Homework Solutions Week 7

2022-10-08

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).

References:

- https://www.statology.org/classification-and-regression-trees-in-r/
- https://www.rdocumentation.org/packages/tree/versions/1.0-42

```
# load crime data
# http://www.statsci.org/data/general/uscrime.html
crime <- read.delim("uscrime.txt", header=TRUE)
head(crime)
```

```
##      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
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

```
# data for a new state
new_state <- data.frame(
  M=14.0, So = 0, Ed = 10.0, Po1 = 12.0, Po2 = 15.5, LF = 0.640, M.F = 94.0, Pop = 150, NW = 1.1,
  U1 = 0.120, U2 = 3.6, Wealth = 3200, Ineq = 20.1, Prob = 0.04, Time = 39.0)
```

Once again we read in our favourite, the crime dataset. Unlike previous weeks I'm not going to introduce it, but I do show the head as validation the data is read in as expected.

Using the rpart function we can get a basic regression tree output for the crime data set, with rpart the splitting is controlled using a control function, rpart.control(), which has several control parameters.

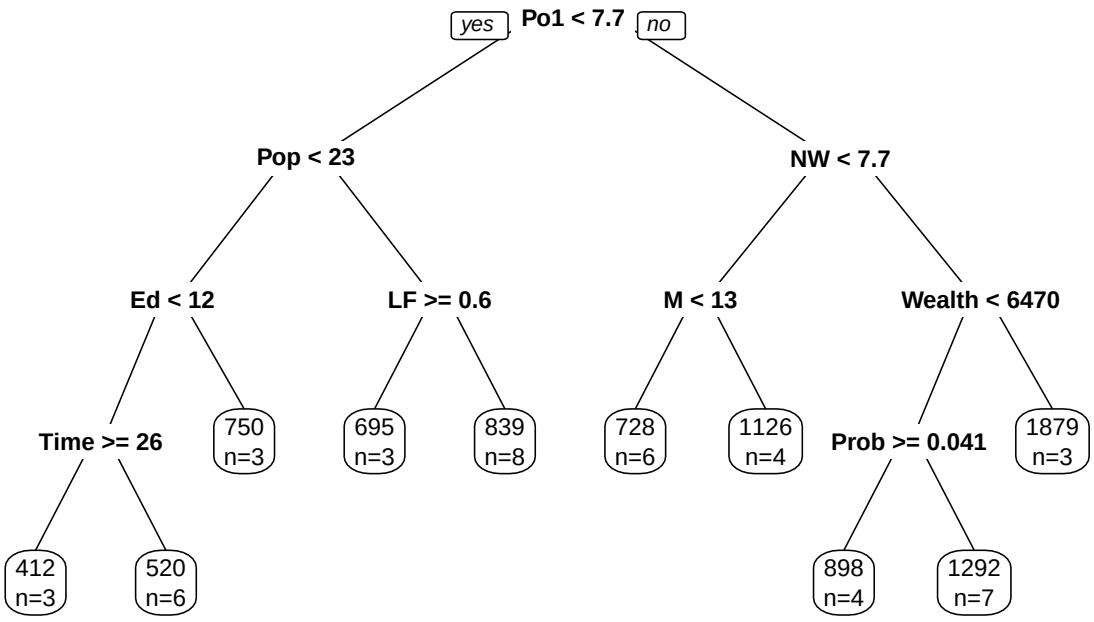
For our purposes there are two that are important:

1. minbucket: sets a floor for the min leaf node observations; set at 3 (rounding up to the nearest integer for 5% of data points). By default rpart with take 3 * minbucket as a threshold number of observations to consider branching - this can be changed but I don't think it's required.
2. cp is the "complexity parameter" which we can think of as the threshold; to start with I set this close to zero, which will create the maximum number of splits.

```
set.seed(88)
reg.tree <- rpart(Crime~., data=crime, control=rpart.control(minbucket=3, cp=0.00000001))
printcp(reg.tree)
```

```
##
## Regression tree:
## rpart(formula = Crime ~ ., data = crime, control = rpart.control(minbucket = 3,
##     cp = 1e-08))
##
## Variables actually used in tree construction:
## [1] Ed     LF     M      NW     Po1    Pop    Prob   Time   Wealth
##
## Root node error: 6880928/47 = 146403
##
## n= 47
##
##           CP nsplit rel error  xerror    xstd
## 1 0.36296293      0   1.00000 1.05855 0.26567
## 2 0.16540024      1   0.63704 0.99339 0.23007
## 3 0.05730143      3   0.30624 1.47015 0.31647
## 4 0.05538867      4   0.24894 1.58332 0.32282
## 5 0.05173165      5   0.19355 1.57927 0.32357
## 6 0.02305931      6   0.14181 1.54930 0.31908
## 7 0.00659405      7   0.11876 1.51328 0.31931
## 8 0.00340071      8   0.11216 1.53476 0.31980
## 9 0.00000001      9   0.10876 1.53476 0.31980
```

```
prp(reg.tree, extra=1, faclen=0)
```

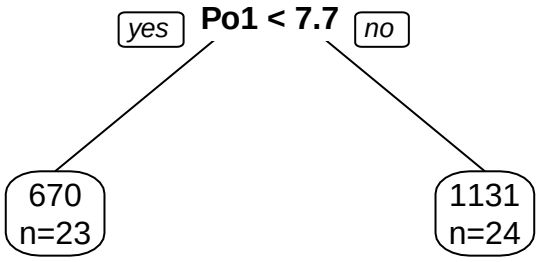


We can use the printcp function to view a table which shows the number of splits, complexity parameter, error and standard deviation. Then we can prune our tree by tuning to use the value of CP which minimises the xerror from this table.

From our original regression tree we see that the primary branch is a split on Po1, the per capita expenditure on policing that year, before branching off. For High police expenditure states the next branch was related to the size of the population, whereas the low police expenditure states seemed to next branch on the number of non-whites.

I'll stop the description there, because to start pruning the tree, we can solve for the value of cp which minimised the xerror. We find the model with the smallest error is just the first branch.

