Project

MICHAEL AHANA

2024-04-05

1

This involves the use of cross-validation in classification on the German Credit Risk. The purpose of this analysis is to implement a machine learning algorithm to predict the credit risk (good or bad) of a consumer in the German market.

Part A

The goal of this data analysis is to develop a machine learning model that accurately predicts the credit risk of consumers in the German Market. The aim is to predict whether a consumer's credit risk is categorized as "good" or "bad" based on feature available in the dataset.

Inputs (Features):

a. Account Balance
b. Duration of Credit (month)
c. Payment Status of Previous Credit
d. Purpose
e. Credit Amount
f. Value Savings/Stocks
g. Length of current employment
h. Instalment per cent
i. Sex & Marital Status
j. Guarantors
k. Duration in Current address
l. Most valuable available asset
m. Age (years)
n. Concurrent Credits
o. Type of apartment
p. No of Credits at this Bank
q. Occupation
r. No of dependents
s. Telephone
t. Foreign Worker

Output (Target):

u. Creditability

EXPLORATORY ANALYSIS

Understanding the characteristics of the dataset before diving into model building or analysis.

Structure of the data (Summary Statistics)

```r
# Read the CSV file into a data frame
credit_data <- read.csv("german_credit.csv")

# Check the structure of the data
str(credit_data)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ Creditability                    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Account.Balance                  : int  1 1 2 1 1 1 1 1 4 2 ...
##  $ Duration.of.Credit..month.       : int  18 9 12 12 12 10 8 6 18 24 ...
##  $ Payment.Status.of.Previous.Credit: int  4 4 2 4 4 4 4 4 4 2 ...
##  $ Purpose                          : int  2 0 9 0 0 0 0 0 3 3 ...
##  $ Credit.Amount                    : int  1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
##  $ Value.Savings.Stocks             : int  1 1 2 1 1 1 1 1 1 3 ...
##  $ Length.of.current.employment     : int  2 3 4 3 3 2 4 2 1 1 ...
##  $ Instalment.per.cent              : int  4 2 2 3 4 1 1 2 4 1 ...
##  $ Sex...Marital.Status             : int  2 3 2 3 3 3 3 3 2 2 ...
##  $ Guarantors                       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Duration.in.Current.address      : int  4 2 4 2 4 3 4 4 4 4 ...
##  $ Most.valuable.available.asset    : int  2 1 1 1 2 1 1 1 3 4 ...
##  $ Age..years.                      : int  21 36 23 39 38 48 39 40 65 23 ...
##  $ Concurrent.Credits               : int  3 3 3 3 1 3 3 3 3 3 ...
##  $ Type.of.apartment                : int  1 1 1 1 2 1 2 2 2 1 ...
##  $ No.of.Credits.at.this.Bank       : int  1 2 1 2 2 2 2 1 2 1 ...
##  $ Occupation                       : int  3 3 2 2 2 2 2 2 1 1 ...
##  $ No.of.dependents                 : int  1 2 1 2 1 2 1 2 1 1 ...
##  $ Telephone                        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Foreign.Worker                   : int  1 1 1 2 2 2 2 2 1 1 ...
```

```r
# Display the first few rows of the data
head(credit_data)
```

```
##   Creditability Account.Balance Duration.of.Credit..month.
## 1             1               1                         18
## 2             1               1                          9
## 3             1               2                         12
## 4             1               1                         12
## 5             1               1                         12
## 6             1               1                         10
##   Payment.Status.of.Previous.Credit Purpose Credit.Amount Value.Savings.Stocks
## 1                                 4       2          1049                    1
## 2                                 4       0          2799                    1
## 3                                 2       9           841                    2
## 4                                 4       0          2122                    1
## 5                                 4       0          2171                    1
## 6                                 4       0          2241                    1
##   Length.of.current.employment Instalment.per.cent Sex...Marital.Status
## 1                            2                   4                    2
## 2                            3                   2                    3
## 3                            4                   2                    2
## 4                            3                   3                    3
## 5                            3                   4                    3
## 6                            2                   1                    3
```

```
##   Guarantors Duration.in.Current.address Most.valuable.available.asset
## 1          1                           4                             2
## 2          1                           2                             1
## 3          1                           4                             1
## 4          1                           2                             1
## 5          1                           4                             2
## 6          1                           3                             1
##   Age..years. Concurrent.Credits Type.of.apartment No.of.Credits.at.this.Bank
## 1          21                  3                 1                          1
## 2          36                  3                 1                          2
## 3          23                  3                 1                          1
## 4          39                  3                 1                          2
## 5          38                  1                 2                          2
## 6          48                  3                 1                          2
##   Occupation No.of.dependents Telephone Foreign.Worker
## 1          3                1         1              1
## 2          3                2         1              1
## 3          2                1         1              1
## 4          2                2         1              2
## 5          2                1         1              2
## 6          2                2         1              2
```

Checking for missing values

```r
# Check for missing values in each column
missing_values <- colSums(is.na(credit_data))

# Display columns with missing values and their corresponding counts
missing_values[missing_values > 0]
```

```
## named numeric(0)
```

it means that there are no missing values in the dataset. This is good news !

Data Transformation

Feature Scaling: Feature scaling is important to ensure that features with different scales and units contribute equally to the model training process. In the dataset, some features like "Credit Amount" and "Age (years)" have much larger scales compared to others like "Duration of Credit (month)" and "Instalment per cent". Scaling these features can improve the performance of the machine learning algorithms.

```r
# Identify numerical columns excluding the response variable
numeric_columns <- names(credit_data)[sapply(credit_data, is.numeric)]
numeric_columns <- setdiff(numeric_columns, "Creditability")

# Standardize numerical features
credit_data[numeric_columns] <- scale(credit_data[numeric_columns])
head(credit_data)
```

```
##   Creditability Account.Balance Duration.of.Credit..month.
## 1             1      -1.2539382                 -0.2407368
## 2             1      -1.2539382                 -0.9870788
## 3             1      -0.4587967                 -0.7382981
## 4             1      -1.2539382                 -0.7382981
## 5             1      -1.2539382                 -0.7382981
## 6             1      -1.2539382                 -0.9041519
```

```
##   Payment.Status.of.Previous.Credit   Purpose Credit.Amount
## 1                         1.3433419 -0.301701    -0.7872630
## 2                         1.3433419 -1.030447    -0.1673006
## 3                        -0.5031762  2.248911    -0.8609500
## 4                         1.3433419 -1.030447    -0.4071375
## 5                         1.3433419 -1.030447    -0.3897785
## 6                         1.3433419 -1.030447    -0.3649800
##   Value.Savings.Stocks Length.of.current.employment Instalment.per.cent
## 1          -0.69935708                   -1.1454050          0.91801781
## 2          -0.69935708                   -0.3178002         -0.86974813
## 3          -0.06645474                    0.5098045         -0.86974813
## 4          -0.69935708                   -0.3178002          0.02413484
## 5          -0.69935708                   -0.3178002          0.91801781
## 6          -0.69935708                   -1.1454050         -1.76363111
##   Sex...Marital.Status Guarantors Duration.in.Current.address
## 1           -0.9631679 -0.3035339                   1.0464631
## 2            0.4491018 -0.3035339                  -0.7655942
## 3           -0.9631679 -0.3035339                   1.0464631
## 4            0.4491018 -0.3035339                  -0.7655942
## 5            0.4491018 -0.3035339                   1.0464631
## 6            0.4491018 -0.3035339                   0.1404344
##   Most.valuable.available.asset Age..years. Concurrent.Credits
## 1                    -0.3408845 -1.28093214          0.4606002
## 2                    -1.2930760  0.04034293          0.4606002
## 3                    -1.2930760 -1.10476213          0.4606002
## 4                    -1.2930760  0.30459795          0.4606002
## 5                    -0.3408845  0.21651294         -2.3738626
## 6                    -1.2930760  1.09736299          0.4606002
##   Type.of.apartment No.of.Credits.at.this.Bank Occupation No.of.dependents
## 1        -1.7503294                 -0.7045734  0.1468757       -0.4280754
## 2        -1.7503294                  1.0265652  0.1468757        2.3337012
## 3        -1.7503294                 -0.7045734 -1.3830794       -0.4280754
## 4        -1.7503294                  1.0265652 -1.3830794        2.3337012
## 5         0.1358014                  1.0265652 -1.3830794       -0.4280754
## 6        -1.7503294                  1.0265652 -1.3830794        2.3337012
##    Telephone Foreign.Worker
## 1 -0.8229061     -0.1959163
## 2 -0.8229061     -0.1959163
## 3 -0.8229061     -0.1959163
## 4 -0.8229061      5.0991176
## 5 -0.8229061      5.0991176
## 6 -0.8229061      5.0991176
```

This transformation ensures that features with larger scales do not dominate the learning process and helps the algorithm converge faster.

Part B

Building a classifier to predict the creditability of a consumer using an appropriate machine learning algorithm.

```r
# Load required library
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
# Define formula for logistic regression
formula <- Creditability ~ .

# Train logistic regression model
logistic_model <- glm(formula, data = credit_data, family = binomial)

# Summarize the model
summary(logistic_model)
```

```
##
## Call:
## glm(formula = formula, family = binomial, data = credit_data)
##
## Coefficients:
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                        1.14834    0.08886  12.924  < 2e-16 ***
## Account.Balance                    0.72934    0.08808   8.280  < 2e-16 ***
## Duration.of.Credit..month.        -0.29629    0.10521  -2.816 0.004862 **
## Payment.Status.of.Previous.Credit  0.41396    0.09467   4.373 1.23e-05 ***
## Purpose                            0.08653    0.08257   1.048 0.294697
## Credit.Amount                     -0.26366    0.11325  -2.328 0.019908 *
## Value.Savings.Stocks               0.37780    0.09207   4.104 4.07e-05 ***
## Length.of.current.employment       0.18334    0.08600   2.132 0.033027 *
## Instalment.per.cent               -0.33375    0.09258  -3.605 0.000312 ***
## Sex...Marital.Status               0.18225    0.08194   2.224 0.026131 *
## Guarantors                         0.16589    0.08489   1.954 0.050681 .
## Duration.in.Current.address       -0.01558    0.08545  -0.182 0.855335
## Most.valuable.available.asset     -0.19202    0.09557  -2.009 0.044521 *
## Age..years.                        0.10123    0.09316   1.087 0.277218
## Concurrent.Credits                 0.17068    0.07837   2.178 0.029420 *
## Type.of.apartment                  0.15538    0.08891   1.748 0.080527 .
## No.of.Credits.at.this.Bank        -0.14071    0.09300  -1.513 0.130257
## Occupation                         0.01235    0.08934   0.138 0.890081
## No.of.dependents                  -0.06183    0.08398  -0.736 0.461567
## Telephone                          0.14467    0.09230   1.567 0.117024
## Foreign.Worker                     0.21875    0.11478   1.906 0.056680 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1221.73  on 999  degrees of freedom
## Residual deviance:  956.56  on 979  degrees of freedom
## AIC: 998.56
##
## Number of Fisher Scoring iterations: 5
```

