Homework 7: ISYE 6501 - Introduction to Analytics Modeling

Question 10.1

Prompt

Using the same crime data set `uscrime.txt` as in Questions 8.2 and 9.1, find the best model you can using:

(a) a regression tree model, and
(b) a random forest model

In R, you can use the `tree` package or the `rpart` package, and the `randomForest` package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

Solution

Approach

The initial approach to the problem is by importing the crime data and inspecting the data set. First, I set my current working directory to `HW7` and loaded the `uscrime.txt` data set into a table. Using `head()` I display the data to determine if the import was successful and observe the data set. Observing the data, there are 16 columns (variables) that contain 47 rows of data (for all 47 states in 1960).

```
# Load packages
library(rpart) # Regression tree model
library(rpart.plot) # Plotting regression tree

# Set seed so results are reproducible
set.seed(123)

# Set the working directory
setwd("~/projects/ISYE6501/HW7")

# Load the US crime data data into a table
data <- read.table("data/uscrime.txt", stringsAsFactors = FALSE, header = TRUE)
cat(paste("\nUS Crime Data:\n"))
```

```
##
## US Crime Data:
```

```
print(head(data, 5))
```

```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
```

Regression Tree Model

To find the best regression tree model, the `rpart` package is used to fit a regression tree. `rpart` is a package that can be utilized for recursive partitioning and regression trees [1].

Using the US crime data from `uscrime.txt`, a model can be created by first defining a formula. By using the `formula = Crime ~ .` defines `Crime` as the response variable and to include all 15 predictor variables. In addition, `rpart` requires the consideration of the type of response. Since the data should be utilized to predict crime rate, we are looking for a continuous response model (rather than binary), which is defined as the *anova model* [1]. This leads to developing a regression tree model and fitting the regression tree by defining `rpart(Crime ~ ., data = data, method = "anova")`.

```r
# Fit the regression tree model
rpart_tree_model <- rpart(Crime ~ ., data = data, method = "anova")

# Print the summary of the model
summary(rpart_tree_model)
```

```
## Call:
## rpart(formula = Crime ~ ., data = data, method = "anova")
##   n= 47
##
##           CP nsplit rel error   xerror      xstd
## 1 0.36296293      0 1.0000000 1.067320 0.2603150
## 2 0.14814320      1 0.6370371 1.053288 0.2172616
## 3 0.05173165      2 0.4888939 1.030801 0.2400202
## 4 0.01000000      3 0.4371622 1.019464 0.2412517
##
## Variable importance
##    Po1    Po2 Wealth   Ineq   Prob      M     NW    Pop   Time     Ed     LF
##     17     17     11     11     10     10      9      5      4      4      1
##     So
##      1
##
## Node number 1: 47 observations,    complexity param=0.3629629
##   mean=905.0851, MSE=146402.7
##   left son=2 (23 obs) right son=3 (24 obs)
##   Primary splits:
##       Po1    < 7.65      to the left,  improve=0.3629629, (0 missing)
##       Po2    < 7.2       to the left,  improve=0.3629629, (0 missing)
```

```
##       Prob   < 0.0418485 to the right, improve=0.3217700, (0 missing)
##       NW     < 7.65      to the left,  improve=0.2356621, (0 missing)
##       Wealth < 6240      to the left,  improve=0.2002403, (0 missing)
##   Surrogate splits:
##       Po2    < 7.2       to the left,  agree=1.000, adj=1.000, (0 split)
##       Wealth < 5330      to the left,  agree=0.830, adj=0.652, (0 split)
##       Prob   < 0.043598  to the right, agree=0.809, adj=0.609, (0 split)
##       M      < 13.25     to the right, agree=0.745, adj=0.478, (0 split)
##       Ineq   < 17.15     to the right, agree=0.745, adj=0.478, (0 split)
##
## Node number 2: 23 observations,    complexity param=0.05173165
##   mean=669.6087, MSE=33880.15
##   left son=4 (12 obs) right son=5 (11 obs)
##   Primary splits:
##       Pop < 22.5      to the left,  improve=0.4568043, (0 missing)
##       M   < 14.5      to the left,  improve=0.3931567, (0 missing)
##       NW  < 5.4       to the left,  improve=0.3184074, (0 missing)
##       Po1 < 5.75      to the left,  improve=0.2310098, (0 missing)
##       U1  < 0.093     to the right, improve=0.2119062, (0 missing)
##   Surrogate splits:
##       NW   < 5.4       to the left,  agree=0.826, adj=0.636, (0 split)
##       M    < 14.5      to the left,  agree=0.783, adj=0.545, (0 split)
##       Time < 22.30055  to the left,  agree=0.783, adj=0.545, (0 split)
##       So   < 0.5       to the left,  agree=0.739, adj=0.455, (0 split)
##       Ed   < 10.85     to the right, agree=0.739, adj=0.455, (0 split)
##
## Node number 3: 24 observations,    complexity param=0.1481432
##   mean=1130.75, MSE=150173.4
##   left son=6 (10 obs) right son=7 (14 obs)
##   Primary splits:
##       NW   < 7.65      to the left,  improve=0.2828293, (0 missing)
##       M    < 13.05     to the left,  improve=0.2714159, (0 missing)
##       Time < 21.9001   to the left,  improve=0.2060170, (0 missing)
##       M.F  < 99.2      to the left,  improve=0.1703438, (0 missing)
##       Po1  < 10.75     to the left,  improve=0.1659433, (0 missing)
##   Surrogate splits:
##       Ed   < 11.45     to the right, agree=0.750, adj=0.4, (0 split)
##       Ineq < 16.25     to the left,  agree=0.750, adj=0.4, (0 split)
##       Time < 21.9001   to the left,  agree=0.750, adj=0.4, (0 split)
##       Pop  < 30        to the left,  agree=0.708, adj=0.3, (0 split)
##       LF   < 0.5885    to the right, agree=0.667, adj=0.2, (0 split)
##
## Node number 4: 12 observations
##   mean=550.5, MSE=20317.58
##
## Node number 5: 11 observations
##   mean=799.5455, MSE=16315.52
##
## Node number 6: 10 observations
##   mean=886.9, MSE=55757.49
##
## Node number 7: 14 observations
##   mean=1304.929, MSE=144801.8
```