```
best.cp <- reg.tree$cptable[which.min(reg.tree$cptable[, "xerror"]), "CP"]
reg.pruned <- rpart::prune(reg.tree, best.cp)
prp(reg.pruned, extra=1, faclen=0)
```

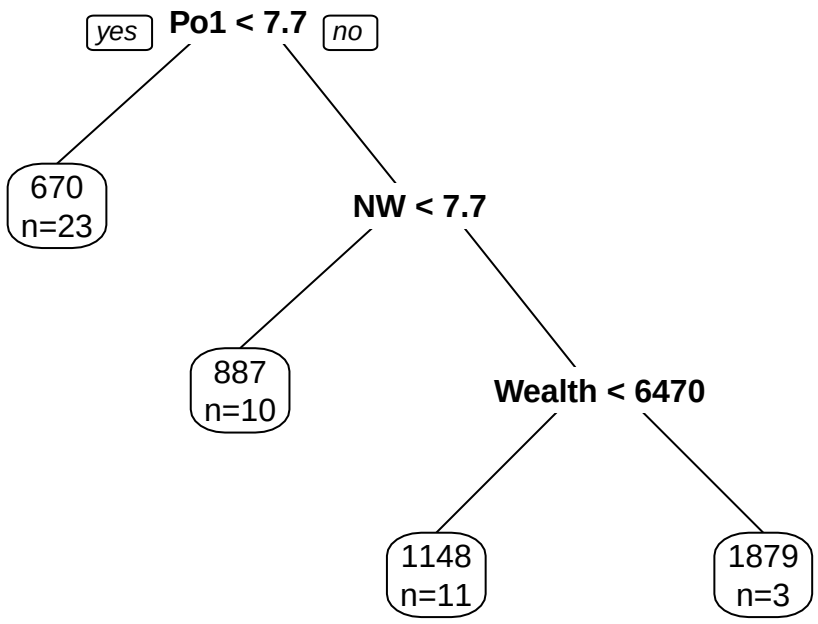


Depending on the application, I would suggest this is a relatively poor model. We only use a single feature of the data - despite there being intuition behind why other features would have predictive power. Because the classification ends with only two branches our only have two estimates for the number of crimes per 100,000 people; which would be fine if we only wanted to classify into high and low crime states.

If our objective was to estimate the number of crimes in a state, (and we had to use a regression tree for a reason such as the ease of explanation), or to cluster into more groups, it may be worth trading off a having more error for more utility.

In the case of the model below, setting cp close to the second best error, we now end up with 4-leaves predicting a gradiation of crimes per 100,000.

This model starts as before, with Po1, the per capita spend on policing where a high police spend predicts the low crime state. The next division is NW, non-white, where a non-white population creates the 2nd lowest crime state, finally where NW is low at this point in the tree the two highest crime states are split by wealth with the lowest wealth of states in this branch generating the most crime.



The question didn't ask us too, but out of interest I used each of the model to run a prediction on the "new state" data we were given.

```
cat("New State Predictions:\n")
```

```
## New State Predictions:
```

```
cat("- basic prediction:", round(predict(reg.tree, newdata=new_state), 0), "\n")
```

```
## - basic prediction: 1126
```

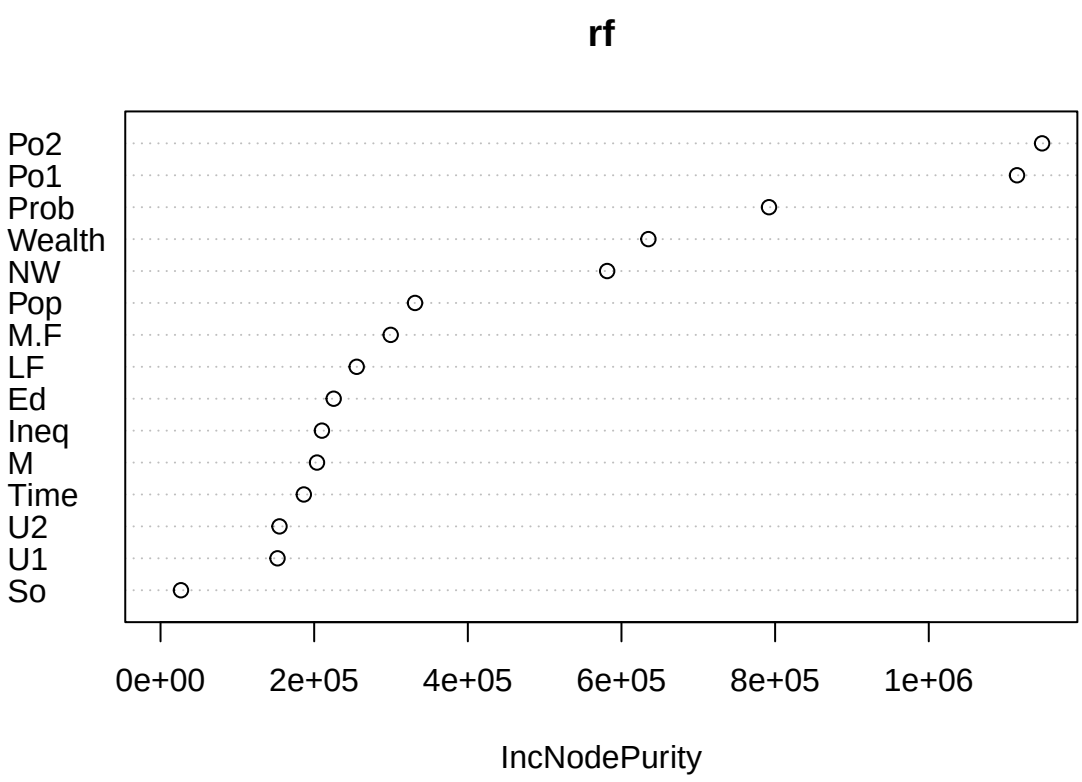
```
cat("- best pruned:", round(predict(reg.pruned, newdata=new_state), 0), "\n")
```

```
## - best pruned: 1131
```

```
cat("- trade off model:", round(predict(reg.pruned2, newdata=new_state), 0), "\n")
```

