ISYE 6501 Homework 7

2023-10-02

Question 10.1

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).

Regression Tree Model

For the purpose of this assignment, I did not split the crime_data data frame into training and test sets. Because the data frame is so small, splitting it into two parts may not leave enough information to effectively train and test the model.

```
#load the data
crime_data <- read.table("http://www.statsci.org/data/general/uscrime.txt", header=TRUE)
```

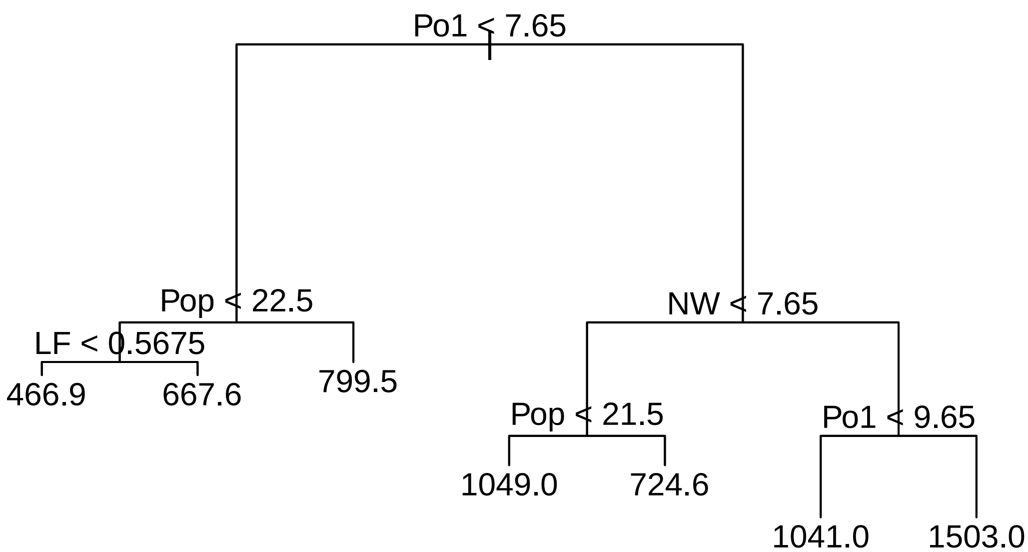
I fit a regression tree model using the crime_data data frame. The visualization below shows the model's node information and splitting criterion. The regression tree has 7 terminal nodes and uses 4 variables in model construction: Po1, Pop, LF, and NW. Since Po1 is the most important variable, it is used as the first split.

Qualitative Takeaway 1: The model is fairly easy to read, but could be better. Reducing the number of nodes would make the model easier to interpret.

```
#create regression tree model
tree_model <- tree(Crime~., data=crime_data, method="anova")
summary(tree_model)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = crime_data, method = "anova")
## 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
```

```r
#visualize model
plot(tree_model)
text(tree_model)
```



I used the predict() function to obtain the model's fitted values. Using these values, I calculated the model's r-squared. An r-squared value of 0.72 indicates that the model accounts for 72% of the variability in the data.

```r
#make predictions
prediction <- predict(tree_model, newdata=crime_data)

#calculate r-squared
SSR <- sum((prediction - crime_data$Crime)^2)
SST <- sum((crime_data$Crime - mean(crime_data$Crime))^2)
R2 <- 1 - SSR/SST
R2
```

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

To ensure I was creating the best model, I performed cross-validation to identify the complexity parameter that minimizes error. As shown in the code block below, the optimal number of nodes to use in the tree model is 7 (the same as my previous model).

```r
#perform cv to determine complexity parameter that minimizes error
cv_results <- cv.tree(tree_model, FUN=prune.tree)
cv_results
```

```
## $size
## [1] 7 6 5 4 3 2 1
##
## $dev
## [1] 8990230 8895007 8894822 8736473 8758596 7040212 7656123
##
## $k
## [1]      -Inf  117534.9  263412.9  355961.8  731412.1 1019362.7 2497521.7
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
best_cp <- cv_results$size[which.min(cv_results$dev)]
best_cp
```

