Statistical Learning Final Project

Siddharth, Nikhil, Sai Venkat

2024-03-16

Table of Contents

Data Requirements:

- You can pick any data you want as long as it is a **classification** problem and has **at least 500 observations**.

  The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

  There are two datasets:

  1) bank-additional-full.csv with all examples (41188) and 20 inputs, ordered by date (from May 2008 to November 2010), very close to the data analyzed in $Moro et al.$,2014
  2) bank-additional.csv with 10% of the examples (4119), randomly selected from 1), and 20 inputs.

  We have chosen to work on a sample of our data which comprises of approximately 4200 observations. as it was computationally expensive to work on models like SVM testing because the full dataset had approximately 42000 observations

There are 9 numeric columns and 11 categorical columns excluding the target variable y.

bank client data has personal details and it has details whether they took any loans and have they defaulted on any loan.

bank client data:

1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5 - default: has credit in default? (categorical: 'no','yes','unknown')
6 - housing: has housing loan? (categorical: 'no','yes','unknown')
7 - loan: has personal loan? (categorical: 'no','yes','unknown')

last contact of current campaign has details when and how the bank has contacted the clients as part of the marketing campaign.

related with the last contact of the current campaign:

8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

other attributes talk about history of the past contacts to the clients.

other attributes:

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

social and economic attributes talk about banks personal details on quarterly level. and we have our binary target variable y which says whether the client has subscribed to the term deposit or not.

social and economic context attributes

16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):
21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

Relevant Information about dataset:

This dataset is based on "Bank Marketing" UCI dataset (please check the description at: http://archive.ics.uci.edu/ml/datasets/Bank+Marketing). The data is enriched by the addition of five new social and economic features/attributes (national wide indicators from a ~10M population country), published by the Banco de Portugal and publicly available at: https://www.bportugal.pt/estatisticasweb. This dataset is almost identical to the one used in

$$Moro et al., 2014$$

(it does not include all attributes due to privacy concerns). Using the rminer package and R tool (http://cran.r-project.org/web/packages/rminer/), we found that the addition of the five new social and economic attributes (made available here) lead to substantial improvement in the prediction of a success, even when the duration of the call is not included. Note: the file can be read in R using: d=read.table("bank-additional-full.csv",header=TRUE,sep=";")

Read your data in R and call it df. For the rest of this document y refers to the variable you are predicting

```
df = df = read.csv("C:/Users/siddh/OneDrive/Desktop/Term 3/Stat Learning/Fina
l Project/bank-additional.csv", header = TRUE, sep = ";")

head(df)

##   age         job marital         education default housing    loan   cont
act
## 1  30 blue-collar married          basic.9y      no     yes      no  cellu
lar
## 2  39    services  single       high.school      no      no      no teleph
```

```
one
## 3  25    services married       high.school      no     yes      no teleph
one
## 4  38    services married          basic.9y      no unknown unknown teleph
one
## 5  47      admin. married university.degree      no     yes      no  cellu
lar
## 6  32    services  single university.degree      no      no      no  cellu
lar
##   month day_of_week duration campaign pdays previous    poutcome emp.var.r
ate
## 1   may         fri      487        2   999        0 nonexistent         -
1.8
## 2   may         fri      346        4   999        0 nonexistent
1.1
## 3   jun         wed      227        1   999        0 nonexistent
1.4
## 4   jun         fri       17        3   999        0 nonexistent
1.4
## 5   nov         mon       58        1   999        0 nonexistent         -
0.1
## 6   sep         thu      128        3   999        2     failure         -
1.1
##   cons.price.idx cons.conf.idx euribor3m nr.employed  y
## 1         92.893         -46.2     1.313      5099.1 no
## 2         93.994         -36.4     4.855      5191.0 no
## 3         94.465         -41.8     4.962      5228.1 no
## 4         94.465         -41.8     4.959      5228.1 no
## 5         93.200         -42.0     4.191      5195.8 no
## 6         94.199         -37.5     0.884      4963.6 no
```

```
summary(df)
```

```
##       age            job              marital           education
##  Min.   :18.00   Length:4119        Length:4119        Length:4119
##  1st Qu.:32.00   Class :character   Class :character   Class :character
##  Median :38.00   Mode  :character   Mode  :character   Mode  :character
##  Mean   :40.11
##  3rd Qu.:47.00
##  Max.   :88.00
##    default            housing              loan             contact
##  Length:4119        Length:4119        Length:4119        Length:4119
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##     month           day_of_week           duration         campaign
##  Length:4119        Length:4119        Min.   :   0.0   Min.   : 1.000
##  Class :character   Class :character   1st Qu.: 103.0   1st Qu.: 1.000
```

```
##  Mode  :character   Mode  :character   Median : 181.0   Median : 2.000
##                                        Mean   : 256.8   Mean   : 2.537
##                                        3rd Qu.: 317.0   3rd Qu.: 3.000
##                                        Max.   :3643.0   Max.   :35.000
##      pdays          previous        poutcome          emp.var.rate
##  Min.   :  0.0   Min.   :0.0000   Length:4119        Min.   :-3.40000
##  1st Qu.:999.0   1st Qu.:0.0000   Class :character   1st Qu.:-1.80000
##  Median :999.0   Median :0.0000   Mode  :character   Median : 1.10000
##  Mean   :960.4   Mean   :0.1903                      Mean   : 0.08497
##  3rd Qu.:999.0   3rd Qu.:0.0000                      3rd Qu.: 1.40000
##  Max.   :999.0   Max.   :6.0000                      Max.   : 1.40000
##  cons.price.idx  cons.conf.idx     euribor3m      nr.employed
##  Min.   :92.20   Min.   :-50.8   Min.   :0.635   Min.   :4964
##  1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.334   1st Qu.:5099
##  Median :93.75   Median :-41.8   Median :4.857   Median :5191
##  Mean   :93.58   Mean   :-40.5   Mean   :3.621   Mean   :5166
##  3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228
##  Max.   :94.77   Max.   :-26.9   Max.   :5.045   Max.   :5228
##       y
##  Length:4119
##  Class :character
##  Mode  :character
##
##
##
```

The grading rubric can be found below:

	R code	Decision/Why	Communication of findings
Percentage of Assigned Points	30%	35%	35%

- **Decision/why?**: Explain your reasoning behind your choice of the procedure, set of variables and such for the question.
  - Explain why you use the procedure/model/variable
  - To exceed this criterion, describe steps taken to implement the procedure in a non technical way.
- **Communication of your findings**: Explain your results in terms of training, testing, and prediction of the variable Y
  - Explain why you think one model is better than the other.
  - To exceed this criterion, explain your model and how it predicts y in a non technical way.

Part 1: Exploratory Data Analysis (20 points)

1. Check for existence of NA's (missing data), if necessary, impute the missing data.

```r
any(is.na(df))
```

```
## [1] FALSE
```

There arent any missing in the dataset as shown by the output above. The source file provided was easily interpretable for performing EDA and applying regression techniques.

2. If necessary, classify all categorical variables except the one you are predicting as factors. Calculate the summary statistics of the entire data set.

```r
categorical_columns <- sapply(df, is.character)
categorical_columns["y"] <- FALSE
df[categorical_columns] <- lapply(df[categorical_columns], as.factor)

summary(df)
```

```
##       age                 job           marital                    educatio
n
##  Min.   :18.00   admin.     :1012   divorced: 446   university.degree  :12
64
##  1st Qu.:32.00   blue-collar: 884   married :2509   high.school        : 9
21
##  Median :38.00   technician : 691   single  :1153   basic.9y           : 5
74
##  Mean   :40.11   services   : 393   unknown :  11   professional.course: 5
35
##  3rd Qu.:47.00   management : 324                   basic.4y           : 4
29
##  Max.   :88.00   retired    : 166                   basic.6y           : 2
28
##                  (Other)    : 649                   (Other)            : 1
68
##     default        housing          loan           contact         month
##  no     :3315   no     :1839   no     :3349   cellular :2652   may    :137
8
##  unknown: 803   unknown: 105   unknown: 105   telephone:1467   jul    : 71
1
##  yes    :   1   yes    :2175   yes    : 665                    aug    : 63
6
##                                                               jun    : 53
0
##                                                               nov    : 44
6
##                                                               apr    : 21
5
##                                                               (Other): 20
3
##  day_of_week    duration         campaign          pdays          previous
```