```
## - trade off model: 887
```

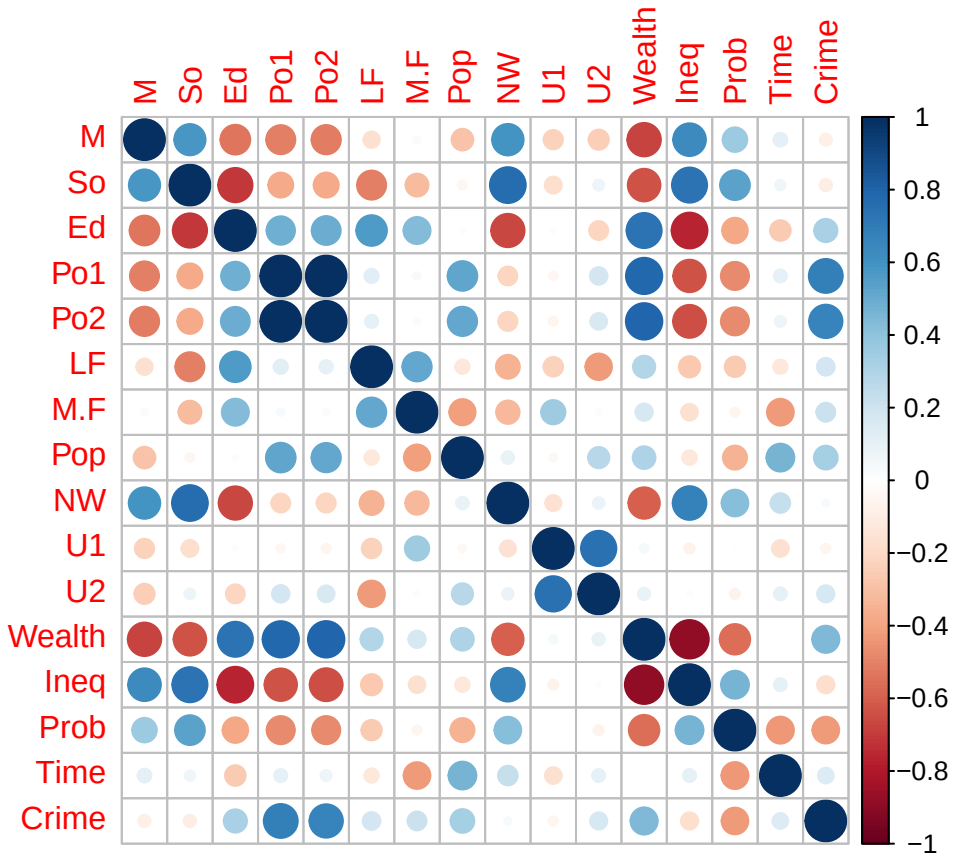
```
set.seed(88)
rf <- randomForest(Crime~., data=crime, ntree=1000)
varImpPlot(rf)
```



We can't plot a random forest model in the same way that we plot the regression tree. The randomForest package has a a variable importance plot which helps us visualise the relative importance of the features. A few key observations:

1. Most important features are Po1 and Po2, which as we noted earlier are the per capita expenditures on policing from last for the current and previous year.
2. Prob, the probability of imprisonment was the second most important feature. This is interesting because it is something which wasn't prominent in any of the regression tree models.
3. Wealth and NW, non-white, populations.

Considering the correlation matrix of the data could help out understanding. In the plot below I've scaled and centered the crime data before plotting in the correlation matrix. What we can see is that Po1 and Po2 have a correlation of very close to 1, which implies multicolliniarity in the data. It also means that from the perspective of a tree model the factors are interchangeable.



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.

I work in finance in the field of Multi-Asset Portfolio Management. A problem for which logistic regression would be useful is to try and model the probability that the US economy will be in recession at a time period in the future - say 6-months from now.

In creating our model, we could use the NBER classifications as the binary response. For features we would use:

1. Slope of the US Treasury yield curve - measured by the difference in the yield between the "on the run" 10-year vs. 2-year treasury bonds
2. The ISM Manufacturing Index which is a survey measure of growth in the US manufacturing sector.
3. 3-month change in the unemployment rate
4. *"Real wage growth"* using the Atlanta Fed wage growth tracker minus the most recent year-on-year CPI inflation rate.
5. OECD Amplitude Adjusted Composite Leading Indicator for OECD countries ex. the US. Which is a forward looking composite indicator for growth and would represent the rest-of-world impact on the US.

Question 10.3

1. **Using the GermanCredit 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.**

References:

- https://www.statology.org/logistic-regression-in-r/
- https://stats.oarc.ucla.edu/r/dae/logit-regression/

```
# read in credit data as table (doesn't like delim); data as provided has no headers
credit <- read.table("germancredit.txt")
glimpse(credit)
```

```
## Rows: 1,000
## Columns: 21
## $ V1  <chr> "A11", "A12", "A14", "A11", "A11", "A14", "A14", "A12", "A14", "A1~
## $ V2  <int> 6, 48, 12, 42, 24, 36, 24, 36, 12, 30, 12, 48, 12, 24, 15, 24, 24,~
## $ V3  <chr> "A34", "A32", "A34", "A32", "A33", "A32", "A32", "A32", "A32", "A3~
## $ V4  <chr> "A43", "A43", "A46", "A42", "A40", "A46", "A42", "A41", "A43", "A4~
## $ V5  <int> 1169, 5951, 2096, 7882, 4870, 9055, 2835, 6948, 3059, 5234, 1295, ~
## $ V6  <chr> "A65", "A61", "A61", "A61", "A61", "A65", "A63", "A61", "A64", "A6~
## $ V7  <chr> "A75", "A73", "A74", "A74", "A73", "A73", "A75", "A73", "A74", "A7~
## $ V8  <int> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, 1, 4, 2, 4, 4, 2, 4, 3, 4, 2, ~
## $ V9  <chr> "A93", "A92", "A93", "A93", "A93", "A93", "A93", "A93", "A91", "A9~
## $ V10 <chr> "A101", "A101", "A101", "A103", "A101", "A101", "A101", "A101", "A~
## $ V11 <int> 4, 2, 3, 4, 4, 4, 4, 2, 4, 2, 1, 4, 1, 4, 4, 2, 4, 3, 2, 2, 4, 3, ~
## $ V12 <chr> "A121", "A121", "A121", "A122", "A124", "A124", "A122", "A123", "A~
## $ V13 <int> 67, 22, 49, 45, 53, 35, 53, 35, 61, 28, 25, 24, 22, 60, 28, 32, 53~
## $ V14 <chr> "A143", "A143", "A143", "A143", "A143", "A143", "A143", "A143", "A~
## $ V15 <chr> "A152", "A152", "A152", "A153", "A153", "A153", "A152", "A151", "A~
## $ V16 <int> 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 3, 1, 1, 3, 1, ~
## $ V17 <chr> "A173", "A173", "A172", "A173", "A173", "A172", "A173", "A174", "A~
## $ V18 <int> 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, ~
## $ V19 <chr> "A192", "A191", "A191", "A191", "A191", "A192", "A191", "A192", "A~
## $ V20 <chr> "A201", "A201", "A201", "A201", "A201", "A201", "A201", "A201", "A~
## $ V21 <int> 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, ~
```

This is a new dataset to us, we can see that there are 20-features and 1-response column, with 1000 data points and the dataset has no headers (so we will us VN as per the R default).

Of the features:

- 13 are categorical with characters
- 7 are integers (non-binary) which represent things like credit value, term, time in current property etc.; glimpsing the data we can see V18 looks like it's binary but the variable represents the number of dependants.

The response as a slightly odd structure, where 1 is positive and 2 is negative. As part of my pre-processing I convert the values to 0 (bad) and 1 (good) and then coerce the vector to be a factor.

```
# the response V21 is weirdly 1 (good) or 2 (bad) rather than 0 (bad) or 1 (good)
credit$V21 <- replace(credit$V21, credit$V21 == 2, 0)
#credit$V21 <- as.integer(credit$V21)

# dataset description shows there is a mix of numerical & categorical data
# https://stackoverflow.com/questions/33180058/coerce-multiple-columns-to-factors-at-once
fcols <- c("V1","V3","V4","V6","V7","V9","V10","V12","V14","V15","V17","V19","V20", "V21")
credit[fcols] <- lapply(credit[fcols], as.factor)

# show head
head(credit)
```

```
##    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
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   1
## 2 A191 A201   0
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   0
## 6 A192 A201   1
```

With pre-processing of the data complete, I split the data 65:35 into train and test sets, then calibrate my logistic regression model using the training set.

```
set.seed(8)
sample <- sample(c(TRUE, FALSE), nrow(credit), replace=TRUE, prob=c(0.65, 0.35))
train <- credit[sample, ]
test <- credit[!sample, ]
```

I run the model using all the features in the training set, look at the p-values from the software summary table as well as finding a value for the psuedo-R^2.