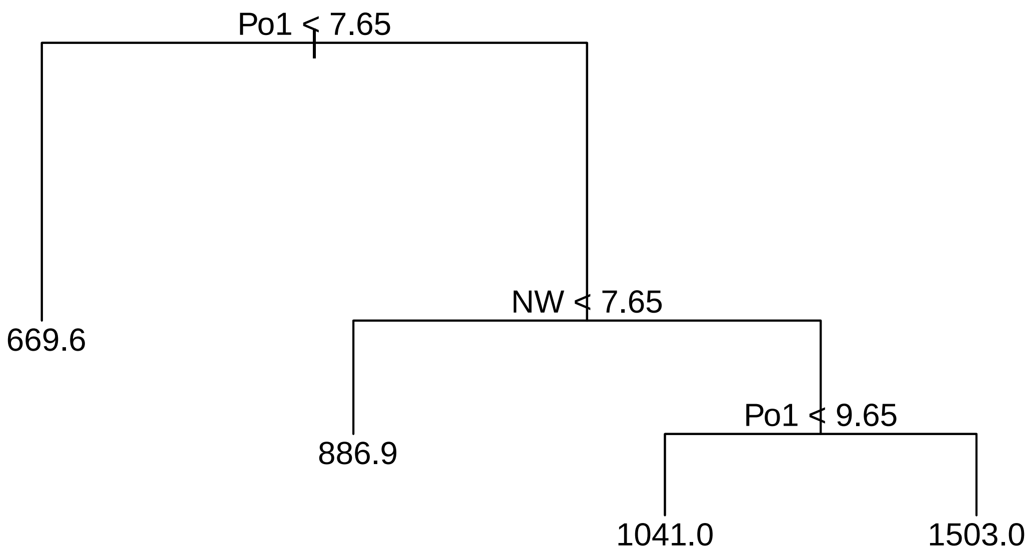
```
## [1] 2
```

Out of curiosity and to create a simpler model, I manually pruned my tree. The code block below uses the prune.tree() function to simplify my model into 4 nodes.

```r
#prune tree
tree_model_pruned <- prune.tree(tree_model, best = 4)
summary(tree_model_pruned)
```

```
##
## Regression tree:
## snip.tree(tree = tree_model, nodes = c(6L, 2L))
## Variables actually used in tree construction:
## [1] "Po1" "NW"
## Number of terminal nodes:  4
## Residual mean deviance:  61220 = 2633000 / 43
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -573.90 -152.60   35.39    0.00  158.90  490.10
```

```r
#visualize model
plot(tree_model_pruned)
text(tree_model_pruned)
```



Then, I calculated the model's r-squared. This model has an r-squared value of 0.62, meaning that it accounts for ~10% less variability in the data compared to my previous model.

Qualitative Takeaway 2: Although this model may not be as "good" as the previous model, it is much simpler. It may be preferred if having a simple, explainable model is of high importance.

```
#make predictions
pruned_prediction <- predict(tree_model_pruned, newdata=crime_data)

#calculate r-squared
pruned_SSR <- sum((pruned_prediction - crime_data$Crime)^2)
pruned_SST <- sum((crime_data$Crime - mean(crime_data$Crime))^2)
pruned_R2 <- 1 - pruned_SSR/pruned_SST
pruned_R2
```

```
## [1] 0.6174017
```

Random Forest Model

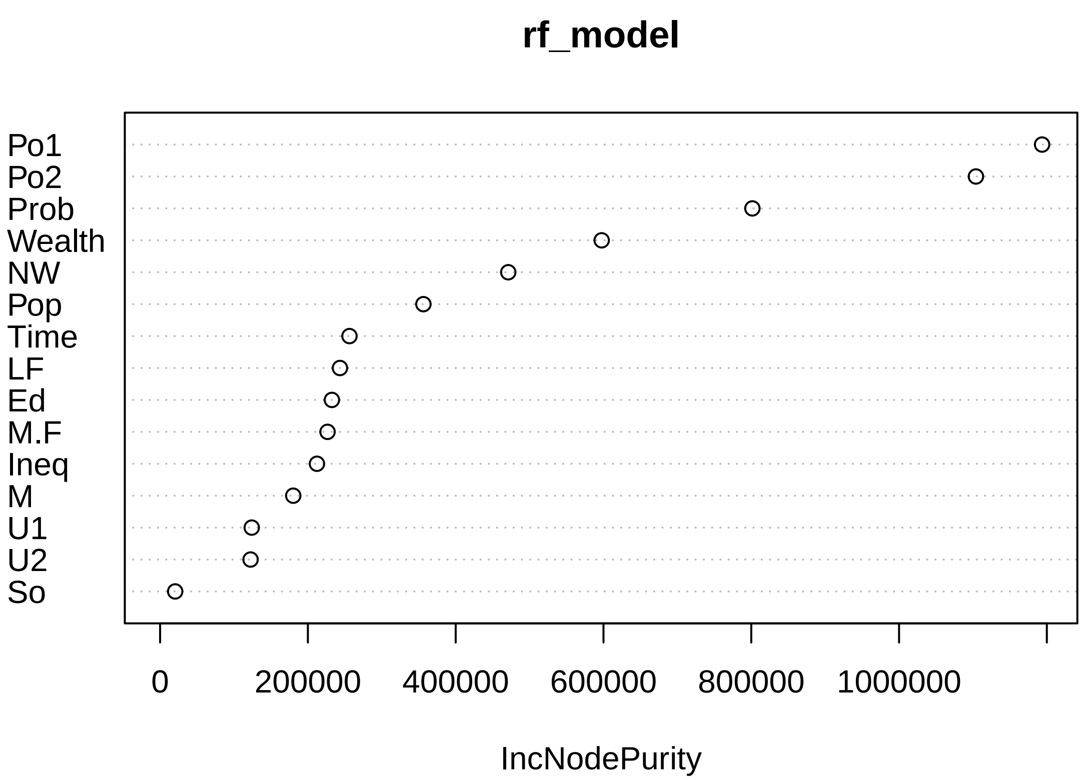
I fit a random forest model using the crime_data data frame. Then, I analyzed the importance of each variable. Similar to my regression tree above, Po1 is the most important variable.

```
#create random forest model
rf_model <- randomForest(Crime~., data=crime_data, ntree=100)
```

```r
#visualize importance of each variable
importance(rf_model)
```

```
##        IncNodePurity
## M          180166.88
## So          20404.28
## Ed         232496.27
## Po1       1193548.59
## Po2       1104008.24
## LF         243386.55
## M.F        226583.96
## Pop        356332.42
## NW         471169.21
## U1         124016.52
## U2         122441.35
## Wealth     597567.82
## Ineq       212269.43
## Prob       801494.51
## Time       256283.13
```

```r
varImpPlot(rf_model)
```



