PES University, Bangalore
Established under the Karnataka Act No. 16 of 2013

UE20CS312 - Data Analytics
Worksheet 2c - Logistic Regression

Ishita Bharadwaj - PES1UG20CS648

Collaborated with Hita - PES1UG20CS645

Prerequisites

To download the data required for this worksheet, visit this Github link. Use the following libraries and read the dataset:

```
library(tidyverse)
library(InformationValue)
char_preds <- read.csv('got_characters.csv')
```

The Logit Model

The linear regression techniques discussed so far are used to model the relationship between a quantitative response variable and one or more explanatory variables. When the response variable is categorical, other techniques are more suited to the task of classification.

The **logit model**, or **logistic model** models the probability, p, that a dichotomous (binary), dependent variable takes on one of two possible outcomes. It achieves this by setting the natural logarithm of the odds of the response variable, called the *log-odds* or *logit*, as a linear function of the explanatory variables.

$$Z_i = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x + \ldots + \beta_n x_n \text{ for } p \in (0,1)$$

Here, Z_i is the log-odds of the response variable taking on a value with probability p.

Logistic regression is an algorithm that estimates the parameters, or coefficients, of the linear combination of the logit model. In this worksheet, we will classify a certain binary feature of a dataset using logistic regression.

Solutions

Solution 1

How many characters from the SoIaF world does this dataset contain information on? Calculate the percentage of missing data for each column of the dataset and print them out in descending order as a dataframe.

```
#No. of characters listed in the dataframe
sprintf("The no. of characters listed in the data = %d", nrow(char_preds))
```

```
## [1] "The no. of characters listed in the data = 1946"
```

```
#Set empty entries to NA
char_preds[char_preds == "" | char_preds == " "] <- NA

# Make a DF for number of null vals in each column
df <- data.frame(colSums(is.na(char_preds)) / nrow(char_preds) * 100)


# Rename column with an appropriate header
colnames(df) <- c('% Missing')

# Reset index and make a feature of col names
df$Columns <- rownames(df)
rownames(df) <- NULL

# Order in decreasing order of percentages -
new_df <- df[order(df$'% Missing', decreasing = TRUE), ]
rownames(new_df) <- NULL
new_df
```

```
##    % Missing           Columns
## 1   98.92086            mother
## 2   98.92086     isAliveMother
## 3   98.81809              heir
## 4   98.81809       isAliveHeir
## 5   98.66393            father
## 6   98.66393     isAliveFather
## 7   85.81706            spouse
## 8   85.81706     isAliveSpouse
## 9   77.74923       dateOfBirth
## 10  77.74923               age
## 11  65.21069           culture
## 12  51.79856             title
## 13  21.94245             house
## 14   0.00000                 X
## 15   0.00000              S.No
## 16   0.00000            actual
## 17   0.00000              name
## 18   0.00000              male
## 19   0.00000             book1
## 20   0.00000             book2
## 21   0.00000             book3
## 22   0.00000             book4
## 23   0.00000             book5
## 24   0.00000         isMarried
## 25   0.00000           isNoble
## 26   0.00000  numDeadRelations
## 27   0.00000 boolDeadRelations
## 28   0.00000         isPopular
## 29   0.00000        popularity
```

Data from both missing values in numeric fields and empty strings in descriptive fields. All missing placeholders is converted to type NA.

Solution 2

Step 1

What are the inferences you can draw from the output dataframe of the previous problem? How can we handle columns with extremely high proportions of missing data before moving on?

Hint: Does missing data in a column tell you something about the target variable (`actual`)? If not, set a missing percentage threshold of 80%, deeming the column as having insufficient data, and drop the column.

```
#What are the inferences you can draw from the output dataframe of
#the previous problem? How can we handle columns with extremely
#high proportions of missing data before moving on?


#If the missing data in a column does not tell you something about the target
#variable (`actual`) or it is MCAR, set a missing percentage threshold of 80%,
#deeming the column as having insufficient data, and drop the column.
remove <- new_df[new_df$'% Missing' > 80,'Columns']
char_preds <- char_preds[!names(char_preds) %in% remove]

#Function to compute mode (if you need this for any inputation)
# Function to calculate mode
get_mode <- function(x) {
  mode0 <- names(which.max(table(x)))
  if(is.numeric(x)) return(as.numeric(mode0))
  mode0
}
```

