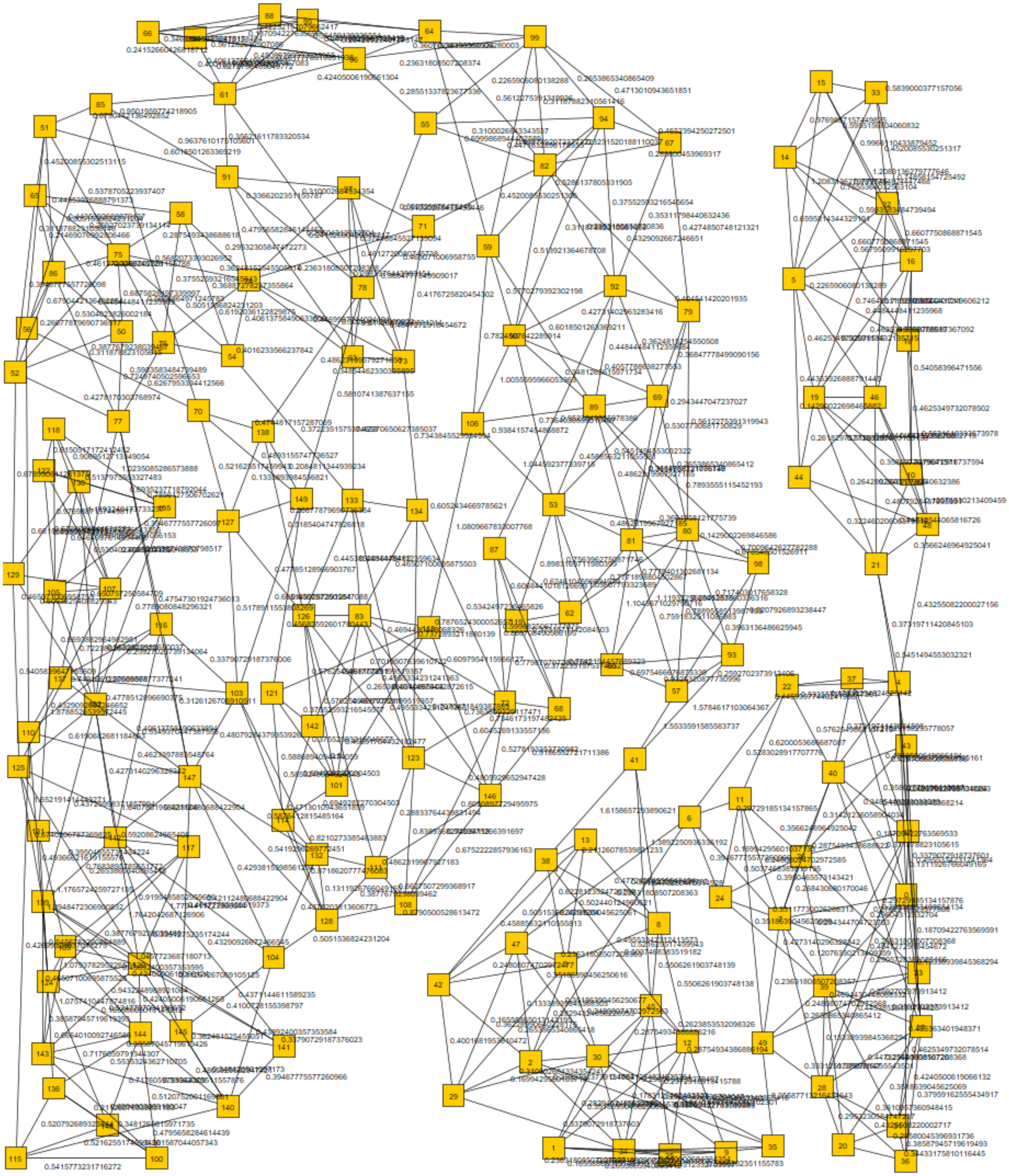


Figure 3: k-nearest-neighbours for Iris dataset samples

5. c)

```
# Identify the common edges in the MST and k-NN graphs:
MST_edges = list(five_a_result.edges)
kNNG_edges = list(five_b_result.edges)
common_edges = list(set(MST_edges).intersection(kNNG_edges))

# Generate yEd output
G = nx.Graph()
G.add_nodes_from(i_names)
G.add_edges_from(common_edges)
nx.write_gml(G, 'graph_5c.gml')

# Tabulate the clusters
cluster = []
for i in i_names:
  cluster.append(sorted(list(nx.node_connected_component(G, i))))
u = [list(x) for x in set(tuple(x) for x in cluster)]
count = len(u)
clusters = list(zip(range(count), u))
clusters = pd.DataFrame(clusters, columns = ['cluster_label', 'samples'])
```

5. d)

The output file graph_5c.gml was processed using yEd Graph Editor to produce the following visualisation in `Figure XXXX`:

```
graph_5d <- readPNG("graph_5d.png")
grid.raster(graph_5d)
```