To evaluate how much variability my model explains, I calculated its r-squared value. The model has an r-squared value of 0.89.

Qualitative Takeaway 1: An r-squared value of 0.89 is very high and likely the result of overfitting. Overfitting is likely since I did not split my data into test and train sets.

```
#make predictions
prediction <- predict(rf_model, newdata=crime_data)

#calculate r-squared
SSR <- sum((prediction - crime_data$Crime)^2)
SST <- sum((crime_data$Crime - mean(crime_data$Crime))^2)
R2 <- 1 - SSR/SST
R2
```

```
## [1] 0.8912695
```

To create a simpler model, I selected the six features with the highest importance (Po1, Po2, Wealth, Prob, NW, and Pop) and refitted my random forest model.

```
#refit model using variables with highest importance
rf_model2 <- randomForest(Crime~Po1+Po2+Prob+Wealth+NW+Pop, data=crime_data, ntree=100)
```

Then, I calculated my r-squared value. The model's r-squared is about 0.88.

Qualitative Takeaway 2: This model has a slightly lower r-squared value, but it is not much different from my previous model. This further confirms that the variables removed from my second model were not of high importance.

```
#make predictions
prediction <- predict(rf_model2, newdata=crime_data)

#calculate r2
SSR <- sum((prediction - crime_data$Crime)^2)
SST <- sum((crime_data$Crime - mean(crime_data$Crime))^2)
R2 <- 1 - SSR/SST
R2
```

```
## [1] 0.8589887
```

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.

Logistic regression could be used to determine whether a patient has a particular medical condition. Predictors may include age, gender, DMI, family history, and blood pressure, among other relevant factors.

Question 10.3

Part 1

Using the GermanCredit data set germancredit.txt, 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.

I started by cleaning my data and splitting it into training and test sets.

```r
#load the data
german_data <- read.table("~/Z. OMSA/Intro to Analytics Modeling/Week 7 Homework/german.txt.data", quote

#change response to 0 (bad) and 1 (good)
german_data$V21[german_data$V21==1] <- 0
german_data$V21[german_data$V21==2] <- 1

#split data into training and test sets
set.seed(123)
sample <- sample(1:nrow(german_data), size=0.6*nrow(german_data))
train_data <- german_data[sample,]
test_data <- german_data[-sample,]
```