Regression Tree Predictions

```r
# Make predictions
rpart_predictions <- predict(rpart_tree_model)

# Calculate Mean Squared Error (MSE)
rpart_mse <- mean((data$Crime - rpart_predictions)^2)
cat("Mean Squared Error:", rpart_mse, "\n")
```

```
## Mean Squared Error: 64001.74
```

The mean squared error is quite high, and could be improved. Before discussing improvements, the regression tree should be visualized.

Regression Tree

To visualize the regression tree, a plot is created by using `rpart.plot`. The plot is configured for the continuous data with `type = 4` to label all nodes, not just leaves [2]. The title is added after the plot is generated using `title` to adjust the whitespace between the tree and the plot title.

```
# Plot the tree with rpart.plot
rpart.plot(rpart_tree_model,
           type = 4,           # Type of plot
           extra = 101,       # Show fitted values at nodes
           under = TRUE,      # Show splits under nodes
           box.palette = "RdYlGn", # Continuous
           shadow.col = "gray",      # Add shadow effect
           nn = TRUE)         # Add node numbers

# Add the title manually with less space between tree with "line"
title(main = "Regression Tree for Crime Data",
      # Center the title
      adj = 0.5,
      # Adjust whitespace between title and tree
      line = 3)
```

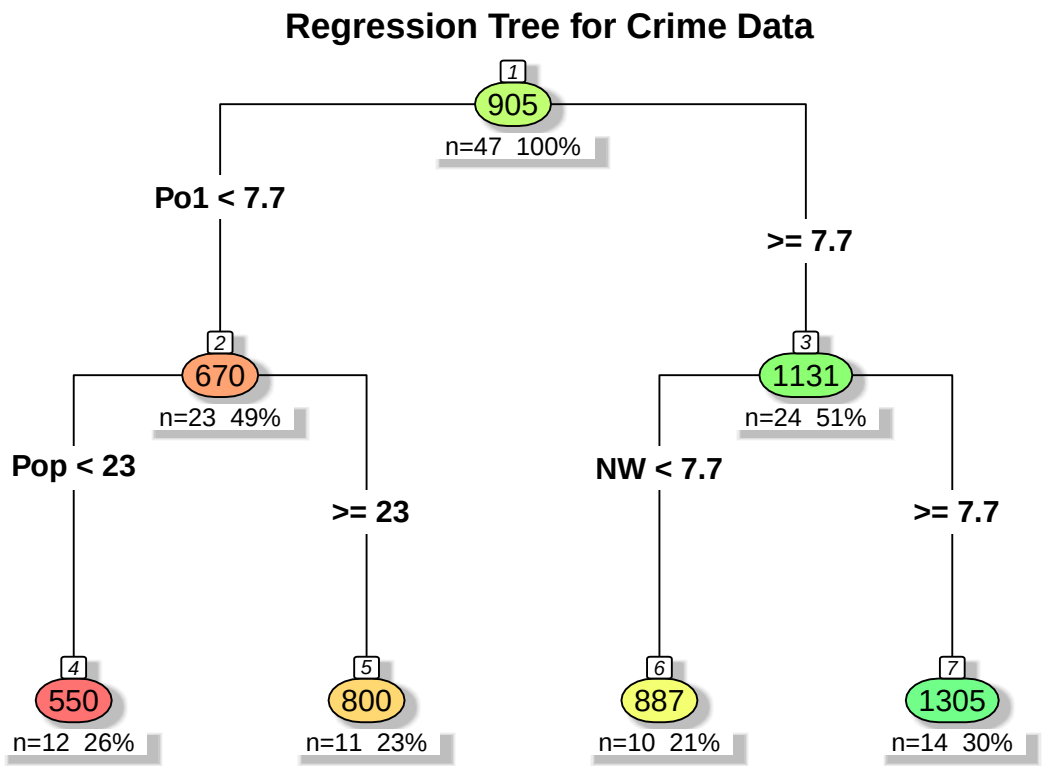


Figure 1: US Crime Regression Tree

Tree Pruning

With the initial regression tree fit using `rpart`, the complexity parameter (CP) table should be examined. Examining the CP table will assist in determining the optimal level for tree pruning. To determine the optimal CP value cross-validation should be used to determine which CP value minimizes the error. The reasoning behind the importance of cross-validation is that building regression trees on the training data set may produce good results, but is likely to overfit the data which leads to poor test set performance [3].

