ANLY 512: Homework 4

Brody Vogel

2/15/2018

Preparation

```
set.seed(1234)
library(ISLR)
library(MASS)
```

Problem 1 (#6)

a)

$p(x) = \frac{e^{-6+.05\times 40+1\times 3.5}}{1+e^{-6+.05\times 40+1\times 3.5}} \approx .3775$

There's almost a 38% chance the student gets an A.

b)

$.5 = \frac{e^{-6+.05\times x+1\times 3.5}}{1+e^{-6+.05\times x+1\times 3.5}}$

$.5 \times (1 + e^{-6+.05\times x+1\times 3.5}) = e^{-6+.05\times x+1\times 3.5}$

$.5 + .5 \times (e^{-6+.05\times x+1\times 3.5}) = e^{-6+.05\times x+1\times 3.5}$

$.5 = .5 \times e^{-6+.05\times x+1\times 3.5}$

$1 = e^{-6+.05\times x+1\times 3.5}$

$log(1) = log(e^{-6+.05\times x+1\times 3.5})$

$0 = -2.5 + .05 \times x$

$2.5 = .05 \times x$

$50 = x$

The student would have to study an estimated 50 hours to have a 50% chance of getting an A.

Problem 2 (#13 Logistic Only)

I first built the training and test datasets by splitting the Boston data in two - half for training the model and half for testing. I then checked for correlation between crime rate and the other variables, adding those to the model that were most correlated. When I ran this through prediction testing, it did fairly well - the model accurately predicted whether the crime rate for a given entry would be above or below the median with about 89% accuracy.

However, some of the variables that I included in the logistic model were not significant: lstat, medv, and dis (at the p = .05 level). So I took those out. This got my error rate below 10%. This time, also, the predictor "black" was not significant, so I threw it out too. That didn't really affect the model's predictive power.

My residuals definitely showed a pattern, though, and my Q-Q plot was far from normal. So I added a log-transform to the variables with the largest values - tax and nox. This got my error rate down to about 8%, but it didn't improve my plots. This, then, was where I stopped, with the model: $\hat{p(HighCrime)} = \frac{e^{\beta_0+\beta_1 indus+\beta_2 rad+\beta_3 log(tax)+\beta_4 log(nox)}}{1+e^{\beta_0+\beta_1 indus+\beta_2 rad+\beta_3 log(tax)+\beta_4 log(nox)}}$.

```
data(Boston)

crime.fac <- rep(0, length(Boston$crim))
crime.fac[Boston$crim > median(Boston$crim)] = 1

Boston <- data.frame(Boston, crime.fac)

train = Boston[1:(length(Boston$crim)%/%2), ]
test = Boston[(length(Boston$crim)%/%2 + 1):length(Boston$crim), ]

#cor(Boston)

glm.fit <- glm(crime.fac~. -tax + log(tax) -nox + log(nox) - crim - zn - chas - rm - age - black - ptra

probs <- predict(glm.fit, test, type = 'response')

pred <- rep(0, length(probs))

pred[probs > .5] = 1

mean(pred != test$crime.fac)
```

```
## [1] 0.08300395
```

```
#summary(glm.fit)

par(mfrow = c(2,2))

#plot(glm.fit)
```

Part 2

I grabbed all the indexes of train_y that correspond to 0 or 1. I then built subsets of train_x and train_y that only contain those rows, my.trainx and my.trainy

```
mnist <- load('/Users/brodyvogel/Downloads/mnist_all.RData')

plot_digit <- function(j){
  arr784 <- as.numeric(train$x[j,])
  col = gray(12:1/12)
  image(matrix(arr784, nrow=28)[,28:1], col = col,
              main = paste("this is a ", train$y[j]))
}

#plot_digit(34)

z1 <- which(train$y %in% c(0,1))
my.trainy <- train$y[z1]
```

```
my.trainx <- train$x[z1, ]
```

Problem 1

I grabbed the column of my.trainx with the most variance (407), and used it to construct the model: $\hat{p}(Image = 1) = \frac{e^{-4.358+.0367\times column_{407}}}{1+e^{-4.358+.0367\times column_{407}}}$.

