Assignment 7

Loading the required libraries

```
library(dendextend)
```

```
##
## ---------------------
## Welcome to dendextend version 1.5.2
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------
```

```
##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(class)
library(arules)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'arules'
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

Question 1

Part a : Clustering the given dataset

Cluster the dataset into 10 clusters, each of which represents a digit.
Function : kmeans()
Parameters :
1. n = 10 as number of clusters is the same as number of digits.
2. nstart = 20 i.e 20 different starting assignments will be tested and the one with lowest will be seleted withing cluster variaion.

```
data = read.csv("optdigits.csv")
mdata = subset(data, select = -c(digit))

set.seed(10)    # ensure reproducibility
```

```r
digits_cluster = kmeans(mdata, 10, nstart = 20,  iter.max = 200)
tab = table(digits_cluster$cluster, data$digit)
print("T1 : Table containing instances of each digit in a cluster")
```

```
## [1] "T1 : Table containing instances of each digit in a cluster"
```

```r
print(tab)
```

```
##
##         0   1   2   3   4   5   6   7   8   9
##    1    1 113   0   5  30   6   0   6   5  97
##    2    0  15 329   5   0   0   0   0   0   0
##    3    0 250   0   2   6   0   3   5  29   2
##    4    0   0   4  10  29   0   0 373   1  24
##    5    0   1   0   4   7 289   0   0   3   1
##    6  373   0   0   0   0   0   0   0   0   0
##    7    1   1   1   0   4   1 373   0   4   0
##    8    1   0   0   0 306   0   1   0   0   0
##    9    0   9  19 346   0  80   0   0   9 256
##    10   0   0  27  17   5   0   0   3 329   2
```

```r
inittable = data.frame("Cluster Number" = c(), "Digit" = c())

for(x in 1:10){
    digit = which.max(tab[x, ])[[1]] - 1
    inittable = rbind(inittable, data.frame("Cluster Number" = c(x), "Digit" = c(digit)))
}
print("T2 : Table containing cluster number and the digit it corresponds to")
```

```
## [1] "T2 : Table containing cluster number and the digit it corresponds to"
```

```r
print(inittable)
```

```
##    Cluster.Number Digit
## 1               1     1
## 2               2     2
## 3               3     1
## 4               4     7
## 5               5     5
## 6               6     0
## 7               7     6
## 8               8     4
## 9               9     3
## 10             10     8
```

Part b : Hierarchical clustering for cluster 1

From T1, we can see that cluster 1 has an almost equal distribution of digit 9 (97 instances) and digit 1 (113 instances).
We perform hierarchical clustering on this cluster to try distinguish between the two digits.
First, we extract all the instances that belong to cluster 1.

```r
hdata = data.frame()     # Data frame for all the values in cluster 1
for(x in 1:nrow(data)){
    if(digits_cluster$cluster[x] == 1){
        hdata = rbind(hdata, data.frame(data[x,]))
```

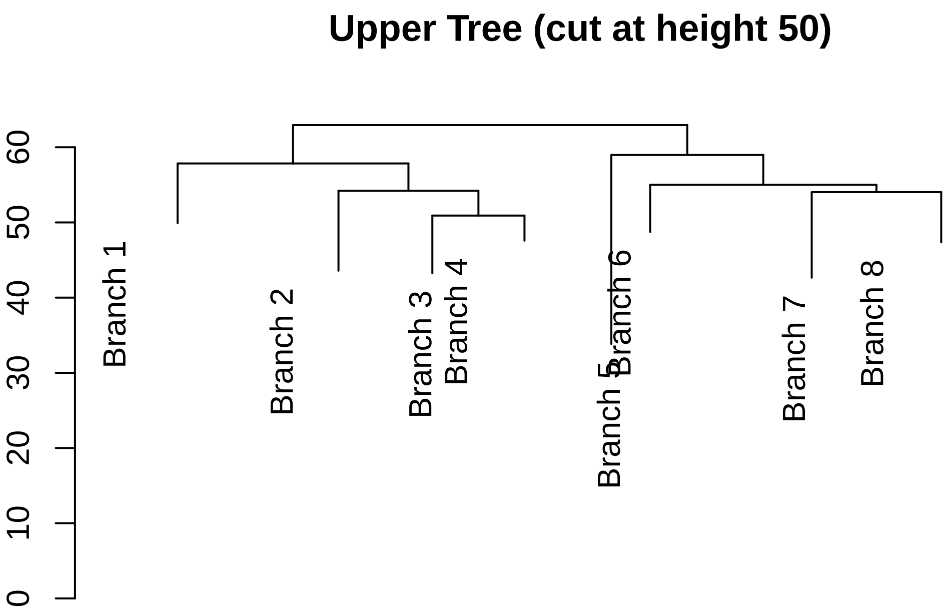
```
    }
}
ndata = subset(hdata, select=-c(digit))
```

Next, we build a dendrogram after applying the hierarchical clustering algorithm.

