Bienvenido Hiyas Jr.
13824819

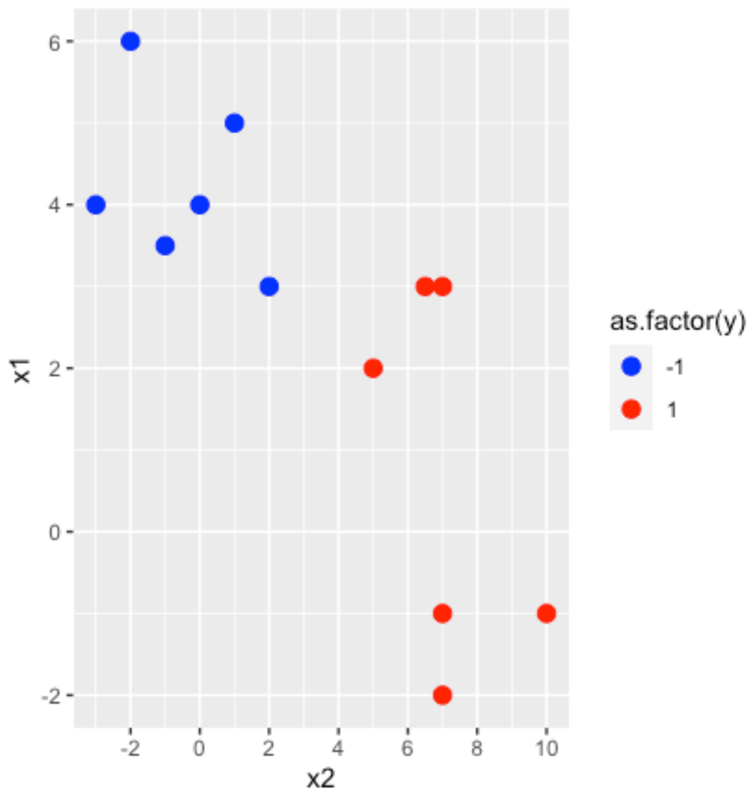
27/09/2021

Part I: An analytical problem

1. Draw a scatter plot to represent the points with Red colour for the class Y = 1 and Blue colour for Y = −1. X_1 is on the vertical axis while X_2 is on the horizontal axis.

   Ans: See R code below.

```
install.packages("ggplot2")
library(ggplot2)
install.packages("quadprog")
library(quadprog)

x1<-c(3, 4, 3.5, 5, 4, 6, 2, -1, 3, 3, -2, -1)
x2<-c(2, 0, -1, 1, -3, -2, 5, 7, 6.5, 7, 7, 10)
y<-c(-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1)

data<-data.frame(x1,x2,y)
colnames(data)<-c("x1","x2","y")
data
ggplot(data=data, aes(x=x2, y=x1, colour= as.factor(y))) +
  geom_point(size=3) +
  scale_x_continuous(breaks = seq(-4,10,2)) +
  scale_color_manual(values =c("blue", "red"))
```



2. > Find the optimal separating hyperplane of the classification problem using the function solve.QP() from quadprog in R. Show your work.

   Ans:

```
set.seed(123)
p<-ncol(data)
n<-nrow(data)
eps<- 1e-8

Dmat<- matrix(rep(0,p*p),nrow = p,ncol = p)
diag(Dmat)<-1
Dmat[p,p]<- eps

Amat<- cbind(as.matrix(data[,c(2,1)]), y = rep(1,n))
Amat<- Amat* data$y

dVec<- rep(0,p)
bVec<- rep(1,n)

Opt_hyperplane<- solve.QP(Dmat, dVec, t(Amat), bVec)
Opt_hyperplane$solution

> Opt_hyperplane$solution
[1]  0.6000000 -0.2000001 -1.5999998
```

   As shown in R result for Opt_hyperplane$solution, the values for betas are β_0= -1.60, β_1= 0.60, β_2= -0.20

   > Sketch the optimal separating hyperplane in the scatter plot obtained in Question 1.

   Ans:

   The dimensions for the hyperplane can be defined by the equation (4.1) below from Topic 2: Linear Support Vector.

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p = 0 \qquad (4.1)$$

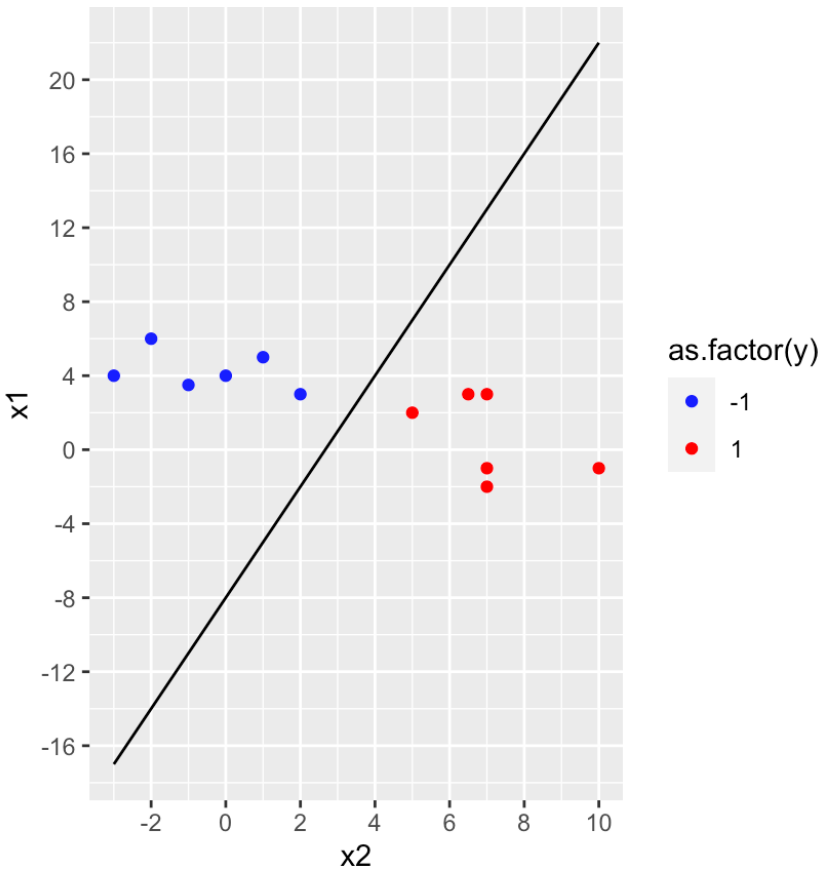
   where $\beta_0 + \beta_1 \ldots \beta_p$ are the parameters.

   Substituting the beta coefficients of Opt_hyperplane$solution to the equation, we will have:

$$-1.6 + 0.6X_2 - 0.2X_1 = 0$$

   Plotting the optimal separating hyperplane with the data set from Q1 in scatterplot we will have:

```
betas <- Opt_hyperplane$solution
x2_Opt_hyperplane <- data$x2
x1_Opt_hyperplane <- (-betas[3]-betas[1]*x2_Opt_hyperplane)/betas[2]
Opt_sep_hyperplane <- data.frame(x2_Opt_hyperplane, x1_Opt_hyperplane)

ggplot() +
  geom_line(aes(x2_Opt_hyperplane, x1_Opt_hyperplane)) +
  geom_point(data=data, aes(x2, x1, colour = as.factor(y))) +
  labs(x = "x2", y = "x1") +
  scale_colour_manual(values = c("blue", "red")) +
  scale_x_continuous(breaks = seq(-4,20,2)) +
  scale_y_continuous(breaks = seq(-20,20,4))
```



The plot above shows the optimal Hyperplane (black line) separating factors -1 (blue) and 1 (red) successfully. It lies at a ‘maximum distance’ from both classes and gives the maximum separation.

