ISYE 6501 - Homework 7

Justin Lewis

February 21, 2019

Start by clearing the environment and importing the relevant packages. Still struggling to prevent these annoying outputs.

```
rm(list=ls())
library(kernlab, quietly=TRUE)
```

```
## Warning: package 'kernlab' was built under R version 3.5.2
```

```
library(knitr, quietly=TRUE)
```

```
## Warning: package 'knitr' was built under R version 3.5.2
```

```
library(ggplot2, quietly=TRUE)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
library(tree, quietly=TRUE)
```

```
## Warning: package 'tree' was built under R version 3.5.2
```

```
library(randomForest, quietly=TRUE)
```

```
## Warning: package 'randomForest' was built under R version 3.5.2
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

Question 10.1

Q:

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.

A:

```
#First going to load in the uscrime data
uscrime <- as.data.frame(read.csv('uscrime.txt', header=TRUE, sep='\t'))
resp <- uscrime$Crime
attribs <- uscrime[1:15]
#a) a regression tree model

#Split data up into train/test
train_size <- floor(0.75*nrow(uscrime))
set.seed(42)
train_indices <- sample(seq_len(nrow(uscrime)), size=train_size)
train <- uscrime[train_indices, ]
test <- uscrime[-train_indices, ]

#First try using all predictors to make the model
tree_all <- tree(train$Crime ~., train[1:15])

#We can check how accurate the model is on our training data
train_acc <- 100*sum(tree_all$y == train$Crime)/length(train$Crime)
```

Just looking at the training accuracy, we see we have 100%. This is almost definitely far too overfit to the training data, and we know the resulting SSE/R^2 will be 0/1 respectively. We can confirm the overfitting by checking with the test data.

```
#We can first see what happens if we simply use all 15 predictors to make a tree model
predictions <- predict(tree_all, test[1:15], type='vector')

#Can show how overfit the data is with this accuracy
test_acc <- 100*sum(predictions == test$Crime)/length(test$Crime)

#Also check R^2 metric
sst_test <- sum((test$Crime-mean(test$Crime))^2)
sse_test <- sum((test$Crime - predictions)^2)
R2 <- 1 - sse_test/sst_test
```

The resulting test accuracy is 0%, but this is expected for regression. Checking the R^2 value gives 0.0086245 which is terrible as expected. With these results I would not want to use this model. Lets try using some cross-validation to get a model which performs more realistically.

```
#Do a cross validation wth 5 folds
cv_tree <- cv.tree(tree_all, k=5)
```

```
## Warning in cvdev + plearn$dev: longer object length is not a multiple of
## shorter object length

## Warning in cvdev + plearn$dev: longer object length is not a multiple of
## shorter object length

## Warning in cvdev + plearn$dev: longer object length is not a multiple of
## shorter object length

## Warning in cvdev + plearn$dev: longer object length is not a multiple of
## shorter object length

## Warning in cvdev + plearn$dev: longer object length is not a multiple of
## shorter object length
```

```
#Make predictions
cv_preds <- predict(cv_tree, train[1:15])

#Check the R^2
cvtr_sse <- sum((train$Crime - cv_preds)^2)
cvtr_sst <- sum((train$Crime - mean(train$Crime))^2)
cvtr_R2 <- 1- cvtr_sse/cvtr_sst

