Supervised Classification

SVM vs. GBM

Biljana Simonovikj

26/01/2021

Libraries used in this Assessment are:

```
# for solving quadratic programming problem
library(quadprog)
# for modeling
library(funModeling)
library(Hmisc)
library(xgboost)
library(caret)
library(gbm)
# for visualizations
library(ggplot2)
library(knitr)
library(kableExtra)
library(readxl)
library(dplyr)
library(tidyverse)
library(corrplot)
library(GGally)
library(AppliedPredictiveModeling)
library(e1071)
```

1 Part I: An analytical problem (25 marks)

Note: Only solve.QP() from quadprog in R is allowed in Part I. No other packages containing svm are allowed.

We consider a training data with 12 observations with two dimensions, (X_1, X_2). For each observation, there is an associated class label $Y = \{-1, 1\}$ as follows

X_1	X_2	Y
3	2	-1
4	0	-1
3.5	-1	-1
5	1	-1
4	-3	-1
6	-2	-1
2	5	1
-1	7	1
3	6.5	1
3	7	1
-2	7	1
-1	10	1

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.

```
# Assign 3 variables (X1, X2 and Y) as atomic vectors
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(rep(-1, 6), rep(1, 6))

# Create data frame and assign Y as factor
df <- data.frame(cbind(X1,X2,Y))
df$Y <- as.factor(df$Y)

# List types of each variable
sapply(df, class)
```

```
##        X1        X2         Y
## "numeric" "numeric"  "factor"
```

```
# Draw a scatter plot to show the points with red colour for class Y = 1 and blue color for Y = -1
ggplot(df, aes(x=X2, y=X1, color=Y)) +
    geom_point(shape=19, size = 2.5) +
    geom_text(aes(x = X2 + 0.3, label = rownames(df))) +
          scale_color_manual(values=c('dodgerblue','red')) +
          scale_x_continuous(name = "X2", breaks = c(-3:10),
               labels = scales::comma, position = "bottom") +
          scale_y_continuous(name ="X1",breaks = c(-3:10),
               labels = scales::comma, position = "left") +
            theme_minimal() +
               labs(title = "Scatter Plot of X1 vs X2") +
               theme(plot.title = element_text(hjust = 0.5, face = 'bold'))
```

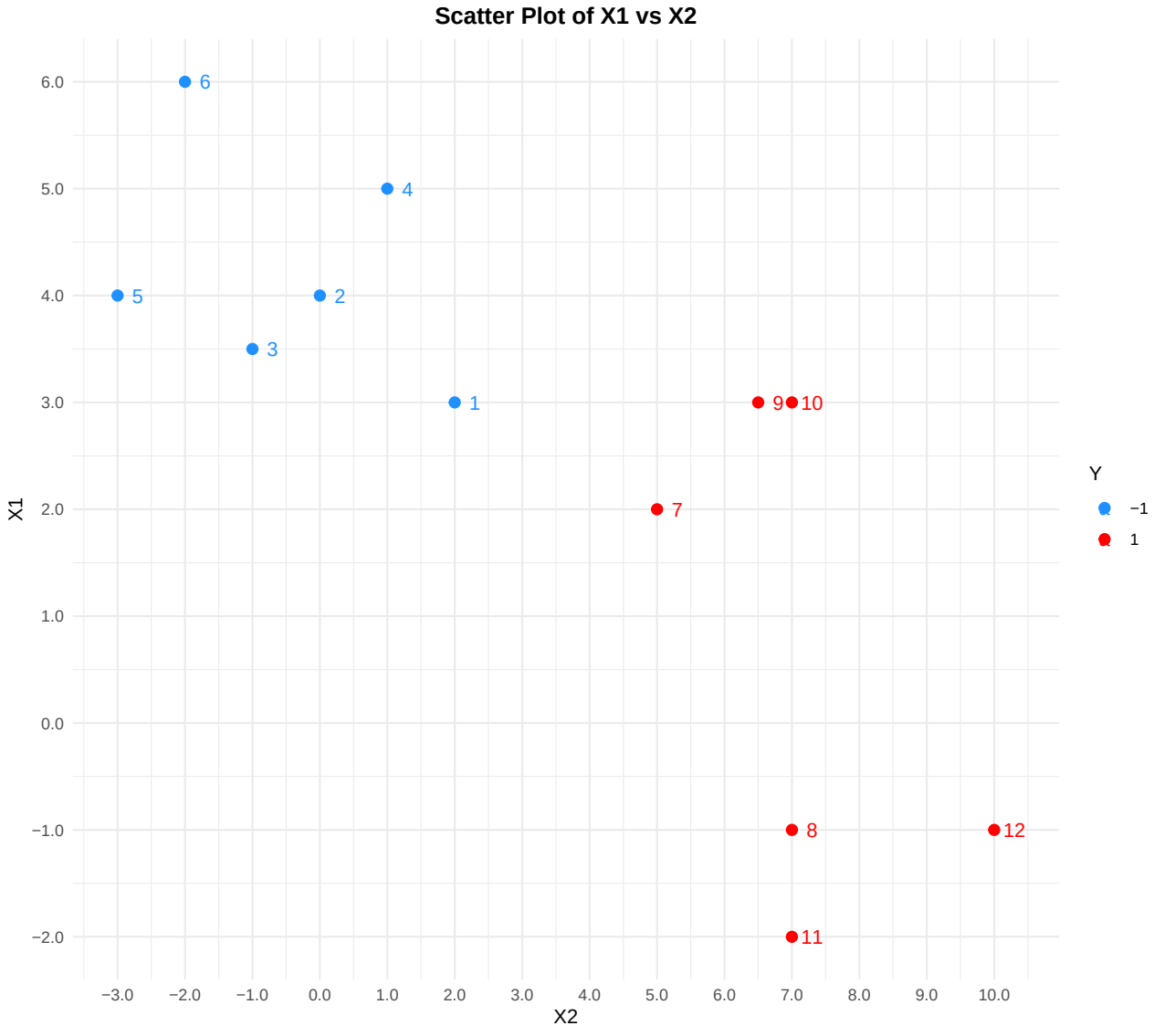

Figure 1: There are two classes of observations labeled with blue and red. As shown in the plot, the *optimal separating hyperplane* has to be between the observations (2, 3) and (5, 2).

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

Finding the optimal separating hyperplane of the given linearly separable dataset $(x_i, y_i)_{i=1}^N$ with input vectors $x_i \in X \subseteq R$ and labels $y_i \in (-1, 1)$ can be formulated as a convex quadrating programming problem which can be solved with function solve.QP() from quadprog in R. From the documentation in quadprog package (Hornik (2009)) in R, the function solve.QP() implements the dual method of (Goldfarb and Idnani (1982)) and (Goldfarb and Idnani (1983)), for solving strictly convex quadratic programming problems of the equation:

$$min(-d^T b + 1/2 b^T D b) \tag{1}$$

with the constraints: $A^T b >= b_0$

where:

1. b is the column vector of decision variables of $x_1, ...x_n$
2. D is the square matrix (Hessian) in the middle of the quadratic objective function
3. d is the column vector specifying the linear part in the quadratic objective function

This mathematical formulation demonstrates an approach to find the minimum of a quadratic function for a given linear set. The matrix D and vector d are used to define any quadratic objective function on these variables, while the matrix-vector couples (A, b_0) and (A, b) respectively define inequality and equality constraints.

In R, the solve.QP() can be used to solve quadratic programming problem by the following code:

$$solve.QP(Dmat,\ dvec,\ Amat,\ bvec,\ meq = 0,\ factorized = FALSE)$$

The arguments of the function are:

- $Dmat$ - matrix to be minimized
- $dvec$ - vector to be minimized
- $Amat$ - matrix defining the constraints to minimize the quadratic function
- $bvec$ - vector of coefficients
- meq - refers to top equality constrain
- $factorized$ - logical flag: if TRUE, then the $Dmat$ will pass inverse of R where $D = R^T R$ instead of hessian matrix D in the argument $Dmat$

The value of the function is a list with the following components:

- solution - vector containing the solution of the quadratic programming problem
- value - scalar, the value of the quadratic function at the solution
- unconstrained.solution - vector containing the unconstrained minimizer of the quadratic function
- iterations vector of length 2, the first component contains the number of iterations the algorithm needed, the second indicates how often constraints became inactive after becoming

active first

- Lagrangian - vector with the Lagragian at the solution
- iact - vector with the indices of the active constraints at the solution

(**Hint**: R **might send you an message about the matrix is not positive definite. You can address this problem by adding a small value on the diagonal of the matrix, e.g.** $1e^{-8}$).

The R code for solving this problem is adopted from article called "More of quadratic programming in R", written by R. Walker, 2015.

```
# Convert the data into matrices
X <- as.matrix(df[,c('X2', 'X1')])
y <- as.matrix(Y)
n <- dim(X)[1]
```

The quadratic programming routine that implements the dual method mentioned above requires that the D matrix is symmetric positive definite. We can overcome this obstacle by diagonal perturbance of matrix D with a very small value of $eps = 1e^{-8}$ and obtain positive definite matrix.

```
# Adding a small value to ensure matrix D is positive definite
eps <- 1e-8

# Parameter mapping
# Create square (n x n) positive definite (Hessian) matrix
Q <- sapply(1:n, function(i) y[i]*t(X)[,i])
# Quadratic coefficients D are mapped to Dmat
D <- t(Q)%*%Q

# Linear coefficients d are mapped to dvec
d <- matrix(1, nrow=n)

# Constraint equalities or inequalities b0 are provided in bvec
# Column vector b0 for the decision variables
b0 <- rbind( matrix(0, nrow=1, ncol=1) , matrix(0, nrow=n, ncol=1) )

# Constraints matrix A is mapped to Amat
A <- t(rbind(matrix(y, nrow=1, ncol=n), diag(nrow=n)))

# Solve the problem using solve.QP()
# Parameter meq= 1 sets the firs n entries as equality constraints; all
# further constraints are inequality
qp_solve <- solve.QP(D +eps*diag(n), d, A, b0, meq=1, factorized=FALSE)
qp_solution <- matrix(qp_solve$solution, nrow=n)

# Function to calculate hyperplanes for plotting
hyperplane <- function(a, Y, X){
 nonzero <-  abs(a) > 1e-1
```

```
  W <- rowSums(sapply(which(nonzero), function(i) a[i]*Y[i]*X[i,]))
  b <- mean(sapply(which(nonzero), function(i) X[i,]%*%W- Y[i]))
  beta1 <- b - 1
  beta2 <- b + 1
  slope <- -W[1]/W[2]
  intercept <- b/W[2]
  intercept_upper <- beta1/W[2]
  intercept_lower <- beta2/W[2]
  return(c(intercept,slope, intercept_upper, intercept_lower))
}
hyperplane <- hyperplane(qp_solution, Y, X)

# Define slope and intercepts for optimal separating hyperplane (decision boundary)
# and the lower and upper margin of the decision boundary
intercept = hyperplane[1]
slope = hyperplane[2]
intercept_upper = hyperplane[3]
intercept_lower = hyperplane[4]

# Display results
df_results <- data.frame(slope, intercept, intercept_upper, intercept_lower)
colnames(df_results) <- c("Slope", "Intercept", "Intercept_UM", "Intercept_LM")
knitr::kable(df_results,
        caption = "Slope and intercept values for the decision boundary
        including upper (Intercept_UM) and lower margins(Intercept_LM) for
        the classification problem solved by SVM/QP classifier",
        align = 'l', format = "markdown")
```

Table 1: Slope and intercept values for the decision boundary including upper (Intercept_UM) and lower margins(Intercept_LM) for the classification problem solved by SVM/QP classifier

	Slope	Intercept	Intercept_UM	Intercept_LM
X2	3.000001	-8.000005	-3.000002	-13.00001