The results were tabulated in part c, and the results printed below:

```
print(clusters)
```

```
##     cluster_label          samples
## 0               0             [13]
## 1               1  [19, 44, 46, 48]
## 2               2             [42]
## 3               3      [51, 56, 85]
## 4               4       [16, 18, 5]
## ..            ...              ...
## 68             68         [106, 59]
## 69             69             [74]
## 70             70             [10]
## 71             71  [63, 73, 91, 97]
## 72             72             [70]
##
## [73 rows x 2 columns]
```

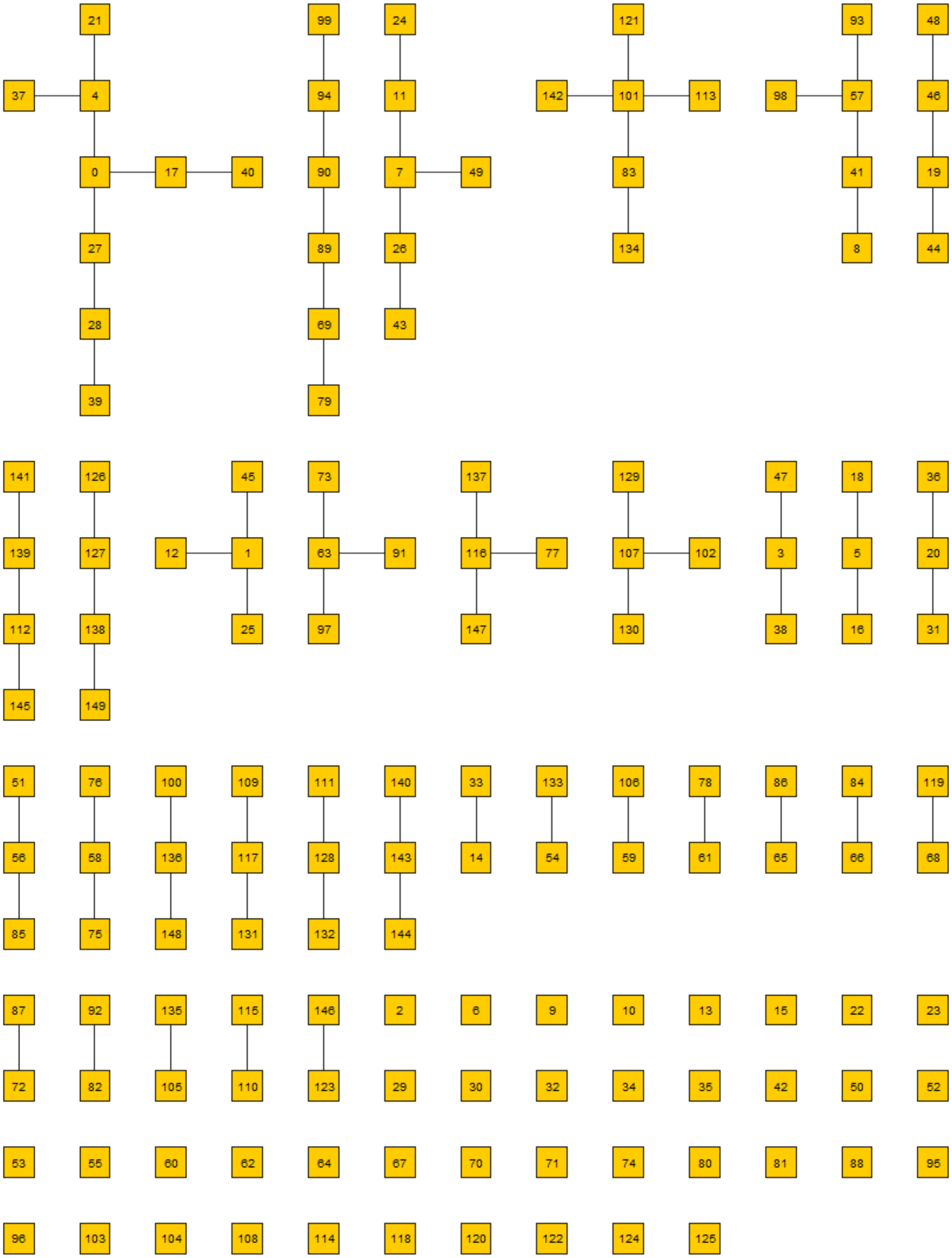


Figure 4: MST-kNN for Iris dataset samples

Exercise 6

6. a)

6. b)

6. c)

Exercise 7

7. a)

In classification, lazy learning is a type of learning process which lacks an explicit generalisation phase; no function to discriminate between classes is defined before queries are received. Instead, most of the generalisation tends to occur at consultation time, ie. when the algorithm is first presented with test data to classify, comparisons to the training set are performed and a result is deduced. Their main advantage is having the flexibility to handle evolution of the problem domain, however lazy learners are typically slower at making predictions than the alternative: eager learners. These classifiers perform the majority of their computation in advance of seeing the data to evaluate; they construct a target function upon receiving training data.
An example of a lazy classifier is the k-nearest neighbour algorithm.

