AnalyticsModeling_HW7

Fall 2024

Question 10.1

Using the same crime data set, find the best model you can using a (a) Regression Tree Model and a (b) Random Forest Model.

The initial regression tree model split on the variables `Po1`, `Pop`, `LF`, and `NW` and ultimately ended up with seven leaves.

Since the original data set is very small, overfitting is likely occurring here. I used cross validation to evaluate the data to find the optimal level of tree complexity. While the initial model's complexity of seven leaves performs better than most other possible number of leaves, it looks like there is some potential with a model with only two leaves.

I also decided to check the R2 values of each potential model. Based off this, it appears there may be some potential in a model with six leaves.

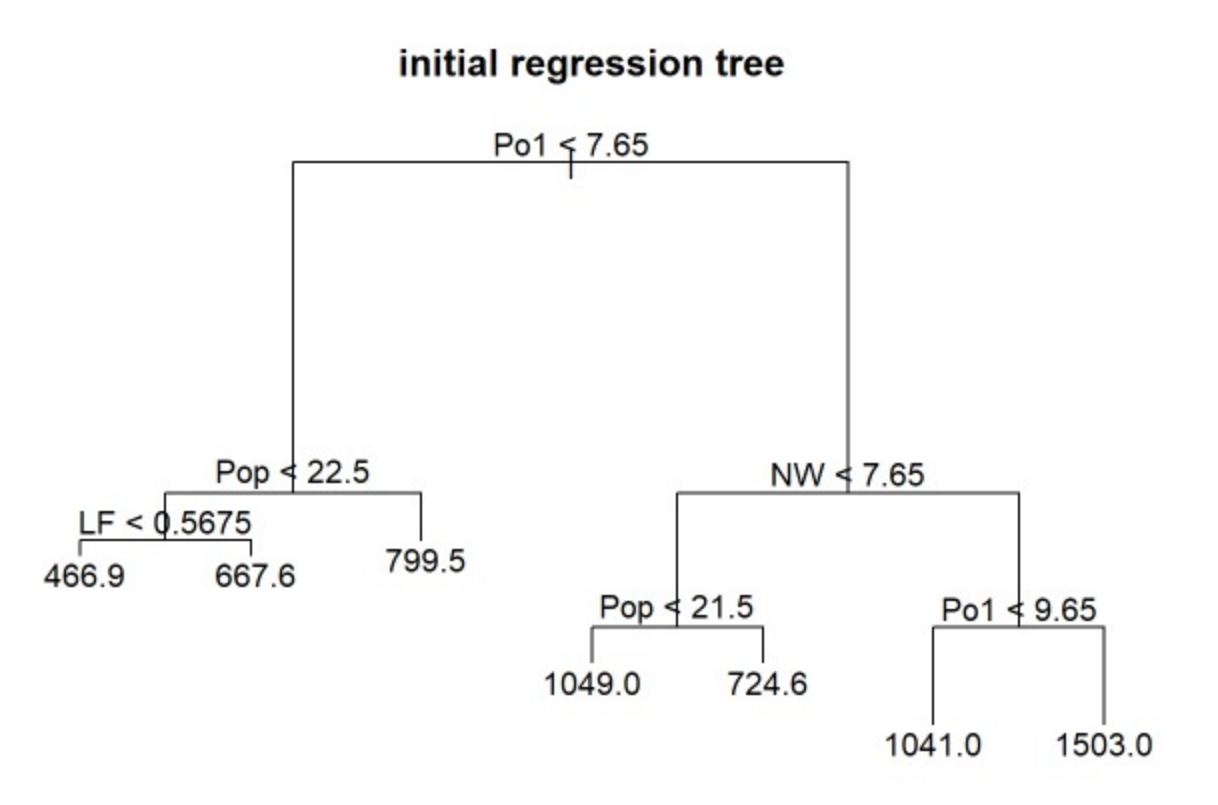
```
# set seed for reproducibility
set.seed(42)

# load data
crime_df <- read.table("uscrime.txt",  stringsAsFactors = FALSE, header = TRUE)
tail(crime_df, 5)
```

```
##       M So   Ed  Po1 Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 43 16.2  1  9.9  7.5 7.0 0.522  99.6  40 20.8 0.073 2.7   4960 22.4 0.054902
## 44 13.6  0 12.1  9.5 9.6 0.574 101.2  29  3.6 0.111 3.7   6220 16.2 0.028100
## 45 13.9  1  8.8  4.6 4.1 0.480  96.8  19  4.9 0.135 5.3   4570 24.9 0.056202
## 46 12.6  0 10.4 10.6 9.7 0.599  98.9  40  2.4 0.078 2.5   5930 17.1 0.046598
## 47 13.0  0 12.1  9.0 9.1 0.623 104.9   3  2.2 0.113 4.0   5880 16.0 0.052802
##       Time Crime
## 43 31.9989   823
## 44 30.0001  1030
## 45 32.5996   455
## 46 16.6999   508
## 47 16.0997   849
```

```
# train regression tree model
initialTreeModel <- tree(Crime~.            # response~.predictors
                         , data = crime_df
                         )
summary(initialTreeModel)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = crime_df)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