#And on the test set?
cv_test_preds <- predict(cv_tree, test[1:15])
cvte_sse <- sum((test$Crime - cv_test_preds)^2)
cvte_sst <- sum((test$Crime - mean(train$Crime))^2)
cvte_R2 <- 1 - cvte_sse/cvte_sst
```

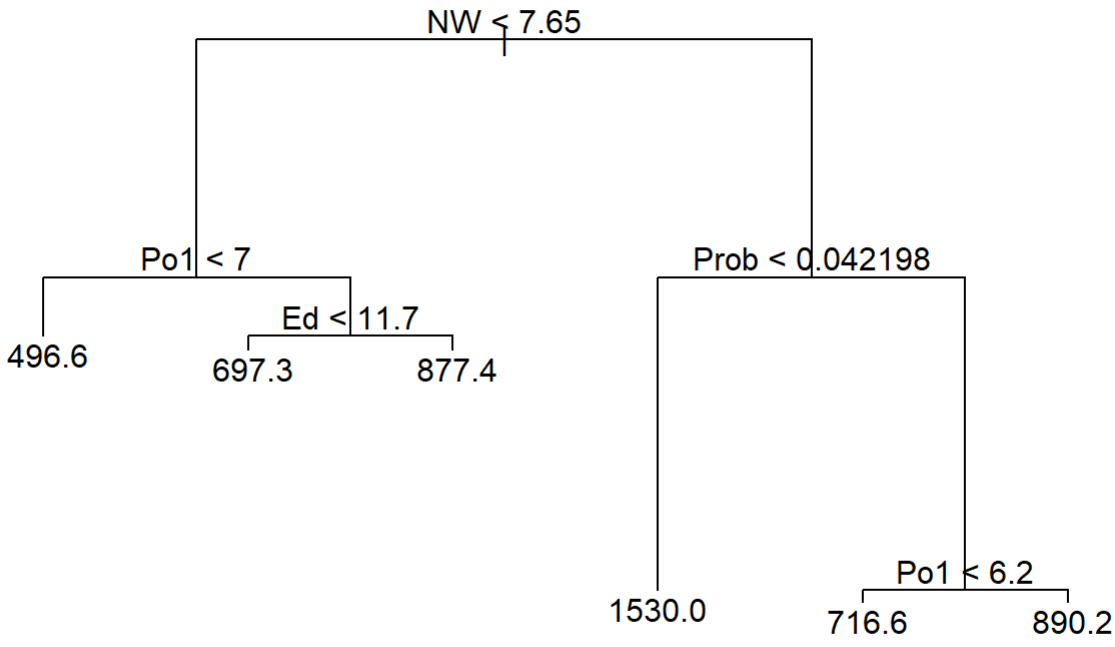
With some cross-validation we get a more reasonable value for our train set R^2 with 0.780997, and a test set R^2 of 0.3003156. With such a small dataset it's unsurprising that it is hard to make a strong model without overfitting to the data used in training.

To take some information away from the tree result we can check the summary, as well as visualizing the splits

```
summary(cv_tree)
```

```
##
## Regression tree:
## tree(formula = train$Crime ~ ., data = train[1:15])
## Variables actually used in tree construction:
## [1] "NW"   "Po1"  "Ed"   "Prob"
## Number of terminal nodes:  6
## Residual mean deviance:  38810 = 1126000 / 29
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -379.00  -95.41   14.38    0.00   89.70  463.00
```

```
plot(cv_tree, main='Tree Diagram')
text(cv_tree)
```



From the summary we see that the resulting tree has 6 leaves, and the splits are based only on the columns ‘NW’, ‘Po1’, ‘Ed’, and ‘Prob’. An interesting result is that Po1 is used to split on both sides of the tree after the original split, but with different thresholds for splitting.

```
#b) A random forest model
#RandomForest allows us to feed in both a training set and a test set.
rf <- randomForest(train[1:15], y=train$Crime, xtest=test[1:15], ytest=test$Crime, importance=TR
UE, mtry=4)

#We can find the average R-squared value across all the training trees
mean(rf$rsq)
```

```
## [1] 0.4357336
```

```
#And the average R-squared value for all the test trees
mean(rf$test$rsq)
```