3. Describe the classification rule for the maximal margin classifier.

   Ans:

   The classification rule for the maximal margin classifier is described in the equation (4.4) below from Topic 2: Linear Support Vector.

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0 \qquad (4.4)$$

Hence, if the equation for the Optimal Hyperplane is $-1.6 + 0.6X_2 - 0.2X_1 = 0$, then the margin classifiers are determined if they are above or below the Optimal Hyperplane (black line), hence:

for y = -1 (blue),

$$-1.6 + 0.6X_2 - 0.2X_1 > 0$$

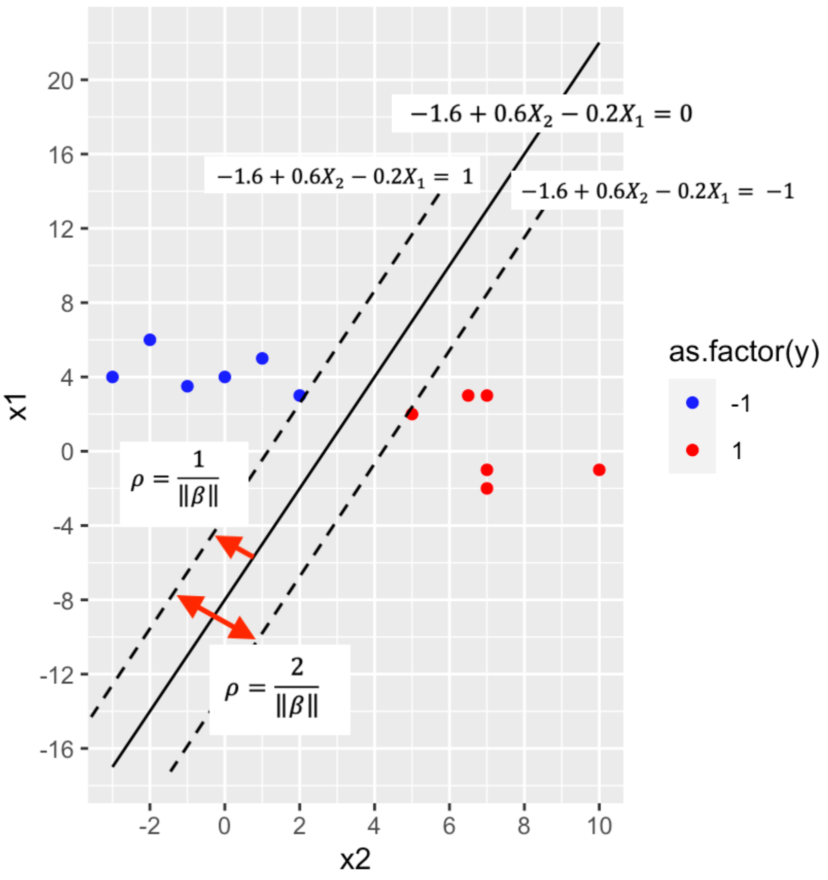
For y =1 (red),

$$-1.6 + 0.6X_2 - 0.2 < 0$$

4. Compute the margin of the classifier.

Ans:

From the figure below.



The margin M has a total width of $\frac{2}{\|\beta\|}$. From Opt_hyperplane$solution, the beta coefficients are (0.6, -0.2). Solving for the margin, we have.

$$\frac{2}{\sqrt{0.6^2+(-0.2)^2}} = 3.162$$

Part II: An application

2.1 Background on Credit Card Dataset

The data, “CreditCard Data.xls”, is based on Yeh and hui Lien (2009). The data contains 30,000 observations and 23 explanatory variables. The response variable, Y, is a binary variable where “1” refers to default payment and “0” implies non-default payment.

First, the file “CreditCard Data.xls” is imported in R using the read_excel function and store in “data2”. Then, str(data2) is used to check the data types of the variables. The variables Sex, Education, Marriage, Pay_0-Pay6 and ‘default payment next month’ are converted from numerical into character or text data types. Finally, sum(is.na(data2)) is used to check if there are missing values in data2 which shows 0 NA’s.

```
data2<- read_excel("CreditCard_Data.xls", skip = 1)
str(data2) #checking data types

#convert the parameters below into characters.
data2$SEX<-as.character(data2$SEX)
data2$EDUCATION<-as.character(data2$EDUCATION)
data2$MARRIAGE<-as.character(data2$MARRIAGE)
data2$PAY_0<-as.character(data2$PAY_0)
data2$PAY_2<-as.character(data2$PAY_2)
data2$PAY_3<-as.character(data2$PAY_3)
data2$PAY_4<-as.character(data2$PAY_4)
data2$PAY_5<-as.character(data2$PAY_5)
data2$PAY_6<-as.character(data2$PAY_6)
data2$`default payment next month`<-as.character(data2$`default payment next month`)
str(data2) #checking data types again
sum(is.na(data2))
> sum(is.na(data2))
[1] 0
```

However, upon checking on the values in the parameters, it is found that the value “5” and “6” in the EDUCATION feature means unknown. Moreover, the value “0” in both EDUCATION and MARRIAGE also means unknown. Hence, we need to assume that these are missing values and convert them into “NA”. We can omit them since they are only small against the whole data set. However, we can also use imputation since most tree-based classifier algorithms are able to handle missing values during the construction of the model. Finally, the response variable was change from ‘default payment next month’ to DEFAULT.

```
data2$EDUCATION[data2$EDUCATION == "5"] <- NA
data2$EDUCATION[data2$EDUCATION == "6"] <- NA
data2$EDUCATION[data2$EDUCATION == "0"] <- NA
data2$MARRIAGE[data2$MARRIAGE == "0"] <- NA
na_educ<-sum(is.na(data2$EDUCATION))
na_marri<-sum(is.na(data2$MARRIAGE))
cbind(na_educ,na_marri)

> cbind(na_educ,na_marri)
    na_educ na_marri
[1,]     345       54  #missing values

data2<- mutate_if(data2, is.character, as.factor)
data2<- rename(data2, DEFAULT = 'default payment next month') #
```

The first variable in the data set "ID" can be deleted as this is not useful for the algorithm. Therefore, the dimensions for the training data are accurately 21000 X 24. It can also be noticed that the dataset was unbalanced. While this is common for datasets, this can greatly affect the performance of the model. To solve this issue, we can adjust probabilities and different sampling methods as well as choose a cut off points for the predicted probabilities.

NOTE: Running the function for tuning RF and SVM took a lot of time thinking my laptop might have stopped or crashed. Tried to cut-off the data half from 30000 to 15000 with observations proportional DEFAULT (0 and 1). I got the results, but I tried to do the full dataset again but reduced the parameters for tuning and got a good result. I observed that the performance when I tried half and full were almost the same. But then I continue to use the full dataset for further tasks.

2.2 Assessment Tasks

2.2.1 Data

(a) Select a random sample of 70% of the full dataset as the training data, retain the rest as test data. Provide the code and print out the dimensions of the training data.

Ans:

```
#Splitting the data2 into training (70%) and test sets
set.seed(123)
ind<- sample(nrow(data2), nrow(data2)*0.7)
train<- data2[ind,]
test<- data2[-ind,]
# print number of observations in test vs. train c(nrow(train), nrow(test))
c(nrow(train), ncol(train))

> c(nrow(train), ncol(train))
[1] 21000    25
```

R results shows the training data has a row of 21000 and 25 columns.