The dataframe with high(>80%) percentage of missing value are: 98.92086 mother

98.92086 mother

98.92086 isAliveMother

98.81809 heir
98.81809 isAliveHeir

98.66393 father

98.66393 isAliveFather

85.81706 spouse
85.81706 isAliveSpouse

These columns with extremely high proportions of missing data have to be understood as to which type of missing data it belongs to. A threshold is set(80%), deeming the column data as insufficient if missing values are > 80%.
The attributes about the alive status of the character's immediate family(That is their mother, father, heir and spouse) are missing.
It could be that if we do not know the name of the character's mother, there's a high chance we don't know if the mother is alive (because we don't know who we're looking for).
In any case, the target (whether the character is alive or dead) is not dependant on whether their parents, heir or spouse is alive or not. This is MCAR.

Step 2

Impute missing data in the remaining numeric features with a reasonable statistic. Make sure you observe the distribution of the data in the columns to pick out a reasonable statistic. For categorical variables, convert the columns to numeric features. *Hint: Use the `unclass` function and impute missing categorical feature values with a `-1`.*

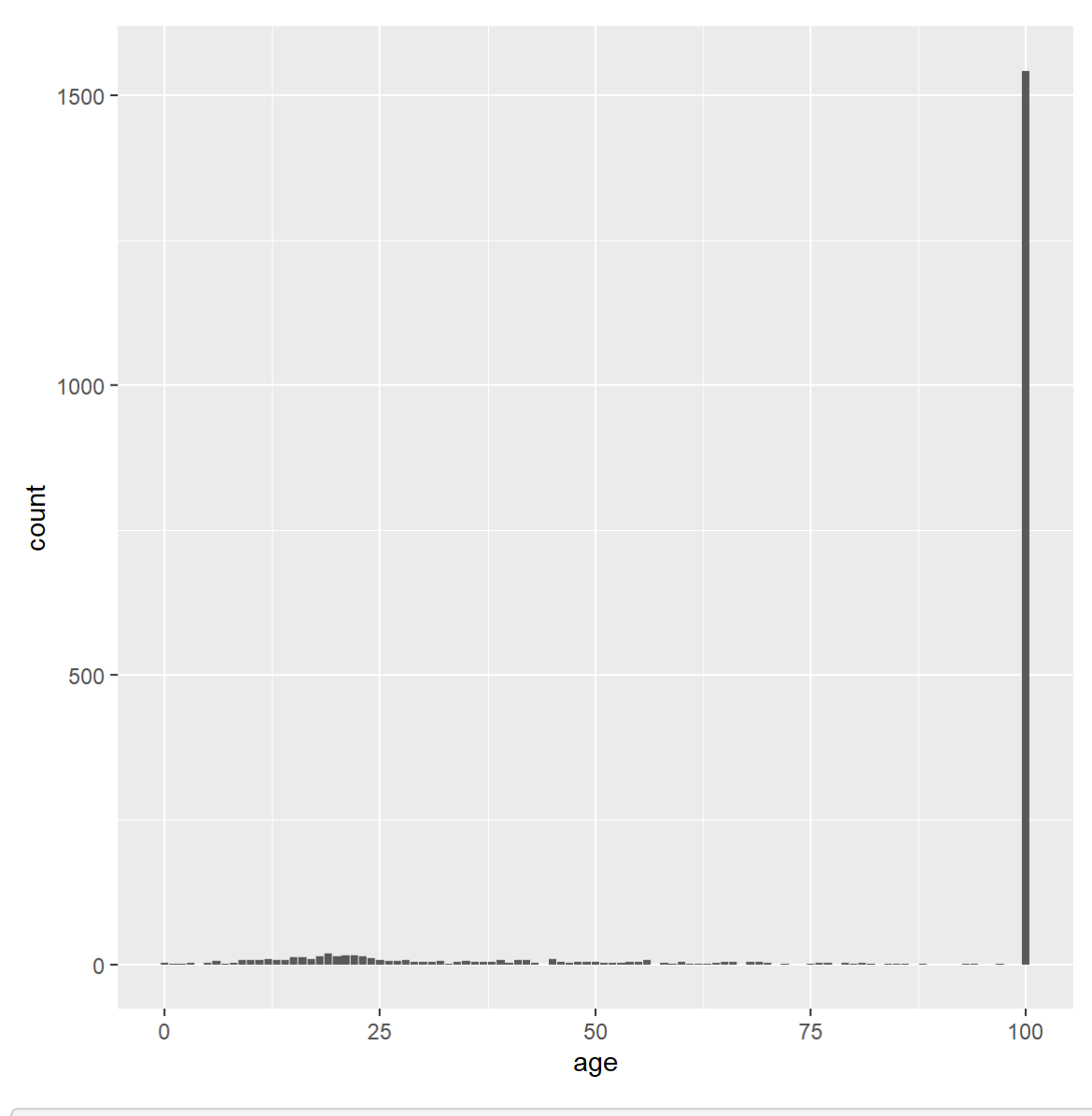
```
char_categorical <- c('culture','title','house')
char_preds[, char_categorical] <- lapply(char_preds[, char_categorical], as.factor)
char_preds[, char_categorical] <- sapply(char_preds[, char_categorical], unclass)
char_preds$age[is.na(char_preds$age)] <- get_mode(char_preds$age)

# Replace missing with -1
char_preds[is.na(char_preds)] = -1
#summary(char_preds)
```

