INFO411 Assignment 1

Name: Le Minh Vu

Student ID: 7558909

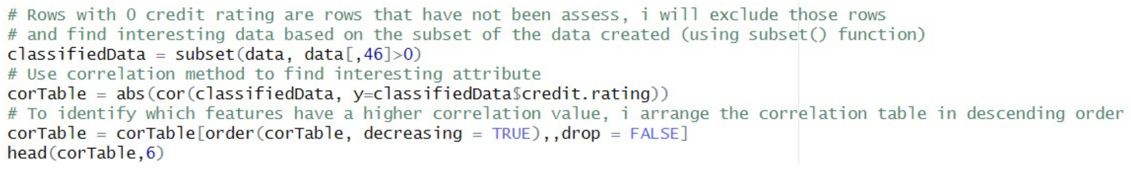
Preface:

Banks are often posed with a problem to whether or not a client is credit worthy. Banks commonly employ data mining techniques to classify a customer into risk categories such as category A (highest rating) or category C (lowest rating).

Question 1:

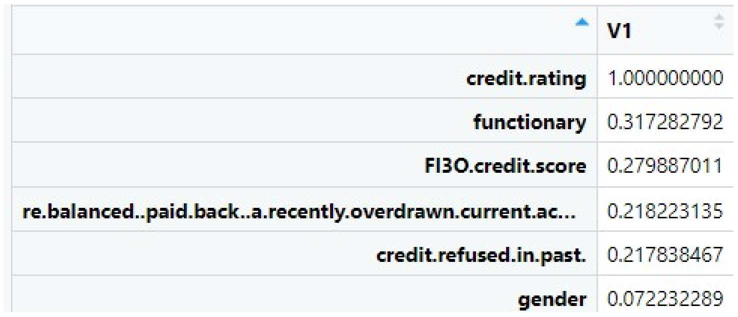
Firstly, rows with credit rating value of 0 are excluded for further analysis. The filtered data is then stored in "classifiedData" using the subset function. The correlation between all variables (including the target variable "credit.rating") is calculated using the cor function.

The absolute value of the correlation table is taken so that both positive and negative correlations are considered. The resulting correlation table is ordered in a descending order to identify the features with the highest correlation with the target variable.



```
# Rows with 0 credit rating are rows that have not been assess, i will exclude those rows
# and find interesting data based on the subset of the data created (using subset() function)
classifiedData = subset(data, data[,46]>0)
# Use correlation method to find interesting attribute
corTable = abs(cor(classifiedData, y=classifiedData$credit.rating))
# To identify which features have a higher correlation value, i arrange the correlation table in descending order
corTable = corTable[order(corTable, decreasing = TRUE),,drop = FALSE]
head(corTable,6)
```

The top 6 rows of the sorted correlation table are then displayed using the head function. The below snapshot showed the content of the corTable.



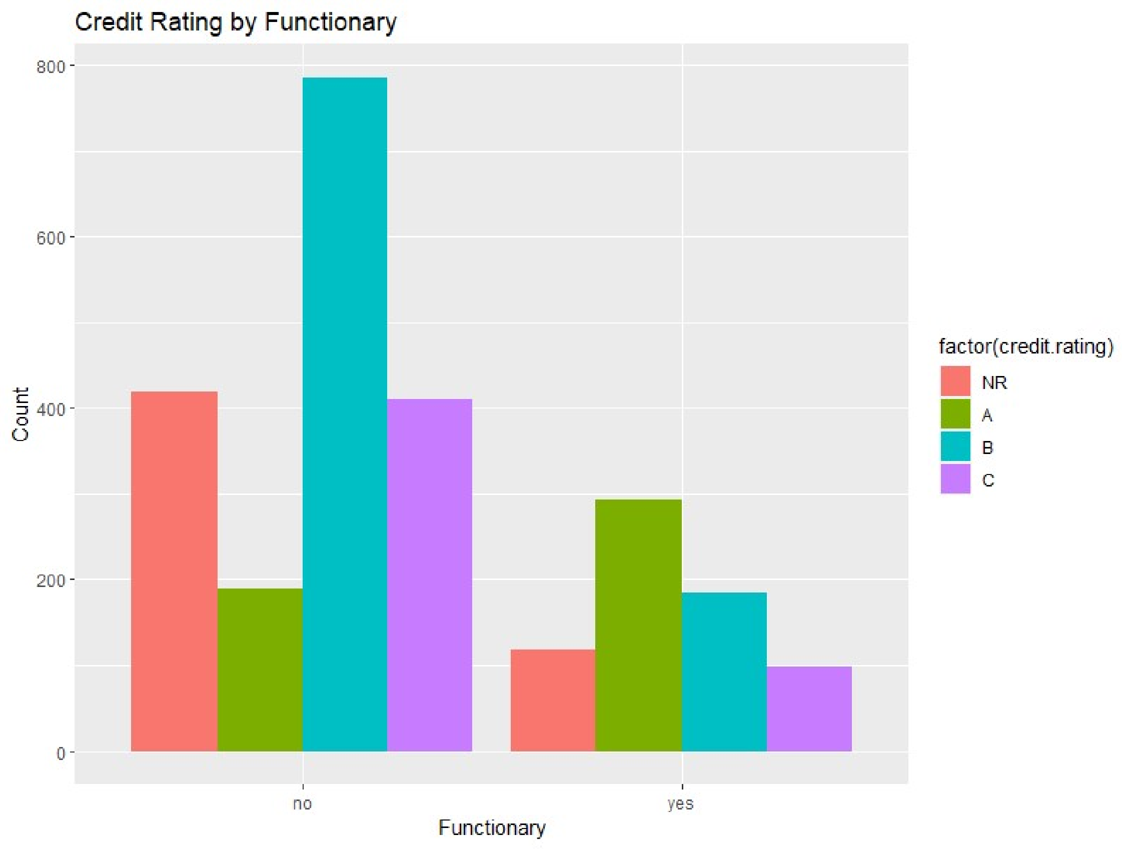
	V1
credit.rating	1.000000000
functionary	0.317282792
FI3O.credit.score	0.279887011
re.balanced..paid.back..a.recently.overdrawn.current.ac...	0.218223135
credit.refused.in.past.	0.217838467
gender	0.072232289

From this result, I’ve considered the attributes “functionary”, “FICO credit score”, “re-balanced (paid back) a recently overdrawn current account”, “credit refused in the past” and “gender” to be interesting and further analyse them with credit rating.

1st pair: credit rating vs functionary

```
##1
ggplot(data_plot, aes(x=functionary, fill=factor(credit.rating))) +
  geom_bar(position="dodge") +
  xlab("Functionary") +
  ylab("Count") +
  ggtitle("Credit Rating by Functionary")
```

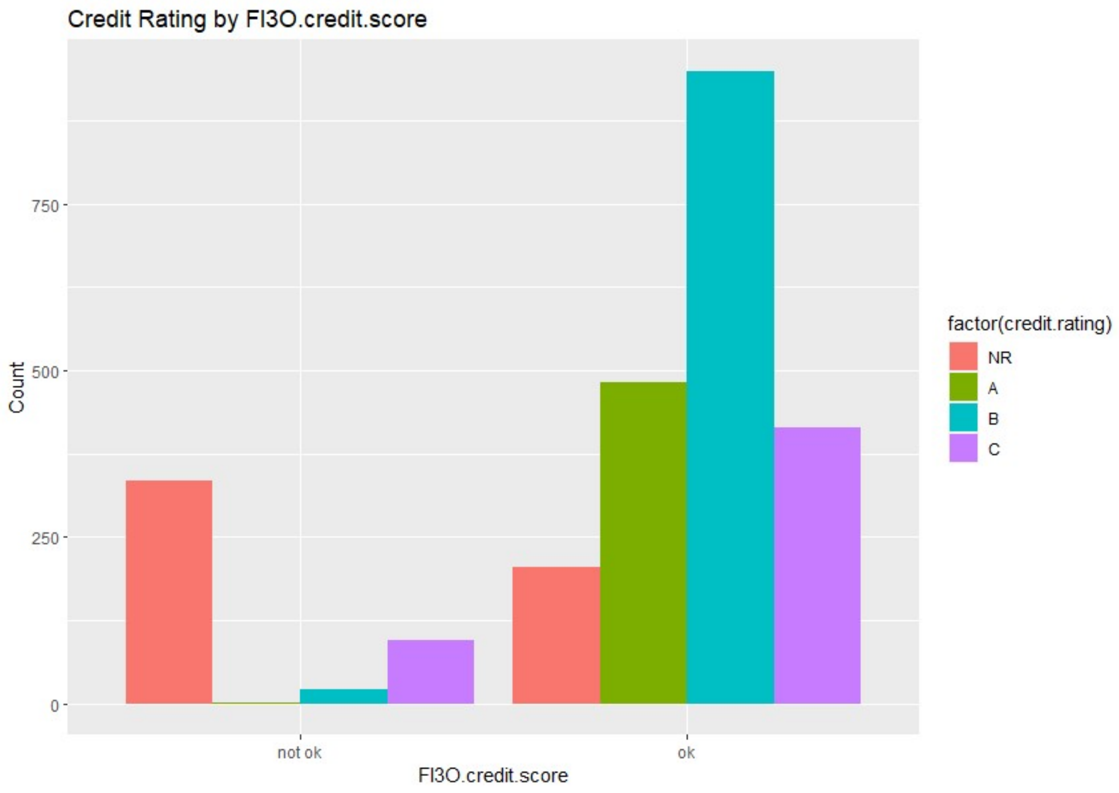
From the bar plot, I can see that the data is quite varied. For example, number of people without functionary have a higher number of B and C credit rating compared to the number of people who have functionary but for A rating, it is the opposite.