Class imbalance is the issue of having differing distributions of samples across each of the target class present in the data, where these group sizes vary slightly or severely ('balanced classes' means there are approximately the same number of samples for each group). It may arise as a natural result of the problem domain, or due to sampling error. The class imbalance problem often leads to bias in a variety of predictive performance measures, particularly accuracy. Many algorithms are not inherently designed to deal with such issues; they tend to favour improving performance on the majority class, and the predictive performance of the infrequent class suffers. This is particularly undesirable in cases such as anomaly detection, where the minority class is much more important to detect than the majority (eg. scanning millions of transactions for examples of credit card fraud). This issue must be assessed on a case-by-case basis; there is no one-size-fits-all solution, although specialised techniques to assist do exist (eg. oversampling the minority class, generating synthetic samples for the minority class, or simply using a different performance metric like precision, recall or F1-score).

7. b)

The sensitivity of a test refers to its true positive rate, ie. the proportion of correctly identified positive outcomes out of all those that are genuinely positive. An increase in sensitivity attempts to reduce the number of false negative outcomes (and therefore type II errors). In a medical context, sensitivity may be referred to as the detection rate, and refers to the proportion of people who test positive for a condition within a sample who have it, thereby minimising misclassification of people requiring treatment. Sensitivity as defined as $\frac{TP}{(TP+FN)}$.

In contrast, the specificity of a test measures its ability to correctly identify negative results amongst all those that are actually negative, reducing the risk of false positive outcomes (type I errors). A complementary example to that above would be the proportion of people who test negative for a condition within a sample of people who do not have said condition. Therefore, sensitivity and specifity form a trade-off; increasing one will usually naturally reduce the other. Specificity is equal to $\frac{TN}{(FP+TN)}$.

Accuracy is a measure of a test's ability to correctly classify both positive and negative results amongst all outcomes. It is calculated as $\frac{(TP+TN)}{(TP+FP+TN+FN)}$.

Matthew's Correlation Coefficient is another measure of quality, which is regarded as a typically 'balanced' measure when assessing confusion matrices of results. Its formula is $\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$, and its range of results is [-1,1] with similar implications to correlation coefficients.

Examples of these measures being computed are given below. The context of such an example may be that three new tests for an illness are being assessed on a group of 300 people where 150 people are genuinely ill, and 150 are not. These conditions of well-balanced classes are less likely to occur in real-life scenarios.

```
has_illness = c(rep(TRUE, 150), rep(FALSE, 150))
# Test 1
( m1 <- table(prediction = c(rep(TRUE, 120), rep(FALSE, 70), rep(TRUE, 110)), has_illness) )
```

```
##            has_illness
## prediction FALSE TRUE
##      FALSE    40   30
##      TRUE    110  120
```

From this table we see there are 40 true negative values, 30 false negative value, 110 false positive values and 120 true positive values. The results are:
$Sensitivity = \frac{120}{(120+30)} = 80\%$
$Specificity = \frac{40}{(30+40)} = 57.14\%$
$Accuracy = \frac{(120+40)}{(120+110+40+30)} = 53.33\% \ (2.d.p)$
$MCC = \frac{120*40-110*30}{\sqrt{(120+40)(120+30)(40+110)(40+30)}} = 9.45\% \ (2.d.p)$

Note that the sensitivity is quite high; higher than the specificity, yet the accuracy is worse than both and the MCC is very low. Inferences from these scores should only be made if the true population intended for this test to be performed on has a similar class split to that in this sample, due to the differences inherently introduced by imbalance.

```
# Test 2
( m2 <- table(prediction = c(rep(TRUE, 50), rep(FALSE, 230), rep(TRUE, 20)), has_illness) )
```

```
##            has_illness
## prediction FALSE TRUE
##      FALSE   130  100
##      TRUE     20   50
```

From this table we see there are 130 true negative values, 100 false negative value, 20 false positive values and 50 true positive values. The results are:
$Sensitivity = \frac{50}{(50+100)} = 33.33\% \ (2.d.p)$
$Specificity = \frac{130}{(20+130)} = 86.67\% \ (2.d.p)$
$Accuracy = \frac{(50+130)}{(50+20+130+100)} = 60\%$
$MCC = \frac{50*130-20*100}{\sqrt{(50+20)(50+100)(130+20)(130+100)}} = 23.64\% \ (2.d.p)$

Note that this test has higher specificity than sensitivity, but the accuracy is still quite low. The MCC has increased somewhat compared to test 1.

```
# Test 3
( m3 <- table(prediction = c(rep(TRUE, 140), rep(FALSE, 130), rep(TRUE, 30)), has_illness) )
```

```
##           has_illness
## prediction FALSE TRUE
##      FALSE   120   10
##      TRUE     30  140
```

From this table we see there are 120 true negative values, 10 false negative value, 30 false positive values and 140 true positive values. The results are:
$Sensitivity = \frac{140}{(140+10)} = 93.33\% \ (2.d.p)$
$Specificity = \frac{1200}{(30+120)} = 80\%$
$Accuracy = \frac{(140+120)}{(140+30+10+120)} = 86.67\% \ (2.d.p)$
$MCC = \frac{140*120-30*10}{\sqrt{(140+120)(140+10)(120+30)(120+10)}} = 59.83\% \ (2.d.p)$

This test has relatively high sensitivity and specificity, and its accuracy and MCC are also quite high.

Exercise 8

- How is this different from Exercise 6? Should we incorporate those 2 things and a last one?