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

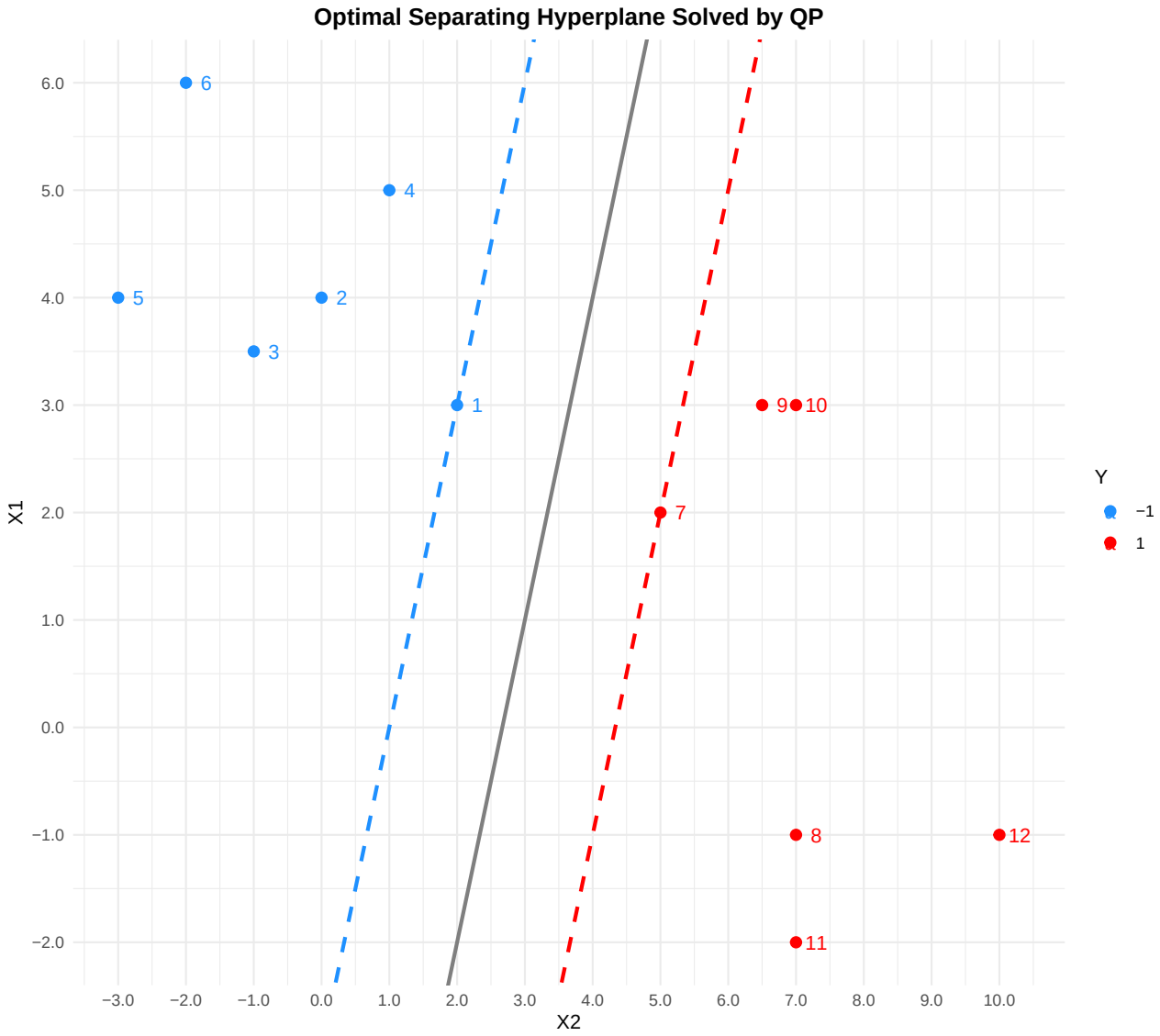

Figure 2: Two classes of observations are shown in blue and in red, along with the *optimal separating hyperplane* or *maximal margin hyperplane*. The hyperplane is shown as a solid line and the margins are shown as dashed lines. The margin is the distance from the solid line to either of the dashed lines. Red observations: 8, 11, 12, 9 and 10 are on the correct side of the margin and observation 7 is on the margin. Blue observations: 2, 3, 4, 5 and 6 are on the correct side of the margin and observation 1 is on the margin. No observations are on the wrong side of the hyperplane or the margin. The two points (1 and 7) that lie on the dashed lines are known as the support vectors.

We showed that the two-dimensional linearly separable data can be separated by a line perfectly according to their class labels defined with the function of the line: $y = a \times x + b$. We can rename x with x_1 and y with x_2 and we can get:

$$a \times x_1 - x_2 + b = 0 \tag{2}$$

If we define $x = (x_1, x_2)$ and $w = (a, -1)$, we get:

$$w \times x + b = 0 \ \ or \ \ w^T \times x + b = 0 \tag{3}$$

It is a equation of the hyperplane or *affine*, mathematically it is equation of a line. The vector w which represents the weights is normal to the hyperplane, and $\frac{b}{\|w\|}$ is the perpendicular distance from the hyperplane to the origin. The vector x represents the observation values and b represents the intercept y or β_0. Support vectors are points that are located the closest to the separating hyperplane and the minimal perpendicular distance between a point and the hyperplane is called margin.

The functional margin uses the measure $y_i(x_i \times w + b)$ while the geometric margin measures the Euclidean distance from the point to the hyperplane (Sra, Nowozin, and Wright (2012)). The margin of the hyperplane is the minimum geometric margin of all correctly classified points. If our data can be perfectly separated by a hyperplane, then there will exist an infinite number of hyperplanes.

We showed that it is possible to construct two parallel hyperplanes in R^n which are orthogonal to w but with different intercepts, that contain no data points between them. Among the infinite number of hyperplanes, the pair (w, b) for which $\| \ w\|^2$ is minimal is the one for which the separation is the greatest.

The aim of SVM is to orientate the separating hyperplane as far as possible from the closest members of both classes in order to become maximal margin hyperplane (or optimal separating hyperplane) which has the farthest minimum distance from the closest observations. Therefore, the maximal margin hyperplane is the separating hyperplane that maximizes the margin.

SVM introduces a fixed functional half-margin of 1, so the points from the our dataset can be described by the following equations:

$$x_i \times w + b \geqslant 1 \ \ for \ \ y_i = 1 \tag{4}$$

$$x_i \times w + b \leqslant -1 \ \ for \ \ y_i = -1 \tag{5}$$

These equations can be combined into:

$$y_i \times (x_i \times w + b) - 1 \geqslant 0 \quad \forall_i \tag{6}$$

Planes, where support vectors lie, are called decision boundaries and can be described by:

$$x \times w + b = 1 \tag{7}$$

$$x \times w + b = -1 \tag{8}$$

In general, in order to optimize the hyperplane, the perpendicular line between the two support vector hyperplanes can be calculated as the maximum distance lying halfway along the perpendicular line. As a result, we have equal maximum distances on either side of the optimal separating hyperplane. Thus, the optimal separating hyperplane can be estimated by either calculating half the difference between the two intercept values or by calculating half the perpendicular distance between the two hyperplanes which is where the optimal separating hyperplane is located.

The idea of finding the maximal margin hyperplane, while keeping the points classified correctly, can be formalized in the following way:

$$\min_{w,b} \frac{1}{2} \| w\|^2 \tag{9}$$

$$subject\ to\ y_1(w \times x_i + b)$$

This problem is referred to as a primal problem and a Lagrangian function associated with this problem is:

$$L = \frac{1}{2} \| w\|^2 + \sum_{i=1}^{N} \alpha_i (1 - y_i \times (w \times x_i + b)) \tag{10}$$

This optimization problem can be converted into the dual form, which is a convex quadratic problem where the objective function depends on Lagrangian multipliers α_i:

$$\min_{\alpha} \Phi(\alpha) = \min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j (x_i \times x_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i \tag{11}$$

$$subject\ to\ \alpha_i \leqslant 0$$

$$\sum_{i=1}^{N} y_i \alpha_i = 0 \tag{12}$$

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

The classification rule for the maximal margin classifier will be something along the lines: “Classify to Red if:

$$13X_2 + 3X_1 - 8 > 0 \tag{13}$$

and classify to Blue otherwise”. The found optimal separating hyperplane separates the two classes and maximizes the distance to the closest point from either class (Vapnik and Lerner (1963), Vapnik and Chervonenkis (1964)). This provides a unique solution to the separating hyperplane problem and also it maximizes the margin between the two classes on the analyzed data. The classification rule determines into which class each point will get classified or whether each point is going to lie

above or below the optimal separating hyperplane. Each point will receive a value greater or less than zero when applying the following equation:

$$\beta_0 + X_2 - Y \tag{14}$$

In our example, the value of X_1 represents Y because it lies on the y-axis. Therefore, the classification rule will be:

$$\beta_0 + \beta_1 X_2 - Y = +\ value\ \ for\ \ X_{2i}\ \ having\ \ the\ \ class\ \ 1 \tag{15}$$

$$\beta_0 + \beta_1 X_2 - Y = -\ value\ \ for\ \ X_{2i}\ \ having\ \ the\ \ class\ \ -1 \tag{16}$$

Applying these equations to the dataset gives the following values:

```
df$Class_Rule <-  intercept + slope * df$X2 - df$X1
knitr::kable(df,
        caption = "Generated classification rule values indicating membership of one
        of two classes for the dataset used in analysis with SVM/QP classifier",
        align = 'c', format = "markdown")
```

Table 2: Generated classification rule values indicating membership of one of two classes for the dataset used in analysis with SVM/QP classifier

X1	X2	Y	Class_Rule
3.0	2.0	-1	-5.000002
4.0	0.0	-1	-12.000005
3.5	-1.0	-1	-14.500006
5.0	1.0	-1	-10.000004
4.0	-3.0	-1	-21.000009
6.0	-2.0	-1	-20.000008
2.0	5.0	1	5.000002
-1.0	7.0	1	14.000005
3.0	6.5	1	8.500004
3.0	7.0	1	10.000005
-2.0	7.0	1	15.000005
-1.0	10.0	1	23.000009

We can see that all the negative class rule values correspond to a class label of -1 and all positive class rule values match with a class label of 1.

4. Compute the margin of the classifier.

In order to computing the maximal margin hyperplane based on a set of n observations $X_1, \ldots,$ $X_n \in R^p$ and associated class labels $Y_1, ..., Y_n \in (-1, 1)$, we have to consider that the maximal margin hyperplane is the solution to the optimization problem:

$$\max_{\beta_0,\beta_1,...\beta_p} M \tag{17}$$

$$subject\ to \sum_{j=1}^{p}\ \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geqslant M \qquad \forall_i \tag{18}$$

The equation(17) is defining a margin M by tuning the coefficients of all the observations and as a result the margin is maximized. The equation (18) shows that the multiplication of class of each observation with its equation should be greater than margin. These two equations ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane. The minimal perpendicular distance between a point and the hyperplane is called margin.

```
# 4. Compute the margin of the classifier.
distance<- function(points, slope, intercept){
  a <- c(1, intercept + slope)
  b <- c(-intercept/slope, 0)
  x1 <- a - b
  x2 <- points - a
  x3 <- cbind(x1, x2)
  x4 <- sqrt(sum(x1 * x1))
  x5 <- det(x3)
  Margin <- x5/x4
  return(Margin)
}
for (i in 1:n){
  df$Margin[i] <- distance(c(df[i,1],df[i,2]), slope, intercept)
}

knitr::kable(df,
        caption = "Display of margin values and classification rule values
        indicating membership of one of two classes obtained with SVM/QP classifier",
        align = 'c', format = "markdown")
```

Table 3: Display of margin values and classification rule values indicating membership of one of two classes obtained with SVM/QP classifier

X1	X2	Y	Class_Rule	Margin
3.0	2.0	-1	-5.000002	-0.3162279
4.0	0.0	-1	-12.000005	1.2649108
3.5	-1.0	-1	-14.500006	1.1067967
5.0	1.0	-1	-10.000004	1.8973665
4.0	-3.0	-1	-21.000009	2.2135937
6.0	-2.0	-1	-20.000008	3.7947327
2.0	5.0	1	5.000002	-2.2135941
-1.0	7.0	1	14.000005	-5.6920994
3.0	6.5	1	8.500004	-1.7392522
3.0	7.0	1	10.000005	-1.8973660
-2.0	7.0	1	15.000005	-6.6407827
-1.0	10.0	1	23.000009	-6.6407823

2.1 Background on Credit Card Dataset

The data, **"CreditCard_Data.xls"**, is based on Yeh and 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. The description of 23 explanatory variables is as follows:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- X2: Gender (1 = male; 2 = female).
- X3: Education (0 = unknown; 1 = graduate school; 2 = university; 3 = high school; 4 = others; 5 = unknown; 6 = unknown).
- X4: Marital status (0 = unknown; 1 = married; 2 = single; 3 = others).
- X5: Age (year).
- X6 - X11: History of past payment. The data was tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .; X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -2= no consumption, -1=pay duly, 0 = the use of revolving credit; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
- X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .; X23 = amount paid in April, 2005.

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. (**5 marks**)

Import the Dataset

The dataset is provided in an excel file and has been imported into R for the analysis. The readxl package makes it easy to get data out of Excel and into R.

```
# Load the dataset
CreditCard <- read_excel("/Users/Biljana/Datasets/Credit Card/CreditCard_Data.xls",
                         sheet = 1, skip = 1)
```

Analyze Data

The objective of this step is to better understand the problem. Lets start by confirming the dimensions of the data.

```
# Dimensions of data
dim(CreditCard)
```

```
## [1] 30000    25
```

We can see that we have 30 000 rows and 25 attributes. We can preview the first 6 rows to see what we are working with.

```
# Peek
head(CreditCard[,1:10])
```

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
1	20000	2	2	1	24	2	2	-1	-1
2	120000	2	2	2	26	-1	2	0	0
3	90000	2	2	2	34	0	0	0	0
4	50000	2	2	1	37	0	0	0	0
5	50000	1	2	1	57	-1	0	-1	0
6	50000	1	1	2	37	0	0	0	0

Table 1: Glimpse at the first 6 rows of the first ten columns of the CreditCard dataset.

