MA5832 Assessment 2

Nikki Fitzherbert 13848336

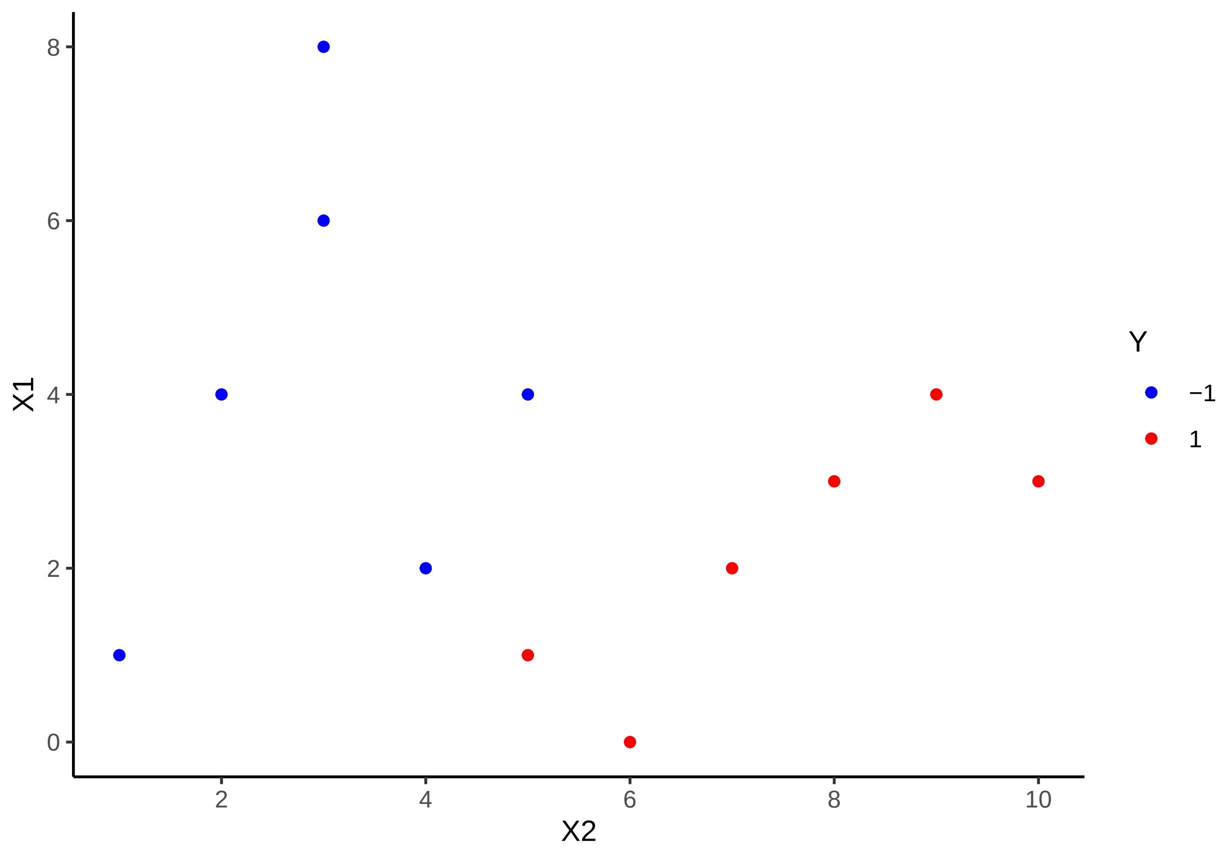
31/05/2020

Part I: An analytical problem

Question 1

The following R code produces a simple scatter plot of the training data with X_1 on the vertical axis, X_2 on the horizontal axis, the $Y = 1$ class in red and the $Y = -1$ class in blue.

```
train <- data.frame(X1 = c(1,2,4,6,4,8,1,2,4,3,3,0),
                    X2 = c(1,4,2,3,5,3,5,7,9,8,10,6),
                    Y = c(rep(-1, times = 6), rep(1, times = 6)))

library(tidyverse)

train %>%
  ggplot(aes(X2, X1, colour = as.factor(Y))) +
  geom_point() +
  scale_colour_manual(values = c("blue", "red"), name = "Y") +
  scale_x_continuous(breaks = seq(0,10,2)) +
  theme_classic()
```



Question 2

Section A If a hyperplane in p dimensions can be defined by the equation:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0$$

where $\beta_0 + \beta_1 + \cdots + \beta_p$ are the parameters, then the equation for the optimal separating hyperplane for the training data displayed in Question 1's scatter plot is:

$$\beta_0 + \beta_1 X2 + \beta_2 X1 = 0$$

Section B According to James et al (2017, p.343)[1], the optimal separating hyperplane is the solution to the optimisation problem:

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{maximise}\ M\ subject\ to\ \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \quad for\ all\ i = 1, \ldots, n)$$

[1]James, G., Witten, D., Hastie, T., & Tibshirani, R., 2013, *An Introduction to Statistical Learning with Applications in R*. New York, NY: Springer.

In R, this can be solved using quadprog's solve.QP function, which implements the dual method of Goldfarb and Idnani (1982, 1983) for solving quadratic programming problems of the form $min(-d^T b + 1/2b^T Db)$ with the constraints $A^T b \geq b_0$[2].

```r
library(quadprog)

set.seed(2020)

p = ncol(train)
n = nrow(train)
eps <- 1e-8

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

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

dvec <- rep(0,p)
bvec <- rep(1,n)

sol <- solve.QP(Dmat, dvec, t(Amat), bvec)
sol$solution
```

```
## [1]  0.9999999 -1.0000001 -2.9999995
```

The output from the R code indicates that, to two decimal places, $\beta_0 \approx -3.00$, $\beta_1 \approx 1.00$ and $\beta_0 \approx -1.00$, which means that the equation for the optimal separating hyperplane for the train data can be expressed as:

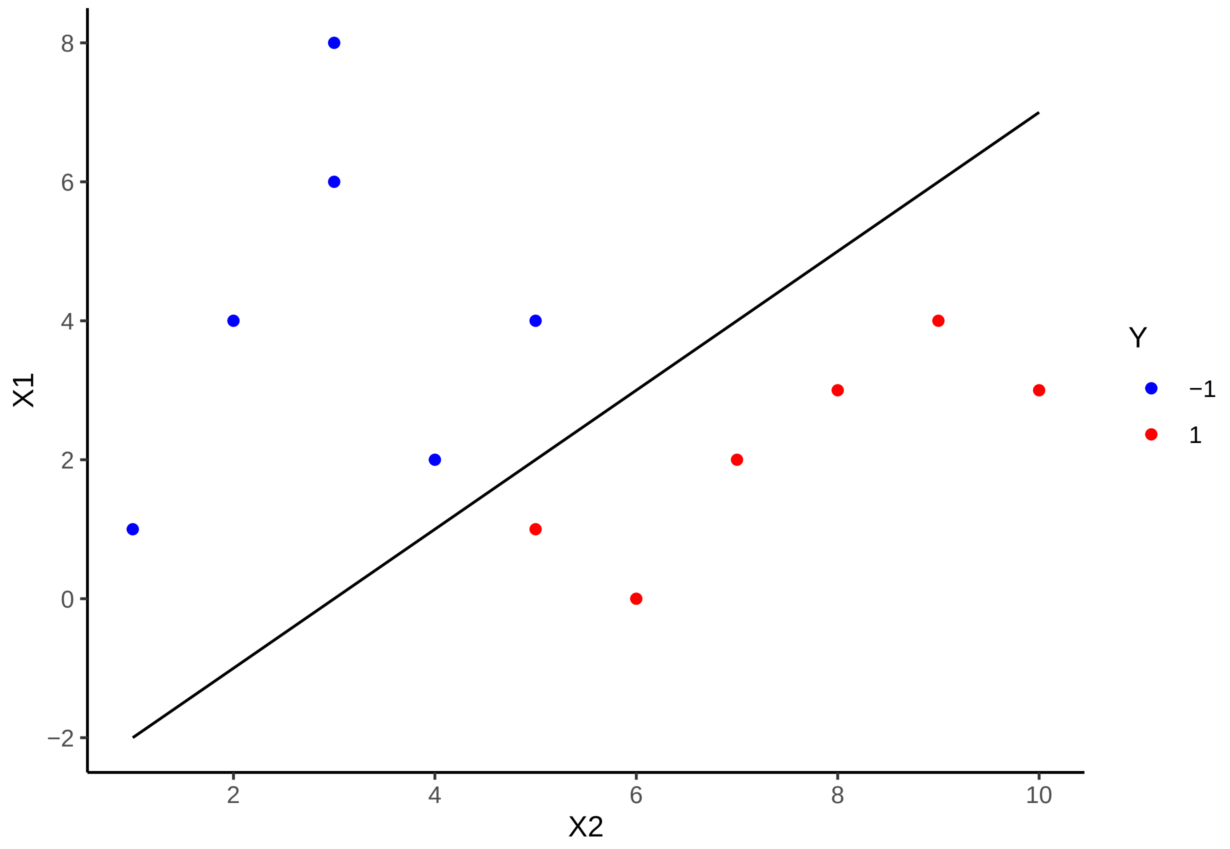
$$-3 + X2 - X1 = 0$$

Section C The R code below adds the optimal separating hyperplane for the train data to the scatter plot from Question 1:

```r
beta <- sol$solution
x2_osh <- train$X2
x1_osh <- (-beta[3]-beta[1]*x2_osh)/beta[2]
osh <- data.frame(x2_osh, x1_osh)

osh %>%
  ggplot() +
  geom_line(aes(x2_osh, x1_osh)) +
  geom_point(data = train, aes(X2, X1, colour = as.factor(Y))) +
  scale_colour_manual(values = c("blue", "red"), name = "Y") +
```

[2]https://www.rdocumentation.org/packages/quadprog/versions/1.5-8/topics/solve.QP

```
  scale_x_continuous(breaks = seq(0,10,2)) +
  scale_y_continuous(breaks = seq(-2,8,2)) +
  labs(x = "X2", y = "X1") +
  theme_classic()
```



Question 3

If the general form of the classification rule for the maximal marginal classifier is:

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

where $y_1, y_2, \dots, y_n \,\epsilon\, \{-1, 1\}$, then the classification rule for the current data can be expressed as: *For any data point, classify to* $Y = -1$ *if* $-3 + X2 - X1 > 0$ *and classify to* $Y = 1$ *if* $-3 + X2 - X1 < 0$.

Question 4

The margin M has a total width of $2 \times \frac{1}{||\beta||} = \frac{2}{||\beta||}$. Given that $\beta \approx (1, -1)$, the margin for the `train` data is $\frac{2}{\sqrt{1^2 \times (-1)^2}} = \frac{2}{\sqrt{2}} \approx 1.41$.

The value of the margin can also be confirmed using R:

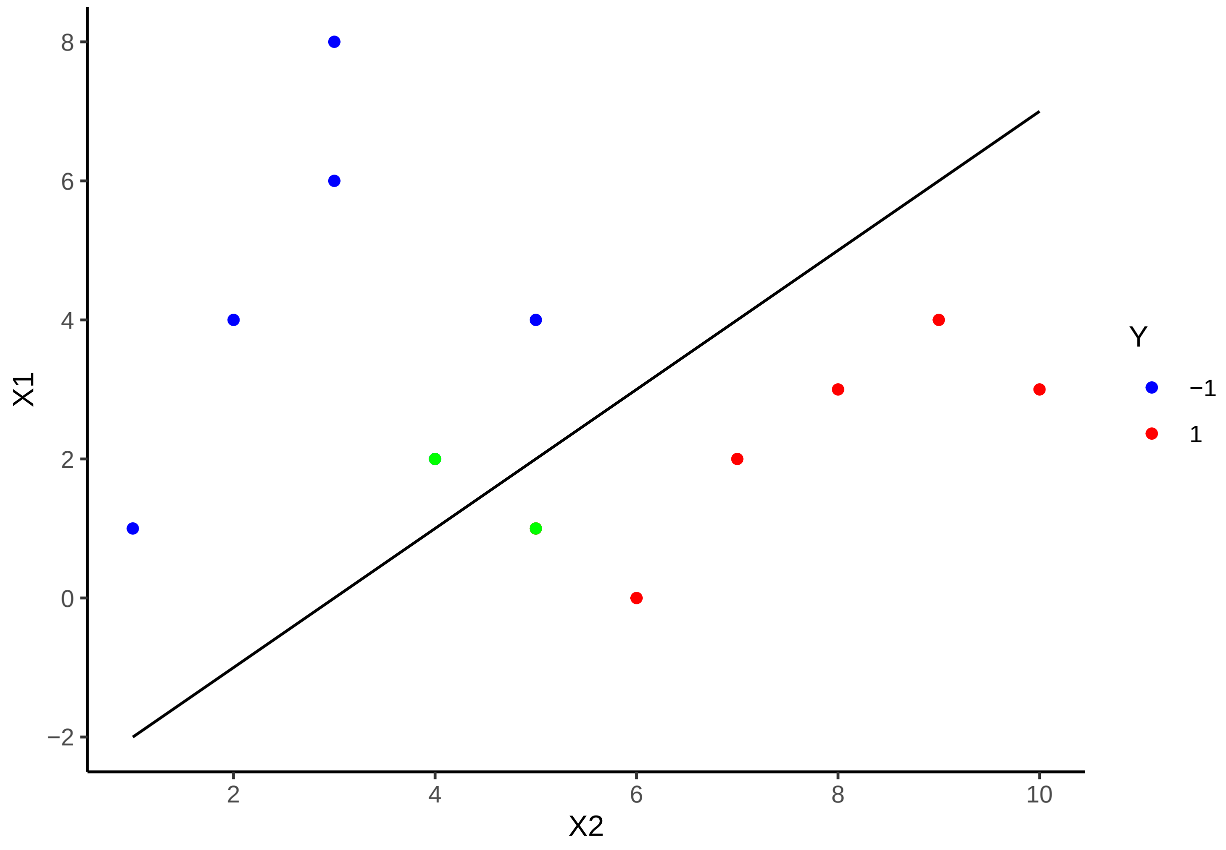
```r
denominator <- sqrt(sol$solution[1]^2 + sol$solution[2]^2)
margin <- 2/denominator
margin
```

```
## [1] 1.414214
```

Question 5

The support vectors for the maximal marginal classifier are the points $(X2, X1) = (5, 1)\ and\ (X2, X1) = (4, 2)$. They are highlighted in green on the scatter plot following the R code:

```r
sv <- data.frame(X1 = c(1,2),
                 X2 = c(5,4))

osh %>%
  ggplot() +
  geom_line(aes(x2_osh, x1_osh)) +
  geom_point(data = train, aes(X2, X1, colour = as.factor(Y))) +
  geom_point(data = sv, aes(X2, X1), colour = "green") +
  scale_colour_manual(values = c("blue", "red"), name = "Y") +
  scale_x_continuous(breaks = seq(0,10,2)) +
  scale_y_continuous(breaks = seq(-2,8,2)) +
  labs(x = "X2", y = "X1") +
  theme_classic()
```



Part 2: An application

2.2.1 Data

The following R code:

a) imports the `CreditCard` data file that will be used to generate and compare a tree-based model to a support vector classifier and performs data pre-processing

b) checks the data for missing values and other factors that might affect model development and interpretation

c) splits the data into a training and test set using a 70:30 ratio and displays the dimensions of the training set