Nearly every entry was 0, so it was difficult to get a false positive rate above .04. In my estimation, if the FPR were .1, the TPR would be essentially 1.

```
my.col <- which.max(apply(my.trainx, 2, var))

glm.fit1 <- glm(my.trainy~my.trainx[, my.col], family = binomial)

summary(glm.fit1)
```

```
##
## Call:
## glm(formula = my.trainy ~ my.trainx[, my.col], family = binomial)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1328  -0.1535   0.1218   0.1263   2.9825
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -4.4357927  0.1131475  -39.20   <2e-16 ***
## my.trainx[, my.col]  0.0366105  0.0007086   51.66   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 17504.4  on 12664  degrees of freedom
## Residual deviance:  1704.2  on 12663  degrees of freedom
## AIC: 1708.2
##
## Number of Fisher Scoring iterations: 7
```

```
probs <- predict(glm.fit1, data.frame(my.trainx), type = 'response')

pred <- rep(0, length(probs))

pred[probs > .012] = 1

t <- table(my.trainy, pred)

t
```

```
##          pred
## my.trainy    0    1
##         0 5692  231
##         1   32 6710
```

```r
fpr <- t[1, 2]/(t[1, 2] + t[1, 1])

tpr <- t[2, 2]/(t[2, 2] + t[2, 1])

fpr
```

```
## [1] 0.03900051
```

```r
tpr
```

```
## [1] 0.9952536
```

Problem 2

It looks like columns 404 and 443 of my.trainx are lightly correlated (about .294). When I fit a model to the training data using these two variables, that model actually did a very good job classifying the digits as 1s or 0s; it achieved an AUC of .8759. Surprisingly, this didn't improve when I added an interaction term between the correlated variables.

In the plot, you can see that almost every 1 in the training data had a value of 0 for variable 443. Moreover, you can see that the majority of observations had 0s for both variable 403 and 443. Furthermore, as you can see from the confusion matrix, the model usually predicted a 0; and an even greater percentage of the observations were actually 0. My thinking, then, is that the model predicted 0 if an entry's variables 403 and 443 were both sufficiently close ot 0, and predicted a 1 otherwise. And, as per the aforementioned AUC, this did pretty well.

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
glm.fit2 <- glm(my.trainy~my.trainx[, 404]+my.trainx[, 443] , family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

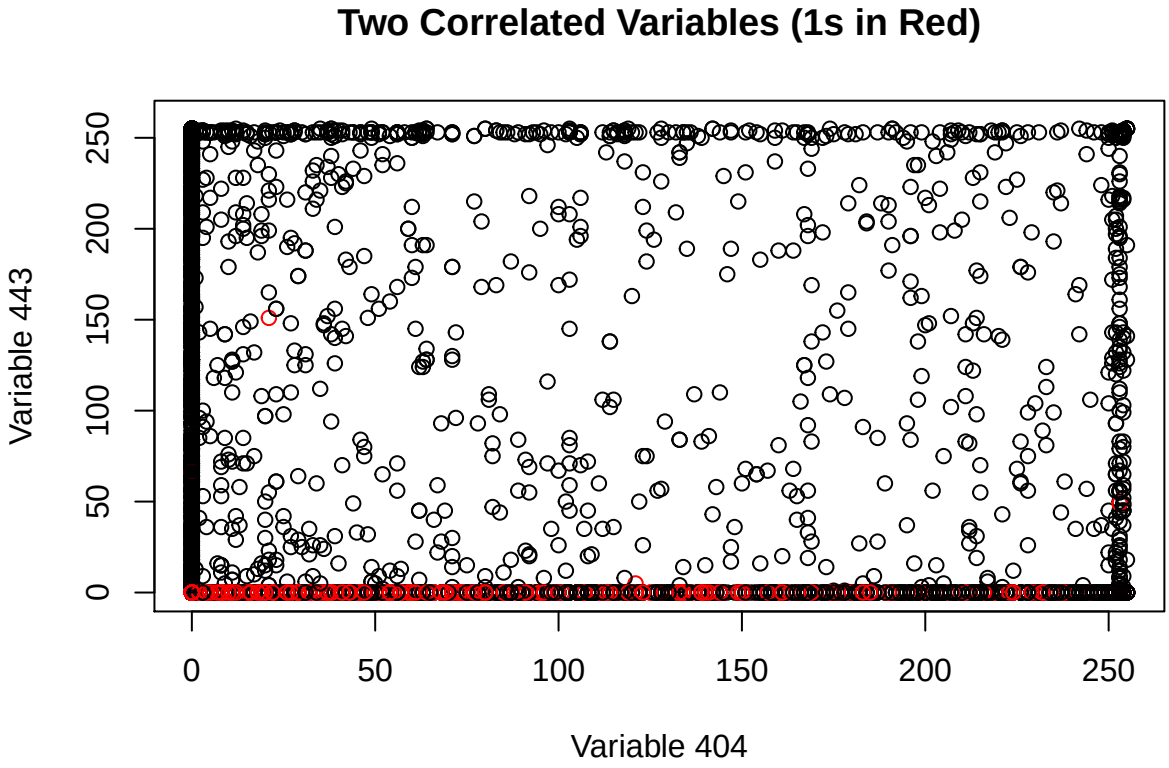
```r
probs <- predict(glm.fit2, data.frame(my.trainx), type = 'response')

pred <- rep(0, length(probs))

pred[probs > .5] = 1

table(pred, my.trainy)
```

```
##     my.trainy
## pred    0    1
##    0 4681  259
##    1 1242 6483
```

```r
auc(my.trainy, pred)
```

```
## Area under the curve: 0.8759
```

```r
plotDF <- data.frame(my.trainx[, 404], my.trainx[, 443], my.trainy)

plot(plotDF[, 1], plotDF[, 2], col = plotDF[, 3] + 1, ylim = c(0, 260), xlab = 'Variable 404', ylab = '
```



