Logistic Regression in a Nutshell

Ted Laderas and Shannon McWeeney

August 18, 2016

Here are some tips on getting started on running your analysis. One of the skills you will need to bring as a data scientist is interpreting the results. Here we use logistic regression as a way to predict readmissions. However, there are multiple other routes to work with the data, such as:

- Looking at groupings in the data (clustering)
- Linear regression
- Machine Learning to learn the data (Support Vector Machines, Linear Discriminant Analysis, Classification Trees)
- Can you think of anything else?

Exploratory Data Analysis (EDA) of Our Data

Before we even start to model, we want to have an idea of how our outcome (30 day readmission, `Readmit30`) is distributed among our scores. One simple way to do this is to do cross-tabs of the data. Below, we calculat4 a cross-table between `LScore` and `Readmit30`. Looking at the second row (`Readmit30` = 1), we note that as `LScore` increases, the proportion of readmits/nonReadmits increases.

```
#load analytic data
analytic <- read.delim("analytic-table-LACE.txt")

#because AScore is categorical (0/3), we need to cast it as a factor:
analytic$AScore <- factor(analytic$AScore)

#cross tab of LScore - is there a correlation between Readmit30 and LScore?
readmitLscoreTab <- table(analytic$Readmit30, analytic$LScore)

readmitLscoreTab
```

```
##
##        1    2    3    4    5    7
##   0 4315 4150 4248 7640 6133 2877
##   1  458  494  550 1161 1192 1314
```

```
#calculate proportions between readmits=TRUE and readmits=FALSE
#note that as L increases, this proportion increases.
readmitLscoreTab[2,]/readmitLscoreTab[1,]
```

```
##         1         2         3         4         5         7
## 0.1061414 0.1190361 0.1294727 0.1519634 0.1943584 0.4567258
```

Let's also do a crosstab between `AScore` and `Readmit30`.

```
table(analytic$Readmit30, analytic$AScore)
```

```
##
##          0     3
##   0 16244 13119
##   1  2016  3153
```

Separating a small portion of the data out for testing

One of the things we might like to check is the predictive power of the model. For this reason, we want to save a little bit of our data that the model doesn't "see" for testing the predicitve power of the model.

```r
#how many rows?
nrow(analytic)
```

```
## [1] 34532
```

```r
#show analytic table
analytic[1:10,]
```

```
##    patientid Event_ID encounter_type         outcome Admit_date
## 1          1      108             22             SNF 2014-01-01
## 2          2     1333             22 Discharged Home 2014-01-01
## 3          3       71             22             SNF 2014-01-01
## 4          4      886             22 Discharged Home 2014-01-01
## 5          5       73             22 Discharged Home 2014-01-01
## 6          6       98             22             SNF 2014-01-01
## 7          7      893             22           Rehab 2014-01-01
## 8          8     2556             22             SNF 2014-01-01
## 9          9      649             22             SNF 2014-01-01
## 10        10      979             22             SNF 2014-01-01
##    Discharge_date   Admit_source indexadmit Readmit30 LengthOfStay LScore
## 1      2014-01-08 Emergency Room          1         1            7      5
## 2      2014-01-13         Clinic          1         0           12      5
## 3      2014-01-07       Transfer          1         0            6      4
## 4      2014-01-07 Emergency Room          1         0            6      4
## 5      2014-01-08 Emergency Room          1         1            7      5
## 6      2014-01-08 Emergency Room          1         1            7      5
## 7      2014-01-16 Emergency Room          1         0           15      7
## 8      2014-01-06       Transfer          1         0            5      4
## 9      2014-01-08 Emergency Room          1         0            7      5
## 10     2014-01-04 Emergency Room          1         0            3      3
##    AScore EScore MyoComorbid DCComorbid CScore LACE
## 1       3      0           2          0      2   10
## 2       0      0           0          0      0    5
## 3       0      0           0          0      0    4
## 4       3      0           0          0      0    7
## 5       3      0           0          0      0    8
## 6       3      0           0          0      0    8
## 7       3      0           0          0      0   10
## 8       0      0           0          0      0    4
## 9       3      0           2          0      2   10
## 10      3      0           0          0      0    6
```