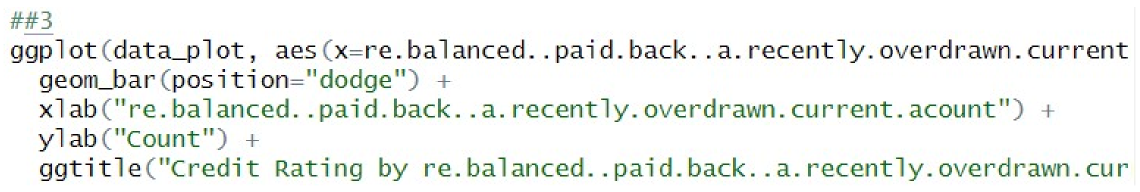
2nd pair: credit rating vs FICO credit score

```
##2
ggplot(data_plot, aes(x=FI3O.credit.score, fill=factor(credit.rating))) +
  geom_bar(position="dodge") +
  xlab("FI3O.credit.score") +
  ylab("Count") +
  ggtitle("Credit Rating by FI3O.credit.score")
```

From the bar plot below, we can clearly see that majority of people who got their credit assessed comes from people who have a good FICO credit score. Credit rating “B” has the highest among the other ratings from what I see below.

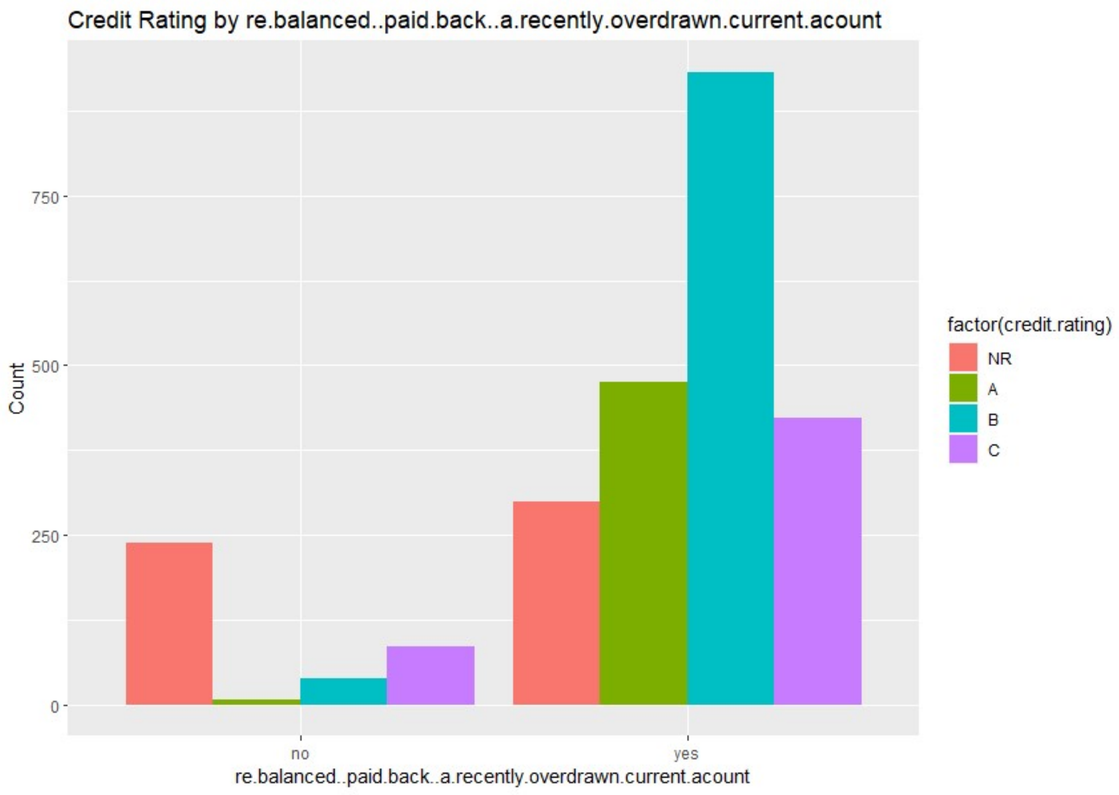


3rd pair: credit rating vs overdraft repayment

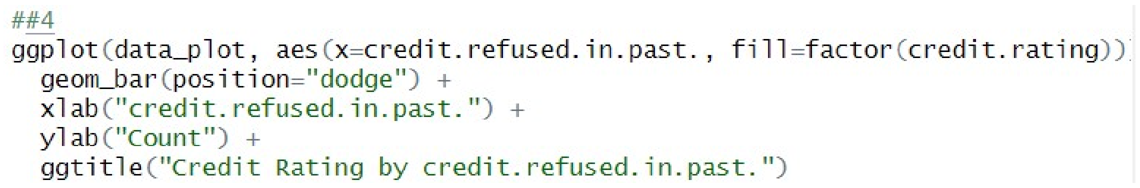


```
##3
ggplot(data_plot, aes(x=re.balanced..paid.back..a.recently.overdrawn.current
  geom_bar(position="dodge") +
  xlab("re.balanced..paid.back..a.recently.overdrawn.current.acount") +
  ylab("Count") +
  ggtitle("Credit Rating by re.balanced..paid.back..a.recently.overdrawn.cur
```

For “re-balanced (paid back) a recently overdrawn current account” or in short term “overdraft repayment” attribute seems to have a similar results comparing to the FICO credit score.

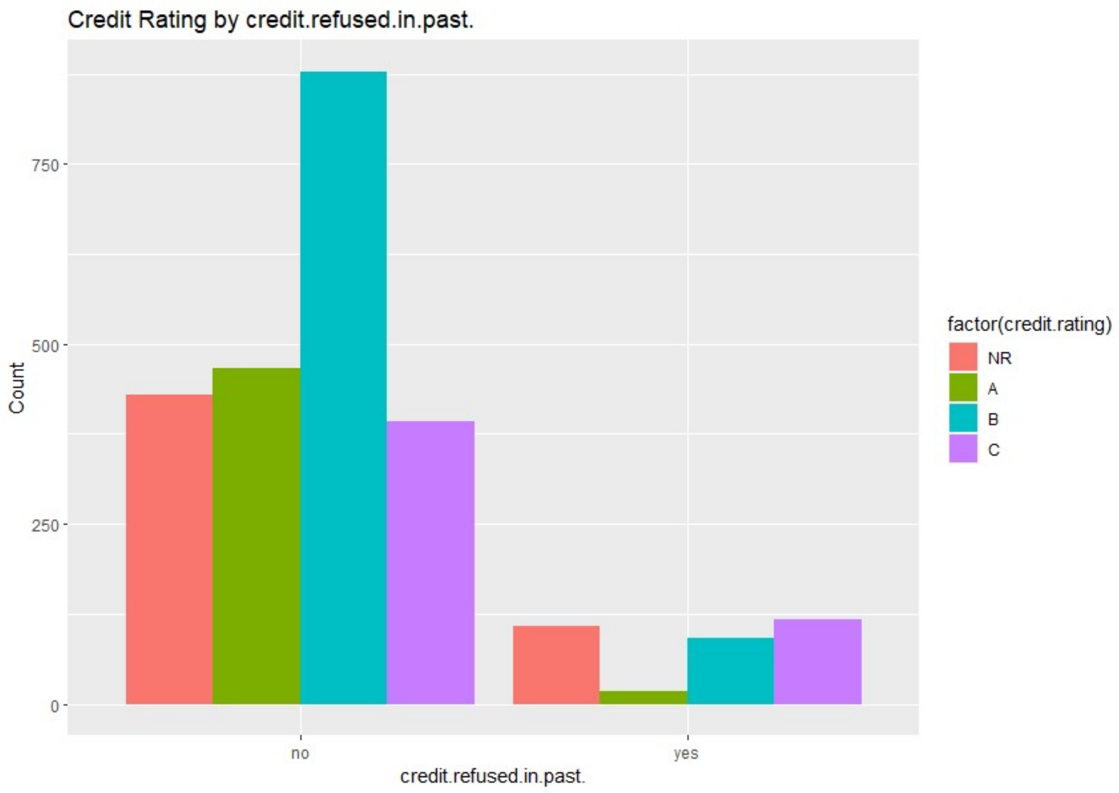


4th pair: credit rating vs credit refused in the past

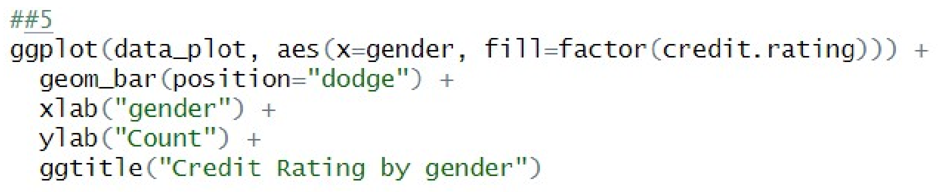


```
##4
ggplot(data_plot, aes(x=credit.refused.in.past., fill=factor(credit.rating))
  geom_bar(position="dodge") +
  xlab("credit.refused.in.past.") +
  ylab("Count") +
  ggtitle("Credit Rating by credit.refused.in.past.")
```

Another interesting data found. Majority of people who get credit rating also has no credit refused in the past. Additionally, credit rating B is still being the highest among other ratings.