Note that for the code to run the name of the local copy of the `CreditCard` excel file needs to match the filename in the `read_excel` function.

```
library(tidyverse)
library(readxl)

data <- read_excel("Assessment 2_CreditCard_Data.xls", skip = 1,
                   col_types = c("numeric", "numeric", "text", "text",
                                 "text", "numeric", "text", "text",  "text",
```

```r
                                  "text", "text", "text", "numeric",
                                  "numeric", "numeric", "numeric", "numeric",
                                  "numeric", "numeric", "numeric", "numeric",
                                  "numeric", "numeric", "numeric", "text"))

# looking for missing values
paste("Number of missing values:", sum(is.na(data)))
```

```
## [1] "Number of missing values: 0"
```

```r
# marking 'unknown' values as missing and performing a recount
data$EDUCATION[data$EDUCATION == "0"] <- NA
data$EDUCATION[data$EDUCATION == "5"] <- NA
data$EDUCATION[data$EDUCATION == "6"] <- NA
data$MARRIAGE[data$MARRIAGE == "0"] <- NA
```

```
## [1] "There are 345 values missing from `EDUCATION`."
```

```
## [1] "There are 54 values missing from `MARRIAGE`."
```

```r
data <- mutate_if(data, is.character, as.factor)
data <- rename(data, DEFAULT = `default payment next month`)

# splitting the data into training and test sets
set.seed(2020)
train_ind <- sample(nrow(data), nrow(data)*0.7)
train <- data[train_ind, ]
test <- data[-train_ind, ]
```

```
## [1] "In the training dataset there are: 21000 rows, and 25 columns (including ID)."
```

```r
# checking the distribution of observations across the two values of the
#  outcome variable
tbl1 <- data %>%
  group_by(DEFAULT) %>%
  summarise(count = n(), per_cent = round(count/nrow(data)*100,2))

tbl2 <- train %>%
  group_by(DEFAULT) %>%
  summarise(count = n(), per_cent = round(count/nrow(data)*100,2))

tbl3 <- test %>%
  group_by(DEFAULT) %>%
  summarise(count = n(), per_cent = round(count/nrow(data)*100,2))

sum_data <- rbind(tbl1, tbl2, tbl3)
sum_tbl <- data.frame(row.names = c("data", "data ", "train", "train ",
                                "test", "test "),
                    sum_data)
```

	DEFAULT	count	per_cent
data	0	23364	77.88
data	1	6636	22.12
train	0	16370	54.57
train	1	4630	15.43
test	0	6994	23.31
test	1	2006	6.69

An initial calculation indicated that there were no missing observations with missing values in the dataset. However, a closer look at the definition of the individual levels for the categorical features revealed that “0” in both `EDUCATION` and `MARRIAGE` and “5” and “6” in `EDUCATION` indicated that the value was `unknown`. These values were therefore recategorised as missing values and dealt with appropriately for each type of classifier algorithm using imputation if possible or simply omitted as most tree-based ensemble classifer algorithm and support vector machines are able to handle missing values during model construction. The ideal approach was be to use imputation, but as the number of missing values made up less than two per cent of the total dataset, simply omitting them was also considered reasonable.

In addition, data-preprocessing indicated that the dataset was very much unbalanced. Although this is unfortunately a common feature of datasets with a binary or multi-class outcome variable, it can have a significant adverse impact on model performance. Possible approaches to help counteract the effect of class imbalance include choosing alternative cutoff points for the predicted probabilities, adjusting prior probabilities and a variety of sampling methods. However, due to time constraints this issue was not investigated and was left as an area for later research.

A quick look at the names of the variables in the dataset revealed that one of the variables was an `ID` variable. This variable was removed from all subsequent analysis as it would provide no useful information for either algorithm during model training. Therefore, it is possibly more accurate to conclude that the training data has dimensions 21000×24 (rows by columns).

The explanatory features of the `CreditCard` dataset could be divided into four main groups: demographic, payment action history, statement amount history and amount paid history. Each of the features in the latter three groups are almost certainly correlated with each other and therefore it would have been worth considering the effect on model performance if a dimensionality reduction method had been used as part of data pre-processing in order to obtain a (reduced) set of independent features from which to model whether a client would default or not.

2.2.2 Tree-Based Algorithm

Section A

Tree-based algorithms involve stratifying or segmenting the predictor space into a number of different regions for classification or regression problems. They are popular modelling tools because they are simple and easy to interpret and implement. However, models built with individual decision trees often suffer from poor predictive performance and/or high sensitivity to changes in the underlying dataset. As a result, individual decision trees are usually combined with bagging (bootstrap aggregation), boosting or random forest techniques to address both of these issues[3].

Both the random forest and boosting ensemble-methods usually perform better than a bagged tree and both often result in similar predictive performance for different reasons. Random forest introduces a random element into the tree construction process and decorrelates the trees by only allowing a sub-sample of the features to be considered at any one split. This means that the effect of a strong predictor (if one is present) or significant noise in the dataset is reduced. Boosting works similar to bagging, except that each tree is grown sequentially rather than simultaneously, and fit on a modified version of the dataset instead of a bootstrapped sample. As a result, the learning rate of the algorithm is slowed down and the weakest areas of the previous tree can be targeted[4].

For this particular dataset, random forest was chosen over a gradient boosted machine because:

1. Boosting can be prone to over-fitting due to noisy data for example, whereas this is generally not a problem with random forests if a sufficiently large number of trees are used to train the model. The problem of over-fitting can be addressed by introducing a learning rate or shrinkage parameter, commonly denoted by λ, but the value of the parameter is inversely proportional to the number of trees required and therefore increases the computation time required to train a model[5].

2. Boosted decision trees have a larger number of key hyperparameters that require careful tuning in order to maximise model performance, which could require significant computation power and time if the dimensions of the training data are large. In contrast, random forests have fewer key hyperparameters and model performance is not quite as sensitive to their values provided the ones chosen are reasonable.

3. Training a boosting algorithm can in general be far more computationally intensive and time-consuming compared to a random forest algorithm due to the sequential tree construction process as well as the aforementioned calibration of key model hyperparameters.

Much of the information currently available indicates that random forest models work reasonably well 'out of the box' with their default parameters. However, there is also a general consensus that the hyperparameter `mtry` (or the number of variables randomly sampled as candidates at each split) has the largest influence on model perfomance and should be tuned. There is some research

[3]James et al, 2013, *An Introduction to Statistical Learning with Applications in R*. New York, NY: Springer, p.316; Kuhn, M. and Johnson, K., 2016, *Applied Predictive Modeling*. New York, NY: Springer, pp.174-175,369.

[4]James et al, 2013, *An Introduction to Statistical Learning with Applications in R*. New York, NY: Springer, p.321.

[5]Kuhn, M. and Johnson, K., 2016, *Applied Predictive Modeling*. New York, NY: Springer, pp.206.

that indicates `nodesize` (minimum size of the terminal nodes) and `sampsize` (size of the sample to draw) can also influence model performance[6] so these hyperparameters were also included in the tuning process[7].

The number of trees is generally not considered a hyperparameter to be tuned in the same way as `mtry` and its value can be dictated moreso by the computational resources available relative to the size of the dataset. The only general guidance is that it is sufficiently large. In `randomForest` the default value is 500 trees, although Kuhn[8] suggested starting with 1,000 and only increasing if model performance is not yet close to a plateau. Therefore, 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.