```
## [1] 0.2660626
```

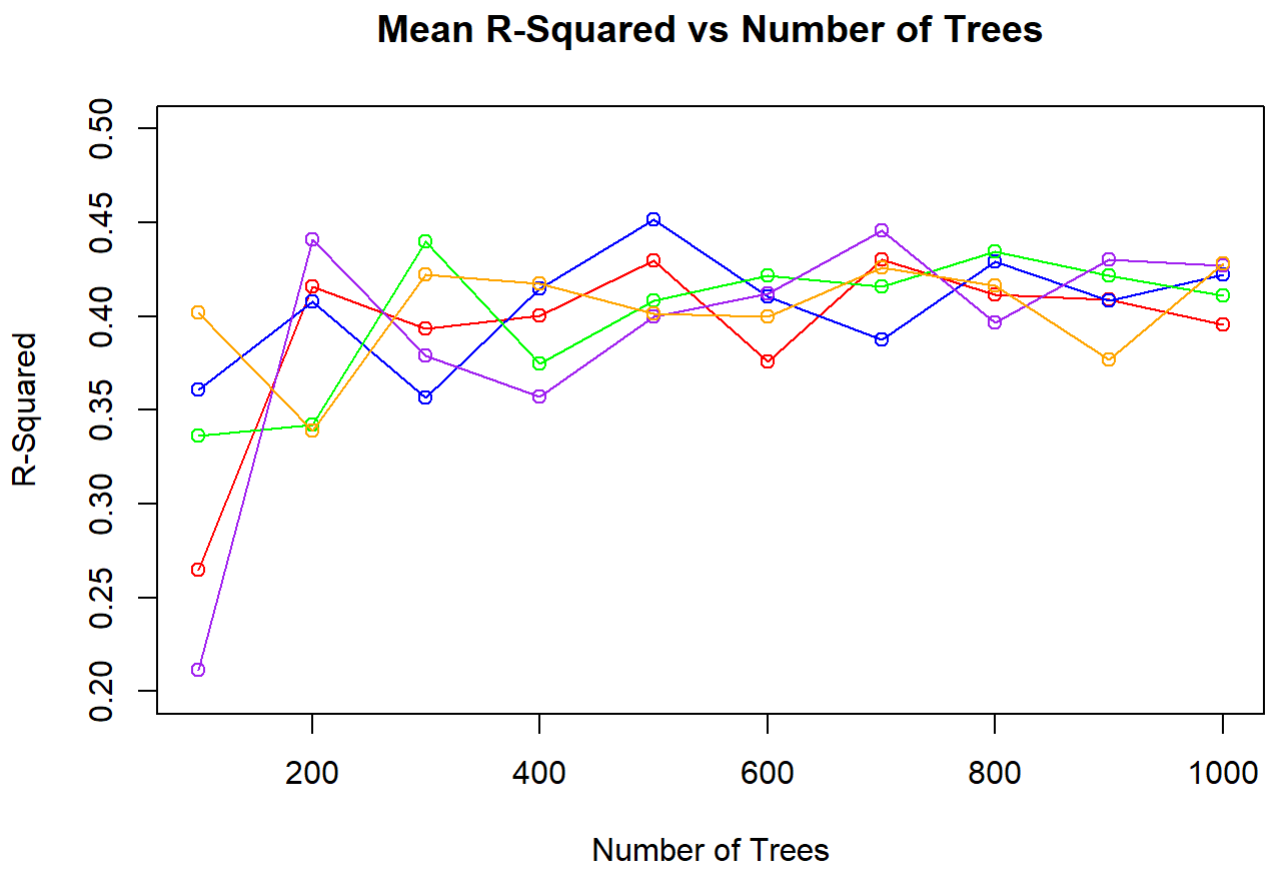
```
#Importance shows the mean decrease in accuracy, and the mean decrease in MSE
rf$importance
```