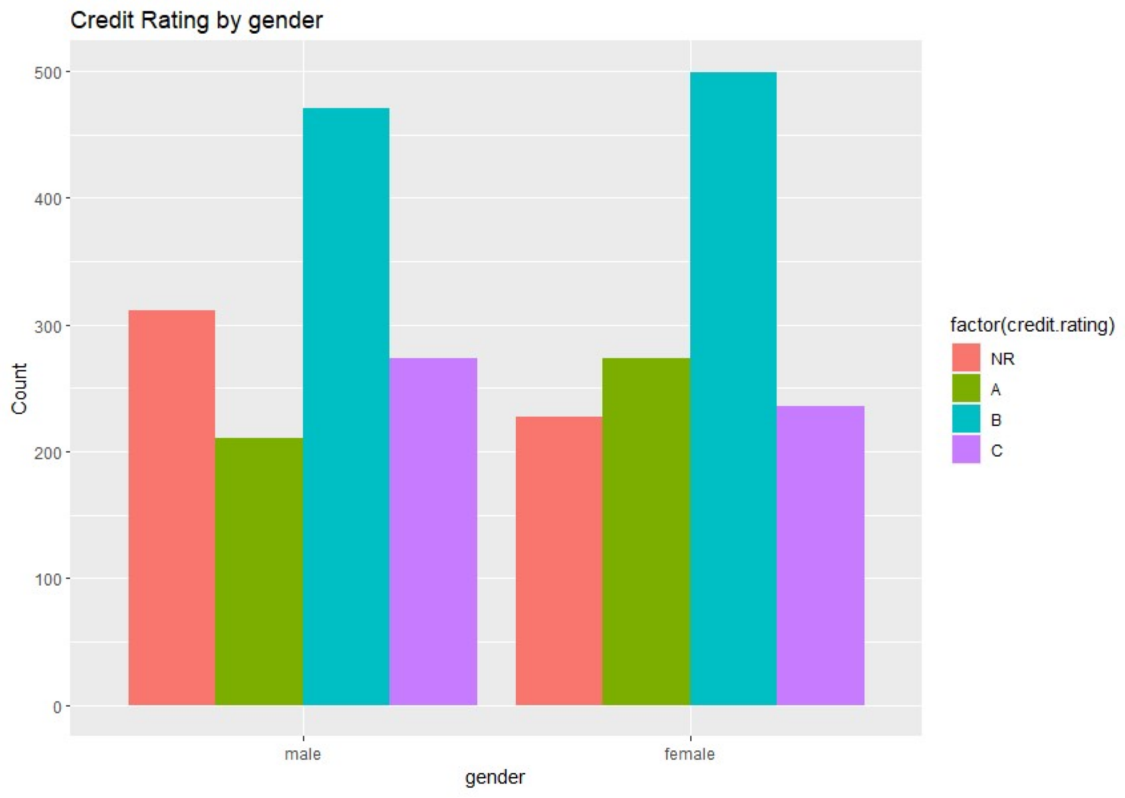


5th pair: credit rating vs gender



```
##5
ggplot(data_plot, aes(x=gender, fill=factor(credit.rating))) +
  geom_bar(position="dodge") +
  xlab("gender") +
  ylab("Count") +
  ggtitle("Credit Rating by gender")
```

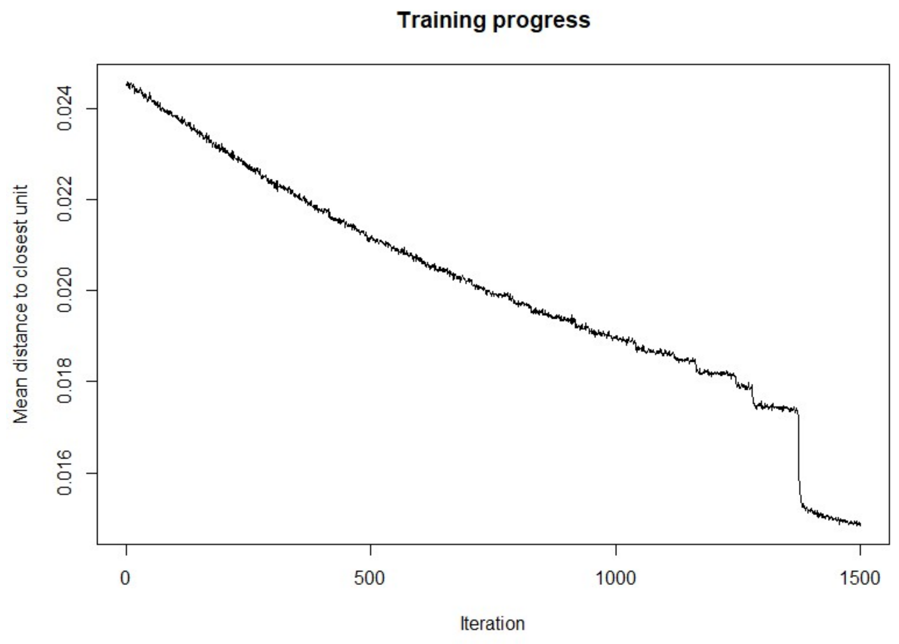
In this case, it looks like male and female are pretty much equally assessed and given credit rating to with B rating being the highest.



SOM training

```
# To train the SOM model, i chose to include all features for better
# insight to the relationship of the features exclude credit rating
data_train = data[, c(1:45)]

# now train the SOM using the Kohonen method
data_train_matrix <- as.matrix(scale(data_train))
names(data_train_matrix) <- names(data_train)
require(kohonen)
x_dim=20
y_dim=20

som_grid <- somgrid(xdim = x_dim, ydim=y_dim, topo="hexagonal")

if (packageVersion("kohonen") < 3){
  system.time(som_model <- som(data_train_matrix,
                               grid=som_grid,
                               rlen=1000,
                               alpha=c(0.9,0.01),
                               n.hood = "circular",
                               keep.data = TRUE ))
}else{
  system.time(som_model <- som(data_train_matrix,
                               grid=som_grid,
                               rlen=1000,
                               alpha=c(0.9,0.01),
                               mode="online",
                               normalizeDataLayers=false,
                               keep.data = TRUE ))
}
summary(som_model)
rm(som_grid, data_train_matrix)
```

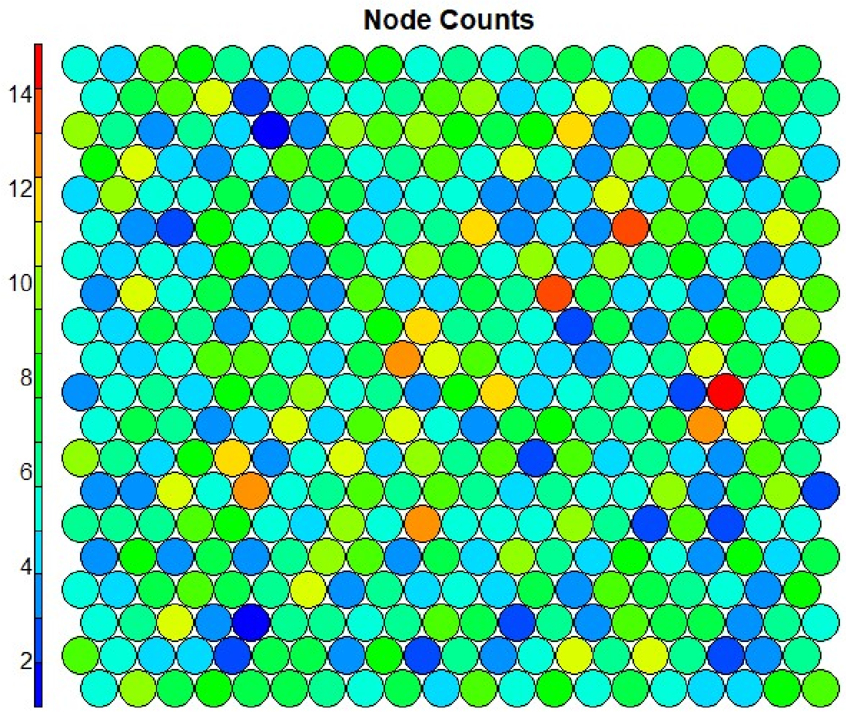
The Self-Organizing Map (SOM) is trained to determine the point at which further iterations will not result in significant improvement. The analysis of the plot reveals that the steepest decline occurred around 1400 iterations, and subsequent iterations show little to no improvement, as the curve flattens out.



Node counts

```
#counts within nodes
plot(som_model, type = "counts", main="Node Counts", palette.name=coolBlueHotRed)
```

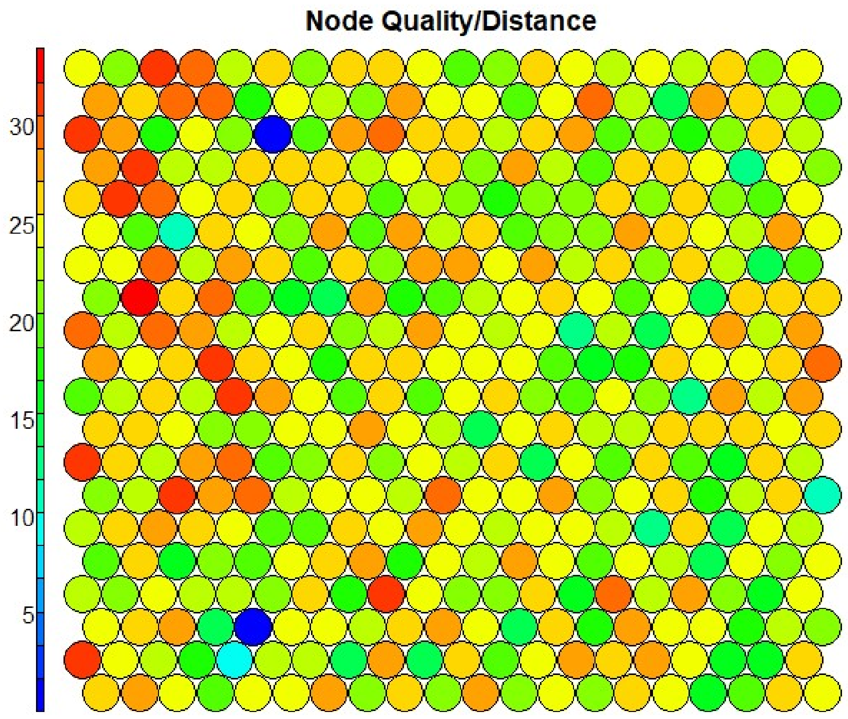
Visualise the count of how many samples are mapped to each node on the map. This metric can be used to measure the quality of a map. In our case, the samples distribution is quite uniform, a lot of 4-10 samples per node with low number of too small (1-2 count) and too large (12-14 count) samples.