```
# run model on training set & print summary
model.log <- glm(V21~., family="binomial", data=train)
summary(model.log)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7369  -0.6108   0.3244   0.6662   2.2837
```

```
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.192e+00  1.374e+00  -1.596 0.110596
## V1A12        5.699e-01  2.791e-01   2.042 0.041144 *
## V1A13        1.533e+00  5.207e-01   2.944 0.003238 **
## V1A14        1.902e+00  3.049e-01   6.239 4.42e-10 ***
## V2          -2.299e-02  1.250e-02  -1.839 0.065866 .
## V3A31        1.692e-01  7.452e-01   0.227 0.820376
## V3A32        7.835e-01  5.646e-01   1.388 0.165225
## V3A33        7.868e-01  6.340e-01   1.241 0.214562
## V3A34        1.177e+00  5.807e-01   2.027 0.042660 *
## V4A41        1.862e+00  4.729e-01   3.938 8.21e-05 ***
## V4A410       1.407e+00  9.800e-01   1.435 0.151208
## V4A42        9.365e-01  3.380e-01   2.771 0.005595 **
## V4A43        1.283e+00  3.187e-01   4.026 5.67e-05 ***
## V4A44        4.641e-01  9.332e-01   0.497 0.618949
## V4A45        2.238e-01  6.388e-01   0.350 0.726042
## V4A46       -1.825e-01  5.185e-01  -0.352 0.724774
## V4A48        1.599e+01  5.123e+02   0.031 0.975104
## V4A49        1.208e+00  4.581e-01   2.638 0.008347 **
## V5          -2.441e-04  5.927e-05  -4.119 3.81e-05 ***
## V6A62        1.820e-01  3.861e-01   0.472 0.637283
## V6A63        2.022e-01  5.215e-01   0.388 0.698238
## V6A64        1.556e+00  6.636e-01   2.344 0.019058 *
## V6A65        1.177e+00  3.408e-01   3.454 0.000553 ***
## V7A72        1.276e-01  5.784e-01   0.221 0.825428
## V7A73        5.357e-01  5.655e-01   0.947 0.343519
## V7A74        7.928e-01  6.038e-01   1.313 0.189194
## V7A75        2.432e-01  5.658e-01   0.430 0.667332
## V8          -5.024e-01  1.193e-01  -4.212 2.53e-05 ***
## V9A92        9.861e-01  5.148e-01   1.916 0.055416 .
## V9A93        1.709e+00  5.232e-01   3.266 0.001089 **
## V9A94        1.077e+00  5.900e-01   1.825 0.067940 .
## V10A102     -2.333e-01  5.953e-01  -0.392 0.695136
## V10A103      6.228e-01  5.190e-01   1.200 0.230107
## V11          6.236e-02  1.158e-01   0.538 0.590316
## V12A122     -1.273e-01  3.251e-01  -0.392 0.695336
## V12A123     -6.077e-02  3.118e-01  -0.195 0.845452
## V12A124     -1.711e-01  6.214e-01  -0.275 0.783047
## V13          1.878e-02  1.250e-02   1.502 0.133093
## V14A142      5.370e-01  5.634e-01   0.953 0.340575
## V14A143      9.673e-01  3.171e-01   3.050 0.002285 **
## V15A152      5.190e-01  3.060e-01   1.696 0.089822 .
## V15A153      3.886e-01  6.753e-01   0.575 0.565028
## V16          2.029e-01  2.317e-01   0.876 0.381248
## V17A172     -2.549e-01  8.951e-01  -0.285 0.775814
## V17A173     -3.715e-01  8.511e-01  -0.437 0.662449
## V17A174     -3.454e-01  8.392e-01  -0.412 0.680654
## V18         -5.658e-01  3.322e-01  -1.704 0.088464 .
## V19A192      3.758e-01  2.658e-01   1.414 0.157436
## V20A202      2.537e+00  1.001e+00   2.533 0.011297 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 791.76  on 653  degrees of freedom
## Residual deviance: 546.29  on 605  degrees of freedom
## AIC: 644.29
##
## Number of Fisher Scoring iterations: 14
```

```
# also show the McFadden pseudo-R-squared
pscl::pR2(model.log)["McFadden"]
```

```
## fitting null model for pseudo-r2
```

```
##  McFadden
## 0.3100389
```

When we consider the summary table we can see there are severl p-stats which don't look to be adding to the model. I iterate to produce a model below which uses fewer features but comparable pseudo-R^2.

```
credit.formula <- V21 ~ V1+V2+V3+V4+V5+V6+V8+V9+V14+V20
model.log2 <- glm(credit.formula, family="binomial", data=train)
summary(model.log2)
```

```
##
## Call:
## glm(formula = credit.formula, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6372  -0.6746   0.3591   0.6941   2.3357
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.044e+00  8.341e-01  -1.252 0.210573
## V1A12        5.504e-01  2.676e-01   2.057 0.039684 *
## V1A13        1.472e+00  4.887e-01   3.012 0.002598 **
## V1A14        1.947e+00  2.924e-01   6.659 2.76e-11 ***
## V2          -2.361e-02  1.163e-02  -2.030 0.042347 *
## V3A31        4.607e-02  6.985e-01   0.066 0.947421
## V3A32        7.521e-01  5.337e-01   1.409 0.158828
## V3A33        8.649e-01  6.077e-01   1.423 0.154705
## V3A34        1.365e+00  5.620e-01   2.428 0.015189 *
## V4A41        1.676e+00  4.482e-01   3.739 0.000185 ***
## V4A410       1.383e+00  9.245e-01   1.496 0.134645
## V4A42        8.309e-01  3.180e-01   2.612 0.008990 **
## V4A43        1.312e+00  3.047e-01   4.306 1.66e-05 ***
## V4A44        4.811e-01  8.884e-01   0.541 0.588165
## V4A45        3.661e-01  6.326e-01   0.579 0.562779
## V4A46       -2.573e-01  4.917e-01  -0.523 0.600809
## V4A48        1.602e+01  5.279e+02   0.030 0.975795
## V4A49        1.219e+00  4.369e-01   2.790 0.005274 **
```

```
## V5          -2.188e-04  5.387e-05  -4.061 4.88e-05 ***
## V6A62        6.673e-02  3.665e-01   0.182 0.855517
## V6A63        1.460e-01  4.971e-01   0.294 0.769023
## V6A64        1.452e+00  6.239e-01   2.326 0.019993 *
## V6A65        1.185e+00  3.280e-01   3.612 0.000304 ***
## V8          -4.620e-01  1.138e-01  -4.060 4.90e-05 ***
## V9A92        7.031e-01  4.934e-01   1.425 0.154143
## V9A93        1.486e+00  4.964e-01   2.993 0.002758 **
## V9A94        8.995e-01  5.694e-01   1.580 0.114176
## V14A142      5.391e-01  5.375e-01   1.003 0.315884
## V14A143      8.462e-01  2.993e-01   2.827 0.004695 **
## V20A202      2.387e+00  1.028e+00   2.322 0.020207 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 791.76  on 653  degrees of freedom
## Residual deviance: 566.40  on 624  degrees of freedom
## AIC: 626.4
##
## Number of Fisher Scoring iterations: 14
```

```
pscl::pR2(model.log2)["McFadden"]
```

```
## fitting null model for pseudo-r2
```

```
##  McFadden
## 0.2846311
```

To validate my calibrated model, I use the BIC and the broad criteria we were given in Week 5 which implies that our calibrated model is very likely better.