Variable Selection

we use the step() function to perform stepwise selection. The argument direction = "both" allows the algorithm to consider both forward and backward steps.The final model will contain only the significant variables according to the chosen criteria

Finally, display the summary of the final model to see which variables were selected and their coefficients.

```r
# Fit the initial model
initial_model <- glm(Creditability ~ ., family = binomial, data = credit_data)

# Perform stepwise selection
final_model <- step(initial_model, direction = "both")
```

```
## Start:  AIC=998.56
## Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Purpose + Credit.Amount +
##     Value.Savings.Stocks + Length.of.current.employment + Instalment.per.cent +
##     Sex...Marital.Status + Guarantors + Duration.in.Current.address +
##     Most.valuable.available.asset + Age..years. + Concurrent.Credits +
##     Type.of.apartment + No.of.Credits.at.this.Bank + Occupation +
##     No.of.dependents + Telephone + Foreign.Worker
##
##                                     Df Deviance     AIC
## - Occupation                         1   956.58  996.58
## - Duration.in.Current.address        1   956.59  996.59
## - No.of.dependents                   1   957.09  997.09
## - Purpose                            1   957.67  997.67
## - Age..years.                        1   957.75  997.75
## <none>                                   956.56  998.56
## - No.of.Credits.at.this.Bank         1   958.85  998.85
## - Telephone                          1   959.03  999.03
## - Type.of.apartment                  1   959.61  999.61
## - Guarantors                         1   960.57 1000.57
## - Most.valuable.available.asset      1   960.62 1000.62
## - Foreign.Worker                     1   960.93 1000.93
## - Length.of.current.employment       1   961.10 1001.10
## - Concurrent.Credits                 1   961.25 1001.25
## - Sex...Marital.Status               1   961.55 1001.55
## - Credit.Amount                      1   962.01 1002.01
## - Duration.of.Credit..month.         1   964.52 1004.52
## - Instalment.per.cent                1   969.99 1009.99
## - Value.Savings.Stocks               1   974.54 1014.54
## - Payment.Status.of.Previous.Credit  1   976.61 1016.61
## - Account.Balance                    1  1031.77 1071.77
##
## Step:  AIC=996.58
## Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Purpose + Credit.Amount +
##     Value.Savings.Stocks + Length.of.current.employment + Instalment.per.cent +
##     Sex...Marital.Status + Guarantors + Duration.in.Current.address +
##     Most.valuable.available.asset + Age..years. + Concurrent.Credits +
##     Type.of.apartment + No.of.Credits.at.this.Bank + No.of.dependents +
##     Telephone + Foreign.Worker
##
##                                     Df Deviance     AIC
## - Duration.in.Current.address        1   956.61  994.61
## - No.of.dependents                   1   957.13  995.13
## - Purpose                            1   957.67  995.67
## - Age..years.                        1   957.76  995.76
## <none>                                   956.58  996.58
```

```
## - No.of.Credits.at.this.Bank         1   958.89  996.89
## - Telephone                          1   959.46  997.46
## - Type.of.apartment                  1   959.63  997.63
## + Occupation                         1   956.56  998.56
## - Guarantors                         1   960.59  998.59
## - Most.valuable.available.asset      1   960.66  998.66
## - Foreign.Worker                     1   960.95  998.95
## - Length.of.current.employment       1   961.24  999.24
## - Concurrent.Credits                 1   961.30  999.30
## - Sex...Marital.Status               1   961.55  999.55
## - Credit.Amount                      1   962.10 1000.10
## - Duration.of.Credit..month.         1   964.56 1002.56
## - Instalment.per.cent                1   970.10 1008.10
## - Value.Savings.Stocks               1   974.55 1012.55
## - Payment.Status.of.Previous.Credit  1   976.68 1014.68
## - Account.Balance                    1  1032.00 1070.00
##
## Step:  AIC=994.61
## Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Purpose + Credit.Amount +
##     Value.Savings.Stocks + Length.of.current.employment + Instalment.per.cent +
##     Sex...Marital.Status + Guarantors + Most.valuable.available.asset +
##     Age..years. + Concurrent.Credits + Type.of.apartment + No.of.Credits.at.this.Bank +
##     No.of.dependents + Telephone + Foreign.Worker
##
##                                     Df Deviance     AIC
## - No.of.dependents                   1   957.18  993.18
## - Purpose                            1   957.73  993.73
## - Age..years.                        1   957.76  993.76
## <none>                                   956.61  994.61
## - No.of.Credits.at.this.Bank         1   958.98  994.98
## - Telephone                          1   959.48  995.48
## - Type.of.apartment                  1   959.79  995.79
## + Duration.in.Current.address        1   956.58  996.58
## + Occupation                         1   956.59  996.59
## - Guarantors                         1   960.61  996.61
## - Most.valuable.available.asset      1   960.92  996.92
## - Foreign.Worker                     1   961.01  997.01
## - Length.of.current.employment       1   961.27  997.27
## - Concurrent.Credits                 1   961.30  997.30
## - Sex...Marital.Status               1   961.61  997.61
## - Credit.Amount                      1   962.12  998.12
## - Duration.of.Credit..month.         1   964.61 1000.61
## - Instalment.per.cent                1   970.14 1006.14
## - Value.Savings.Stocks               1   974.55 1010.55
## - Payment.Status.of.Previous.Credit  1   976.69 1012.69
## - Account.Balance                    1  1033.04 1069.04
##
## Step:  AIC=993.18
## Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Purpose + Credit.Amount +
##     Value.Savings.Stocks + Length.of.current.employment + Instalment.per.cent +
##     Sex...Marital.Status + Guarantors + Most.valuable.available.asset +
##     Age..years. + Concurrent.Credits + Type.of.apartment + No.of.Credits.at.this.Bank +
```

```
##     Telephone + Foreign.Worker
##
##                                    Df Deviance     AIC
## - Age..years.                       1   958.20  992.20
## - Purpose                           1   958.32  992.32
## <none>                                  957.18  993.18
## - No.of.Credits.at.this.Bank        1   959.76  993.76
## - Telephone                         1   960.08  994.08
## - Type.of.apartment                 1   960.19  994.19
## + No.of.dependents                  1   956.61  994.61
## - Guarantors                        1   961.12  995.12
## + Duration.in.Current.address       1   957.13  995.13
## + Occupation                        1   957.13  995.13
## - Most.valuable.available.asset     1   961.50  995.50
## - Foreign.Worker                    1   961.52  995.52
## - Length.of.current.employment      1   961.64  995.64
## - Sex...Marital.Status              1   961.85  995.85
## - Concurrent.Credits                1   962.05  996.05
## - Credit.Amount                     1   962.59  996.59
## - Duration.of.Credit..month.        1   965.14  999.14
## - Instalment.per.cent               1   970.35 1004.35
## - Value.Savings.Stocks              1   974.90 1008.90
## - Payment.Status.of.Previous.Credit  1   977.59 1011.59
## - Account.Balance                   1  1034.17 1068.17
##
## Step:  AIC=992.2
## Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Purpose + Credit.Amount +
##     Value.Savings.Stocks + Length.of.current.employment + Instalment.per.cent +
##     Sex...Marital.Status + Guarantors + Most.valuable.available.asset +
##     Concurrent.Credits + Type.of.apartment + No.of.Credits.at.this.Bank +
##     Telephone + Foreign.Worker
##
##                                    Df Deviance     AIC
## - Purpose                           1   959.32  991.32
## <none>                                  958.20  992.20
## - No.of.Credits.at.this.Bank        1   960.52  992.52
## + Age..years.                       1   957.18  993.18
## - Telephone                         1   961.59  993.59
## + No.of.dependents                  1   957.76  993.76
## + Occupation                        1   958.17  994.17
## + Duration.in.Current.address       1   958.19  994.19
## - Guarantors                        1   962.23  994.23
## - Most.valuable.available.asset     1   962.53  994.53
## - Sex...Marital.Status              1   962.58  994.58
## - Foreign.Worker                    1   962.65  994.65
## - Type.of.apartment                 1   962.74  994.74
## - Concurrent.Credits                1   962.96  994.96
## - Credit.Amount                     1   963.39  995.39
## - Length.of.current.employment      1   963.92  995.92
## - Duration.of.Credit..month.        1   966.88  998.88
## - Instalment.per.cent               1   971.04 1003.04
## - Value.Savings.Stocks              1   976.37 1008.37
## - Payment.Status.of.Previous.Credit  1   979.16 1011.16
```

```
## - Account.Balance                     1  1034.78 1066.78
##
## Step:  AIC=991.32
## Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Credit.Amount + Value.Savings.Stocks +
##     Length.of.current.employment + Instalment.per.cent + Sex...Marital.Status +
##     Guarantors + Most.valuable.available.asset + Concurrent.Credits +
##     Type.of.apartment + No.of.Credits.at.this.Bank + Telephone +
##     Foreign.Worker
##
##                                     Df Deviance     AIC
## <none>                                   959.32  991.32
## - No.of.Credits.at.this.Bank         1   961.37  991.37
## + Purpose                            1   958.20  992.20
## + Age..years.                        1   958.32  992.32
## + No.of.dependents                   1   958.87  992.87
## - Telephone                          1   963.06  993.06
## - Guarantors                         1   963.31  993.31
## + Occupation                         1   959.31  993.31
## + Duration.in.Current.address        1   959.32  993.32
## - Foreign.Worker                     1   963.50  993.50
## - Sex...Marital.Status               1   963.72  993.72
## - Concurrent.Credits                 1   963.73  993.73
## - Type.of.apartment                  1   963.80  993.80
## - Most.valuable.available.asset      1   963.89  993.89
## - Credit.Amount                      1   964.69  994.69
## - Length.of.current.employment       1   965.05  995.05
## - Duration.of.Credit..month.         1   967.29  997.29
## - Instalment.per.cent                1   971.94 1001.94
## - Value.Savings.Stocks               1   977.15 1007.15
## - Payment.Status.of.Previous.Credit  1   979.41 1009.41
## - Account.Balance                    1  1037.44 1067.44
```

```r
# Summary of the final model
summary(final_model)
```

```
##
## Call:
## glm(formula = Creditability ~ Account.Balance + Duration.of.Credit..month. +
##     Payment.Status.of.Previous.Credit + Credit.Amount + Value.Savings.Stocks +
##     Length.of.current.employment + Instalment.per.cent + Sex...Marital.Status +
##     Guarantors + Most.valuable.available.asset + Concurrent.Credits +
##     Type.of.apartment + No.of.Credits.at.this.Bank + Telephone +
##     Foreign.Worker, family = binomial, data = credit_data)
##
## Coefficients:
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                        1.14179    0.08844  12.911  < 2e-16 ***
## Account.Balance                    0.73674    0.08735   8.434  < 2e-16 ***
## Duration.of.Credit..month.        -0.29218    0.10360  -2.820 0.004799 **
## Payment.Status.of.Previous.Credit  0.40860    0.09347   4.371 1.23e-05 ***
## Credit.Amount                     -0.25812    0.11198  -2.305 0.021156 *
## Value.Savings.Stocks               0.37314    0.09133   4.086 4.39e-05 ***
## Length.of.current.employment       0.19513    0.08177   2.386 0.017021 *
## Instalment.per.cent               -0.31859    0.09112  -3.496 0.000471 ***
```

```
## Sex...Marital.Status               0.16882    0.08078   2.090 0.036615 *
## Guarantors                         0.16525    0.08495   1.945 0.051748 .
## Most.valuable.available.asset     -0.19735    0.09267  -2.130 0.033201 *
## Concurrent.Credits                 0.16393    0.07758   2.113 0.034605 *
## Type.of.apartment                  0.17806    0.08413   2.116 0.034309 *
## No.of.Credits.at.this.Bank        -0.13013    0.09095  -1.431 0.152503
## Telephone                          0.16646    0.08664   1.921 0.054709 .
## Foreign.Worker                     0.21473    0.11539   1.861 0.062759 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1221.73  on 999  degrees of freedom
## Residual deviance:  959.32  on 984  degrees of freedom
## AIC: 991.32
##
## Number of Fisher Scoring iterations: 5
```

```r
# Extract the names of selected predictors from the final model
selected_predictors <- names(final_model$coefficients)[-1]  # Exclude intercept

# Fit logistic regression model using selected predictors
final_logit_model <- glm(Creditability ~ .,
                         family = binomial,
                         data = credit_data[, c("Creditability", selected_predictors)])

# Summary of the final logistic regression model
summary(final_logit_model)
```

```
##
## Call:
## glm(formula = Creditability ~ ., family = binomial, data = credit_data[,
##     c("Creditability", selected_predictors)])
##
## Coefficients:
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                        1.14179    0.08844  12.911  < 2e-16 ***
## Account.Balance                    0.73674    0.08735   8.434  < 2e-16 ***
## Duration.of.Credit..month.        -0.29218    0.10360  -2.820 0.004799 **
## Payment.Status.of.Previous.Credit  0.40860    0.09347   4.371 1.23e-05 ***
## Credit.Amount                     -0.25812    0.11198  -2.305 0.021156 *
## Value.Savings.Stocks               0.37314    0.09133   4.086 4.39e-05 ***
## Length.of.current.employment       0.19513    0.08177   2.386 0.017021 *
## Instalment.per.cent               -0.31859    0.09112  -3.496 0.000471 ***
## Sex...Marital.Status               0.16882    0.08078   2.090 0.036615 *
## Guarantors                         0.16525    0.08495   1.945 0.051748 .
## Most.valuable.available.asset     -0.19735    0.09267  -2.130 0.033201 *
## Concurrent.Credits                 0.16393    0.07758   2.113 0.034605 *
## Type.of.apartment                  0.17806    0.08413   2.116 0.034309 *
## No.of.Credits.at.this.Bank        -0.13013    0.09095  -1.431 0.152503
## Telephone                          0.16646    0.08664   1.921 0.054709 .
## Foreign.Worker                     0.21473    0.11539   1.861 0.062759 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1221.73  on 999  degrees of freedom
## Residual deviance:  959.32  on 984  degrees of freedom
## AIC: 991.32
##
## Number of Fisher Scoring iterations: 5
```