```
##             %IncMSE IncNodePurity
## M       5787.008588     135922.06
## So       996.225939      21758.16
## Ed     13218.103920     349072.43
## Po1    31850.410787     919814.79
## Po2    23775.878397     788176.81
## LF      6490.205382     160899.04
## M.F     4550.247969     318462.09
## Pop     3307.866831     325965.25
## NW     30315.424386     457793.04
## U1      1704.182560      65368.89
## U2        -7.024009      98809.31
## Wealth  6406.410717     454638.99
## Ineq    4965.003458     156172.17
## Prob    5868.096446     405022.56
## Time    3789.041959     128992.03
```

While the training data R-squared is significantly lower, our test data R-squared is not that much lower. This is most likely due to the random forest avoiding overfitting to the training data by taking an average of so many models (default is 500 trees). To find a better model, we can try adjusting the number of trees.

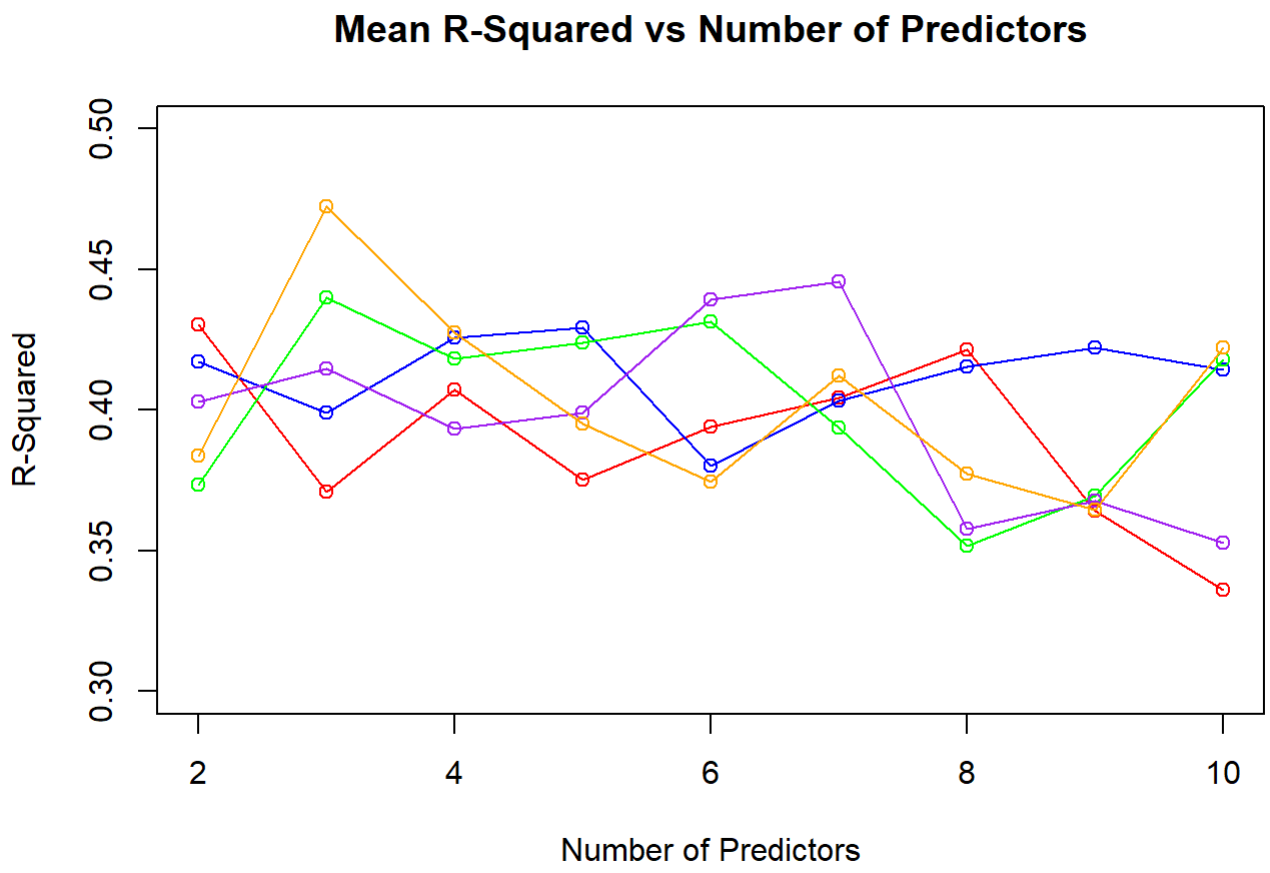
```
#Initialize a list to store the results of each run
run_results = list()
#For loop to perform 5 runs
for (j in seq(1,5)){
  i=1
  #Initialize list to store the R^2 values for each number of trees
  rsquareds = list()
  #Iterate through the different numbers of trees, making a new R2 entry for each
  for (num in c(100, 200, 300, 400, 500, 600, 700, 800, 900, 1000)){
    difftrees_rf <- randomForest(train[1:15], y=train$Crime, importance=TRUE, ntree=num)
    rsquareds[[i]] <- mean(difftrees_rf$rsq)
    i = i+1
  }
  #Store the run results in our list
  run_results[[j]] <- rsquareds
}
```

However we can see below, there is no consistently better setting for the number of trees in the forest. I ran through 5 cycles to see if there was any consistently better parameter, but the R-squared values fluctuate across the entire range for each cycle.



To try another adjustable parameter we can try changing the number of predictors to be checked at each split point.