```
# graph the tree
plot(initialTreeModel)
text(initialTreeModel)
title("initial regression tree")
```




```
# let's look at how the branching was done
initialTreeModel$frame
```

```
##       var  n        dev      yval splits.cutleft splits.cutright
## 1     Po1 47 6880927.66  905.0851          <7.65           >7.65
## 2     Pop 23  779243.48  669.6087          <22.5           >22.5
## 4      LF 12  243811.00  550.5000        <0.5675         >0.5675
## 8  <leaf>  7   48518.86  466.8571
## 9  <leaf>  5   77757.20  667.6000
## 5  <leaf> 11  179470.73  799.5455
## 3      NW 24 3604162.50 1130.7500          <7.65           >7.65
## 6     Pop 10  557574.90  886.9000          <21.5           >21.5
## 12 <leaf>  5  146390.80 1049.2000
## 13 <leaf>  5  147771.20  724.6000
## 7     Po1 14 2027224.93 1304.9286          <9.65           >9.65
## 14 <leaf>  6  170828.00 1041.0000
## 15 <leaf>  8 1124984.88 1502.8750
```

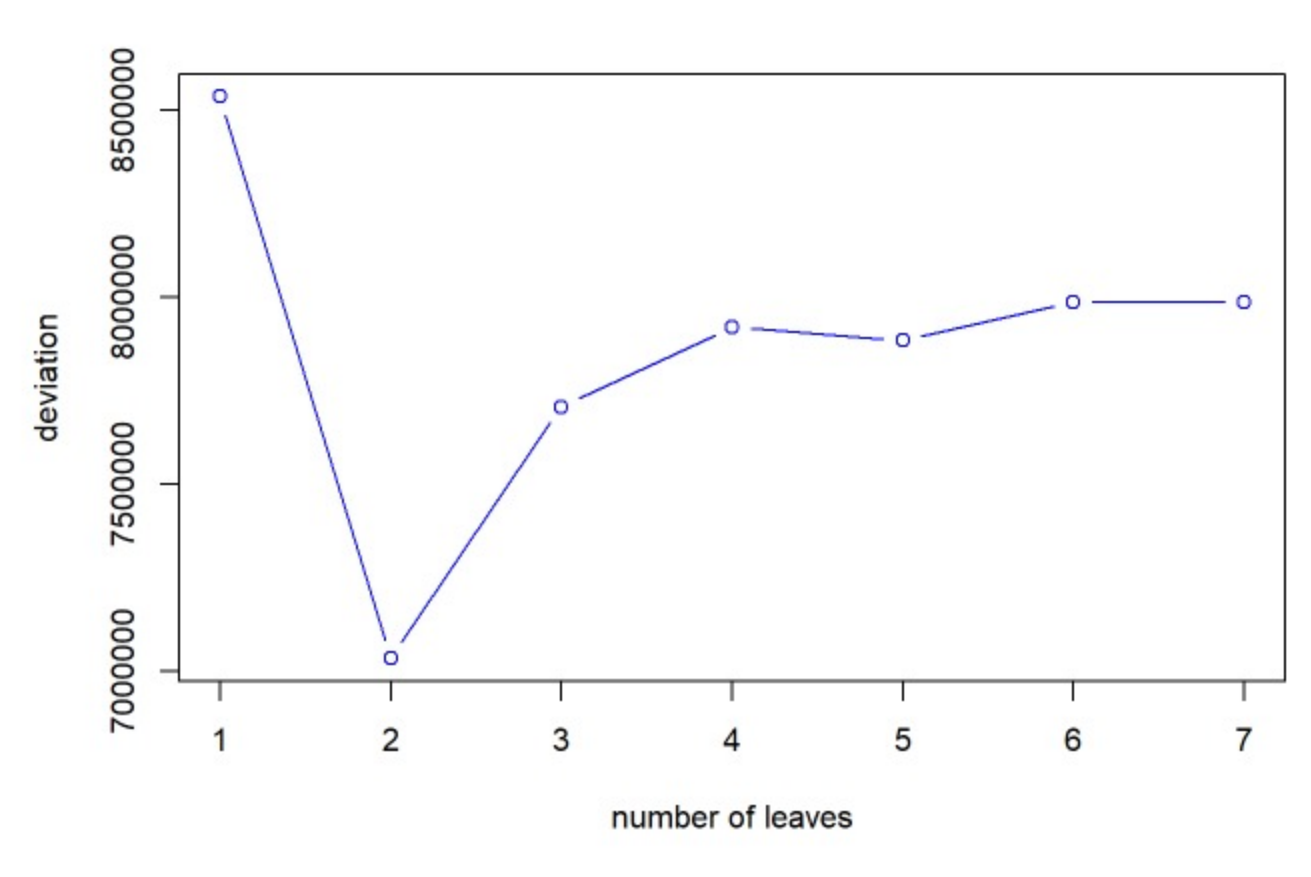
```
# calculate R2 of initial model
initialTreePreds <- predict(initialTreeModel, data = crime_df[,1:15])
RSS <- sum((initialTreePreds - crime_df[,16])^2)
TSS <- sum((crime_df[,16] - mean(crime_df[,16]))^2)
R2 <- 1-(RSS/TSS)
R2
```