Node quality

```
#map quality
plot(som_model, type = "quality", main="Node Quality/Distance", palette.name=coolBlueHotRed)
```

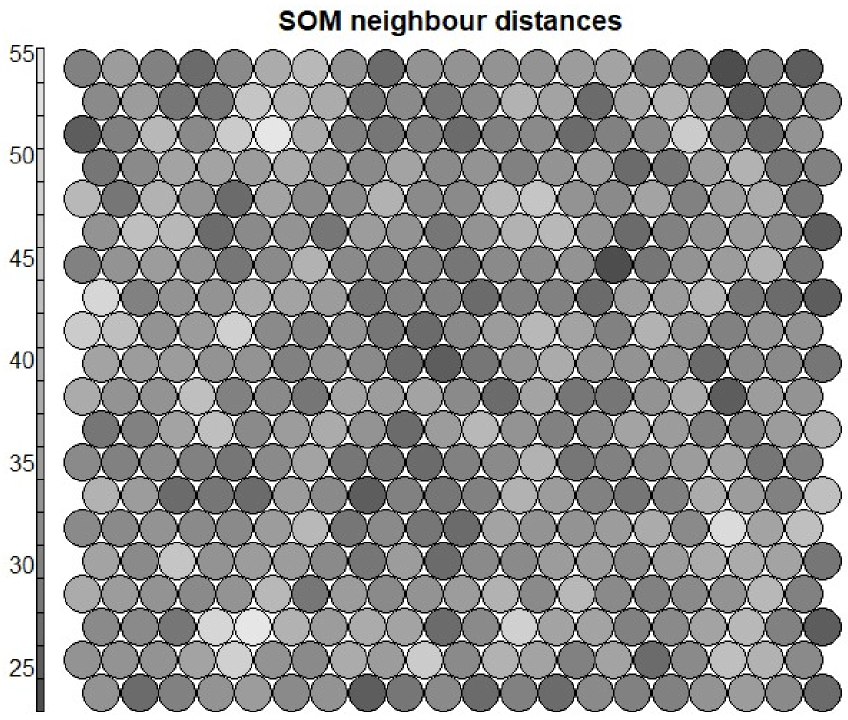
Visualize the quality or distance of each node in the SOM grid. The quality or distance of a node represents the similarity of the data points assigned to that node. In the plot, I can see that the nodes are leaning more on the high quality side (15-25)



Neighbour distances

```
#neighbor distances
plot(som_model, type="dist.neighbours", main = "SOM neighbour distances", palette.name=grey.colors)
```

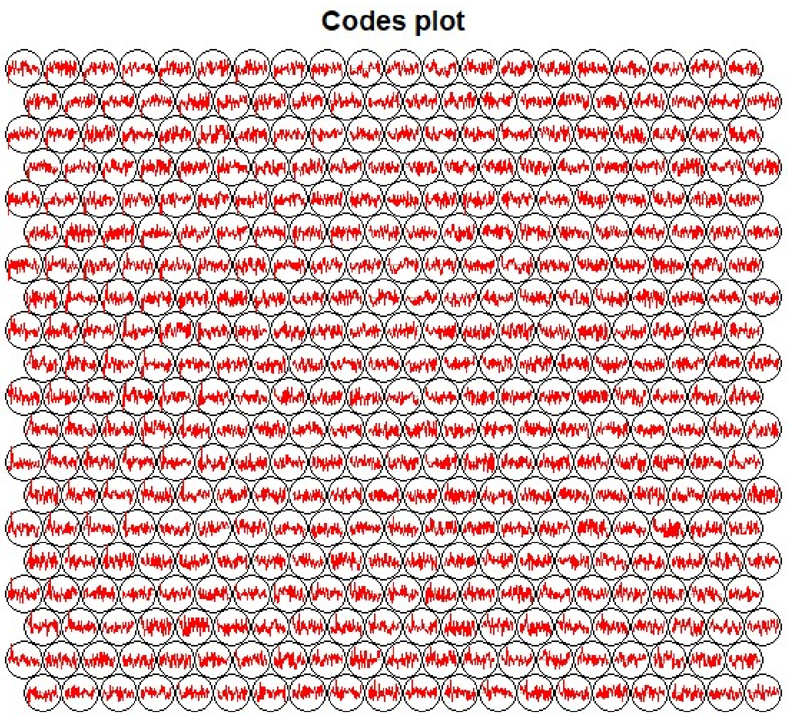
This visualisation is the distance between each node and its neighbours. Typically viewed with a grayscale palette, areas of low neighbour distance indicate groups of nodes that are similar. Areas with large distances indicate the nodes are much more different. In our case overall, it seems that the nodes are quite dark and not much high value nodes.



Code spread

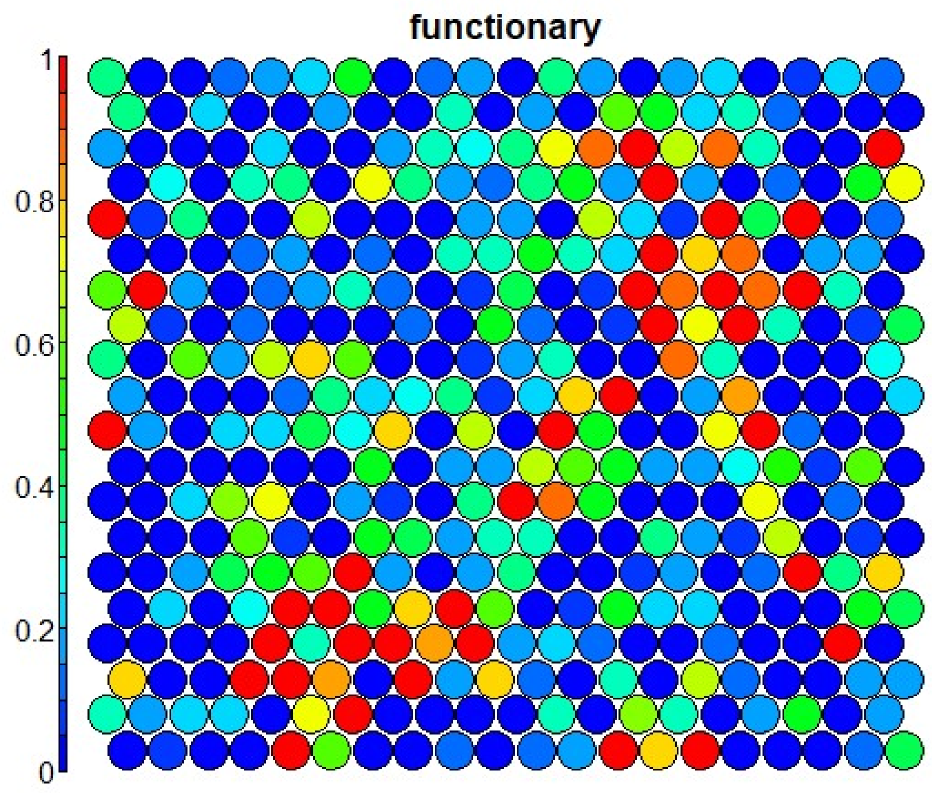
```
#code spread
plot(som_model, type = "codes")
```

Visualize the distribution of data vectors within the SOM grid. The red lines in the plot indicate the distribution of codebook. The length of the lines represents the density of data vectors assigned to each node. The code spread plot helps in understanding the distribution of data points in the SOM grid and can be useful in identifying clusters or patterns in the data.



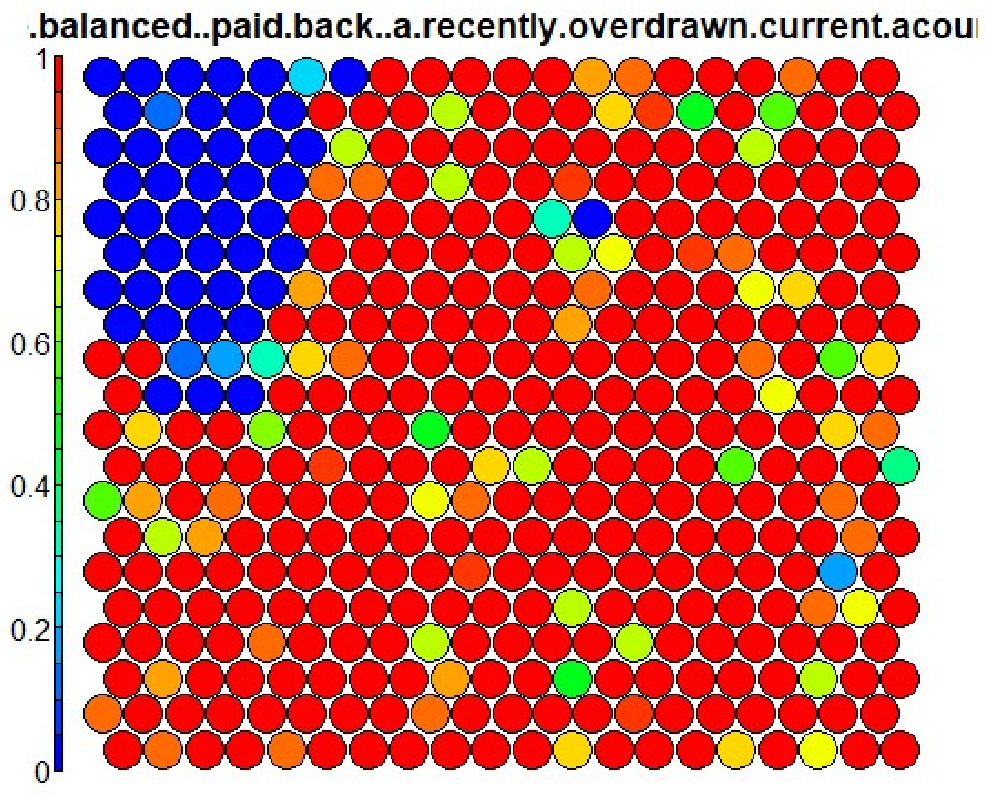
Functionary heat map

I observed that there are a lot more blues than reds and it seems that the colour distribution are quite scattered. There is no distinct presence of a cluster in the plot.