```
#Same code as above, this time adjusting mtry rather than ntree parameters
run_results = list()
for (j in seq(1,5)){
  i=1
  rsquareds = list()
  for (num in seq(2,10)){
    difftrees_rf <- randomForest(train[1:15], y=train$Crime, importance=TRUE, ntree=500, mtry=nu
m)
    rsquareds[[i]] <- mean(difftrees_rf$rsq)
    i = i+1
  }
  run_results[[j]] <- rsquareds
}
```



In this case we do see across all 5 runs the original used value of mtry=4 looks to produce the best results. Since the default values of ntree=500 and mtry=4 appear to be near optimal I will not test the model on the test data, because there are no real changes to be made from what I have seen here.

Question 10.2

Q:

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.

A:

In most online dating apps there are two groups to put the people in. For instance with Tinder one either swipes left (0) or swipes right (1). One could possibly build a logistic regression model to predict which group each profile would be put in based on age, distance away, if there is a bio written (yes/no), and an appearance rating method (based on how many other times the person was swiped right on?). With these predictors and the amount of data available, logistic regression could certainly be applied in this setting. The threshold value for a right or left swipe could also be adjusted depending on the user.

Question 10.3.1

Q:

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.

A:

First going to load the data in, and supply the column names given from the data source website. Note that the column for good or bad credit risk was named 'Target', and the values are 0 for good, and 1 for bad. So if a person is a good risk to take, they have a value of 0.

```
german <- as.data.frame(read.csv('germancredit.txt', header=FALSE, sep='', col.names = c('Status
of Current Account', 'Duration in Month', 'Credit History', 'Purpose', 'Credit Amount', 'Saving
s', 'Present Employment Since', 'Installment Rate in Percentage of Disposable Income', 'Personal
Status and Sex', 'Other Debtors', 'Present Residence Since', 'Property', 'Age', 'Other Installme
nt Plans', 'Housing', 'Number of Existing Credits at Bank', 'Job', 'Number of People Liable', 'P
hone' , 'Foreign Worker', 'Target')))

#Prefer the target to be 0/1, so subtracting 1 from the column of 1s and 2s.
german$Target <- german$Target - 1

#Lets see how well distributed the target variable is
tcount <- table(german$Target)
barplot(tcount, col='lightblue', xlab='Target Value', main='Target Distribution')
```

So we see while the target is not exactly equal, we do not have an imbalanced dataset so we do not have to be concerned with resampling the data to even out the classes.

Now we can split up the dataset into training and test sets to begin creating the model we want. Note, in this portion of the question we were not asked to consider the costs of different misclassifications so the threshold will be left at 0.5 for this section.

```
#Split data up into train/test
train_size <- floor(0.75*nrow(german))
train_indices <- sample(seq_len(nrow(german)), size=train_size)
germantrain <- german[train_indices, ]
germantest <- german[-train_indices, ]

#Fit a model
logreg <- glm(germantrain$Target ~., data=germantrain[1:20], family='binomial'(link='logit'))

#Predictions assuming 0.5 as the threshold
logreg_preds <- factor(as.integer(logreg$fitted.values+0.5))

#Now build a confusion matrix for the predictions and the actual target in our training data
cm <- caret::confusionMatrix(logreg_preds, factor(germantrain$Target))

#Check just the matrix
cm$table
```

```
##           Reference
## Prediction   0   1
##          0 491 103
##          1  47 109
```

```
cmtable <- as.data.frame(cm$table)
tp <- cmtable[['Freq']][1]
tn <- cmtable[['Freq']][4]
fp <- cmtable[['Freq']][3]
fn <- cmtable[['Freq']][2]
```

Using a threshold of 0.5, we see that we have 491 correct positive predictions, 109 true negative predictions, and then 103 and 47 false positives and false negatives respectively. This gives us an accuracy of 0.8. Additionally the specificity is 0.5141509 with 0.9126394 sensitivity, and a model AIC of 738.6734008.

Initially I used all the columns, but this may not be the ideal use. One way to cut out some of the columns is looking over the coefficients and corresponding p-values. If the p-values are high and the coefficient is near zero, it may be beneficial to remove those predictors. The coefficients and p-values can be checked using the summary call, but the output is extremely long. I looked through them, and based on near-zero coefficients as well as high (>0.1) p-values, removed ‘Present Employment Since’, ‘Personal Status and Sex’, ‘Other Debtors’, ‘Present Residence Since’, ‘Property’, ‘Age’, ‘Other Installment Plans’, ‘Job’, ‘Number of People Liable’, and ‘Phone’. All of these predictors had a combination of high p-values for each of the values in the category, as well as near-zero coefficients with large standard errors.