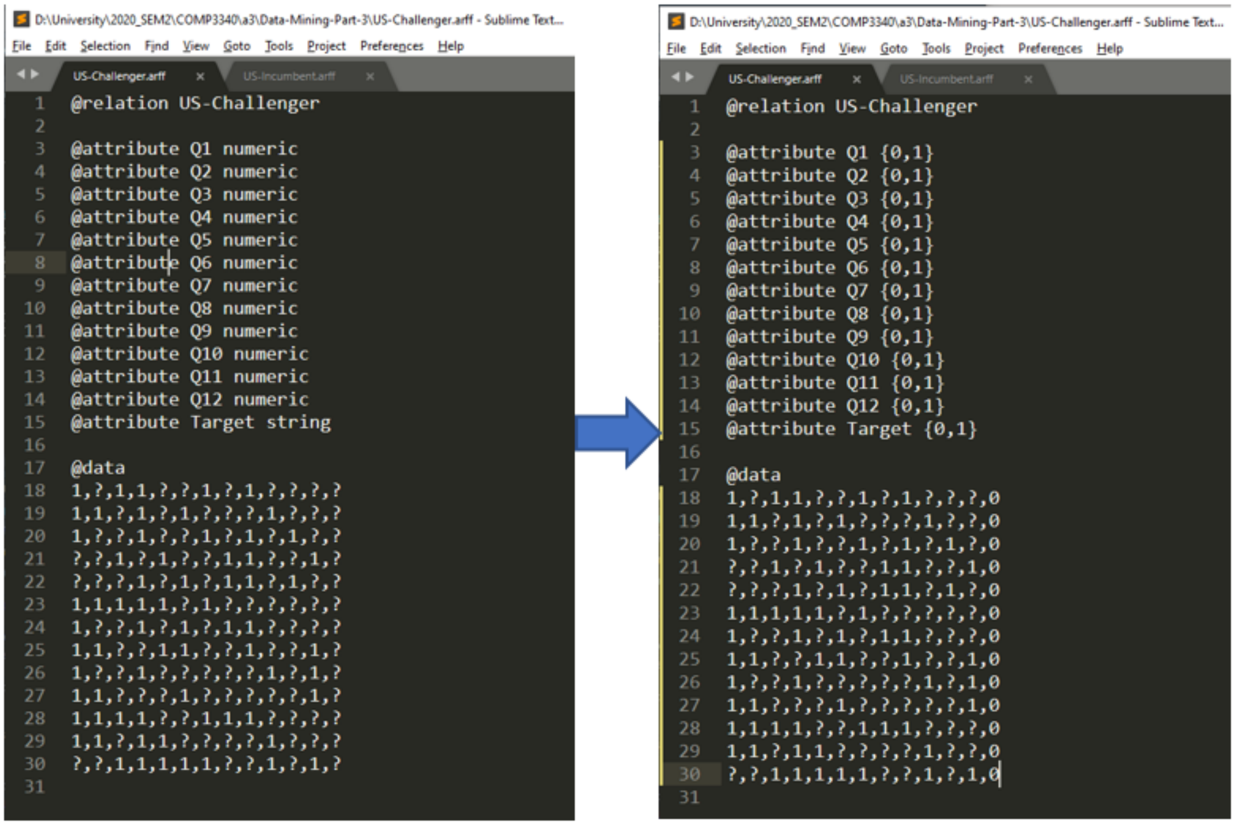
Exercise 9

9. a)

```
pres_data <- read.csv("USPresidency.csv", stringsAsFactors = TRUE)

# Where Target == 0:
US_Challenger <- pres_data[pres_data$Target == 0,]
US_Challenger <- US_Challenger[, c("Q1", "Q2", "Q3", "Q4", "Q5", "Q6", "Q7", "Q8", "Q9",
                                   "Q10", "Q11", "Q12", "Target")]
US_Challenger[US_Challenger == 0] <- noquote("?")
write.csv(US_Challenger, "US-Challenger.csv", row.names = FALSE)

# Where Target == 1:
US_Incumbent <- pres_data[pres_data$Target == 1,]
US_Incumbent <- US_Incumbent[, c("Q1", "Q2", "Q3", "Q4", "Q5", "Q6", "Q7", "Q8", "Q9",
                                 "Q10", "Q11", "Q12", "Target")]
US_Incumbent[US_Incumbent == 0] <- noquote("?")
write.csv(US_Incumbent, "US-Incumbent.csv", row.names = FALSE)
```

Each file was then opened in Weka and saved as a .arff file. These files were then opened in a text editor and modified as shown in Figure X below, using US-Challenger.arff as an example:

```
q9a_process <- readPNG("q9_process_3.png")
grid.raster(q9a_process)
```


D:\University\2020_SEM2\COMP3340\a3\Data-Mining-Part-3\US-Challenger.arff - Sublime Text...
File Edit Selection Find View Goto Tools Project Preferences Help
US-Challenger.arff
US-Incumbent.arff
D:\University\2020_SEM2\COMP3340\a3\Data-Mining-Part-3\US-Challenger.arff - Sublime Text...
File Edit Selection Find View Goto Tools Project Preferences Help
US-Challenger.arff
US-Incumbent.arff