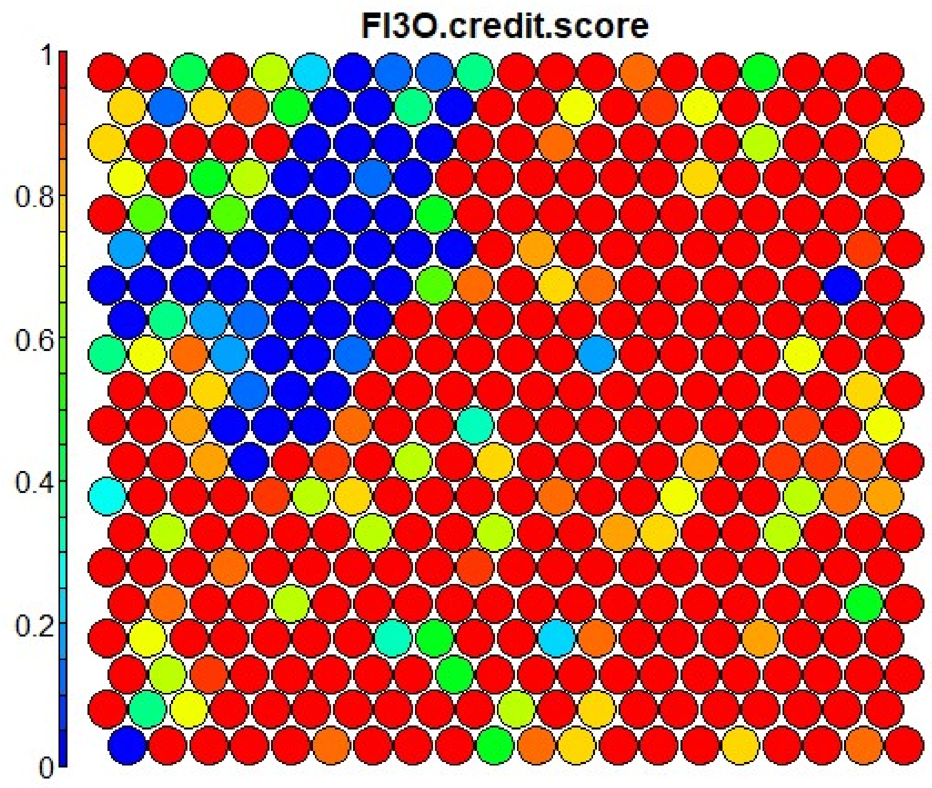
Re-balanced heat map

For re-balance, there’s an immediate observation that there are 2 distinct clusters of reds and blues separate one each other. Blue cluster is much smaller compared to the red cluster. A few outliers can be seen as well.



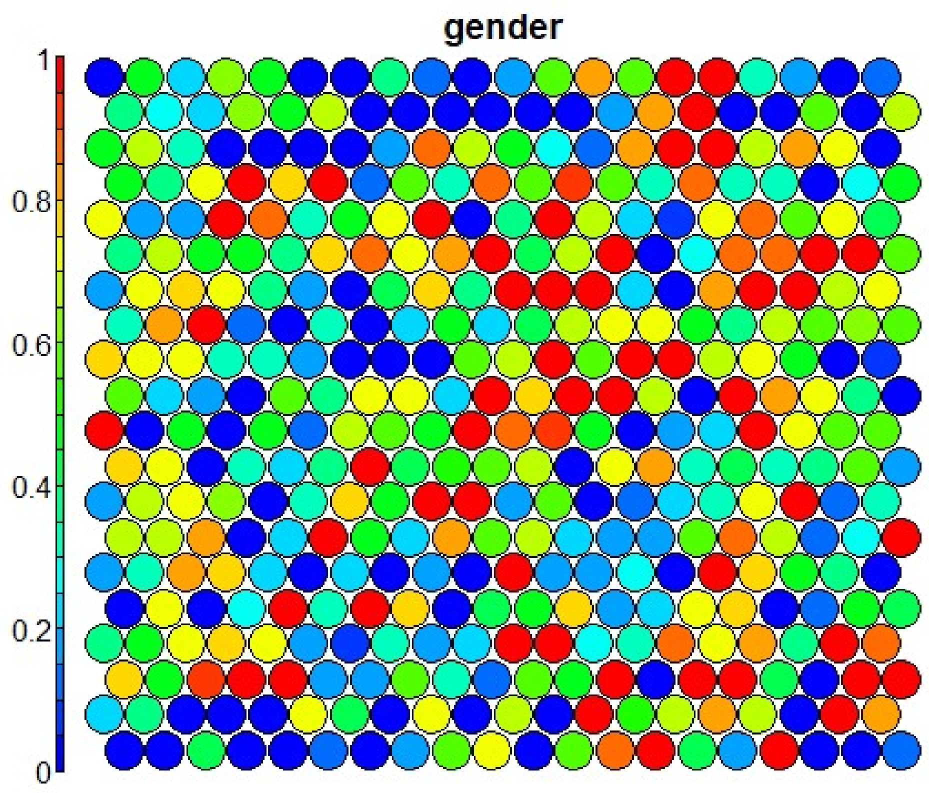
FICO credit score heat map

Fico as observed, is quite similar to re-balance plot but just a little bit less consistent. I can still see a small cluster of blues and large cluster of reds.



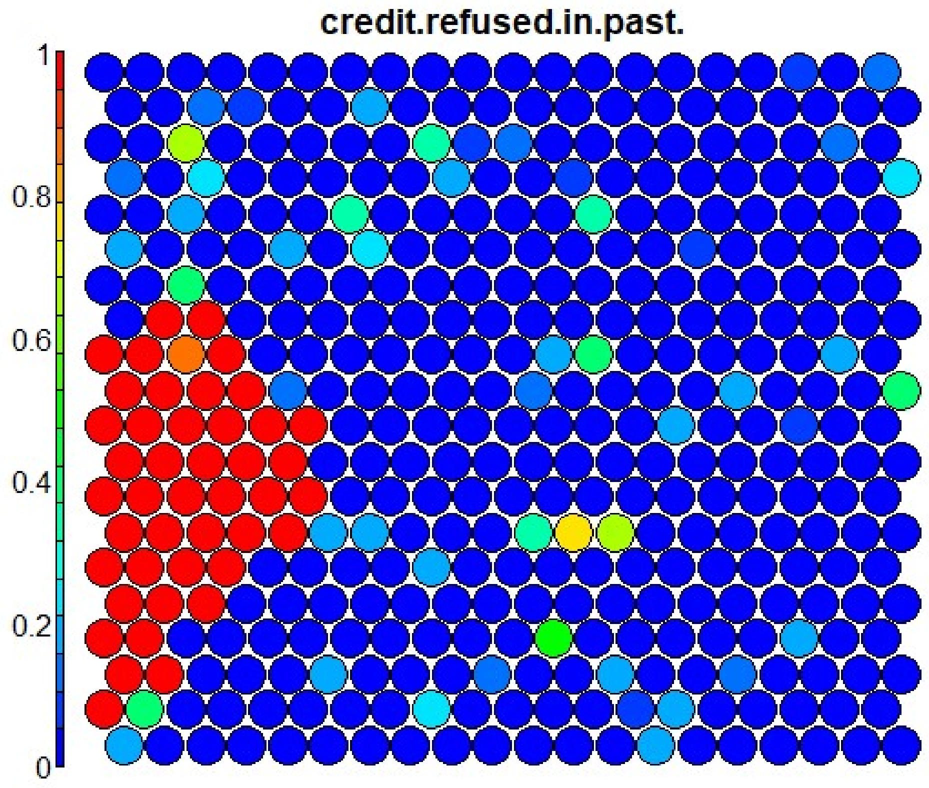
Gender heat map

Gender heat map main colour is on the average value but it is clear that the distribution is very scattered. There are no clusters seen and the data are all quite varied.



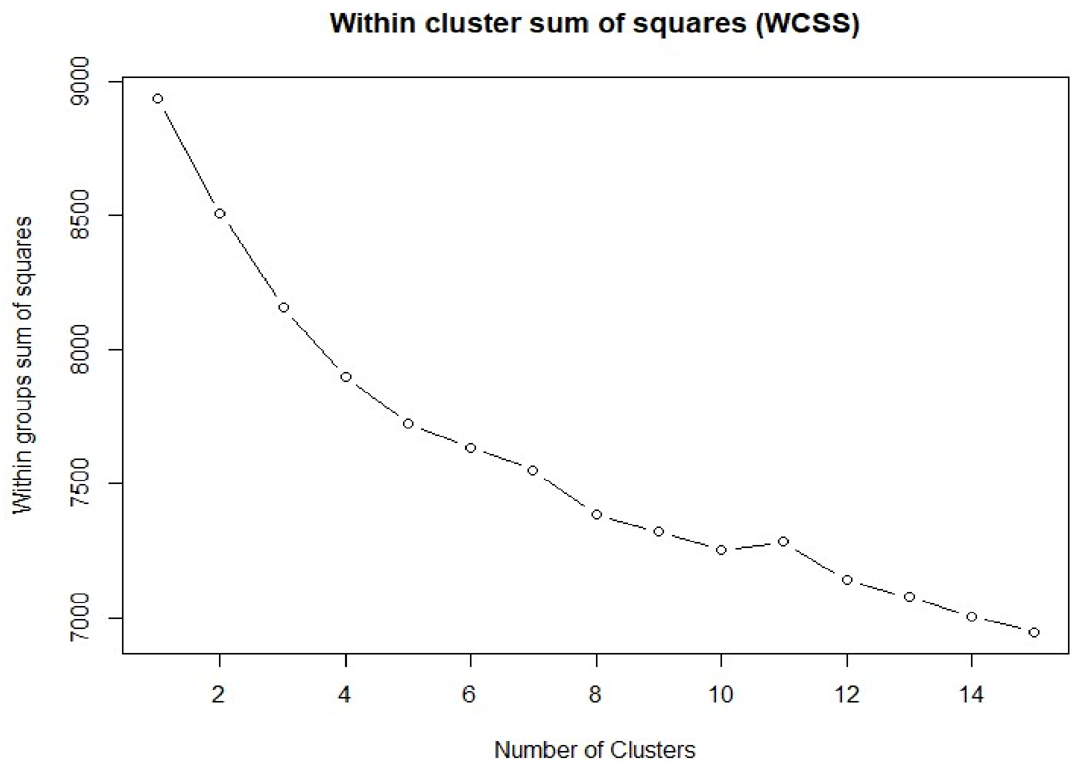
Credit refused in the past heat map

For credit refused, we have something opposite. There’s a much bigger blue cluster while the red cluster is quite small. A few outliers can be seen as well.

