ISYE6501 HW4

2024-06-06

Question 9.1

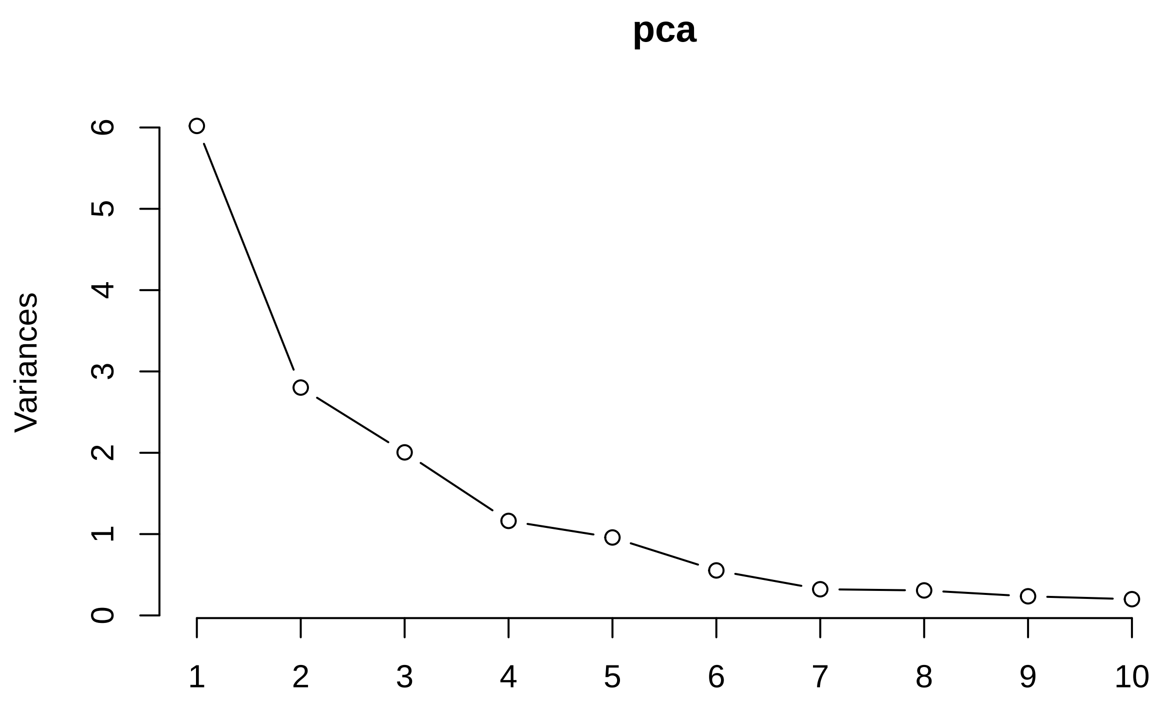
Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components.

Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2.
Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

```r
crime <- read.table("USCrime.txt", header=TRUE)

# Get the means of each column
mus <- colMeans(crime[,-16])

# Split into train test sets
indexes <- sample(2, nrow(crime),
              replace = TRUE,
              prob = c(0.8, 0.2))
c.train <- crime[indexes==1,]
c.test <- crime[indexes==2,]


# Do PCA, scaling the data
pca <- prcomp(crime[-16], scale. = TRUE, center = TRUE)
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377 0.56729
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688 0.02145
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996 0.92142
##                            PC8     PC9    PC10    PC11    PC12    PC13   PC14
## Standard deviation     0.55444 0.48493 0.44708 0.41915 0.35804 0.26333 0.2418
## Proportion of Variance 0.02049 0.01568 0.01333 0.01171 0.00855 0.00462 0.0039
## Cumulative Proportion  0.94191 0.95759 0.97091 0.98263 0.99117 0.99579 0.9997
##                           PC15
## Standard deviation     0.06793
## Proportion of Variance 0.00031
## Cumulative Proportion  1.00000
```

```r
screeplot(pca, type = "line")
```



```r
cor(pca$x[1:4,1:4])
```

```
##             PC1        PC2        PC3        PC4
## PC1  1.0000000 -0.4469727  0.8513955 -0.5037776
## PC2 -0.4469727  1.0000000 -0.1534458 -0.5474158
## PC3  0.8513955 -0.1534458  1.0000000 -0.6559740
## PC4 -0.5037776 -0.5474158 -0.6559740  1.0000000
```