```
#Define the new columns to use, and build the predictions
newcols <- c('Status.of.Current.Account', 'Duration.in.Month', 'Credit.History', 'Purpose', 'Cre
dit.Amount', 'Savings', 'Installment.Rate.in.Percentage.of.Disposable.Income', 'Housing', 'Numbe
r.of.Existing.Credits.at.Bank', 'Foreign.Worker')
#Build the new model
reducedmodel <- glm(germantrain$Target ~., germantrain[, newcols], family='binomial'(link='logi
t'))
#Get the predictions
reducedpredictions <- factor(as.integer(reducedmodel$fitted.values+0.5))
#Make the new confusion matrix
redcm <- caret::confusionMatrix(reducedpredictions, factor(germantrain$Target))
redcm$table
```

```
##           Reference
## Prediction   0   1
##          0 495 116
##          1  43  96
```

With the reduced predictors we get accuracy of 0.788, specificity of 0.4528302, 0.9200743 sensitivity, and a model AIC of 742.8714117.

So the effect on accuracy, specificity, and sensitivity is minimal with maybe an improvement in one but a reduction in one of the others, and a small change to AIC. With such minor changes to the metrics and half as many predictors, this more simple model would be my choice.

All the factors and their relevant coefficients can be displayed using the summary call:

```
#Look over the coefficients for all the factors
s <- summary(reducedmodel)
kable(s$coefficients)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-0.8270580	0.8062023	-1.0258691	0.3049533
Status.of.Current.AccountA12	-0.2766133	0.2426145	-1.1401355	0.2542299
Status.of.Current.AccountA13	-0.8182240	0.4141848	-1.9755049	0.0482109
Status.of.Current.AccountA14	-1.6334624	0.2670248	-6.1172694	0.0000000
Duration.in.Month	0.0332296	0.0104384	3.1833901	0.0014556
Credit.HistoryA31	0.9005372	0.7107664	1.2669947	0.2051572
Credit.HistoryA32	-0.5452760	0.5610483	-0.9718878	0.3311064
Credit.HistoryA33	-0.9001335	0.5982065	-1.5047203	0.1323960
Credit.HistoryA34	-1.3857403	0.5681858	-2.4388856	0.0147326
PurposeA41	-1.7188845	0.4368768	-3.9344829	0.0000834
PurposeA410	-1.9755134	1.2110878	-1.6311892	0.1028504
PurposeA42	-0.3446574	0.2871793	-1.2001471	0.2300822
PurposeA43	-0.8306635	0.2685167	-3.0935269	0.0019779
PurposeA44	0.8443009	0.9931488	0.8501253	0.3952554
PurposeA45	-0.1967025	0.6198553	-0.3173362	0.7509885
PurposeA46	-0.0903444	0.4452701	-0.2028980	0.8392148
PurposeA48	-14.7557538	615.2990500	-0.0239814	0.9808674
PurposeA49	-0.8628086	0.3778730	-2.2833298	0.0224110
Credit.Amount	0.0000804	0.0000501	1.6033416	0.1088593
SavingsA62	-0.2051226	0.3323011	-0.6172793	0.5370505
SavingsA63	-0.9632230	0.5213636	-1.8475073	0.0646736
SavingsA64	-0.5797230	0.5074019	-1.1425322	0.2532329
SavingsA65	-0.9204654	0.2979680	-3.0891416	0.0020074
Installment.Rate.in.Percentage.of.Disposable.Income	0.3175002	0.0987111	3.2164581	0.0012978
HousingA152	-0.3652547	0.2505951	-1.4575491	0.1449649
HousingA153	-0.2646821	0.3784944	-0.6993026	0.4843630
Number.of.Existing.Credits.at.Bank	0.2488747	0.2085683	1.1932528	0.2327704

	Estimate	Std. Error	z value	Pr(>|z|)
Foreign.WorkerA202	-1.1520382	0.6559717	-1.7562315	0.0790489