We can see that the sample number (ID) is probably not going to be useful and we are going to remove it. Demographic variables (SEX, EDUCATION, MARRIAGE and AGE) are also removed for the analysis this time. For modeling is more useful to work with the data as numbers than factors. We can see that all of the input attributes are integers (positive and negative) and there are a lot of zeros in the data.

Data manipulation

All of the attributes were renamed with smaller cases, the response variable Y which is called "default payment next month" was renamed as "class" and converted into factor with two levels. It refers to whether a client has failed to make a payment (defaulted) by the due date which is Oct 2005. If this happens, the credit card financial institution will impose penalty charge to the customer, and if the case has become delinquency, legal action will be taken to enforce payment. The "PAY_x" attributes represent delinquency status and the first attribute is "PAY_0" then follows "PAY_2" which means that "PAY_1" is omitted so we rename it with "pay_1." In addition, for the delinquency status variables in this data set, "-1" signifies a particular customer was current on the payment. However, it can be observed that the delinquency status variables contain values "0" and "-2," which are out of expectations. We re-coded the values "-1" into "1" and all of the remaining sub-groups "-2" and ("1"- 8") were combined into sub-group"2". There are 732 duplicated rows which were removed as well.

```
# Remove redundant ID column and demographic variables
CreditCard <- CreditCard %>% dplyr::select(-ID, -SEX, -EDUCATION, -MARRIAGE, -AGE) %>%
```

```
# Rename the variables to lower case
 rename_with(str_to_lower, everything()) %>%
# Rename "pay_0" to "pay_1" and "default payment next month" to "class"
 dplyr::rename("pay_1" = "pay_0", "class" = "default payment next month")

# Are there any duplicates?
cat("The number of non-duplicate observations within the data set is",
   nrow(unique(CreditCard)), "out of", "\n",
   nrow(CreditCard),
   "indicating that there are",
   nrow(CreditCard) - nrow(unique(CreditCard)),
   "duplicates within the dataset.","\n")
```

```
## The number of non-duplicate observations within the data set is 29268 out of
##  30000 indicating that there are 732 duplicates within the dataset.
```

```
# Remove the duplicated rows
CreditCard<- unique(CreditCard)

# Assign pay_x variables as binary numeric variables
CreditCard$pay_1 <- ifelse(CreditCard$pay_1 == -1,1,2)
CreditCard$pay_2 <- ifelse(CreditCard$pay_2 == -1,1,2)
CreditCard$pay_3 <- ifelse(CreditCard$pay_3 == -1,1,2)
CreditCard$pay_4 <- ifelse(CreditCard$pay_4 == -1,1,2)
CreditCard$pay_5 <- ifelse(CreditCard$pay_5 == -1,1,2)
CreditCard$pay_6 <- ifelse(CreditCard$pay_6 == -1,1,2)

# Assign response variable "class"as a factor with two levels: 0=non_default and 1=default payment
CreditCard$class <- factor(CreditCard$class, levels = c("0","1"),
                           labels = c("credible", "nocredible"))
```

Curent overview of data types and missing values

The df_status() function coming in package funModeling can help us by showing numbers of missing values (NA in R) in relative and percentage values and numbers of unique values for each variable in the dataset. Surprisingly, there is no NA records.

```
# Overview of data types and missing values
CreditCard_status <- df_status(CreditCard, print_results = F) %>% arrange(type)
CreditCard_status
```

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
class	0	0.00	0	0	0	0	factor	2
limit_bal	0	0.00	0	0	0	0	numeric	81
pay_1	0	0.00	0	0	0	0	numeric	2
pay_2	0	0.00	0	0	0	0	numeric	2
pay_3	0	0.00	0	0	0	0	numeric	2
pay_4	0	0.00	0	0	0	0	numeric	2
pay_5	0	0.00	0	0	0	0	numeric	2
pay_6	0	0.00	0	0	0	0	numeric	2
bill_amt1	1356	4.63	0	0	0	0	numeric	22723
bill_amt2	1841	6.29	0	0	0	0	numeric	22346
bill_amt3	2202	7.52	0	0	0	0	numeric	22026
bill_amt4	2527	8.63	0	0	0	0	numeric	21548
bill_amt5	2836	9.69	0	0	0	0	numeric	21010
bill_amt6	3350	11.45	0	0	0	0	numeric	20604
pay_amt1	4546	15.53	0	0	0	0	numeric	7943
pay_amt2	4690	16.02	0	0	0	0	numeric	7899
pay_amt3	5263	17.98	0	0	0	0	numeric	7518
pay_amt4	5701	19.48	0	0	0	0	numeric	6937
pay_amt5	5996	20.49	0	0	0	0	numeric	6897
pay_amt6	6466	22.09	0	0	0	0	numeric	6939

Table 2: Current status of missing values, zeros, infinite values, data types, and unique values in CreditCard dataset.

Colinearity

This is interesting. We can see (Figure.1) that some of the attributes have a strong correlation (e.g. > 0:70 or < 0:70). For example: bill_atm_x attributes with 0.90 and more and pay-x attributes with 0.70. These attributes are candidates for removal because correlated attributes are affecting Linear Kernel of Support Vector Machines. At this stage, we like to consider multicolinearity because we are not sure what type of kernel (linear, radial, polynomial) of SVMs is going to be used in the analysis.

```
# Create a dataframe for correlation plot
dataset_cor <- CreditCard[ ,-20]

# Plot the correlation matrix
correlation_matrix <- cor(dataset_cor)

# Correlation plot
ggcorr(CreditCard[ , -20], label = TRUE, hjust = .85, size = 3,
     color = "grey50", layout.exp = 2)
```

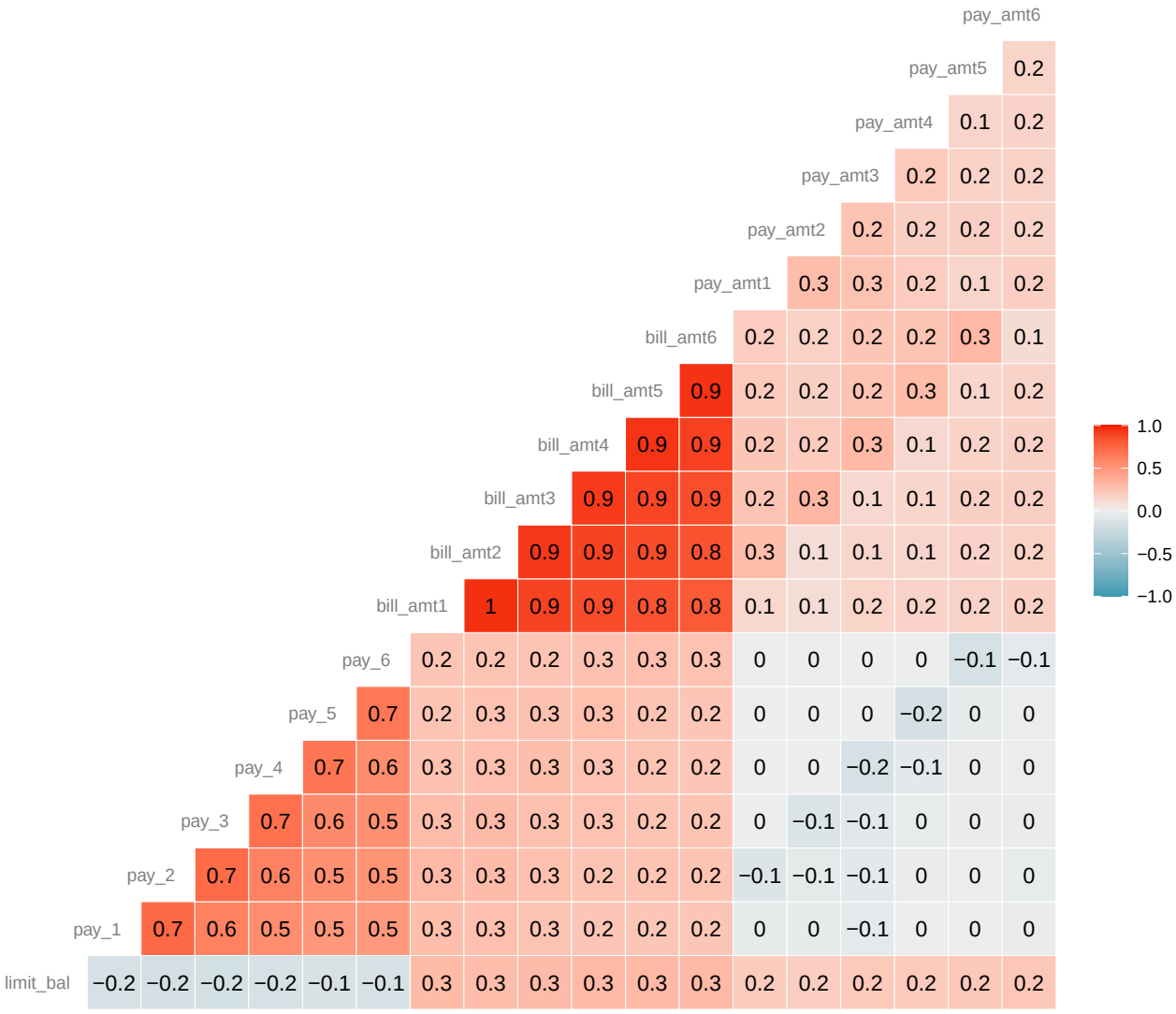

Figure 1: Correlation plot of CreditCard dataset.

```
# Remove correlated attributes
set.seed(7)
cutoff <- 0.70
highly_correlated <- findCorrelation(correlation_matrix, cutoff = cutoff)
for (value in highly_correlated) {
  print(names(dataset_cor)[value])
}
```

```
## [1] "bill_amt4"
## [1] "bill_amt3"
## [1] "bill_amt5"
## [1] "bill_amt2"
## [1] "bill_amt6"
## [1] "pay_2"
```

```
# Create a new dataset without highly correlated attributes
dataset_features <- dataset_cor[,-highly_correlated]
```

Skewness

It is good to calculate the skewness of the numeric attributes even though getting a feeling for the skew is much easier with histograms and density plots. The calculation of the skew is performed with `skew()` function from e1071 package.

```
options (width = 600)
# Calculate skewness for each variable
skew <- apply(dataset_features[,c(1,7:10)], 2, skewness)
skew
```

```
## limit_bal bill_amt1  pay_amt1  pay_amt2  pay_amt3
##  1.010545  2.636970 14.525767 30.132549 17.038192
```

Table 3: Output of skewness. The further the distribution of the skew value from zero, the larger the skew to the left (negative skew value) or right (positive skew value). The distribution is heavily skewed to the right.

There is a lot of structure in this dataset. We need to think about transformations that we could apply such as:

- Feature selection and removing the most correlated attributes that has been taken care of
- Normalization of the dataset to reduce the effect of differing scales
- Standardization of the dataset to reduce the effect of differing distributions
- Box-Cox transform to see if modifying distributions towards normality will improve accuracy

We will perform log transformation to all numeric variables to reduce the effect of differing distributions before splitting the data into training and testing datasets with hope that it will improve the accuracy of the models.

```
# Log transformation of the predictors
dataset_features <- log(dataset_features)
dataset_features[dataset_features == -Inf] <- 0
dataset_features[dataset_features == "NaN"] <- 0

# Attach class attribute and assign as factor
class <- data.frame(class = CreditCard$class)
dataset_features <- cbind(class, dataset_features)
dataset_features <- data.frame(dataset_features)
dataset_features$class <- factor(dataset_features$class, levels = c("credible", "nocredible"))
```

Validation dataset

It is a good idea to use a validation hold out set which is a sample of the data that we hold back from our analysis and modeling. We will use it at the end of our project to confirm the accuracy of our final models. Validation dataset or unseen testing dataset is created through data splitting that involves partitioning the data into an explicit training dataset used to prepare the model and an unseen testing dataset used to evaluate the models performance on unseen data. The applied method of splitting in our analysis is called "simple splitting based on the outcome" that is performed with the `createDataPartition` function from caret package which creates a single 70/30 split of the dataset. The response variable from the dataset is the y argument in the function and if it is a factor as in our case, the random sampling occurs within each class and should preserve the overall class distribution of the data. The argument `list = FALSE` prevents returning data as a list and the argument `times` creates multiple splits at once. Set seed is initiated for reproducibility.

```
# Define an 70%/30% train/test split of the dataset
set.seed(7) # setting seed for reproducibility
validationIndex <- createDataPartition(y = dataset_features$class, times = 1, p=0.70, list=FALSE)
# Select 30% of the data for validation
validation <- dataset_features[-validationIndex,]
# Use the remaining 70% of data to training and testing the models
dataset <- dataset_features[validationIndex,]

# Define train and validation labels as vectors
creditClassTrain <- class[validationIndex]
creditClassTest <- class[-validationIndex]
```

Dimensions of the dataset

We have 20489 instances to work with and can confirm the data has 14 attributes including the class attribute "class."

```
# Dimensions of the dataset
dim(dataset)
```

```
## [1] 20489    14
```

```
# Class distribution
freq(data=dataset, input = c("class"), plot = FALSE)
```

class	frequency	percentage	cumulative_perc
credible	16034	78.26	78.26
nocredible	4455	21.74	100.00

Table 4: Output class breakdown. We can note that "class" is a pretty unbalanced factor but we don't think to rebalance it, not yet. There is indeed 78% to 21% split between credible and non-credible customers of the bank in Taiwan.