Bonus

After plotting the `age` column, do you notice any discrepancies in the graph? What do you think might have given rise to a such a distribution?

```
ggplot(char_preds, aes(x=age)) + geom_bar()
```




```
summary(char_preds$age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00  100.00  100.00   85.92  100.00  100.00
```

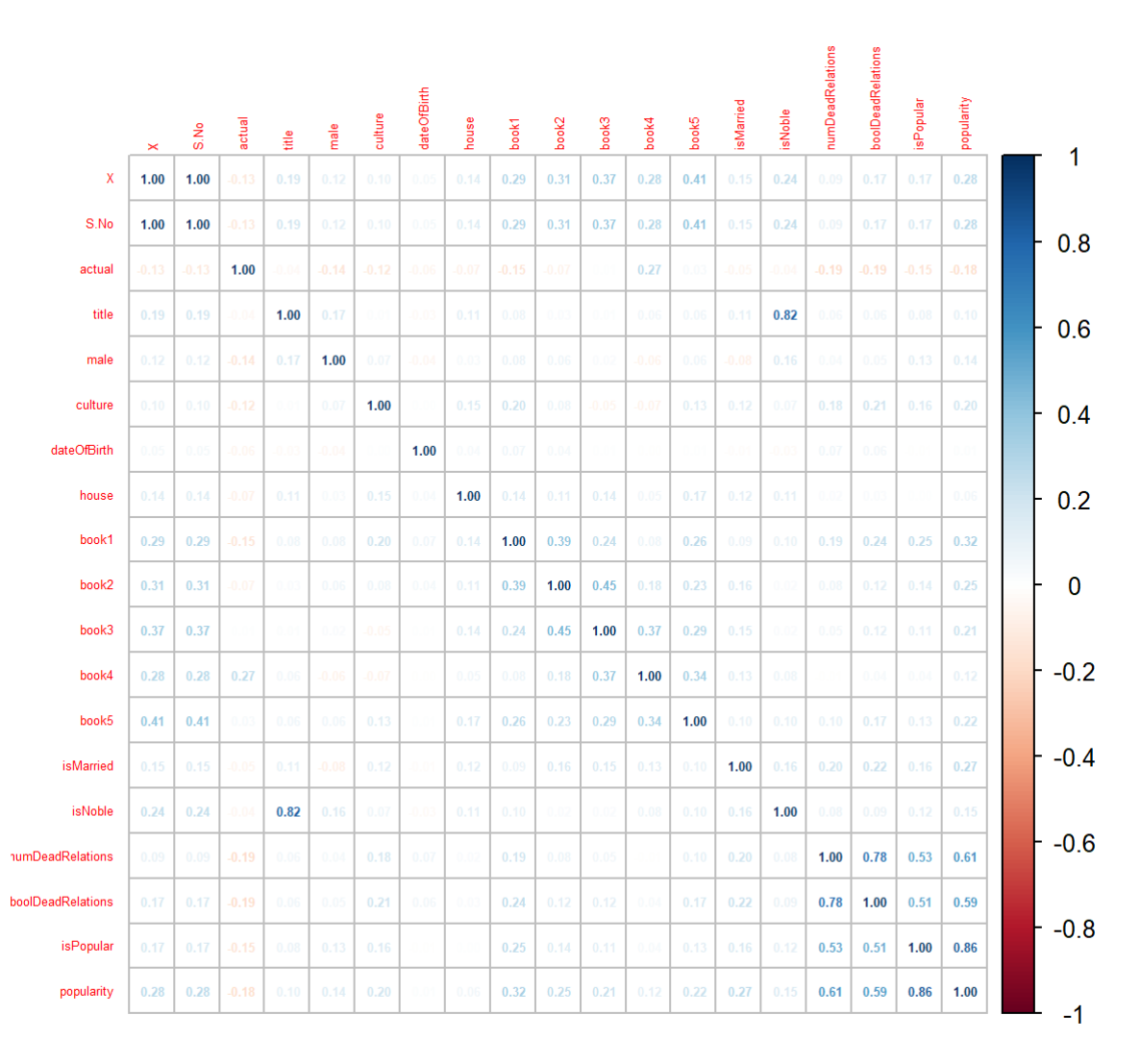
```
drop <- c("age")
char_preds = char_preds[,!(names(char_preds) %in% drop)]
```