Then, I used my training data to fit a logistic regression model.

```r
#fit logistic regression model
logit_model <- glm(V21~., data=train_data, family="binomial")
summary(logit_model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = "binomial", data = train_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.151e+00  1.501e+00  -0.767 0.443166
## V1A12       -3.438e-01  2.991e-01  -1.150 0.250346
## V1A13       -1.321e+00  5.320e-01  -2.483 0.013011 *
## V1A14       -1.677e+00  3.213e-01  -5.217 1.82e-07 ***
## V2           1.733e-02  1.260e-02   1.376 0.168967
## V3A31        5.934e-01  7.917e-01   0.750 0.453528
## V3A32       -6.162e-01  6.488e-01  -0.950 0.342240
## V3A33       -5.384e-01  7.119e-01  -0.756 0.449486
## V3A34       -1.354e+00  6.554e-01  -2.066 0.038801 *
## V4A41       -1.923e+00  5.045e-01  -3.812 0.000138 ***
## V4A410      -1.657e+00  1.079e+00  -1.536 0.124626
## V4A42       -6.691e-01  3.571e-01  -1.874 0.060956 .
## V4A43       -6.018e-01  3.409e-01  -1.765 0.077544 .
## V4A44        2.239e-01  1.039e+00   0.215 0.829408
## V4A45        6.113e-01  7.534e-01   0.811 0.417172
## V4A46       -5.990e-01  5.665e-01  -1.057 0.290331
## V4A48       -1.330e+00  1.385e+00  -0.961 0.336700
## V4A49       -8.619e-01  4.669e-01  -1.846 0.064855 .
## V5           1.978e-04  5.986e-05   3.305 0.000951 ***
## V6A62       -6.761e-01  4.098e-01  -1.650 0.098988 .
## V6A63       -9.207e-01  5.640e-01  -1.632 0.102599
## V6A64       -1.373e+00  7.182e-01  -1.912 0.055913 .
## V6A65       -1.370e+00  3.756e-01  -3.648 0.000264 ***
```

```
## V7A72        1.690e+00  6.229e-01   2.713 0.006675 **
## V7A73        1.050e+00  5.960e-01   1.762 0.078033 .
## V7A74        2.126e-01  6.261e-01   0.340 0.734152
## V7A75        5.012e-01  5.965e-01   0.840 0.400765
## V8           3.874e-01  1.181e-01   3.281 0.001033 **
## V9A92       -1.130e-01  5.328e-01  -0.212 0.832071
## V9A93       -7.177e-01  5.233e-01  -1.372 0.170194
## V9A94       -2.217e-01  6.145e-01  -0.361 0.718221
## V10A102      9.879e-01  5.827e-01   1.695 0.090005 .
## V10A103     -1.086e+00  5.789e-01  -1.876 0.060611 .
## V11          8.045e-02  1.189e-01   0.677 0.498686
## V12A122      3.275e-01  3.485e-01   0.940 0.347424
## V12A123      5.903e-01  3.220e-01   1.834 0.066726 .
## V12A124      1.261e+00  6.961e-01   1.812 0.070013 .
## V13         -3.498e-03  1.214e-02  -0.288 0.773317
## V14A142      4.777e-02  6.074e-01   0.079 0.937311
## V14A143     -2.657e-01  3.335e-01  -0.797 0.425592
## V15A152     -4.751e-01  3.211e-01  -1.480 0.138933
## V15A153     -6.840e-01  7.444e-01  -0.919 0.358183
## V16          3.139e-01  2.779e-01   1.129 0.258742
## V17A172     -7.220e-01  8.539e-01  -0.846 0.397757
## V17A173     -5.794e-01  8.192e-01  -0.707 0.479396
## V17A174     -6.665e-01  8.259e-01  -0.807 0.419633
## V18          3.384e-01  3.359e-01   1.007 0.313780
## V19A192     -4.037e-01  2.822e-01  -1.431 0.152477
## V20A202     -8.599e-01  8.034e-01  -1.070 0.284485
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 727.88  on 599  degrees of freedom
## Residual deviance: 505.13  on 551  degrees of freedom
## AIC: 603.13
##
## Number of Fisher Scoring iterations: 5
```

I used a confusion matrix to estimate the performance of my model (NOTE: I used a classification threshold of p=0.5). As shown in the code block below, my model accuracy is ~73%.

```
#make predictions
prediction_prob <- predict(logit_model, newdata=test_data, type="response")
prediction <- as.integer(prediction_prob > 0.5)

#create confusion matrix
real_data <- as.factor(test_data$V21)
prediction <- as.factor(prediction)
cm <- confusionMatrix(real_data, prediction)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
```

```
##          0 236  41
##          1  64  59
##
##                Accuracy : 0.7375
##                  95% CI : (0.6915, 0.78)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 0.73916
##
##                   Kappa : 0.3498
##
##  Mcnemar's Test P-Value : 0.03179
##
##             Sensitivity : 0.7867
##             Specificity : 0.5900
##          Pos Pred Value : 0.8520
##          Neg Pred Value : 0.4797
##              Prevalence : 0.7500
##          Detection Rate : 0.5900
##    Detection Prevalence : 0.6925
##       Balanced Accuracy : 0.6883
##
##        'Positive' Class : 0
##
```