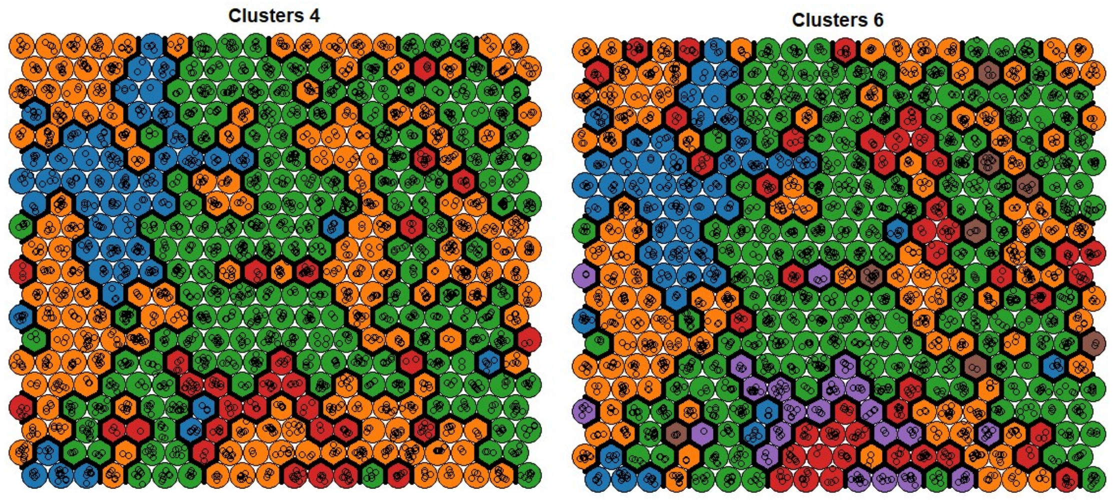


WCSS

WCSS is plotted to find the optimal number of clusters. Looking at the plot below, there are quite a few elbow points. The 1st one could be 4-6, 2nd one 7-9 or 3rd one at 11-13. I'll be using the 1st elbow to see the cluster result.



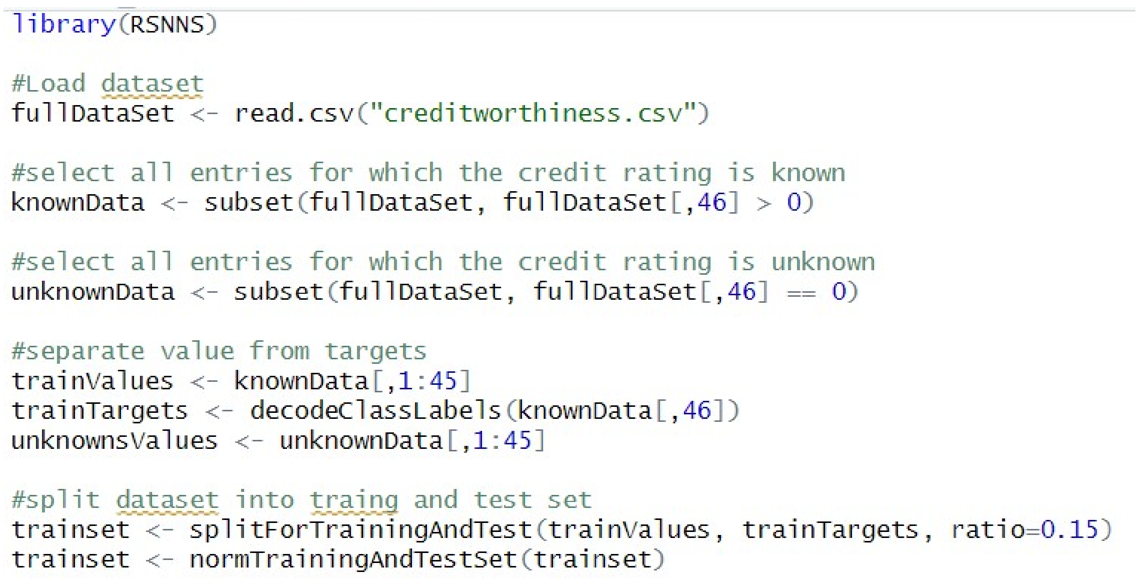
Cluster result



From the observation, Cluster 4 seems to separate the cluster better compared to Cluster 6.

Question 2

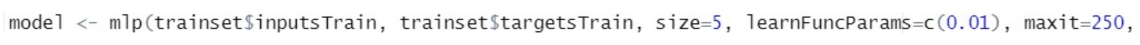
Prepare the data



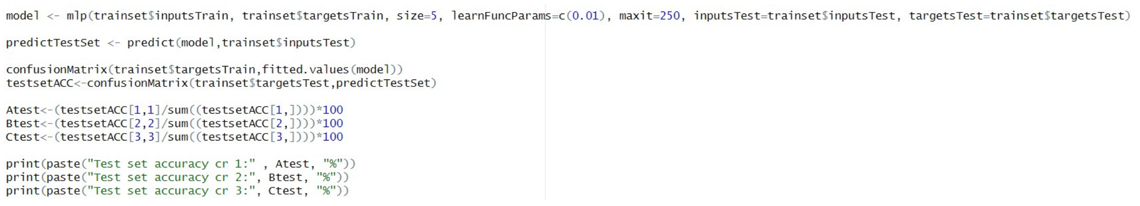
```
library(RSNNS)

#Load dataset
fullDataSet <- read.csv("creditworthiness.csv")

#select all entries for which the credit rating is known
knownData <- subset(fullDataSet, fullDataSet[,46] > 0)

#select all entries for which the credit rating is unknown
unknownData <- subset(fullDataSet, fullDataSet[,46] == 0)

#separate value from targets
trainValues <- knownData[,1:45]
trainTargets <- decodeClassLabels(knownData[,46])
unknownsValues <- unknownData[,1:45]

#split dataset into traing and test set
trainset <- splitForTrainingAndTest(trainValues, trainTargets, ratio=0.15)
trainset <- normTrainingAndTestSet(trainset)
```

First I divide the fullDataSet into two parts: "knownData" and "unknownData". The knownData contains the data points with credit rating, while the unknownData only contains data points without a credit rating. I then split the "trainValues" and "trainTargets" into training and testing sets using the with a ratio of 0.15.

Train the MLP model

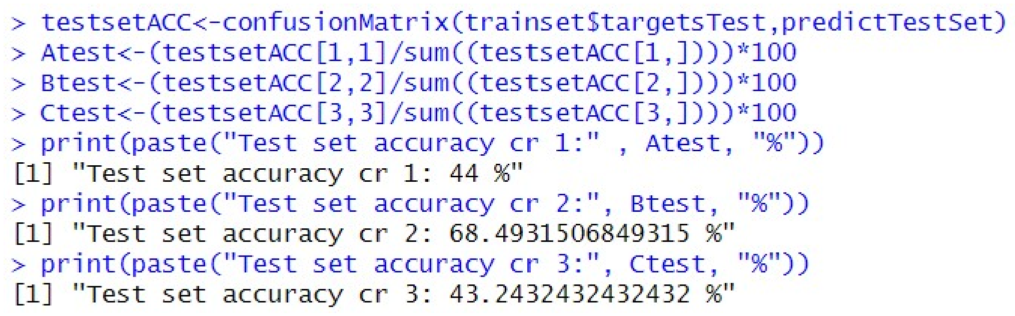


```
model <- mlp(trainset$inputsTrain, trainset$targetsTrain, size=5, learnFuncParams=c(0.01), maxit=250,
```



```
model <- mlp(trainset$inputsTrain, trainset$targetsTrain, size=5, learnFuncParams=c(0.01), maxit=250, inputsTest=trainset$inputsTest, targetsTest=trainset$targetsTest)

predictTestSet <- predict(model,trainset$inputsTest)

confusionMatrix(trainset$targetsTrain,fitted.values(model))
testsetACC<-confusionMatrix(trainset$targetsTest,predictTestSet)

Atest<-(testsetACC[1,1]/sum((testsetACC[1,])))*100
Btest<-(testsetACC[2,2]/sum((testsetACC[2,])))*100
Ctest<-(testsetACC[3,3]/sum((testsetACC[3,])))*100

print(paste("Test set accuracy cr 1:" , Atest, "%"))
print(paste("Test set accuracy cr 2:", Btest, "%"))
print(paste("Test set accuracy cr 3:", Ctest, "%"))
```

I first use a size 5, learn parameter 0.01 and maxit 250 to train the model. The result is as per below:

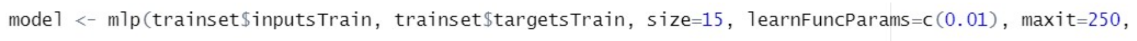


```
> testsetACC<-confusionMatrix(trainset$targetsTest,predictTestSet)
> Atest<-(testsetACC[1,1]/sum((testsetACC[1,])))*100
> Btest<-(testsetACC[2,2]/sum((testsetACC[2,])))*100
> Ctest<-(testsetACC[3,3]/sum((testsetACC[3,])))*100
> print(paste("Test set accuracy cr 1:" , Atest, "%"))
[1] "Test set accuracy cr 1: 44 %"
> print(paste("Test set accuracy cr 2:", Btest, "%"))
[1] "Test set accuracy cr 2: 68.4931506849315 %"
> print(paste("Test set accuracy cr 3:", Ctest, "%"))
[1] "Test set accuracy cr 3: 43.2432432432432 %"
```

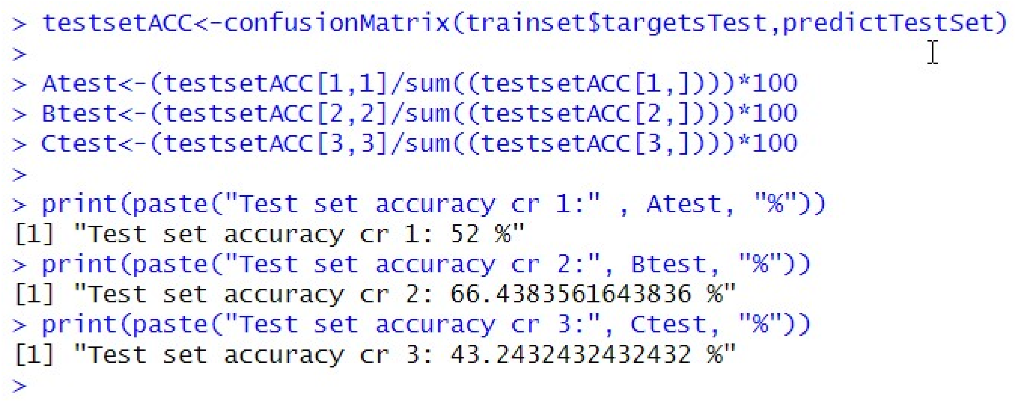
Strategies

There are a few strategies that can maximize the accuracy of the prediction

- Increasing the size of the MLP network can help to capture more complex relationships in the data which can lead to better accuracy. From the result, I can see that test accuracy A does improved quite a bit when increasing the size to 15