2.2.2 Tree Based Algorithms

(a). Use an appropriate tree-based algorithm to classify credible and non-credible clients. Specify any underlying assumptions. Justify your model choice as well as hyper-parameters which are required to be specified in R.

Ans:

There are different tree-based algorithms to classify credible and non-credible clients. These are the bagging, boosting and random forest techniques. These algorithms are usually combined with individual decision trees to improve predictive performance or high sensitivity to the changes in the dataset.

The tree-based algorithm that is selected for this certain dataset to classify credible and non-credible clients is the Random Forest. It introduces a random element into the tree construction process and decorrelates the trees by allowing a sub-sample of the features to be considered at any split. Compared to boosting, it is less computationally intensive and time consuming because of the sequential tree construction process as well as the calibration of key model hyperparameters.
One major pitfall of bagging is that trees are correlated. When the trees are correlated, averaging many trees only reduces part of the variance. The other part of the variance is controlled by correlation between trees. Therefore, the goal of Random Forest is to produce de-correlated trees and average the results of these trees.

Moreover, Random Forest have fewer key hyperparameters than the boosting. This could result to less computation power and time if the training data is large. Finally, Random Forest is less susceptible to over-fitting if a sufficiently large number of trees are used in the model compared boosting. Over-fitting is caused by noise in the data.

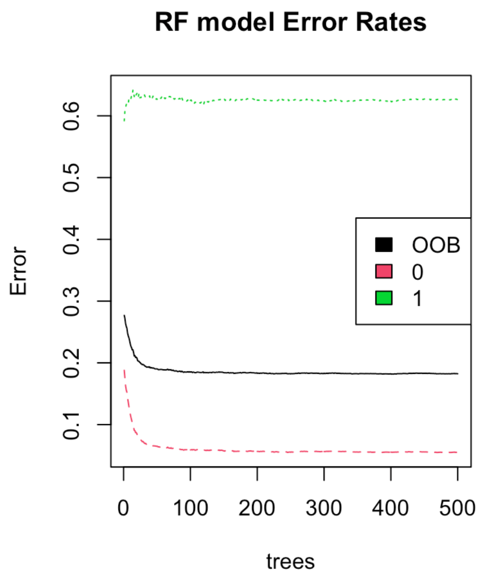
Now, we will check how tuning hyperparameters affect the result against the default values of Random Forest. First, we need to see the result of the default Random Forest. The initial model was trained with 500 trees and the out-of-bag (OOB) error rates used to determine if training over a larger number of trees was required before proceeding with hyperparameter tuning.

```
set.seed(123)
train_RF <- na.roughfix(train)
model_RF<- randomForest(DEFAULT ~.-ID, data = train_RF) #time around 40sec
model_RF
> model_RF

Call:
 randomForest(formula = DEFAULT ~ . - ID, data = train_RF)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 18.23%
Confusion matrix:
      0    1 class.error
0 15422  899  0.05508241
1  2930 1749  0.62620218
```

From the R results above, the default number of trees is 500, the OOB error rate is 18.23% with the confusion matrix indicated.

Plotting the result, we have:

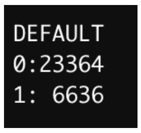
```
model_oob_err <- model_RF$err.rate
plot(model_RF, main = "RF model Error Rates") +
legend(x = "right", legend = colnames(model_oob_err),
       fill = 1:ncol(model_oob_err))
```



The Optimal Model is defined as the one that minimised OOB error. randomForest only have the mtry hyperparameter for tuning therefore, we will use the for-loop and expand.grid function to optimise the selected set of hyperparameters in the grid.

```
hyper_grid <- expand.grid(
  mtry = seq(3, 6, by = 1),
  sampsize = nrow(train_RF)*c(0.632, 0.8, 1.0),
  nodesize = seq(1, 5, by = 1),
  oob_err = 0
)
for(i in 1:nrow(hyper_grid)) {
  tune_rf <- randomForest(DEFAULT ~.-ID, train_RF,
                        mtry = hyper_grid$mtry[i],
                        sampsize = hyper_grid$sampsize[i],
                        nodesize = hyper_grid$nodesize[i],
                        seed = 2020)
  hyper_grid$oob_err[i] <- tune_rf$err.rate[nrow(tune_rf$err.rate), "OOB"]
}
> tune_rf

Call:
 randomForest(formula = DEFAULT ~ . - ID, data = train_RF, mtry = hyper_grid$mtry[i],
sampsize = hyper_grid$sampsize[i], nodesize = hyper_grid$nodesize[i],      seed = 2020)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 18.2%
Confusion matrix:
      0    1 class.error
0 15438  883  0.05410208
1  2938 1741  0.62791195

hyper_table <- hyper_grid %>%
  arrange(oob_err) %>%
  head(10)
knitr::kable(hyper_table)
+   arrange(oob_err) %>%
+   head(10)
> knitr::kable(hyper_table)


| mtry| sampsize| nodesize|   oob_err|
|----:|--------:|--------:|---------:|
|    4|    13272|        1| 0.1795238|
|    4|    16800|        5| 0.1804286|
|    5|    13272|        5| 0.1807619|
|    6|    13272|        5| 0.1809524|
|    6|    13272|        1| 0.1810476|
|    6|    13272|        4| 0.1810476|
|    5|    13272|        4| 0.1811429|
|    4|    13272|        4| 0.1811905|
|    4|    21000|        2| 0.1812381|
|    4|    13272|        5| 0.1812857|
```

The R result above shows the optimal randomForest model had the hyperparameter values; mtry = 4, sample size=13272 and node size 1. One thing to notice is that 13272 is 63.2 percent of the training set. It also is the number of observations when we need to have a balance data since the number of DEFAULT 1 is 6636, then we need a DEFAULT 0 to have the same 6636 observation to have a balance data of 13272.



```
DEFAULT
0:23364
1: 6636
```

It was also observe that even though the parameters for tuning like: mtry, sampsize and nodesize has been reduced, it would still take around 20-30 mins to finish the function.

(b). Display model summary and discuss the relationship between the response variable versus selected features.

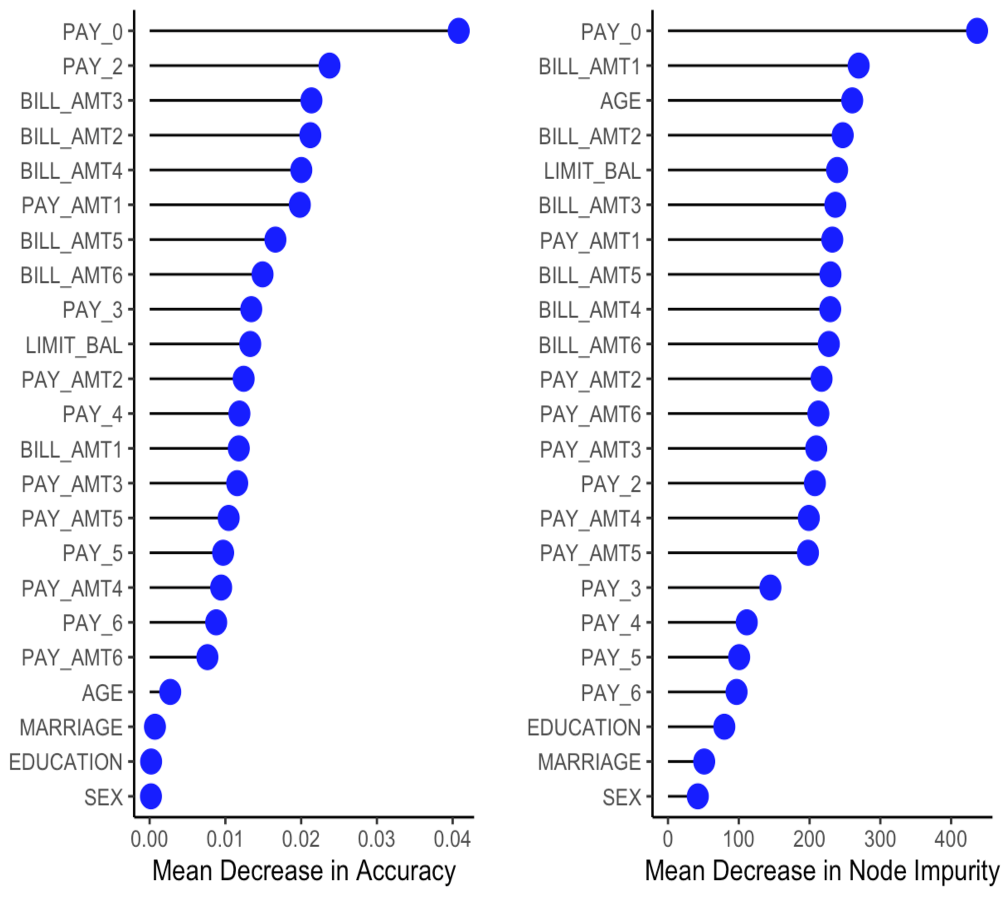
Ans:

Applying the tuned hyperparameters tto he random forest function.

```
model_RF2<- randomForest(DEFAULT ~.-ID, data = train_RF, importance=TRUE,
                 mtry = 4,
                 sampsize = 13272,
                 nodesize = 1)
model_RF2
> model_RF2

Call:
 randomForest(formula = DEFAULT ~ . - ID, data = train_RF, importance = TRUE,      mtry = 4,
sampsize = 13272, nodesize = 1)
           Type of random forest: classification
                 Number of trees: 500
No. of variables tried at each split: 4

      OOB estimate of  error rate: 18.13%
Confusion matrix:
     0    1 class.error
0 15436  885  0.05422462
1  2923 1756  0.62470613
```

Now we are going to plot the contributions of the feature (predictor) variables. The Mean Decrease in Accuracy (MDA) and the Mean Decrease in Node Impurity (MDI) can be calculated by the randomForest package. The MDA can be calculated by permuting OOB data, and the MDI is the age total decrease in node impurity from splitting on the feature.

```
#Variable Importance
VarImp_RF<- importance(model_RF2, scale = FALSE)
feature_Var<-as.vector(names(train_RF)[2:24])
VarImp_RF_table<- data_frame(feature_Var, MDA = as.numeric(VarImp_RF[,3]),
                             MDG = as.numeric(VarImp_RF[,4]))
RF_MDA_plot <- VarImp_RF_table %>%
 arrange(MDA) %>%
 mutate(feature_Var = factor(feature_Var, levels = feature_Var)) %>%
 ggplot(aes(x = feature_Var, y = MDA)) +
 geom_segment(aes(xend = feature_Var, yend = 0)) +
 geom_point(size = 4, colour = "blue") +
 coord_flip() +
 theme_classic() +
 xlab("") +
 ylab("Mean Decrease in Accuracy")
RF_MDG_plot <- VarImp_RF_table %>%
 arrange(MDG) %>%
 mutate(feature_Var = factor(feature_Var, levels = feature_Var)) %>%
 ggplot(aes(x = feature_Var, y = MDG)) +
 geom_segment(aes(xend = feature_Var, yend = 0)) +
 geom_point(size = 4, colour = "blue") +
 coord_flip() +
 theme_classic() +
 xlab("") +
 ylab("Mean Decrease in Node Impurity")
gridExtra::grid.arrange(RF_MDA_plot, RF_MDG_plot, ncol = 2)
```



The plot above shows that the PAY_0 is the highest predictor for credit card DEFAULT while on the other hand, SEX is the lowest predictor along with MARRIAGE. It can also be noticed that AGE has lower importance in MDA while has higher in MDI making into the top 3. Furthermore, BILL** features showed to have higher importance than PAY** features in both MDA and MDI. Lastly, the

(c). Evaluate the performance of the algorithm on the training data and comment on the results.

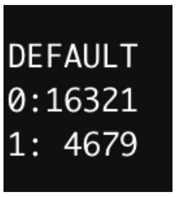
Ans:

Evaluation the performance of the algorithm can be done by interpreting the confusion matrix.

```
RF_oob_err <- model_RF2$err.rate
RF_oob_err_final <- RF_oob_err[nrow(RF_oob_err ), "OOB"]
RF_oob_err_model <- model_oob_err [nrow(model_oob_err ), "OOB"]
RF_CM <- caret::confusionMatrix(data = model_RF2$predicted,
                        reference = train_RF$DEFAULT)
RF_overall_metrics <- data.frame(OOB_error = RF_oob_err_final[[1]],
                         Accuracy =RF_CM$overall[[1]],
                         Kappa = RF_CM$overall[[2]])
knitr::kable(RF_overall_metrics)

> knitr::kable(RF_overall_metrics)


| OOB_error|  Accuracy|     Kappa|
|---------:|---------:|---------:|
| 0.1813333| 0.8186667| 0.3801196|
```

The result above shows that the overall accuracy of the model is good at 81.87%. The tuned random forest shows a reduction of OOB error rate from 18.23% of the base (default) model (model_RF) to 18.13% of the tuned rf model (model_RF2). This shows that the tuned random Forest performs better than base random forest. However, the difference is almost negligible, yet we can say that the base model performs well enough already in this dataset.

As stated before, the dataset is very much unbalance and a large dataset with 30000 variables. The defaulted payment is only 22% of the total training set. This could have greatly affected the time consumed in the calculation and performance of randomForest.



```
DEFAULT
0:16321
1: 4679
```

2.2.3 Support vector classifier

(a) Use an appropriate support vector classifier to classify the credible and non-credible clients. Justify your model choice as well as hyper-parameters which are required to be specified in R. (10 marks)

Ans:

Support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis but generally for classifying data. SVM can be divided int o2 categories. First is the Linear SVM :- *In linear SVM, we can use a single straight line to classify the dataset and such type of dataset is known as Linearly Separable data. Second is the* Non-Linear SVM *:- In non-linear SVM, we cannot use a single straight line to classify the dataset because data is not linearly separable*

Support vector classifiers attempts to partition a feature space into groups by finding the optimal means based on their known class labels. They are also called as maximal maginal classifer that separate data into two classes using a linear hyperplane located furthes from the data.

Suport vector machines on the other hand is an extension of a support vector classifier with non-linear boundaries using a concept known as a kernel. *Kernel Trick* helps to convert the *low dimensional* input space into *higher dimensional* space. It uses some *mapping function* to do so. SVM uses different types of kernels like *Linear*, *Polynomial*, *Radial Basis Function Kernel* etc.

The advantages of SVM includes:

1. *It works very well on a range of datasets.*
2. *SVM works well on low as well as on high dimensional data.*
3. *As compared to other models, overfitting is less observed here.*
4. *They are flexible with structured, unstructured and semi structured data.*
5. *It uses only a subset of training data (i.e. support vector), that's why it is memory efficient also.*

The disadvantages include:

1. *It requires more training time.*
2. *Choosing the appropriate kernel is not easy.*
3. *It does not perform well on large datasets.*
4. *SVM is difficult to understand and interpret.*
5. *In case of noisy dataset it does not perform very well.*

The SVM that is chosen for the CreditCard dataset is the SVM with linear kernel function. The main reason is that the data is not linearly separable. Moreover, it can accommodate a wider variety of decision boundary types. In addition, Linear Kernel allows us to determine the importance of each feature variables which is required on the task because linear kernel does not transform the data into another feature space.