```
ndata = subset(hdata, select=-c(digit))
h_clusters = hclust(dist(ndata))    # hierarchical clustering
newcut_f = as.dendrogram(h_clusters)  # create dendrogram
```

Cut the dendrogram above the height 50.

```
plot(cut(newcut_f, h = 50)$upper, main = "Upper Tree (cut at height 50)")
```



We then bring down the number of clusters formed by hierarchical clustering down to 2.

```
newcut_t = cutree(h_clusters, 2)
tab2 = table(newcut_t, hdata$digit)
print("T3 : Hierarchical clustering on cluster 1")
```

```
## [1] "T3 : Hierarchical clustering on cluster 1"
```

```
print(tab2)
```

```
##
## newcut_t   0   1   3   4   5   7   8   9
##        1   1   2   5  25   6   2   2  96
##        2   0 111   0   5   0   4   3   1
```

From the above table, we can see that cluster 1 has more instances of digit 9 (96 instances) and cluster 2 has a majority of digit 1 (111 instances).

```
hcluster = data.frame("Cluster Number" = c(1,2), "Digit" = c(9,1))
print("Cluster Number with corresponding label")
```

```
## [1] "Cluster Number with corresponding label"
```

```
print(hcluster)
```

```
##   Cluster.Number Digit
## 1              1     9
## 2              2     1
```

Using the above table, the number of instances in each cluster are,

```
hcluster = data.frame("Cluster Number" = c(1,2),
                      "Digit" = c(9,1),
                      "Number of instances" = c(sum(tab2[1,]),
                                                 sum(tab2[2,])))
print("Cluster Number with corresponding label")
```

```
## [1] "Cluster Number with corresponding label"
```

```
print(hcluster)
```

```
##   Cluster.Number Digit Number.of.instances
## 1              1     9                 139
## 2              2     1                 124
```

Part c : Classifying the test data

1. Load the given test data.
2. Calculate the distance of each vector to the center of the clusters and identify the digit.

```
test_data = read.csv("optdigits_test.csv")   # read input from csv
centers = digits_cluster$centers    # get the matrix of cluster centers

test_result = data.frame("Image Number" = c(), "Cluster Number" = c(), "Digit" = c())

for(x in 1:nrow(test_data)){
    d = c()
    for(y in 1:10){
      # calculate eucledian distance from each center
      d = c(d, dist(rbind(subset(test_data, select=-c(imageno))[x,], centers[y,])))
    }
    # find the cluster number which has minimum distance
    cluster_number = which.min(d)
    # find the digit corresponding to that cluster
    digit = which.max(tab[cluster_number, ])[[1]] - 1
    test_result = rbind(test_result,
                        data.frame("Image Number" = c(test_data$imageno[x]),
                                   "Cluster Number" = c(cluster_number),
                                   "Digit" = c(digit)))
}
print(test_result)  # print the result
```

```
##    Image.Number Cluster.Number Digit
## 1             1              9     3
## 2             2              3     1
## 3             3              6     0
## 4             4              1     1
## 5             5              2     2
## 6             6              4     7
## 7             7              8     4
## 8             8              5     5
```

```
## 9               9              7     6
## 10             10             10     8
## 11             11              9     3
## 12             12              6     0
## 13             13              1     1
## 14             14              2     2
## 15             15              9     3
## 16             16              8     4
## 17             17              5     5
## 18             18              7     6
## 19             19              4     7
## 20             20             10     8
```

Part d : Classifying the test data using kNN

From part a and b, it is clear that there is an ambiguity when it comes to cluster 1 since it has an almost equal distribution of the digits 1 and 9.
Since cluster 3 has a majority of digit 1, we can label cluster 1 with the digit 9 and cluster 3 with digit 1.
After correct labelling of clusters we have,

```
# table containing correct labels corresponding to digit
clustable = data.frame("Cluster Number" = c(), "Digit" = c())
for(x in 1:10){
    if(x == 1){     # if cluster number is 1, corresponding digit is 9
        digit = 9
    }
    else{
        digit = which.max(tab[x, ])[[1]] - 1
    }
    clustable = rbind(clustable, data.frame("Cluster Number" = c(x),
                                            "Digit" = c(digit)))
}
print(clustable)    # print the table
```