Now we can take this reduced model and see how it generalizes to the testing data.

```
#Build the predictions
testpreds <- factor(as.integer(predict.glm(reducedmodel, germantest[, newcols], type='response')
+0.5))

#Create a confusion matrix
testcm <- caret::confusionMatrix(testpreds, factor(germantest$Target))

#See how it looks
testcm$table
```

```
##           Reference
## Prediction   0   1
##          0 147  53
##          1  15  35
```

We see from our confusion matrix the model still performs fairly well on the test set, with an accuracy of 0.728, specificity of 0.3977273, and a sensitivity of 0.9074074. Based on this portion of the question I would say this is a good predictive model since at this point we are not considering differences in the cost of a false positive vs a false negative.

Question 10.3.2

Q:

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.

A:

In this section we are saying that the cost of a false positive is 5x most costly than a false negative, so we want to prioritize avoiding false positives to a certain extent. I am going to continue with the predictors from the previous section but now adjust the threshold value used to make the predictions. To see the affect of the threshold value, we can plot an ROC curve.

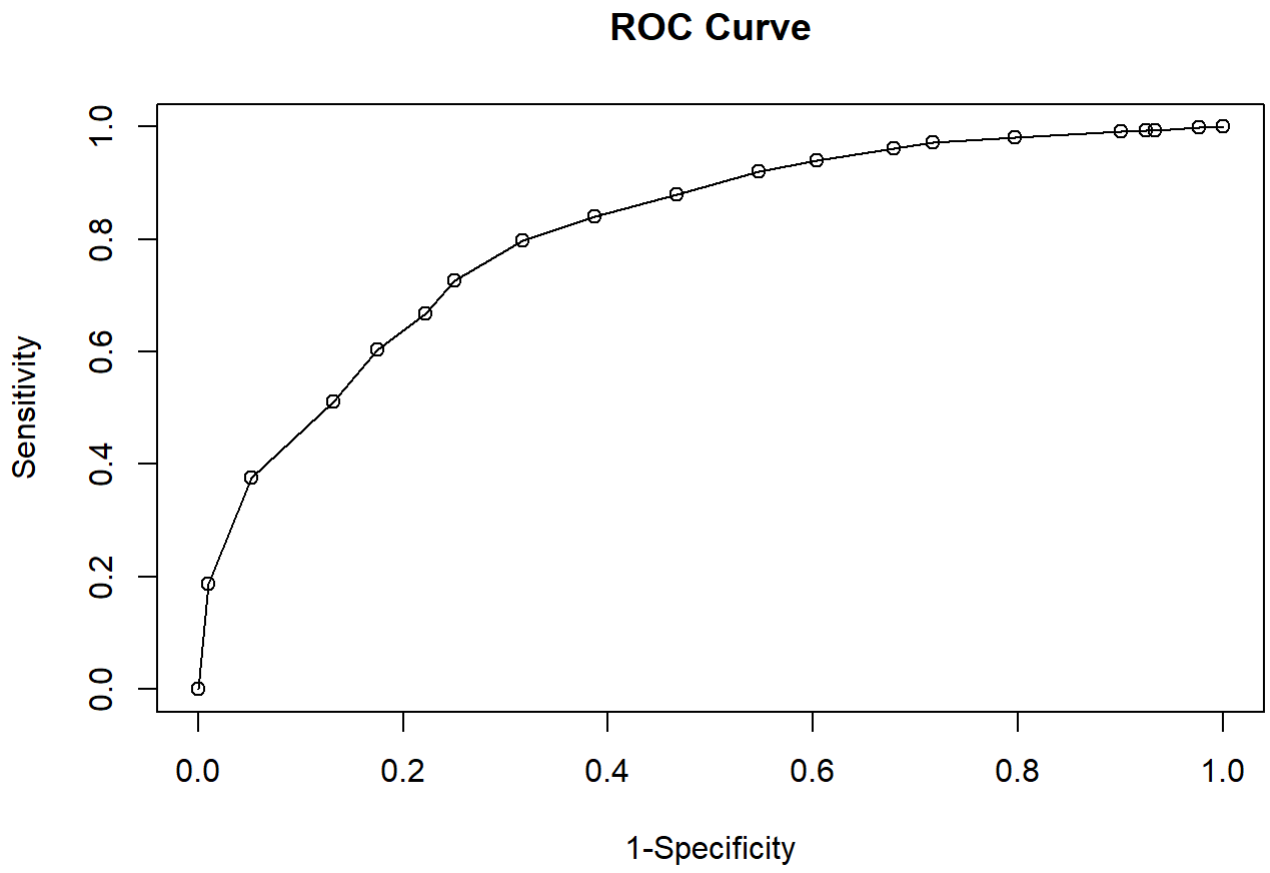
```
#Adding a value makes the threshold 1-value. So if the value added is 0.2, the threshold is 0.8.
Anything below this will be under 1 and round to a zero
specificitiesm1 = list()
sensitivities = list()
i <- 1
for (val in seq(0,1,0.05)){
  predictions <- factor(as.integer(reducedmodel$fitted.values+val))
  threshcm <- caret::confusionMatrix(predictions, factor(germantrain$Target))
  specificitiesm1[[i]] <- 1-threshcm$byClass['Specificity']
  sensitivities[[i]] <- threshcm$byClass['Sensitivity']
  i = i+1
}
```

```
## Warning in confusionMatrix.default(predictions,
## factor(germantrain$Target)): Levels are not in the same order for reference
## and data. Refactoring data to match.

## Warning in confusionMatrix.default(predictions,
## factor(germantrain$Target)): Levels are not in the same order for reference
## and data. Refactoring data to match.

## Warning in confusionMatrix.default(predictions,
## factor(germantrain$Target)): Levels are not in the same order for reference
## and data. Refactoring data to match.
```

```
plot(x=specificitiesm1, y=sensitivities, type='o', ylim=c(0,1), xlim=c(0,1), xlab='1-Specificit
y', ylab='Sensitivity', main='ROC Curve')
```