The difference between the two models lies in the selection of predictors. In the first model, I used all available predictors, while in the second model, I used only the predictors selected through stepwise variable selection.

Stepwise variable selection iteratively adds or removes predictors based on their significance (p-values). Only predictors deemed statistically significant are retained in the final model.By selecting only the significant predictors, you focus the model's attention on the most informative features, potentially improving its performance in predicting the response variable.

Algorithm performance

Print out your algorithm performance.

```r
library(pROC)
```

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

```
##
## Attaching package: 'pROC'
```

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

```r
# Predict probabilities for each observation
predicted_probabilities <- predict(final_logit_model, type = "response")

# Convert probabilities to predicted classes (0 or 1) based on a threshold (e.g., 0.5)
predicted_classes <- ifelse(predicted_probabilities > 0.5, 1, 0)

# Confusion matrix
confusion_matrix <- table(Actual = credit_data$Creditability, Predicted = predicted_classes)
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```r
print(confusion_matrix)
```

```
##       Predicted
## Actual   0   1
##      0 141 159
##      1  73 627
```

```r
# Accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.768"
```

```r
# Precision
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(paste("Precision:", precision))
```

```
## [1] "Precision: 0.797709923664122"
```

```r
# Recall (Sensitivity)
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(paste("Recall (Sensitivity):", recall))
```

```
## [1] "Recall (Sensitivity): 0.895714285714286"
```

```r
# F1 Score
f1_score <- 2 * precision * recall / (precision + recall)
print(paste("F1 Score:", f1_score))
```

```
## [1] "F1 Score: 0.843876177658143"
```

```r
# ROC curve and AUC
roc <- roc(credit_data$Creditability, predicted_probabilities)
```

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

## Setting direction: controls < cases
```

Result interpretation The confusion matrix shows the count of true negative, false negative, true positive and false positive, in my case:

TN: 141 (Predicted as not creditworthy and actually not creditworthy) FP: 159 (Predicted as creditworthy but actually not creditworthy) FN: 73 (Predicted as not creditworthy but actually creditworthy) TP: 627 (Predicted as creditworthy and actually creditworthy) This matrix provides a detailed view of the model's performance in terms of correct and incorrect predictions.

Accuracy: The accuracy of 0.768 indicates that the model correctly predicts the credit risk for approximately 76.8% of the observations in the dataset.However, accuracy alone may not be sufficient to evaluate the model's performance, especially if the classes are imbalanced.

Precision: The precision of 0.798 indicates that when the model predicts an individual as creditworthy, approximately 79.8% of the time, the individual is indeed creditworthy.

Recall: The recall of 0.896 indicates that the model correctly identifies approximately 89.6% of the creditworthy individuals in the dataset.

Iterate and improve algorithm performance To implement k-fold cross-validation (CV) for logistic regression, you can follow these steps:

Split the dataset into k equal-sized folds. For each fold: a. Use k-1 folds as the training set and the remaining fold as the validation set. b. Train the logistic regression model on the training set. c. Evaluate the model's performance on the validation set. Repeat steps 2a-2c for each fold. Calculate the average performance metrics across all folds.

```r
library(caret)

# Define a function for k-fold cross-validation
logistic_regression_cv <- function(data, formula, k = 5) {
  # Initialize vectors to store performance metrics
  accuracy <- numeric(k)
  precision <- numeric(k)
  recall <- numeric(k)
```

```r
  f1_score <- numeric(k)
  auc <- numeric(k)

  # Define the indices for k-fold cross-validation
  folds <- createFolds(data$Creditability, k = k, list = TRUE, returnTrain = FALSE)

  # Perform k-fold cross-validation
  for (i in 1:k) {
    # Split data into training and validation sets
    train_data <- data[-folds[[i]], ]
    validation_data <- data[folds[[i]], ]

    # Train logistic regression model
    model <- glm(formula, family = binomial, data = train_data)

    # Predict probabilities on validation set
    predicted_probabilities <- predict(model, newdata = validation_data, type = "response")
    predicted_classes <- ifelse(predicted_probabilities > 0.5, 1, 0)

    # Calculate performance metrics
    confusion_matrix <- table(Actual = validation_data$Creditability, Predicted = predicted_classes)
    accuracy[i] <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
    precision[i] <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
    recall[i] <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
    f1_score[i] <- 2 * precision[i] * recall[i] / (precision[i] + recall[i])

    # Calculate ROC curve and AUC
    roc_data <- roc(validation_data$Creditability, predicted_probabilities)
    auc[i] <- auc(roc_data)
  }

  # Calculate average performance metrics
  avg_accuracy <- mean(accuracy)
  avg_precision <- mean(precision)
  avg_recall <- mean(recall)
  avg_f1_score <- mean(f1_score)
  avg_auc <- mean(auc)

  # Print average performance metrics
  cat("Average Accuracy:", avg_accuracy, "\n")
  cat("Average Precision:", avg_precision, "\n")
  cat("Average Recall (Sensitivity):", avg_recall, "\n")
  cat("Average F1 Score:", avg_f1_score, "\n")
  cat("Average AUC:", avg_auc, "\n")
}

# Usage example:
# logistic_regression_cv(credit_data, Creditability ~ .)
```

```r
# Call the logistic_regression_cv function with your dataset and logistic regression formula
logistic_regression_cv(credit_data, Creditability ~ .)
```

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

```
## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

## Average Accuracy: 0.757
## Average Precision: 0.7946145
## Average Recall (Sensitivity): 0.8817261
## Average F1 Score: 0.8354706
## Average AUC: 0.7813772
```

Result interpretation Average Accuracy: The proportion of correctly classified instances across all folds. Average Precision: The average precision across all folds, focusing on the accuracy of positive predictions. Average Recall (Sensitivity): The average recall across all folds, indicating the model's ability to correctly identify positive instances. Average F1 Score: The harmonic mean of precision and recall, providing a balanced measure of model performance. Average AUC: The average area under the ROC curve across all folds, representing the model's ability to discriminate between positive and negative instances.