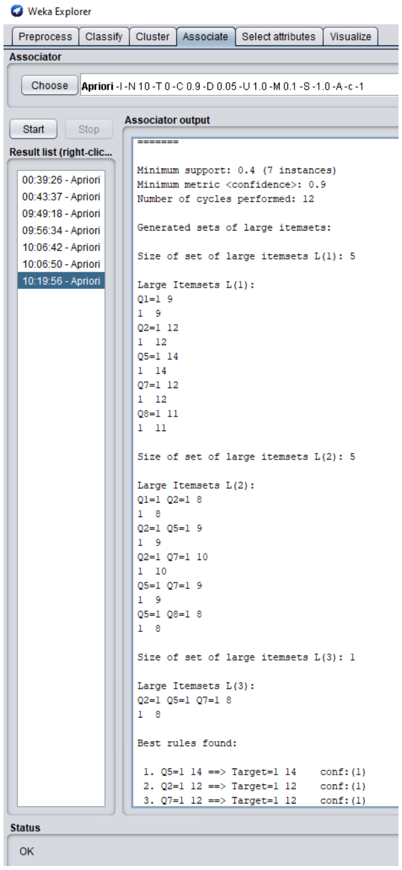
```
@relation US-Challenger

@attribute Q1 numeric
@attribute Q2 numeric
@attribute Q3 numeric
@attribute Q4 numeric
@attribute Q5 numeric
@attribute Q6 numeric
@attribute Q7 numeric
@attribute Q8 numeric
@attribute Q9 numeric
@attribute Q10 numeric
@attribute Q11 numeric
@attribute Q12 numeric
@attribute Target string

@data
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,?,?,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,?
```

```
@relation US-Challenger

@attribute Q1 {0,1}
@attribute Q2 {0,1}
@attribute Q3 {0,1}
@attribute Q4 {0,1}
@attribute Q5 {0,1}
@attribute Q6 {0,1}
@attribute Q7 {0,1}
@attribute Q8 {0,1}
@attribute Q9 {0,1}
@attribute Q10 {0,1}
@attribute Q11 {0,1}
@attribute Q12 {0,1}
@attribute Target {0,1}

@data
1,?,1,1,?,?,1,?,1,?,?,?,0
1,1,?,1,?,1,?,?,?,1,?,?,0
1,?,?,1,?,?,1,?,1,?,1,?,0
?,?,1,?,1,?,?,1,1,?,?,1,0
?,?,?,1,?,1,?,1,1,?,1,?,0
1,1,1,1,1,?,1,?,?,?,?,?,0
1,?,?,1,?,1,?,1,1,?,?,?,0
1,1,?,?,1,1,?,?,1,?,?,1,0
1,?,?,1,?,?,?,?,?,1,?,1,0
1,1,?,?,?,1,?,?,?,?,?,1,0
1,1,1,1,?,?,1,1,1,?,?,?,0
1,1,?,1,1,?,?,?,?,1,?,?,0
?,?,1,1,1,1,1,?,?,1,?,1,0
```

Figure 5: Modification of US-Challenger.arff

Next, US-Incumbent.arff was opened in Weka and the Apriori algorithm run with `car=TRUE` after ensuring that Target was set to `Attribute as class`. The output is as follows in Figure X.

```r
q9a_out1 <- readPNG("q9_incumbent_output.png")
grid.raster(q9a_out1)
```



Apriori -I -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -A -c -1

```
=======

Minimum support: 0.4 (7 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 12

Generated sets of large itemsets:

Size of set of large itemsets L(1): 5

Large Itemsets L(1):
Q1=1 9
1  9
Q2=1 12
1  12
Q5=1 14
1  14
Q7=1 12
1  12
Q8=1 11
1  11

Size of set of large itemsets L(2): 5

Large Itemsets L(2):
Q1=1 Q2=1 8
1  8
Q2=1 Q5=1 9
1  9
Q2=1 Q7=1 10
1  10
Q5=1 Q7=1 9
1  9
Q5=1 Q8=1 8
1  8

Size of set of large itemsets L(3): 1

Large Itemsets L(3):
Q2=1 Q5=1 Q7=1 8
1  8

Best rules found:

 1. Q5=1 14 ==> Target=1 14    conf:(1)
 2. Q2=1 12 ==> Target=1 12    conf:(1)
 3. Q7=1 12 ==> Target=1 12    conf:(1)
```

Figure 6: Apriori Output for US-Incumbent.arff

As can be seen above, the largest frequent itemset found for this set was size 3: {Q2, Q5, Q7}.