We see the expected curve here. Now since false positives are much more costly we want to err on the side with fewer false positives and therefore higher specificity. This means we should focus on the side where 1-Specificity is lower and we were using a much higher threshold for what would be considered a positive result.

```
#Build and store confusion matrices for 3 threshold below the 0.5 default
threshes <- c(0.15, 0.25, 0.35)
tables = list()
j <- 1
#Run through the threshold values and store a confusion matrix for each one
for (t in threshes){
  predictions <- factor(as.integer(reducedmodel$fitted.values+(1-t)))
  tcm <- caret::confusionMatrix(predictions, factor(germantrain$Target))
  tables[[j]] <- as.table(tcm$table)
  j = j+1
}
```

```
## [1] "85% Threshold"
```

```
##          Reference
## Prediction   0   1
##          0 275  28
##          1 263 184
```

```
## [1] "75% Threshold"
```

```
##           Reference
## Prediction   0   1
##          0 359  47
##          1 179 165
```

```
## [1] "65% Threshold"
```

```
##           Reference
## Prediction   0   1
##          0 429  67
##          1 109 145
```

Note that zero is the value for a ‘good’ customer in this case. So predicted value of 0 and real response of 1 is a false positive, while predicted value of 1 and real value of 0 is a false negative.

From the above tables we see that a good threshold given the costs is 75% probability that a customer is a ‘good’ one. This results in significantly fewer false positives than false negatives, but still keeps the number of false negatives reasonably low. This is not an exact optimization of the costs. I wanted to avoid exactly minimizing because we are working with the training data still, and it may be different for new data coming in. By choosing a reasonable value but not quite optimal it should remain robust for future data without being overfit to these particular points.

Now lets check how the new threshold does on the test data.

```
testpreds <- factor(as.integer(predict.glm(reducedmodel, germantest[, newcols], type='response')
+0.75))

#Create a confusion matrix
testcm <- caret::confusionMatrix(testpreds, factor(germantest$Target))

#See how it looks
kable(as.table(testcm$table), rownames=c('Predicted'), caption='Test Results - 75% Threshold')
```

Test Results - 75% Threshold

Predicted	0	1
0	116	27
1	46	61

With the new threshold we successfully get significantly more false negatives than false positives on the test data, an accuracy of 0.708, and a specificity of 0.6931818. A reduction in accuracy compared to our model with 0.5, but an increase to the specificity which is what we wanted.