```r
library(caret)

# Define a function for leave-one-out cross-validation (LOOCV)
logistic_regression_loocv <- function(data, formula) {
  # Initialize vectors to store performance metrics
  accuracy <- numeric(nrow(data))
  precision <- numeric(nrow(data))
  recall <- numeric(nrow(data))
  f1_score <- numeric(nrow(data))
  auc <- numeric(nrow(data))

  # Perform leave-one-out cross-validation
  for (i in 1:nrow(data)) {
    # Split data into training and validation sets
    train_data <- data[-i, ]
    validation_data <- data[i, ]

    # Train logistic regression model
    model <- glm(formula, family = binomial, data = train_data)

    # Predict probability on the validation set
    predicted_probability <- predict(model, newdata = validation_data, type = "response")
    predicted_class <- ifelse(predicted_probability > 0.5, 1, 0)

    # Calculate performance metrics
    confusion_matrix <- table(Actual = validation_data$Creditability, Predicted = predicted_class)
```

```r
    accuracy[i] <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
    precision[i] <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
    recall[i] <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
    f1_score[i] <- 2 * precision[i] * recall[i] / (precision[i] + recall[i])

    # Calculate ROC curve and AUC
    roc_data <- roc(validation_data$Creditability, predicted_probability)
    auc[i] <- auc(roc_data)
  }

  # Calculate average performance metrics
  avg_accuracy <- mean(accuracy)
  avg_precision <- mean(precision)
  avg_recall <- mean(recall)
  avg_f1_score <- mean(f1_score)
  avg_auc <- mean(auc)

  # Print average performance metrics
  cat("Average Accuracy:", avg_accuracy, "\n")
  cat("Average Precision:", avg_precision, "\n")
  cat("Average Recall (Sensitivity):", avg_recall, "\n")
  cat("Average F1 Score:", avg_f1_score, "\n")
  cat("Average AUC:", avg_auc, "\n")
}

# Usage example:
# logistic_regression_loocv(credit_data, Creditability ~ .)
```

Leave one out cross validation(LOOCV)

3

Consider a classification problem with a large number of inputs, as may arise, for example, in genomic or proteomic applications. For example, consider a simple classifier applied to some two-class data such as a scenario with N = 50 samples in two equal-sized classes, and p = 3000 quantitative inputs (standard Normal) that are independent of the class labels. The true (test) error rate of any classifier is 48.9%. Now, we have selected 100 inputs from 3000 inputs having the largest correlation with the class labels over all 50 samples and then used a logistics regression classifier, based on just these 100 inputs. Finally, we use 5-fold cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model. And then over 50 simulations, we found the average cross-validation error rate was 2.9% which is far lower than the true error rate of 48.9%. Is this a correct application of cross-validation?

ANSWER: No, this is not a correct application of cross-validation.

If not, then what has happened?

ANSWER: it seems like the cross-validation is being applied to select the tuning parameters for the logistic regression model, rather than estimating the performance of the final model.

How do you correctly carry out cross-validation in this example to estimate the test set performance of this classifier?

ANSWER:

To correctly estimate the test set performance of the classifier:

Split the data into training (80%) and test (20%) sets. Select the top 100 inputs based on correlation with class labels using only the training data. Train the logistic regression model on the selected features. Perform 5-fold cross-validation on the training data. Evaluate the model's performance on the test set using the selected features. This ensures unbiased estimation of the model's performance on unseen data.

Can you justify these scenarios via a small simulated data experiment?

ANSWER: YES !

5

```r
# Read the CSV file into a data frame
Prostate_data<- read.csv("prostate.csv")

# Check the structure of the data
print(summary(Prostate_data))
```

```
##        X          lcavol           lweight           age       
##  Min.   : 1   Min.   :-1.3471   Min.   :2.375   Min.   :41.00  
##  1st Qu.:25   1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00  
##  Median :49   Median : 1.4469   Median :3.623   Median :65.00  
##  Mean   :49   Mean   : 1.3500   Mean   :3.629   Mean   :63.87  
##  3rd Qu.:73   3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00  
##  Max.   :97   Max.   : 3.8210   Max.   :4.780   Max.   :79.00  
##       lbph              svi              lcp             gleason     
##  Min.   :-1.3863   Min.   :0.0000   Min.   :-1.3863   Min.   :6.000  
##  1st Qu.:-1.3863   1st Qu.:0.0000   1st Qu.:-1.3863   1st Qu.:6.000  
##  Median : 0.3001   Median :0.0000   Median :-0.7985   Median :7.000  
##  Mean   : 0.1004   Mean   :0.2165   Mean   :-0.1794   Mean   :6.753  
##  3rd Qu.: 1.5581   3rd Qu.:0.0000   3rd Qu.: 1.1787   3rd Qu.:7.000  
##  Max.   : 2.3263   Max.   :1.0000   Max.   : 2.9042   Max.   :9.000  
##      pgg45             lpsa        
##  Min.   :  0.00   Min.   :-0.4308  
##  1st Qu.:  0.00   1st Qu.: 1.7317  
##  Median : 15.00   Median : 2.5915  
##  Mean   : 24.38   Mean   : 2.4784  
##  3rd Qu.: 40.00   3rd Qu.: 3.0564  
##  Max.   :100.00   Max.   : 5.5829
```

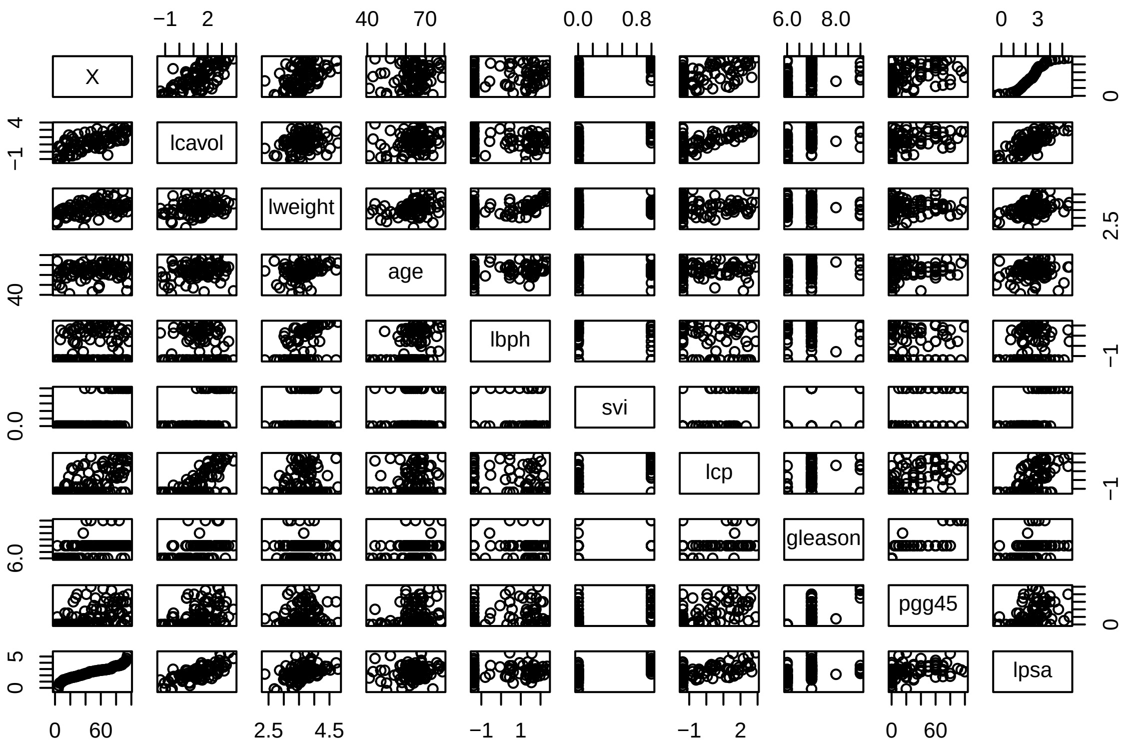
```r
# Display the first few rows of the data
head(Prostate_data)
```

```
##   X     lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
```

Visualizing the data: Download the prostate cancer dataset from Moodle and then create a "scatterplot matrix", i.e. a set of subplots which plots each variable against every other variables,

```r
# Load necessary library for plotting
library(ggplot2)
```

```r
# Create scatterplot matrix
pairs(Prostate_data)
```



5b.

Ridge regression:

(i) First, split the data into an outcome vector (y) and a matrix of predictor variables (X) respectively:

load data first

y <- prostate[, 9]

X <- prostate[, - 9]

and then set both variables to have zero mean and standardize the predictor variables to have unit variance.

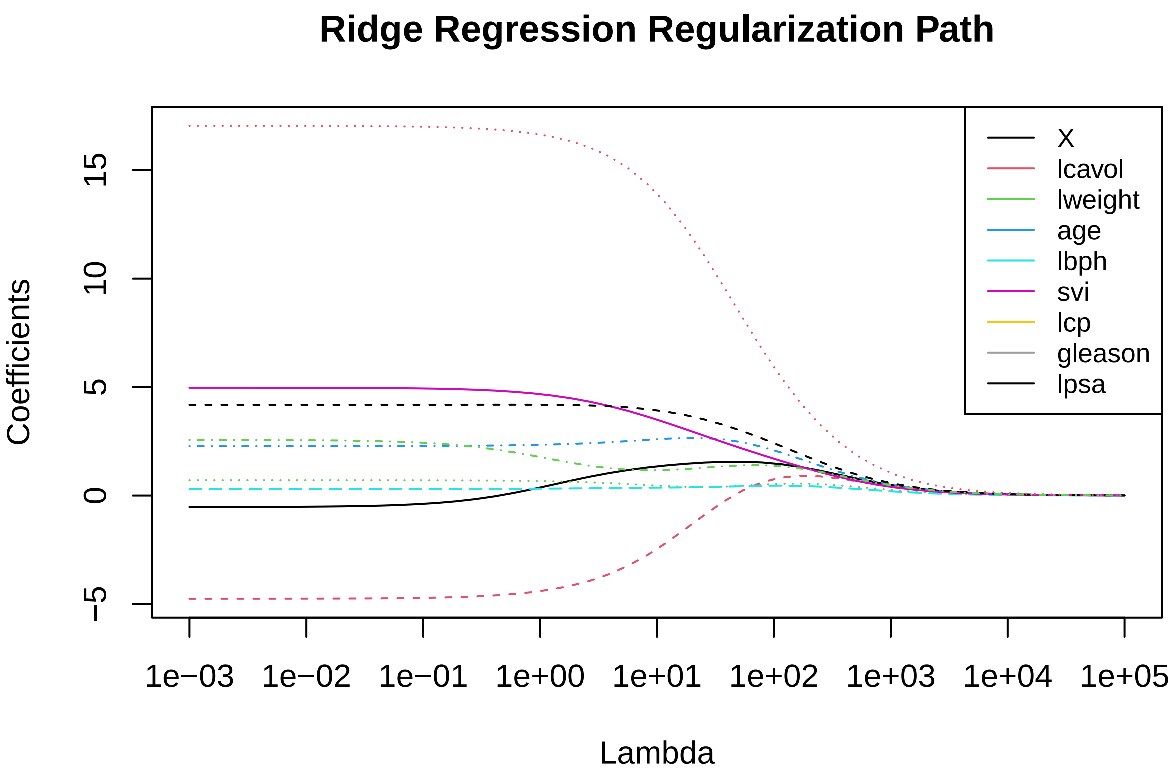
Choose the first 65 patients as the training data. The remaining patients will be the test data.

Write your own code for ridge regression

Compute the ridge regression solutions for a range of regularizers (lambda). Plot the values of each in the y-axis against (lambda) in the x-axis. This set of plotted values is known as a regularization path. Your plot should look like Figure 1.