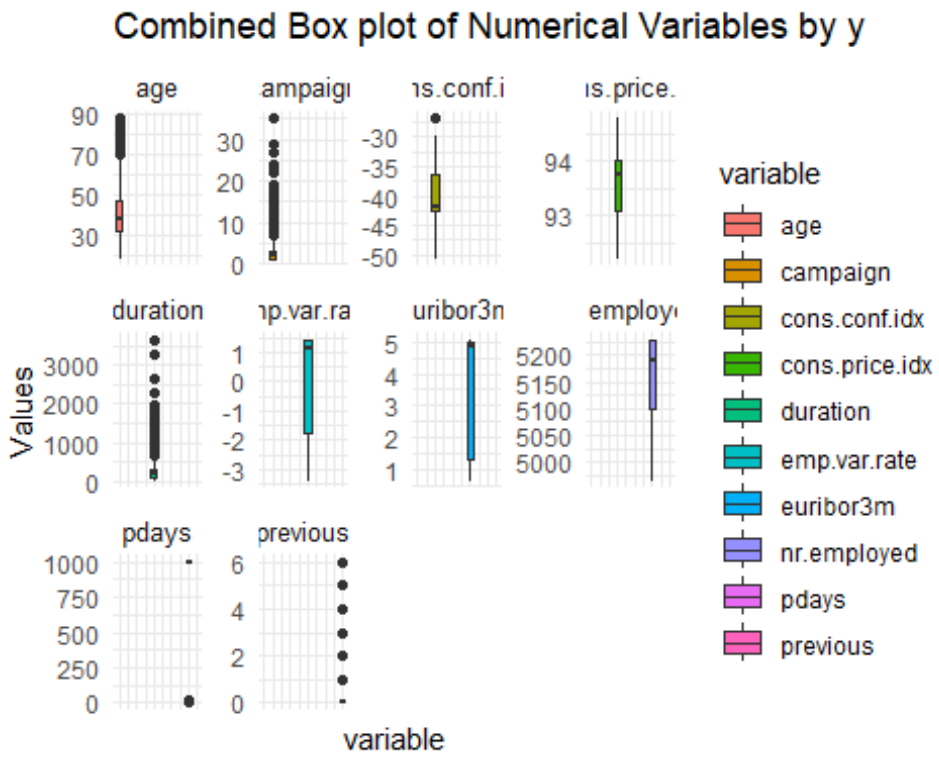
```
##  fri:768     Min.   :   0.0   Min.   : 1.000   Min.   :  0.0   Min.   :0.0
000
##  mon:855     1st Qu.: 103.0   1st Qu.: 1.000   1st Qu.:999.0   1st Qu.:0.0
000
##  thu:860     Median : 181.0   Median : 2.000   Median :999.0   Median :0.0
000
##  tue:841     Mean   : 256.8   Mean   : 2.537   Mean   :960.4   Mean   :0.1
903
##  wed:795     3rd Qu.: 317.0   3rd Qu.: 3.000   3rd Qu.:999.0   3rd Qu.:0.0
000
##              Max.   :3643.0   Max.   :35.000   Max.   :999.0   Max.   :6.0
000
##
##         poutcome     emp.var.rate      cons.price.idx  cons.conf.idx
##  failure    : 454   Min.   :-3.40000   Min.   :92.20   Min.   :-50.8
##  nonexistent:3523   1st Qu.:-1.80000   1st Qu.:93.08   1st Qu.:-42.7
##  success    : 142   Median : 1.10000   Median :93.75   Median :-41.8
##                     Mean   : 0.08497   Mean   :93.58   Mean   :-40.5
##                     3rd Qu.: 1.40000   3rd Qu.:93.99   3rd Qu.:-36.4
##                     Max.   : 1.40000   Max.   :94.77   Max.   :-26.9
##
##    euribor3m      nr.employed        y
##  Min.   :0.635   Min.   :4964   Length:4119
##  1st Qu.:1.334   1st Qu.:5099   Class :character
##  Median :4.857   Median :5191   Mode  :character
##  Mean   :3.621   Mean   :5166
##  3rd Qu.:4.961   3rd Qu.:5228
##  Max.   :5.045   Max.   :5228
##
```

performed factoring on all the categorical variables. It helped us visualize the levels and the counts of each level for each categorical variable in the dataframe.

3. **For the numerical variables, plot box plots based on values of y. Do you see a difference between the box plots for any of the variables you choose?**

```
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.3.2

library(dplyr)

## Warning: package 'dplyr' was built under R version 4.3.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidyr)

## Warning: package 'tidyr' was built under R version 4.3.2

# Marking y as factor
df$y <- as.factor(df$y)

# Selecting only numeric variables from df
num_vars <- df %>% select_if(is.numeric)

# Convert the data to long format using pivot_longer
long_data <- pivot_longer(
  data = num_vars,
  cols = everything(),
  names_to = "variable",
  values_to = "value"
)

# Plotting Long data
ggplot(long_data, aes(x = variable, y = value, fill = variable)) +
  geom_boxplot() +
  facet_wrap(~variable, scales = "free_y") +
  theme_minimal() +
  theme(
    axis.text.x = element_blank(),
    strip.background = element_blank(),
    strip.text.x = element_text(size = 10)
  ) +
  labs(title = "Combined Box plot of Numerical Variables by y", y = "Values")
```