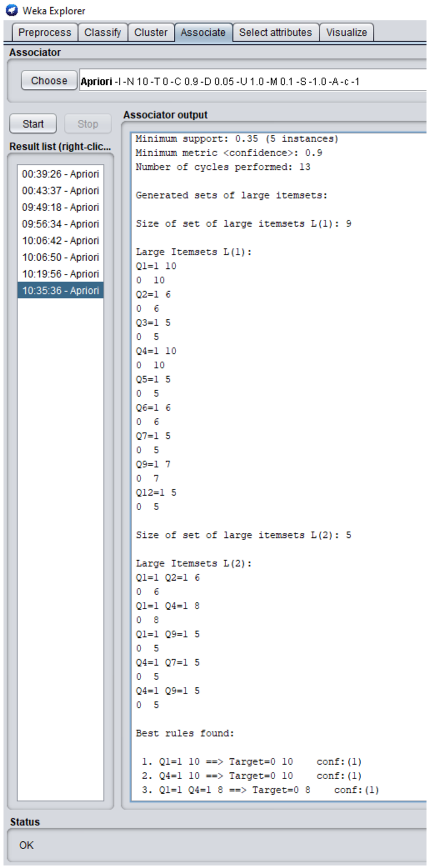
Association rule 1: $Q5 \Rightarrow Target$
$Confidence(Q5 \Rightarrow Target) = \frac{14}{14} = 1$
$Lift((Q5 \Rightarrow Target)) = \frac{Confidence(Q5 \Rightarrow Target)}{Support(Target)} = \frac{1}{18} = 0.055\ (3.d.p)$

Association rule 2: $Q2 \Rightarrow Target$
$Confidence(Q2 \Rightarrow Target) = \frac{Confidence(Q2 \Rightarrow Target)}{Support(Target)} = \frac{12}{12} = 1$
$Lift((Q2 \Rightarrow Target)) = \frac{1}{18} = 0.055\ (3.d.p)$

Association rule 3: $Q7 \Rightarrow Target$
$Confidence(Q7 \Rightarrow Target) = \frac{Confidence(Q7 \Rightarrow Target)}{Support(Target)} = \frac{12}{12} = 1$
$Lift((Q7 \Rightarrow Target)) = \frac{1}{18} = 0.055\ (3.d.p)$

Finally, US-Challenger.arff was opened in Weka and the Apriori algorithm run with `car=TRUE`. The output is shown in Figure X.

```r
q9a_out2 <- readPNG("q9_challenger_output.png")
grid.raster(q9a_out2)
```



Apriori -I -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -A -c -1

```
Minimum support: 0.35 (5 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 13

Generated sets of large itemsets:

Size of set of large itemsets L(1): 9

Large Itemsets L(1):
Q1=1 10
0  10
Q2=1 6
0  6
Q3=1 5
0  5
Q4=1 10
0  10
Q5=1 5
0  5
Q6=1 6
0  6
Q7=1 5
0  5
Q9=1 7
0  7
Q12=1 5
0  5

Size of set of large itemsets L(2): 5

Large Itemsets L(2):
Q1=1 Q2=1 6
0  6
Q1=1 Q4=1 8
0  8
Q1=1 Q9=1 5
0  5
Q4=1 Q7=1 5
0  5
Q4=1 Q9=1 5
0  5

Best rules found:

 1. Q1=1 10 ==> Target=0 10    conf:(1)
 2. Q4=1 10 ==> Target=0 10    conf:(1)
 3. Q1=1 Q4=1 8 ==> Target=0 8    conf:(1)
```

OK

Figure 7: Apriori Output for US-Challenger.arff

As can be seen above, the largest frequent itemset found for this set was size 2: {Q1, Q2}, {Q1, Q4}, {Q1, Q9}, {Q4, Q7}, {Q4, Q9}. Of these, {Q1, Q4} has the highest support.

Association rule 1: $Q1 \Rightarrow Target$
$Confidence(Q1 \Rightarrow Target) = \frac{10}{10} = 1$
$Lift((Q1 \Rightarrow Target)) = \frac{Confidence(Q1 \Rightarrow Target)}{Support(Target)} = \frac{1}{13} = 0.077\ (3.d.p)$

Association rule 2: $Q4 \Rightarrow Target$
$Confidence(Q4 \Rightarrow Target) = \frac{Confidence(Q4 \Rightarrow Target)}{Support(Target)} = \frac{10}{10} = 1$
$Lift((Q4 \Rightarrow Target)) = \frac{1}{13} = 0.077\ (3.d.p)$

Association rule 3: $\{Q1, Q4\} \Rightarrow Target$
$Confidence(\{Q1, Q4\} \Rightarrow Target) = \frac{Confidence(\{Q1,Q4\} \Rightarrow Target)}{Support(Target)} = \frac{8}{8} = 1$
$Lift((\{Q1, Q4\} \Rightarrow Target)) = \frac{1}{13} = 0.077\ (3.d.p)$

9. b)

We will start using the Challenger Victory data, US-Challenger.arff, which has 13 samples:
We first set the minimum support threshold to 60% by setting `lowerBoundMinSupport=0.6`, and run

the Apriori algorithm. The algorithm will run from the `upperBoundMinSupport` of 1.0 down to the `lowerBoundMinSupport`, 0.6, in increments of `delta` which is 0.05 by default.

The minimum support starts at 100% (expressed as a proportion of the number of instances) or 13 (expressed as the number of instances), the upper bound. At `k=1`, there are no itemsets with support 13, and by the second Monotonicity rule we know that 'if an itemset is infrequent, then none of its supersets will be frequent' so it terminates and the minimum support threshold is adjusted. The support is decreased by 5% to 95%, or 12.35 ≈ 12 (rounded to nearest integer). Again at `k=1` there are no itemsets with support 12 so the support is decreased by 5% to 90%, or 11.7 ≈ 12 for the next cycle. Once more we find no itemsets with support 12 for `k=1` so the support is decreased by 5% to 85%, or 11.05 ≈ 11. At `k=1` there is a frequent itemset having support 11: {Q4}, however this generates only one rule and the parameter `numRules` is set to 10, meaning the algorithm will try to find up to 10 rules having confidence greater than 60%. Shorthand will be used to describe the steps from here:

Cycle 4:
minSupport = 80% or 10.4 ≈ 10
`k=1`: {Q4} has support 11, {Q1} has support 10
`k=2`: no itemsets.