To implement cross-validation, the `caret` package is used in conjunction with the initial fit from the `rpart` regression tree model. The `caret` package uses `trainControl` with `number = 5` to perform 5-fold cross-validation. After cross-validation is performed, the model can be tuned by first creating a grid to find the optimal CP value. Using crime data, `train_control`, and the tuning grid, the model is re-fit with cross-validation. With the refit model, the optimal CP value can be determined from `model_cv$bestTune$cp`. Using the optimal CP value, the regression tree can be pruned. The optimal CP value determined is printed below.

```
library(caret)

# Fit the regression tree model
rpart_tree_model <- rpart(Crime ~ ., data = data, method = "anova")

# 5-fold cross-validation
train_control <- trainControl(method = "cv", number = 5)

# Tune the model using cross-validation to find the optimal CP
tune_grid <- expand.grid(cp = seq(0.01, 0.1, by = 0.01))

# Fit the model with cross-validation
model_cv <- train(Crime ~ ., data = data, method = "rpart",
                  trControl = train_control, tuneGrid = tune_grid)

# Prune the initial tree using the optimal CP value
pruned_tree_model <- prune(rpart_tree_model, cp = model_cv$bestTune$cp)

# Print the best CP value based on cross-validation
cat("\nOptimal CP value for pruning:", model_cv$bestTune$cp, "\n")
```

```
##
## Optimal CP value for pruning: 0.07
```

```
summary(pruned_tree_model)
```

```
## Call:
## rpart(formula = Crime ~ ., data = data, method = "anova")
##   n= 47
##
##          CP nsplit rel error    xerror      xstd
## 1 0.3629629      0 1.0000000 1.0627176 0.2641620
## 2 0.1481432      1 0.6370371 0.7664987 0.1693164
## 3 0.0700000      2 0.4888939 0.9683905 0.2428916
##
## Variable importance
##    Po1    Po2 Wealth   Ineq   Prob      M     NW     Ed   Time    Pop     LF
```

```
##     19     19     12     12     11      9      8      3      3      2      2
##
## Node number 1: 47 observations,    complexity param=0.3629629
##   mean=905.0851, MSE=146402.7
##   left son=2 (23 obs) right son=3 (24 obs)
##   Primary splits:
##       Po1    < 7.65      to the left,  improve=0.3629629, (0 missing)
##       Po2    < 7.2       to the left,  improve=0.3629629, (0 missing)
##       Prob   < 0.0418485 to the right, improve=0.3217700, (0 missing)
##       NW     < 7.65      to the left,  improve=0.2356621, (0 missing)
##       Wealth < 6240      to the left,  improve=0.2002403, (0 missing)
##   Surrogate splits:
##       Po2    < 7.2       to the left,  agree=1.000, adj=1.000, (0 split)
##       Wealth < 5330      to the left,  agree=0.830, adj=0.652, (0 split)
##       Prob   < 0.043598  to the right, agree=0.809, adj=0.609, (0 split)
##       M      < 13.25     to the right, agree=0.745, adj=0.478, (0 split)
##       Ineq   < 17.15     to the right, agree=0.745, adj=0.478, (0 split)
##
## Node number 2: 23 observations
##   mean=669.6087, MSE=33880.15
##
## Node number 3: 24 observations,    complexity param=0.1481432
##   mean=1130.75, MSE=150173.4
##   left son=6 (10 obs) right son=7 (14 obs)
##   Primary splits:
##       NW   < 7.65      to the left,  improve=0.2828293, (0 missing)
##       M    < 13.05     to the left,  improve=0.2714159, (0 missing)
##       Time < 21.9001   to the left,  improve=0.2060170, (0 missing)
##       M.F  < 99.2      to the left,  improve=0.1703438, (0 missing)
##       Po1  < 10.75     to the left,  improve=0.1659433, (0 missing)
##   Surrogate splits:
##       Ed   < 11.45     to the right, agree=0.750, adj=0.4, (0 split)
##       Ineq < 16.25     to the left,  agree=0.750, adj=0.4, (0 split)
##       Time < 21.9001   to the left,  agree=0.750, adj=0.4, (0 split)
##       Pop  < 30        to the left,  agree=0.708, adj=0.3, (0 split)
##       LF   < 0.5885    to the right, agree=0.667, adj=0.2, (0 split)
##
## Node number 6: 10 observations
##   mean=886.9, MSE=55757.49
##
## Node number 7: 14 observations
##   mean=1304.929, MSE=144801.8
```

Pruned Regression Tree

The regression tree was refit above with 5-fold cross-validation to determine the optimal CP value. With the optimal CP value, the regression tree was pruned. To visualize the tree, the code below generates a pruned regression tree to observe.

```
# Visualize the pruned tree
# Plot the tree with rpart.plot
rpart.plot(pruned_tree_model,
           # main = "Regression Tree for Crime Data",
           type = 4,           # Type of plot
           extra = 101,       # Show fitted values at terminal nodes
           under = TRUE,      # Show splits under nodes
           box.palette = "RdYlGn", # "Blues",  # Continuous
           shadow.col = "gray",      # Add shadow effect
           nn = TRUE)         # Add node numbers

# Add the title manually with less space between tree with "line"
title(main = "Pruned Regression Tree for Crime Data",
      # Center the title
      adj = 0.5,
      # Adjust whitespace between title and tree
      line = 3)
```