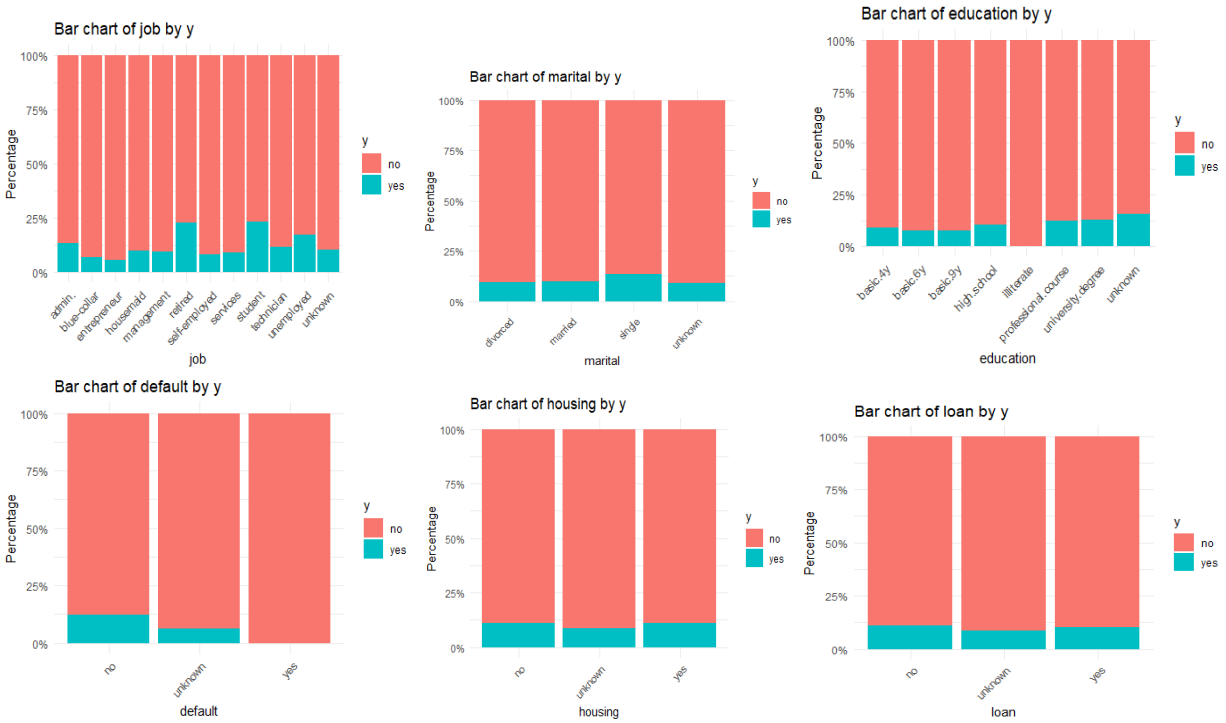


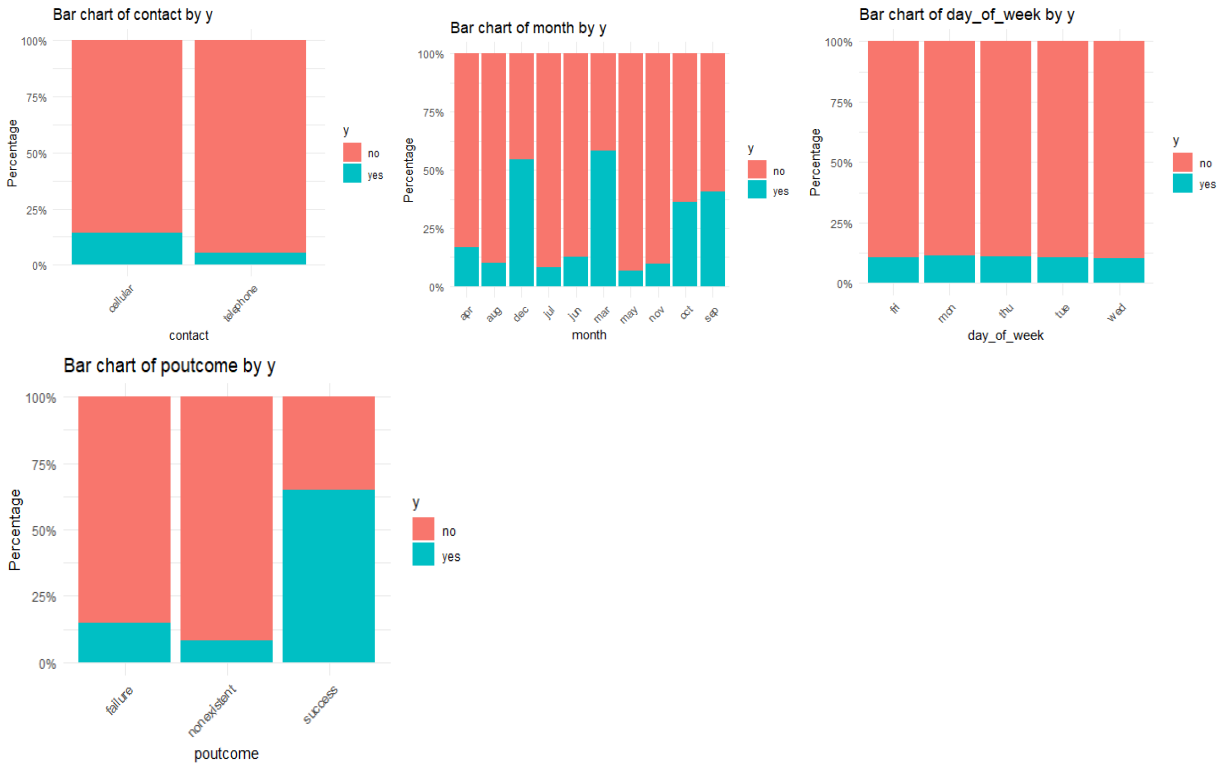
This image represents box plots for all the numerical variables present in the dataset. Its interesting to observe the variables “campaign”, “duration” and “previous” are the ones which have been suggested to be removed by Lasso down further in the analysis.

4. **For the categorical variables, plot bar charts for the different values of y. Do you see a difference between plots for any of the variables you choose?**

```
library(ggplot2)

categorical_variables <- names(df)[sapply(df, is.factor) & names(df) != 'y']

for (variable in categorical_variables) {
  p <- ggplot(df, aes_string(x = variable, fill = 'y')) +
    geom_bar(position = "fill") +
    scale_y_continuous(labels = scales::percent) +
    labs(title = paste("Bar chart of", variable, "by y"), x = variable, y = "
Percentage") +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))

  print(p)
}
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## ℹ Please use tidy evaluation idioms with `aes()`.
## ℹ See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```





We have provided individual histograms for every categorical variable in the dataset. Apart from "job", "month" and "poutcome" the rest of the variables almost showed relatively similar number of observations across all its levels.

Test/training separation: Separate your data into 80% training and 20% testing data. Do not forget to set seed. Please use the same separation for the whole assignment, as it is needed to be able to compare the models.

```
library(caret)

## Warning: package 'caret' was built under R version 4.3.2

## Loading required package: lattice

set.seed(45)
df$y = as.factor(df$y)
index = createDataPartition(df$y, p=.80, list = FALSE)
train=df[index,]
test=df[-index,]

c(nrow(df), nrow(train), nrow(test))

## [1] 4119 3296  823
```

performed stratified sampling based on the target variable as its imperative to use equal proportions of levels in testing and training data.

Part 2: Logistic Regression or LDA (15 points)

1. **Develop a classification model where the variable y is the dependent variable using the Logistic Regression or LDA, rest of the variables, and your training data set.**

```
library(caret)
library(MASS)

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

#install.packages("pROC")
library(pROC)

## Warning: package 'pROC' was built under R version 4.3.2

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

# Logistic Regression
set.seed(45)
logistic_fit <- glm(y ~ ., data = train, family = "binomial")

# Linear Discriminant Analysis
set.seed(45)
lda_fit=lda(y~., data=train)

## Warning in lda.default(x, grouping, ...): variables are collinear

# To compare models using ROC curve and AUC:
#For Logistic model
logistic_preds <- predict(logistic_fit, newdata = test, type = "response")
logistic_res <- roc(response = test$y, predictor = logistic_preds)

## Setting levels: control = no, case = yes

## Setting direction: controls < cases
```

```r
# For LDA model
lda_preds <- predict(lda_fit, newdata = test)$posterior[, "yes"]
lda_res <- roc(response = test$y, predictor = lda_preds)

## Setting levels: control = no, case = yes
## Setting direction: controls < cases

cat("AUC for Logistic Regression:", auc(logistic_res), "\n")

## AUC for Logistic Regression: 0.9211915

cat("AUC for LDA:", auc(lda_res), "\n")

## AUC for LDA: 0.9230863
```

approximately both logistic and LDA have same AUC. so we are choosing with logistic regression because LDA makes more assumptions about the underlying data so its not robust to outliers, whereas logistic regression is the more flexible and more robust method in case of violations of these assumptions.

```r
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.3.2

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.3.2

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-8

x <- model.matrix(y ~ . - 1, data=train)  # '-1' to exclude the intercept
y <- as.numeric(train$y) - 1  # Convert factor to numeric (0 and 1)

set.seed(45)  # For reproducibility
cv_lasso <- cv.glmnet(x, y, family="binomial", alpha=1)

# Optimal lambda
opt_lambda_lasso <- cv_lasso$lambda.min

lasso_coefs <- coef(cv_lasso, s = "lambda.min")

print(lasso_coefs)
```

```
## 55 x 1 sparse Matrix of class "dgCMatrix"
##                                       s1
## (Intercept)                   50.6585245160
## age                            .
## jobadmin.                      .
## jobblue-collar                 .
## jobentrepreneur               -0.4556030044
## jobhousemaid                   .
## jobmanagement                 -0.1147423311
## jobretired                    -0.0492980488
## jobself-employed              -0.0858386403
## jobservices                    .
## jobstudent                     .
## jobtechnician                  0.2158854419
## jobunemployed                  .
## jobunknown                     .
## maritalmarried                -0.0134033271
## maritalsingle                  0.0413690231
## maritalunknown                 .
## educationbasic.6y              .
## educationbasic.9y              .
## educationhigh.school           .
## educationilliterate            .
## educationprofessional.course   .
## educationuniversity.degree     .
## educationunknown               .
## defaultunknown                 .
## defaultyes                     .
## housingunknown                -0.0248545509
## housingyes                     .
## loanunknown                    .
## loanyes                        .
## contacttelephone              -0.2487895857
## monthaug                       0.0467547721
## monthdec                       1.2354963752
## monthjul                       .
## monthjun                       0.5032635694
## monthmar                       1.5631203142
## monthmay                      -0.5560849939
## monthnov                      -0.1836799590
## monthoct                       .
## monthsep                      -0.2552679016
## day_of_weekmon                 .
## day_of_weekthu                 .
## day_of_weektue                -0.0258929094
## day_of_weekwed                 .
## duration                       0.0047792850
## campaign                      -0.0203305097
## pdays                         -0.0008660701
## previous                       .
```

```
## poutcomenonexistent           0.2282509012
## poutcomesuccess               0.9514258360
## emp.var.rate                 -0.1554985363
## cons.price.idx                .
## cons.conf.idx                 0.0228007359
## euribor3m                     .
## nr.employed                  -0.0102910962
```

Performed LDA and logistic regression, found out there is high multicollinearity, in an effort to identify we have performed Lasso regression to identify such variables which cause multicollinearity as Lasso shrinks all the unnecessary features coefficients to zero making it easy for our modelling and analysis.