Looking at the correlation of the first four principal components, there is some pretty heavy correlation between PC1 and PC3. But for this homework I will use the first 4 principal components, and together they account for 79.93% of the variance.

```r
# Unrotate pca to get original (but scaled) data
pca_scaled <- pca$x[,1:4] %*% t(pca$rotation[,1:4])

# Examine correlation
# cor(pca_scaled)

# Unscale data
pca_unscaled <- scale(pca_scaled, center = -mus, scale = FALSE)

# Convert to df and add crime data
pca_unscaled <- data.frame(pca_unscaled)
pca_unscaled$Crime <- crime$Crime

# Run the linear regression on the unscaled PCA data
fit <- lm(Crime ~ ., data = pca_unscaled)
```

```
summary(fit)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = pca_unscaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -557.76 -210.91  -29.08  197.26  810.35
##
## Coefficients: (11 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2182.77    3303.58  -0.661   0.5124
## M              87.65     194.15   0.451   0.6540
## So            -63.46     176.30  -0.360   0.7207
## Ed            -46.35     140.14  -0.331   0.7425
## Po1           280.53     118.15   2.374   0.0222 *
## Po2               NA         NA      NA       NA
## LF                NA         NA      NA       NA
## M.F               NA         NA      NA       NA
## Pop               NA         NA      NA       NA
## NW                NA         NA      NA       NA
## U1                NA         NA      NA       NA
## U2                NA         NA      NA       NA
## Wealth            NA         NA      NA       NA
## Ineq              NA         NA      NA       NA
## Prob              NA         NA      NA       NA
## Time              NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 336.4 on 42 degrees of freedom
## Multiple R-squared:  0.3091, Adjusted R-squared:  0.2433
## F-statistic: 4.698 on 4 and 42 DF,  p-value: 0.003178
```

This is the linear regression done on the unscaled PCA output. The NAs are due to too much multicollinearity between the predictors. Based on the summary, the only predictor that is significant using $\alpha = 0.5$ is Po1. The adjusted R-squared is only 0.2433, which is not particularly great but still signficant because the USCrime data is based on real randomness of the cities and crime data, but is still not at a level that I would be fully confident in it.

```
M = 14.0
So = 0
Ed = 10.0
Po1 = 12.0

crime_rate = fit$coefficients[1] + M * fit$coefficients[2] + So * fit$coefficients[3] + Ed * fit$coeffi
crime_rate
```

```
## (Intercept)
##    1947.156
```

Based on the linear model, the crime rate for the given city is 1947.156 crime per 100,000 population. This is much greater than the value I got on last week's homework, which was a crime rate of 1380 per 100,000 population. That could be due to the differences in how each model treats multicollinearity and subsequent feature selection. For instance, the model I selected last week included U2, Ineq, and Prob, and did not include So, so features that were more multicorrelated were not included in this model.

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

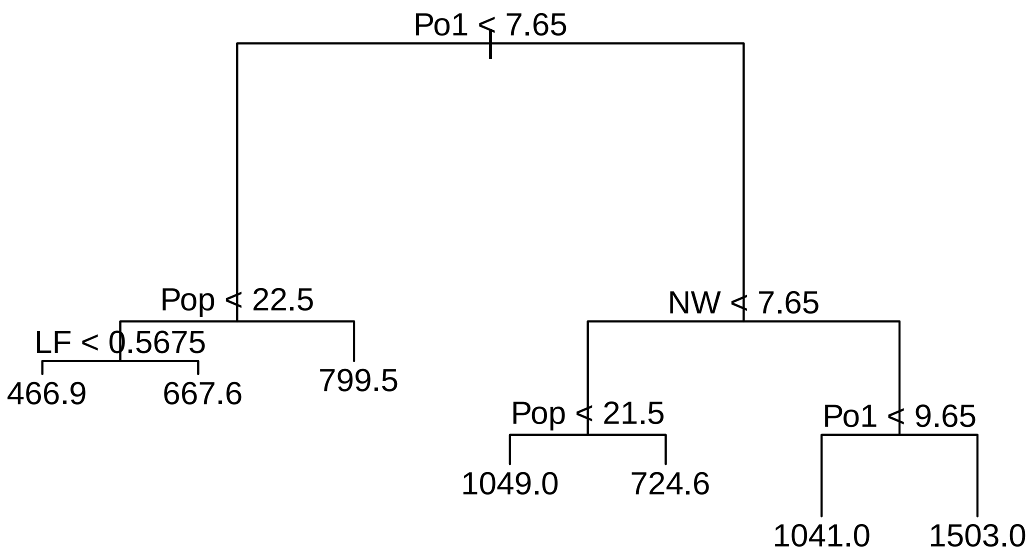
```r
## Regression tree model

# 5% of original data is rule of thumb
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.2.3
```