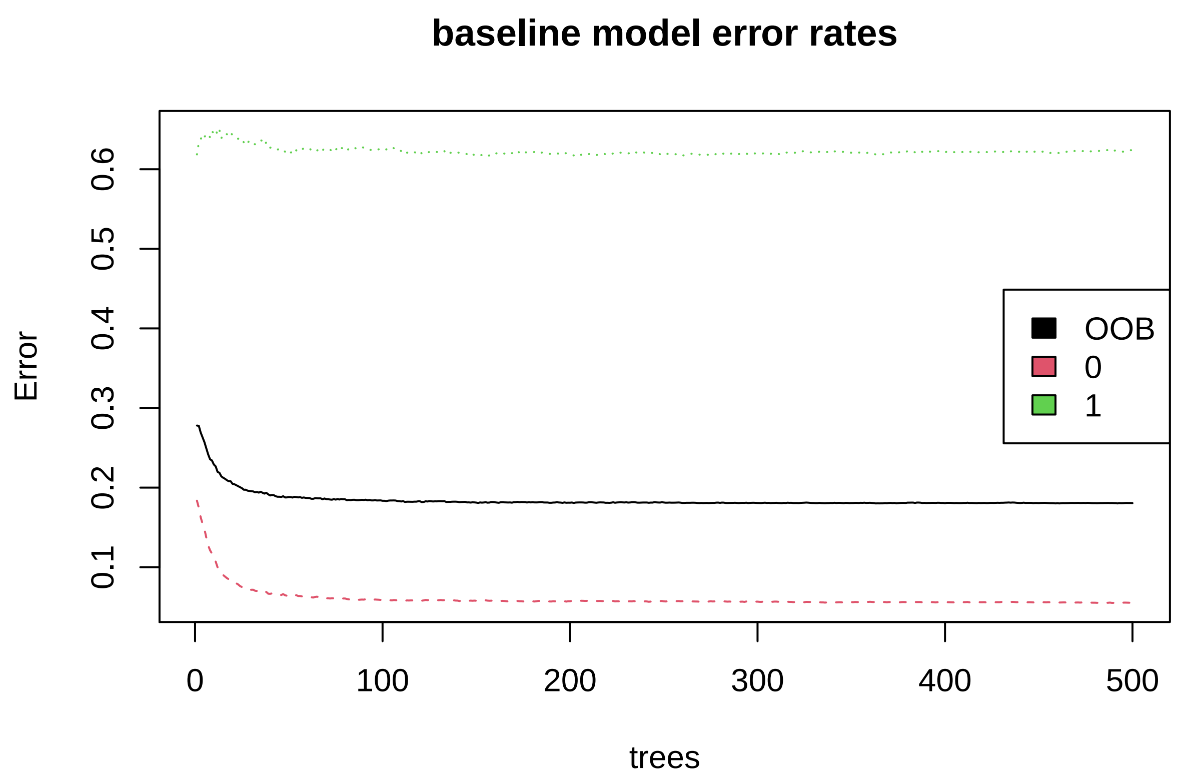
In the first instance, a random forest with all hyperparameters at their default levels (according to the `randomForest` package) was trained in order to obtain a baseline measure of model performance. Missing values were imputed using median/mode imputation. (Imputation using a proximity matrix was not possible in this instance due to computational constraints). The plot of the OOB error rate also indicated that 500 trees was clearly sufficient for the dataset and no significant gain would be achieved by increasing the number.

```r
library(randomForest)

set.seed(2020)
train_imp <- na.roughfix(train)
base_mod <- randomForest(DEFAULT ~.-ID, data = train_imp)
base_mod

base_oob_err <- base_mod$err.rate
plot(base_mod, main = "baseline model error rates")
legend(x = "right", legend = colnames(base_oob_err),
       fill = 1:ncol(base_oob_err))
```

[6]Probst, P., Wright, M. and Boulesteix, A., 2018, Hyperparameters and tuning strategies for random forest, *WIREs: Data Mining and Knowledge Discovery*.

[7]Default values of these hyperparameters for a `randomForest` classification tree are: $\sqrt{p}$ (`mtry`), 1 (`nodesize`) and $nrow(x)$ if sampling with replacement (default) and $max(0.632 \times nrow(x))$ if sampling without replacement (`sampsize`).

[8]Kuhn, M. and Johnson, K., 2016, *Applied Predictive Modeling*. New York, NY: Springer, pp.387.



`randomForest` does not have an in-built tuning function for any hyperparameter except for `mtry`, so the `expand.grid` function and a for-loop was used to optimise the selected set of hyperparameters within the bounds set by the grid. The optimal model was defined as the one that minimised OOB error.

```r
hyper_grid <- expand.grid(
  mtry = seq(3, 10, by = 1),
  sampsize = nrow(train)*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_imp,
                           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)
```

Question B

The result of the tuning process performed at the end of the last section indicated that the optimal random forest model had `mtry` = 3, `sampsize` = 13,272 (63.2 per cent of the training set), and `nodesize` = 3. However, the reduction in OOB error from the baseline model was only relatively small (18.03 per cent compared to 18.05 per cent), which confirmed the previous statements in section A about the random forest algorithm having reasonable 'out of the box' performance.

```
#mtry = hyper_table[1,1]
#sampsize = hyper_table[1,2]
#nodesize = hyper_table[1,3]

# inserting know optimal hyperparameter values due to knitting
#  difficulties
mtry = 3
sampsize = 13272
nodesize = 3

set.seed(2020)
rf_mod <- randomForest(DEFAULT ~.-ID, train_imp, importance = TRUE,
                       mtry = mtry,
                       sampsize = sampsize,
                       nodesize = nodesize)
rf_mod
```

```
##
## Call:
##  randomForest(formula = DEFAULT ~ . - ID, data = train_imp, importance = TRUE,      m
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.03%
## Confusion matrix:
##       0    1 class.error
## 0 15518  852  0.05204643
## 1  2935 1695  0.63390929
```

The `randomForest` package assesses feature importance using the Mean Decrease in Accuracy (MDA) (calculated by permuting OOB data) and the Mean Decrease in Node Impurity (MDI) (average total decrease in node impurity from splitting on the feature)[9].

The plots indicated that the features in the dataset could be divided into three or four main groups for both measures, although the members of those groups differed. Furthermore, the two measures generally disagreed on the relative rankings of the features. One of the few commonalities was the choice of `PAY_0` as the strongest predictor for credit card default. In terms of the other features making up the top six:

- both measures agreed that `BILL_AMT2`, `BILL_AMT3` and `PAY_AMT1` had similar relative ranks

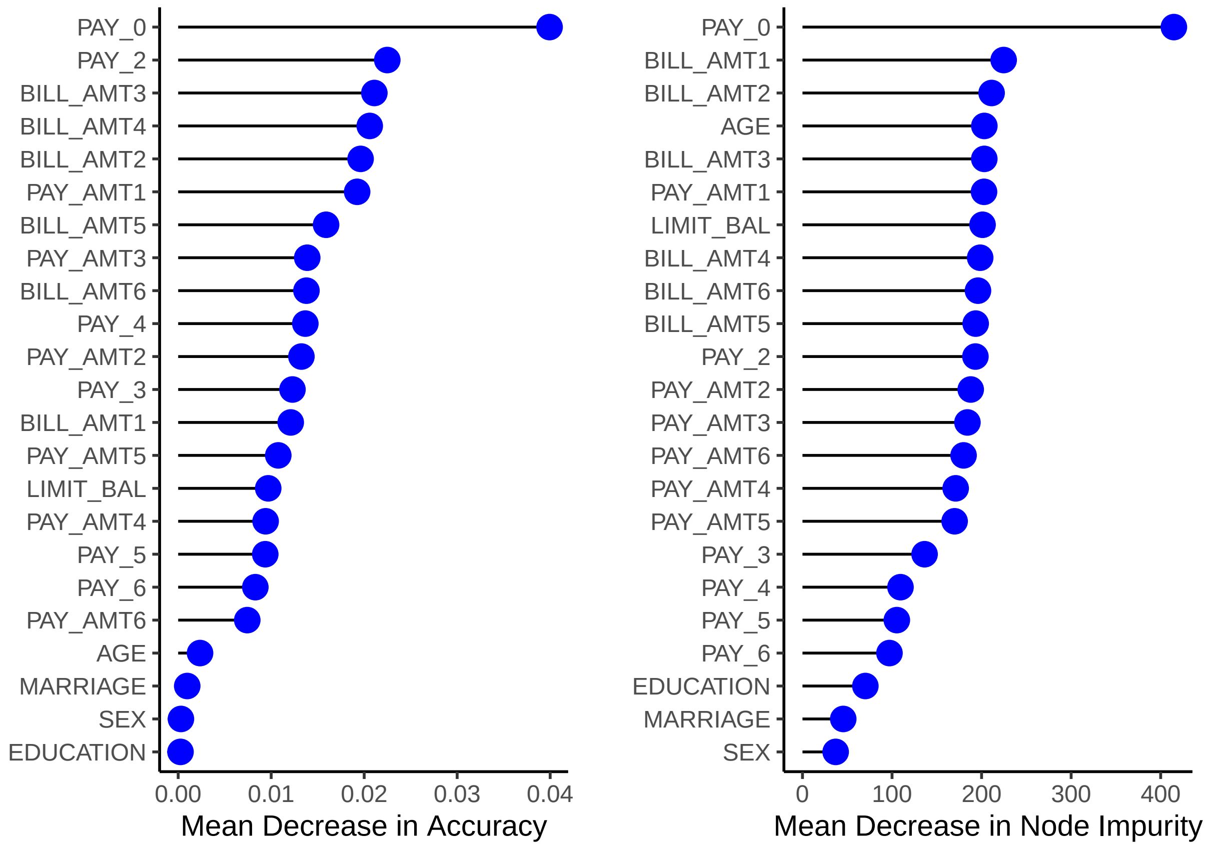
- the MDA ranked `PAY_2` very highly whereas the MDI ranked it relatively low
- the MDI ranked `BILL_AMT1` and `AGE` very highly but the MDA ranked them around the middle and very low respectively
- `BILL_AMT4` was not considered quite as important for the MDI compared to the MDA.