Then, I used R's step function to perform feature selection on my initial model. This function uses AIC to determine the most relevant variables to include in my model.

```
#choose a model by AIC in a stepwise algorithm
step(logit_model)
```

```
## Start:  AIC=603.13
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V17   3   506.01 598.01
## - V14   2   505.94 599.94
## - V13   1   505.21 601.21
## - V15   2   507.55 601.55
## - V11   1   505.59 601.59
## - V18   1   506.14 602.14
## - V12   3   510.17 602.17
## - V20   1   506.40 602.40
## - V16   1   506.40 602.40
## - V9    3   510.56 602.56
## - V2    1   507.03 603.03
## <none>      505.13 603.13
## - V19   1   507.20 603.20
## - V10   2   512.34 606.34
## - V4    9   526.90 606.90
## - V3    4   518.85 608.85
## - V7    4   522.14 612.14
## - V8    1   516.39 612.39
```

```
## - V5    1   516.56 612.56
## - V6    4   524.50 614.50
## - V1    3   540.75 632.75
##
## Step:  AIC=598.01
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V14 + V15 + V16 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V14   2   506.71 594.71
## - V13   1   506.16 596.16
## - V11   1   506.41 596.41
## - V15   2   508.51 596.51
## - V18   1   506.94 596.94
## - V20   1   507.32 597.32
## - V12   3   511.33 597.33
## - V9    3   511.36 597.36
## - V16   1   507.49 597.49
## <none>       506.01 598.01
## - V2    1   508.04 598.04
## - V19   1   508.35 598.35
## - V10   2   513.32 601.32
## - V4    9   528.31 602.31
## - V3    4   520.22 604.22
## - V8    1   517.02 607.02
## - V5    1   517.05 607.05
## - V7    4   523.44 607.44
## - V6    4   524.67 608.67
## - V1    3   542.01 628.01
##
## Step:  AIC=594.71
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V15 + V16 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V13   1   506.80 592.80
## - V15   2   509.00 593.00
## - V11   1   507.10 593.10
## - V18   1   507.70 593.70
## - V9    3   511.85 593.85
## - V20   1   507.87 593.87
## - V16   1   508.27 594.27
## - V12   3   512.32 594.32
## <none>       506.71 594.71
## - V2    1   508.86 594.86
## - V19   1   509.11 595.11
## - V10   2   513.80 597.80
## - V4    9   528.51 598.51
## - V5    1   517.55 603.55
## - V3    4   523.89 603.89
## - V8    1   517.95 603.95
## - V7    4   523.99 603.99
## - V6    4   524.99 604.99
## - V1    3   542.99 624.99
```

```
##
## Step:  AIC=592.8
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V15 + V16 + V18 + V19 + V20
##
##         Df Deviance    AIC
## - V11    1   507.16 591.16
## - V15    2   509.35 591.35
## - V18    1   507.77 591.77
## - V9     3   511.94 591.94
## - V20    1   507.98 591.98
## - V16    1   508.30 592.30
## - V12    3   512.42 592.42
## <none>       506.80 592.80
## - V2     1   509.03 593.03
## - V19    1   509.34 593.34
## - V10    2   513.89 595.89
## - V4     9   528.51 596.51
## - V5     1   517.58 601.58
## - V8     1   518.09 602.09
## - V3     4   524.10 602.10
## - V6     4   525.06 603.06
## - V7     4   525.11 603.11
## - V1     3   543.01 623.01
##
## Step:  AIC=591.16
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V12 +
##     V15 + V16 + V18 + V19 + V20
##
##         Df Deviance    AIC
## - V15    2   510.12 590.12
## - V18    1   508.14 590.14
## - V9     3   512.19 590.19
## - V20    1   508.35 590.35
## - V16    1   508.74 590.74
## - V12    3   512.79 590.79
## <none>       507.16 591.16
## - V2     1   509.40 591.40
## - V19    1   509.53 591.53
## - V10    2   514.13 594.13
## - V4     9   528.70 594.70
## - V5     1   517.80 599.80
## - V3     4   524.23 600.23
## - V8     1   518.64 600.64
## - V6     4   525.09 601.09
## - V7     4   525.15 601.15
## - V1     3   544.13 622.13
##
## Step:  AIC=590.12
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V12 +
##     V16 + V18 + V19 + V20
##
##         Df Deviance    AIC
## - V20    1   510.97 588.97
```

```
## - V18   1   511.16 589.16
## - V16   1   511.73 589.73
## - V2    1   512.05 590.05
## - V9    3   516.10 590.10
## <none>      510.12 590.12
## - V19   1   512.74 590.74
## - V12   3   517.41 591.41
## - V4    9   530.36 592.36
## - V10   2   517.14 593.14
## - V6    4   526.89 598.89
## - V8    1   520.93 598.93
## - V5    1   521.18 599.18
## - V3    4   527.43 599.43
## - V7    4   529.07 601.07
## - V1    3   548.98 622.98
##
## Step:  AIC=588.97
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V12 +
##     V16 + V18 + V19
##
##        Df Deviance    AIC
## - V18   1   511.96 587.96
## - V16   1   512.58 588.58
## - V9    3   516.82 588.82
## <none>      510.97 588.97
## - V2    1   513.14 589.14
## - V19   1   513.37 589.37
## - V12   3   518.28 590.28
## - V4    9   531.06 591.06
## - V10   2   518.63 592.63
## - V5    1   521.77 597.77
## - V6    4   527.99 597.99
## - V8    1   522.35 598.35
## - V3    4   528.80 598.80
## - V7    4   529.61 599.61
## - V1    3   549.69 621.69
##
## Step:  AIC=587.96
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V12 +
##     V16 + V19
##
##        Df Deviance    AIC
## - V9    3   516.91 586.91
## - V16   1   513.86 587.86
## <none>      511.96 587.96
## - V2    1   514.15 588.15
## - V19   1   514.37 588.37
## - V12   3   519.72 589.72
## - V4    9   532.21 590.21
## - V10   2   519.54 591.54
## - V5    1   522.34 596.34
## - V6    4   528.64 596.64
## - V8    1   522.80 596.80
## - V3    4   530.32 598.32
```

```
## - V7    4   530.90 598.90
## - V1    3   550.86 620.86
##
## Step:  AIC=586.91
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V10 + V12 + V16 +
##     V19
##
##        Df Deviance    AIC
## - V16   1   518.44 586.44
## <none>      516.91 586.91
## - V2    1   518.97 586.97
## - V19   1   519.68 587.68
## - V12   3   524.57 588.57
## - V4    9   536.93 588.93
## - V10   2   525.00 591.00
## - V8    1   526.11 594.11
## - V5    1   526.31 594.31
## - V6    4   533.85 595.85
## - V3    4   534.97 596.97
## - V7    4   540.06 602.06
## - V1    3   557.76 621.76
##
## Step:  AIC=586.44
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V10 + V12 + V19
##
##        Df Deviance    AIC
## - V2    1   520.36 586.36
## <none>      518.44 586.44
## - V19   1   521.06 587.06
## - V12   3   526.06 588.06
## - V4    9   538.62 588.62
## - V10   2   526.37 590.37
## - V8    1   527.43 593.43
## - V5    1   527.48 593.48
## - V3    4   535.12 595.12
## - V6    4   535.46 595.46
## - V7    4   540.67 600.67
## - V1    3   559.19 621.19
##
## Step:  AIC=586.36
## V21 ~ V1 + V3 + V4 + V5 + V6 + V7 + V8 + V10 + V12 + V19
##
##        Df Deviance    AIC
## <none>      520.36 586.36
## - V19   1   523.15 587.15
## - V4    9   540.08 588.08
## - V12   3   529.54 589.54
## - V10   2   527.97 589.97
## - V6    4   536.94 594.94
## - V8    1   531.55 595.55
## - V3    4   537.87 595.87
## - V7    4   542.71 600.71
## - V5    1   540.11 604.11
## - V1    3   561.37 621.37
```

```
##
## Call:  glm(formula = V21 ~ V1 + V3 + V4 + V5 + V6 + V7 + V8 + V10 +
##     V12 + V19, family = "binomial", data = train_data)
##
## Coefficients:
## (Intercept)       V1A12       V1A13       V1A14       V3A31       V3A32
##  -1.2676573  -0.4491763  -1.4781983  -1.7658864   0.3809179  -0.8785835
##       V3A33       V3A34       V4A41      V4A410       V4A42       V4A43
##  -0.7137824  -1.4832597  -1.7296963  -1.6489034  -0.5015803  -0.4881956
##       V4A44       V4A45       V4A46       V4A48       V4A49          V5
##   0.2555983   0.7259720  -0.3657147  -1.3419258  -0.5421971   0.0002119
##       V6A62       V6A63       V6A64       V6A65       V7A72       V7A73
##  -0.5648224  -0.8307852  -1.0611338  -1.2337563   1.6374862   0.8417286
##       V7A74       V7A75          V8     V10A102     V10A103     V12A122
##   0.0472212   0.3607149   0.3606261   1.0352347  -1.0254742   0.3226335
##     V12A123     V12A124     V19A192
##   0.7167595   1.1285705  -0.4224933
##
## Degrees of Freedom: 599 Total (i.e. Null);  567 Residual
## Null Deviance:       727.9
## Residual Deviance: 520.4     AIC: 586.4
```