From the plot, number of characters with age 100 appear to be abnormally high. Age of 100 years, expecially in game of thrones is not the most common.
So this abnormally high value might be because the actual age of the characters is not known so a default value of 100 is entered.
This might not be valid data, and might throw our analysis off.

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
#remove <- c('culture','title', 'house','name',)
cc <- subset (char_preds, select = -name)
#cc <- char_preds[~remove]
c <- cor(cc)
corrplot(c, method='number',addCoef.col = 1, number.cex = 0.4, tl.cex = 0.4)
```



In the correlation plot, `age` is not strongly correlated with the `actual` variable. We also observed the summary statistics of the logistic model (included `age` as one of the attributes) and noticed that `age` was not significant. Also because `age` has an abnormal value and the fact that 77% values are missing, imputing it with any central tendency value will again cause 1 value to be extremely high. So we will drop the age attribute.

Solution 3

Step 1: Check for Class Bias

Ideally, the proportion of both classes of the target variable should be the same. Is this so in the case of the target variable in this dataset?

```
# Original distribution
print("Actual propotion")
```

```
## [1] "Actual propotion"
```

```
table(char_preds$actual)
```

```
##
##    0    1
##  495 1451
```

In all of the data, proportion of both classes is not same. Class 0 samples: 495 Class 1 samples: 1451

Step 2: Create Training and Test Samples

Perform undersampling in case of a class bias in the dataset. Create train and test splits.

Hint: To create the training sample set, sample 70% of the class with lesser rows and sample the same number from the other class. Use the remaining rows from both classes to create the test split.

```
# Create training data
input_ones <- char_preds[which(char_preds$actual == 1), ]
input_zeros <- char_preds[which(char_preds$actual == 0), ]  # all 0's
set.seed(100)

# Sample from all alive characters
input_ones_training_rows <- sample(1:nrow(input_ones), 0.7*nrow(input_zeros))
# Sample from all dead characters
input_zeros_training_rows <- sample(1:nrow(input_zeros), 0.7*nrow(input_zeros))

training_ones <- input_ones[input_ones_training_rows, ]
training_zeros <- input_zeros[input_zeros_training_rows, ]

# Row bind both class dataframes
trainingData <- rbind(training_ones, training_zeros)
# Shuffle rows
trainingData <- trainingData[sample(1:nrow(trainingData)), ]


# Create testing data
test_ones <- input_ones[-input_ones_training_rows, ]
test_zeros <- input_zeros[-input_zeros_training_rows, ]

# Row bind both class dataframes
testData <- rbind(test_ones, test_zeros)
# Shuffle rows
testData <- testData[sample(1:nrow(testData)), ]

#Create testing data for zeros and bind the dataframes as shown in the above lines
#testData is the variable that will store the dataframe after binding


#Check the distribution of classes in the splits
print("Training and Test data propotion")
```

```
## [1] "Training and Test data propotion"
```

```
table(trainingData$actual)
```

```
##
##   0   1
## 346 346
```

```
table(testData$actual)
```

```
##
##    0    1
##  149 1105
```

Because 70% of 495 for class 0 = 346, since we want to keep our class 1 proportion same, so training samples for class 0 = 346. The remaining go to test ones(1105) and test zeros(149), making up the testData.

Solution 4

Step 1: Build the Logisitic Regression Model

Train a logistic regression model to predict whether a character is dead by Book 5 (feature: `actual`) using the training split. Use the `summary` function to print the beta coefficients, p values and other statistics. Predict characters' deaths on the test split available.