```
## [1] 0.7244962
```

```
# perform cross validation to determine the optimal level of tree complexity
cv.TreeModel <- cv.tree(initialTreeModel)
cv.TreeModel
```

```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1] 7986688 7986688 7885099 7920849 7707536 7035494 8536032
##
## $k
## [1]      -Inf  117534.9  263412.9  355961.8  731412.1 1019362.7 2497521.7
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
# plot the deviance against the number of nodes
plot(cv.TreeModel$size, cv.TreeModel$dev, type = 'b', col = 'blue', xlab = "number of leaves", ylab = "deviation")
```




```
# let's calculate the R2 scores for each model with leaves between 2 and 7
R2_vector <- vector()

for (leaves in 2:8) {
  # prune the initial tree model
  pruned.TreeModel <- prune.tree(initialTreeModel
                                 , best = leaves)

  # get the predictions
  TreePreds <- predict(pruned.TreeModel, data = crime_df[,1:15])

  # calculate R2
  RSS <- sum((TreePreds - crime_df[,16])^2)
  TSS <- sum((crime_df[,16] - mean(crime_df[,16]))^2)
  R2 <- 1-(RSS/TSS)

  # store each model's R2 in this vector
  R2_vector <- c(R2_vector, R2)
}
```

```
## Warning in prune.tree(initialTreeModel, best = leaves): best is bigger than
## tree size
```