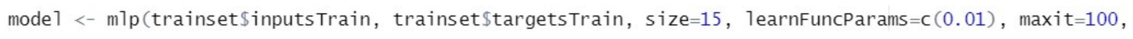


```
model <- mlp(trainset$inputsTrain, trainset$targetsTrain, size=15, learnFuncParams=c(0.01), maxit=250,
```

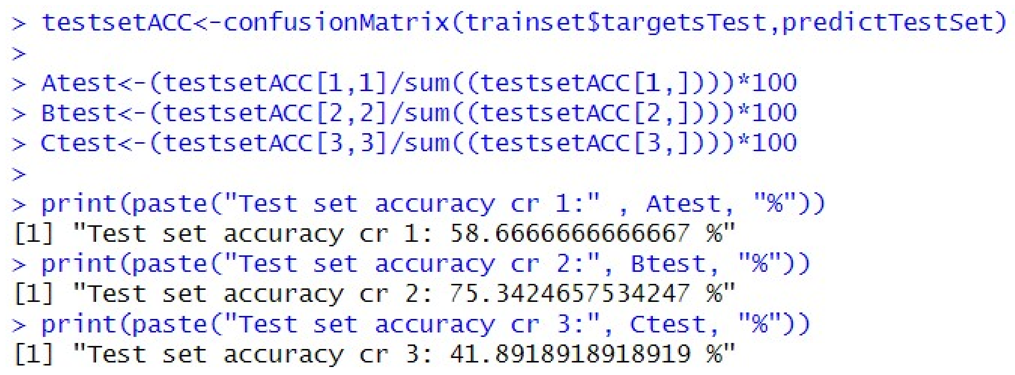


```
> testsetACC<-confusionMatrix(trainset$targetsTest,predictTestSet)
>
> Atest<-(testsetACC[1,1]/sum((testsetACC[1,])))*100
> Btest<-(testsetACC[2,2]/sum((testsetACC[2,])))*100
> Ctest<-(testsetACC[3,3]/sum((testsetACC[3,])))*100
>
> print(paste("Test set accuracy cr 1:" , Atest, "%"))
[1] "Test set accuracy cr 1: 52 %"
> print(paste("Test set accuracy cr 2:", Btest, "%"))
[1] "Test set accuracy cr 2: 66.4383561643836 %"
> print(paste("Test set accuracy cr 3:", Ctest, "%"))
[1] "Test set accuracy cr 3: 43.2432432432432 %"
>
```

- Fine-tune the learning parameters rate can also increase the accuracy, however, using 0.01 seems to yield better results compared to other numbers through what I tried so far.
- Number of iterations can help improve the accuracy as well. After a few trial and errors, I'll use number of iterations to be 100 to use the model. The results improve as below:

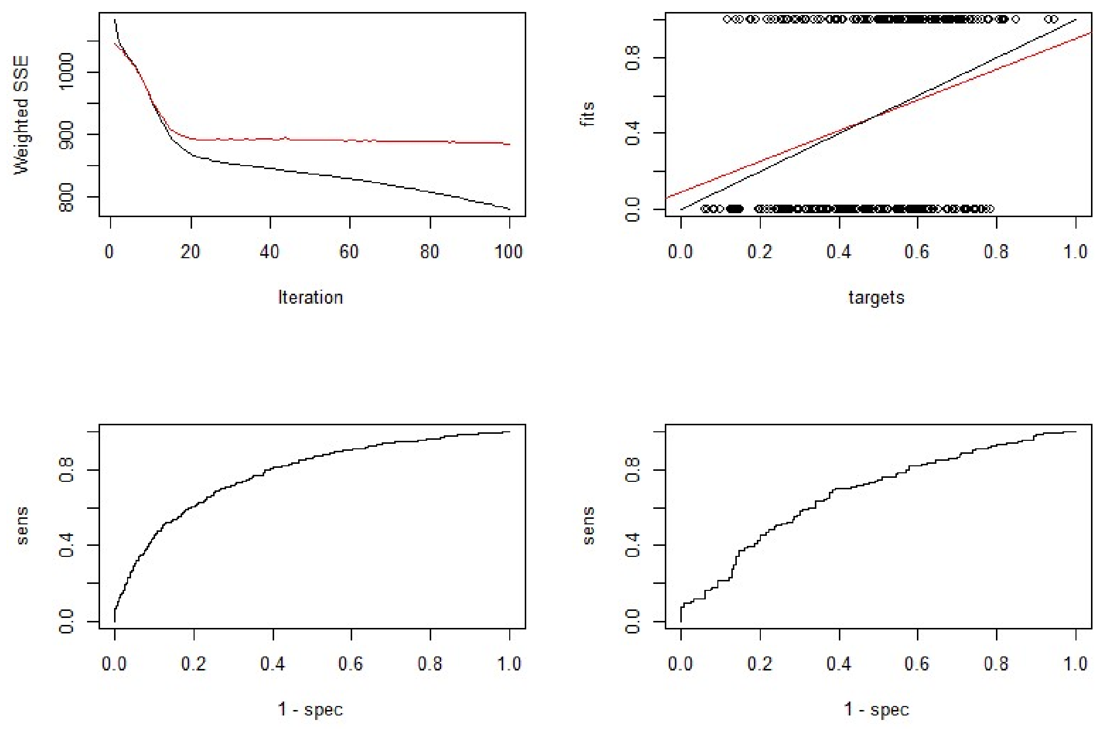


```
model <- mlp(trainset$inputsTrain, trainset$targetsTrain, size=15, learnFuncParams=c(0.01), maxit=100,
```



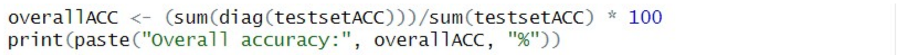
```
> testsetACC<-confusionMatrix(trainset$targetsTest,predictTestSet)
>
> Atest<-(testsetACC[1,1]/sum((testsetACC[1,])))*100
> Btest<-(testsetACC[2,2]/sum((testsetACC[2,])))*100
> Ctest<-(testsetACC[3,3]/sum((testsetACC[3,])))*100
>
> print(paste("Test set accuracy cr 1:" , Atest, "%"))
[1] "Test set accuracy cr 1: 58.6666666666667 %"
> print(paste("Test set accuracy cr 2:", Btest, "%"))
[1] "Test set accuracy cr 2: 75.3424657534247 %"
> print(paste("Test set accuracy cr 3:", Ctest, "%"))
[1] "Test set accuracy cr 3: 41.8918918918919 %"
```

Parameters plot

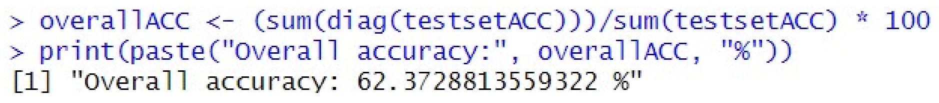


Observation

Calculate the overall accuracy of the model:



```
overallACC <- (sum(diag(testsetACC)))/sum(testsetACC) * 100
print(paste("Overall accuracy:", overallACC, "%"))
```



```
> overallACC <- (sum(diag(testsetACC)))/sum(testsetACC) * 100
> print(paste("Overall accuracy:", overallACC, "%"))
[1] "Overall accuracy: 62.3728813559322 %"
```

The overall accuracy of 62.4% means that, on average, 62.4% of the credit ratings in the test set were correctly predicted by the MLP model. This accuracy can be interpreted as a measure of how well the model generalizes to new, unseen data.