Cycle 5:
minSupport = 75% or 9.75 ≈ 10
`k=1`: {Q4} has support 11, {Q1} has support 10
`k=2`: no itemsets.

Cycle 6:
minSupport = 70% or 9.1 ≈ 9
`k=1`: {Q4} has support 11, {Q1} has support 10
`k=2`: no itemsets.

Cycle 7:
minSupport = 65% or 8.45 ≈ 8
`k=1`: {Q4} has support 11, {Q1} has support 10
`k=2`: {Q1, Q4} has support 8
`k=3`: no itemsets.

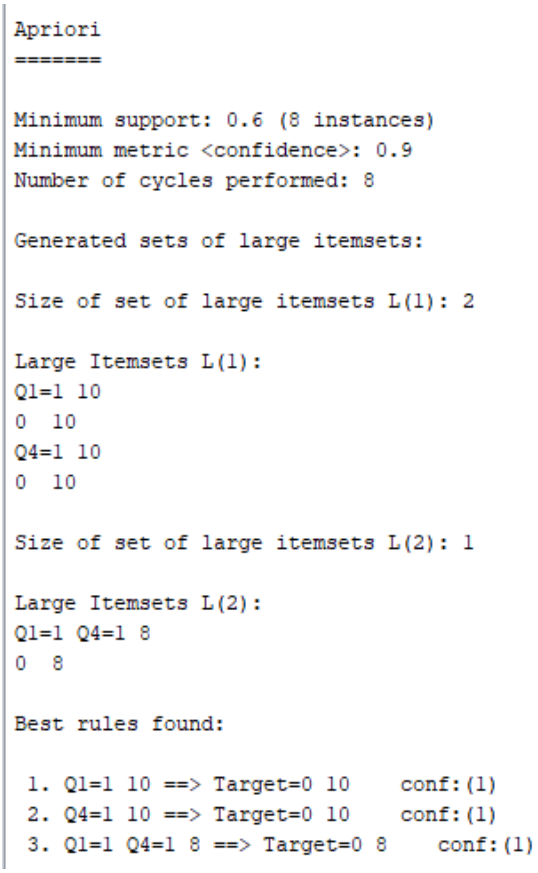
Cycle 8:
minSupport = 60% or 7.8 ≈ 8
`k=1`: {Q4} has support 11, {Q1} has support 10
`k=2`: {Q1, Q4} has support 8
`k=3`: no itemsets.

At this point, we have found 3 association rules but reached the minimum support threshold of 60%, which is a stopping point for the algorithm so it halts. The rules are as follows:

$\{Q4\} \Rightarrow Target$
$\{Q1\} \Rightarrow Target$
$\{Q1, Q4\} \Rightarrow Target$

This is supported by the output in Weka, shown in Figure X:

```
q9b_out1 <- readPNG("q9b_challenger_output.png")
grid.raster(q9b_out1)
```



```
Apriori
=======

Minimum support: 0.6 (8 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 8

Generated sets of large itemsets:

Size of set of large itemsets L(1): 2

Large Itemsets L(1):
Q1=1 10
0  10
Q4=1 10
0  10

Size of set of large itemsets L(2): 1

Large Itemsets L(2):
Q1=1 Q4=1 8
0  8

Best rules found:

 1. Q1=1 10 ==> Target=0 10    conf:(1)
 2. Q4=1 10 ==> Target=0 10    conf:(1)
 3. Q1=1 Q4=1 8 ==> Target=0 8    conf:(1)
```

Figure 8: Apriori Output for US-Challenger with minConf = 0.6.arff

Now considering the Incumbent Victory data, US-Incumbent.arff, which has 18 samples:
We again first set the minimum support threshold to 60% by setting `lowerBoundMinSupport=0.6`, and run the Apriori algorithm. The steps are as follows:
Start with minimum support 100% (expressed as a proportion of the number of instances) or 18 (the number of instances), the upper bound. At `k=1`, there are no frequent itemsets having support of 18, so the minimum support is decreased to 95% or 17.1 ≈ 17, where at `k=1` there are once more no itemsets with minimum support 17:

Cycle 2:
minSupport = 90% or 16.2 ≈ 16
`k=1`: no itemsets.

Cycle 3:
minSupport = 85% or $15.3 \approx 15$
`k=1`: no itemsets.

Cycle 4:
minSupport = 80% or $14.4 \approx 14$
`k=1`: {Q5} has support 14
`k=2`: no itemsets.

Cycle 5:
minSupport = 75% or $13.5 \approx 14$
`k=1`: {Q5} has support 14
`k=2`: no itemsets.