```
## Relative BIC is 103.06 calibrated model is very likely better
```

As a final consideration, I wanted to compare the accuracy of the models. As we know this isn't the ultimate consideration for this model, because we will be considering "cost" in 10.3.2.

To do this I set a probability threshold at 0.5 then compare the training response to to the training data and the test data to the predicted response; I do this for both the base model and the calibrated model. To help present this I use the functions below:

```
# convert vector of probabilities to binary
prob.pred <- function (x, p=0.5) {return (ifelse(x >= p, 1, 0))}

accuracy_test <- function (x, y=NULL, p=0.5) {
  x <- prob.pred(x, p)
  return (sum(x == y) / length(x))
}

accuracy_display <- function (model) {
```

```
  # prediction data
  prediction.train <- predict(model, train[, -21], type="response")
  prediction.test <- predict(model, test[, -21], type="response")

  # accuracy test assuming 0.5 as threshold
  cat("training accuracy at p=0.5:", accuracy_test(prediction.train, train$V21), "\n")
  cat("test accuracy at p=0.5:",accuracy_test(prediction.test, test$V21))

}
```

```
## For base model:

## training accuracy at p=0.5: 0.8165138
## test accuracy at p=0.5: 0.7225434

## For calibrated model:

## training accuracy at p=0.5: 0.8027523
## test accuracy at p=0.5: 0.734104
```

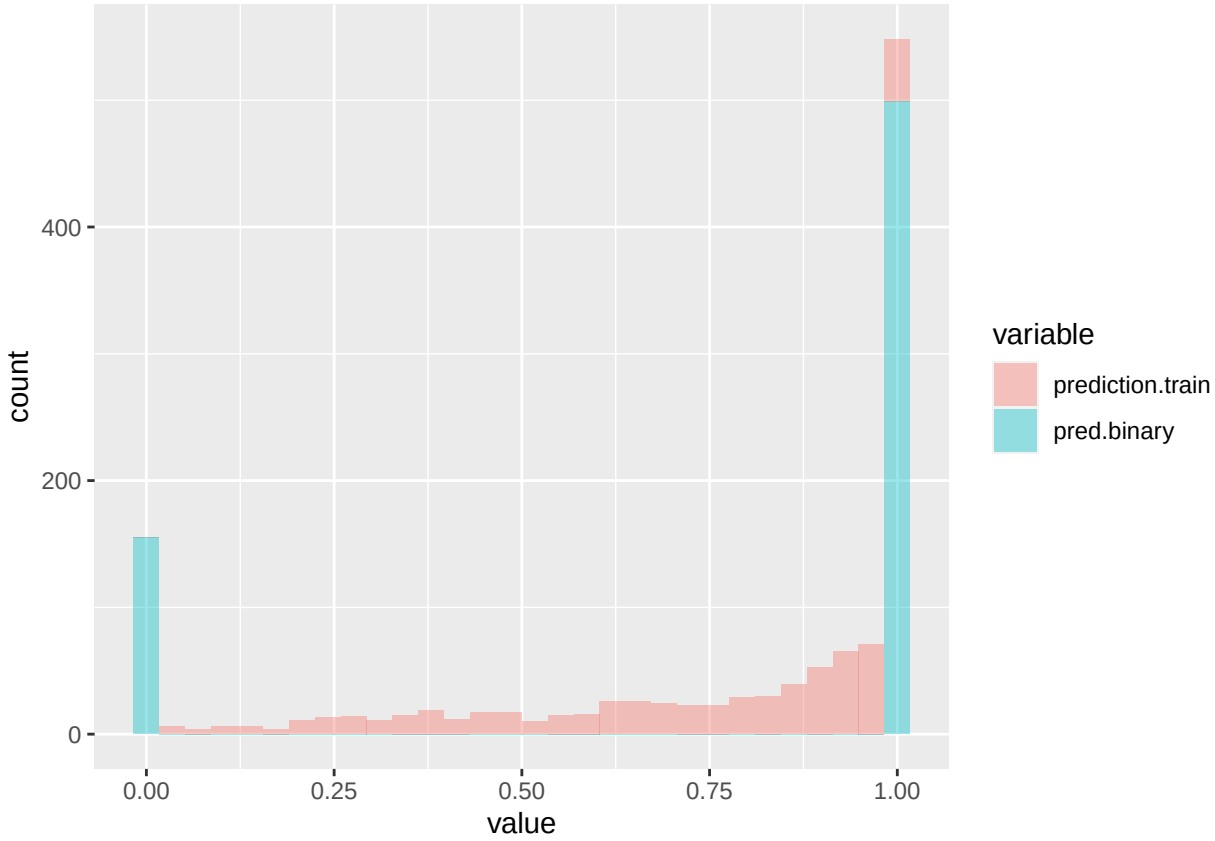
We can see in both cases, as we would expect, the accuracy on the test data is lower than the accuracy on the training data - but although the base model has a higher training accuracy than the calibrated model. When we use the *reserved test data* the accuracy of the calibrated model is higher than the base model.

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.**

Before we start, we need to remember the point the output from the logistic regression model is a probability. We can see this below, what we need to do is apply a threshold level, like 0.5 above, which determines if the prediction is either a 1 or 0.

```
prediction.train <- predict(model.log2, train[, -21], type="response")
pred.binary <- prob.pred(prediction.train, 0.5)
df <- data.frame(prediction.train, pred.binary)

df %>%
  melt(id.vars=prediction.train) %>%
  ggplot(aes(x=value, fill=variable)) +
  geom_histogram(alpha = 0.4)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



To solve for this problem I construct a confusion matrix using the calibrated model. By default the confusionMatrix function in R will show the output as 0, 1 in the columns and rows rather than 1, 0 which is more intuitive to me (and how we learned it in class), so I flip the table order around.

```
x <- prob.pred(prediction.train, 0.9)
cm <- confusionMatrix(as.factor(train$V21), as.factor(x))
cm$table <- cm$table[2:1, 2:1]
cm$table
```

```
##           Reference
## Prediction   1   0
##          1 200 262
##          0  11 181
```

We can verify the accuracy score from the previous question using the confusion matrix, where $accuracy = \frac{(TP+TN)}{observations}$

```
cat("accuracy score for p=0.5:", (cm$table[1, 1] + cm$table[2, 2]) / sum(cm$table))
```

```
## accuracy score for p=0.5: 0.5825688
```

In order to calculate find a calculation of the threshold, we want to estimate the *"cost"* implied by both False Negatives and False Positives.