```
#converting target to numeric for train and test
train$y <- factor(train$y, levels = c("no", "yes"), labels = c("0", "1"))

test$y <- factor(test$y, levels = c("no", "yes"), labels = c("0", "1"))

logistic_fit2 <- glm(y ~ 0+. -age - education - default - loan - previous - c
ons.price.idx - euribor3m - duration, data = train, family = "binomial")

logistic_preds2 <- predict(logistic_fit2, newdata = test, type = "response")
```

Noticed the high multicollinearity between the variables such as age , education, default, loan, previous, cons.price.idx, euribor3m and duration since it destabilises the model, we are removing them and recalculating logistic regression in an attempt to extract a meaningful model. Note: We have also removed the feature “duration” because as described in the data source it’s not a variable that would be known before a call is performed so it should not be included in a realistic predictive model. This is the reason why we have excluded duration in our models. and because of that we have seen quite varying results.

2. **Obtain the confusion matrix and compute the testing error rate based on the logistic regression classification.**

```
# Convert probabilities to class labels based on a threshold
pred_labels <- ifelse(logistic_preds2 > 0.5, 1, 0)
pred_labels <- factor(pred_labels)
cm <- confusionMatrix(data = pred_labels, reference = test$y)
print(cm)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 717  73
##          1  16  17
```

```
##
##                Accuracy : 0.8919
##                  95% CI : (0.8686, 0.9123)
##     No Information Rate : 0.8906
##     P-Value [Acc > NIR] : 0.4835
##
##                   Kappa : 0.2313
##
##  Mcnemar's Test P-Value : 2.921e-09
##
##             Sensitivity : 0.9782
##             Specificity : 0.1889
##          Pos Pred Value : 0.9076
##          Neg Pred Value : 0.5152
##              Prevalence : 0.8906
##          Detection Rate : 0.8712
##    Detection Prevalence : 0.9599
##       Balanced Accuracy : 0.5835
##
##        'Positive' Class : 0
##
```

```
#testing error
print(paste("Logistic testing error", 1-cm$overall[1]))
```

```
## [1] "Logistic testing error 0.108140947752126"
```

2. **Explain your choices and communicate your results.**

   The model with updated set of predictors performs well in identifying true positives (high sensitivity) and generally classifies instances correctly with high accuracy. However, it has a considerably lower specificity of mere 18%, indicating a challenge in correctly identifying true negative instances. This can be even more understood by looking at the Neg Prediction Value score which is only 51%. So, Overall this model has moderate performance and it needs improvement in my opinion because of the low specificity score, possibly, due to the imbalanced nature of the dataset (as indicated by the negative prevalence rate).

Part 3: KNN (15 points)

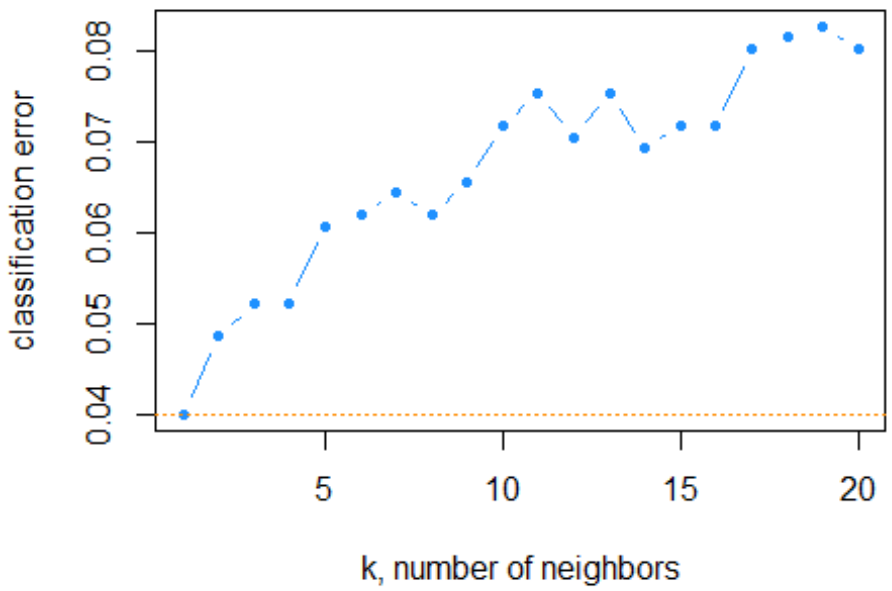
1. Apply a KNN classification to the training data using.

```
library(class)
library(caret)

set.seed(45)

# Create a model matrix for the training data
model_matrix_train <- model.matrix(~ 0+. -age - education - default - loan -
```

```
previous - cons.price.idx - euribor3m - duration, data = train)

# Similarly, for the testing data
model_matrix_test <- model.matrix(~ 0+. -age - education - default - loan - p
revious - cons.price.idx - euribor3m - duration, data = test)

#choosing best k
set.seed(45)
k.grid=1:20
error=rep(0, length(k.grid))

for (i in seq_along(k.grid)) {
  pred = knn(train = scale(model_matrix_train),
             test  = scale(model_matrix_test),
             cl    = train$y,
             k     = k.grid[i])
  error[i] = mean(test$y!=pred)
}

min(error)

## [1] 0.04009721
```

We have determined the optimal number of neighbors (k) to use in the KNN algorithm by running a loop (for (i in seq_along(k.grid))) over a range of k values from 1 to 20. For each k calculated the KNN prediction using scaled training data as input and compare these predictions to the actual outcomes in the testing set to calculate the error rate (error *i* = mean(test$y!=pred)). After computing the error rates for each k value, we have identify the minimum error rate using min(error). The k value corresponding to this minimum error rate is considered optimal for KNN

```
plot(error, type = "b", col = "dodgerblue", cex = 1, pch = 20,
     xlab = "k, number of neighbors", ylab = "classification error")
# add line for min error seen
abline(h = min(error), col = "darkorange", lty = 3)
```



before we run the KNN we want to make sure that we have found the best k-value with has the least classification error. based on the results, k=1 has the least error of value 0.04009721. So using that k, we are preparing the KNN based on this.

```
# Apply KNN
set.seed(45)
k <- 1 # Number of neighbors
knn_pred <- knn(train = model_matrix_train, test = model_matrix_test, cl = tr
ain$y, k = k)
```

2. **Obtain the confusion matrix and compute the testing error rate based on the KNN classification.**

```
# Evaluate the model
#table(Predicted = knn_pred, Actual = test$y)

cmk = confusionMatrix(knn_pred, test$y)
cmk

## Confusion Matrix and Statistics
##
##            Reference
## Prediction   0   1
##          0 710  47
##          1  23  43
##
```

```
##                Accuracy : 0.9149
##                  95% CI : (0.8938, 0.9331)
##     No Information Rate : 0.8906
##     P-Value [Acc > NIR] : 0.012575
##
##                   Kappa : 0.5055
##
##  Mcnemar's Test P-Value : 0.005977
##
##             Sensitivity : 0.9686
##             Specificity : 0.4778
##          Pos Pred Value : 0.9379
##          Neg Pred Value : 0.6515
##              Prevalence : 0.8906
##          Detection Rate : 0.8627
##    Detection Prevalence : 0.9198
##       Balanced Accuracy : 0.7232
##
##        'Positive' Class : 0
##
```

```
# Calculate and print testing error rate
error_rate <- 1 - cmk$overall['Accuracy']
print(paste("KNN testing error",error_rate))
```