```
# Build the Logistic Regression model with a list of features...
#Select your own list here (perfectly valid to select all features too)
logitmod <- glm(actual ~ culture + male + book1 +
                  isMarried + boolDeadRelations + isPopular + popularity,
                family = binomial(link="logit"), data=trainingData)

summary(logitmod)
```

```
##
## Call:
## glm(formula = actual ~ culture + male + book1 + isMarried + boolDeadRelations +
##     isPopular + popularity, family = binomial(link = "logit"),
##     data = trainingData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5371  -1.1490   0.2287   1.0869   2.2513
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        0.774944   0.159721   4.852 1.22e-06 ***
## culture           -0.024939   0.006941  -3.593 0.000327 ***
## male              -0.535808   0.172341  -3.109 0.001877 **
## book1             -0.392157   0.201892  -1.942 0.052087 .
## isMarried          0.150902   0.234400   0.644 0.519719
## boolDeadRelations -0.539806   0.361703  -1.492 0.135595
## isPopular          0.422893   0.630152   0.671 0.502159
## popularity        -1.767655   1.136614  -1.555 0.119900
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 959.32  on 691  degrees of freedom
## Residual deviance: 898.90  on 684  degrees of freedom
## AIC: 914.9
##
## Number of Fisher Scoring iterations: 4
```

```
predicted <- plogis(predict(logitmod, testData))  # predicted scores
#print(predicted)
```

The logistic model predicted all the test instance characters as dead.
It has an FPR =0.118 (FP = 149 ANF TN= 1104).
The confusion matrix is printed in the below cells.

Step 2: Decide on the Optimal Prediction Probability Cutoff

The default cutoff prediction probability score is 0.5 or the ratio of 1's and 0's in the training data. But sometimes, tuning the probability cutoff can improve the accuracy in both the training and test samples. Compute the optimal cutoff score (using the test split) that minimizes the misclassification error for the trained model.

Hint: Use a function from the InformationValue library to perform this task.

```
#The default cut-off, as suggested by Tanish is 0.5
#What is the optimal cutoff?
#You can use the following lines of code for this:
library(InformationValue)
optCutOff <- optimalCutoff(testData$actual, predicted)[1]
optCutOff
```

```
## [1] 0.051273
```

The optimum cutoff value is 0.051273.

Solution 5

Using the optimal cutoff probability, compute and print the following using the predictions on the test set:

1. Misclassification error
2. Confusion Matrix
3. Sensitivity
4. Specificity

Plot the ROC Curve (Receiver Operating Characteristics Curve). What is the area under the curve?

Hint: Use predefined functions for this problem.

```
# Build the Logistic Regression model with a list of features...

logitmod <- glm(actual ~ culture + male + book1 +
                  isMarried + boolDeadRelations + isPopular + popularity,
                family = binomial(link="logit"), data=trainingData)

summary(logitmod)
```

```
##
## Call:
## glm(formula = actual ~ culture + male + book1 + isMarried + boolDeadRelations +
##     isPopular + popularity, family = binomial(link = "logit"),
##     data = trainingData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5371  -1.1490   0.2287   1.0869   2.2513
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        0.774944   0.159721   4.852 1.22e-06 ***
## culture           -0.024939   0.006941  -3.593 0.000327 ***
## male              -0.535808   0.172341  -3.109 0.001877 **
## book1             -0.392157   0.201892  -1.942 0.052087 .
## isMarried          0.150902   0.234400   0.644 0.519719
## boolDeadRelations -0.539806   0.361703  -1.492 0.135595
## isPopular          0.422893   0.630152   0.671 0.502159
## popularity        -1.767655   1.136614  -1.555 0.119900
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 959.32  on 691  degrees of freedom
## Residual deviance: 898.90  on 684  degrees of freedom
## AIC: 914.9
##
## Number of Fisher Scoring iterations: 4
```

```
predicted <- plogis(predict(logitmod, testData))  # predicted scores


library(InformationValue)
optCutOff <- optimalCutoff(testData$actual, predicted)[1]
optCutOff
```