We could write this:

$cost = (FP \cdot penalty) \times FN$

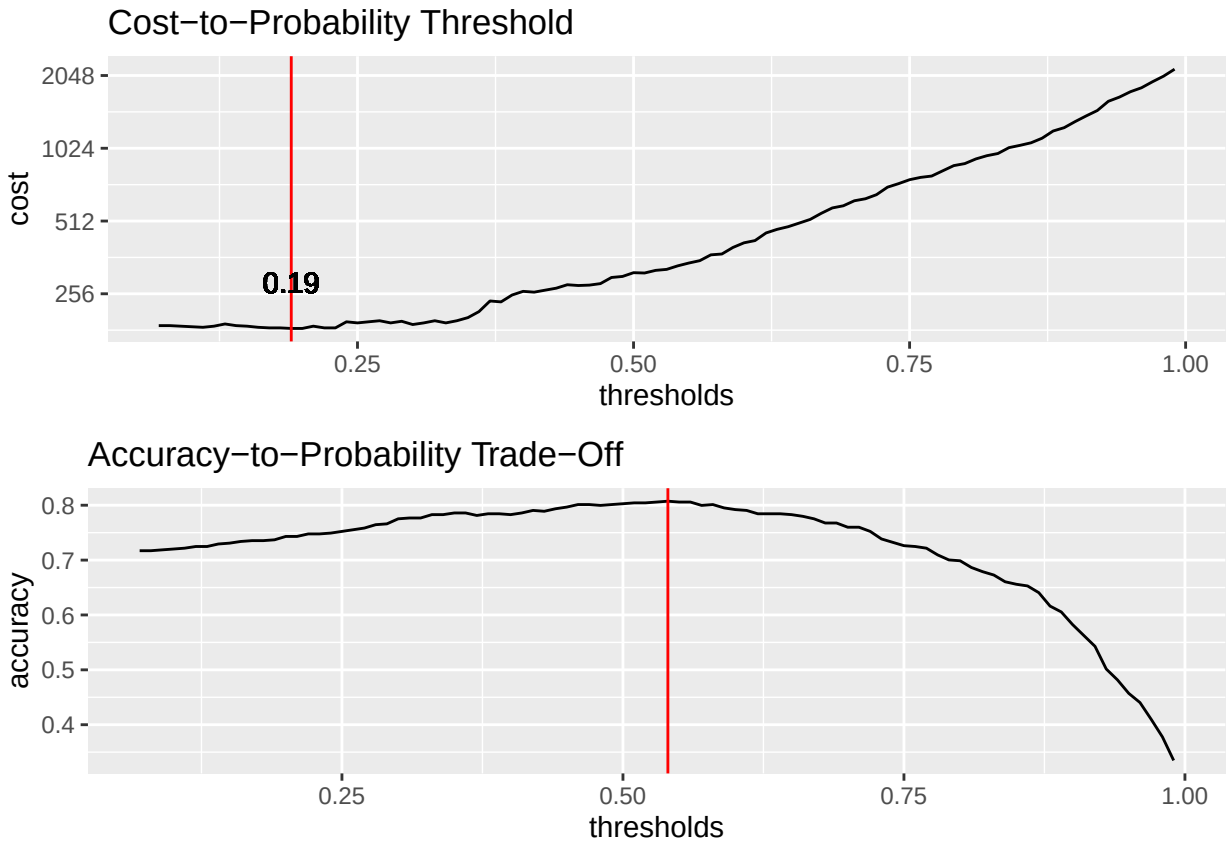
The function below iterates over a series of threshold values of p for the prediction, then plots the corresponding cost. We annotate the chart with the value of p which generates the lowest cost.

On the same plot we also include accuracy vs. threshold. We don't need this second plot for the question, but it is interesting to observe how the probability threshold which maximises accuracy is not necessarily the same as the one which minimises cost!

```
confusing_func <- function (prediction, response_, c=5) {

  # dummy vectors, which populate dataframe later
  thresholds <- seq(0.07, 0.99, 0.01)    # probability thresholds
  cost <- vector()
  accuracy <- vector()

  # iterate over thresholds
  for (p in thresholds) {

    # convert prediction probailities into binary output with prob threshold
    # create confusion matrix & flip table cols / rows to look more like we'd expect
    x <- prob.pred(prediction, p)
    cm <- confusionMatrix(as.factor(response_), as.factor(x))
    cm$table <- cm$table[2:1, 2:1]

    # cost formula (FP * c) + (FN * 1)
    p_cost <- (cm$table[1, 2] * c) + (cm$table[2, 1] * 1)

    # append cost & accuracy to vectors
    cost <- append(cost, p_cost)
    accuracy <- append(accuracy, (cm$table[1, 1] + cm$table[2, 2]) / sum(cm$table))
  }

  # create dataframe
  cost <- data.frame(thresholds, cost, accuracy)
  cost.min <- cost[which.min(cost$cost), ]    # find minimum cost data type

  # cost to prob trade off plot
  fig1 <- cost %>%
    ggplot(aes(x=thresholds, y=cost)) + geom_line() +
    ggtitle("Cost-to-Probability Threshold") +
    scale_y_continuous(trans='log2') +
    geom_vline(xintercept = cost.min[, 1], color="red") +
    geom_text(label=cost.min[, 1], aes(x=cost.min[, 1], y=cost.min[, 2]+100))

  # accuracy plot
  fig2 <-cost %>%
    ggplot(aes(x=thresholds, y=accuracy)) + geom_line() +
    geom_vline(xintercept = cost[which.max(cost$accuracy), 1], color="red") +
    ggtitle("Accuracy-to-Probability Trade-Off")

  # combine figures into a single plot
  plot(ggarrange(fig1, fig2, nrow=2))

}
```

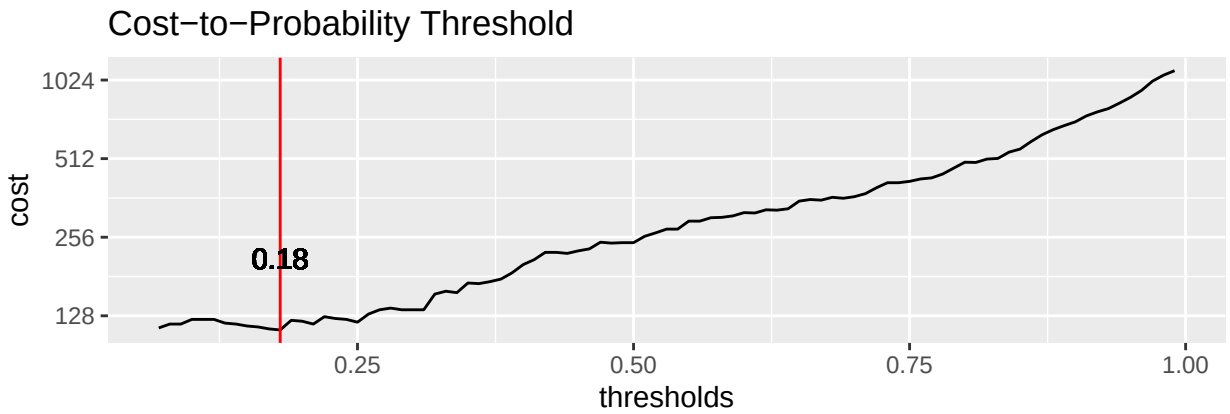
Cost to Probability Trade off on Training Data What we observe is that the cost trade-off is quite flat for threshold probabilities from approx. 0.07-0.25 after which it looks like it picks up and the cost of the False Positives becomes exponentially painful (y is on a log scale). The point at which cost is minimised is when the threshold probability is 0.19, so I'll set my calibration point for p as 0.2.