Finally, linear kernel has only one parameter to tune which is the cost parameter (C) which could be helpful in tuning the CreditCard dataset.
The lower the value of C makes a larger margin and results to a simpler decision function, however with the cost of training accuracy. In contrast, the higher value of C makes a smaller margin but with the risk of incorrect classification and observation. The default value of C in the package e1071 is 1.

The R code below is how the training data is run with default SVM with linear Kernel. Then we get the confusion matrix to interpret the result and finally the SVM is tuned to compare with the default SVM. The results of these functions are discussed in the next task.

```
library(e1071)
set.seed(123)
# data pre-processing
train_svm <- mutate_if(train[1:24], is.factor, as.numeric)
train_svm <- as.data.frame(impute(train_svm[1:24], what = "median"))
train_svm$DEFAULT <- train$DEFAULT
model_svm <- svm(DEFAULT ~.-ID, data = train_svm, kernel = "linear",
           scale = TRUE, probability = TRUE)
summary(model_svm)

#Confusion Matrix
pred_svm <- predict(model_svm, data = train_svm)
caret::confusionMatrix(data = pred_svm,
                  reference = train_svm$DEFAULT)

#Tuning SVM hyperparameter
tune_svm <- tune(svm, DEFAULT ~.-ID, data = train_svm,
            kernel = "linear", scale = TRUE, probability = TRUE,
            ranges = list(cost = c(0.01, 0.1, 1, 3, 5)),
            tunecontrol = tune.control(sampling = "cross", cross = 3))
summary(tune_svm)
```

For time restriction we have reduced the number of crossovers from 10 to 3 allowing shorter time to process and finish earlier. However, based on observation it would still take more than an hour waiting for the SVM function to finish.

The result of Tuning indicates that best cost parameter is 0.1, with error of 0.222 as shown below.

```
> summary(tune_svm)

Parameter tuning of 'svm':

- sampling method: 3-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.2192857

- Detailed performance results:
  cost     error  dispersion
1 0.01 0.2220000 0.003295018
2 0.10 0.2192857 0.005400302
3 1.00 0.2192857 0.005400302
4 3.00 0.2192857 0.005400302
5 5.00 0.2220952 0.004373700
```

(b) Display model summary and discuss the relationship between the response variable versus selected features. (10 marks)

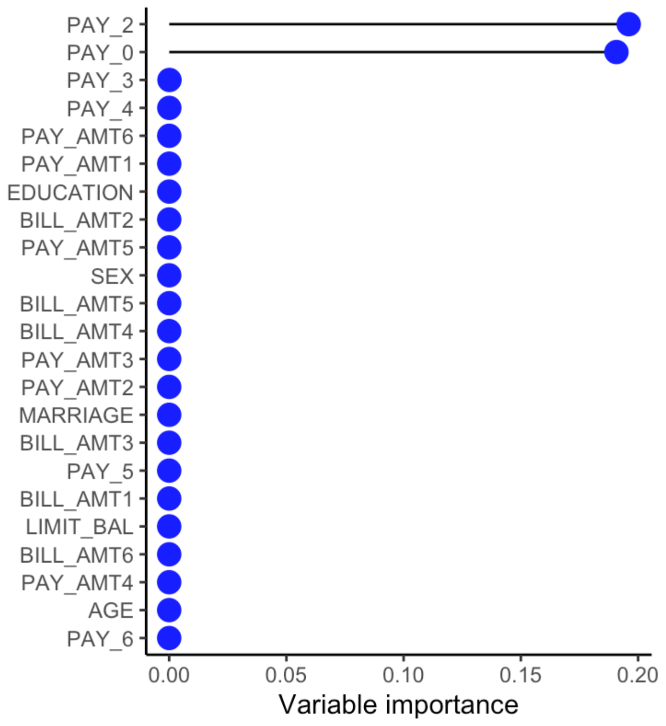
Ans:

```
> summary(model_svm)

Call:
svm(formula = DEFAULT ~ . - ID, data = train_svm, kernel = "linear", probability = TRUE,
    scale = TRUE)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  10122
 ( 5450 4672 )
Number of Classes:  2

Levels:
 0 1
```

```
# Default SVM
> pred_svm <- predict(model_svm, data = train_svm)
> caret::confusionMatrix(data = pred_svm,
+                        reference = train_svm$DEFAULT)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 16285  4610
         1    36    69

               Accuracy : 0.7788
                 95% CI : (0.7731, 0.7844)
    No Information Rate : 0.7772
    P-Value [Acc > NIR] : 0.2953

                  Kappa : 0.0193

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

            Sensitivity : 0.99779
            Specificity : 0.01475
         Pos Pred Value : 0.77937
         Neg Pred Value : 0.65714
             Prevalence : 0.77719
         Detection Rate : 0.77548
   Detection Prevalence : 0.99500
      Balanced Accuracy : 0.50627

       'Positive' Class : 0
```

Now were going to check the variable importance of SVM. It can be assessed using the absolute value of the coefficient weights.

```
#SVM Variable Importance Plot
svm_coefs <- t(model_svm2$coefs) %*% model_svm2$SV
feature <- as.vector(names(train_svm)[2:24])

abs_weight <- as.vector(abs(svm_coefs))
SVM_VarImp_table <- data.frame(feature, importance = abs_weight)

svm_plot <- SVM_VarImp_table %>%
  arrange(importance) %>%
  mutate(feature = factor(feature, levels = feature)) %>%
  ggplot(aes(x = feature, y = importance)) +
  geom_segment(aes(xend = feature, yend = 0)) +
  geom_point(size = 4, colour = "blue") +
  coord_flip() +
  theme_classic() +
  xlab("") +
  ylab("Variable importance")
svm_plot
```

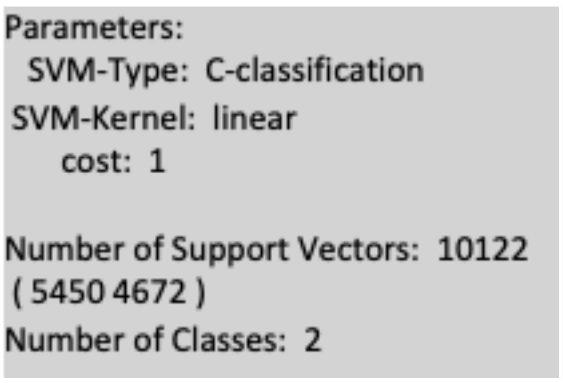


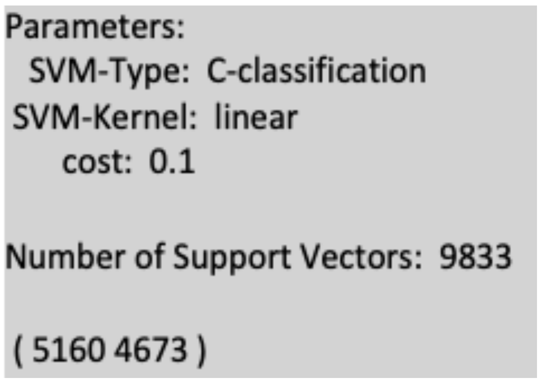
The graph above shows the result of variable imports for SVM. It shows that the model has considered PAY_0 and PAY_2 to be most important variables or features.
This means that the likelihood of credit card to be defaulted increases with the values of repayment status of PAY_0 and PAY_2. This could also mean that since PAY_0 is a categorical feature, it has a linear relationship with the DEFAULT. This is due to the kernel choice.

(c) Evaluate the performance of the algorithm on the training data and comment on the results.