Data Visualisations

Let's look at visualizations of individual attributes and start with histograms to get a sense of the data distributions. Most of the attributes have centered distribution or slightly to the left even after applying log transformation.

```
# Create a dataframe for visualisations
df_plot<- dataset[,c(1, 2, 8, 9, 10, 11, 12,13, 14)]

# Plot histograms for each input (explanatory) variable
par(mfrow = c(4,3), oma = c(4,3,4,3), mar = c(1,4,3,2))
for (i in 2:7) {
  graphics::hist(df_plot[,i], main = names(df_plot[i]), xlab = "", col = "grey50")
}
```

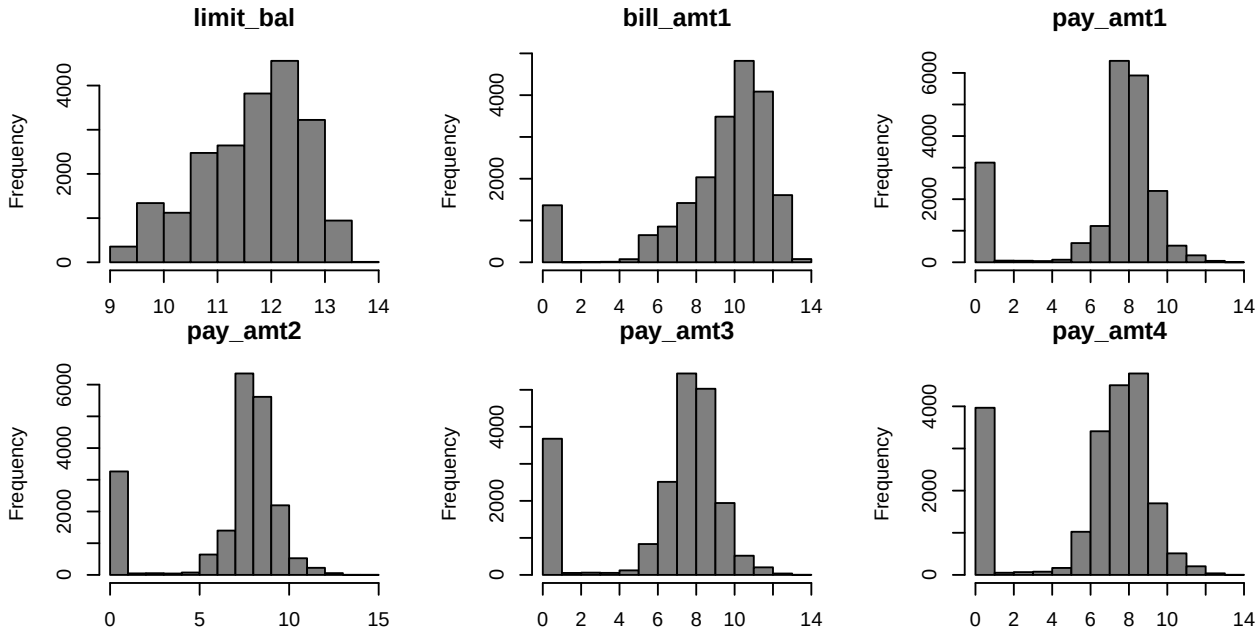

Figure 2: Histograms of selected numeric explanatory attributes of CreditCard dataset.

```r
# Density plots
mycolors = c('red','darkturquoise','blue',"orange", "magenta1","green", "red",
             "darkturquoise", "blue", "orange", "magenta1", "green", "red")
par(mfrow = c(4,3), oma = c(0,1,0,1), mar = c(1,6,3,2))
for (i in 2:13) {
  graphics::plot(density(dataset[,i]), main = names(dataset)[i], col = mycolors[i])
}
```

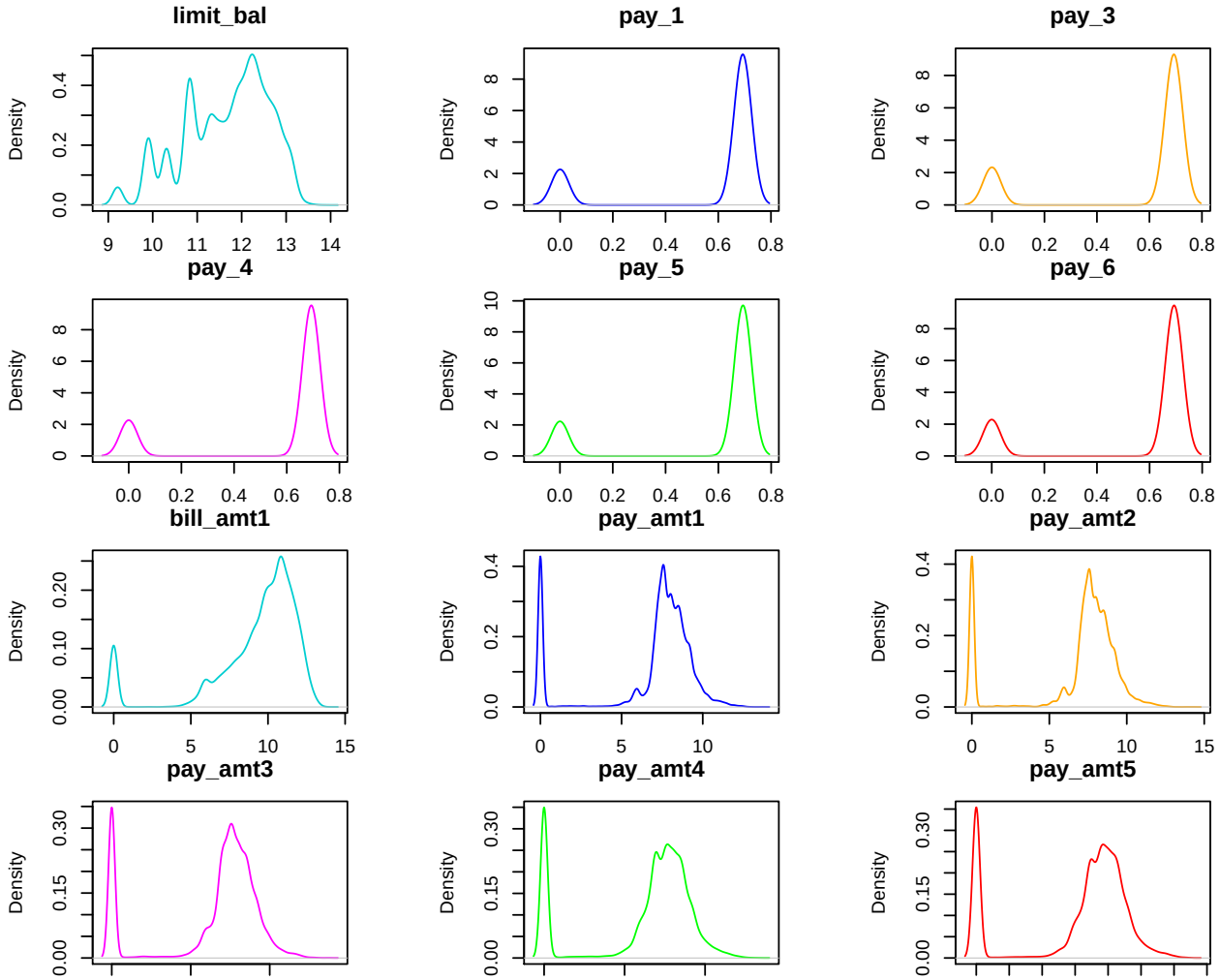

Figure 3: Density plots of selected numeric explanatory attributes of CreditCard dataset.

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 specifed in R. (**10 marks**)

We are going to use the binary response attribute y, called "default payment next month" which is renamed as "class" with two unbalanced classes: "0" and "1" to predict whether or not an individual credit card holder in Taiwan bank defaulted on his credit card payments in October 2005. For each observation, if $y = 1$, then the card holder defaulted on his credit card payments or otherwise.

Let's train another machine learning model with caret using gradient boosting with repeated cross-validation. Keep in mind that in reality, we would want to address the problem of having unbalanced classes but let's focus on hyperparameters tuning now.

When we try to predict the response variable with any machine learning technique, usually there is a difference between predicted and actual values that is caused by noise, variance, and bias. In order to reduce these factors and achieve better results, we use collection of predictors together or mean of predictors called ensemble which in general, are performing better than a single predictor. These ensembling techniques are further classified into Bagging and Boosting.

Bagging involves creating multiple copies of the original training data set using the bootstrap method (James et al. (2013)). It fits a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Each tree is built independently. Boosting is an ensemble technique in which the predictors are not built independently, but sequentially, by putting more weight on instances with wrong predictions and high errors. They can be chosen from a range of models like decision trees, regressors, classifiers, etc. Boosting technique is based on fitting small trees to the residuals instead of using the response variable, and with each small step it improves the performance of the algorithm. In general, statistical learning approaches that learn slowly tend to perform well. The general idea behind this is that instances, which are hard to predict correctly will be focused on during learning, so that the model learns from past mistakes.

Gradient Boosting is an ensemble learner. This means it will create a final model based on a collection of individual models. The predictive power of these individual models is weak and prone to overfitting but combining many such weak models in an ensemble will lead to an overall much improved result.

Gradient boosting is considered a gradient descent algorithm. The general idea of gradient descent is to activate parameters iteratively in order to minimize a loss function. In each round of training, the weak learner is built and its predictions are compared to the correct outcome. The distance between prediction and truth represents the error rate of our model. These errors can now be used to calculate the gradient. The gradient is the partial derivative of the loss function, so it describes the steepness of the error function. An important parameter in gradient descent is the size of the steps which is controlled by the learning rate. If the learning rate is too small, then the algorithm will take many iterations (steps) to find the minimum. On the other hand, if the learning rate is

too high, the algorithm will take bigger steps and will end up further away than the starting point. Therefore, the gradient can be used to find the direction in which to change the model parameters in order to maximally reduce the error in the next round of training by "descending the gradient."

Stochastic gradient descent refers to sampling a fraction of the training observations and growing a tree using that subsample. This makes the algorithm faster but the stochastic nature of random sampling also adds some random nature in descending the loss function's gradient.

In Gradient Boosting we are combining the predictions of multiple models, so we are not optimizing the model parameters directly but the boosted model predictions. Therefore, the gradients will be added to the running training process by fitting the next tree also to these values.

My initial attempt was to use Extreme Gradient Boosting or XGBoost model due to its high level of flexibility and ability to provide high accuracy of unbalanced classification problems. As a result of limitations with my computer power, after many attempts, I managed to perform this task with Stochastic Gradient Boosting Algorithm in caret package even though this algorithm is not the ideal choice for the task.

Gradient descent algorithms are not for this task because we have a dataset of training instances and a response variable with two unbalanced classes, so we cant optimize 0-1 loss (binary classification problem) with gradient descent because any small changes in the weights will have no effect on the loss functions, thus, the predictions will not change and the gradient descent wont go anywhere. If we are on classification boundary, the cost is discontinuous, which is not better, as well. We cant certainly optimize 0-1 loss with gradient descent. Since, this algorithm is a boosting gradient descent, we will give it a try.

Hyperparameters of Stochastic Gradient Boosting Algorithm (GBM)

- **ETA** – Learning rate ("the step size") with which we descend the gradient.
- **Number of trees**, parameter B or $n.trees$: the total number of trees in the sequence or ensemble or the number of Boosting iterations. SGDs often require many trees (it is not uncommon to have many thousands of trees) but since they can easily overfit we must find the optimal number of trees that minimize the loss function of interest with cross validation.
- **Learning rate**, shrinkage, (λ): is a small positive number, that controls the rate of learning with default values 0.01 or 0.001. Very small λ means that B has to be very large in order to achieve good performance. Generally, the smaller this value, the more accurate the model can be but also will require more trees in the sequence. Smaller values also make it easier to stop prior to overfitting. However, they increase the risk of not reaching the optimum with a fixed number of trees and are more computationally demanding.
- **Loss** function, or cost function: takes all the input, sometimes millions of parameters as an input and provides a single value, which tells how much adjustment is needed in our model, it acts as a guide which tells the model that it is performing poorly and it needs some modification regarding its weights and biases.

The two main tree hyperparameters included in the model are:

- **Tree depth** or parameter d written as $interaction.depth$ in caret: controls the depth of

the individual trees. Typical values range from a depth of 3–8 to 1. Smaller depth trees such as decision stumps are computationally efficient (but require more trees). However, higher depth trees allow the algorithm to capture unique interactions but also increase the risk of over-fitting. More generally d can be considered as interaction depth.

- **Minimum number** of observations in terminal nodes or $n.minobsinnode$: with typical values from 5-15 where higher values prevent the model from over-fitting, however, smaller values can help with imbalanced target classes in classification problem. Also, controls the complexity of each tree.