Next, I refitted my model with features chosen by the step function.

```r
#fit model with significant features
logit_model2 <- glm(V21~V1+V3+V4+V5+V6+V7+V8+V10+V12+V19, data=train_data, family="binomial")
summary(logit_model2)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V3 + V4 + V5 + V6 + V7 + V8 + V10 +
##     V12 + V19, family = "binomial", data = train_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.268e+00  9.023e-01  -1.405 0.160053
## V1A12       -4.492e-01  2.827e-01  -1.589 0.112134
## V1A13       -1.478e+00  5.129e-01  -2.882 0.003949 **
## V1A14       -1.766e+00  3.117e-01  -5.665 1.47e-08 ***
## V3A31        3.809e-01  7.192e-01   0.530 0.596375
## V3A32       -8.786e-01  5.946e-01  -1.478 0.139490
## V3A33       -7.138e-01  6.756e-01  -1.057 0.290701
## V3A34       -1.483e+00  6.253e-01  -2.372 0.017696 *
## V4A41       -1.730e+00  4.808e-01  -3.597 0.000322 ***
## V4A410      -1.649e+00  1.093e+00  -1.508 0.131532
## V4A42       -5.016e-01  3.353e-01  -1.496 0.134634
## V4A43       -4.882e-01  3.209e-01  -1.521 0.128209
## V4A44        2.556e-01  1.004e+00   0.254 0.799129
## V4A45        7.260e-01  7.443e-01   0.975 0.329392
## V4A46       -3.657e-01  5.526e-01  -0.662 0.508096
## V4A48       -1.342e+00  1.351e+00  -0.993 0.320648
## V4A49       -5.422e-01  4.386e-01  -1.236 0.216411
## V5           2.119e-04  4.958e-05   4.273 1.92e-05 ***
```

```
## V6A62       -5.648e-01  3.948e-01  -1.431 0.152551
## V6A63       -8.308e-01  5.391e-01  -1.541 0.123277
## V6A64       -1.061e+00  6.729e-01  -1.577 0.114822
## V6A65       -1.234e+00  3.596e-01  -3.431 0.000602 ***
## V7A72        1.637e+00  5.321e-01   3.078 0.002086 **
## V7A73        8.417e-01  4.948e-01   1.701 0.088896 .
## V7A74        4.722e-02  5.375e-01   0.088 0.929998
## V7A75        3.607e-01  5.143e-01   0.701 0.483096
## V8           3.606e-01  1.104e-01   3.266 0.001091 **
## V10A102      1.035e+00  5.668e-01   1.827 0.067767 .
## V10A103     -1.025e+00  5.553e-01  -1.847 0.064779 .
## V12A122      3.226e-01  3.295e-01   0.979 0.327558
## V12A123      7.168e-01  3.077e-01   2.330 0.019826 *
## V12A124      1.129e+00  4.117e-01   2.741 0.006122 **
## V19A192     -4.225e-01  2.543e-01  -1.661 0.096696 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 727.88  on 599  degrees of freedom
## Residual deviance: 520.36  on 567  degrees of freedom
## AIC: 586.36
##
## Number of Fisher Scoring iterations: 5
```