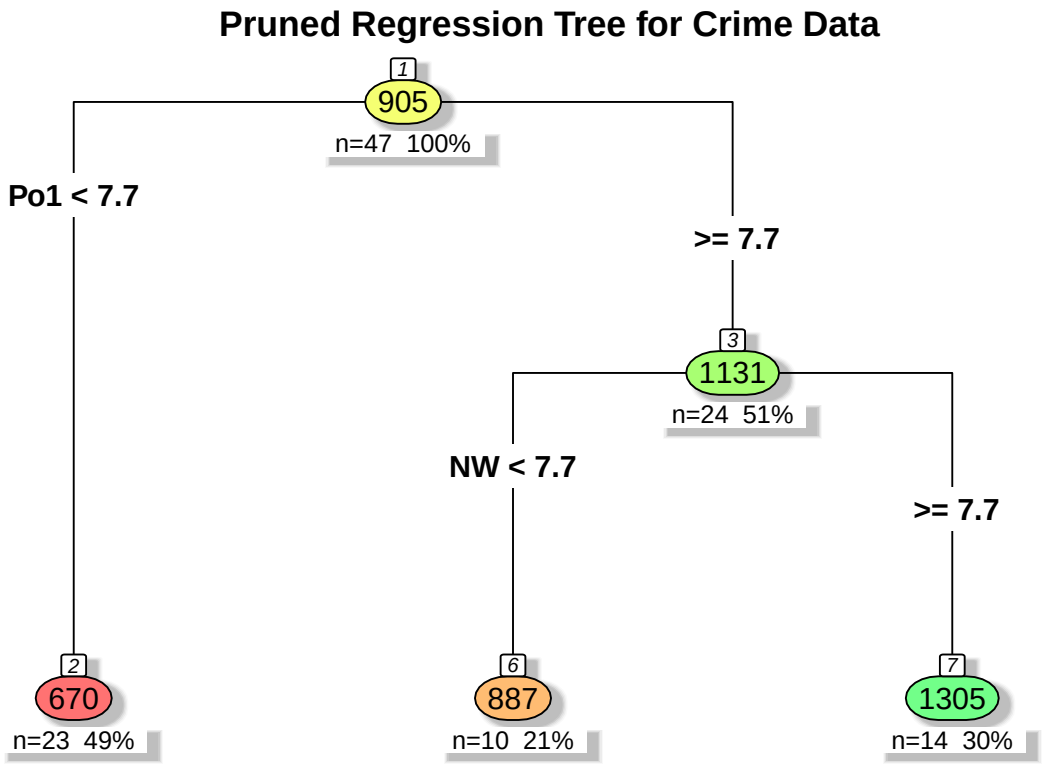


Figure 2: US Crime Pruned Regression Tree

Random Forest Model

Finding the best random forest model is determined by utilizing the `randomForest` package. Random forests are a combination of tree predictors where each tree depends on the values of a random vector sampled independently with analogous distribution for all trees in the forest [4]. `randomForest` implements the random forest algorithm by Breiman for classification and regression [5].

```
library(randomForest)
# Create the random forest model
rf_model <- randomForest(Crime ~ .,
                         data = data,
                         # Calc. variable importance
                         importance = TRUE,
                         ntree = 500)  # Number of trees

summary(rf_model)
```

```
##                 Length Class  Mode
## call              5    -none- call
## type              1    -none- character
## predicted        47    -none- numeric
## mse             500    -none- numeric
## rsq             500    -none- numeric
## oob.times        47    -none- numeric
## importance       30    -none- numeric
## importanceSD     15    -none- numeric
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y                47    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

Variable Importance

With a random forest model formulated, check the variable importance, which will print out data for each variable for `%IncMSE` and `IncNodePurity`.

The first column, `%IncMSE`, is the percent increase in mean squared error. The percent increase in mean squared error reflects how much the mean squared error (MSE) increases if the predictor is excluded. If the variable is important, excluding the values will decrease the accuracy of the model. A higher `%IncMSE` indicates that the variable is more important. Observing the results, `Po1`, `Po2` and `Prob` are the most important predictors to determine crime rate, while `M`, `Pop` `U1` are not significant predictors to help predictions.

The second column, `IncNodePurity`, is the increase in node purity. Node purity is a term that refers to how well a decision tree splits the data, as every split has the goal to make each resulting node as pure as possible. `IncNodePurity` measures the total decrease in node impurity that a predictor causes across all trees in the forst. The larger the value, the more significant the predictor. `Po1`, `Po2` and `Prob` are the most important predictors. `So`, `U1` and `U2` are the least significant.

```
# Check variable importance
importance(rf_model)
```

```
##           %IncMSE IncNodePurity
## M      -0.3166669     200609.82
## So      3.1956050      21720.47
## Ed      3.8460475     231937.89
## Po1    12.4126688    1210514.72
## Po2    11.5117581    1217014.35
## LF      2.0156561     266780.70
## M.F     0.7951210     233823.12
## Pop    -0.8065943     314797.77
## NW      9.0817676     537534.27
## U1     -0.2836526     145784.25
## U2      1.2021625     172699.43
## Wealth  3.1665818     633752.67
## Ineq    0.6150785     240048.91
## Prob    8.5727816     778831.47
## Time    3.0661715     236091.12
```

The importance of predictors are discussed above by values associated with `%IncMSE` and `IncNodePurity`. To visualize the calculated values and their significance, a plot is generated below that shows variable important. The plot is generated with `varImpPlot` to create a dotchart of variable importance as measured by a random forest [6]. The plotting function uses `sort` to arrange the importance of predictors in descending order. The title is included after the plot function to allow for adjusting the vertical whitespace between the plot and the title.

```
# Plot variable importance
varImpPlot(rf_model,
           sort = TRUE,
           main = "",  # No main title
           col = "red",
           # Adjust label sizes
           cex = 0.8)