Ans:

The result of the summary(model_svm) shows the default value of Cost =1 shows the number of Support Vectors: 10122 (5450 4672). After Tuning and using Cost=0.1 The number of Support Vectors: 9833 (5160 4673).


```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  10122
( 5450 4672 )
Number of Classes:  2
```


```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  9833

( 5160 4673 )
```

The number of support vectors still depends on how much slack is allowed, but it also depends on the complexity the model. The output of an SVM is the support vectors and an alpha, which in essence is defining how much influence that specific support vector has on the final decision.

The Accuracy depends on the trade-off between a high-complexity model which may over-fit the data and a large-margin which will incorrectly classify some of the training data in the interest of better generalization. The number of support vectors can range from very few to every single data point if you completely over-fit your data. This trade-off is controlled via C and through the choice of kernel and kernel parameters.

In terms of computational complexity, the computational complexity of the model is linear in the number of support vectors. Fewer support vectors mean faster classification of test points. In this case C=0.1 has fewer support vectors which means its faster to finish calculating in R than C=1.

However, with regards to Accuracy, the R result shows that they are the same (77.88%) for both C=1 and on the left and C =0.1 in the right as shown in the confusion matrix below.

```
# Default SVM
> pred_svm <- predict(model_svm, data =
train_svm)
> caret::confusionMatrix(data = pred_svm,
+                    reference =
train_svm$DEFAULT)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 16285  4610
         1    36    69

               Accuracy : 0.7788
                 95% CI : (0.7731, 0.7844)
    No Information Rate : 0.7772
    P-Value [Acc > NIR] : 0.2953

                  Kappa : 0.0193

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

            Sensitivity : 0.99779
            Specificity : 0.01475
         Pos Pred Value : 0.77937
         Neg Pred Value : 0.65714
             Prevalence : 0.77719
         Detection Rate : 0.77548
   Detection Prevalence : 0.99500
      Balanced Accuracy : 0.50627

       'Positive' Class : 0
```

```
#Tuned SVM
>pred_svm2<- predict(model_svm2, data =
train_svm)
> caret::confusionMatrix(data = pred_svm2,
+                    reference =
train_svm$DEFAULT)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 16285  4610
         1    36    69

               Accuracy : 0.7788
                 95% CI : (0.7731, 0.7844)
    No Information Rate : 0.7772
    P-Value [Acc > NIR] : 0.2953

                  Kappa : 0.0193

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

            Sensitivity : 0.99779
            Specificity : 0.01475
         Pos Pred Value : 0.77937
         Neg Pred Value : 0.65714
             Prevalence : 0.77719
         Detection Rate : 0.77548
   Detection Prevalence : 0.99500
      Balanced Accuracy : 0.50627

       'Positive' Class : 0
```

As shown above, while the Accuracy for the SVM is 77.88% is acceptable, Kappa is very low only 0.0193 for both C. This means that there was no agreement between the observed and predicted accuracy. The model was simply guessing randomly according to the frequency of each outcome class. This can be justified in the result of confusion matrix where the model has predicted 99.78% (16285) non-Defaulted observations successfully. However, it only predicted **1.47%** (69) of Defaulted observations since there are significantly less Defaulted observations. Hence, the precision is very low.

```
> confusion_matrix <- data.frame(Accuracy$table)
> knitr::kable(confusion_matrix)

|Prediction |Reference |  Freq|
|:----------|:---------|-----:|
|0          |0         | 16285|
|1          |0         |   36.  |
|0          |1         |  4610 |
|1          |1         |    69.   |
```

2.2.4 Prediction

Apply your fitted models in 2.2.2 and 2.2.3 to make prediction on the test data. Evaluate the performance of the algorithms on test data. Which models do you prefer? Are there any suggestions to further improve the performance of the algorithms? Justify your answers.

Ans:

```
# Applying model_RF to test set performance -
rf_pred <- predict(model_RF2, newdata = test, type = "response")
rf_test_CM <- caret::confusionMatrix(data = rf_pred,
                                     reference = test$DEFAULT)
rf_test_CM
 > rf_test_CM
Confusion Matrix and Statistics
          Reference
Prediction    0    1
         0 6634 1234
         1  409  723
               Accuracy : 0.8174
                 95% CI : (0.8093, 0.8254)
    No Information Rate : 0.7826
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3673

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9419
            Specificity : 0.3694
         Pos Pred Value : 0.8432
         Neg Pred Value : 0.6387
             Prevalence : 0.7826
         Detection Rate : 0.7371
   Detection Prevalence : 0.8742
      Balanced Accuracy : 0.6557

       'Positive' Class : 0
```

```
# Applying model_svm to test set performance -
test2 <- mutate_if(test[1:24], is.factor, as.numeric)
test2$DEFAULT <- test$DEFAULT
svm_pred <- predict(model_svm2, newdata = test2)
svm_test_CM <- caret::confusionMatrix(data = svm_pred,
                                  reference = test$DEFAULT)
svm_test_CM
 > svm_test_CM
Confusion Matrix and Statistics
          Reference
Prediction    0    1
         0 7024 1927
         1   19   30
               Accuracy : 0.7838
                 95% CI : (0.7751, 0.7922)
    No Information Rate : 0.7826
    P-Value [Acc > NIR] : 0.3951

                  Kappa : 0.0195

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

            Sensitivity : 0.99730
            Specificity : 0.01533
         Pos Pred Value : 0.78472
         Neg Pred Value : 0.61224
             Prevalence : 0.78256
         Detection Rate : 0.78044
   Detection Prevalence : 0.99456
      Balanced Accuracy : 0.50632

       'Positive' Class : 0
```

Summarizing all the results of test and Train.

Table 1: Summary of Random Forest and SVM model to Train and Test data.

	Random Forest		SVM	
	Train	Test	Train	Test
Accuracy	0.8186667	0.8174	0.7788	0.7838
Kappa	0.3801196	0.3673	0.0193	0.0195
Sensitivity	0.9458	0.9419	0.99779	0.9973

We will then try to use AUC curve to look on our data.

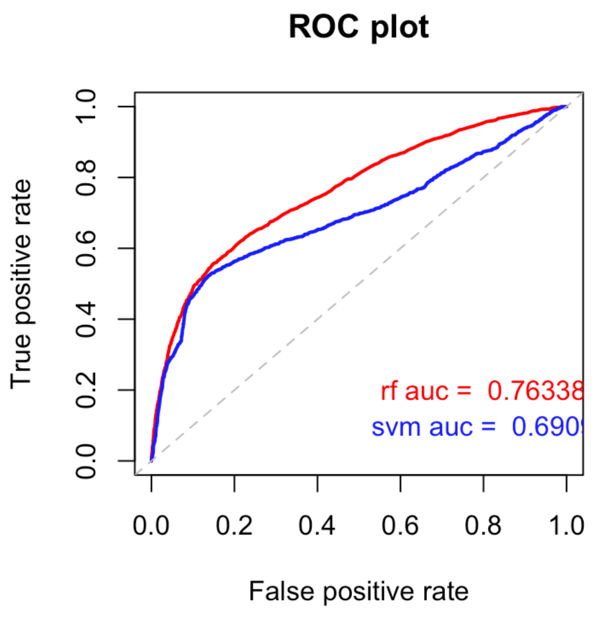
An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate

- False Positive Rate

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve. AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example

```
# Applying AUC and ROCR on model_RF
library(ROCR)
rf_pred2 <- predict(model_RF2, newdata = test, type = "prob")
pred <- prediction(predictions = rf_pred2[,2], labels = test$DEFAULT)
perf_auc_rf <- performance(pred, "auc")
rf_auc <- perf_auc_rf@y.values[[1]]
roc_rf <- performance(pred, "tpr", "fpr")
# Applying AUC and ROCR on model_svm
svm_pred2 <- predict(model_svm2, newdata = test2, probability = TRUE)
pred2 <- prediction(predictions = attr(svm_pred2, "probabilities")[,2],
                labels = test2$DEFAULT)
perf_auc_svm <- performance(pred2, "auc")
svm_auc <- perf_auc_svm@y.values[[1]]
roc_svm <- performance(pred2, "tpr", "fpr")
#plot the ROCR result
plot(roc_rf, main = "ROC plot", col = "red", lwd = 2)
plot(roc_svm, lwd = 2, col = "blue", add = TRUE)
abline(a = 0, b = 1, lty = 2, col = "grey")
text(0.8, 0.2, col = "red",
    paste("rf auc = ", format(rf_auc, digits = 5, scientific = FALSE)))
text(0.8, 0.1, col = "blue",
    paste("svm auc = ", format(svm_auc, digits = 5,
                         scientific = FALSE, col = "blue")))
```



As shown in table 1, The accuracy and kappa values for both Random Forest and SVM were slightly different but overall, this means that the model does not overfit the data during training and both are able to cope up with unforeseen patterns in the test set.

As shown in the ROC plot, the Random Forest model has better and has consistent performance than the SVM.

In terms of running the model in my laptop especially in tuning and training, Random Forest finished faster than the SVM for the Credit Card dataset.

Hence, basing on all the results of the model performance (summary of models and confusion matrix from Table 1.), the ROC Plot and the time to finished in running the model (Tuning RF and SVM). The preferred classification model for the Credit Card dataset is the **Random Forest.**

Appendix:

R code for Assessment 2.

```
install.packages("ggplot2")
library(ggplot2)
install.packages("quadprog")
library(quadprog)
install.packages("tidyverse")

library(tidyverse)
library(readxl)

library(randomForest)
library("dplyr")


x1<-c(3, 4, 3.5, 5, 4, 6, 2, -1, 3, 3, -2, -1)
x2<-c(2, 0, -1, 1, -3, -2, 5, 7, 6.5, 7, 7, 10)
y<-c(-1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1)


#Question 1:
#data<-matrix(c(x1,x2,y), ncol = 3, nrow = 12)
data<-data.frame(x1,x2,y)
colnames(data)<-c("x1","x2","y")
data
ggplot(data=data, aes(x=x2, y=x1, colour= as.factor(y))) +
  geom_point(size=3) +
  scale_x_continuous(breaks = seq(-4,10,2)) +
  scale_color_manual(values =c("blue", "red"))
```

```
#Question 1.2
set.seed(123)
p<-ncol(data)
n<-nrow(data)
eps<- 1e-8

Dmat<- matrix(rep(0,p*p),nrow = p,ncol = p)
diag(Dmat)<-1
Dmat[p,p]<- eps

Amat<- cbind(as.matrix(data[,c(2,1)]), y = rep(1,n))
Amat<- Amat* data$y

dVec<- rep(0,p)
bVec<- rep(1,n)

Opt_hyperplane<- solve.QP(Dmat, dVec, t(Amat), bVec)
Opt_hyperplane$solution
#?solve.QP

betas <- Opt_hyperplane$solution
x2_Opt_hyperplane <- data$x2
x1_Opt_hyperplane <- (-betas[3]-betas[1]*x2_Opt_hyperplane)/betas[2]
Opt_sep_hyperplane <- data.frame(x2_Opt_hyperplane, x1_Opt_hyperplane)

ggplot() +
  geom_line(aes(x2_Opt_hyperplane, x1_Opt_hyperplane)) +
  geom_point(data=data, aes(x2, x1, colour = as.factor(y))) +
  labs(x = "x2", y = "x1") +
  scale_colour_manual(values = c("blue", "red")) +
  scale_x_continuous(breaks = seq(-4,20,2)) +
  scale_y_continuous(breaks = seq(-20,20,4))

#Question 2.1

data2<- read_excel("CreditCard_Data.xls", skip = 1)
str(data2) #checking data types
#convert the parameters below into characters.
data2$SEX<-as.character(data2$SEX)
data2$EDUCATION<-as.character(data2$EDUCATION)
data2$MARRIAGE<-as.character(data2$MARRIAGE)
data2$PAY_0<-as.character(data2$PAY_0)
data2$PAY_2<-as.character(data2$PAY_2)
data2$PAY_3<-as.character(data2$PAY_3)
data2$PAY_4<-as.character(data2$PAY_4)
```

```
data2$PAY_5<-as.character(data2$PAY_5)
data2$PAY_6<-as.character(data2$PAY_6)
data2$`default payment next month`<-as.character(data2$`default payment next
month`)
str(data2) #checking data types again
sum(is.na(data2))

#Convert unknown values to NA's
data2$EDUCATION[data2$EDUCATION == "5"] <- NA
data2$EDUCATION[data2$EDUCATION == "6"] <- NA
data2$EDUCATION[data2$EDUCATION == "0"] <- NA
data2$MARRIAGE[data2$MARRIAGE == "0"] <- NA
na_educ<-sum(is.na(data2$EDUCATION))
na_marri<-sum(is.na(data2$MARRIAGE))
cbind(na_educ,na_marri)

data2<- mutate_if(data2, is.character, as.factor)
data2<- rename(data2, DEFAULT = 'default payment next month') #

summary(data2)


#Question 2.2.1
#Splitting the data2 into training (70%) and test sets
set.seed(123)
ind<- sample(nrow(data2), nrow(data2)*0.7)
train<- data2[ind,]
test<- data2[-ind,]
# print number of observations in test vs. train c(nrow(train), nrow(test))
c(nrow(train), ncol(train))

#Question 2.2.2 (a).
set.seed(123)
train_RF <- na.roughfix(train)
model_RF<- randomForest(DEFAULT ~.-ID, data = train_RF) #time around 40sec
model_RF

#Plotting the result
model_oob_err <- model_RF$err.rate
plot(model_RF, main = "RF model Error Rates") +
legend(x = "right", legend = colnames(model_oob_err),
    fill = 1:ncol(model_oob_err))

#Using hypergrid
hyper_grid <- expand.grid(
  mtry = seq(3, 6, by = 1),
  sampsize = nrow(train_RF)*c(0.632, 0.8, 1.0),
  nodesize = seq(1, 5, by = 1),
  oob_err = 0
)
for(i in 1:nrow(hyper_grid)) {
  tune_rf <- randomForest(DEFAULT ~.-ID, train_RF,
                        mtry = hyper_grid$mtry[i],
                        sampsize = hyper_grid$sampsize[i],
                        nodesize = hyper_grid$nodesize[i],
                        seed = 2020)
  hyper_grid$oob_err[i] <- tune_rf$err.rate[nrow(tune_rf$err.rate), "OOB"]
}
hyper_table <- hyper_grid %>%
  arrange(oob_err) %>%
  head(10)
knitr::kable(hyper_table)


hyper_grid <- expand.grid(
  mtry = seq(3, 5, by = 1),
  sampsize = nrow(train_RF)*c(0.632, 0.8, 1.0),
  nodesize = seq(1, 5, by = 1),
  oob_err = 0
)

for(i in 1:nrow(hyper_grid)) {
  tune_rf <- randomForest(DEFAULT ~.-ID, train_RF,
                        mtry = hyper_grid$mtry[i],
                        sampsize = hyper_grid$sampsize[i],
                        nodesize = hyper_grid$nodesize[i],
                        seed = 2020)
  hyper_grid$oob_err[i] <- tune_rf$err.rate[nrow(tune_rf$err.rate), "OOB"]
}
hyper_table<- hyper_grid%>%
  arrange(oob_err)%>%
  head(10)
knitr::kable(hyper_table)

# 2.2.2 b)
model_RF2<- randomForest(DEFAULT ~.-ID, data = train_RF, importance=TRUE,
                       mtry = 4,
                       sampsize = 13272,
                       nodesize = 1)
model_RF2
#Variable Importance
VarImp_RF<- importance(model_RF2, scale = FALSE)
feature_Var<-as.vector(names(train_RF)[2:24])
VarImp_RF_table<- data_frame(feature_Var, MDA = as.numeric(VarImp_RF[,3]),
                                MDG = as.numeric(VarImp_RF[,4]))

RF_MDA_plot <- VarImp_RF_table %>%
  arrange(MDA) %>%
  mutate(feature_Var = factor(feature_Var, levels = feature_Var)) %>%
  ggplot(aes(x = feature_Var, y = MDA)) +
  geom_segment(aes(xend = feature_Var, yend = 0)) +
  geom_point(size = 4, colour = "blue") +
  coord_flip() +
  theme_classic() +
  xlab("") +
  ylab("Mean Decrease in Accuracy")
RF_MDG_plot <- VarImp_RF_table %>%
  arrange(MDG) %>%
  mutate(feature_Var = factor(feature_Var, levels = feature_Var)) %>%
  ggplot(aes(x = feature_Var, y = MDG)) +
  geom_segment(aes(xend = feature_Var, yend = 0)) +
  geom_point(size = 4, colour = "blue") +
  coord_flip() +
  theme_classic() +
  xlab("") +
  ylab("Mean Decrease in Node Impurity")
gridExtra::grid.arrange(RF_MDA_plot, RF_MDG_plot, ncol = 2)

#Question 2.2.2 (c).
#Confusion Matrix
RF_oob_err <- model_RF2$err.rate
RF_oob_err_final <- RF_oob_err[nrow(RF_oob_err ), "OOB"]
RF_oob_err_model <- model_oob_err [nrow(model_oob_err ), "OOB"]
RF_CM <- caret::confusionMatrix(data = model_RF2$predicted,
                               reference = train_RF$DEFAULT)
RF_overall_metrics <- data.frame(OOB_error = RF_oob_err_final[[1]],
                                Accuracy =RF_CM$overall[[1]],
                                Kappa = RF_CM$overall[[2]])
knitr::kable(RF_overall_metrics)

summary(train_RF)

#Question 2.2.3 (a). SVM
library(e1071)
set.seed(123)
# data pre-processing
train_svm <- mutate_if(train[1:24], is.factor, as.numeric)
train_svm <- as.data.frame(impute(train_svm[1:24], what = "median"))
train_svm$DEFAULT <- train$DEFAULT

model_svm <- svm(DEFAULT ~.-ID, data = train_svm, kernel = "linear",
        scale = TRUE, probability = TRUE)

#Question 2.2.3 (b)
summary(model_svm)
#SVM Confusion Matrix
pred_svm <- predict(model_svm, data = train_svm)
caret::confusionMatrix(data = pred_svm,
           reference = train_svm$DEFAULT)

#tuning SVM parameters
tune_svm <- tune(svm, DEFAULT ~.-ID, data = train_svm,
        kernel = "linear", scale = TRUE, probability = TRUE,
        ranges = list(cost = c(0.01, 0.1, 1, 3, 5)),
        tunecontrol = tune.control(sampling = "cross", cross = 3))
summary(tune_svm)

#Using Tuned C parameters in SVM
model_svm2<- svm(DEFAULT ~.-ID, data = train_svm, kernel = "linear",
        scale = TRUE, probability = TRUE, cost =.1)

summary(model_svm2)

pred_svm2<- predict(model_svm2, data = train_svm)
caret::confusionMatrix(data = pred_svm2,
           reference = train_svm$DEFAULT)
#SVM Variable Importance Plot
svm_coefs <- t(model_svm2$coefs) %*% model_svm2$SV
feature <- as.vector(names(train_svm)[2:24])

abs_weight <- as.vector(abs(svm_coefs))
SVM_VarImp_table <- data.frame(feature, importance = abs_weight)

svm_plot <- SVM_VarImp_table %>%
  arrange(importance) %>%
  mutate(feature = factor(feature, levels = feature)) %>%
  ggplot(aes(x = feature, y = importance)) +
  geom_segment(aes(xend = feature, yend = 0)) +
  geom_point(size = 4, colour = "blue") +
  coord_flip() +
  theme_classic() +
  xlab("") +
  ylab("Variable importance")
svm_plot

# Accuracy and Kappa
Accuracy <- caret::confusionMatrix(data = pred_svm2,
                reference = train_svm$DEFAULT)
```

```
overall_metrics <- data.frame(Accuracy = Accuracy$overall[[1]],
                              Kappa = Accuracy$overall[[2]])
knitr::kable(overall_metrics)

confusion_matrix <- data.frame(Accuracy$table)
knitr::kable(confusion_matrix)

#Question 2.2.4 Prediction
num_na <- sum(is.na(test))
num_na

#data tidying, replace missing values in test set.
getmode <- function(values) {
  uniqv <- unique(values)
  uniqv[which.max(tabulate(match(values, uniqv)))]
}
mode_EDUC <- getmode(train$EDUCATION)
mode_MARRIAGE <- as.numeric(getmode(train$MARRIAGE))
test$EDUCATION <- replace_na(test$EDUCATION, mode_EDUC)
test$MARRIAGE <- replace_na(test$MARRIAGE, mode_EDUC)

# Applying model_RF to test set performance -
rf_pred <- predict(model_RF2, newdata = test, type = "response")
rf_test_CM <- caret::confusionMatrix(data = rf_pred,
                                     reference = test$DEFAULT)
rf_test_CM
# Applying model_svm to test set performance -
test2 <- mutate_if(test[1:24], is.factor, as.numeric)
test2$DEFAULT <- test$DEFAULT
svm_pred <- predict(model_svm2, newdata = test2)
svm_test_CM <- caret::confusionMatrix(data = svm_pred,
                                      reference = test$DEFAULT)
svm_test_CM

# Applying AUC and ROCR on model_RF
library(ROCR)
rf_pred2 <- predict(model_RF2, newdata = test, type = "prob")
pred <- prediction(predictions = rf_pred2[,2], labels = test$DEFAULT)
perf_auc_rf <- performance(pred, "auc")
rf_auc <- perf_auc_rf@y.values[[1]]
roc_rf <- performance(pred, "tpr", "fpr")
# Applying AUC and ROCR on model_svm
svm_pred2 <- predict(model_svm2, newdata = test2, probability = TRUE)
pred2 <- prediction(predictions = attr(svm_pred2, "probabilities")[,2],
                    labels = test2$DEFAULT)
perf_auc_svm <- performance(pred2, "auc")
svm_auc <- perf_auc_svm@y.values[[1]]
```

```
roc_svm <- performance(pred2, "tpr", "fpr")
#plot the ROCR result
plot(roc_rf, main = "ROC plot", col = "red", lwd = 2)
plot(roc_svm, lwd = 2, col = "blue", add = TRUE)
abline(a = 0, b = 1, lty = 2, col = "grey")
text(0.8, 0.2, col = "red",
    paste("rf auc = ", format(rf_auc, digits = 5, scientific = FALSE)))
text(0.8, 0.1, col = "blue",
    paste("svm auc = ", format(svm_auc, digits = 5,
                          scientific = FALSE, col = "blue")))
```