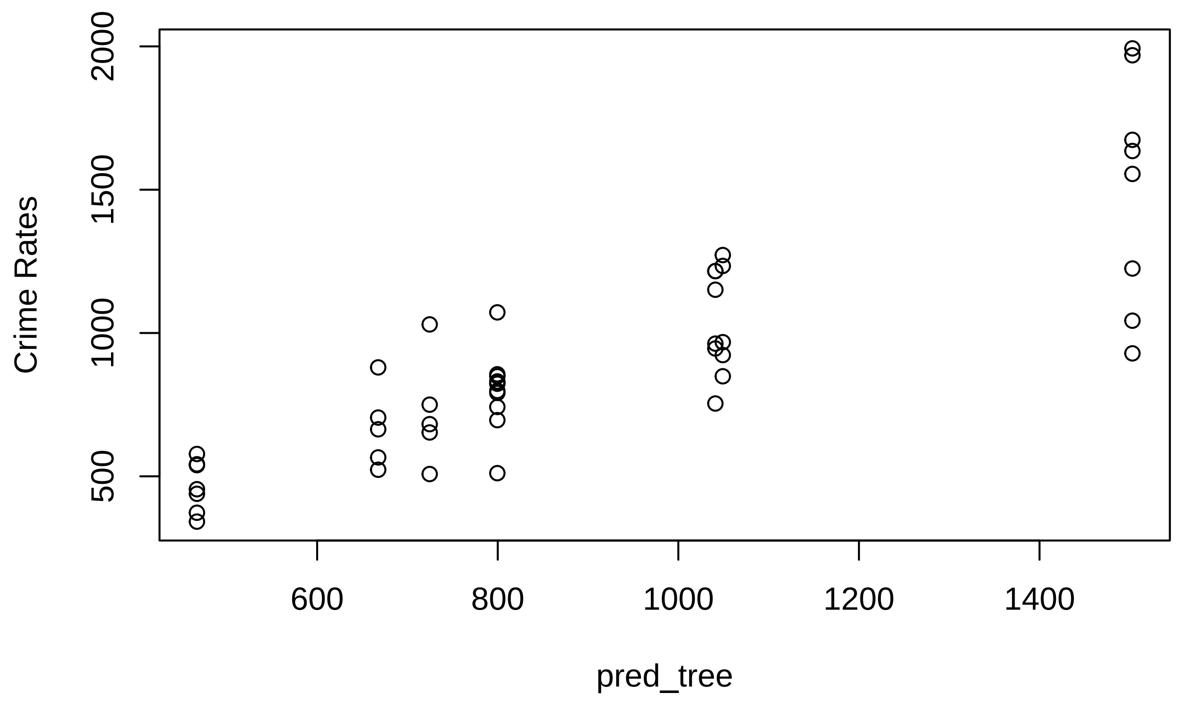
```r
c_tree <- tree(Crime~., data=crime)
summary(c_tree)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = crime)
## 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
plot(c_tree)
text(c_tree)
```



```r
# Make predictions based on tree
pred_tree <- predict(c_tree)
# Calculate r-squared
r_sq <- function (x, y) cor(x, y) ^ 2
plot(pred_tree, crime$Crime, ylab="Crime Rates")
```



```r
# Prune the tree training deviance
prune.tree(c_tree)$size
```