```
## [1] "KNN testing error 0.0850546780072904"
```

This model now has significant improvement over the previous model by looking at the specificity and balanced accuracy.

3. Explain your choices and communicate your results.

Well, by choosing the optimal k=1 using cross validation by calculating the min error, the specificity of the models (identifying no in these cases) has increased drastically from 18% to 47%. Also, KNN model outperforms the Logistic Regression model in this case when considering the overall accuracy and both predictive values. It's especially more capable of correctly classifying false negatives, where the Logistic Regression model is struggling. However, both models exhibit a large difference between sensitivity and specificity. The No Information Rate also suggests that the model accuracy is significantly better than this rate, as indicated by the P-Value

$$Acc\ NIR$$

, which is small (0.012575). It means that the model has significantly predicted higher no of times for both the classes instead of randomly predicting class 0 being true. KNN does seem to have an edge here when compared to Logistic regression.

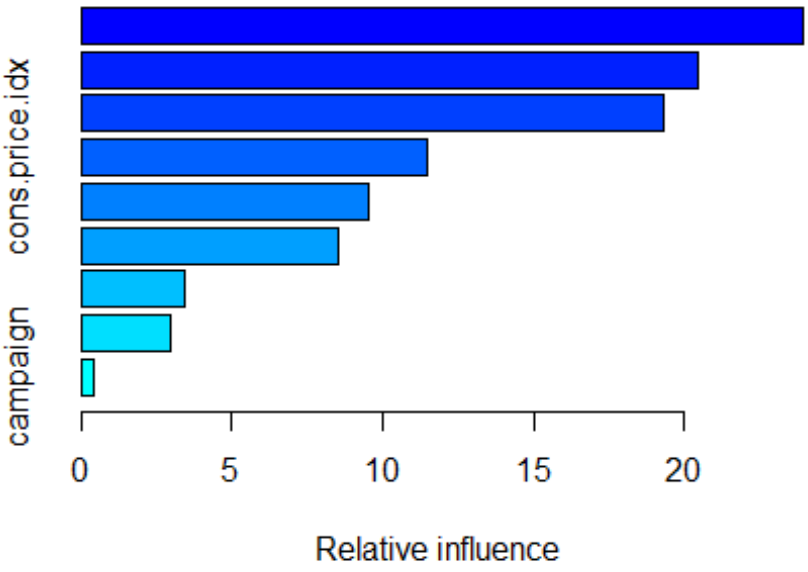
Part 4: Tree Based Model (15 points)

1. **Apply one of the following models to your training data: *Random Forrest, Bagging or Boosting***

```
df[,"y"]=ifelse(df[,"y"] == "yes", 1, 0)

separator =  sample(1:2,size=nrow(df), prob=c(0.80, 0.20), replace=TRUE)
train=df[separator == 1, ]
test=df[separator == 2, ]

numericalColumnsTrain <- sapply(train, is.numeric)

# Subset the dataset to include only numerical columns
train_numeric_only <- train[, numericalColumnsTrain]

library(gbm)

## Warning: package 'gbm' was built under R version 4.3.3

## Loaded gbm 2.1.9

## This version of gbm is no longer under development. Consider transitioning
to gbm3, https://github.com/gbm-developers/gbm3

set.seed(45)
model.boos <- gbm(formula = y ~ .-duration, distribution="bernoulli", data=tr
ain_numeric_only, n.trees = 10000)
print(model.boos)

## gbm(formula = y ~ . - duration, distribution = "bernoulli", data = train_n
umeric_only,
##     n.trees = 10000)
## A gradient boosted model with bernoulli loss function.
## 10000 iterations were performed.
## There were 9 predictors of which 9 had non-zero influence.
```

we have filtered the train and test for numeric data only to perform boosting because it inherently handle numerical inputs for their split decisions in the trees that they construct during the learning process. Each decision node in a tree represents a single feature, and the algorithm determines the best threshold for splitting the data into two groups to minimize the loss function.

```
summary(model.boos)
```




```
##                             var    rel.inf
## age                         age 23.9343432
## euribor3m           euribor3m 20.4322942
## cons.conf.idx   cons.conf.idx 19.3307934
## cons.price.idx cons.price.idx 11.4594300
## nr.employed       nr.employed  9.5153164
## pdays                   pdays  8.5498342
## emp.var.rate     emp.var.rate  3.4126372
## previous             previous  2.9362452
## campaign             campaign  0.4291062
```

We have selected numeric columns only for gradient descent boosting to determine how well they explain the target variable 'y'. It says all the 10 numeric predictors had influence on the target variable in the 1000 iterations. Later we have generated the model summary as shown in the image. It shows that consumer price index and the number of days passed since the last contact to the client are more relevant when determining the likelihood of a client subscribing to a term deposit It also shows that no of time time the client was contacted during this campaign are irrelevant.

2. **Obtain the confusion matrix and compute the testing error rate based on your chosen tree based model.**

```
#performing similar preliminary steps for test dataset as well
```

```r
# Identify numerical columns
numericalColumnsTest <- sapply(test, is.numeric)

# Subset the dataset to include only numerical columns
test_numeric_only <- test[, numericalColumnsTest]

test_numeric_only$binary <- as.numeric(as.character(test$y))

#Obtaining Predictions and Computing the Test Error Rate
pred.boost=predict(model.boos, newdata=test_numeric_only,n.trees=10000, distr
ibution="bernoulli", type="response")

head(pred.boost)
```

```
## [1] 0.03757157 0.04595555 0.05065403 0.06448122 0.04906312 0.02553918
```

```r
boostpred=ifelse(pred.boost < 0.5, 0, 1)
head(boostpred)
```

```
## [1] 0 0 0 0 0 0
```

The head(pred.boost) output has shown the first six predicted probabilities of the positive class (1). These values are between 0 and 1, representing the probability of the positive outcome as predicted by the model. when we round them up in the head(boostpred) we can see that the first six predictions made by the model are 0's

```r
cmb=confusionMatrix(factor(test_numeric_only[,"binary"]), factor(boostpred))
cmb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 727  18
##          1  64  21
##
##                Accuracy : 0.9012
##                  95% CI : (0.8789, 0.9207)
##     No Information Rate : 0.953
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2932
##
##  Mcnemar's Test P-Value : 6.715e-07
##
##             Sensitivity : 0.9191
##             Specificity : 0.5385
##          Pos Pred Value : 0.9758
```

```
##          Neg Pred Value : 0.2471
##              Prevalence : 0.9530
##          Detection Rate : 0.8759
##    Detection Prevalence : 0.8976
##       Balanced Accuracy : 0.7288
##
##        'Positive' Class : 0
##
```

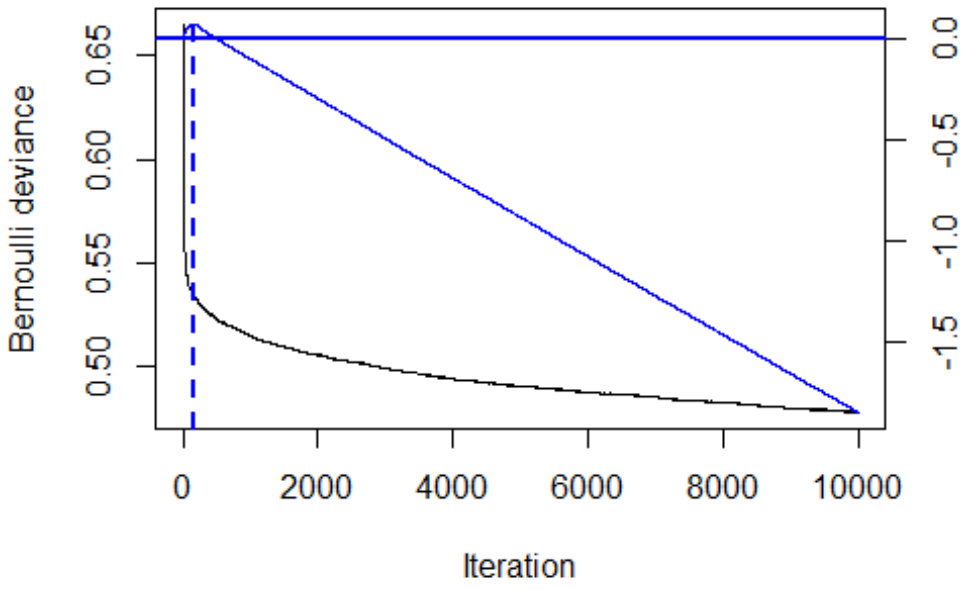
```
# Calculate and print testing error rate
error_rate <- 1 - cmb$overall['Accuracy']
print(paste("Gradient Descent Boosting testing error",error_rate))
```

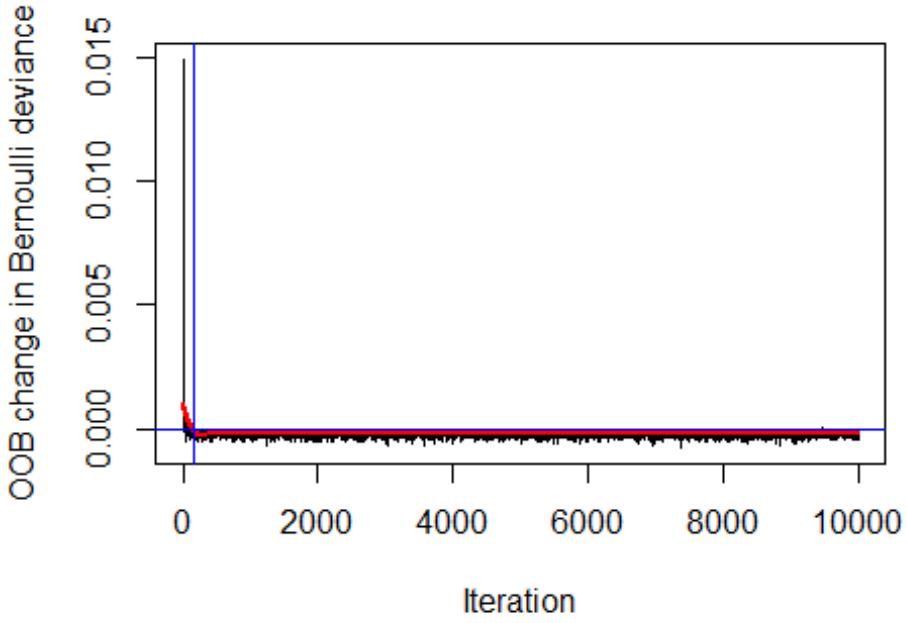
```
## [1] "Gradient Descent Boosting testing error 0.0987951807228916"
```

An accuracy of approximately 91.34% indicates that the majority of predictions match the actual outcomes. This model has specificity of 68.33% of actual negatives (class 1) were correctly identified, which is still moderate and suggests room for improvement in correctly predicting class 1. the model overall accuracy is 79.84% which is decent performance considering how imbalanced the data was.