General tuning strategy for GBM algorithms

- Choose a relatively high learning rate between 0.05–0.2
- Determine the optimum number of trees for this learning rate
- Fix tree hyperparameters and tune learning rate and examine speed vs. performance
- Tune tree-specific parameters for decided learning rate
- Once tree-specific parameters have been found, lower the learning rate to check for any improvements in accuracy
- Use final hyperparameter settings and increase CV procedures to get more robust estimates.

Analysis with Stochastic Gradient Boosting Algorithm

When we type getModelInfo("gbm"), we retrieve the specifics of the model from the caret catalog.

Parameters and Hyper-Parameters of this model choice in caret package

For tuning and training our models we use `train()` function from caret package that sets up a grid of tuning parameters for a number of classification and regression problems

Train function arguments applied:

- Method - string specifying which classification or regression model to use. We specify to "gbm" to apply Stochastic Gradient Boosting for Classification.
- TuneGrid - a data frame with possible tuning values. The columns are named the same as the tuning parameters. We specified the values of n.trees, interaction.depth, shrinkage and n.minobsinnode that we want to analyze with `expand.grid()` function.
- Metric - a string that specifies what summary metric will be used to select the optimal model evaluation. We set to "Accuracy" for this classification task.
- TrainControl - a list of values that define how this function acts that are defined with the object we create.

Train Control function arguments applied:

- Method - determines the type of sampling/validation to be undertaken. Repeated cross validation is chosen to estimate the tuning parameters.
- Number - number of folds for cross validation. This is set at 3 for 3-fold cross validation.

- Repeats- the number of repetitions of cross validation to be undertaken. This is set to 5 due to computational time.

Steps of Analysis

1. Step: Build quick base-line model based on default caret tuned interaction depth and the number of trees values. When we examine the model object closely, we can see that caret already did some automatic hyperparameter tuning for us: the train function automatically creates a grid of tuning parameters. By default, if p is the number of tuning parameters, the grid size is 3^p. But we can also specify the number of different values to try for each hyperparameter.

```
# Load the model
gbm_model <- readRDS("./gbm_model.rds")
```

```
# 1. Build quick-baseline model
# Define trainControl object with repeated cross-validation
fitControl <- trainControl(method = "repeatedcv",number = 3, repeats = 5)

# Set seed & train model
set.seed(42)
gbm_model <- train(class ~ ., data = dataset, method = "gbm", trControl = fitControl,
                   verbose = FALSE)
```

2. Step: Perform Cartesian grid search by using `expand.grid` function to manually define single values for every hyperparameter. This function is used to define a grid of hyper-parameters because it creates a grid of all possible combinations of hyperparameters given. We compared different values for the number of trees and tree complexity while shrinkage and the minimum number of observations per node was kept constant.

```
# Load the model
gbm_model_grid1 <- readRDS("./gbm_model_grid1.rds")
```

```
# 2. Tune hyperparameters manually
#  Define the Cartesian grid of hyperparameters
 man_grid <-expand.grid(n.trees = c(100, 200, 250), interaction.depth = c(1, 4, 6),
                shrinkage = 0.1, n.minobsinnode = 10)

# Define trainControl object with repeated cross-validation
fitControl <- trainControl(method = "repeatedcv", number = 3, repeats = 5,
                    search = "grid")

# Set seed & train model
# set.seed(42)
gbm_model_grid1 <- train(class ~ ., data = dataset, method = "gbm", trControl = fitControl,
                   verbose = FALSE, tuneGrid = man_grid)
```

```
# Load the model
gbm_model_big_grid <- readRDS("./gbm_model_big_grid.rds")
```

```
# 3. Define the Cartesian grid with hyperparameter ranges
big_grid <- expand.grid(n.trees = seq(from = 10, to = 300, by = 50),
                        interaction.depth = seq(from = 1, to = 10,
                        length.out = 6), shrinkage = 0.1, n.minobsinnode = 10)

# Define trainControl object
 fitControl <- trainControl(method = "repeatedcv", number = 3, repeats = 5,
                            search = "grid")

# Train the model
set.seed(42)
gbm_model_big_grid <- train(class ~ ., data = dataset,method = "gbm", trControl = fitControl,
                            verbose = FALSE, tuneGrid = big_grid)
```

3.Step: Compare Cartesian grid search versus random search.

```
# Load the model
gbm_model_random <- readRDS("./gbm_model_random.rds")
```

```
# 4. Define a random search in train() control function
fitControl <- trainControl(method = "repeatedcv", number = 3, repeats = 5,
                           search = "random")

# Set tuneLength argument
set.seed(42)
gbm_model_random <- train(class ~ ., data = dataset, method = "gbm",
                          trControl = fitControl,
                          verbose = FALSE,
                          tuneLength = 5)
```

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

In the table of the results, we can see the distribution of both the Accuracy and the Kappa of the models. We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 3 times (3 fold cross validation).

```
options (width = 1000)
# Display summary of results
results <- resamples(list(GBM=gbm_model, GBM_grid=gbm_model_grid1,
                          GBM_big_grid=gbm_model_big_grid,
                          GBM_random=gbm_model_random))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: GBM, GBM_grid, GBM_big_grid, GBM_random
## Number of resamples: 15
##
## Accuracy
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GBM         0.7840410 0.7870113 0.7879941 0.7882669 0.7892965 0.7916240    0
## GBM_grid    0.7843338 0.7885798 0.7890190 0.7890674 0.7905564 0.7923561    0
## GBM_big_grid 0.7863836 0.7878477 0.7890190 0.7895847 0.7911999 0.7948755    0
## GBM_random   0.7709767 0.7738653 0.7754026 0.7749134 0.7760451 0.7783309    0
##
## Kappa
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## GBM         0.1037915 0.1087744 0.1227687 0.1194041 0.1292128 0.1334196    0
## GBM_grid    0.1237981 0.1446850 0.1527885 0.1506131 0.1560341 0.1704404    0
## GBM_big_grid 0.1422490 0.1467800 0.1551155 0.1561658 0.1624585 0.1790408    0
## GBM_random   0.1700427 0.1771362 0.1866583 0.1836083 0.1896538 0.1948838    0
```

Figure 4: Output of estimated Accuracy and Kappa of models on modified dataset

It looks like that GBM_random has the lowest level of Accuracy followed closely by the algorithms that performed Cartesian grid search. The best performing algorithm with Accuracy of 79.4% is GBM_big_grid. Accuracy is the percentage of correctly classified instances out of all instances. It is more useful on binary classification than on multi-class classification. Kappa is the default metric used to evaluate algorithms on binary and multi-class classification datasets in caret. Kappa or Cohen's Kappa is like classifcation accuracy, except that it is normalized at the baseline of random chance on the dataset. It is a very useful measure on problems class imbalance in the classes (e.g. a 70% to 30% split for classes 0 and 1 and you can achieve 70% accuracy by predicting all instances are for class 0). The values for Kappa in our analysis are not very satisfactory, despite high levels of

accuracy.

```
# Plot the results with density plots of accuracy
scales <- list(x=list(relation="free"), y=list(relation="free"))
densityplot(results, scales=scales, pch = "|")
```

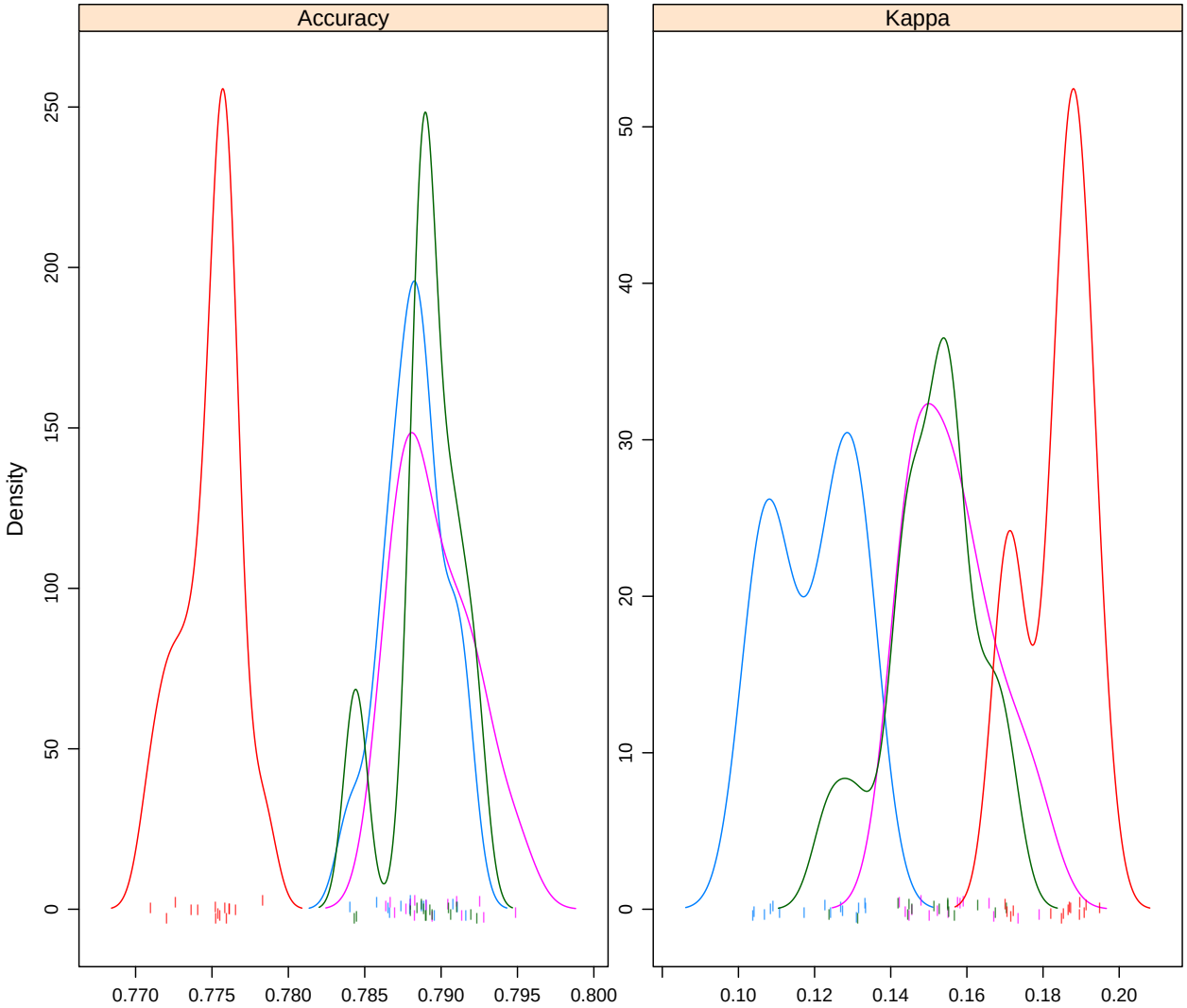

Figure 5: Density plot of estimated Accuracy and Kappa of models on modified dataset

```
# Plot variable importance
gbmImp <- caret::varImp(gbm_model_big_grid, scale = FALSE)
plot(gbmImp, top = 20)
```

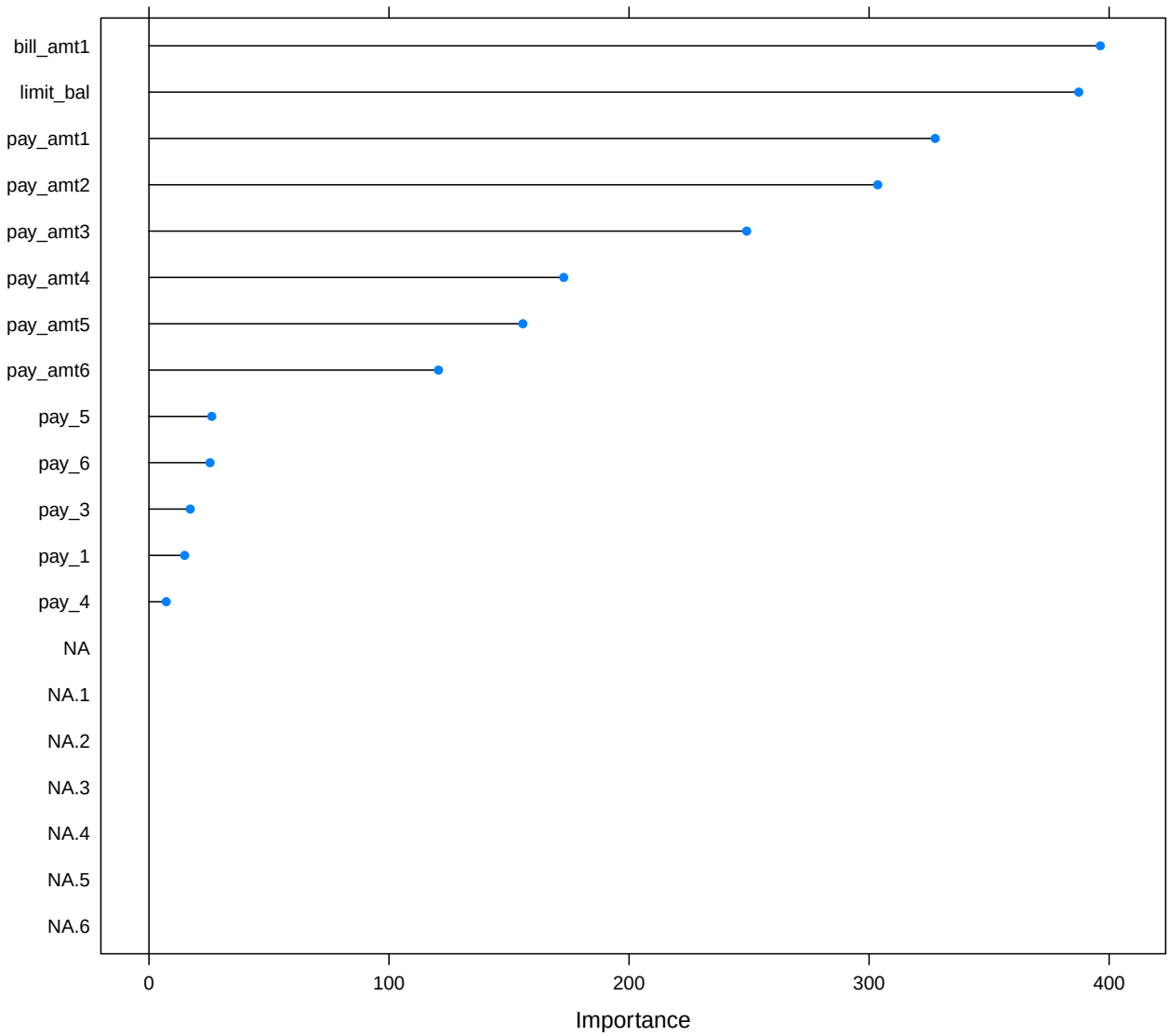

Figure 6: Stochastic Gradient Boosting model feature importance table of CreditCard dataset