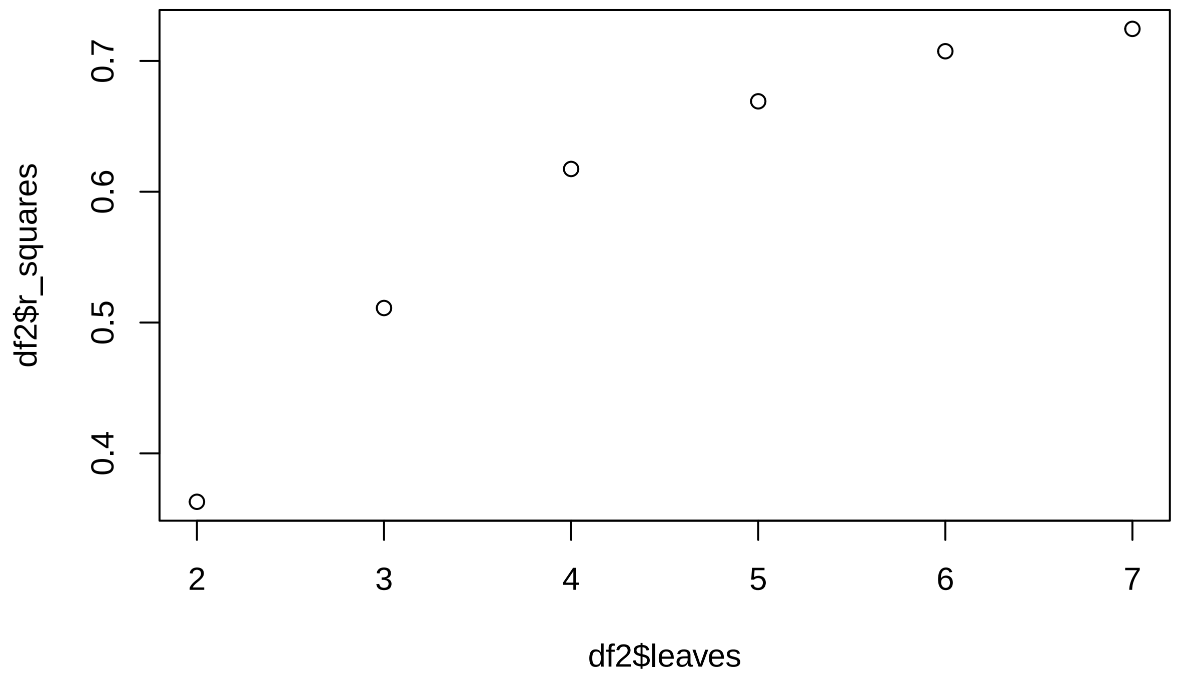
```
## [1] 7 6 5 4 3 2 1
```

```r
t <- prune.tree(c_tree)$dev
# CV deviance
v <- cv.tree(c_tree)$dev

library(knitr)
pruned <- data.frame(seq(1,7), t, v)
kable(pruned, format="simple", col.names = c("Num Leaves", "Training dev", "CV dev"))
```

Num Leaves	Training dev	CV dev
1	1895722	8174932
2	2013257	8244603
3	2276670	8328702
4	2632631	8464014
5	3364043	8914888
6	4383406	8669857
7	6880928	8187986

The deviance of the training data and cross-validation both seem to favor trees with less leaves, likely due to overfitting and the already small size of the dataset.

```r
# Now compare trees with different number of leaves
leaves <- seq(2,7)
r_squares <- rep(0, 6)

# Loop through each leaf model
for (l in 2:7){
  pruned <- prune.tree(c_tree, best = l)
  preds <- predict(pruned)
  r_squares[l-1] <- cor(preds, crime$Crime) ^ 2
}

df2 <- data.frame(leaves, r_squares)
plot(df2$leaves, df2$r_squares)
```



Based on the plot, the r-squared is the best for the tree with 7 leaves with an r-squared of 0.7244962 and that model would explain 72.45% of the variance. The r-squares seem to increase with the addition of each leaf. This is probably because more leaves increases the amount of variance that the model covers, but is also probably overfit as a result.