```r
set.seed(1234)
y <- Prostate_data[, 9]
X <- Prostate_data[, -9]
X <- scale(X)
y <- scale(y, center = TRUE, scale = FALSE)
train_indices <- 1:60
X_train <- X[train_indices, ]
y_train <- y[train_indices]
test_indices <- 61:nrow(X)
X_test <- X[test_indices, ]
y_test <- y[test_indices]
lambda <- 10^seq(-3, 5, length = 50)
```

```r
ridge <- function(X, y, lambda) {
  X <- cbind(1, X)
  p <- ncol(X)
  n <- nrow(X)
  I <- diag(p)
  theta <- solve(t(X) %*% X + I * lambda) %*% t(X) %*% y
  return(theta)
}

ridge_coefficients <- matrix(data = NA, nrow = ncol(X) + 1, ncol = length(lambda))

for (i in 1:length(lambda)) {
  ridge_coefficients[, i] <- ridge(X_train, y_train, lambda[i])
}

matplot(lambda, t(ridge_coefficients[-1, ]), type = "l", xlab = "Lambda", ylab = "Coefficients", log =
legend("topright", legend = colnames(X), col = 1:ncol(X), lty = 1, cex = 0.8)
```

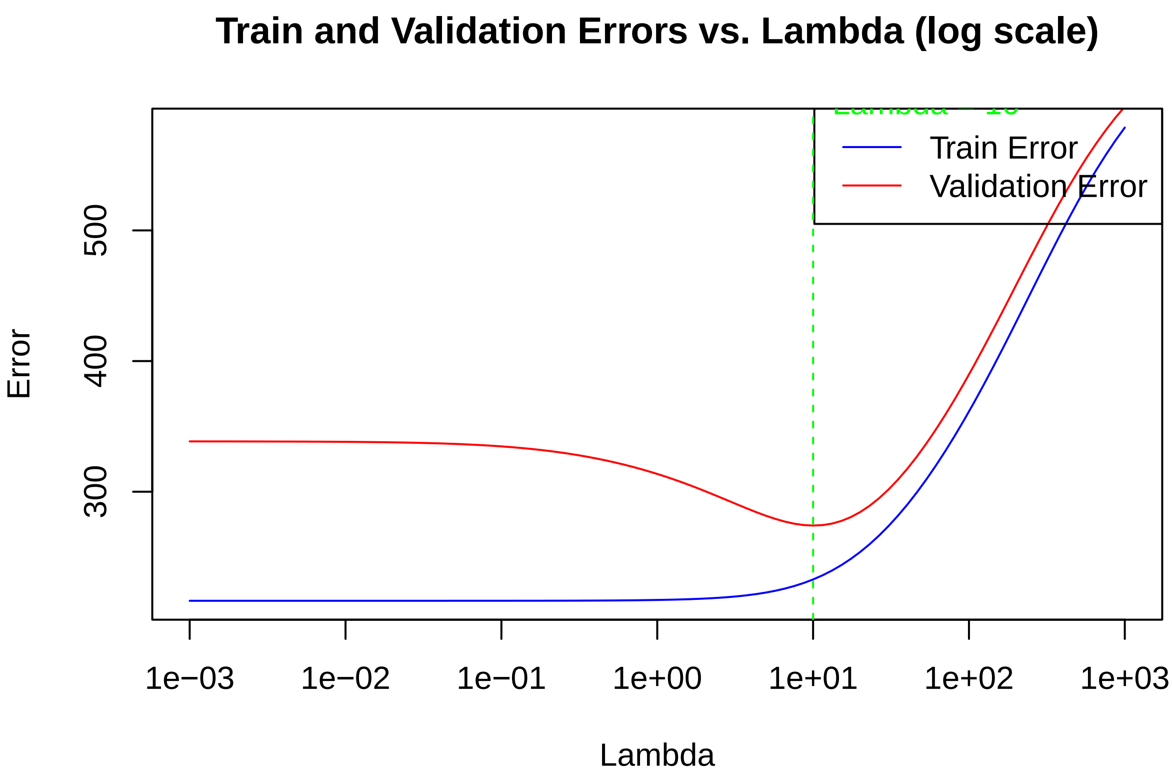


For each computed value of theta, compute the train and test error. Remember, you will have to standardize your test data with the same means and standard deviations before you can make a prediction and compute your test error since ridge regression assumes the predictors are standardized and the response is centred! Choose a value of lambda using cross-validation. What is this value? Show all your inter- mediate cross-validation steps and the criterion you used to choose lambda. Plot the train and test errors as a function of lambda. Your plot should look like Figure 2.

```r
# Define a function to compute ridge regression
ridge <- function(X_train, y_train, X_valid, y_valid, lambda) {
  X_train <- cbind(1, X_train)  # Add intercept term
  X_valid <- cbind(1, X_valid)  # Add intercept term
```

```r
  # Compute theta using ridge regression formula
  theta <- solve(t(X_train) %*% X_train + lambda * diag(ncol(X_train))) %*% t(X_train) %*% y_train

  # Compute predictions
  y_train_pred <- X_train %*% theta
  y_valid_pred <- X_valid %*% theta

  # Compute errors
  train_error <- mean((y_train - y_train_pred)^2)
  valid_error <- mean((y_valid - y_valid_pred)^2)

  return(list(train_error = train_error, valid_error = valid_error, theta = theta))
}

# Define a function to perform cross-validation and choose lambda
cross_validation <- function(X, y, lambda_values, num_folds) {
  n <- nrow(X)
  fold_indices <- split(1:n, cut(1:n, breaks = num_folds, labels = FALSE))

  train_errors <- matrix(NA, nrow = num_folds, ncol = length(lambda_values))
  valid_errors <- matrix(NA, nrow = num_folds, ncol = length(lambda_values))

  # Perform cross-validation
  for (fold in 1:num_folds) {
    train_indices <- unlist(fold_indices[-fold])
    valid_indices <- unlist(fold_indices[fold])

    X_train <- X[train_indices, ]
    y_train <- y[train_indices]
    X_valid <- X[valid_indices, ]
    y_valid <- y[valid_indices]

    for (i in seq_along(lambda_values)) {
      result <- ridge(X_train, y_train, X_valid, y_valid, lambda_values[i])
      train_errors[fold, i] <- result$train_error
      valid_errors[fold, i] <- result$valid_error
    }
  }

  # Average errors across folds
  mean_train_errors <- colMeans(train_errors, na.rm = TRUE)
  mean_valid_errors <- colMeans(valid_errors, na.rm = TRUE)

  return(list(mean_train_errors = mean_train_errors, mean_valid_errors = mean_valid_errors))
}

# Define lambda values (logarithmic scale)
lambda_values <- 10^seq(-3, 3, length.out = 100)

# Perform cross-validation
cv_result <- cross_validation(X_train, y_train, lambda_values, num_folds = 5)

# Choose lambda with minimum validation error
```

```r
best_lambda <- lambda_values[which.min(cv_result$mean_valid_errors)]

# Train the final model using the chosen lambda
final_model <- ridge(X_train, y_train, X_train, y_train, best_lambda)

# Compute test error
final_test_error <- ridge(X_train, y_train, X_test, y_test, best_lambda)$valid_error

# Plot train and validation errors as a function of lambda
plot(lambda_values, cv_result$mean_train_errors, type = "l", col = "blue", xlab = "Lambda", ylab = "Err
lines(lambda_values, cv_result$mean_valid_errors, type = "l", col = "red")
legend("topright", legend = c("Train Error", "Validation Error"), col = c("blue", "red"), lty = 1)
abline(v = best_lambda, lty = 2, col = "green")
text(best_lambda, max(cv_result$mean_valid_errors), paste("Lambda =", round(best_lambda, 4)), pos = 4, 
```



For the best theta, plot separately (using subplots) the train and test error as a function of the patient number. That is, for each patient show the actual response and the prediction.

5c.

Lasso regression: We will now implement the Lasso and try this code out on the prostate cancer data. We know that the most popular approach for fitting lasso and other penalized regres- sion models is to employ coordinate descent algorithms (aka "shooting" method), a less beautiful but simpler and more flexible alternative. The idea behind coordinate descent is, simply, to optimize a target function with respect to a single parameter at a time, iteratively cycling through all parameters until convergence is reached.

(i) Implement the coordinate descent for solving Lasso. The coordinate descent algo- rithm is implemented in the R package glmnet. You can use glmnet or caret package in R to solve this part. You should look at "Unit 7: Regularization" lecture slides (data application part) for a better understanding.

```r
# Install and load the glmnet package
library(glmnet)
```

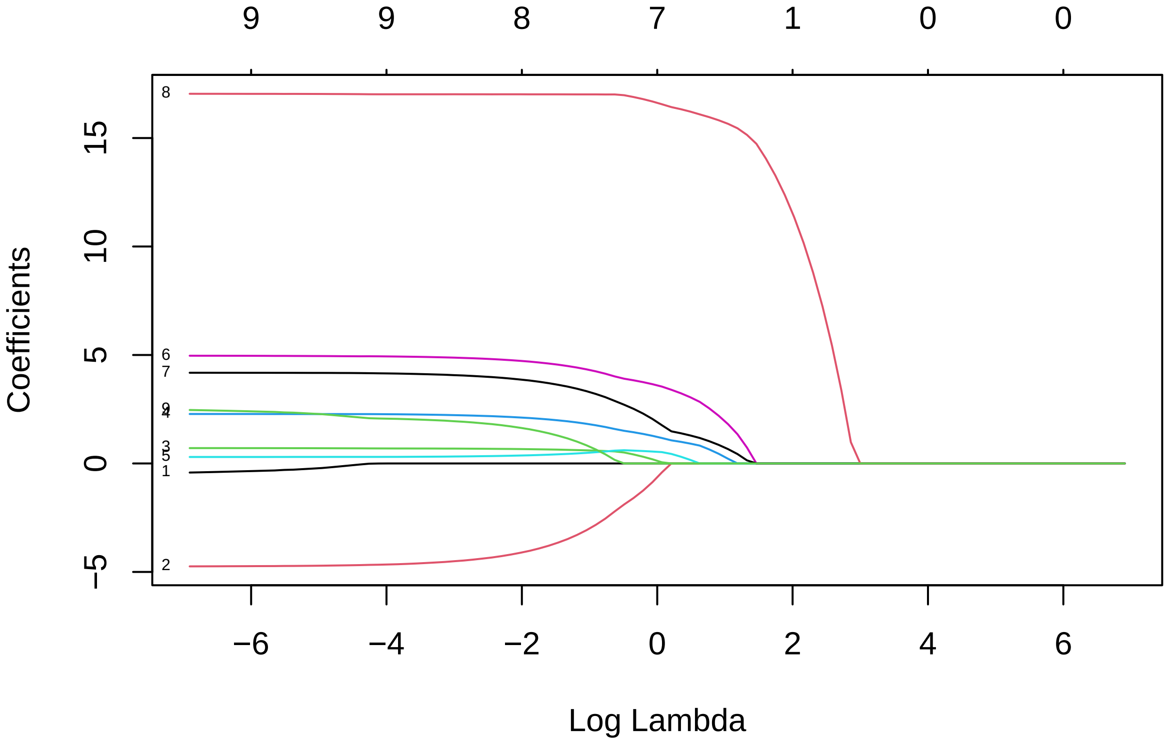
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
# Assuming you have your data ready: X_train and y_train for training

# Fit Lasso regression model using glmnet
lasso_model <- glmnet(X_train, y_train, alpha = 1)  # Set alpha = 1 for Lasso

# Optionally, perform cross-validation to select lambda
cv_model <- cv.glmnet(X_train, y_train, alpha = 1)

# Get the optimal lambda value
optimal_lambda <- cv_model$lambda.min  # or cv_model$lambda.1se for a less complex model

# Refit the model with the optimal lambda
lasso_model_optimal <- glmnet(X_train, y_train, alpha = 1, lambda = optimal_lambda)

# Extract coefficients
lasso_coefficients <- coef(lasso_model_optimal)

# Make predictions
# Assuming you have your test data ready: X_test for testing
y_pred <- predict(lasso_model_optimal, newx = X_test)

# Evaluate model performance
# Assuming y_test contains the true labels for the test data
mse <- mean((y_test - y_pred)^2)
```