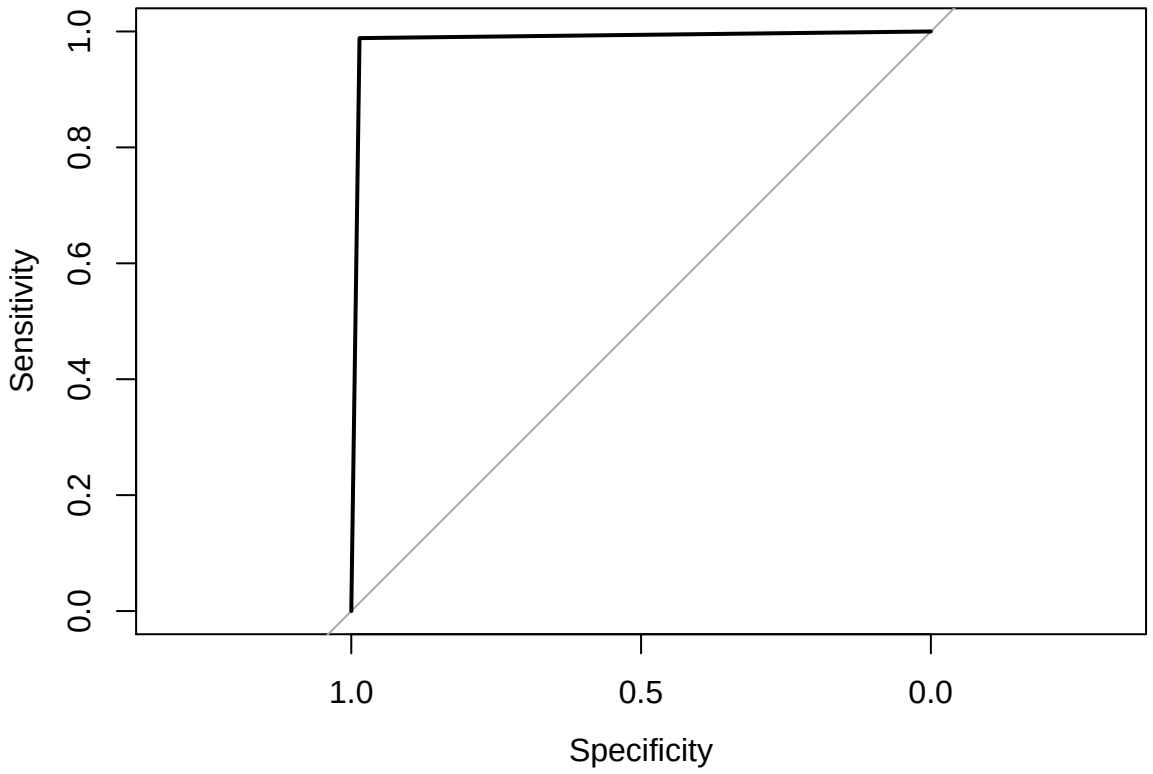
Problem 3

The classifier did very well - *on the training data*. It produced an AUC of .9872. But, as you can see from the ROC curve, it performed identically at every level of sensitivity/specificity. This means that the model has massively overfit to the training data, and would probably be next to useless on a test data set.

```r
order(-apply(my.trainx, 2, var))[1:10]
```

```
##  [1] 407 435 379 463 462 352 351 380 490 434
```

```r
glm.fit3 <- glm(my.trainy~my.trainx[, 407]+my.trainx[, 435]+my.trainx[, 379]+my.trainx[, 463]+my.trainx

probs <- predict(glm.fit3, data.frame(my.trainx), type = 'response')

pred <- rep(0, length(probs))

pred[probs > .5] = 1

r <- roc(my.trainy, pred)

plot(r)
```




```
auc(my.trainy, pred)
```

```
## Area under the curve: 0.9872
```