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")
```

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 and by column
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):
```

```python
    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:

```python
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.

```python
# 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)
## 48    0.0                5.3  ...                1.5               0.2
## 73    1.0                6.1  ...                4.7               1.2
## 36    0.0                5.5  ...                1.3               0.2
## 21    0.0                5.1  ...                1.5               0.4
```

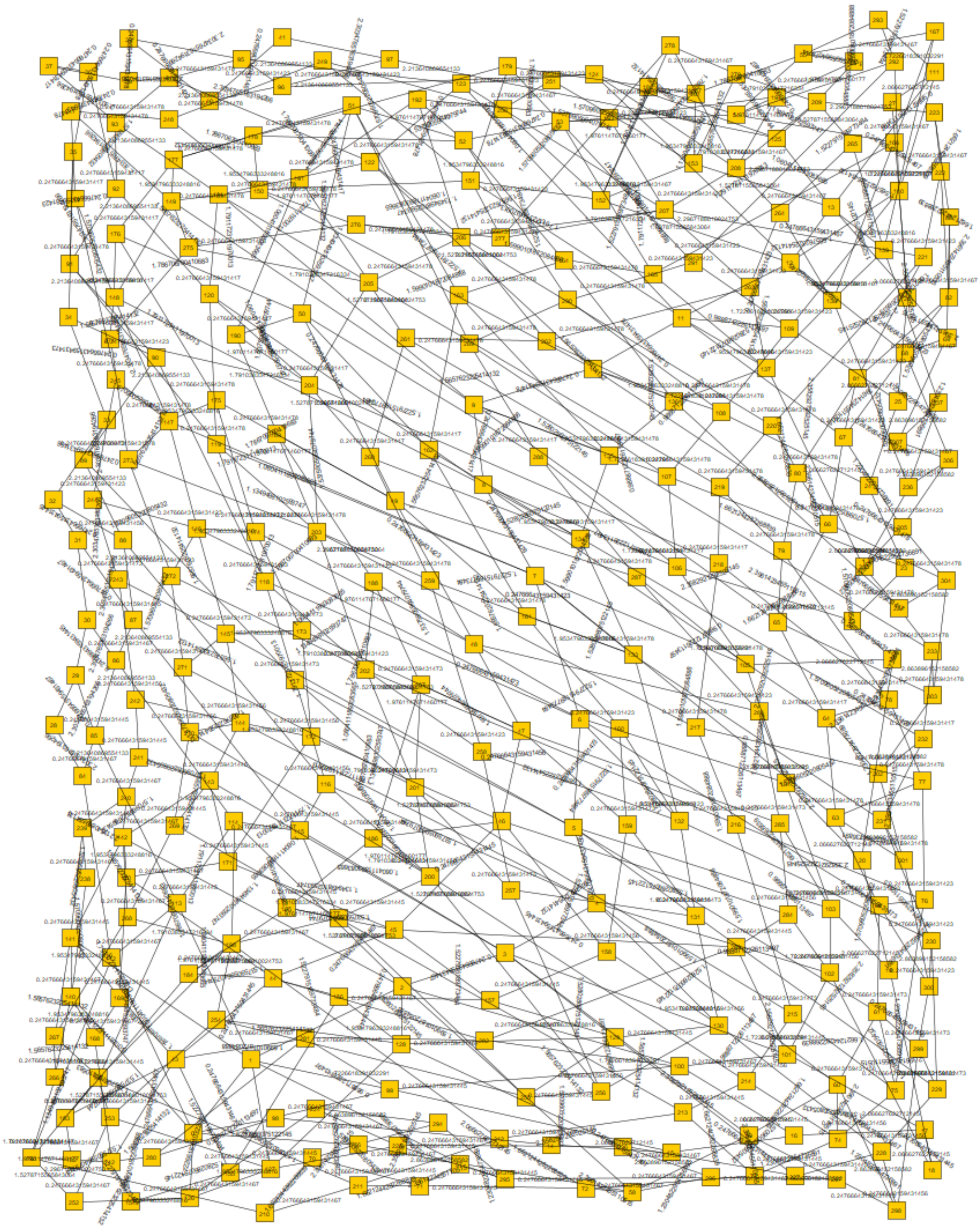


Figure 1: Relative Neighbourhood Graph for samples

```
## 37    0.0                4.9  ...               1.4              0.1
##
## [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, summa
                                                                   fromLast = TRUE)]
head(train_data)
```

```
##   sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) CLASS
## 1     '(-inf-5.55]'     '(3.45-inf)'     '(-inf-2.45]'     '(-inf-0.8]'     0
## 2      '(5.55-inf)'    '(-inf-3.45]'     '(2.45-4.85]'     '(0.8-1.75]'     1
## 3     '(-inf-5.55]'     '(3.45-inf)'     '(-inf-2.45]'     '(-inf-0.8]'     0
## 4     '(-inf-5.55]'     '(3.45-inf)'     '(-inf-2.45]'     '(-inf-0.8]'     0
## 5     '(-inf-5.55]'     '(3.45-inf)'     '(-inf-2.45]'     '(-inf-0.8]'     0
## 6     '(-inf-5.55]'    '(-inf-3.45]'     '(2.45-4.85]'     '(0.8-1.75]'     1
```

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
  }
}
```

```
## [1] "0"
## [1] "'(-inf-5.55]'" "'(5.55-inf)'"
## [1] "2"
## [1] "'(-inf-5.55]'" "'(5.55-inf)'"
## [1] "1"
## [1] "'(-inf-3.45]'" "'(3.45-inf)'"
## [1] "0"
## [1] "'(-inf-3.45]'" "'(3.45-inf)'"
## [1] "0"
## [1] "'(-inf-2.45]'" "'(2.45-4.85]'" "'(4.85-inf)'"
## [1] "1"
## [1] "'(-inf-2.45]'" "'(2.45-4.85]'" "'(4.85-inf)'"
## [1] "2"
## [1] "'(-inf-2.45]'" "'(2.45-4.85]'" "'(4.85-inf)'"
## [1] "0"
## [1] "'(-inf-0.8]'" "'(0.8-1.75]'" "'(1.75-inf)'"
```

```
## [1] "1"
## [1] "'(-inf-0.8]'" "'(0.8-1.75]'" "'(1.75-inf)'"
## [1] "2"
## [1] "'(-inf-0.8]'" "'(0.8-1.75]'" "'(1.75-inf)'"
```

```
# 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 $x_I^{(i)} \neq x_I^{(j)}$?

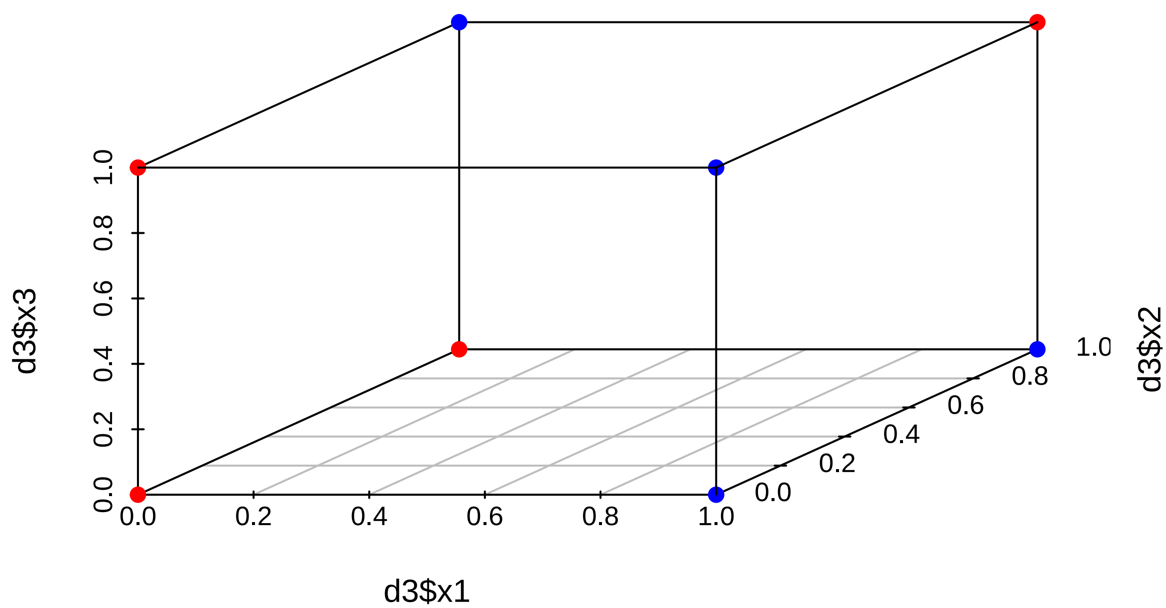
An example of the problem is as follows, where variables `x1`, `x2`, `x3`, `x4` represent characteristics of a student and `y` represents their grade on a test.

```
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

- Use Iris dataset
- Will draw heavily from assignment 1. I have copied in MST and kNN code below, as well as for aggregating results.