# Add the title manually with less space between plots
title(main = "Variable Importance for Crime Rate",
```

```
      # Center the title
      adj = 0.5,
      # Adjust whitespace between title and tree
      line = 0)
```

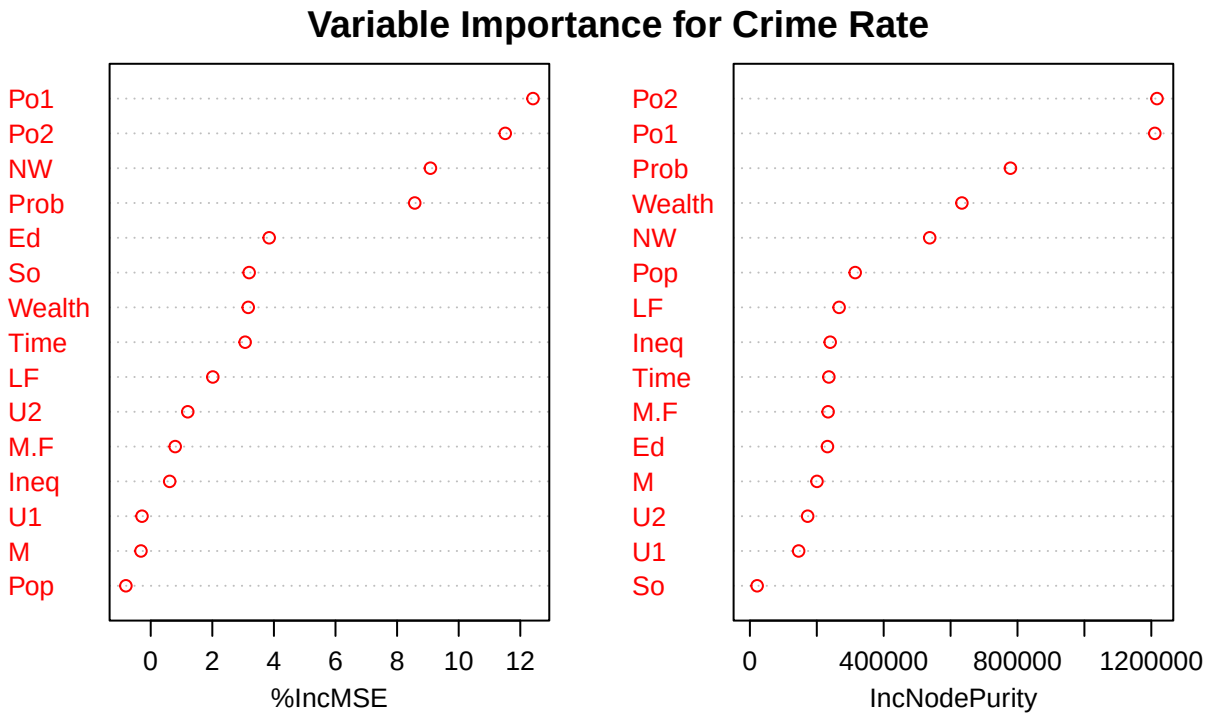


Figure 3: Predictor Significance for Random Forest Tree

Observing the plot, it can be determined that the most important predictors are `Po1` and `Po2` both from `%IncMSE` and `IncNodePurity`.

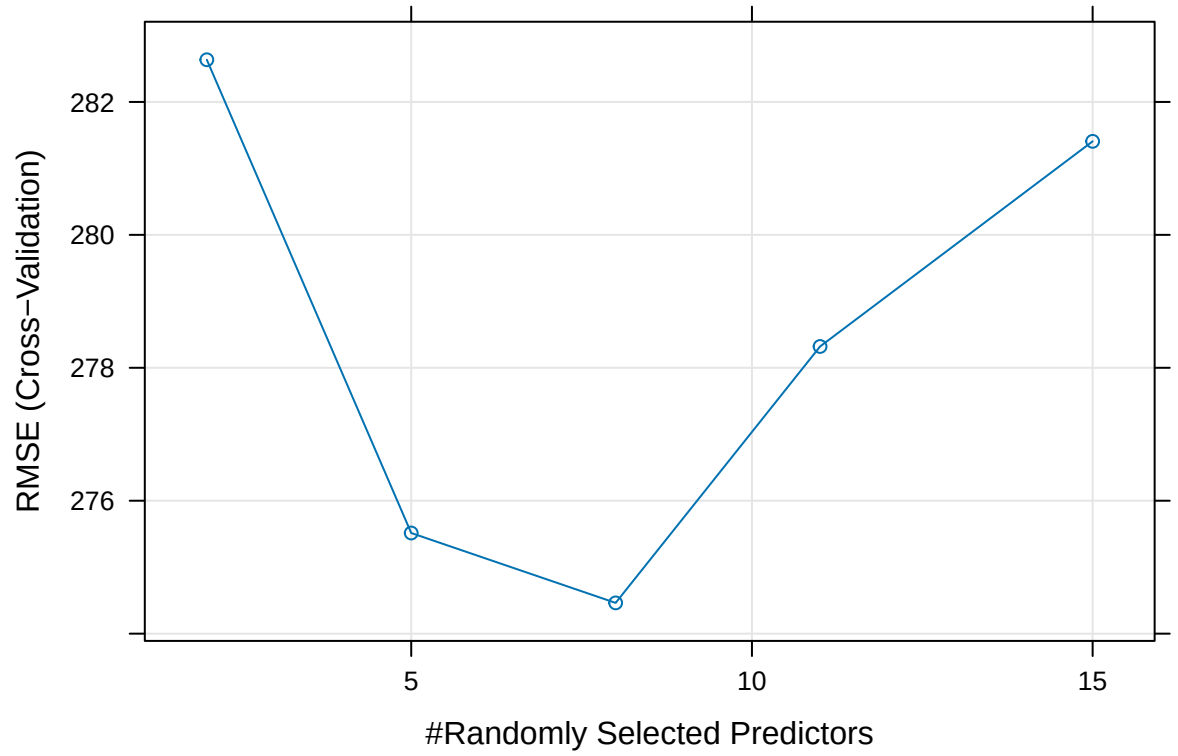
```
# Define cross-validation method
train_control <- trainControl(method = "cv", number = 5)  # 5-fold cross-validation

# Train the model with cross-validation
tuned_rf <- train(Crime ~ .,
                  data = data,
                  method = "rf",
                  trControl = train_control,
                  importance = TRUE,
                  ntree = 500,
                  tuneLength = 5)
print(tuned_rf)
```