It is also interesting to note that although both measures generally considered the `BILL_AMT` features to be more important to the model than the `PAY_AMT` and `PAY` features, the groupings are not only far clearer in the MDI rankings, but the MDI also ranked all but one of the `PAY_AMT` features above the `PAY` features.

```r
rf_importance <- importance(rf_mod, scale = FALSE)
feature <- as.vector(names(train_imp)[2:24])
rf_imp_table <- data.frame(feature,
                           MDA = as.numeric(rf_importance[,3]),
                           MDG = as.numeric(rf_importance[,4]))

rf_MDA_plot <- rf_imp_table %>%
  arrange(MDA) %>%
  mutate(feature = factor(feature, levels = feature)) %>%
  ggplot(aes(x = feature, y = MDA)) +
  geom_segment(aes(xend = feature, yend = 0)) +
  geom_point(size = 4, colour = "blue") +
  coord_flip() +
  theme_classic() +
  xlab("") +
  ylab("Mean Decrease in Accuracy")

rf_MDG_plot <- rf_imp_table %>%
  arrange(MDG) %>%
  mutate(feature = factor(feature, levels = feature)) %>%
  ggplot(aes(x = feature, y = MDG)) +
  geom_segment(aes(xend = feature, yend = 0)) +
  geom_point(size = 4, colour = "blue") +
  coord_flip() +
  theme_classic() +
```

[9] https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/importance

```
  xlab("") +
  ylab("Mean Decrease in Node Impurity")

gridExtra::grid.arrange(rf_MDA_plot, rf_MDG_plot, ncol = 2)
```



Partial dependency plots (PDPs) provide a graphical depiction of the marginal effects of a feature on the class probability in a classification problem. Due to the large number of features used in the random forest, only the four features that both importance measures agreed were important to predicting if a client would default on their credit card debt, have been plotted and discussed.

`PAY_0` is the repayment status in September 2005. The PDP indicates that clients who delayed their payment for anything longer than a single month (category 1) were more likely to default on their debt, and clients took any other action (such as pay on time or use revolving credit) were more likely to not default.

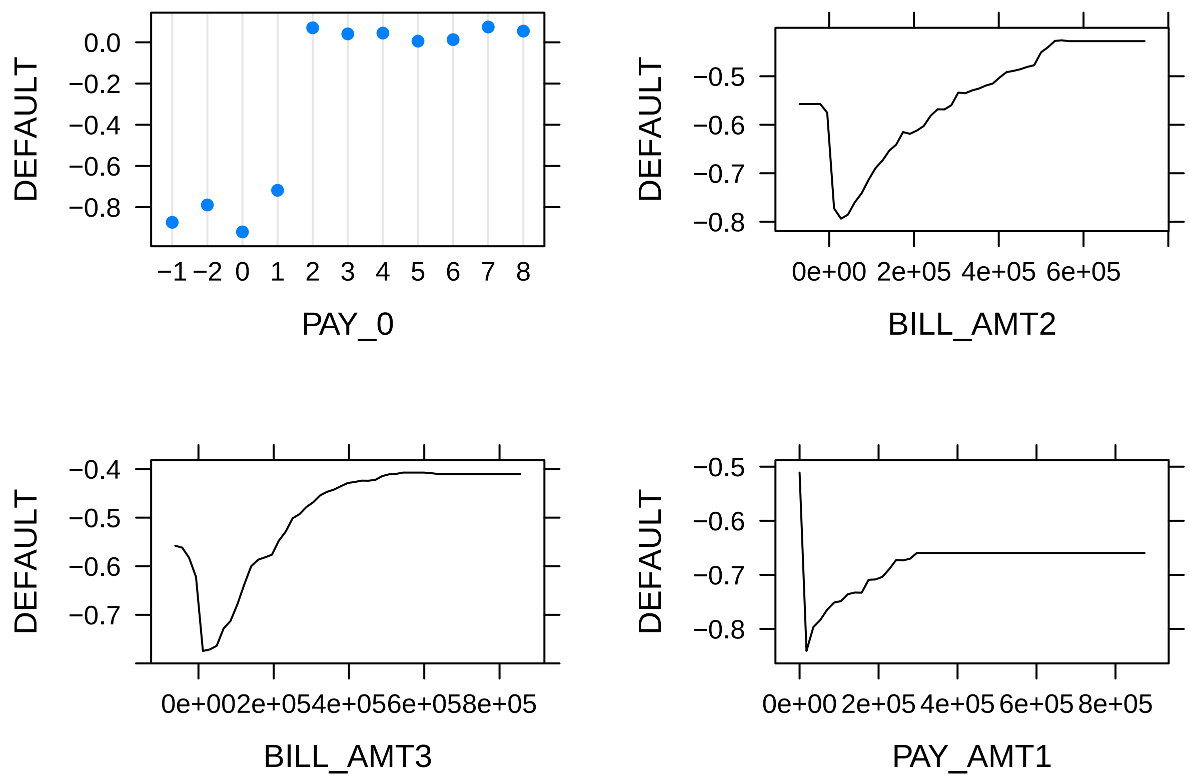
`BILL_AMT2` is the credit card statement bill in August 2005. The PDP indicates that the probability of client credit card default generally monotonically increases as the size of the bill increases up until a bill of 600,000 NT dollars where there was no additional marginal effect on the probability. However, the spareness of the training data at higher values also indicates that any conclusions or predictions made using data in that region should potentially be treated with caution.

`BILL_AMT3` is the credit card statement bill in July 2005. The PDP looks very similar to that of `BILL_AMT2` and therefore would have a similar interpretation; that is, the probability of a client defaulting on their credit card debt generally increases in a monotonic pattern up until about 600,000 NT dollars.

PAY_AMT1 is the amount paid by the client in September 2005. Again, similar to what was observed for the previous two features, the probability of credit card default generally increases monotonically as the amount paid by the client in that month increased. The plot flatlines after about 300,000 NT dollars, which could indicate that the model was unsure about the marginal effect of the feature on the outcome variable at values higher than 300,000 NT dollars. Alternatively, given that the feature was ranked in the top six of both the MDA and MDI feature importance measures, it could indicate that a weaker main effect is masking a stronger interaction effect with at least one other feature in the dataset.

All three numeric features exhibit a curious feature in their PDPs in which the probability of default initially decreases rapidly as the bill amount or amount paid, respectively, increases.

```
library(pdp)

# reducing pdp calculations to only those that will be plotted
feature <- c("PAY_0", "BILL_AMT2", "BILL_AMT3", "PAY_AMT1",
             "PAY_2", "BILL_AMT1", "AGE", "BILL_AMT4")

partials <- list()
for(i in seq_along(feature)){
  partials[[i]] <- partial(rf_mod, pred.var = feature[i], which.class = 2)
}

plots <- list()
for(i in seq_along(feature)){
  plots[[i]] <- plotPartial(partials[[i]], xlab = feature[i],
                            ylab = expression(DEFAULT))
}

gridExtra::grid.arrange(plots[[1]], plots[[2]],
                        plots[[3]], plots[[4]],
                        nrow = 2, ncol = 2)
```



```r
# plotting the other four most importent features (MDA or MDI)
gridExtra::grid.arrange(plots[[5]], plots[[6]], plots[[7]], plots[[8]],
                        nrow = 2, ncol = 2)
```

Question C

After tuning the random forest model, the OOB error on the training dataset was 18.03 per cent compared to 18.05 per cent - a very minor improvement in performance, but overall accuracy on the training data still looked good at 81.97 per cent.

```r
rf_oob_err <- rf_mod$err.rate

final_rf_oob_err <- rf_oob_err[nrow(rf_oob_err), "OOB"]
base_rf_oob_err <- base_oob_err[nrow(base_oob_err), "OOB"]

rf_CM <- caret::confusionMatrix(data = rf_mod$predicted,
                                reference = train_imp$DEFAULT)

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

OOB_error	Accuracy	Kappa
0.1803333	0.8196667	0.3744522

However, these overall figures masked the fact that the apparently recent model performance was almost entirely due to the model's ability to predict which clients did not default on their debt. The model was far less able to predict the clients who *did* default on their credit card debt. Only 5.2 per cent of the credible clients were misclassified as ones who would default on the debt (a model sensitivity of 94.8 per cent). In contrast, 63.4 per cent of non-credible clients were misclassified (a model specificity of 36.6 per cent).

Cohen's Kappa statistic provides a quantitative measure of the magnitude of agreement between observers, or in this case, between the actual and predicted outcome. It can therefore be of more value in assessing model performance when the dataset is unbalanced than other metrics such as accuracy or precision. There is no standardardised way to interpret its value, but according to a popularly cited scale[10], a value of 37.4 per cent indicates that there was only fair agreement between the observed and actual accuracy or that the model was only performing 37.4 per cent better than would have been expected if it was simply guessing at random according to the frequency of each outcome class.