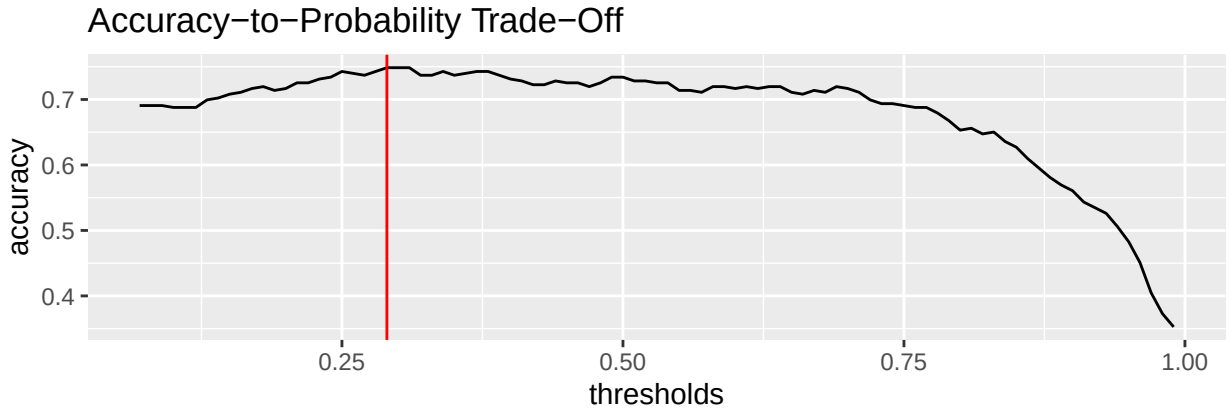
We can also see that if the objective was to maximise accuracy with an equal trade off between FP and FN then the threshold is closer to 0.5.



Cost to Probability Trade off on Test Data Runnint he same tests on the test data, most importantly we ses that our estimate of p=0.2 seems holds in the test data. Here the minimisation threshold is p=0.18 but we obseve the same pattern that the trade off is relatively flat for thresholds $p < 0.25$ and the difference between the training and test result is probably not statistically significant.

```
prediction.test <- predict(model.log2, test[, -21], type="response")
confusing_func(prediction.test, test$V21)
```



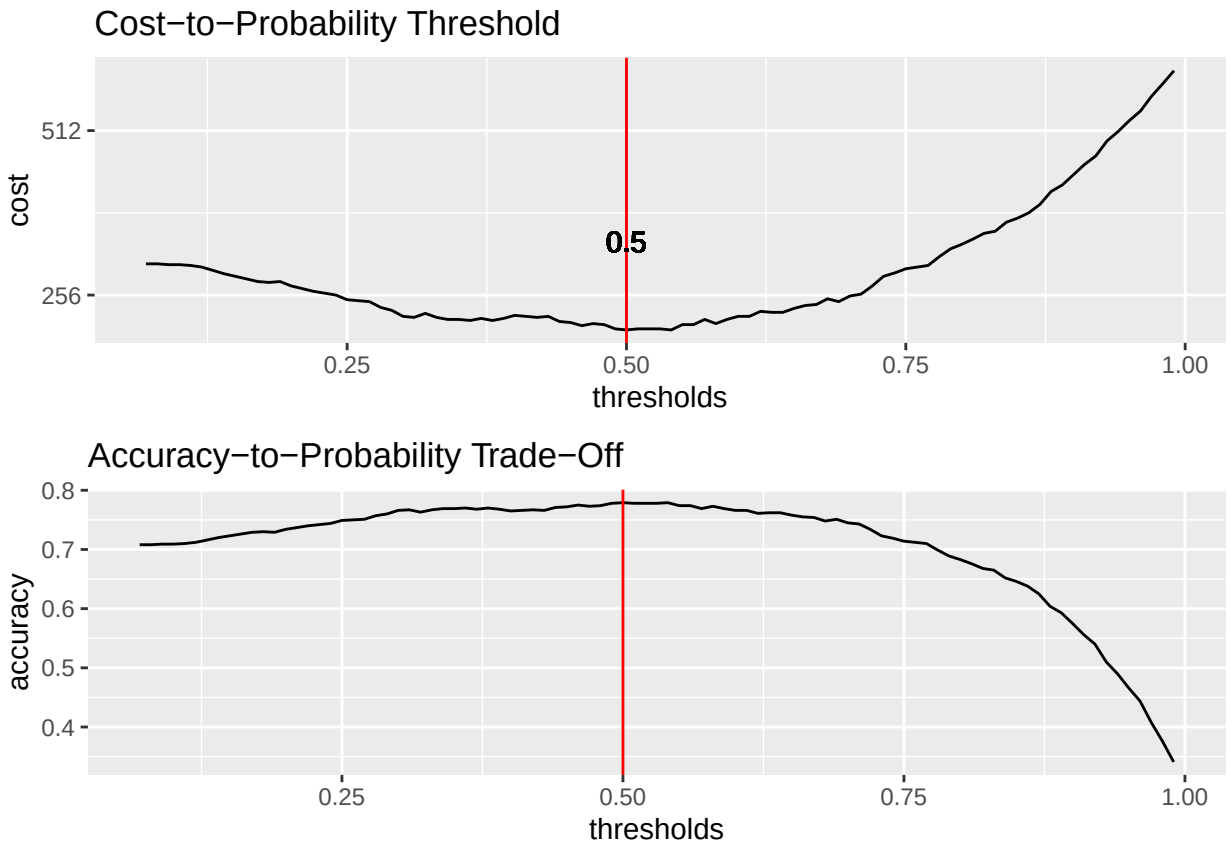


Changing the Cost Parameter Our function was written such that we can adjust the cost of a False Positive, by varying an input c (where the default is 5 as per the question). I find it interesting to look at a value of c = 1 which matches the inuition that if a FP is felt equally to an FN then the cost minimisation threshold, is also the accuracy maximisation threshold, which is equal to 0.5; for this test I use our calbrated model and the full data set (rather than train / test)

```
print("For c=1")
```

```
## [1] "For c=1"
```

```
prediction <- predict(model.log2, credit[, -21], type="response")
confusing_func(prediction, credit$V21, c=1)
```

