Logistic Regression

Ryan Porter

The goal of this problem is to figure out if credit applicants are a good credit risk or not. I used logistic regression to see predict what category they would fall in. The dataset is from UCI and it is comprised of german credit card applicants.

```
library(tidyverse)
library(caret)
library(pROC)




#setwd("/Users/Ryan/Desktop/DS")

credit_data = read.table("germancredit.txt.",
                         sep="",
                         fill=FALSE,
                         strip.white=TRUE,
                         header = FALSE)
```

Load in packages and the dataset used for this problem.

```
head(credit_data)
```

```
##    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   2
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   2
## 6 A192 A201   1
```

```
#convert 1&2 to 0&1
credit_data$V21[credit_data$V21==1] <- 0
credit_data$V21[credit_data$V21==2] <- 1

#train and test datasets
```

```r
credit_index <- createDataPartition(credit_data$V21, times = 1, p = 0.7, list = FALSE)
credit_train <- credit_data[credit_index,]
credit_test <- credit_data[-credit_index,]

#logistic regression model
credit_glm <- glm(V21 ~.,
                  family = binomial(link='logit'), #logistic into linear model
                  data = credit_train)

summary(credit_glm)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = credit_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1764  -0.7069  -0.3346   0.6163   2.5798
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.605e+00  1.402e+00  -1.145 0.252393
## V1A12       -3.096e-01  2.730e-01  -1.134 0.256761
## V1A13       -7.614e-01  4.431e-01  -1.718 0.085767 .
## V1A14       -1.629e+00  2.849e-01  -5.719 1.07e-08 ***
## V2           3.158e-02  1.159e-02   2.726 0.006416 **
## V3A31        1.080e+00  6.705e-01   1.611 0.107214
## V3A32       -2.122e-01  5.139e-01  -0.413 0.679662
## V3A33       -5.467e-01  5.680e-01  -0.963 0.335763
## V3A34       -1.330e+00  5.165e-01  -2.575 0.010027 *
## V4A41       -1.433e+00  4.506e-01  -3.180 0.001471 **
## V4A410      -1.940e+00  1.043e+00  -1.860 0.062874 .
## V4A42       -8.529e-01  3.283e-01  -2.598 0.009370 **
## V4A43       -8.802e-01  3.052e-01  -2.884 0.003932 **
## V4A44       -2.231e+00  1.252e+00  -1.783 0.074643 .
## V4A45       -9.660e-01  7.482e-01  -1.291 0.196683
## V4A46        1.665e-01  4.743e-01   0.351 0.725550
## V4A48       -2.032e+00  1.259e+00  -1.613 0.106643
## V4A49       -5.773e-01  3.967e-01  -1.455 0.145631
## V5           1.105e-04  5.567e-05   1.985 0.047184 *
## V6A62       -3.726e-01  3.509e-01  -1.062 0.288261
## V6A63       -1.748e-01  4.623e-01  -0.378 0.705429
## V6A64       -1.598e+00  6.784e-01  -2.355 0.018503 *
## V6A65       -1.233e+00  3.271e-01  -3.769 0.000164 ***
## V7A72        6.454e-02  5.248e-01   0.123 0.902121
## V7A73       -2.290e-01  5.091e-01  -0.450 0.652806
## V7A74       -8.042e-01  5.536e-01  -1.453 0.146321
## V7A75       -3.753e-01  5.034e-01  -0.746 0.455958
## V8           4.454e-01  1.116e-01   3.990 6.60e-05 ***
## V9A92        2.094e-01  4.993e-01   0.419 0.674942
## V9A93       -2.208e-01  4.911e-01  -0.450 0.652922
## V9A94       -3.280e-02  5.796e-01  -0.057 0.954870
## V10A102      3.076e-01  5.367e-01   0.573 0.566589
```

```
## V10A103     -1.385e+00  5.823e-01  -2.379 0.017350 *
## V11         -1.818e-02  1.073e-01  -0.169 0.865455
## V12A122      1.943e-02  3.175e-01   0.061 0.951200
## V12A123     -9.503e-02  2.941e-01  -0.323 0.746634
## V12A124      7.146e-01  4.982e-01   1.434 0.151442
## V13         -2.305e-02  1.164e-02  -1.981 0.047641 *
## V14A142      1.513e-01  4.951e-01   0.306 0.759851
## V14A143     -3.227e-01  3.105e-01  -1.039 0.298652
## V15A152     -1.868e-01  2.999e-01  -0.623 0.533505
## V15A153     -5.111e-01  5.866e-01  -0.871 0.383594
## V16          6.343e-01  2.490e-01   2.547 0.010863 *
## V17A172      8.754e-01  8.253e-01   1.061 0.288836
## V17A173      9.709e-01  7.998e-01   1.214 0.224775
## V17A174      9.831e-01  8.219e-01   1.196 0.231653
## V18          2.755e-01  3.173e-01   0.868 0.385301
## V19A192     -4.065e-01  2.464e-01  -1.650 0.098996 .
## V20A202     -1.641e+01  7.009e+02  -0.023 0.981319
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 855.21  on 699  degrees of freedom
## Residual deviance: 601.55  on 651  degrees of freedom
## AIC: 699.55
##
## Number of Fisher Scoring iterations: 16
```

```
AIC(credit_glm)
```

```
## [1] 699.5503
```

We start off by converting the response variable to 0 and 1. Then after that we split the data into train and test set to run our models on. We just run a simple linear regression and see that out AIC score is higher than last weeks score (658). We arent going to do any variable selection in this model so it will have to do.

```
yhat <- predict(credit_glm, credit_test, type="response")

roc(credit_test$V21, round(yhat))
```

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

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

```
##
## Call:
## roc.default(response = credit_test$V21, predictor = round(yhat))
##
## Data: round(yhat) in 210 controls (credit_test$V21 0) < 90 cases (credit_test$V21 1).
## Area under the curve: 0.6563
```

```
thresh <- 0.7
yhat_thresh <- as.integer(yhat > thresh)
conf_matrix <- as.matrix(table(yhat_thresh, credit_test$V21))
conf_matrix
```

```
##
## yhat_thresh   0   1
##           0 203  69
##           1   7  21
```

```
cost <- 202*(-1) + 66*(5) + 7*(1) + 25*(0)
cost
```

```
## [1] 135
```

```
# accuracy
accuracy <- (conf_matrix[1,1] + conf_matrix[2,2]) / sum(conf_matrix)
accuracy
```

```
## [1] 0.7466667
```

```
# sensitivity
sensitivity <- (conf_matrix[1,1]) / (conf_matrix[1,1] + conf_matrix[2,1])
sensitivity
```

```
## [1] 0.9666667
```

For coming up with a threshold it was a game of guess and check method. I was looking for thresholds that would make my cost function low and my accuracy as high as possible without overfitting. We see that the area under the curve is 0.6796. The confusion matrix shows us the classfication of the responses which we'll use to see how accurate we are. Using the confusion matrix we can calculate the cost function, I looked up how to do the cost function so hopefully it's accruate. But you can see that '66*(5)' is accounting for the 5 times worse if a bad customer is identifed as good. So those 66 people are the ones that are missclassifed as good credit users when in reality it is the opposite. The accuracy is 0.75 which isnt too bad but still a lot of roon for improvement. I want to be somewhere in the 80% but I couldn't find a combination that would give me that. This is where feature selection would have made that 80% possible. Our sensitivity is really high, 0.9665, but that would be at the cost of accuracy.