```r
confusion_matrix <- data.frame(rf_CM$table)

rf_class_metrics <- data.frame(Sensitivity = rf_CM$byClass[[1]],
                               Specificity = rf_CM$byClass[[2]],
                               Precision = rf_CM$byClass[[5]],
                               Recall = rf_CM$byClass[[6]])
knitr::kable(confusion_matrix)
```

Prediction	Reference	Freq
0	0	15518
1	0	852
0	1	2935
1	1	1695

```r
knitr::kable(rf_class_metrics)
```

Sensitivity	Specificity	Precision	Recall
0.9479536	0.3660907	0.8409473	0.9479536

The by-class misclassification rates are not entirely surprising given that the outcome classes were

[10]Viera, A. and Garrett, J., 2005, Understanding interobserver agreement: the Kappa statistic, *Family Medicine*, 37(5), p.362.

already very unbalanced as was mentioned in the opening section of this analysis. Only 22 per cent of the total 21,000 observations in the training set were non-credible clients. It is likely that the unbalanced nature of the dataset will have implications for the performance of the random forest algorithm on the test set, which will be assessed in the final section of the analysis.

2.2.3 Support-Vector Classifiers

Section A

Support vector classifiers are a class of model that have been around for many years and attempt to partition a feature space into two or groups by trying to find an optimal means of separating those grops based on their known class labels. Support vector classifiers are a generalisation of a simple and initutive classifier known as a *maximal maginal classifier*, which perfectly separate data into two classes using a linear hyperplane optimally located furthest from the data. In many cases, there is no way to perfectly separate the data into two classes using a linear hyperplane and so the maximal marginal classifer was generalised into the support vector classifier to account for this type of situation. Support vector classifiers can be more robust to individiual observations and more accurately classify most of the training observations compared to maximal marginal classifiers[11].

Support vector machines (SVMs) are in turn an extension of a support vector classifier that accomodate situations in which the class boundaries are non-linear using a concept known as a kernel. Kernels are functions that quantify the similarity of two observations and are what make support vector machine algorithms incredibly flexible. In fact, a support vector classifier is therefore just a special case of a support vector machine when a linear kernel is chosen.

Due to their extreme flexibility, SVMs do also come with a number of downsides and extra considerations[12] compared to other classification algorithms such as random forests:

1. SVMs are both prone to overfitting the data and sensitive to the choice of kernel parameters. Therefore model hyperparameters such as the choice of kernel function, relevant kernel function parameters, and cost value need to be tuned appropriately.
2. The input dataset needs to be centred and scaled so that features with large magnitudes do not dominate the calculations and skew the results. This need to normalise the data is due to the presence of the dot product in the kernel function.
3. The incorporation of a kernel function means that the interpretation of model results can be difficult if not impossible.

Nevertheless, a SVM with a linear kernel was chosen to model the data for the following reasons:

1. A SVM can accomodate a wider variety of decision boundary types (relative to the simpler support vector classifier).

[11] James et al, 2013, *An Introduction to Statistical Learning with Applications in R*. New York, NY: Springer, pp.344-345.

[12] Kuhn, M. and Johnson, K., 2016, *Applied Predictive Modeling*. New York, NY: Springer, p.347.

2. One of the objectives of this analysis is to determine how important the individual features are to the model and investigate their marginal effect on the outcome variable. This type of analysis is only possible with a linear kernel as the other kernels transform the data into another feature space that cannot be related back to the input feature space. The implicit assumption is that the decision boundary between the two classes is generally linear, such that an optimal hyperplane exists that will correctly divide the majority of observations into their correct outcome classes based on the given feature set.
3. It has already been observed that model tuning is somewhat time-intensive for this dataset and the linear kernel has only one parameter to tune - the cost parameter.

A linear kernel has the form:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij}x_{i'j}$$

The only key hyperparameter required to optimise model fit for a linear kernel is the cost parameter. This is the regularisation term in the Lagrange formulation and controls the size of the penalty for outcome misclassification. Lower values of C encourage a larger margin and therefore a simpler decision function, but at the cost of training accuracy. In contrast, a higher value of C encourages a smaller margin, but a higher penalty if the algorithm incorrectly classifies an observation. The default value of C in `e1071` is 1.

The training of the SVM model willl generally follow the same process as the random forest, with allowances for factors such as differences in model output and appropriate model performance metrics. As with the training of the random forest model, a SVM with all hyperparameters at their default levels (according to the `e1071` package) was trained in order to obtain a baseline indication of model performance. Missing values were imputed using median/mode imputation.

Unlike the random forest algorithm, the SVM algorithm is unable to handle factors (except for the outcome variable) so all such features were converted to numeric features after imputation.

```r
library(e1071)

set.seed(2020)
# data pre-processing (imputation and feature conversion)
train_imp <- mutate_if(train[1:24], is.factor, as.numeric)
train_imp <- as.data.frame(impute(train_imp[1:24], what = "median"))
train_imp$DEFAULT <- train$DEFAULT

set.seed(2020)
base_mod <- svm(DEFAULT ~.-ID, data = train_imp, kernel = "linear",
                scale = TRUE, probability = TRUE)
summary(base_mod)

pred <- predict(base_mod, data = train_imp)
caret::confusionMatrix(data = pred,
                      reference = train_imp$DEFAULT)
```

```
# turning off model tuning and using base model for knitting
# model tuning using 3-fold cross-validation
set.seed(2020)
tune_svm <- tune(svm, DEFAULT ~.-ID, data = train_imp,
                 kernel = "linear", scale = TRUE, probability = TRUE,
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10)),
                 tunecontrol = tune.control(sampling = "cross", cross = 3))
summary(tune_svm)
```

Question B

Due to computational restrictions, the number of folds in the tuning cross-validation was reduced from its default of 10 to three. The result of the tuning process performed at the end of the last section indicated that the optimal support vector machine had `cost` = 0.01. Nevertheless, there was no change in the training data statistics indicating that either the default cost parameter was reasonable, or that the optimal value for the dataset was outside of grid provided in the tuning process.