```
## Random Forest
```

```
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 38, 36, 38, 38, 38
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    282.6347  0.5318684  216.2490
##    5    275.5141  0.5361815  212.3032
##    8    274.4629  0.5389533  214.6496
##   11    278.3206  0.5194497  218.8204
##   15    281.4073  0.5040219  225.0038
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 8.
```

```r
# Plot the performance of different hyperparameter combinations
plot(tuned_rf)
```




```r
# Extract the best model after cross-validation
best_rf <- tuned_rf$finalModel
# Plot variable importance
```

```r
varImpPlot(best_rf,
           sort = TRUE,
           main = "",  # No main title
           col = "red",
           # Adjust label sizes
           cex = 0.8)

# Add the title manually with less space between plots
title(main = "Variable Importance for Crime Rate",
      # Center the title
      adj = 0.5,
      # Adjust whitespace between title and tree
      line = 0)
```

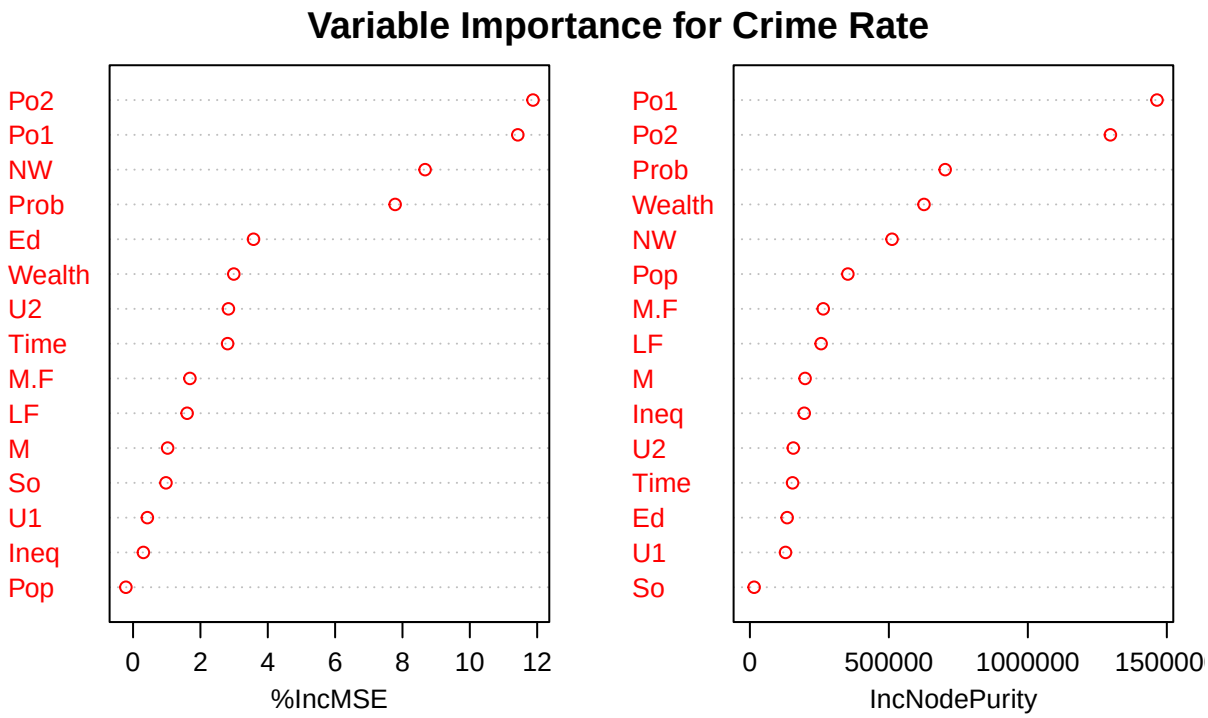


Figure 4: Predictor Significance for Random Forest Tree with 5-fold Cross-Validation

Question 10.2

Prompt

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

Solution

NFL 4th Down Conversion

Consider the National Football League (NFL), where you have 4 downs to convert a certain amount of yards. A logistic regression model would be appropriate in predicting whether a team will convert a 4th down attempt, defined as success or failure. The logistic regression model is appropriate considering this is a binary outcome where success is 1 and failure is 0. Some of the potential predictors include:

- **Yards to Gain:** The number of yards needed to convert is important as longer distances decrease the probability of conversion.
- **Field Position:** Depending where the attempt on the field is being made, this could indicate the team willingness to attempt converting a 4th down attempt to convert.
- **Down and Distance Tendency:** This predictors considers the frequency a team attempts 4th down in a similar situation. A prepared team with aggressive strategies could have a higher chance of success.
- **Quarterback Passer Rating:** The success of the quarterback is essential to offensive performance and could indicate a higher chance of success for plays on 4th down.
- **Defensive Strength of the Opposing Team:** Strength of the defense could be considered as a stronger team could have a great potential to stop a 4th down conversion. This is typically measured by metrics like opponent 4th down stop rate or overall defensive efficiency.

Question 10.3

Prompt

Part I:

Using the German Credit data set `germancredit.txt` from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/ (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not.
Show your model (factors used and their coefficients), the software output, and the quality of fit.
You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use `family=binomial(link="logit")` in your `glm` function call.

Part II:

Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers.
In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad.
Determine a good threshold probability based on your model.

Solution: Part I

Approach

The German Credit data set [7] is utilized to use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Logistic regression models or logit models can be utilized to model the combination of categorical variables [8].

To approach this problem, the seed is set with `set.seed(123)` so that the results are reproducible. Next, the working directory is set to easily load `germancredit.txt` stored in `data/`. German credit data is imported from the text file and the first 5 rows of the data are printed to inspect.