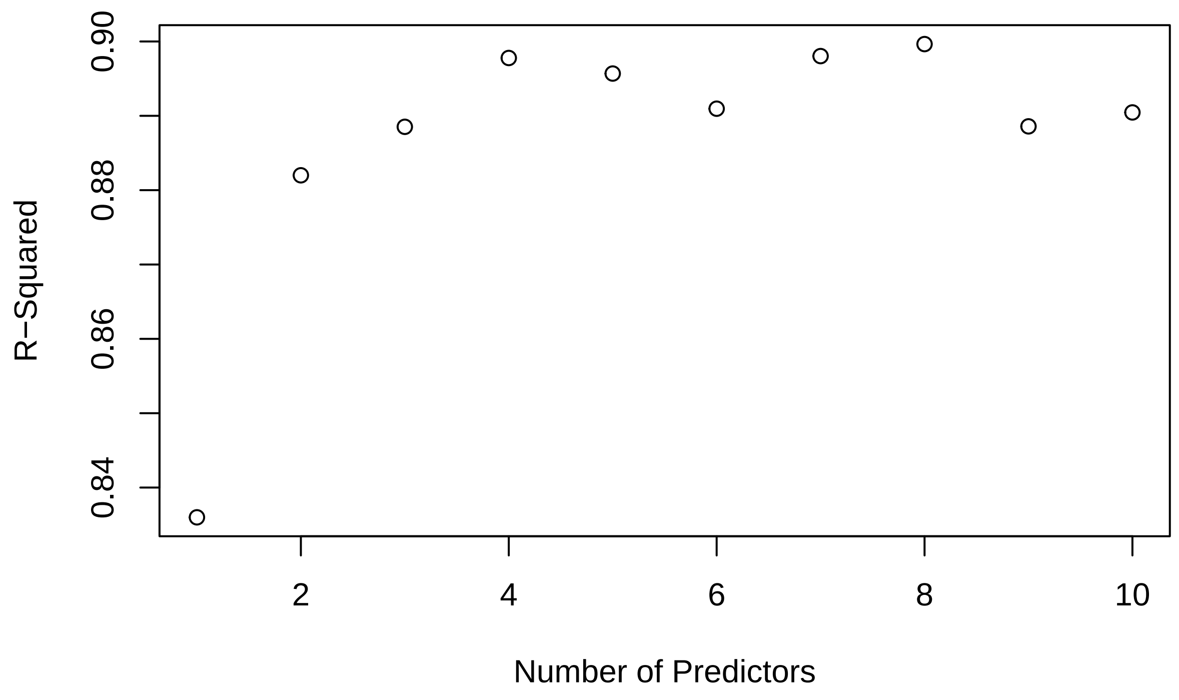
(b) a random forest model

```r
## Random forest model
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
```

```
# Initialize variables
r_sqs2 <- rep(0,10)
actual_c <- crime$Crime


# Loop through each number of predictors to get r-squared
for (p in 1:10) {
  c_ranfor <- randomForest(Crime~., data = crime,
                           mtry = p,
                           importance = TRUE,
                           ntree = 500)
  pred <- unname(predict(c_ranfor, crime))
  r_sq <- 1 - (sum((actual_c-pred)^2) / sum((actual_c-mean(actual_c))^2))
  # Add r-squared value
  r_sqs2[p] <- r_sq
}

df3 <- data.frame(1:10, r_sqs2)
plot(y=df3$r_sqs2, x=df3$X1.10, ylab="R-Squared", xlab="Number of Predictors")
```



Using the random forest, the best model has 9 predictors, with an R-squared of 0.8983468 and explains 89.83% of the data's variance. Again, with such a high r-squared value, I would be careful about overfitting and would need to train the model on different training sets and cross-validation before I'd be confident in the model.

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.

You could use a logistic regression model to predict whether or not someone will vote democratic or republican in an election. There's a five-factor model of personality that rates one's extraversion, agreeableness, conscientiousness, neuroticism, and openness to experience. Some research studies have found that these factors influence one's political orientation, so I'd be interested to see how much each of those factors affects someone's political leanings. Other demographic information like age, education, ethnicity, being rural, suburban, or urban could also be helpful predictors.