```
#svm_mod <- tune_svm$best.model
svm_mod <- base_mod
summary(svm_mod)
```

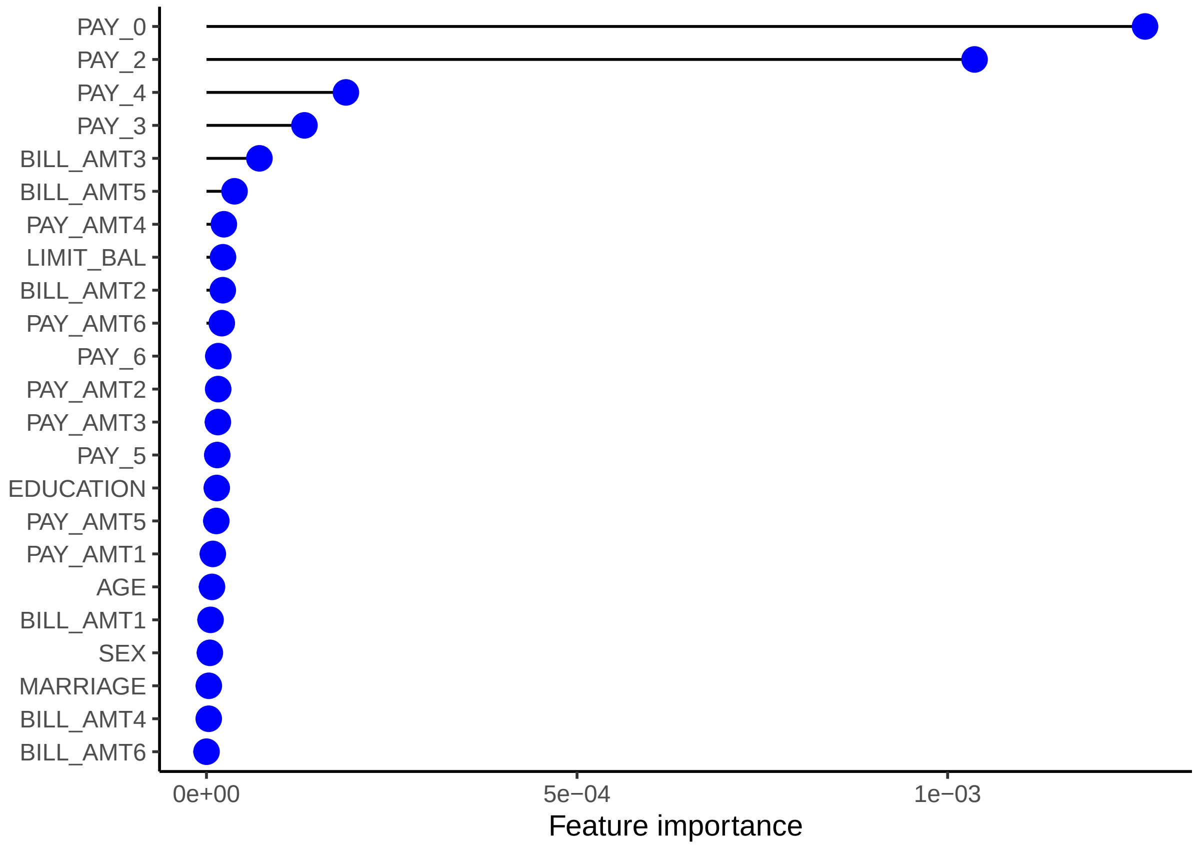
```
##
## Call:
## svm(formula = DEFAULT ~ . - ID, data = train_imp, kernel = "linear",
##     probability = TRUE, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  10096
##
##  ( 5466 4630 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
pred <- predict(svm_mod, data = train_imp)
caret::confusionMatrix(data = pred,
                                   reference = train_imp$DEFAULT)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction     0     1
##          0 16370  4630
##          1     0     0
##                                           
##                Accuracy : 0.7795          
##                  95% CI : (0.7739, 0.7851)
##     No Information Rate : 0.7795          
##     P-Value [Acc > NIR] : 0.5039          
##                                           
##                   Kappa : 0               
##                                           
##  Mcnemar's Test P-Value : <2e-16          
##                                           
##             Sensitivity : 1.0000          
##             Specificity : 0.0000          
##          Pos Pred Value : 0.7795          
##          Neg Pred Value :    NaN          
##              Prevalence : 0.7795          
##          Detection Rate : 0.7795          
##    Detection Prevalence : 1.0000          
##       Balanced Accuracy : 0.5000          
##                                           
##        'Positive' Class : 0               
## 
```

In a linear SVM, relative feature importance can be assessed using the absolute value of the coefficient weights. The plot indicates that the model considered `PAY_0` and `PAY_2` to be the most important features by far, followed distantly by `PAY_4` and `PAY_3`. Except for `PAY_0`, this is quite different to the relative rankings of the same features in the random forest model.

```
coef_w <- t(svm_mod$coefs) %*% svm_mod$SV
feature <- as.vector(names(train_imp)[2:24])

abs_weight <- as.vector(abs(coef_w))
feature <- as.vector(names(train_imp)[2:24])
svm_imp_table <- data.frame(feature, importance = abs_weight)

svm_plot <- svm_imp_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("Feature importance")
svm_plot
```



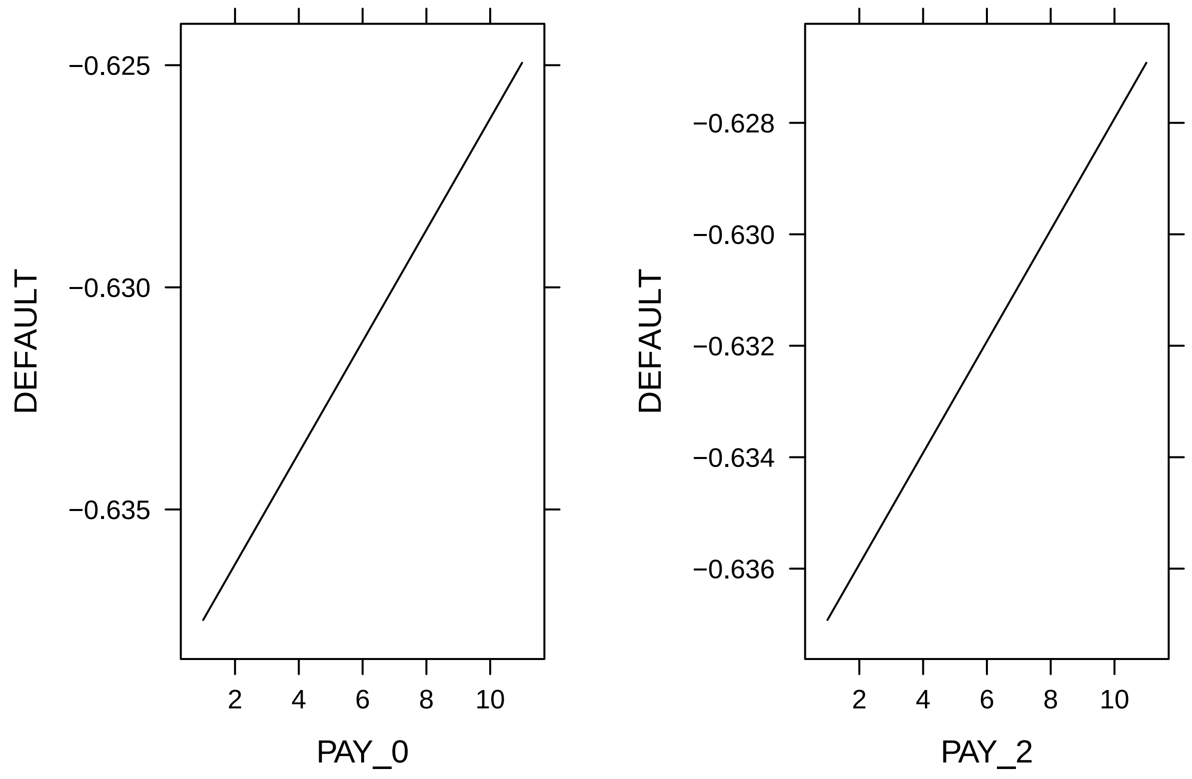
Given that the top two features dominated the feature importance rankings, the partial dependencies for only these two variables were calculated and plotted. However, it would be a simple exercise to extend the code to cover all 23 features.

As expected, the partial dependency plots indicated the likelihood of credit card default increased with the values of repayment status in September 2005 (`PAY_0`) and August 2005 (‘PAY_2’).

It is somewhat unlikely that categorical features such as `PAY_0` have a linear relationship with whether or not a client defaults on their credit card debt, but recall that this relationship was forced by the choice of kernel in order to be able to obtain the relative importance of each explanatory feature.

```
library(pdp)
```

```r
feature = c("PAY_0", "PAY_2")

partials <- list()
for(i in seq_along(feature)){
  partials[[i]] <- partial(svm_mod, pred.var = feature[i], which.class = 2)
}

plots <- list()
for(i in seq_along(feature)){
  plots[[i]] <- plotPartial(partials[[i]], xlab = feature[i],
                            ylab = expression(DEFAULT))
}

gridExtra::grid.arrange(plots[[1]], plots[[2]], ncol = 2)
```



Question C

As observed previously, the performance metrics of the SVM did not change after tuning the cost parameter and overall accuracy was slightly lower than the random forest model at 77.95 per cent.

```
svm_CM <- caret::confusionMatrix(data = pred,
                                 reference = train_imp$DEFAULT)

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

Accuracy	Kappa
0.7795238	0

However, whilst the overall accuracy of the model may be acceptable, it masked the fact that the class-specific misclassification rate for the clients that *did* default on their credit card debt was abysmal. In short the model placed every observation into the majority class. Another indication of the model's poor predictive performance on the training data was the value of the Kappa statistic; the zero value meant that there was absolutely no agreement between the observed and predicted accuracy and the model was performing no better than if it had been simply guessing at random according to the frequency of each outcome class.