```
cat("Max R2 achieved: ", max(R2_vector), "\n")
```

```
## Max R2 achieved:  0.7244962
```

```
cat("Best performing number of leaves: ", which.max(R2_vector))
```

```
## Best performing number of leaves:  6
```

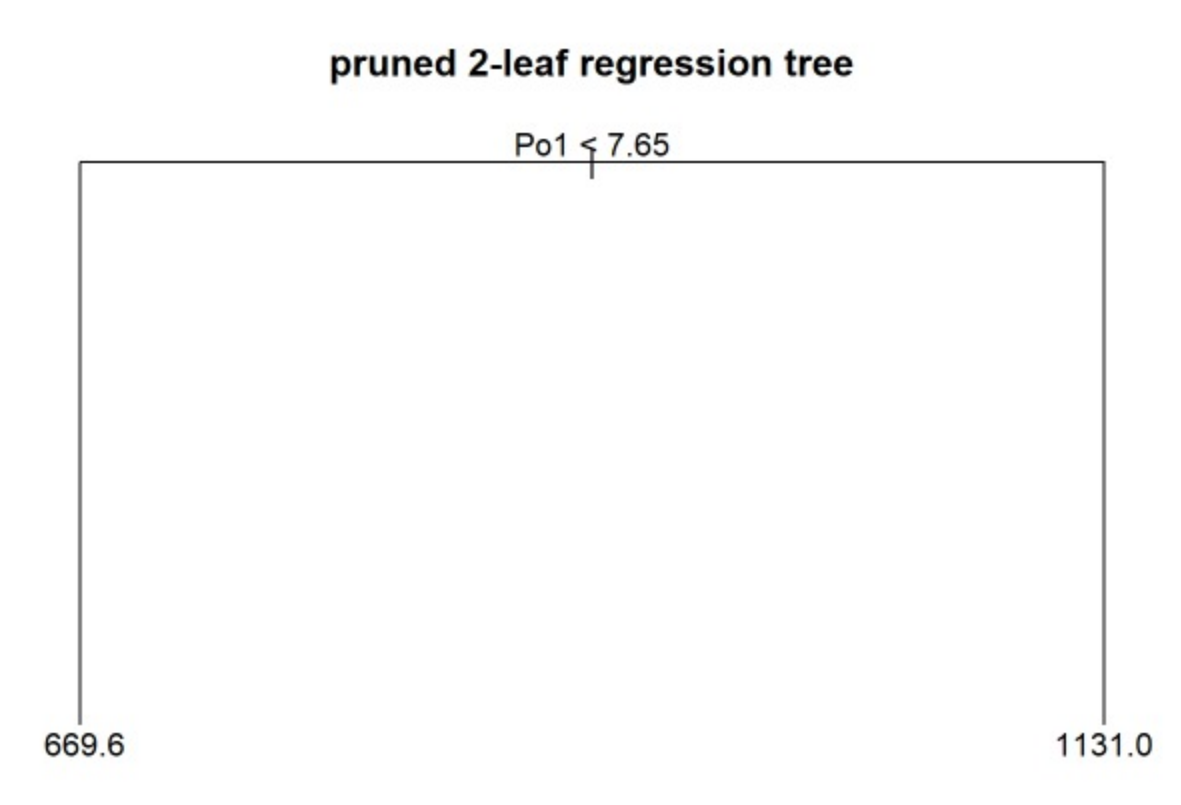
```
R2_vector
```

```
## [1] 0.3629629 0.5111061 0.6174017 0.6691333 0.7074149 0.7244962 0.7244962
```

```
# prune initial tree model down to 2 leaves
pruned2.TreeModel <- prune.tree(initialTreeModel
                                , best = 2)
summary(pruned2.TreeModel)
```

```
##
## Regression tree:
## snip.tree(tree = initialTreeModel, nodes = 2:3)
## Variables actually used in tree construction:
## [1] "Po1"
## Number of terminal nodes:  2
## Residual mean deviance:  97410 = 4383000 / 45
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -622.800 -193.200   -5.609    0.000  147.300  862.200
```

```
# graph the tree
plot(pruned2.TreeModel)
text(pruned2.TreeModel)
title("pruned 2-leaf regression tree")
```