```
ntree.oob.opt=gbm.perf(model.boos, method="OOB", oobag.curve=TRUE)
```

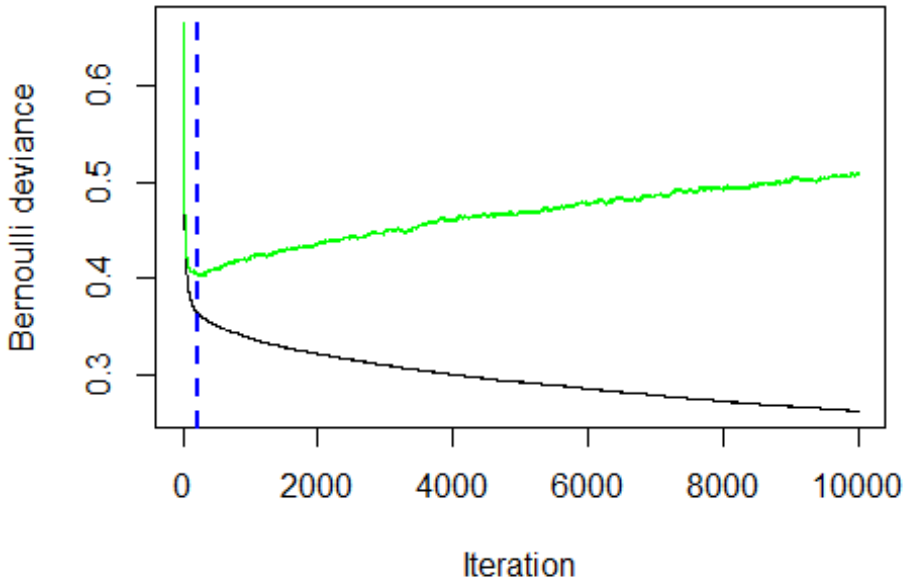
```
## OOB generally underestimates the optimal number of iterations although pre
dictive performance is reasonably competitive. Using cv_folds>1 when calling
gbm usually results in improved predictive performance.
```






```r
set.seed(45)
model.boos.cv <- gbm(y ~ .,
                     distribution = "bernoulli",
                     train_numeric_only,n.trees = 10000,
```

```
                          cv.folds = 3)
ntree.cv.opt=gbm.perf(model.boos.cv, method="cv")
```




```
print(paste0("Optimal ntrees (OOB Estimate): ", ntree.oob.opt))

## [1] "Optimal ntrees (OOB Estimate): 153"

print(paste0("Optimal ntrees (CV Estimate): ", ntree.cv.opt))

## [1] "Optimal ntrees (CV Estimate): 225"
```

So, to mitigate this we have performed both OOB and Cross validation to estimate the optimal number of trees required for this boosting model. It has shown that only 225 trees produced by the CV estimate and 153 trees produced by the OOB estimate are required to get the best predictions on the test dataset, so to find which one among them gave us more accuracy and specificity, we ran the gradient descent boosting models again.

```
#optimal trees model
model.boos <- gbm(formula = y ~ .-duration, distribution="bernoulli", data=tr
ain_numeric_only, n.trees = ntree.oob.opt)
print(model.boos)

## gbm(formula = y ~ . - duration, distribution = "bernoulli", data = train_n
umeric_only,
##     n.trees = ntree.oob.opt)
```

```
## A gradient boosted model with bernoulli loss function.
## 153 iterations were performed.
## There were 9 predictors of which 9 had non-zero influence.

#optimal trees model
model.boos.cv <- gbm(formula = y ~ .-duration, distribution="bernoulli", data
=train_numeric_only, n.trees = ntree.cv.opt)
print(model.boos)

## gbm(formula = y ~ . - duration, distribution = "bernoulli", data = train_n
umeric_only,
##     n.trees = ntree.oob.opt)
## A gradient boosted model with bernoulli loss function.
## 153 iterations were performed.
## There were 9 predictors of which 9 had non-zero influence.

pred.1=predict(object = model.boos,
                  newdata = test_numeric_only,
                  n.trees = ntree.oob.opt)
pred.2=predict(object = model.boos.cv,
                  newdata = test_numeric_only,
                  n.trees = ntree.cv.opt)
head(pred.1)

## [1] -2.957978 -2.930387 -2.971041 -2.957978 -2.930387 -2.978347

boostpred.oob=ifelse(pred.1 < 0.5, 0, 1)
head(boostpred.oob)

## [1] 0 0 0 0 0 0

head(pred.2)

## [1] -3.015180 -3.004894 -3.006718 -2.962027 -2.951741 -3.095696

boostpred.cv=ifelse(pred.2 < 0.5, 0, 1)
head(boostpred.cv)

## [1] 0 0 0 0 0 0

#confusion matrix for the optimal model
cmbo=confusionMatrix(factor(test_numeric_only[,"y"]), factor(boostpred.oob))
cmbo

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 742   3
##          1  73  12
##
##                Accuracy : 0.9084
```

```
##                  95% CI : (0.8867, 0.9272)
##     No Information Rate : 0.9819
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2159
##
##  Mcnemar's Test P-Value : 2.476e-15
##
##             Sensitivity : 0.9104
##             Specificity : 0.8000
##          Pos Pred Value : 0.9960
##          Neg Pred Value : 0.1412
##              Prevalence : 0.9819
##          Detection Rate : 0.8940
##    Detection Prevalence : 0.8976
##       Balanced Accuracy : 0.8552
##
##        'Positive' Class : 0
##
```

```
# Calculate and print testing error rate for the optimal model
error_rate <- 1 - cmbo$overall['Accuracy']
print(paste("Gradient Descent Boosting optimal OOB model testing error",error
_rate))
```

```
## [1] "Gradient Descent Boosting optimal OOB model testing error 0.091566265
060241"
```

```
#confusion matrix for the optimal model
cmbv=confusionMatrix(factor(test_numeric_only[,"y"]), factor(boostpred.cv))
cmbv
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 740   5
##          1  73  12
##
##                Accuracy : 0.906
##                  95% CI : (0.8841, 0.925)
##     No Information Rate : 0.9795
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2083
##
##  Mcnemar's Test P-Value : 3.293e-14
##
##             Sensitivity : 0.9102
##             Specificity : 0.7059
##          Pos Pred Value : 0.9933
```

```
##          Neg Pred Value : 0.1412
##              Prevalence : 0.9795
##          Detection Rate : 0.8916
##    Detection Prevalence : 0.8976
##       Balanced Accuracy : 0.8080
##
##        'Positive' Class : 0
##
```

```
# Calculate and print testing error rate for the optimal model
error_rate <- 1 - cmbv$overall['Accuracy']
print(paste("Gradient Descent Boosting optimal CV testing error",error_rate))
```

```
## [1] "Gradient Descent Boosting optimal CV testing error 0.0939759036144578
"
```

As explained by these optimal model test results we can see that it has produced 71% specificity and 81.4% of balanced accuracy. The total testing error stands at 9.5% which shows that it has failed quite less and it stands similarly among the other previous models.