Finally, I created another confusion matrix to evaluate the performance of my new model. As you can see below, this model is slightly less accurate than my previous model. However, it is also much simpler.

```r
#make predictions
prediction_prob <- predict(logit_model2, newdata=test_data, type="response")
prediction <- as.integer(prediction_prob > 0.5)

#create confusion matrix
real_data <- as.factor(test_data$V21)
prediction <- as.factor(prediction)
cm <- confusionMatrix(real_data, prediction)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 235  42
##          1  69  54
##
##                Accuracy : 0.7225
##                  95% CI : (0.6758, 0.7658)
##     No Information Rate : 0.76
##     P-Value [Acc > NIR] : 0.96355
##
##                   Kappa : 0.3061
##
##  Mcnemar's Test P-Value : 0.01359
```

```
##
##             Sensitivity : 0.7730
##             Specificity : 0.5625
##          Pos Pred Value : 0.8484
##          Neg Pred Value : 0.4390
##              Prevalence : 0.7600
##          Detection Rate : 0.5875
##    Detection Prevalence : 0.6925
##       Balanced Accuracy : 0.6678
##
##        'Positive' Class : 0
##
```

Part 2

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.

To answer this question, I created a function that given a threshold value, calculates the cost of the model's outcome.

```
#create function to calculate predicted cost based on CM output
cost <- function(threshold){
  prediction <- factor(as.integer(prediction_prob > threshold), levels=c(0,1))
  cm <- confusionMatrix(real_data, prediction)
  total_cost <- cm[["table"]][2]+(5*cm[["table"]][3])
  return(total_cost)
}
```

Next, I created a for loop that calculated the cost for 100 different threshold values and stored them in a table. As you can see in the table and graph below, cost significantly decreases with a higher threshold.