```
# let's look at how the branching was done
pruned2.TreeModel$frame
```

```
##      var  n       dev      yval splits.cutleft splits.cutright
## 1    Po1 47 6880927.7  905.0851          <7.65           >7.65
## 2 <leaf> 23  779243.5  669.6087
## 3 <leaf> 24 3604162.5 1130.7500
```

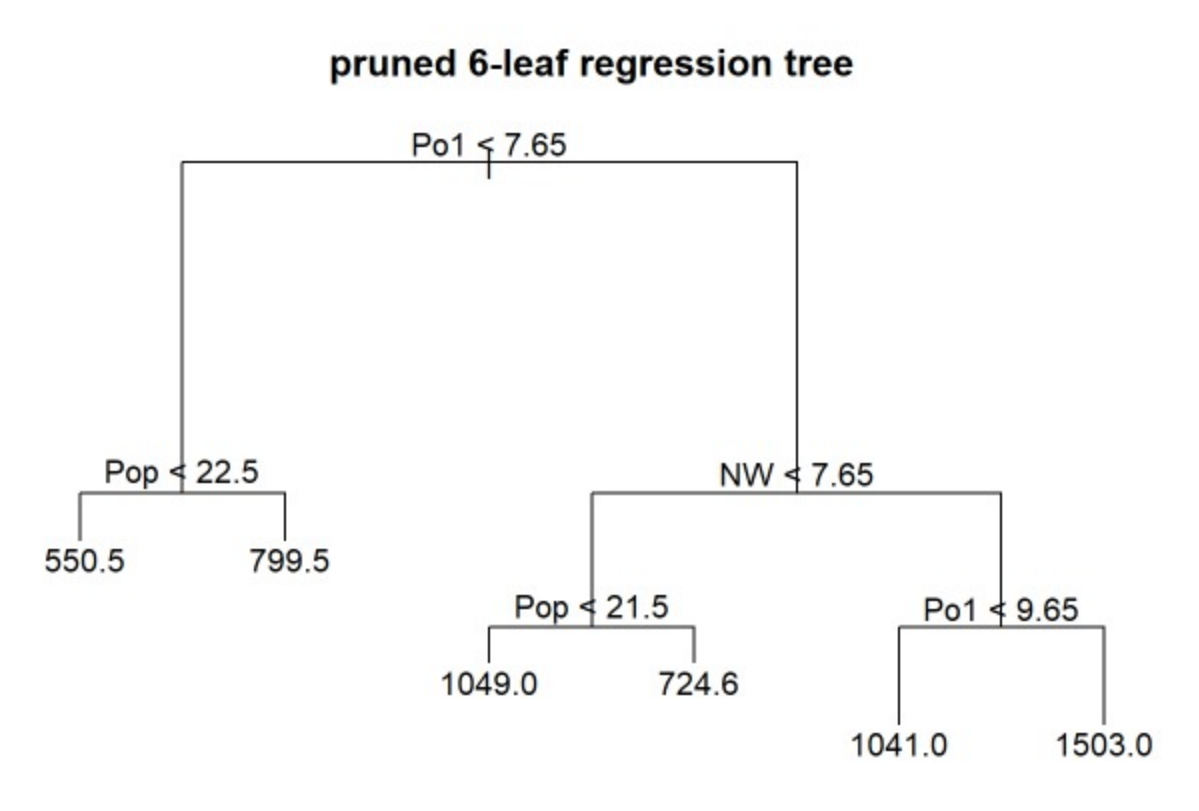
```
# R2 score
R2_vector[1]
```

```
## [1] 0.3629629
```

```
# prune initial tree model down to 6 leaves
pruned6.TreeModel <- prune.tree(initialTreeModel
                                , best = 6)
summary(pruned6.TreeModel)
```

```
##
## Regression tree:
## snip.tree(tree = initialTreeModel, nodes = 4L)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "NW"
## Number of terminal nodes:  6
## Residual mean deviance:  49100 = 2013000 / 41
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -99.520   -1.545    0.000  122.800  490.100
```

```
# graph the tree
plot(pruned6.TreeModel)
text(pruned6.TreeModel)
title("pruned 6-leaf regression tree")
```




```
# let's look at how the branching was done
pruned6.TreeModel$frame
```

```
##       var  n       dev      yval splits.cutleft splits.cutright
## 1     Po1 47 6880927.7  905.0851          <7.65           >7.65
## 2     Pop 23  779243.5  669.6087          <22.5           >22.5
## 4  <leaf> 12  243811.0  550.5000
## 5  <leaf> 11  179470.7  799.5455
## 3      NW 24 3604162.5 1130.7500          <7.65           >7.65
## 6     Pop 10  557574.9  886.9000          <21.5           >21.5
## 12 <leaf>  5  146390.8 1049.2000
## 13 <leaf>  5  147771.2  724.6000
## 7     Po1 14 2027224.9 1304.9286          <9.65           >9.65
## 14 <leaf>  6  170828.0 1041.0000
## 15 <leaf>  8 1124984.9 1502.8750
```