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

Prediction	Reference	Freq
0	0	16370
1	0	0
0	1	4630
1	1	0

```
svm_class_metrics <- data.frame(Sensitivity = svm_CM$byClass[[1]],
                                Specificity = svm_CM$byClass[[2]],
                                Precision = svm_CM$byClass[[5]],
                                Recall = svm_CM$byClass[[6]])
knitr::kable(svm_class_metrics)
```

Sensitivity	Specificity	Precision	Recall
1	0	0.7795238	1

2.2.4 Prediction

The true indication of a model's ability to adequately capture the nuance a dataset and in this situation, be able to accurately predict whether a client will default on their credit card debt, comes from testing it on a set of unknown observations. Before this could be done, all the missing values in the test set needed to be filled with their appropriate values from the training set. Since `EDUCATION` and `MARRIAGE` are both factors, this meant filling with the mode.

```r
# replacing missing values with training set mode
num_na <- sum(is.na(test))

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)

# assessing rf model performance - overall metrics
rf_pred <- predict(rf_mod, newdata = test, type = "response")
rf_test_CM <- caret::confusionMatrix(data = rf_pred,
                                     reference = test$DEFAULT)

rf_test_metrics <- rbind(rf_test_CM$overall[[1]], rf_test_CM$overall[[2]],
                         rf_test_CM$byClass[[1]], rf_test_CM$byClass[[2]],
                         rf_test_CM$byClass[[5]], rf_test_CM$byClass[[6]])

rf_train_metrics <- rbind(t(rf_overall_metrics[,-1]), t(rf_class_metrics))

rf_metrics <- data.frame(train = rf_train_metrics, test = rf_test_metrics)

# assessing svm model performance - overall metrics
test2 <- mutate_if(test[1:24], is.factor, as.numeric)
test2$DEFAULT <- test$DEFAULT

svm_pred <- predict(svm_mod, newdata = test2)
svm_test_CM <- caret::confusionMatrix(data = svm_pred,
                                      reference = test$DEFAULT)

svm_test_metrics <- rbind(svm_test_CM$overall[[1]],
                          svm_test_CM$overall[[2]],
                          svm_test_CM$byClass[[1]],
                          svm_test_CM$byClass[[2]],
                          svm_test_CM$byClass[[5]],
                          svm_test_CM$byClass[[6]])

svm_train_metrics <- rbind(t(svm_overall_metrics),
                                   t(svm_class_metrics))

svm_metrics <- data.frame(train = svm_train_metrics,
                          test = svm_test_metrics)
```

```r
# predicted values = rows, actual values = columns
knitr::kable(rf_test_CM$table)
```

	0	1
0	6640	1312
1	354	694

```r
knitr::kable(svm_test_CM$table)
```

	0	1
0	6994	2006
1	0	0

```r
knitr::kable(rf_metrics)
```

	train	test
Accuracy	0.8196667	0.8148889
Kappa	0.3744522	0.3559672
Sensitivity	0.9479536	0.9493852
Specificity	0.3660907	0.3459621
Precision	0.8409473	0.8350101
Recall	0.9479536	0.9493852

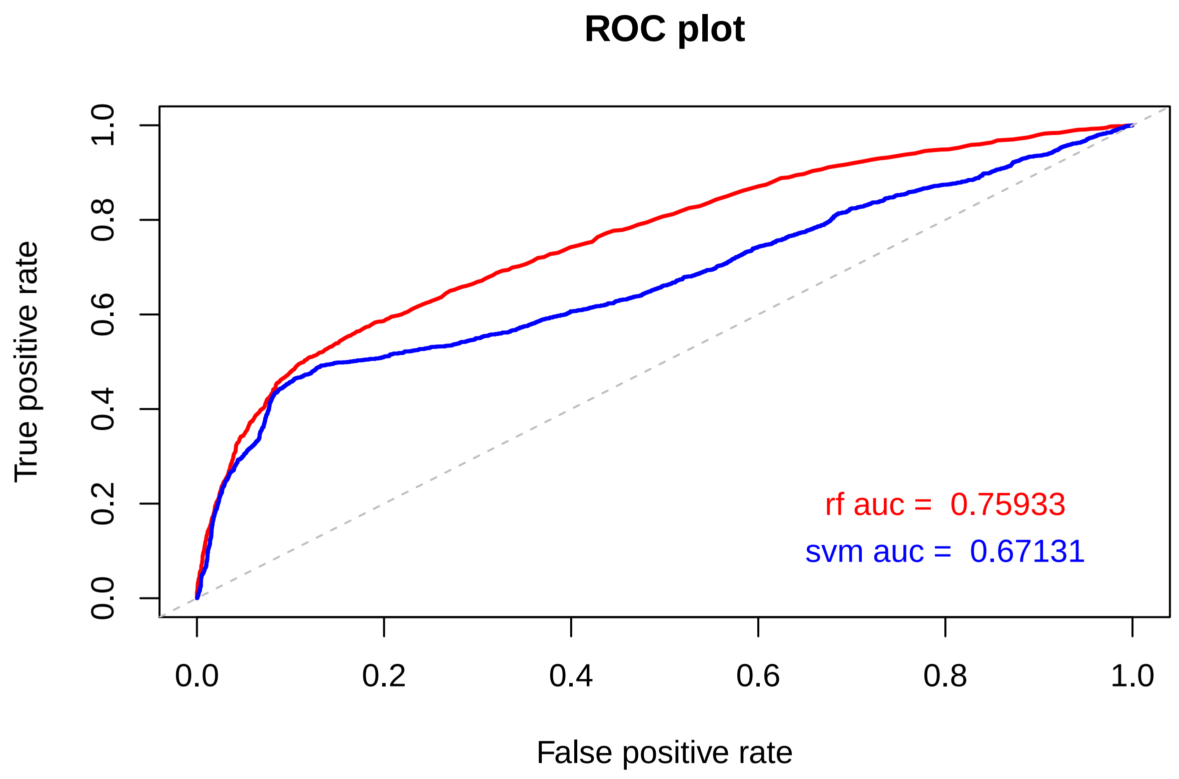
```r
knitr::kable(svm_metrics)
```

	train	test
Accuracy	0.7795238	0.7771111
Kappa	0.0000000	0.0000000
Sensitivity	1.0000000	1.0000000
Specificity	0.0000000	0.0000000
Precision	0.7795238	0.7771111
Recall	1.0000000	1.0000000

The first method used to assess the performance of the random forest model was to look at the overall accuracy of the models, including the Kappa values. First, the acccuracy and Kappa values only dropped slightly between the training and test sets for both the random forest and SVM algorithms. This indicated that neither model overfit the data during the training stage and were both able to cope with any new patterns or relationships that might have been present in the test set but not the training set.

The second metric was to look at the area under the curve (AUC). The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. It therefore provides an indication of the rate of successful classification by a machine learning model. The Receiver Operator Characteristic (ROC) Curve plot indicates that the random forest model is the better classifier for this data, which is consistent with all the other evidence gathered throughout this analysis.

```r
# assessing rf model performance - AUC and ROCR
library(ROCR)
rf_pred2 <- predict(rf_mod, 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")

# assessing svm model performance - AUC and ROCR
svm_pred2 <- predict(svm_mod, 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")

# plotting the ROCR curves
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")))
```



Based on all the evidence from the training process and the assessment of model performance on the test set, the preferred model is the random forest for the following reasons:

1. The random forest was not as computationally intensive to train and tune compared to the SVM for the size of the dataset being analysed.

2. In order to obtain insights on feature importance for example, a linear SVM was required, which therefore assumed that a linear model adequately described the relationship between the outcome variable and the explanatory features. This was a big assumption, and likely influenced the poorer performance of the SVM compared to the random forest. The conclusions might well have been different if the model had been allowed to accomodate a non-linear decision boundary using another kernel such as the Gaussian radial basis function.

3. Irrespective of the reasons for the SVM kernel choice, the SVM generally performed worse in all respects compared to the random forest. For example, the model completely ignored the minority class in its predictions, classifing all observations in both the training and test sets.