```
##    Cluster.Number Digit
## 1               1     9
## 2               2     2
## 3               3     1
## 4               4     7
## 5               5     5
## 6               6     0
## 7               7     6
## 8               8     4
## 9               9     3
## 10             10     8
```

```
df_instances = data.frame("Label (Cluster Number)" = c(),
                          "Number of data points" = c())
instances = c(0,0,0,0,0,0,0,0,0,0)
for(x in 1:nrow(data)){
    instances[data$digit[x] + 1] = instances[data$digit[x] + 1] + 1
}
for(x in 1:10){
    df_instances = rbind(df_instances,
```

```r
                          data.frame("Label (Cluster Number)" = c(x),
                                     "Number of data points" = c(instances[x])))
}
print("Number of instances under each label")
```

```
## [1] "Number of instances under each label"
```

```r
print(df_instances)
```

```
##    Label..Cluster.Number. Number.of.data.points
## 1                       1                   376
## 2                       2                   389
## 3                       3                   380
## 4                       4                   389
## 5                       5                   387
## 6                       6                   376
## 7                       7                   377
## 8                       8                   387
## 9                       9                   380
## 10                     10                   382
```

We first apply the correct labels to the entire training set.

```r
classtab = data     # make a copy of data
for(x in 1:nrow(data)){
    cluster_number = clustable[which(clustable$Digit == data$digit[x]),]$Cluster.Number
    classtab$cluster_number[x] = cluster_number
}
newtab = subset(classtab, select=-c(digit, cluster_number))
```

Applying kNN algorithm we get,

```r
fitknn = knn(train = newtab,test = subset(test_data, select=-c(imageno)),
             cl = classtab$cluster_number, k = 7)  # fit data using knn
knn_result = data.frame("Image Number" = c(),
                        "Cluster Number" = c(),
                        "Digit" = c())
for(x in 1:nrow(test_data)){
    if(fitknn[x] == 1){
        digit = 9
    }
    else{
      # find the digit corresponding to that cluster
        digit = which.max(tab[fitknn[x], ])[[1]] - 1
    }
    knn_result = rbind(knn_result, data.frame("Image Number" = c(test_data$imageno[x]),
                                              "Cluster Number" = c(fitknn[x]),
                                              "Digit" = c(digit)))
}
print(knn_result)   # print knn result
```

```
##    Image.Number Cluster.Number Digit
## 1             1              1     9
## 2             2              3     1
## 3             3              6     0
## 4             4              3     1
```

```
## 5              5              2     2
## 6              6              9     3
## 7              7              3     1
## 8              8              5     5
## 9              9              7     6
## 10            10             10     8
## 11            11              1     9
## 12            12              6     0
## 13            13              3     1
## 14            14              2     2
## 15            15              9     3
## 16            16              8     4
## 17            17              5     5
## 18            18              7     6
## 19            19              4     7
## 20            20             10     8
```

Question 2

Read the input data

```
data = read.csv("handwriting_recognition.csv")
```

To be able to use the apriori() function in the arules library, the data should be in the form of transactions and not a data frame.
We must first create a new csv file by dropping the columns X and Frequency and replicating each row "Freq" number of times.
This is to make sure each row corresponds to a single transaction.

```
ndata = data.frame("Gender" = c(), "Profession" = c(), "Recognition" = c())
for(x in 1:nrow(data)){
  f = data$Freq[x]
  while(f != 0){
    ndata = rbind(ndata, data.frame("Gender" = c(toString(data$Gender[x])),
                                    "Profession" = c(toString(data$Profession[x])),
                                    "Recognition" = c(toString(data$Recognition[x]))))
    f = f - 1
  }
}

write.csv(ndata, file = "ndata.csv")
data = read.transactions("ndata.csv", sep = ",")     # read data as transactions
```

Next, we apply the apriori() function with default parameters to find the association rules.

```
rules_default = apriori(data)
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target   ext
##      10  rules FALSE
##
```

```
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE   FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 452
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[4539 item(s), 4527 transaction(s)] done [0.00s].
## sorting and recoding items ... [10 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [4 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(rules_default)
```

```
##     lhs                      rhs            support   confidence lift
## [1] {Engineer}           => {Male}         0.1237022 0.9572650  1.610382
## [2] {Teacher}            => {Unrecognized} 0.1475591 0.9355742  1.528453
## [3] {Artist}             => {Male}         0.1822399 0.8842444  1.487542
## [4] {Artist,Recognized} => {Male}         0.1130992 0.8519135  1.433152
##     count
## [1] 560
## [2] 668
## [3] 825
## [4] 512
```

Applying the apriori() function with the following parameters to obtain the association rules which contain the “recognition” in the RHS we get,