Appendix

```
# ------------------- PREPROCESSING ---------------------------

# LOAD LIBRARIES - install with:

# install.packages(c("kohonen", "dummies", "ggplot2", "maptools", "sp", "reshape2", "rgeos"))

library(kohonen)

#library(dummies)

library(ggplot2)

library(sp)

library(maptools)

library(reshape2)

library(rgeos)

# Color palette definition

pretty_palette <- c("#1f77b4", '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
'#e377c2')

# DATA PREPARATION

setwd("C:/Users/User/Desktop/SIM/INFO411/Assignment/a1/a1_support_files")

data <- read.csv("./creditworthiness.csv")

# Rows with 0 credit rating are rows that have not been assess, i will exclude those rows

# and find interesting data based on the subset of the data created (using subset() function)

classifiedData = subset(data, data[,46]>0)

# Use correlation method to find interesting attribute

corTable = abs(cor(classifiedData, y=classifiedData$credit.rating))

# To identify which features have a higher correlation value, i arrange the correlation table
in descending order

corTable = corTable[order(corTable, decreasing = TRUE),,drop = FALSE]

head(corTable,6)

data_plot <- data

data_plot$credit.rating <- factor(data$credit.rating, labels = c('NR', 'A', 'B', 'C'))

data_plot$functionary <- factor(data$functionary, labels = c('no', 'yes'))

data_plot$FI3O.credit.score <- factor(data$FI3O.credit.score, labels = c('not ok', 'ok'))

data_plot$re.balanced..paid.back..a.recently.overdrawn.current.acount <-
factor(data$re.balanced..paid.back..a.recently.overdrawn.current.acount, labels = c('no',
'yes'))

data_plot$credit.refused.in.past. <- factor(data$credit.refused.in.past., labels = c('no',
'yes'))

data_plot$gender <- factor(data$gender, labels = c('male', 'female'))

#Plot the data using bar graph based on the attributes decided above

##1

ggplot(data_plot, aes(x=functionary, fill=factor(credit.rating))) +

  geom_bar(position="dodge") +

  xlab("Functionary") +
  ylab("Count") +

  ggtitle("Credit Rating by Functionary")

no_count <- sum(data_plot$functionary == "no")

yes_count <- sum(data_plot$functionary == "yes")

print(no_count)

print(yes_count)

##2

ggplot(data_plot, aes(x=FI3O.credit.score, fill=factor(credit.rating))) +

  geom_bar(position="dodge") +

  xlab("FI3O.credit.score") +

  ylab("Count") +

  ggtitle("Credit Rating by FI3O.credit.score")

##3

ggplot(data_plot, aes(x=re.balanced..paid.back..a.recently.overdrawn.current.acount,
fill=factor(credit.rating))) +

  geom_bar(position="dodge") +

  xlab("re.balanced..paid.back..a.recently.overdrawn.current.acount") +

  ylab("Count") +

  ggtitle("Credit Rating by re.balanced..paid.back..a.recently.overdrawn.current.acount")

##4

ggplot(data_plot, aes(x=credit.refused.in.past., fill=factor(credit.rating))) +

  geom_bar(position="dodge") +

  xlab("credit.refused.in.past.") +

  ylab("Count") +

  ggtitle("Credit Rating by credit.refused.in.past.")

##5

ggplot(data_plot, aes(x=gender, fill=factor(credit.rating))) +

  geom_bar(position="dodge") +

  xlab("gender") +

  ylab("Count") +

  ggtitle("Credit Rating by gender")

#Another plot that i can use

#1

plot(data_plot$credit.rating ~ data_plot$functionary,

     xlab = 'functionary',ylab = 'credit rating',

     main = 'credit rating vs functionary')

#2

plot(data_plot$credit.rating ~ data_plot$FI3O.credit.score,

     xlab = 'FI3O.credit.score',ylab = 'credit rating',

     main = 'credit rating vs FI3O.credit.score')
```

```
#3

plot(data_plot$credit.rating ~
data_plot$re.balanced..paid.back..a.recently.overdrawn.current.acount,

     xlab = 're.balanced..paid.back..a.recently.overdrawn.current.acount',ylab = 'credit
rating',

     main = 'credit rating vs re.balanced..paid.back..a.recently.overdrawn.current.acount')

#4

plot(data_plot$credit.rating ~ data_plot$credit.refused.in.past.,

     xlab = 'credit.refused.in.past.',ylab = 'credit rating',

     main = 'credit rating vs credit.refused.in.past.')

#5

plot(data_plot$credit.rating ~ data_plot$gender,

     xlab = 'gender',ylab = 'credit rating',

     main = 'credit rating vs gender')


# ------------------- SOM TRAINING ---------------------------


#choose the variables with which to train the SOM

#the following selects column 1,2,3,4,6 according to the attributes selected

#data_train <- data[, c(1,2,3,4,6)]

# To train the SOM model, i chose to include all features for better

# insight to the relationship of the features exclude credit rating

data_train = data[, c(1:45)]

# now train the SOM using the Kohonen method

data_train_matrix <- as.matrix(scale(data_train))

names(data_train_matrix) <- names(data_train)

require(kohonen)

x_dim=20

y_dim=20

som_grid <- somgrid(xdim = x_dim, ydim=y_dim, topo="hexagonal")

if (packageVersion("kohonen") < 3){

  system.time(som_model <- som(data_train_matrix,

                             grid=som_grid,

                             rlen=1500,

                             alpha=c(0.9,0.01),

                             n.hood = "circular",

                             keep.data = TRUE ))

}else{

  system.time(som_model <- som(data_train_matrix,

                             grid=som_grid,
                             rlen=1500,

                             alpha=c(0.9,0.01),

                             mode="online",

                             normalizeDataLayers=false,

                             keep.data = TRUE ))

}

summary(som_model)

rm(som_grid, data_train_matrix)


# -------------------- SOM VISUALISATION -----------------


# Visualize the SOM model results

# Plot of the training progress - how the node distances have stabilized over time.

source('./coolBlueHotRed.R')

plot(som_model, type = "changes")

#counts within nodes

plot(som_model, type = "counts", main="Node Counts", palette.name=coolBlueHotRed)

#map quality

plot(som_model, type = "quality", main="Node Quality/Distance", palette.name=coolBlueHotRed)

#neighbor distances

plot(som_model, type="dist.neighbours", main = "SOM neighbour distances",
palette.name=grey.colors)

#code spread

plot(som_model, type = "codes")

# Plot the original scale heat map for a variable from the training set:

var <- 1 #define the variable to plot

var_unscaled <- aggregate(as.numeric(data_train[,var]), by=list(som_model$unit.classif),
FUN=mean, simplify=TRUE)[,2]

plot(som_model, type = "property", property=var_unscaled, main=names(data_train)[var],
palette.name=coolBlueHotRed)

rm(var_unscaled, var)

var <- 2 #define the variable to plot

var_unscaled <- aggregate(as.numeric(data_train[,var]), by=list(som_model$unit.classif),
FUN=mean, simplify=TRUE)[,2]

plot(som_model, type = "property", property=var_unscaled, main=names(data_train)[var],
palette.name=coolBlueHotRed)

rm(var_unscaled, var)

var <- 3 #define the variable to plot

var_unscaled <- aggregate(as.numeric(data_train[,var]), by=list(som_model$unit.classif),
FUN=mean, simplify=TRUE)[,2]

plot(som_model, type = "property", property=var_unscaled, main=names(data_train)[var],
palette.name=coolBlueHotRed)
```

```
rm(var_unscaled, var)

var <- 4 #define the variable to plot

var_unscaled <- aggregate(as.numeric(data_train[,var]), by=list(som_model$unit.classif),
FUN=mean, simplify=TRUE)[,2]

plot(som_model, type = "property", property=var_unscaled, main=names(data_train)[var],
palette.name=coolBlueHotRed)

rm(var_unscaled, var)

var <- 6 #define the variable to plot

var_unscaled <- aggregate(as.numeric(data_train[,var]), by=list(som_model$unit.classif),
FUN=mean, simplify=TRUE)[,2]

plot(som_model, type = "property", property=var_unscaled, main=names(data_train)[var],
palette.name=coolBlueHotRed)

rm(var_unscaled, var)

#plot a variable from the original data set (will be uncapped etc.)

# This function produces a menu for multiple heatmaps.

source('./plotHeatMap.R')

plotHeatMap(som_model, data, variable=0)


# ------------------ Clustering SOM results -------------------

# show the WCSS metric for kmeans for different clustering sizes.

# Can be used as a "rough" indicator of the ideal number of clusters

mydata <- matrix(unlist(som_model$codes), ncol = length(data_train), byrow = FALSE)

wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))

for (i in 2:15) wss[i] <- sum(kmeans(mydata,

                                     centers=i)$withinss)

par(mar=c(5.1,4.1,4.1,2.1))

plot(1:15, wss, type="b", xlab="Number of Clusters",

     ylab="Within groups sum of squares", main="Within cluster sum of squares (WCSS)")

# Form clusters on grid

## use hierarchical clustering to cluster the codebook vectors

var <- 4

title <- "Clusters 4"

som_cluster <- cutree(hclust(dist(mydata)), var)

# Show the map with different colours for every cluster

plot(som_model, type="mapping", bgcol = pretty_palette[som_cluster], main = title)

add.cluster.boundaries(som_model, som_cluster)

#show the same plot with the codes instead of just colours

plot(som_model, type="codes", bgcol = pretty_palette[som_cluster], main = "Clusters")

#add.cluster.boundaries(som_model, som_cluster)

var <- 6

title <- "Clusters 6"
```

```
som_cluster <- cutree(hclust(dist(mydata)), var)

# Show the map with different colours for every cluster

plot(som_model, type="mapping", bgcol = pretty_palette[som_cluster], main = title)

add.cluster.boundaries(som_model, som_cluster)

#show the same plot with the codes instead of just colours

plot(som_model, type="codes", bgcol = pretty_palette[som_cluster], main = "Clusters")

#add.cluster.boundaries(som_model, som_cluster)


# -------------------- Classification -----------------


library(RSNNS)

#Load dataset

fullDataSet <- read.csv("creditworthiness.csv")

#select all entries for which the credit rating is known

knownData <- subset(fullDataSet, fullDataSet[,46] > 0)

#select all entries for which the credit rating is unknown

unknownData <- subset(fullDataSet, fullDataSet[,46] == 0)

#separate value from targets

trainValues <- knownData[,1:45]

trainTargets <- decodeClassLabels(knownData[,46])

unknownsValues <- unknownData[,1:45]

#split dataset into traing and test set

trainset <- splitForTrainingAndTest(trainValues, trainTargets, ratio=0.15)

trainset <- normTrainingAndTestSet(trainset)

model <- mlp(trainset$inputsTrain, trainset$targetsTrain, size=15, learnFuncParams=c(0.01),
maxit=100, inputsTest=trainset$inputsTest, targetsTest=trainset$targetsTest)

predictTestSet <- predict(model,trainset$inputsTest)

confusionMatrix(trainset$targetsTrain,fitted.values(model))

testsetACC<-confusionMatrix(trainset$targetsTest,predictTestSet)

overallACC <- (sum(diag(testsetACC)))/sum(testsetACC) * 100

print(paste("Overall accuracy:", overallACC, "%"))

Atest<-(testsetACC[1,1]/sum((testsetACC[1,])))*100

Btest<-(testsetACC[2,2]/sum((testsetACC[2,])))*100

Ctest<-(testsetACC[3,3]/sum((testsetACC[3,])))*100

print(paste("Test set accuracy cr 1:" , Atest, "%"))

print(paste("Test set accuracy cr 2:", Btest, "%"))

print(paste("Test set accuracy cr 3:", Ctest, "%"))

par(mfrow=c(2,2))

plotIterativeError(model)

plotRegressionError(predictTestSet[,2], trainset$targetsTest[,2])
```

```
plotROC(fitted.values(model)[,2], trainset$targetsTrain[,2])

plotROC(predictTestSet[,2], trainset$targetsTest[,2])


#confusion matrix with 402040-method

confusionMatrix(trainset$targetsTrain, encodeClassLabels(fitted.values(model),method="402040",
l=0.4, h=0.6))

#show detailed information of the model

summary(model)

model

weightMatrix(model)

extractNetInfo(model)
```