Relationships between features and the model results is inspected of the gbm_model_big_grid as the best performing algorithm. Overall, bill_amt1, which shows the amount of bill statements for month of September, 2005 has the greatest impact on classifcations. This is followed by a card holders limit balance used and by a late payment in the most recent month of September, 2005. We can make assumptions that the most recent statement of a card holder payment history would have a greater impact on his likelihood of making upcoming payments compared to his history like, six months ago. Another important observation is that payment amounts of card holders in the last three months had more importance on the model performance than other features.

```
# Plot the model
ggplot(gbm_model_big_grid) + theme_classic() +
  labs(title = "Grid Search with Hyperparameters Ranges")
```

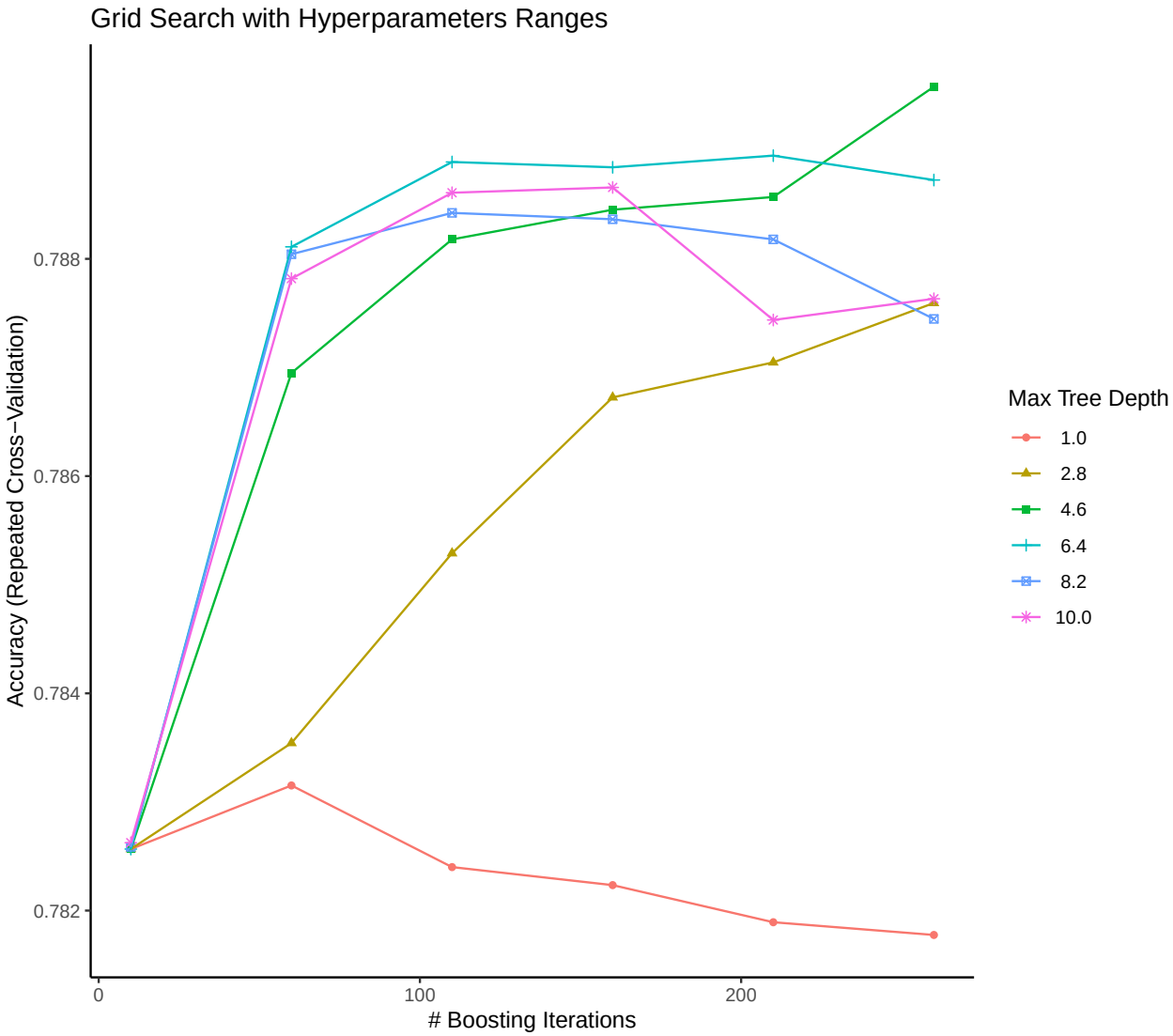

Figure 7: Tuning "gbm" parameters on CreditCard dataset. Every line represents a different hyperparameter for the maximum tree depth. The colors of the lines correspond to this as well. On the x-axis, we see the number of boosting iterations, which comes from the hyperparameter n.trees that we defined to be either 100, 200 or 250. And the y-axis shows the accuracy of the model given these hyperparameter combinations.

```
# Heatmap the KAPPA parameter
plot(gbm_model_big_grid, metric = "Kappa", plotType = "level")
```

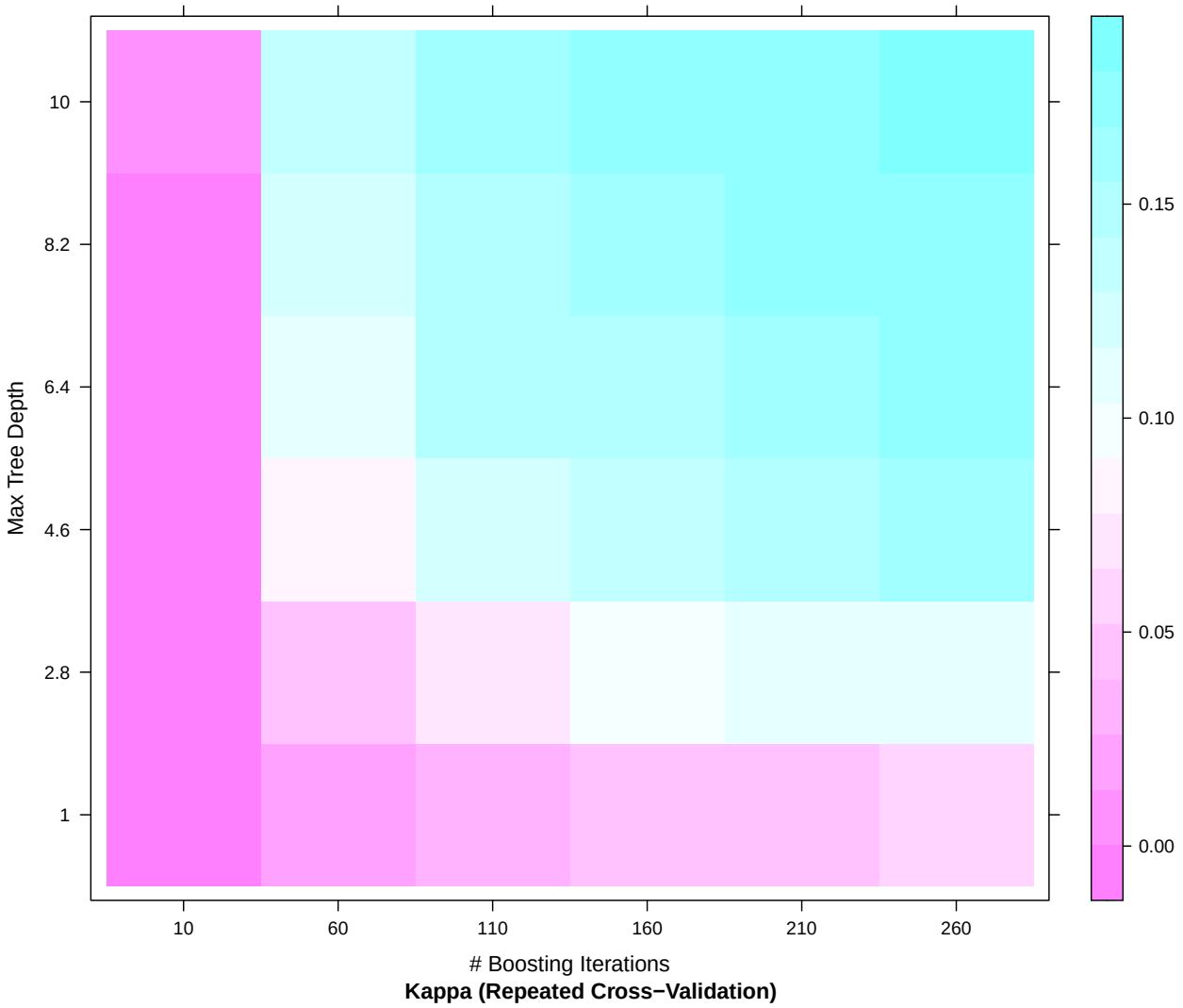

Figure 8: Heatmap showing Kappa values of Stochastic Gradient Boosting model for CreditCard dataset. Kappa is another metric used to evaluate the performance of classification models. It compares an Observed Accuracy with an Expected Accuracy. Kappa values need to be considered in the context of the problem but generally, we want to achieve high Kappa values. The Kappa values are shown on the color scale, while the x-axis shows the number of trees and the y-axis the max tree depth. Here, our Kappa values don't look very good - the reason is that our data was strongly unbalanced, so the accuracy for always assigning the majority class will already be very high. So, we can conclude that while having pretty good accuracy, our model did not in fact perform much better than random.

(c) Evaluate the performance of the algorithm on the training data and comment on the results. (**5 marks**)

Objects produced by the train function contain the "optimized" model in the "finalModel," therefore predictions can be made from these objects as usual. We can make predictions using the models

that we have tuned using caret with the predict.train() function. We can see that caret provides access to the best model from a training run in the finalModel variable. We can use that model to make predictions by calling predict using the best saved model from train() which will automatically use the final model.The argument newdata is used to specify the data one which to make predictions. I applied a function for creation of confusion matrix and evaluation of performance metrics as shown below.

```
# Measures of predicted classes for GBM (gbm_model_big_grid) model on training data:
dataset$class <- factor(dataset$class, levels = c("credible","nocredible"))
levels(dataset)<- c("credible","nocredible")

predmodel_train = predict(gbm_model_big_grid, newdata = dataset)

gbm_cm_train <- confusionMatrix(predmodel_train, dataset$class)
# Function for metrics performance of the models
metrics_classification = function(predicted, observed){
 (confusion_table = table(predicted, observed)) # create the confusion matrix
 TP = confusion_table[1,1]
 TN = confusion_table[2,2]
 FN = confusion_table[2,1]
 FP = confusion_table[1,2]
 accuracy = round((TP + TN) / sum(TP,FP,TN,FN), 4)
 error_rate = round((FP + FN) / sum(TP,FP,TN,FN),4)
 precision = round(TP / sum(TP, FP), 4)
 recall = round(TP / sum(TP, FN), 4)
 sensitivity = round(TP / sum(TP, FN), 4)
 specificity = round(TN / sum(TN, FP), 4)
 f1_score = round((2 * precision * sensitivity) / (precision + sensitivity), 4)
 metrics = c(accuracy, error_rate, precision, recall, sensitivity, specificity, f1_score)
 names(metrics) = c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
 return(metrics)
}

metrics_gbm_train <- metrics_classification(predmodel_train, creditClassTrain)

# Display the results of GBM for metrics performance on training dataset
metrics_gbm_train <- data.frame(metrics_gbm_train)
col_names <- c("Parameters", "GBM")
parameters <- c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
metrics_gbm_train <- cbind(parameters, metrics_gbm_train)
colnames(metrics_gbm_train) <- col_names
row.names(metrics_gbm_train) <- NULL
knitr::kable(metrics_gbm_train, align = 'c', format = "markdown")
```

Parameters	GBM
Accuracy	0.8001
Error_Rate	0.1999
Precision	0.8077
Recall	0.9772
Sensitivity	0.9772
Specificity	0.1627
F1 score	0.8844

Table 3: Output of estimated metrics performances of Stochastic Gradient Boosting model on training dataset. We can see that the accuracy of the final model on the training dataset is 80.00%. This is even higher than the accuracy of the model itself and it is not really optimistic because there are around 20 489 rows. There is a great chance that the model is overfitting the data and it is miss-classifying a lot of instances on behalf of the the majority class, the “credible” card holders which seems that it does not have a large impact on performance metrics.