```
# 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 MS
#     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
# #one_c_result = get_mst(hd_matrix_r, i_names, 'graph_1c.gml')
#
# 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])
#     nx.write_gml(G, output_file)
#     return G
# #five_result = get_k_NNG(2, hd_matrix_r, i_names, 'graph_5.gml')
#
# MST_edges = list(one_c_result.edges)
# kNNG_edges = list(five_result.edges)
```

```
# common_edges = list(set(MST_edges).intersection(kNNG_edges))
#
# # yEd output
# G = nx.Graph()
# G.add_nodes_from(i_names)
# G.add_edges_from(common_edges)
# nx.write_gml(G, 'graph_7.gml')
#
# # tabulate the clusters
# cluster = []
# [cluster.append(sorted(list(nx.node_connected_component(G, n)))) for n in i_names]
# 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', 'elections'])
```

5. a)

5. b)

5. c)

5. d)

Exercise 6

6. a)

6. b)

6. c)

Exercise 7

7. a)

7. b)

Exercise 8

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

Exercise 9

9. a)

```r
pres_data <- read.csv("USPresidency.csv")
pres_data[pres_data$Target == 0,]
```

```
##    Year Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Target
## 19 1860  1  0  1  1  0  0  1  0  1   0   0   0      0
## 20 1876  1  1  0  1  0  1  0  0  0   1   0   0      0
## 21 1884  1  0  0  1  0  0  1  0  1   0   1   0      0
## 22 1892  0  0  1  0  1  0  0  1  1   0   0   1      0
## 23 1896  0  0  0  1  0  1  0  1  1   0   1   0      0
## 24 1912  1  1  1  1  1  0  1  0  0   0   0   0      0
## 25 1920  1  0  0  1  0  1  0  1  1   0   0   0      0
## 26 1932  1  1  0  0  1  1  0  0  1   0   0   1      0
## 27 1952  1  0  0  1  0  0  0  0  0   1   0   1      0
## 28 1960  1  1  0  0  0  1  0  0  0   0   0   1      0
## 29 1968  1  1  1  1  0  0  1  1  1   0   0   0      0
## 30 1976  1  1  0  1  1  0  0  0  0   1   0   0      0
## 31 1980  0  0  1  1  1  1  1  0  0   1   0   1      0
```

```r
pres_data[pres_data$Target == 1,]
```

```
##    Year Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Target
## 1  1864  0  0  0  0  1  0  0  1  1   0   0   0      1
## 2  1868  1  1  0  0  0  0  1  1  1   0   1   0      1
## 3  1872  1  1  0  0  1  0  1  0  0   0   1   0      1
## 4  1880  1  0  0  1  0  0  1  1  0   0   0   0      1
## 5  1888  0  0  0  0  1  0  0  0  0   0   0   0      1
## 6  1900  0  1  0  0  1  0  1  0  0   0   0   1      1
## 7  1904  1  1  0  0  1  0  1  0  0   0   1   0      1
## 8  1908  1  1  0  0  0  1  0  1  0   0   0   0      1
## 9  1916  0  0  0  0  1  0  0  1  0   0   0   0      1
## 10 1924  0  1  1  0  1  0  1  1  0   1   0   0      1
## 11 1928  1  1  0  0  0  0  1  0  0   0   0   0      1
## 12 1936  0  1  0  0  1  1  1  1  0   0   1   0      1
## 13 1940  1  1  0  0  1  1  1  1  0   0   1   0      1
## 14 1944  1  1  0  0  1  0  1  1  0   0   1   0      1
## 15 1948  1  1  1  0  1  0  0  1  0   0   0   0      1
## 16 1956  0  1  0  0  1  0  1  0  0   0   1   0      1
## 17 1964  0  0  0  0  1  0  1  0  0   0   0   0      1
## 18 1972  0  0  0  0  1  0  0  1  1   0   0   0      1
```

9. b)

Exercise 10

Exercise 11

Exercise 12

12. a)

12. b)

12. c)