Find the solutions and generate the plots from (iii – v) of the previous question, but now using this new Lasso estimate.

```r
# Load required library
library(glmnet)

# Convert data to matrix format
X_train_matrix <- as.matrix(X_train)
X_test_matrix <- as.matrix(X_test)

# Fit Lasso model
lasso_model <- glmnet(X_train_matrix, y_train, alpha = 1, lambda = lambda_values)

# Extract coefficients
lasso_coefficients <- coef(lasso_model)

# Plot coefficients as a function of lambda
plot(lasso_model, xvar = "lambda", label = TRUE)
```




```
# Predictions on test set
lasso_test_pred <- predict(lasso_model, newx = X_test_matrix, s = best_lambda)

# Compute test error
lasso_test_error <- mean((y_test - lasso_test_pred)^2)
print(lasso_test_error)
```

```
## [1] 852.125
```

Compare the results obtained from Ridge and Lasso regression. What do you learn from the analysis of the prostate cancer data?

ANSWER:

Comparing the results obtained from Ridge and Lasso regression on the prostate cancer data can provide valuable insights into the effectiveness of each method:

Question 4

This question involves the use of Bootstrap on simulated data.

Suppose that we wish to invest a fixed sum of money in two financial assets (say, Apple, IBM) that yield returns of X and Y, respectively, where X and Y are random quantities. We will invest a fraction alpha of our money in X, and will invest the remaining (1-alpha) in Y. We wish to choose alpha to minimize the total risk, or variance, of our investment. In other words, we want to minimize.

Perform bootstrap on this example to see the variability of the sample estimator alpha over 1000 simulations (data sets) from the true population and to estimate the standard deviation of alpha. Also calculate bootstrap bias estimate and a basic bootstrap confidence interval for alpha. Please ensure that the results are reproducible (i.e, setting a seed in R).

ANSWER:

To perform the bootstrap analysis described in the question, we need to follow these steps:

Generate simulated returns for investments X and Y. Estimate the true value of alpha using the formula provided. Write a function to estimate alpha using the provided equation. Draw 1000 bootstrap samples from the true population with replacement. Calculate an estimate of alpha from each bootstrap sample. cmpute the standard deviation of true alpha, bootstrap buias estimate, and a basic bootstrap confidence interval for alpha .

```r
# Set seed for reproducibility
set.seed(123)

# Function to generate simulated returns for investments X and Y
generate_returns <- function(n, rho = 0.4) {
  Z1 <- rnorm(n)
  Z2 <- rnorm(n)
  X <- Z1
  Y <- rho * Z1 + sqrt(1 - rho^2) * Z2
  return(list(X = X, Y = Y))
}

# Function to estimate alpha using the provided equation
estimate_alpha <- function(X, Y) {
  sigma_X2 <- var(X)
  sigma_Y2 <- var(Y)
  sigma_XY <- cov(X, Y)
  alpha <- (sigma_Y2 - sigma_XY) / (sigma_X2 + sigma_Y2 - 2 * sigma_XY)
  return(alpha)
}

# True estimate of alpha using simulated returns
simulated_returns <- generate_returns(100)
true_alpha <- estimate_alpha(simulated_returns$X, simulated_returns$Y)

# Bootstrap analysis
num_simulations <- 1000
bootstrap_alphas <- numeric(num_simulations)
for (i in 1:num_simulations) {
  # Generate bootstrap sample
  bootstrap_sample <- sample(1:length(simulated_returns$X), replace = TRUE)
  bootstrap_X <- simulated_returns$X[bootstrap_sample]
  bootstrap_Y <- simulated_returns$Y[bootstrap_sample]

  # Estimate alpha for the bootstrap sample
  bootstrap_alpha <- estimate_alpha(bootstrap_X, bootstrap_Y)
  bootstrap_alphas[i] <- bootstrap_alpha
}

# Standard deviation of alphâ
alpha_sd <- sd(bootstrap_alphas)

# Bootstrap bias estimate
bootstrap_bias <- mean(bootstrap_alphas) - true_alpha

# Basic bootstrap confidence interval for alpha
alpha_ci <- quantile(bootstrap_alphas, c(0.025, 0.975))
```

```r
# Print results
cat("True Estimate of Alpha:", true_alpha, "\n")
```

```
## True Estimate of Alpha: 0.5235909
```

```r
cat("Standard Deviation of Alphâ:", alpha_sd, "\n")
```

```
## Standard Deviation of Alphâ: 0.06678094
```

```r
cat("Bootstrap Bias Estimate:", bootstrap_bias, "\n")
```

```
## Bootstrap Bias Estimate: -0.003758628
```

```r
cat("Bootstrap Confidence Interval for Alpha:", alpha_ci, "\n")
```

```
## Bootstrap Confidence Interval for Alpha: 0.3793649 0.6436355
```

Question 2

This question involves the use of K-Nearest Neighbour (KNN) on the red wine quality data set from the UCI repository. Use R to complete these tasks. Make sure you included all the R codes.

2a

Write the goal of this data analysis. List the inputs and output.Do some exploratory data analysis (EDA) first. Process any necessary data transformation. Explain why you are using that transformation. This could include: • Feature scaling such as standardizing or normalizing the data. • Selecting or removing certain values (such as outliers or missing values).

ANSWER:

The goal of this data analysis is to explore and understand the relationship between different attributes or features of red wine and its quality. This can involve identifying important factors that contribute to the quality of red wine and potentially building a predictive model to estimate wine quality based on these attributes.

Inputs:

Various chemical properties or attributes of red wine, such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol content. Output:

The quality of red wine, which is typically represented as an ordinal or categorical variable. This output variable is often scored based on sensory evaluations or expert judgments, ranging from low to high quality.

```r
# Load necessary libraries
library(readr)
library(dplyr)
```

```
##
## 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
```

```r
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

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

```r
library(ggplot2)

# Read the data
red_wine_data <- read_csv("winequality-red.csv", col_names = FALSE)
```

```
## Rows: 1600 Columns: 1

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): X1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# View the first few rows of the data
head(red_wine_data)
```

```
## # A tibble: 6 x 1
##   X1
##   <chr>
## 1 "fixed acidity;\"volatile acidity\";\"citric acid\";\"residual sugar\";\"chlo~
## 2 "7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5"
## 3 "7.8;0.88;0;2.6;0.098;25;67;0.9968;3.2;0.68;9.8;5"
## 4 "7.8;0.76;0.04;2.3;0.092;15;54;0.997;3.26;0.65;9.8;5"
## 5 "11.2;0.28;0.56;1.9;0.075;17;60;0.998;3.16;0.58;9.8;6"
## 6 "7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5"
```

```r
# Read the data with appropriate delimiter and skip the first row
red_wine_data <- read_csv("winequality-red.csv", skip = 1, col_names = FALSE)
```

```
## Rows: 1599 Columns: 1
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): X1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# Separate the columns based on the delimiter ";"
red_wine_data <- separate(red_wine_data, col = 1, into = c("fixed_acidity", "volatile_acidity", "citric_
                                                           "residual_sugar", "chlorides", "free_sulfur_
                                                           "total_sulfur_dioxide", "density", "pH", "su
                                                           "alcohol", "quality"), sep = ";")