```
# R2 score
R2_vector[6]
```

```
## [1] 0.7244962
```

The R^2 of the initial model was calculated as .7245, meaning that 72.45% of the variance in the initial model could be explained by the data. Meanwhile, the R^2 of the pruned 2-leaf model dropped all the way down to .363, a big drop in model quality. While the initial model's residual mean deviance is lower than the pruned 6-leaf model, they have equal R2 values. At this point, I think the best regression tree model is the initial 7-leaf model.

Now, moving on to the Random Forest model:

The original data set has a total of 47 rows and a good rule of thumb is that each leaf ought to have at least 5% of the original data set within it. In this case, that is at least 3 data points in each leaf (rounded up). I will use that as the minimum node size.

Per this week's lectures, we should randomly choose a small number of factors and select the best factor within that set to branch on and a rule of thumb for the number of factors to use for this set is `1+log(n)`, where `n` is the number of factors. We have 15 factors in this data set.

The R^2 of the initial Random Forest model is much lower than the initial Regression Tree model - only 0.4253. I tried a variety of number of trees to see if I could improve the model. Based off this, the best performing model used a forest of 250 trees and had an R^2 score of 0.4482. This is still a big drop in quality from the Regression Tree model, so it's likely that the single Regression Tree method works better for this specific data set and problem.

```
# set seed for reproducibility
set.seed(42)

# get the number of factors on which to branch (1 + log(n))
num_factors <- ceiling(1+log(ncol(crime_df)-1))

# get minimum number of nodes
min_nodes <- ceiling(nrow(crime_df)*.05)

# create initial random forest model
initialRandomForest <- randomForest(Crime~.            # response~.predictors
                                    , data = crime_df
                                    , importance = TRUE
                                    , nodesize = min_nodes
                                    , mtry = num_factors
                                    , ntrees = 100
                                    )

# get predictions
rfTreePreds <- predict(initialRandomForest, data = crime_df[,1:15])

# calculate R2
RSS <- sum((rfTreePreds - crime_df[,16])^2)
TSS <- sum((crime_df[,16] - mean(crime_df[,16]))^2)
R2 <- 1-(RSS/TSS)
R2
```

```
## [1] 0.4302088
```

```
# let's try adjusting the number of trees to see if we can get a better performing model.
num_trees <- c(50, 100, 250, 500, 750, 900, 1000, 1500)
R2_vector <- vector()

for (trees in num_trees) {
  RandomForest <- randomForest(Crime~.            # response~.predictors
                               , data = crime_df
                               , importance = TRUE
                               , nodesize = min_nodes
                               , mtry = num_factors
                               , ntrees = trees
                               )
  # get predictions
  TreePreds <- predict(RandomForest, data = crime_df[,1:15])

  # calculate R2
  RSS <- sum((TreePreds - crime_df[,16])^2)
  TSS <- sum((crime_df[,16] - mean(crime_df[,16]))^2)
  R2 <- 1-(RSS/TSS)

  # store R2 scores in this vector
  R2_vector <- c(R2_vector, R2)
}

cat("Max R2 achieved: ", max(R2_vector), "\n")
```

```
## Max R2 achieved:  0.4482842
```

```
cat("Best performing number of trees: ", which.max(R2_vector), "\n")   # trees = 250
```