Classification with unbalanced binary response variable plays an important part and problem with unbalanced response variable must be addressed apriori building the model. One method that may have improve the model, is incorporating Principal Component Analysis (PCA) to reduce the amount of dimensionality in the credit card datasets. This reduced dataset would have resulted in lower computer processing times and even maybe will increase its accuracy. On the other hand, I am very optimistic because, the miss-classification error of the model is directly related to the loss function. Finding the appropriate loss function for this classification problem might be the solution which requires in-depth analysis.

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.

We will start with the fact that this is very complex dataset with overlapping distributions of variables, almost linearly not separable. In this case no separating hyperplane exists, there is no maximal margin classifier and the optimization problem (equations 19 and 20) has no solution with $M > 0$. Therefore, we cannot exactly separate the two classes. However, we can develop a hyperplane that almost separates the classes, using a so-called soft margin. The classifier that uses the generalization of the maximal margin to the non-separable case is known as the support vector classifier (SVM).

The SVM is a classification algorithm used in classification and regression analysis that atempts to separate the classes with a separating hyperplane in an $n-1$ dimensional space where n is the number of features. It is a powerful classification algorithm because it has a particular regularisation parameter allowing flexibility in the separation of classes and it is less sensitive to outliers and overfitting. The generalization of the maximal margin to the non-separable case in SVM is achieved by implementation on non-linear kernels that are creating the separating hyperplane and as a result the algorithm is very popular with analysis of complex datasets in real life.

$$\max_{\beta_0,\beta_1,...\beta_p} M \tag{19}$$

$$subject\ to \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geqslant M \qquad \forall_i \tag{20}$$

The algorithm of our choice for this analysis is the **Support Vector Machine with Radial Basis Function Kernel** for classification in caret package. When we type getModelInfo("svmRadial"), we retrieve the specifics of the model from the caret catalog.

Parameters and Hyper-Parameters of this model choice in caret package

Train function arguments applied:

- Method - string specifying which classification or regression model to use. We specify to "svmRadial" to apply the radial kernel because of the complexity of the dataset and computational time for analysis. Hyper-parameter of "svmRadial" is sigma called Sigma, type is numeric. Another parameter is C called Cost, type is also numeric. Cost is a regularization parameter that represents the number of mis-classifications that are allowed and thus makes the model less sensitive to outliers and reduces the risk of over fitting the data. A range of

values that we applied are 0.25, 05, to 1. Sigma is the smoother parameter of the kernel and we kept it at 1.

- TuneGrid - a data frame with possible tuning values. The columns are named the same as the tuning parameters. We specified the values Sigma and Cost that we want to analyze with expand.grid() function.
- Metric - a string that specifies what summary metric that will be used to select the optimal model evaluation. We set to "Accuracy" for this classification task.
- TrainControl - a list of values that define how this function acts that are defined with the object we create.

Train Control function arguments applied:

- Method - determines the type of sampling/validation to be undertaken. Repeated cross validation is chosen to estimate the tuning parameters.
- Number - number of folds for cross validation. This is set at 3 for 3-fold cross validation.
- Repeats- the number of repetitions of cross validation to be undertaken. This is set to 5 due to computational time.

Optimization steps

1. Define the repeated cross-validation scheme for caret with 5 folds and 3 repeats with train-Control() function that will create the trainControl object.

2 Define the Cartesian grid with expand.grid() function in caret with the following combinations of hyper-parameters: sigma = 1 and C= (0.25, 0.5, 1). The radial kernel requires setting the smoother parameter sigma and we define sigma = 1. The parameter C is the "cost" of the radial kernel. This parameter controls the complexity of the boundary between support vectors. We will explore a little sensitivity around this parameter and define three values for C in our analysis in order to obtain best values for Accuracy. Note that all of the combinations of sigma and C that we define in expand.grid() function are going to be saved as data frame object.

My initial ideal was to analyze a bigger sequence defined as C = c(0.25, 0.50, 0.75, 0.9, 1, 1.1, 1.25) and sigma = c(.01, .015, 0.2) but due to limitations of the power of my computer and computational time for analysis I could not afford it. Also, if C is fixed, smaller values of sigma yield more complex boundaries that is also aspect for analysis.

3. Train the model with train function in caret with abbreviation of "svmRadial" for the method argument of the function to specify the radial kernel and the "Accuracy" abbreviation for the metric argument of the function in order to assess performance of the algorithm.

```
# Load the file
fit.svm <- readRDS("./fit.svm.rds")
```

```
set.seed(123)
fitControl <- trainControl(method = "repeatedcv",
                           number = 3, # 3fold cross validation
                           repeats = 5)
```

```
metric <- "Accuracy"

man_grid <- expand.grid(sigma = 1, C=c(0.25,0.5,1))

fit.svm <- train(class ~ .,
                 data = dataset,
                 method = "svmRadial",
                 trControl = fitControl,
                 metric = metric,
                 tuneGrid = man_grid)
```

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

```
# Print model summary of the final model
fit.svm$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 0.5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1
##
## Number of Support Vectors : 13264
##
## Objective Function Value : -4092.44
## Training error : 0.197569
```

Table 1: Output of "svmRadial" summary of hyperparameter metrics for the final model.

```
# Print model parameters values
fit.svm$results
```

sigma	C	Accuracy	Kappa	AccuracySD	KappaSD
1	0.25	0.7851530	0.0611177	0.0012683	0.0078886
1	0.50	0.7855435	0.1055251	0.0019920	0.0092992
1	1.00	0.7843331	0.1380123	0.0027198	0.0104043

Table 2: Output of estimated Accuracy and Kappa of the model on modified dataset.

```
# Plot Accuracy of the model
plot(fit.svm)
```

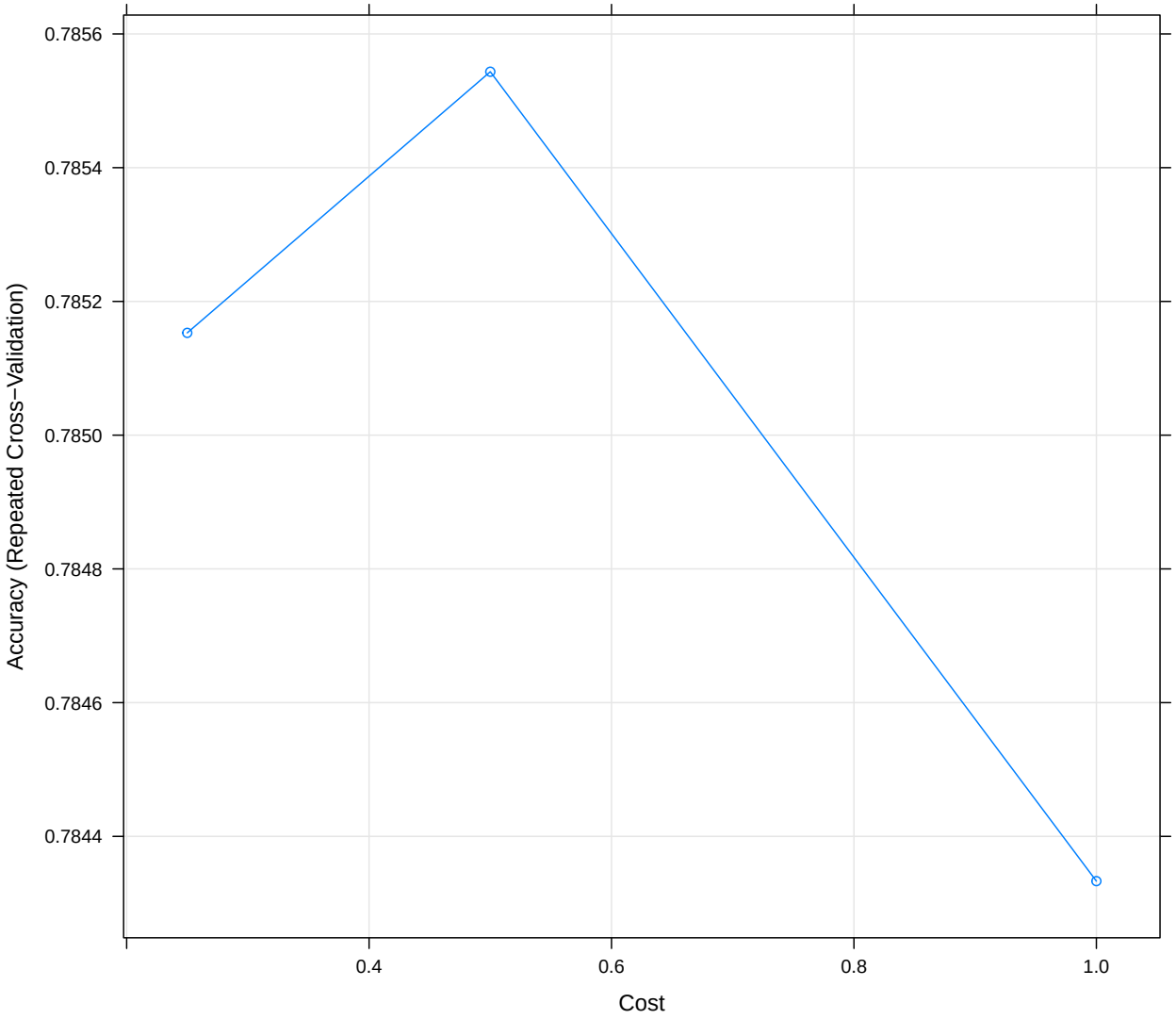

Figure 1: Algorithm tuning result for SVM on the CreditCard dataset. The results show that the best model resulted from setting of sigma = 1, C = 0.5, Number of Support Vectors : 13264 and Training error : 0.197569.

```
# Variable Importance
svm_vi <- varImp(fit.svm)
import <- as.data.frame(svm_vi$importance)
import$names <- row.names(import)

# Variable importance plot
library(ggplot2)
ggplot(import, aes(x = reorder(names, credible), y = nocredible))+
  geom_bar(stat = "identity", fill = "sienna3", color = "grey50") +
  theme_classic()+
  theme(text = element_text(size = 14, face = 'bold'),
        axis.text.x = element_text(vjust = 2, face = 'bold')) +
  labs(title = "Variable Importance for SVM with Radial Basis Function Kernel")+
```

```
ylab( "Variable Importance")+
xlab("Variable") +
coord_flip()
```