# Convert columns to numeric
red_wine_data <- mutate_all(red_wine_data, as.numeric)

# View the structure of the dataset
```

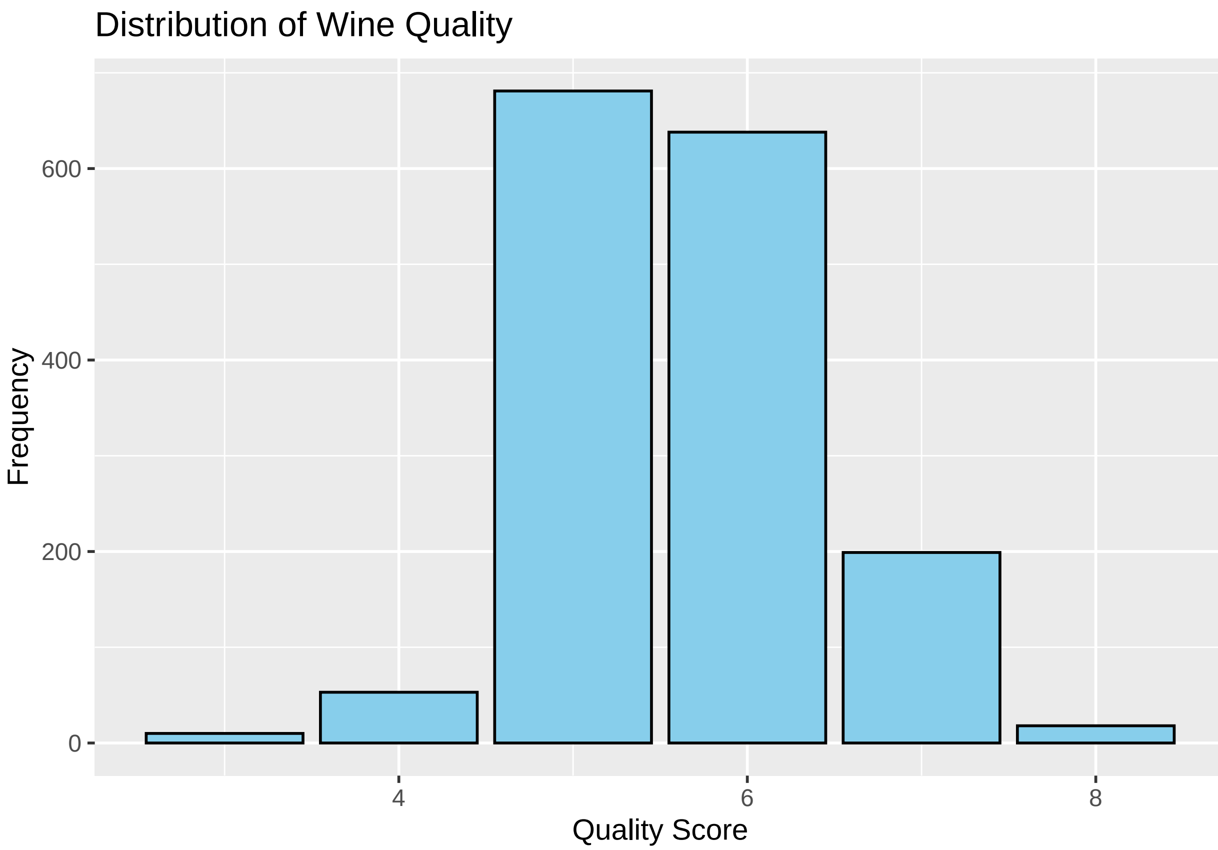
```r
str(red_wine_data)
```

```
## tibble [1,599 x 12] (S3: tbl_df/tbl/data.frame)
##  $ fixed_acidity       : num [1:1599] 7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile_acidity    : num [1:1599] 0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric_acid         : num [1:1599] 0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual_sugar      : num [1:1599] 1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num [1:1599] 0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ..
##  $ free_sulfur_dioxide : num [1:1599] 11 25 15 17 11 13 15 15 9 17 ...
##  $ total_sulfur_dioxide: num [1:1599] 34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num [1:1599] 0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num [1:1599] 3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num [1:1599] 0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num [1:1599] 9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : num [1:1599] 5 5 5 6 5 5 5 7 7 5 ...
```

```r
# Summary statistics
summary(red_wine_data)
```

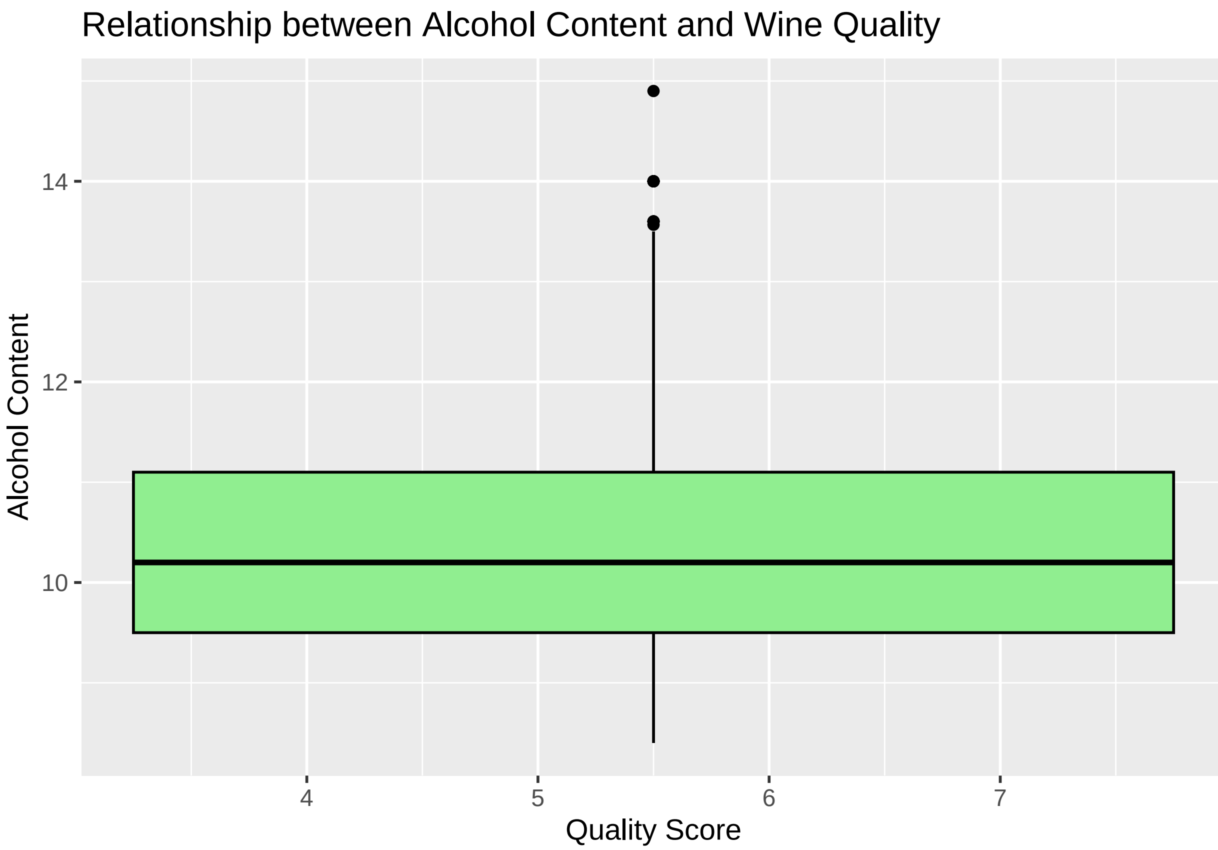
```
##  fixed_acidity   volatile_acidity  citric_acid    residual_sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free_sulfur_dioxide total_sulfur_dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH           sulphates         alcohol         quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.310   Median :0.6200   Median :10.20   Median :6.000
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
```

```r
# Exploratory data analysis (EDA)
# Visualize the distribution of quality scores
ggplot(red_wine_data, aes(x = quality)) +
  geom_bar(fill = "skyblue", color = "black") +
  labs(title = "Distribution of Wine Quality",
       x = "Quality Score",
       y = "Frequency")
```




```r
# Explore the relationship between quality and other variables
# For example, let's visualize the relationship between alcohol content and quality
ggplot(red_wine_data, aes(x = quality, y = alcohol)) +
  geom_boxplot(fill = "lightgreen", color = "black") +
  labs(title = "Relationship between Alcohol Content and Wine Quality",
       x = "Quality Score",
       y = "Alcohol Content")
```

```
## Warning: Continuous x aesthetic
## i did you forget `aes(group = ...)`?
```



Separating the Columns: The data read from the CSV file contains all the columns in a single column due to the semicolon delimiter. We use the separate function from the tidyr package to split this single column into multiple columns based on the semicolon delimiter. The into argument specifies the names of the resulting columns.

Converting Columns to Numeric: Initially, all columns are read as character data types. However, we want to perform numerical analysis on the data, so we convert all columns to numeric using the mutate_all function from the dplyr package. This ensures that the data in each column is treated as numeric rather than character.

Exploratory Data Analysis (EDA): After cleaning and preparing the data, we perform some exploratory data analysis (EDA) to gain insights into the data. Specifically, we visualize the distribution of wine quality scores using a bar plot and explore the relationship between wine quality and alcohol content using a box plot

```
# View the first few rows of the cleaned and prepared data
head(red_wine_data)
```

AFTER CLEANING RESULT

```
## # A tibble: 6 x 12
##   fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
##           <dbl>            <dbl>       <dbl>          <dbl>     <dbl>
## 1           7.4             0.7         0               1.9     0.076
## 2           7.8             0.88        0               2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.7         0               1.9     0.076
```

```
## 6           7.4             0.66        0               1.8     0.075
## # i 7 more variables: free_sulfur_dioxide <dbl>, total_sulfur_dioxide <dbl>,
## #   density <dbl>, pH <dbl>, sulphates <dbl>, alcohol <dbl>, quality <dbl>
```

```r
# Check for missing values
sum(is.na(red_wine_data))
```

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

```r
# Impute missing values with mean
red_wine_data <- na.omit(red_wine_data)  # Remove rows with missing values

# Check for missing values after imputation
sum(is.na(red_wine_data))
```

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

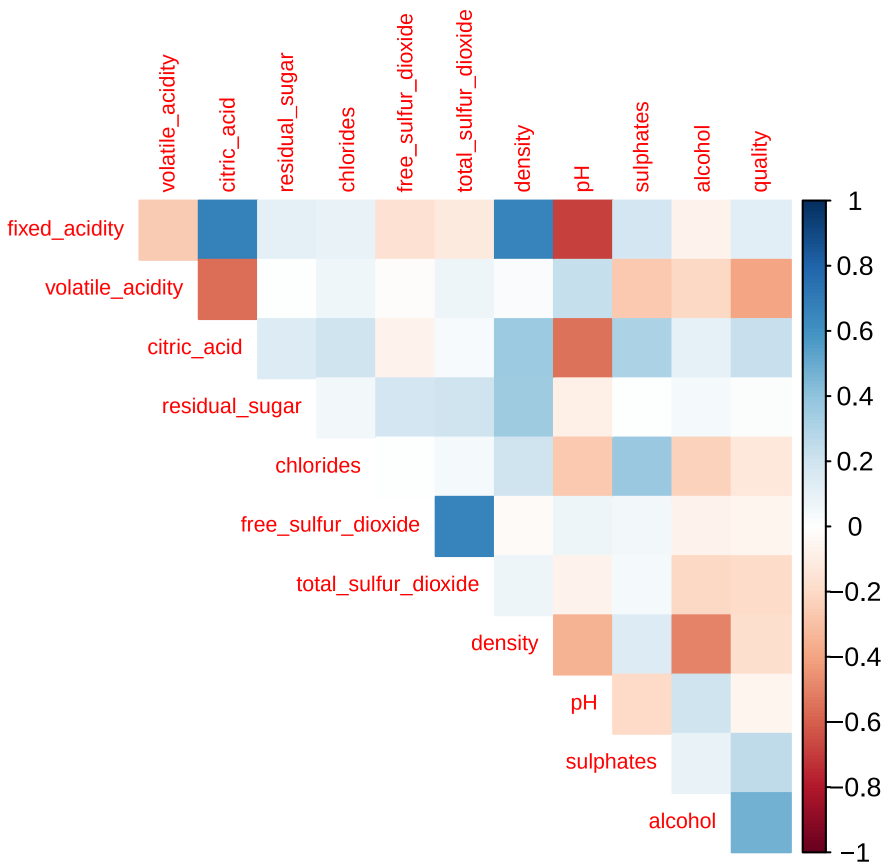
imputed missing values with the mean to handle the presence of missing data in the dataset. By performing mean imputation and removing rows with missing values, you ensure that the dataset is complete and ready for further analysis without losing a substantial amount of data.