```
library(pROC)
# Compare AUC
auc1=auc(test_numeric_only$y,pred.1)  #OOB
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
auc2=auc(test_numeric_only$y,pred.2)  #CV
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
print(paste0("Test set AUC (OOB): ", round(auc1,4)))
```

```
## [1] "Test set AUC (OOB): 0.76"
```

```
print(paste0("Test set AUC (CV): ", round(auc2,4)))
```

```
## [1] "Test set AUC (CV): 0.7606"
```

To further check we computed AUC for both these models. Here, both AUC values are quite close, suggesting that the model's performance is consistent across different methods of evaluation. An AUC of 0.76 indicates that the model has a good ability to discriminate between the positive class (case = 1) and the negative class (control = 0). In the context of a ROC curve, this means that there is a 76% chance that the model will be able to distinguish between a randomly chosen positive instance and a randomly chosen negative instance.

3. **Explain your choices and communicate your results.**

Both the randomly chosen and the optimised boosting models have performed singificantly better than the previous models in terms of specificity and balanced accuracy. The optimised model has performed slightly better than the randomly chosen number of trees models which is to be considered. One thing that we should be aware of is the No information rate is better at identifying the classes than this model. (Its representative by the P-Value *Acc NIR*: 1). This straight away rejects the null hypothesis because P-value = 1, suggesting that there is no statistical evidence from this test that the model is performing better than the NIR.

Part 5: SVM (15 points)

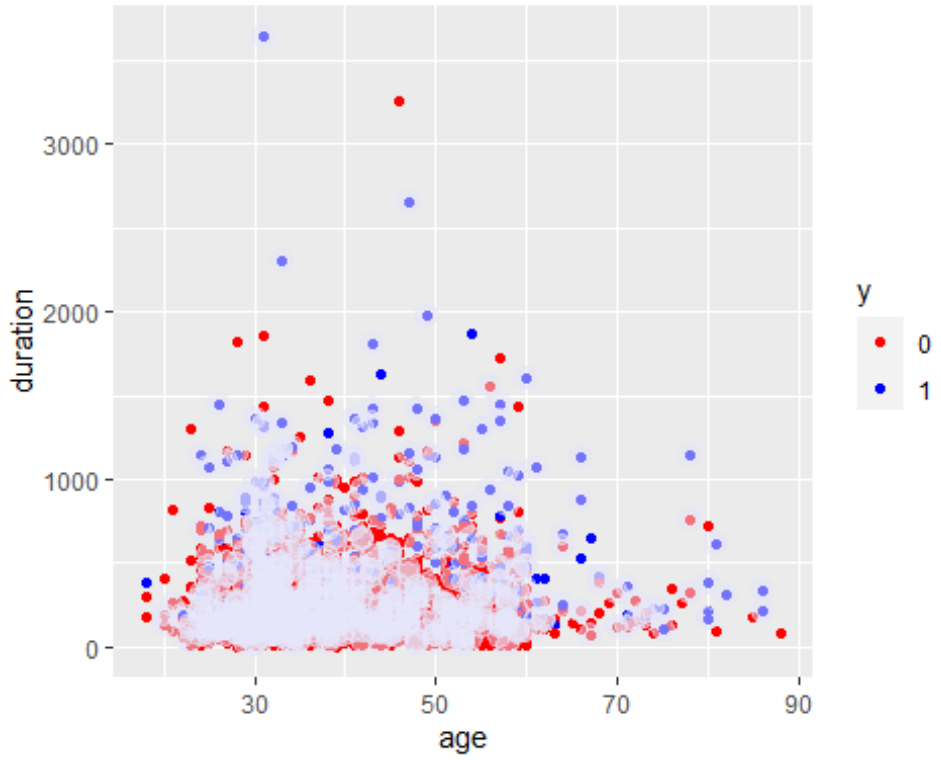
1. **Apply a SVM model to your training data.**

```
set.seed(45)
library(e1071)

## Warning: package 'e1071' was built under R version 4.3.2

#test and train split:
df[, "train"] = ifelse(runif(nrow(df))<.8, 1, 0)
train_svm <- df[df$train == 1, ]
test_svm <- df[df$train == 0, ]
#find index of "train" column
index <- grep("train", names(df))
#remove "train" column from train and test dataset
train_svm <- train_svm[, -index]
test_svm <- test_svm[, -index]

#build svm model, setting required parameters
svm_model<- svm(y ~ . -age - education - default - loan - previous - cons.pri
ce.idx - euribor3m - duration, data = train_svm, type = "C-classification", k
ernel = "linear", scale = FALSE, cost=0.1)
svm_model

##
## Call:
## svm(formula = y ~ . - age - education - default - loan - previous -
##     cons.price.idx - euribor3m - duration, data = train_svm, type = "C-cla
ssification",
##     kernel = "linear", cost = 0.1, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  0.1
##
## Number of Support Vectors:  590
```

in the next step we have performed SVM using the similar set of predictors used in logistic regression and KNN. Initially we have used a linear kernel, where algorithm will attempt to separate the classes with a straight line or a hyperplane. It works well when the data is linearly separable. The parameter cost is set ti 0.1 to assign relatively low penalty for misclassification.

```
train_svm$y = as.factor(train_svm$y)
svm_plot=ggplot(data = train_svm, aes(x = age, y = duration, color = y)) +
    geom_point() +
    scale_color_manual(values = c("red", "blue")) +
    geom_point(data = train_svm[svm_model$index, ], aes(x = age, y = duration
), color = "lavender", size = 4, alpha = 0.5)

svm_plot
```



This is svm_plot which is the scatter plot that visualizes the relationship between age and duration with the points colored according to the y class, if we can see the plot, there is a significant overlap between the class variables, large overlap between the two classes in terms of both age and duration possibly indicating that these two features alone may not be strong discriminators between the two classes for this SVM model.

```
svm_radial<- svm(y ~ . -age - education - default - loan - previous - cons.pr
ice.idx - euribor3m - duration, data = train_svm, type = "C-classification",
kernel = "radial", scale = TRUE)
svm_radial
```

```
##
## Call:
## svm(formula = y ~ . - age - education - default - loan - previous -
##     cons.price.idx - euribor3m - duration, data = train_svm, type = "C-cla
ssification",
##     kernel = "radial", scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  961
```

performing svm radial as well because the RBF kernel is more flexible than the linear kernel. It can handle situations where the boundary between classes is not a straight line. The RBF kernel uses a parameter called gamma, which defines how far the influence of a single training example reaches.

2. Calculate the confusion matrix using the testing data.