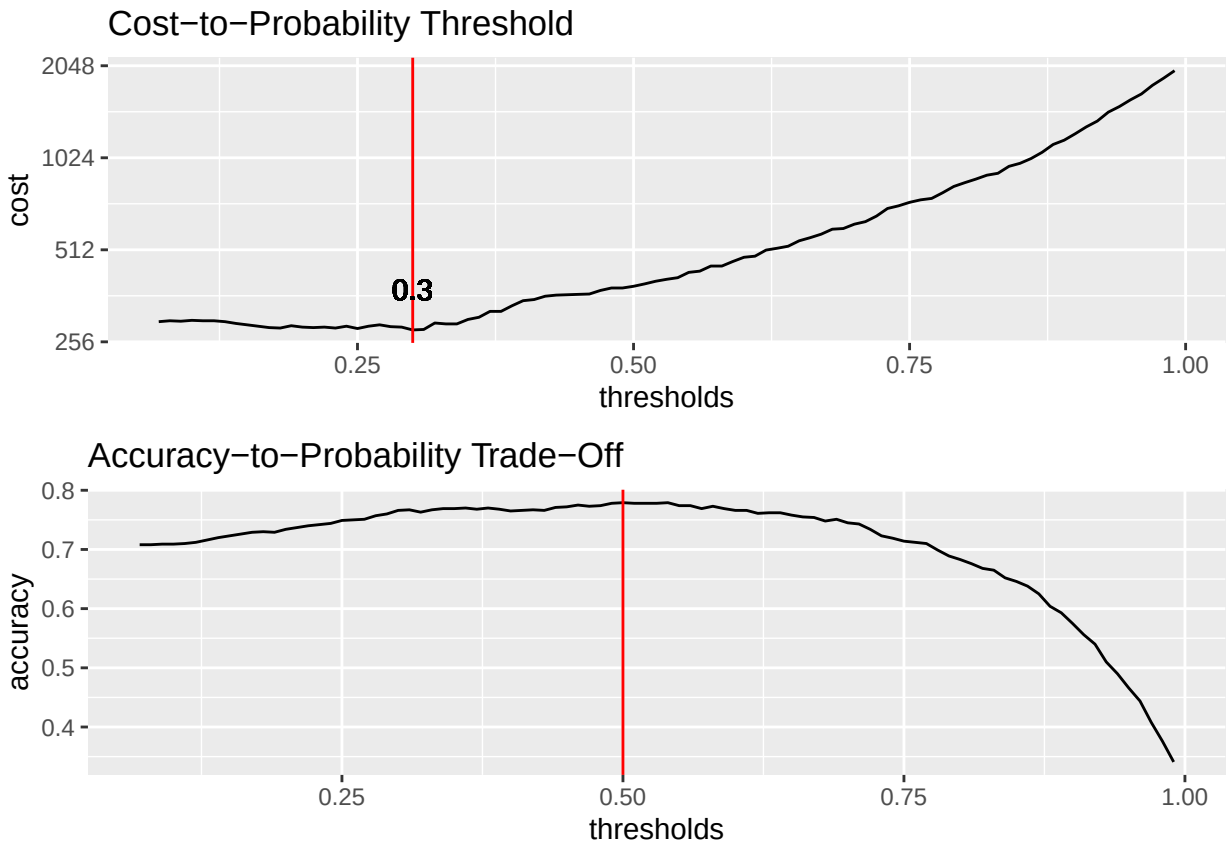


If the cost was varied such that the FP was only felt 3x as greatly as a FN then from the charts below we can see that the threshold is now 0.3. This could happen, from the point of view of the credit issuer, if - for example - recovery rates on defaulted loans shifted mking the cost of lending to someone who subsequently defaults less painful than the 5x initial assumption.

```
print("For c=3")
```

```
## [1] "For c=3"
```

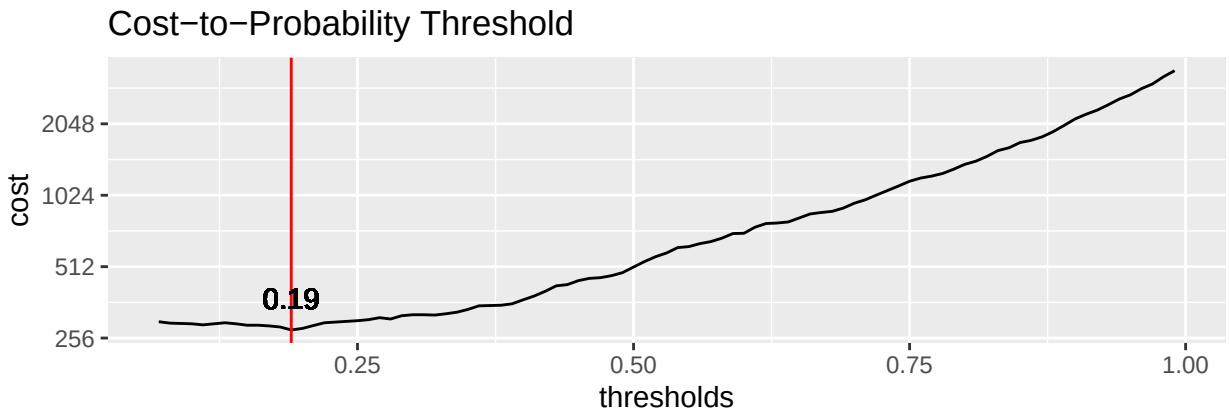
```
prediction <- predict(model.log2, credit[, -21], type="response")
confusing_func(prediction, credit$V21, c=3)
```

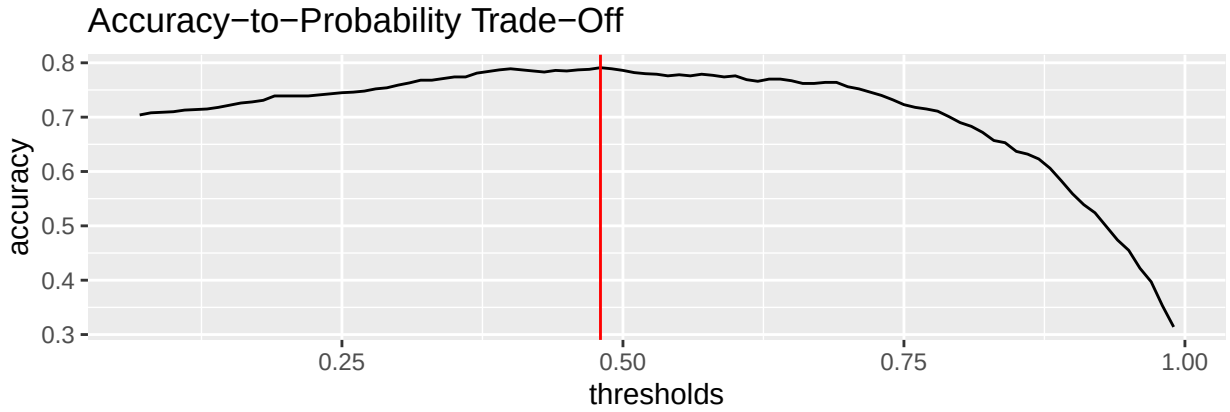


Base Model (Uncalibrated) Finally, we see what would have been predicted even if I can't calibrated the model, or hadn't started with a train/test split.

We can see that it ultimately makes no difference to the probability threshold, which shws that for values p < 0.2 the cost is essentially flat, and that the optimal threshold for this sample of data (with an uncalibrated model) is 0.19 - similar to our estimate of 0.2 from earlier.

```
model.full <- glm(V21~., family="binomial", data=credit)
prediction <- predict(model.full, credit[, -21], type="response")
confusing_func(prediction, credit$V21, c=5)
```


Cost−to−Probability Threshold
2048
1024
512
256
cost
0.19
0.25
0.50
0.75
1.00
thresholds


Accuracy−to−Probability Trade−Off
0.8
0.7
0.6
0.5
0.4
0.3
accuracy
0.25
0.50
0.75
1.00
thresholds

Thank You