```
rules_recognition = apriori(data,parameter = list( support = 0.01,
                                                   confidence = 0.7,
                                                   minlen = 2,
                                                   maxlen = 5),
                            appearance = list( rhs = c('Recognized', 'Unrecognized'),
                                               default = 'lhs'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.7    0.1    1 none FALSE            TRUE       5    0.01      2
##  maxlen target   ext
##       5  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE   FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 45
##
## set item appearances ...[2 item(s)] done [0.00s].
## set transactions ...[4539 item(s), 4527 transaction(s)] done [0.00s].
## sorting and recoding items ... [10 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
```

```
## writing ... [7 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(rules_recognition)
```

```
##     lhs                  rhs            support    confidence lift
## [1] {Doctor}          => {Unrecognized} 0.09653192 0.7482877  1.222482
## [2] {Teacher}         => {Unrecognized} 0.14755909 0.9355742  1.528453
## [3] {Doctor,Female}   => {Unrecognized} 0.06604816 0.7608142  1.242947
## [4] {Doctor,Male}     => {Unrecognized} 0.03048376 0.7225131  1.180374
## [5] {Female,Teacher} => {Unrecognized} 0.07002430 0.9296188  1.518724
## [6] {Male,Teacher}    => {Unrecognized} 0.07753479 0.9410188  1.537348
## [7] {Artist,Female}  => {Recognized}   0.01965982 0.8240741  2.125689
##     count
## [1] 437
## [2] 668
## [3] 299
## [4] 138
## [5] 317
## [6] 351
## [7]  89
```

Applying the apriori() function with the following parameters to obtain the association rules which contain "Gender" in the rhs we get,

```
rules_gender <- apriori(data,parameter = list( support = 0.01,
                                               confidence = 0.6,
                                               minlen = 2,
                                               maxlen = 5),
                        appearance = list( rhs = c( 'Male', 'Female'),
                                           default = 'lhs'))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5    0.01      2
##  maxlen target   ext
##       5  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 45
##
## set item appearances ...[2 item(s)] done [0.00s].
## set transactions ...[4539 item(s), 4527 transaction(s)] done [0.00s].
## sorting and recoding items ... [10 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 done [0.00s].
## writing ... [13 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
inspect(rules_gender)
```

```
##      lhs                           rhs      support    confidence lift
```

```
## [1]  {Doctor}                => {Female} 0.08681246 0.6729452  1.660176
## [2]  {Engineer}              => {Male}   0.12370223 0.9572650  1.610382
## [3]  {Actor}                 => {Female} 0.13099183 0.6459695  1.593626
## [4]  {Artist}                => {Male}   0.18223989 0.8842444  1.487542
## [5]  {Recognized}            => {Male}   0.26463442 0.6826211  1.148356
## [6]  {Doctor,Recognized}     => {Female} 0.02076430 0.6394558  1.577557
## [7]  {Doctor,Unrecognized}   => {Female} 0.06604816 0.6842105  1.687968
## [8]  {Engineer,Recognized}   => {Male}   0.07797658 0.9540541  1.604981
## [9]  {Engineer,Unrecognized} => {Male}   0.04572565 0.9627907  1.619678
## [10] {Actor,Recognized}      => {Female} 0.04462116 0.6273292  1.547640
## [11] {Actor,Unrecognized}    => {Female} 0.08637066 0.6560403  1.618471
## [12] {Artist,Recognized}     => {Male}   0.11309918 0.8519135  1.433152
## [13] {Artist,Unrecognized}   => {Male}   0.06914071 0.9427711  1.586000
##      count
## [1]   393
## [2]   560
## [3]   593
## [4]   825
## [5]  1198
## [6]    94
## [7]   299
## [8]   353
## [9]   207
## [10]  202
## [11]  391
## [12]  512
## [13]  313
```

RESULT

Extracting the required results form the output we have,

Association rules with default settings

#	Rule	Support	Confidence	Lift
1	{Engineer} => {Male}	0.1237022	0.9572650	1.610382
2	{Teacher} => {Unrecognized}	0.1475591	0.9355742	1.528453
3	{Artist} => {Male}	0.1822399	0.8842444	1.487542
4	{Artist,Recognized} => {Male}	0.1130992	0.8519135	1.433152

Answers

#	Rule	Support	Confidence	Lift
1	{Artist,Female} => {Recognized}	0.01965982	0.8240741	2.125689
2	{Engineer} => {Male}	0.1237022	0.9572650	1.610382
3	{Actor,Recognized} => {Female}	0.04462116	0.6273292	1.547640
4	{Doctor,Male} => {Unrecognized}	0.03048376	0.7225131	1.180374