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.

```r
# Prep data, replace credit bad = 2 to bad = 0
gc <- read.table("germancredit.txt")
gc$V21 <- replace(gc$V21, gc$V21 == 2, 0)

# Split into train and test sets
indices <- sample(seq_len(nrow(gc)), size = 0.8 * nrow(gc))
train <- gc[indices, ]
test <- gc[-indices, ]

# 1 = Good, 0 = Bad
fit1 <- glm(V21 ~ ., data = train, family = "binomial")
summary(fit1)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5647  -0.7190   0.3514   0.6838   2.3652
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.829e-02  1.275e+00   0.022 0.982294
## V1A12        3.845e-01  2.487e-01   1.546 0.122135
## V1A13        1.242e+00  4.548e-01   2.732 0.006297 **
## V1A14        1.833e+00  2.672e-01   6.862 6.81e-12 ***
## V2          -2.134e-02  1.088e-02  -1.962 0.049795 *
## V3A31        3.440e-01  6.274e-01   0.548 0.583542
## V3A32        7.188e-01  5.022e-01   1.431 0.152322
## V3A33        1.148e+00  5.523e-01   2.079 0.037622 *
## V3A34        1.686e+00  5.135e-01   3.283 0.001025 **
## V4A41        1.974e+00  4.538e-01   4.350 1.36e-05 ***
## V4A410       1.387e+00  8.180e-01   1.696 0.089920 .
## V4A42        1.008e+00  3.024e-01   3.335 0.000854 ***
## V4A43        7.530e-01  2.747e-01   2.741 0.006122 **
## V4A44        3.919e-01  7.787e-01   0.503 0.614760
## V4A45       -5.007e-01  6.858e-01  -0.730 0.465338
## V4A46       -2.221e-01  4.537e-01  -0.489 0.624494
## V4A48        1.524e+01  4.417e+02   0.034 0.972480
## V4A49        6.906e-01  3.762e-01   1.836 0.066363 .
## V5          -1.160e-04  5.394e-05  -2.151 0.031449 *
## V6A62        3.459e-01  3.324e-01   1.041 0.298003
## V6A63        1.262e-01  4.151e-01   0.304 0.761062
## V6A64        1.295e+00  5.938e-01   2.181 0.029162 *
## V6A65        6.662e-01  2.915e-01   2.285 0.022304 *
## V7A72        3.122e-01  5.274e-01   0.592 0.553879
## V7A73        2.699e-01  5.118e-01   0.527 0.598031
## V7A74        1.277e+00  5.570e-01   2.292 0.021892 *
## V7A75        5.342e-01  5.156e-01   1.036 0.300232
## V8          -2.979e-01  1.019e-01  -2.925 0.003449 **
## V9A92        1.849e-01  4.296e-01   0.430 0.666897
## V9A93        6.550e-01  4.191e-01   1.563 0.118054
## V9A94        2.215e-01  4.949e-01   0.448 0.654396
## V10A102     -7.148e-01  4.611e-01  -1.550 0.121110
## V10A103      7.536e-01  4.659e-01   1.618 0.105752
## V11          9.935e-03  9.926e-02   0.100 0.920270
## V12A122     -2.753e-01  2.892e-01  -0.952 0.341133
## V12A123     -4.168e-01  2.703e-01  -1.542 0.123032
## V12A124     -9.321e-01  4.709e-01  -1.980 0.047748 *
## V13          1.399e-02  1.065e-02   1.313 0.189190
## V14A142      2.415e-01  4.698e-01   0.514 0.607207
## V14A143      8.360e-01  2.712e-01   3.082 0.002056 **
## V15A152      7.452e-01  2.655e-01   2.806 0.005012 **
## V15A153      1.042e+00  5.358e-01   1.946 0.051692 .
## V16         -3.659e-01  2.301e-01  -1.590 0.111840
## V17A172     -1.456e+00  8.364e-01  -1.741 0.081625 .
## V17A173     -1.526e+00  8.095e-01  -1.886 0.059347 .
## V17A174     -1.428e+00  8.129e-01  -1.757 0.078867 .
## V18         -3.927e-01  2.897e-01  -1.356 0.175234
## V19A192      2.283e-01  2.258e-01   1.011 0.312097
## V20A202      1.717e+00  7.415e-01   2.316 0.020569 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 972.25  on 799  degrees of freedom
## Residual deviance: 697.09  on 751  degrees of freedom
## AIC: 795.09
##
## Number of Fisher Scoring iterations: 14
```