```
# Set seed so results are reproducible
set.seed(123)

# Set the working directory
setwd("~/projects/ISYE6501/HW7")

# Load the german credit data into a table 999 obs. of 21 variables
data <- read.table("data/germancredit.txt", stringsAsFactors = FALSE, header = FALSE)
print(head(data, 5))
```

```
##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
##    V19  V20 V21
## 1 A192 A201   1
## 2 A191 A201   2
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   2
```

Implement Logistic Regression Model

With the data confirmed to be imported correctly, a logistic regression model needs to be implemented. The prompt specifies to use the `glm` model, which is used to fit generalized linear models, specified by a symbolic description of the predictor and error distribution [9]. The logistic regression model should utilize `logit` which is defined in the `glm` function by family objects, which provide specifying details of the models [10].

To implement the `logit` logistic regression model, the binomial family is specified, corresponding to logistic regression models. German credit data classifies an applicant in `V21` as a "good risk" with a value of 1, and as a "bad risk with a value of 2. The binomial family recognizes classification values as 0 or 1, so the data must first be prepared by converting the response variable data in `V21` to binary values. This means that values of 1 for"good risk" should be converted to 0 and values of 2 for "bad risk" should be converted to 1.

With the response variable converted to binary values, the `glm` logistic regression value using `family = binomial(link = "logit")` is implemented. An initial fit of the logistic regression model is performed. To summarize the model, the software output is printed by using `print(summary_model)`.

```r
# Convert the response variable (V21) to binary to use logit
# Good risk = 1 needs to be converted to 0
data$V21[data$V21==1] <- 0
# Bad risk = 2 needs to be converted to 1
data$V21[data$V21==2] <- 1

# Fit the logistic regression model
model <- glm(V21 ~ ., data = data, family = binomial(link = "logit"))

# Display the model summary
summary_model <- summary(model)
print(summary_model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.005e-01  1.084e+00   0.369 0.711869
## V1A12       -3.749e-01  2.179e-01  -1.720 0.085400 .
## V1A13       -9.657e-01  3.692e-01  -2.616 0.008905 **
## V1A14       -1.712e+00  2.322e-01  -7.373 1.66e-13 ***
## V2           2.786e-02  9.296e-03   2.997 0.002724 **
## V3A31        1.434e-01  5.489e-01   0.261 0.793921
## V3A32       -5.861e-01  4.305e-01  -1.362 0.173348
## V3A33       -8.532e-01  4.717e-01  -1.809 0.070470 .
## V3A34       -1.436e+00  4.399e-01  -3.264 0.001099 **
## V4A41       -1.666e+00  3.743e-01  -4.452 8.51e-06 ***
## V4A410      -1.489e+00  7.764e-01  -1.918 0.055163 .
## V4A42       -7.916e-01  2.610e-01  -3.033 0.002421 **
## V4A43       -8.916e-01  2.471e-01  -3.609 0.000308 ***
## V4A44       -5.228e-01  7.623e-01  -0.686 0.492831
## V4A45       -2.164e-01  5.500e-01  -0.393 0.694000
## V4A46        3.628e-02  3.965e-01   0.092 0.927082
## V4A48       -2.059e+00  1.212e+00  -1.699 0.089297 .
## V4A49       -7.401e-01  3.339e-01  -2.216 0.026668 *
## V5           1.283e-04  4.444e-05   2.887 0.003894 **
```

```
## V6A62       -3.577e-01  2.861e-01  -1.250 0.211130
## V6A63       -3.761e-01  4.011e-01  -0.938 0.348476
## V6A64       -1.339e+00  5.249e-01  -2.551 0.010729 *
## V6A65       -9.467e-01  2.625e-01  -3.607 0.000310 ***
## V7A72       -6.691e-02  4.270e-01  -0.157 0.875475
## V7A73       -1.828e-01  4.105e-01  -0.445 0.656049
## V7A74       -8.310e-01  4.455e-01  -1.866 0.062110 .
## V7A75       -2.766e-01  4.134e-01  -0.669 0.503410
## V8           3.301e-01  8.828e-02   3.739 0.000185 ***
## V9A92       -2.755e-01  3.865e-01  -0.713 0.476040
## V9A93       -8.161e-01  3.799e-01  -2.148 0.031718 *
## V9A94       -3.671e-01  4.537e-01  -0.809 0.418448
## V10A102      4.360e-01  4.101e-01   1.063 0.287700
## V10A103     -9.786e-01  4.243e-01  -2.307 0.021072 *
## V11          4.776e-03  8.641e-02   0.055 0.955920
## V12A122      2.814e-01  2.534e-01   1.111 0.266630
## V12A123      1.945e-01  2.360e-01   0.824 0.409743
## V12A124      7.304e-01  4.245e-01   1.721 0.085308 .
## V13         -1.454e-02  9.222e-03  -1.576 0.114982
## V14A142     -1.232e-01  4.119e-01  -0.299 0.764878
## V14A143     -6.463e-01  2.391e-01  -2.703 0.006871 **
## V15A152     -4.436e-01  2.347e-01  -1.890 0.058715 .
## V15A153     -6.839e-01  4.770e-01  -1.434 0.151657
## V16          2.721e-01  1.895e-01   1.436 0.151109
## V17A172      5.361e-01  6.796e-01   0.789 0.430160
## V17A173      5.547e-01  6.549e-01   0.847 0.397015
## V17A174      4.795e-01  6.623e-01   0.724 0.469086
## V18          2.647e-01  2.492e-01   1.062 0.288249
## V19A192     -3.000e-01  2.013e-01  -1.491 0.136060
## V20A202     -1.392e+00  6.258e-01  -2.225 0.026095 *
## ---
## 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:  895.82  on 951  degrees of freedom
## AIC: 993.82
##
## Number of Fisher Scoring iterations: 5
```

In the model summary, the **coefficients** are listed under the column, `Estimate`.

Model Accuracy

To calculate the accuracy of the model, the `predict()` function can be utilized to generate predictions from the logistic regression model. Using `type = "response"` returns probabilities instead of log odds. Probabilities are between 0 and 1, which allows using `ifelse(... > 0.5, 1, 0)` to assign values above 0.5 as 1 and below 0.5 as 0 such that there are binary results of good and bad risks. The accuracy is determined by the mean of comparing the predictions to the original data in `data$V21`. The resulting accuracy is 0.786 or 78.6%.