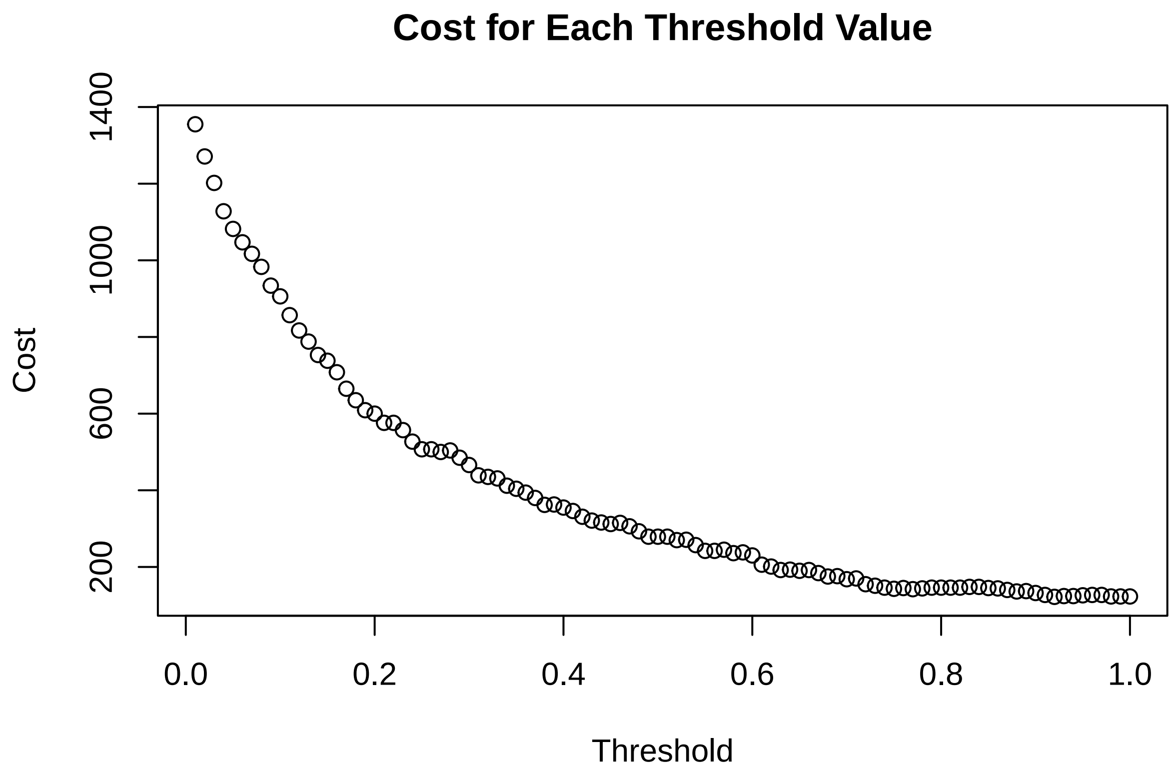
```
#initialize empty table
cost_threshold_table <- data.frame(
  Threshold = rep(seq(0.01, 1, by=0.01), 1),
  Cost = NA
)

#calculate cost at each threshold
for(i in seq(1, 100)){
  threshold <- cost_threshold_table[i,"Threshold"]
  cost_threshold_table[i,"Cost"] <- cost(threshold)
}

#view table
cost_threshold_table
```

```
##     Threshold Cost
## 1        0.01 1355
## 2        0.02 1271
```

```
## 3          0.03 1202
## 4          0.04 1128
## 5          0.05 1082
## 6          0.06 1047
## 7          0.07 1017
## 8          0.08  983
## 9          0.09  934
## 10         0.10  906
## 11         0.11  857
## 12         0.12  817
## 13         0.13  788
## 14         0.14  753
## 15         0.15  738
## 16         0.16  708
## 17         0.17  665
## 18         0.18  635
## 19         0.19  609
## 20         0.20  600
## 21         0.21  576
## 22         0.22  576
## 23         0.23  557
## 24         0.24  527
## 25         0.25  507
## 26         0.26  507
## 27         0.27  500
## 28         0.28  504
## 29         0.29  485
## 30         0.30  466
## 31         0.31  439
## 32         0.32  435
## 33         0.33  431
## 34         0.34  412
## 35         0.35  404
## 36         0.36  394
## 37         0.37  380
## 38         0.38  362
## 39         0.39  363
## 40         0.40  355
## 41         0.41  346
## 42         0.42  331
## 43         0.43  321
## 44         0.44  316
## 45         0.45  312
## 46         0.46  315
## 47         0.47  306
## 48         0.48  293
## 49         0.49  279
## 50         0.50  279
## 51         0.51  279
## 52         0.52  270
## 53         0.53  271
## 54         0.54  257
## 55         0.55  242
## 56         0.56  242
```

```
## 57       0.57  245
## 58       0.58  236
## 59       0.59  238
## 60       0.60  230
## 61       0.61  206
## 62       0.62  201
## 63       0.63  192
## 64       0.64  193
## 65       0.65  190
## 66       0.66  192
## 67       0.67  184
## 68       0.68  175
## 69       0.69  176
## 70       0.70  168
## 71       0.71  170
## 72       0.72  155
## 73       0.73  151
## 74       0.74  146
## 75       0.75  143
## 76       0.76  145
## 77       0.77  142
## 78       0.78  144
## 79       0.79  146
## 80       0.80  146
## 81       0.81  146
## 82       0.82  146
## 83       0.83  148
## 84       0.84  148
## 85       0.85  145
## 86       0.86  144
## 87       0.87  140
## 88       0.88  136
## 89       0.89  137
## 90       0.90  132
## 91       0.91  127
## 92       0.92  122
## 93       0.93  124
## 94       0.94  124
## 95       0.95  126
## 96       0.96  127
## 97       0.97  127
## 98       0.98  123
## 99       0.99  123
## 100      1.00  123
```

```r
#plot results
plot(cost_threshold_table, main="Cost for Each Threshold Value")
```



I would recommend choosing a threshold value between 0.6 and 0.7. If the threshold is too high, almost every customer will be classified as a credit risk and the bank will not be able to make any loans. A threshold between 0.6 and 0.7 will lower costs while still allowing the bank to make loans to good customers.