As a rough estimate, I'll remove the coefficients where every category has $p < 0.5$, which is not very many and better feature selection where each categorical variable is split into dummy variables can be performed in the future.

```r
library(DAAG)
library(boot)
```

```
## Warning: package 'boot' was built under R version 4.2.3
```

```r
# Removed V7 V10 V11 V12 V13 V16 V17 V18 19 V20
fit2 <- glm(V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V14 + V15, data = train, family = "binomial")
summary(fit2)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V14 +
##     V15, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6482  -0.7492   0.3978   0.7214   2.1601
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.362e+00  7.419e-01  -1.836 0.066431 .
## V1A12        3.073e-01  2.324e-01   1.322 0.186166
## V1A13        1.086e+00  4.345e-01   2.500 0.012405 *
## V1A14        1.689e+00  2.536e-01   6.658 2.78e-11 ***
## V2          -2.233e-02  1.008e-02  -2.216 0.026703 *
## V3A31        4.629e-01  5.914e-01   0.783 0.433733
## V3A32        8.609e-01  4.712e-01   1.827 0.067693 .
## V3A33        1.084e+00  5.397e-01   2.008 0.044654 *
## V3A34        1.645e+00  4.959e-01   3.317 0.000911 ***
## V4A41        1.783e+00  4.274e-01   4.171 3.03e-05 ***
## V4A410       1.330e+00  7.584e-01   1.754 0.079437 .
## V4A42        7.745e-01  2.824e-01   2.743 0.006087 **
## V4A43        6.939e-01  2.610e-01   2.659 0.007842 **
## V4A44        2.894e-01  7.410e-01   0.391 0.696123
## V4A45       -4.994e-01  6.235e-01  -0.801 0.423166
## V4A46       -4.605e-01  4.388e-01  -1.049 0.294012
## V4A48        1.506e+01  4.472e+02   0.034 0.973133
## V4A49        6.223e-01  3.622e-01   1.718 0.085730 .
## V5          -1.132e-04  4.801e-05  -2.358 0.018350 *
## V6A62        2.527e-01  3.073e-01   0.822 0.410953
## V6A63        1.727e-01  3.997e-01   0.432 0.665686
## V6A64        1.185e+00  5.552e-01   2.134 0.032829 *
## V6A65        7.246e-01  2.749e-01   2.635 0.008406 **
## V8          -3.034e-01  9.553e-02  -3.176 0.001495 **
## V9A92        1.321e-01  4.063e-01   0.325 0.745079
## V9A93        6.371e-01  3.932e-01   1.620 0.105165
## V9A94        3.208e-01  4.735e-01   0.678 0.498066
## V14A142      4.778e-02  4.466e-01   0.107 0.914815
## V14A143      7.981e-01  2.582e-01   3.091 0.001994 **
## V15A152      7.083e-01  2.435e-01   2.908 0.003632 **
## V15A153      4.271e-01  3.626e-01   1.178 0.238863
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 972.25  on 799  degrees of freedom
## Residual deviance: 737.74  on 769  degrees of freedom
## AIC: 799.74
##
## Number of Fisher Scoring iterations: 14
```

```r
# Cross-validation to see quality of model
c1 <- cv.glm(train, fit1, K=5)
c2 <- cv.glm(train, fit2, K=5)

# Raw and adjusted cross-validation estimate of prediction error
c1$delta
```

```
## [1] 0.1706812 0.1675242
```

```r
c2$delta
```

```
## [1] 0.1700383 0.1680690
```

Based off of the AIC values, the model with selective coefficients seems to perform better with an AIC of 805.05 compared to the 817.15 of the original. The raw and adjusted cross-validation estimate of the prediction error also decreases with the second model, but those values could be overly optimistic and not perform as well on the testing data.

```r
set.seed(888)
# Make predictions using that model, rounding to be 0 or 1
predictions <- predict(fit2, test, type = "response")

# Calculate number of correct predictions
sum(test$V21 == round(predictions)) / nrow(test)
```