```
## Best performing number of trees:  3
```

```
R2_vector
```

```
## [1] 0.4196482 0.4266280 0.4482842 0.4365964 0.4279300 0.4461334 0.4171485
## [8] 0.4121576
```

Question 10.2

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.

At my job working for a gaming company, we want to know the likelihood that a customer is a "seasonal" gamer (i.e., they play seasonally based on what sporting events are ongoing) or if they are an inactive or lost customer. Logistic regression would be useful for estimating this likelihood and I could use predictors such as how their betting history is divided into sports (if they only bet, say, football and no other sports as opposed to a player who bets on every kind of sport), if they usually go dormant during specific months or if they play year-round historically, and how long it has been since their last bet.

Question 10.2

Using the GermanCredit data set, use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors and their coefficients), the software output, and the quality of fit.

This data set has 20 features and the target variable is binary with `1 = good` and `2 = bad`. There are 700 `good` data points and 300 `bad` data points, for a total of 1000 data points. Since we are doing logistic regression on this data, we need the final output to be between 0 and 1, so we have to relabel the target variable with `good = 0` and `bad = 1`.

To measure the effectiveness of this model, we need to use a confusion matrix to then calculate evaluation metrics like `accuracy = (TP+TN)/(TP+TN+FP+FN)`. We can also use an ROC curve and get from that the area under the curve to get a quick estimate of model quality. In this case, the initial model's AUC is 0.6891, which isn't bad. An AUC of 0.5 is equivalent to random guessing, so we're at least doing better than random guessing at this point. The accuracy of this initial model came out to be 0.73, which isn't bad at all.

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 is five times worse than incorrectly classifying a good customer. Determine a good threshold probability based on your model.

After testing several different thresholds, I found that the lower threshold could improve accuracy, but it would also more bad customers to be incorrectly identified as good ones. Thus a higher threshold seems pertinent.

Since the cost of incorrectly identifying a bad customer (x) is five times worse than incorrectly identifying a good customer (y), `x = 5y`. To lower the risk of misclassifying bad customers, we need to increase the threshold. I ended up at threshold = 0.75, which only had ten bad customer misclassifications and an accuracy of nearly 73%.

```
# set seed for reproducibility
set.seed(42)

# load data
credit_df <- read.table("germancredit.txt", header = FALSE)

# relabel target variable
credit_df$V21[credit_df$V21 == 1] <- 0
credit_df$V21[credit_df$V21 == 2] <- 1

# division of good and bad data points
table(credit_df$V21)
```

```
##
##   0   1
## 700 300
```

```
# split data into training and validation sets using the caret package
credit_split <- createDataPartition(credit_df$V21   # target variable
                                    , times = 1     # 1 partition
                                    , p = 0.7       # 70% into training set
                                    , list=FALSE    # results into matrix form
                                    )
trainSet <- credit_df[credit_split,]
validationSet <- credit_df[-credit_split,]