Cycle 6:
minSupport = 70% or $12.6 \approx 13$
`k=1`: {Q5} has support 14
`k=2`: no itemsets.

Cycle 7:
minSupport = 65% or $11.7 \approx 12$
`k=1`: {Q5} has support 14, {Q2} has support 12, {Q7} has support 12
`k=2`: no itemsets.

Cycle 8:
minSupport = 60% or $10.8 \approx 11$
`k=1`: {Q5} has support 14, {Q2} has support 12, {Q7} has support 12, {Q8} has support 11.
`k=2`: no itemsets.

At this point, we have found 4 association rules but reached the minimum support threshold of 60%, which is a stopping point for the algorithm so it halts. The rules are as follows:
$\{Q5\} \Rightarrow Target$
$\{Q2\} \Rightarrow Target$
$\{Q7\} \Rightarrow Target$
$\{Q8\} \Rightarrow Target$

Exercise 10

Yes, the problem is linearly separable. First defining the sign function

$$f(w,b) = sign(b + \sum_{i=1}^{n} w_i x_i)$$

where

$$b + \sum_{i=1}^{n} w_i x_i \geq 1 \ \forall x \in Class_1$$

and

$$b + \sum_{i=1}^{n} w_i x_i \leq -1 \ \forall x \in Class_2$$

Using the subset of features `Q4`, `Q7` and `Q12` alone, after removing duplicate rows against the target variable, the following function is defined to separate the elections:
$f(Q4, Q7, Q12) = 1 + 2 * Q7 - 2 * Q12 - 4 * Q4$
The data preparation and application of the function are demonstrated below:

```r
pres_data <- read.csv("USPresidency.csv")
lin_sep <- pres_data[,c("Year", "Q4", "Q7", "Q12", "Target")]
lin_sep <- unique(lin_sep[,c("Q4", "Q7", "Q12", "Target")])

lin_sep$Output <- 1 + 2 * lin_sep$Q7 - 2 * lin_sep$Q12 - 4 * lin_sep$Q4
print(lin_sep)
```

```
##    Q4 Q7 Q12 Target Output
## 1   0  0   0      1      1
## 2   0  1   0      1      3
## 4   1  1   0      1     -1
## 6   0  1   1      1      1
## 19  1  1   0      0     -1
## 20  1  0   0      0     -3
## 22  0  0   1      0     -1
## 27  1  0   1      0     -5
## 31  1  1   1      0     -3
```

Comparing the `Target` and `Output` columns, we can see that the groups have been discriminated by their sign, where Challenger victories are output negative from the function and Incumbent victories are classified as positive, with one exception. The year 1880 corresponds to the third row of this table, in which both targets are achieved for the same combination of questions. The function provided classifies this single sample in the Challenger class.
Drawing comparison to the sign function, we have:

$$1 + \begin{bmatrix} -4 & -4 & -4 \\ 2 & 2 & 2 \\ -2 & -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \geq 1 \ \forall x \in Incumbent$$

$$1 + \begin{bmatrix} -4 & -4 & -4 & -4 & -4 \\ 2 & 2 & 2 & 2 & 2 \\ -2 & -2 & -2 & -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \leq -1 \ \forall x \in Challenger$$

Exercise 11

Exercise 12

12. a)

Cross-validation is a method of validating model performance. It involves repeatedly 'holding out' a test set as the model is trained on training data, allowing evaluation to be performed on previously unseen samples that were not used in estimating the model and aggregating results across all repetitions to give a final performance evaluation. While there are many variants of cross-validation, K-fold cross-validation is the most popular. It involves creating k (where k is a user-defined parameter) equally-sized subsets or 'folds' of the data, sampled without replacement. For each round of the method, one such fold is selected to be the test data, and the remaining folds form the training set. This process is repeated k times, choosing a new subset for testing each time; in this way, each observation is used for testing only once. The results of all rounds are then averaged to produce a measure of model validation. The value chosen for k corresponds with the bias-variance tradeoff, and should be examined case by case. 5-Fold CV and 10-Fold CV are both popular choices of k, empirically shown to work well with this tradeoff.

12. b)

Bootstrapping is resampling method of model validation, in which the training set is sampled with replacement, meaning that after an observation is selected it is placed back in the pool being drawn from when building a bootstrap sample. This process is repeated until the sample is the desired size. Any remaining observations become the test set, which the model estimated by the training set is evaluated on. This may be repeated for multiple training/test splits and the results aggregated. The size of the sample and how many times the process will be repeated are user-defined parameters.

12. c)

Imputation is the process of 'filling in' missing values in a dataset, using one of various replacement techniques. The means of doing so must be considered on a case-by-case basis: some algorithms can handle missing values, others cannot. Often, the easiest (and most common) way of handling incomplete data is simply deleting all affected samples, but this is not necessarily appropriate or encouraged, particularly when the sample size is already limited, as it is a potential source of bias. Some examples of techniques used to replace missing data include mean (or median) substitution, where the mean/median of all present values for the feature in question is computed and used to fill in the gaps. While relatively simple, this can clearly only be used on numerical data and is not particularly accurate (doesn't account for column-wise correlation). Other techniques include using the mode of the data, computing a regression model to predict the missing values, or 'hot-deck' (randomly choosing the value from a similar observation).