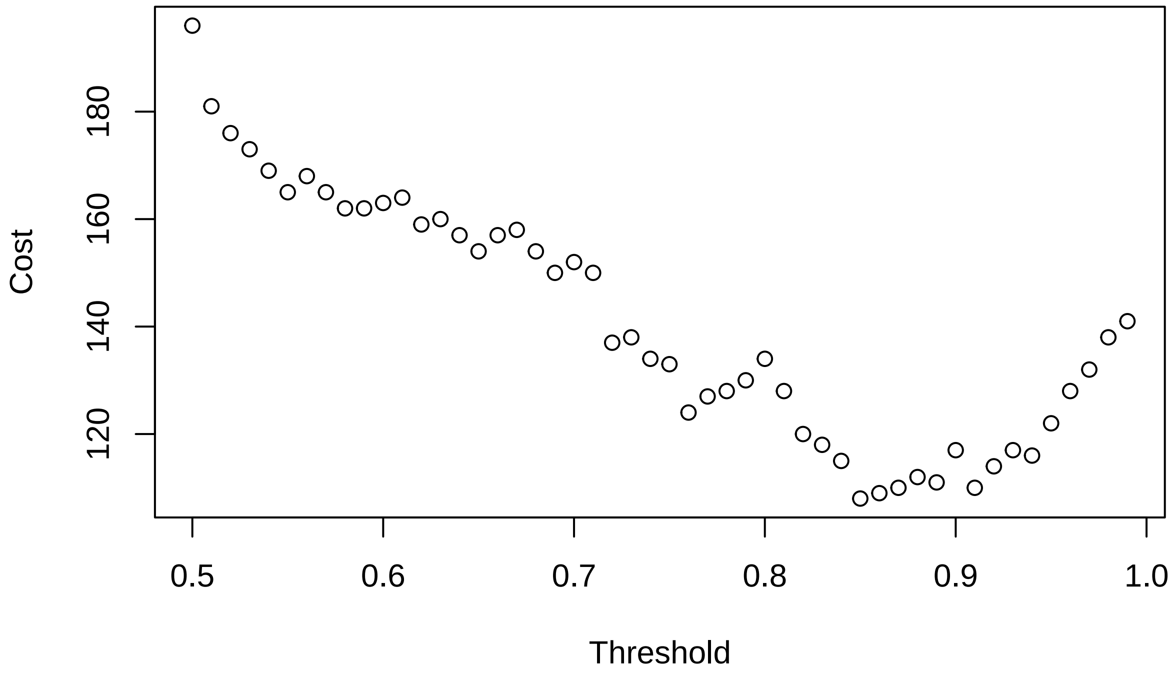
```
## [1] 0.74
```

After running the model on the test set, we only obtained a 75% accuracy. Further model tuning is needed in the future.

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.

For simplicity, I'm going to assume that the cost of incorrectly classifying a good customer as bad has a cost of 1 and incorrectly classifying a bad customer as good has a cost of 5. I will try to find a threshold that minimizes this "cost". In the confusion matrix, since I used good = 1 and bad = 0, the value in [0, 1] or conf_mat[3] is when a good customer is classified as bad and has an associated cost of 1. The value in [1,0] or conf_mat[2] is when a bad customer is classified as good and has a cost of 5.

```r
# Initialize thresholds from 0.5 to 0.99, and a costs dataframe to store cost for each threshold
thresholds <- seq(0.5, 0.99, by=0.01)
costs <- rep(0, length(thresholds))

# Loops through each threshold to calculate the cost
i = 1
for (t in thresholds) {
  above_thresh <- as.integer(predictions > t)
  conf_mat <- as.matrix(table(above_thresh, test$V21))
  costs[i] <- conf_mat[3] + conf_mat[2] * 5
  i <- i+1
}

# Plot costs by threshold
thresh_costs <- data.frame(thresholds, costs)
plot(x=thresh_costs$thresholds, y=thresh_costs$costs, xlab="Threshold", ylab="Cost")
```



From those calculations, it's clear that a threshold of 0.8 minimizes the cost of mis-classifying a customer. A threshold too high starts to turn away too many good customers, and a threshold too low accepts too many bad customers.