```
## [1] 0.051273
```

```
misClassError(testData$actual, predicted, threshold = optCutOff)
```

```
## [1] 0.1196
```

```
sensitivity(testData$actual, predicted, threshold = optCutOff)
```

```
## [1] 0.999095
```

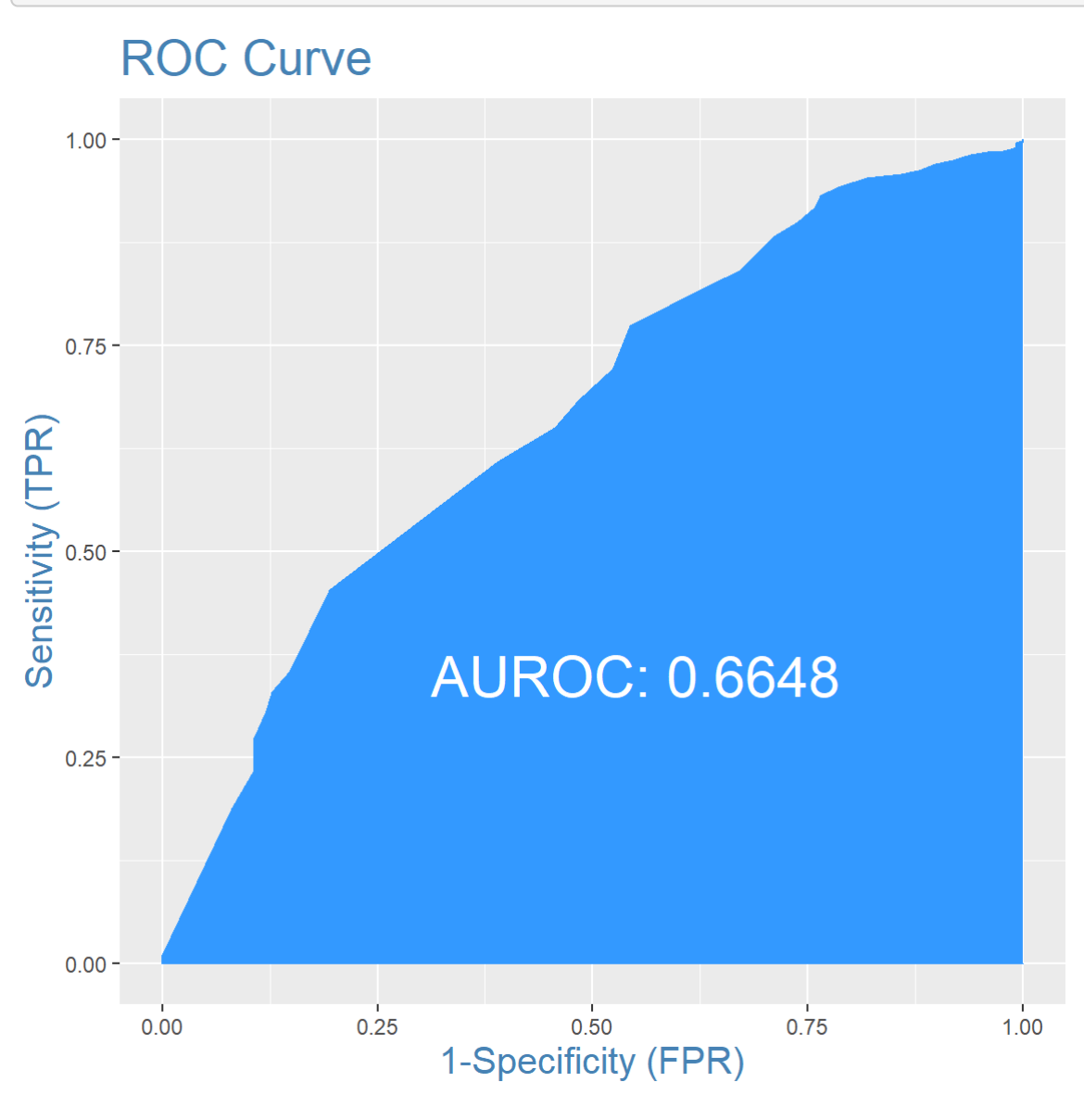
```
specificity(testData$actual, predicted, threshold = optCutOff)
```

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

```
confusionMatrix(testData$actual, predicted, threshold = optCutOff)
```

```
##     0    1
## 0   0    1
## 1 149 1104
```

```
#Plot the RoC curve and report the AUC
plotROC(testData$actual, predicted)
```



The significant attributes are book culture, male, book1, isMarried, boolDeadRelations, isPopular, popularity. The AUROC (Area under the curve) value is 0.6648.