# the division of good and bad data points in training and validation sets
table(trainSet$V21)
```

```
##
##   0   1
## 489 211
```

```
table(validationSet$V21)
```

```
##
##   0   1
## 211  89
```

```
# let's train a logistic regression model on the training set
logisticRegressionModel <- glm(V21~.
                               , data = trainSet
                               , family = binomial(link = "logit")
                               )
summary(logisticRegressionModel)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = trainSet)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  8.689e-01  1.376e+00   0.631 0.527858
## V1A12       -2.942e-01  2.681e-01  -1.097 0.272551
## V1A13       -7.321e-01  4.206e-01  -1.741 0.081707 .
## V1A14       -1.987e+00  2.984e-01  -6.658 2.77e-11 ***
## V2           3.191e-02  1.132e-02   2.818 0.004833 **
## V3A31        6.835e-01  6.917e-01   0.988 0.323066
## V3A32       -5.986e-02  5.478e-01  -0.109 0.912976
## V3A33       -6.697e-01  5.992e-01  -1.118 0.263655
## V3A34       -1.032e+00  5.415e-01  -1.905 0.056734 .
## V4A41       -1.621e+00  4.697e-01  -3.451 0.000558 ***
## V4A410      -2.639e+00  1.181e+00  -2.235 0.025445 *
## V4A42       -5.784e-01  3.268e-01  -1.770 0.076703 .
## V4A43       -8.445e-01  3.095e-01  -2.728 0.006366 **
## V4A44       -1.346e+01  6.838e+02  -0.020 0.984293
## V4A45       -1.459e+00  8.972e-01  -1.627 0.103821
## V4A46        3.105e-01  5.022e-01   0.618 0.536317
## V4A48       -2.024e+00  1.260e+00  -1.606 0.108255
## V4A49       -3.316e-01  3.960e-01  -0.837 0.402448
## V5           1.096e-04  5.551e-05   1.974 0.048327 *
## V6A62       -5.254e-01  3.613e-01  -1.454 0.145865
## V6A63       -3.848e-01  5.007e-01  -0.769 0.442181
## V6A64       -2.777e+00  8.749e-01  -3.174 0.001503 **
## V6A65       -1.144e+00  3.340e-01  -3.426 0.000612 ***
## V7A72       -3.952e-01  5.369e-01  -0.736 0.461679
## V7A73       -5.290e-01  5.154e-01  -1.026 0.304681
## V7A74       -1.208e+00  5.630e-01  -2.145 0.031932 *
## V7A75       -5.486e-01  5.310e-01  -1.033 0.301532
## V8           3.338e-01  1.088e-01   3.068 0.002157 **
## V9A92       -4.902e-01  4.581e-01  -1.070 0.284553
## V9A93       -7.917e-01  4.492e-01  -1.763 0.077972 .
## V9A94       -3.838e-01  5.514e-01  -0.696 0.486459
## V10A102      1.088e+00  5.342e-01   2.037 0.041604 *
## V10A103     -6.461e-01  5.121e-01  -1.262 0.207078
## V11         -7.403e-03  1.055e-01  -0.070 0.944032
## V12A122     -8.013e-02  3.200e-01  -0.250 0.802303
## V12A123      1.771e-01  2.925e-01   0.606 0.544824
## V12A124      9.151e-01  5.103e-01   1.793 0.072935 .
## V13         -3.247e-02  1.197e-02  -2.712 0.006688 **
## V14A142     -1.778e-01  5.202e-01  -0.342 0.732538
## V14A143     -5.615e-01  3.105e-01  -1.808 0.070538 .
## V15A152     -4.190e-01  2.847e-01  -1.472 0.141030
## V15A153     -4.381e-01  5.827e-01  -0.752 0.452157
## V16          2.651e-01  2.508e-01   1.057 0.290634
## V17A172      6.370e-01  8.983e-01   0.709 0.478261
## V17A173      2.650e-01  8.653e-01   0.306 0.759417
## V17A174      4.500e-01  8.674e-01   0.519 0.603910
## V18          3.416e-01  3.189e-01   1.071 0.284006
## V19A192     -4.801e-02  2.518e-01  -0.191 0.848782
## V20A202     -1.094e+00  6.894e-01  -1.587 0.112610
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 856.90  on 699  degrees of freedom
## Residual deviance: 595.59  on 651  degrees of freedom
## AIC: 693.59
##
## Number of Fisher Scoring iterations: 14
```

```
# Let's make predictions on the validation data
yhat <- predict(logisticRegressionModel
                , newdata = validationSet[,-21]
                , type = "response"            # "response" ensures we don't get log-odd predictions
                )

# calculate ROC curve
roc(validationSet$V21, round(yhat))   # round yhat to get binary predictions
```

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

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

```
##
## Call:
## roc.default(response = validationSet$V21, predictor = round(yhat))
##
## Data: round(yhat) in 211 controls (validationSet$V21 0) < 89 cases (validationSet$V21 1).
## Area under the curve: 0.6891
```

```
# set the threshold
threshold <- 0.75
yhat_threshold <- as.integer(yhat > threshold)

# create confusion matrix
confusion_matrix <- as.matrix(table(yhat_threshold,validationSet$V21))
names(dimnames(confusion_matrix)) <- c("predicted", "observed")
confusion_matrix
```

```
##          observed
## predicted   0   1
##         0 201  72
##         1  10  17
```

```
# calculate accuracy = (TP+TN)/(TP+TN+FP+FN)
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) /(confusion_matrix[1,1] + confusion_matrix[2,2] + (confusion_mat
rix[1,2] + confusion_matrix[2,1]))
accuracy
```

```
## [1] 0.7266667
```

```
# check false positives and false negatives
fn <- confusion_matrix[1,2]
fp <- confusion_matrix[2,1]
fn
```

```
## [1] 72
```

```
fp
```

```
## [1] 10
```