```r
# Calculate the model's accuracy
# Return probabilities between 0 and 1 - "response"
# Convert probabilities into binary predictions ifelse(... > 0.5, 1, 0)
predictions <- ifelse(predict(model, type = "response") > 0.5, 1, 0)
accuracy <- mean(predictions == data$V21)
cat("Logistic Regression Model Accuracy:", accuracy)
```

```
## Logistic Regression Model Accuracy: 0.786
```

Model Pseudo R-Squared

For the logistic regression model, Pseudo's R-squared is calculated, where the pseudo R-squared measure compares the model to a null model (model with only an intercept, no predictors). The null model is created with `null_model <- glm(V21 ~ 1` as there are no predictors. The R-squared value is then calculated by subtracting 1 from the ratio of model deviance to null model deviance. Pseudo R-squared is the proportional reduction in deviance due to the inclusion of predictors. This means that if the model is not better than the null model, the ratio is close to 1, and Pseudo R-squared is close to 0. If the model is significantly better than the null model, Pseudo's R-squared is closer to 1.

```r
# Calculate Pseudo's R-squared
# Create null model with no response variables
null_model <- glm(V21 ~ 1, data = data, family = binomial(link = "logit"))
# Take 1 minus ratio of model deviance
pseudo_r2 <- 1 - (model$deviance / null_model$deviance)
cat("Pseudo R-squared:", pseudo_r2)
```

```
## Pseudo R-squared: 0.266762
```

The resulting Pseudo R-squared value was closer to 0 with a value of 0.266762. This shows that the model formed on the original data is not very good, which is expected because there was no data splitting or cross-validation.

Solution: Part II

Threshold

To determine a good threshold probability for classifying applicants as "good risk" or "bad risk", consider that misclassifying a "bad risk" is 5 times worse. The classification threshold needs to be adjusted to reflect the imbalance. Considering that the cost ratio is 5, we can use a threshold that minimizes expected cost such as:

$$\text{Threshold} = \frac{1}{1 + \text{Cost Ratio}} = \frac{1}{1+5} = 0.1667$$

Adjusted Predictions with Threshold

To apply the adjusted threshold, the equation is defined as `1 / (1 + cost_ratio)`. Then using our predicted probabilities, we can adjust them but stating that probabilities must be greater than the threshold, defined as `predictions_adjusted`. Using the adjusted predictions, the accuracy can be recalculated.

```r
# Set the threshold based on cost ratio
cost_ratio <- 5
threshold <- 1 / (1 + cost_ratio)

# Predict probabilities
probabilities <- predict(model, type = "response")

# Classify using the adjusted threshold
predictions_adjusted <- ifelse(probabilities > threshold, 1, 0)

# Calculate the accuracy of the adjusted threshold
accuracy_adjusted <- mean(predictions_adjusted == data$V21)
cat("Accuracy with adjusted threshold:", accuracy_adjusted, "\n")
```

```
## Accuracy with adjusted threshold: 0.653
```

The accuracy with the adjusted threshold decreased from 0.786 to 0.653. Even with a lower accuracy it is important to focus on the reduction of false negatives, which can be observed by checking the changes in classification looking at the confusion matrix.

Comparing Confusion Matrices

Confusion Matrix Before Threshold:

```r
# Confusion matrix
table(Predicted = predictions, Actual = data$V21)
```

```
##          Actual
## Predicted   0   1
##         0 626 140
##         1  74 160
```

Confusion Matrix After Threshold:

```r
# Confusion matrix
table(Predicted = predictions_adjusted, Actual = data$V21)
```

```
##          Actual
## Predicted   0   1
##         0 387  34
##         1 313 266
```

Comparing the confusion matrices, you can see the increase in “bad risks” labeled as 1 with the threshold adjusted predictions.

References

[1] "Rpart: Recursive partitioning and regression trees - r documentation — rdocumentation.org." https://www.rdocumentation.org/packages/rpart/versions/4.1.23/topics/rpart.

[2] "Rpart.plot: Plot an rpart model. A simplified interface to the prp function - r documentation — rdocumentation.org." https://www.rdocumentation.org/packages/rpart.plot/versions/3.1.2/topics/rpart.plot.

[3] J. Gareth, W. Daniela, H. Trevor, and T. Robert, *An introduction to statistical learning: With applications in r*. Spinger, 2013.

[4] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[5] "randomForest: Classification and regression with random forest - r documentation — rdocumentation.org." https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.2/topics/randomForest.

[6] "varImpPlot: Variable importance plot - r documentation — rdocumentation.org." https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/varImpPlot.

[7] U. M. L. Repository, "Statlog (german credit data)." https://archive.ics.uci.edu/dataset/144/statlog+german+credit+data, 1994.

[8] K. Okoye and S. Hosseini, *R programming: Statistical data analysis in research*. Springer Nature, 2024.

[9] "Glm: Fitting generalized linear models - r documentation — rdocumentation.org." https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm.

[10] "Family: Family objects for models - r documentation — rdocumentation.org." https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/family.