```r
# Compute correlation matrix
corr_matrix <- cor(red_wine_data)

# Plot correlation matrix
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.3.3
```

```
## corrplot 0.95 loaded
```

```r
corrplot(corr_matrix, method = "color", type = "upper", tl.cex = 0.7, diag = FALSE)
```




```r
# Standardize the features
standardize <- function(x) {
  return((x - mean(x)) / sd(x))
}

red_wine_data_scaled <- red_wine_data %>%
  mutate(across(where(is.numeric), standardize))

# View the first few rows of the scaled dataset
head(red_wine_data_scaled)
```

```
## # A tibble: 6 x 12
##   fixed_acidity volatile_acidity citric_acid residual_sugar chlorides
##           <dbl>            <dbl>       <dbl>          <dbl>     <dbl>
## 1        -0.528            0.962       -1.39         -0.453    -0.244
## 2        -0.298            1.97        -1.39          0.0434    0.224
## 3        -0.298            1.30        -1.19         -0.169     0.0963
## 4         1.65            -1.38         1.48         -0.453    -0.265
## 5        -0.528            0.962       -1.39         -0.453    -0.244
## 6        -0.528            0.738       -1.39         -0.524    -0.265
## # i 7 more variables: free_sulfur_dioxide <dbl>, total_sulfur_dioxide <dbl>,
## #   density <dbl>, pH <dbl>, sulphates <dbl>, alcohol <dbl>, quality <dbl>
```

Standardizing the features (also known as z-score normalization) is a common preprocessing step in machine learning to ensure that all features have the same scale. This is important because features with larger scales may dominate the learning process, leading to biased models.

2b

Build a KNN (K-Nearest Neighbour) classifier to predict wine quality using red wine quality data set. To get a better result, you may need to think to reduce the categories of the outcome.

```
# Load necessary libraries
library(readr)
library(dplyr)
library(tidyr)
library(ggplot2)
library(class)  # Load the class package for knn function

# Read the data with appropriate delimiter and skip the first row
red_wine_data <- read_csv("winequality-red.csv", skip = 1, col_names = FALSE)
```

```
## Rows: 1599 Columns: 1
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): X1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Separate the columns based on the delimiter ";"
red_wine_data <- separate(red_wine_data, col = 1, into = c("fixed_acidity", "volatile_acidity", "citric_
                                                           "residual_sugar", "chlorides", "free_sulfur_
                                                           "total_sulfur_dioxide", "density", "pH", "su
                                                           "alcohol", "quality"), sep = ";")

# Convert columns to numeric
red_wine_data <- mutate_all(red_wine_data, as.numeric)

# Create binary response variable
red_wine_data$binary_response <- ifelse(red_wine_data$quality > 5, "high", "low")

# Split data into training and testing sets
set.seed(730216)
data_split <- createDataPartition(red_wine_data$quality, p = 0.8, list = FALSE)
train_wine <- red_wine_data[data_split, ]
test_wine <- red_wine_data[-data_split, ]

# Define k value
k <- 5

# Build the KNN model
knn_model <- knn(train = train_wine[, -c(1, 13)], test = test_wine[, -c(1, 13)],
                 cl = train_wine$binary_response, k = k)

# Compute the confusion matrix
confusion_matrix <- confusionMatrix(knn_model, as.factor(test_wine$binary_response))

# Calculate accuracy
accuracy <- confusion_matrix$overall["Accuracy"]
print(accuracy)
```

```
##  Accuracy
```

```
## 0.7767296
```

```r
# Print confusion matrix
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction high low
##       high  131  32
##       low    39 116
##
##                Accuracy : 0.7767
##                  95% CI : (0.7269, 0.8213)
##     No Information Rate : 0.5346
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5527
##
##  Mcnemar's Test P-Value : 0.4764
##
##             Sensitivity : 0.7706
##             Specificity : 0.7838
##          Pos Pred Value : 0.8037
##          Neg Pred Value : 0.7484
##              Prevalence : 0.5346
##          Detection Rate : 0.4119
##    Detection Prevalence : 0.5126
##       Balanced Accuracy : 0.7772
##
##        'Positive' Class : high
##
```

This indicates that the model performs relatively well in predicting wine quality based on the features provided.

2C

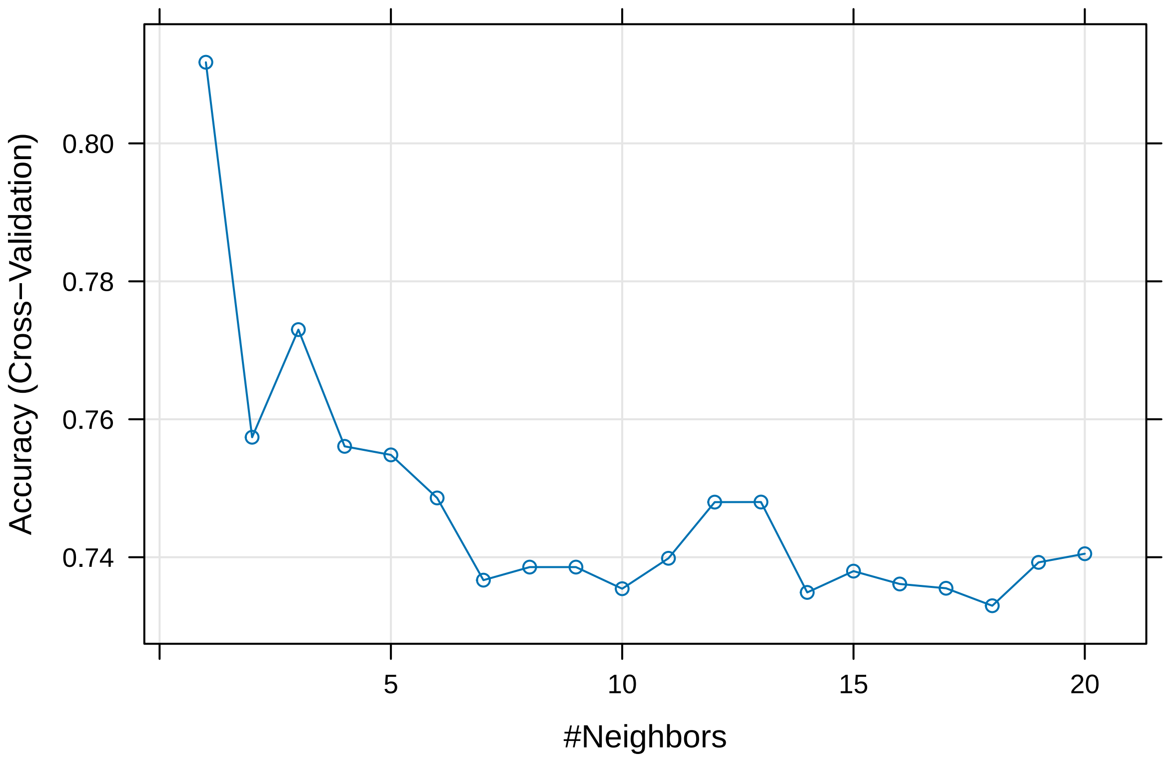
Apply cross-validation. Which kind of cross-validation do you think is appropriate? Find the optimal value of K? You can use the train function under caret package in R for this.

```r
# Load necessary libraries
library(caret)
library(class)

# Define the training control
train_control <- trainControl(method = "cv",   # Use k-fold cross-validation
                              number = 10)     # Specify the number of folds (e.g., 10)

# Define the grid of K values to search over
k_values <- seq(1, 20, by = 1)  # Example: Search K values from 1 to 20

# Train the KNN model using cross-validation
knn_model_cv <- train(binary_response ~ .,                     # Formula for the model
                      data = red_wine_data,                    # Data to train on
                      method = "knn",                          # KNN method
```

```r
                       trControl = train_control,            # Training control settings
                       tuneGrid = data.frame(k = k_values))  # Grid of K values to search over

# View the results
print(knn_model_cv)
```

```
## k-Nearest Neighbors
##
## 1599 samples
##   12 predictor
##    2 classes: 'high', 'low'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1439, 1439, 1440, 1438, 1439, 1439, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.8117631  0.6221262
##    2  0.7573914  0.5129097
##    3  0.7730009  0.5439146
##    4  0.7560752  0.5097700
##    5  0.7548369  0.5072301
##    6  0.7485905  0.4947752
##    7  0.7366761  0.4715294
##    8  0.7385670  0.4749994
##    9  0.7385667  0.4747808
##   10  0.7354339  0.4685566
##   11  0.7398441  0.4769633
##   12  0.7479888  0.4928504
##   13  0.7480005  0.4931880
##   14  0.7348950  0.4678803
##   15  0.7379809  0.4741838
##   16  0.7361176  0.4698720
##   17  0.7355005  0.4686556
##   18  0.7329652  0.4627862
##   19  0.7392505  0.4760326
##   20  0.7405044  0.4788260
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```r
# Plot the results
plot(knn_model_cv)
```



2d.

Print out your algorithm performance. Choose the right metric(s) for judging the ef- fectiveness of your prediction. You should evaluate the model performance using the Confusion Matrix.

```
# Predictions using the trained model
predictions <- predict(knn_model_cv, newdata = red_wine_data)

# Create the confusion matrix
confusion_matrix <- table(predictions, red_wine_data$binary_response)

# Print the confusion matrix
print(confusion_matrix)
```

```
##
## predictions high low
##        high  855   0
##        low     0 744
```

```
# Calculate additional performance metrics
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])  # Positive predictive value
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])     # True positive rate
f1_score <- 2 * (precision * recall) / (precision + recall)

# Print the performance metrics
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 1
```

```
cat("Precision:", precision, "\n")
```

```
## Precision: 1
```

```r
cat("Recall:", recall, "\n")
```

```
## Recall: 1
```

```r
cat("F1-score:", f1_score, "\n")
```

```
## F1-score: 1
```

the perfect performance metrics suggest that the dataset may have been relatively simple or that the model may have overfit the data. It's crucial to further investigate and validate the model's performance on unseen data to ensure its reliability and generalizability.

I learn that the features included in the dataset are highly informative and can effectively discriminate between different quality levels of wine. The KNN algorithm, when applied to this dataset, was able to leverage these features to make accurate predictions.