```
#compute test accuracy
pred_linear <- predict(svm_model, test_svm)

pred_radial=predict(svm_radial,test_svm)

test_svm$y <- factor(test_svm$y)
levels(test_svm$y)
```

```
## [1] "0" "1"
```

```
#linear model CF
cmsv1 <- confusionMatrix(data = pred_linear, reference = test_svm$y)
cmsv1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 732  76
##          1   7  14
##
##                Accuracy : 0.8999
##                  95% CI : (0.8774, 0.9195)
```

```
##     No Information Rate : 0.8914
##     P-Value [Acc > NIR] : 0.2361
##
##                   Kappa : 0.2202
##
##  Mcnemar's Test P-Value : 8.395e-14
##
##             Sensitivity : 0.9905
##             Specificity : 0.1556
##          Pos Pred Value : 0.9059
##          Neg Pred Value : 0.6667
##              Prevalence : 0.8914
##          Detection Rate : 0.8830
##    Detection Prevalence : 0.9747
##       Balanced Accuracy : 0.5730
##
##        'Positive' Class : 0
##
```

```
#radial model CF
cmsv2 <- confusionMatrix(data = pred_radial, reference = test_svm$y)
cmsv2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 730  78
##          1   9  12
##
##                Accuracy : 0.8951
##                  95% CI : (0.8722, 0.9151)
##     No Information Rate : 0.8914
##     P-Value [Acc > NIR] : 0.3953
##
##                   Kappa : 0.1826
##
##  Mcnemar's Test P-Value : 3.091e-13
##
##             Sensitivity : 0.9878
##             Specificity : 0.1333
##          Pos Pred Value : 0.9035
##          Neg Pred Value : 0.5714
##              Prevalence : 0.8914
##          Detection Rate : 0.8806
##    Detection Prevalence : 0.9747
##       Balanced Accuracy : 0.5606
##
##        'Positive' Class : 0
##
```

From your output, both models have high sensitivity (above 98%), indicating they are good at identifying the 'positive' class (class "0"). However, the specificity is very low for both models (15.56% and 13.33%), suggesting that they struggle to correctly identify the 'negative' class (class "1"). The low specificity also drags down the balanced accuracy to around 57%, which is not very high.

In terms of which model performed better, it seems that the first model with linear kernel has slightly higher overall accuracy and balanced accuracy. However, both models do not perform the worst among all in terms of balanced accuracy, due to the low specificity.

```
set.seed (45)
tune.radial=tune(svm, y~. -age - education - default - loan - previous - cons
.price.idx - euribor3m - duration, data=train_svm,
                 kernel ="radial",
                 type = "C-classification",
                 ranges =list(cost=c(0.1,1,10,100, 1000),
                              gamma=c(0.5,1,2,3,4)))

summary(tune.radial)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.1045593
##
## - Detailed performance results:
##     cost gamma     error  dispersion
## 1  1e-01   0.5 0.1097264 0.009659726
## 2  1e+00   0.5 0.1045593 0.010741526
## 3  1e+01   0.5 0.1246201 0.011815505
## 4  1e+02   0.5 0.1361702 0.010509668
## 5  1e+03   0.5 0.1358663 0.012077579
## 6  1e-01   1.0 0.1097264 0.009659726
## 7  1e+00   1.0 0.1100304 0.013802947
## 8  1e+01   1.0 0.1261398 0.010553529
## 9  1e+02   1.0 0.1276596 0.009823059
## 10 1e+03   1.0 0.1276596 0.009823059
## 11 1e-01   2.0 0.1097264 0.009659726
## 12 1e+00   2.0 0.1121581 0.011854536
## 13 1e+01   2.0 0.1249240 0.011854536
## 14 1e+02   2.0 0.1249240 0.011854536
```

```
## 15 1e+03   2.0 0.1249240 0.011854536
## 16 1e-01   3.0 0.1097264 0.009659726
## 17 1e+00   3.0 0.1133739 0.011644860
## 18 1e+01   3.0 0.1246201 0.011902067
## 19 1e+02   3.0 0.1246201 0.011902067
## 20 1e+03   3.0 0.1246201 0.011902067
## 21 1e-01   4.0 0.1097264 0.009659726
## 22 1e+00   4.0 0.1133739 0.011644860
## 23 1e+01   4.0 0.1234043 0.011745797
## 24 1e+02   4.0 0.1234043 0.011745797
## 25 1e+03   4.0 0.1234043 0.011745797
```

performed svm radial tuning to see the best tuning for the hyperparameters. According to the error rate, is achieved with a cost of 1 and gamma of 0.5, resulting in the lowest error rate of approximately 0.1046 (or 10.46%).

```
#linear model accuracy
mean(pred_linear == test_svm$y)

## [1] 0.8998794

#radial model accuracy
mean(pred_radial==test_svm$y)

## [1] 0.8950543

#bestmod model accuracy
bestmod=tune.radial$best.model
pred_test2=predict(bestmod, test_svm)
mean(pred_test2 == test_svm$y)

## [1] 0.8974668
```

3. **Explain your choices and communicate your results.**

   Overall, the calculated the accuracy for each model - linear SVM, radial SVM, and the best tuned radial SVM model. The accuracies are close, with the tuned radial SVM being slightly better than the other two. This suggests that the parameter tuning process did manage to find a better performing model on the test data than the default settings of either the linear or radial models. However, the difference is so minor, possibly due to complexity of the model tuning process as it quite signifies the increase in performance to such minor extent.

Part 6: Conclusion (20 points)

1. **(10 points) Based on the different classification models, which one do you think is the best model to predict y? Please consider the following in your response:**

– Accuracy/error rates

generally regarding the comparison of Error Rates: Direct comparison of testing error rates among different models can be used to assess which model predicts most accurately. Lower testing error rates are generally preferable, So overall all the models have relatively close testing error rate values which speak about the overall accuracies of all the models suggesting that all have performed decently. Balance Between Sensitivity and Specificity: It's also important to consider the balance between sensitivity (true positive rate) and specificity (true negative rate) in the testing errors. Depending on the application, a slightly higher testing error might be acceptable if the model achieves a significantly better sensitivity or specificity. Boositing has outperformed all other models in this case. Contextual Performance: Testing error alone is not the end-all metric. The impact of false positives and false negatives can be drastically different in real-world scenarios. In the case of all the models that have been ran: Logistic regression and LDA showed decent performance, but depending on the complexities of the dataset, they might not capture all the nuances. KNN is very sensitive to the value of k and the distance metric used. Its performance can vary greatly with these parameters. SVM is a strong classifier, especially with a suitable kernel like the radial basis function. However, it requires careful parameter tuning to achieve optimal performance. Boosting typically shows good performance on a variety of datasets due to its iterative approach that focuses on correcting the mistakes of previous models. so probably this is the case why boosting has such low overall error rates.

- Do you think you can improve the model by adding any other information?

Boosting, particularly gradient boosting, is a powerful technique that can produce highly accurate models, especially for complex datasets with non-linear patterns. Here's why boosting might be outperforming the other models in your case: Model Complexity: Boosting builds a model in stages, and it generalizes well by combining the output of many "weak learners" to form a strong learner. This iterative process can capture complex patterns in the data, which might be missed by simpler models like linear regression or even SVM with linear kernel. Handling Imbalanced Data: Boosting can be particularly effective on imbalanced datasets. It focuses more on the difficult cases and can handle the nuances of the minority class better than some other algorithms. Reduction of Variance and Bias: By sequentially applying weak models to modified versions of the data, boosting reduces both the bias and variance, which can lead to improved model performance. In contrast, models like SVM might not handle both bias and variance as effectively without careful tuning. Non-Linearity: The dataset seems to contain non-linear relationships that are not well-captured by linear models. Boosting, with its flexibility, can better model these non-linear interactions. since boosting has performed very well, there is only 1 way we could think of to further improve the model even more by performing feature Engineering. Here more predictive features could improve model performance. This might include creating interaction terms, polynomial features, or aggregating features if you have temporal or spatial data.

2. **(10 points) What are your learning outcomes for this assignment? Please focus on your learning outcomes in terms of statistical learning, model interpretations, and R skills - it is up to you to include this part in your presentation or not.**

As always, there is no particular go to model that we can say absolutely its the best. because there are so many factors influecing the classification starting from the choice of the dataset which has huge influence- the dataset complexity, multicollinearity betwen predictors, imbalance before and/or after test train split, type of sampling techniques used, even the seed value we choose give us lay additional influence on the analyis and the results produced. while working we had to iterate through so many ways to see and understand why the model is performing in that particular way. in that process we have learnt the structure of the dataset is also important. how the variables are distributed and how they influence each other and the target variable. So performing preliminary steps helped us understand and fine tune the models we have got for every part. An additional step of comparision with other counterparts such as models and accuracies helped us gain insights on what to inerpret on the basis of explaining how the predictors are influencing the target variable. In terms of R skills which was quite new to use when we have started this course, we have come far along in terms of the programming language skill. Now we are able to write decent level code with some complexitiy without relying on external help that much. The process followed for every part in these classifications models is relatively same. so it was easy for us and it helped us give more time for analysis and less time spend on writing code and tuning it.