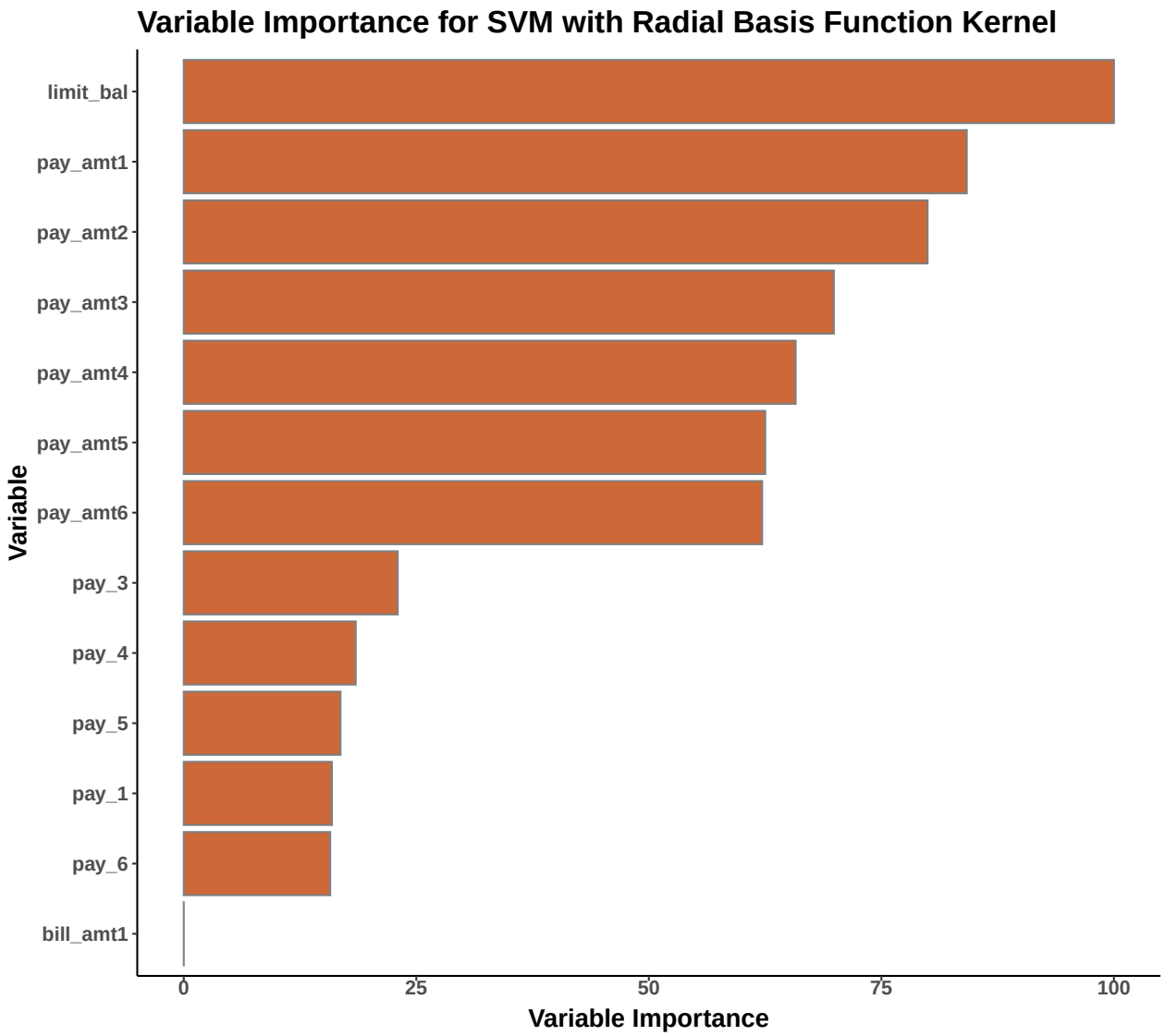

Figure 2: Variable importance or feature importance of the model displayed that lim_bal is the most important variable with 100% in separating the data using the classification tree. However, this variable has second position at variable importance graph of Stochastic Gradient Boosting algorithm. In general, there is a same pattern in classification of the predictor variables in both graphs: pay_amt_x variables in a group are following immediately after the most important variable. Group of pay_x variable is position after the group of pay_amt_x variables on both charts.

Pre-processing of the variables including log-transformation and feature selection was performed before data partitioning of the dataset into training and testing instances. In order to compare algorithm performances of SVM with radial basis function kernel which is non-linear and Stochastic Gradient Boosting which is boosting algorithm, we need to supply them with training instances pre-processed under the same conditions. Perhaps, this is the reason for obtaining similar results.

(c) Evaluate the performance of the algorithm on the training data and comment on the results. (**5 marks**)

```
# Measures of predicted classes for SVM model on training data

predictions_svm_train <- predict(fit.svm, newdata=dataset)
svm_cm_train <- confusionMatrix(predictions_svm_train, dataset$class)


metrics_svm_train <- metrics_classification(predictions_svm_train, creditClassTrain)

# Display the results of SVM of metrics performance on training dataset
metrics_svm_train <- data.frame(metrics_svm_train)
col_names <- c("Parameters", "SVM")
parameters <- c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
metrics_svm_train <- cbind(parameters, metrics_svm_train)
colnames(metrics_svm_train) <- col_names
row.names(metrics_svm_train) <- NULL
knitr::kable(metrics_svm_train, align = 'c', format = "markdown")
```

Parameters	SVM
Accuracy	0.8024
Error_Rate	0.1976
Precision	0.8061
Recall	0.9842
Sensitivity	0.9842
Specificity	0.1481
F1 score	0.8863

Table 3: Output of estimated metrics performances of SVN model on training dataset. Returning to the training model evaluation metrics, the SVM model summary revealed that the algorithm had a relatively high training error rate of 0.20 and accuracy of 78.5%. However, when looking at the evaluation metrics from the cross fold validation, including the KAPPA values, sensitivity and specificity, it is clear that these model evaluation metrics indicate that this support vector machine model also had a difficult time separating the two classes, with a high rate of correct classification of the majority class but a low rate of accuracy classifying the minority class. The results also showed that the algorithm is highly sensitive to distribution of the response variable with values greater than 0.98. The performance of this model can be improved by reducing the dimensionality of the dataset through PCA (Principal Component Analysis) or by analyzing only selected variables corresponding to the first three months as it is a usual way of analysis in the bank instiutions. Another important fact is that when we are working with unbalanced response

variable, it is necessary to perform rebalancing the classes before algorithm evaluation which can be done with several methods. One of the way is to perform “up” rebalancing with caret sampling parameter which would increase the smaller class, “non-credible” instances that might increase overalll algorithm performance.

Furthermore, training of the model was additionally restricted in cross validation and in tuning the parameters due to computational time and lack of computer power. In order to obtain a full picture of this algorithm performance, I need to perform analysis under different conditions:

- a) to analyse results with selected variables with feature selection and without feature selection in order to see the effect of correlated variables on algorithm performance;
- b) to analyze variables with “BoxCox” and without “BoxCox” to inspect the effect of standardization of distributions on algorithm performance;
- c) to manipulate with hyper-parameters values and with grid search options which requires a lot of time and a lot of understanding the problem from statistical point of view;

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.

We can make new predictions using the models that we have tuned using caret with the predict.train() function. We can see that caret provides access to the best model from a training run in the finalModel variable. We can use that model to make predictions by calling predict using the best saved model from train() which will automatically use the final model. The argument newdata is specifying the data one which to make predictions. I applied a function for creation of confusion matrix and evaluation of performance metrics as shown below. The values were compared with the results obtained with confusionMatrix() function from caret for checking the accuracy of the function.

```
# Prepare the validation dataset
validation$class <- factor(validation$class, levels = c("credible","nocredible"))
levels(validation)<- c("credible","nocredible")

# Display the results of GBM for metrics performance on validation datasets
set.seed(42)
predictions_gbm_test <- predict(gbm_model_big_grid, newdata=validation)
gbm_cm_test <- confusionMatrix(predictions_gbm_test, validation$class)
metrics_gbm_test <- metrics_classification(predictions_gbm_test, creditClassTest)

# Display the results GBM of metrics performance on validation dataset
metrics_gbm_test <- data.frame(metrics_gbm_test)
col_names <- c("Parameters", "GBM")
parameters <- c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
metrics_gbm_test <- cbind(parameters, metrics_gbm_test)
colnames(metrics_gbm_test) <- col_names
row.names(metrics_gbm_test) <- NULL
knitr::kable(metrics_gbm_test, align = 'c', format = "markdown")
```

Stochastic Gradient Boost Algorithm Predictions

Parameters	GBM
Accuracy	0.7895
Error_Rate	0.2105
Precision	0.8009
Recall	0.9729
Sensitivity	0.9729
Specificity	0.1289
F1 score	0.8786

Table 3: Estimated performance metrics of predictions of GBM on testing dataset

```
# Measures of predicted classes for SVM model on testing data
# Testing data
set.seed(123)
predictions_svm_test <- predict(fit.svm, newdata=validation)
svm_cm_test<-confusionMatrix(predictions_svm_test, validation$class)

metrics_svm_test <- metrics_classification(predictions_svm_test, creditClassTest)


# Display the results SVM of metrics performance on validation dataset
metrics_svm_test <- data.frame(metrics_svm_test)
col_names <- c("Parameters", "SVM")
parameters <- c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
metrics_svm_test <- cbind(parameters, metrics_svm_test)
colnames(metrics_svm_test) <- col_names
row.names(metrics_svm_test) <- NULL
knitr::kable(metrics_svm_test, align = 'c', format = "markdown")
```

Parameters	SVM
Accuracy	0.7872
Error_Rate	0.2128
Precision	0.7978
Recall	0.9753
Sensitivity	0.9753
Specificity	0.1101
F1 score	0.8777

Table 4: Estimated performance metrics of SVM on testing dataset

```
# Table for performance metrics on train data of 2 classifiers:
#============================================================
library(pander)
col_names <- c("Parameters", "GBM", "SVM")
parameters <- c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
metrics_df_train <- merge(metrics_gbm_train , metrics_svm_train)
colnames(metrics_df_train) <- col_names
row.names(metrics_df_train) <- NULL
knitr::kable(metrics_df_train, align = 'c', format = "markdown")
```

Parameters	GBM	SVM
Accuracy	0.8001	0.8024
Error_Rate	0.1999	0.1976

Parameters	GBM	SVM
F1 score	0.8844	0.8863
Precision	0.8077	0.8061
Recall	0.9772	0.9842
Sensitivity	0.9772	0.9842
Specificity	0.1627	0.1481

Table 5: Prediction results on training data

```
# Table for performance metrics on train data of 2 classifiers:
#============================================================
col_names <- c("Parameters", "GBM", "SVM")
parameters <- c("Accuracy", "Error_Rate", "Precision", "Recall", "Sensitivity", "Specificity", "F1 score")
metrics_df_test <- merge(metrics_gbm_test , metrics_svm_test )
colnames(metrics_df_test) <- col_names
row.names(metrics_df_test) <- NULL
knitr::kable(metrics_df_test, align = 'c', format = "markdown")
```

Parameters	GBM	SVM
Accuracy	0.7895	0.7872
Error_Rate	0.2105	0.2128
F1 score	0.8786	0.8777
Precision	0.8009	0.7978
Recall	0.9729	0.9753
Sensitivity	0.9729	0.9753
Specificity	0.1289	0.1101

Table 2: Prediction results on testing data.

By observing the results, we can see good accuracy across the board. All algorithms have a mean accuracy above 70%, well above the baseline of 22% if we just predicted “non-credible.” The problem is learnable. There is a really very small difference in accuracy outputs including the results of all other metrics analyzed between training and testing datasets with both algorithms. GBM outperformed SVM for only 0.02% on testing dataset. The results achieved by tuned GBM (on training dataset of 80% and on testing dataset of 78.9%) are almost identical with results achieved by tuned SVM (80% on training dataset and 78.7% on testing dataset).

My preferred algorithm is SVM and I would like to tune it further if I can lift the accuracy. The whole idea of creating an accurate model for our dataset was to make predictions on unseen data. There are other tasks that I would like to be concerned, such as creating a standalone model using all training data.

However, SVM performed really good, but GBM is a boosting algorithm and it can perform even better. The main idea of GBM is that it only uses a batch of samples drawn uniformly from training

set in each step. If we have imbalanced data, this introduces the risk of fail to learn the information from minority class. In our dataset the minority class only takes 22% of total samples. If we set 1000 as batch size, we can only learn around 2 samples from minority class in each iteration and maybe there will be no minority class to learn. In extreme case we can treat testing sample as majority class, and rebalance the sample by oversampling the minority class and explore if the training error and testing error are decreasing. There is a lot of motivation to use GBM because there is also redundant information in the imbalanced data, so it is not necessary to learn them at each step. On the other side, GBM is minimizing the average training error and that is equivalent to minimizing the expected loss from empirical distribution. When the training data is large enough,the empirical distribution is a good approximation of the true distribution. Perhaps, our dataset is not large enough.

However, learning with imbalanced data is still a challenging problem for supervised learning. It can take time to find well performing machine learning algorithms for our dataset because of the trial and error nature of applied machine learning. Once we have a shortlist of accurate models, we can apply tuning of the algorithms to get the most from each algorithm. We can also combine predictions of different models together in order to increase the accuracy of the models. There are three popular methods for combining predictions from different models into ensemble prediction: bagging, boosting and stacking. In the future analysis, I will attempt to combine: C5.0 and Stochastic Gradient Boosting for boosting algorithms, and Classification and Regression Trees from bagging group of algorithms and SVM with Radial Basis Kernel Function from stacking algorithms on the same dataset to explore the results and find the best model.

References

Goldfarb, Donald, and Ashok Idnani. 1982. "Dual and Primal-Dual Methods for Solving Strictly Convex Quadratic Programs." In *Numerical Analysis*, 226–39. Springer.

———. 1983. "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs." *Mathematical Programming* 27 (1): 1–33.

Hornik, Maintainer Kurt. 2009. "Package 'Quadprog'."

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Sra, Suvrit, Sebastian Nowozin, and Stephen J Wright. 2012. "Optimization for Machine Learning. Neural Information Processing Series." Mit Press Cambridge.

Vapnik, V, and A Ya Chervonenkis. 1964. "A Class of Algorithms for Pattern Recognition Learning." *Avtomat. I Telemekh* 25 (6): 937–45.

Vapnik, V, and A Ya Lerner. 1963. "Recognition of Patterns with Help of Generalized Portraits." *Avtomat. I Telemekh* 24 (6): 774–80.

Yeh, I-Cheng, and Che-hui Lien. 2009. "The Comparisons of Data Mining Techniques for the Predictive Accuracy of Probability of Default of Credit Card Clients." *Expert Systems with*

Applications 36 (2): 2473–80.