We hold out 10 percent of the data by using the `sample()` function. `sample()` returns a number of rows that we can use to subset the data into two sets: 1) our `test` dataset (10% of our data), which we'll use to test our model's predictive value, and 2) our `training` dataset (90% of our data), which we'll use to actually build (or train) our model.

```
dataSize <- nrow(analytic)
#
testSize <- floor(0.1 * dataSize)
testSize
```

```
## [1] 3453
```

```
#build our sample index from our row numbers
testIndex <- sample(dataSize,size = testSize,replace = FALSE)
#show first 10 indices (each of these corresponds to a row number in analytic)
testIndex[1:10]
```

```
##  [1] 16075 14259 31319  4735 25510 33710 30382  4032 18861  4838
```

```
#confirm that length(testIndex) = testSize
length(testIndex)
```

```
## [1] 3453
```

Now that we have our indices for our testSet, we can do the subsetting into our `testSet` and our `trainSet`. For our `testSet`, we can simply use our index numbers to subset the data.

For our `trainSet`, we can use the `-` (minus) operator. `-testIndex` is an operation that returns all the rows in the `analytic` table that isn't in `testIndex`. Note that this only works within the context of a `data.frame`, `vector` or `matrix`.

```
#build testSet (we'll use this later)
testSet <- analytic[testIndex,]

#show first 10 rows of testSet (compare row numbers to testIndex[1:10])
testIndex[1:10]
```

```
##  [1] 16075 14259 31319  4735 25510 33710 30382  4032 18861  4838
```

```
testSet[1:10,]
```

```
##       patientid Event_ID encounter_type         outcome Admit_date
## 16075     16075    18149             22             SNF 2014-01-01
## 14259     14259    16447             22 Discharged Home 2014-01-01
## 31319     31319    35840             22             SNF 2014-01-01
## 4735       4735     5052             22             SNF 2014-01-01
## 25510     25510    31033             22           Rehab 2014-01-01
## 33710     33710    38398             22             SNF 2014-01-01
## 30382     30382    34763             22 Discharged Home 2014-01-01
## 4032       4032     4289             22 Discharged Home 2014-01-01
## 18861     18861    20668             22 Discharged Home 2014-01-01
```

```
## 4838       4838     5698             22            SNF 2014-01-01
##       Discharge_date   Admit_source indexadmit Readmit30 LengthOfStay
## 16075     2014-01-02 Emergency Room          1         0            1
## 14259     2014-01-20         Clinic          1         1           19
## 31319     2014-01-05 Emergency Room          1         1            4
## 4735      2014-01-15       Transfer          1         0           14
## 25510     2014-01-08            SNF          1         0            7
## 33710     2014-01-20       Transfer          1         0           19
## 30382     2014-01-02         Clinic          1         0            1
## 4032      2014-01-04         Clinic          1         1            3
## 18861     2014-01-09         Clinic          1         0            8
## 4838      2014-01-03       Transfer          1         0            2
##       LScore AScore EScore MyoComorbid DCComorbid CScore LACE
## 16075      1      3      0           0          0      0    4
## 14259      7      0      0           2          0      2    9
## 31319      4      3     NA           0          0      0   NA
## 4735       7      0      0           0          0      0    7
## 25510      5      0      0           0          0      0    5
## 33710      7      0      0           0          0      0    7
## 30382      1      0      0           0          0      0    1
## 4032       3      0      0           0          0      0    3
## 18861      5      0      0           0          0      0    5
## 4838       2      0      0           0          0      0    2
```

```
#confirm that number of rows in testSet is equal to testSize
nrow(testSet)
```

```
## [1] 3453
```

```
#build training set (used to build model)
trainSet <- analytic[-testIndex,]
```

The Formula Interface for R

One of the most confusing things about R is the formula interface. The thing to remember is that formulas have a certain form. If `Y` is our dependent variable and `X1`, `X2` are independent variables, then the formula to predict `Y` has the format `Y ~ X1 + X2`. Usually these variables come from a data.frame, which is supplied by the `data` argument to the function. Note that we don't need quotes to refer to the variables in the data.frame.

Logistic Regression

Here we load the analytic data with our LACE scores and perform a logistic regression using `Readmit30` as our dependent variable and our `LScore` and `AScore` as our independent variables. A logistic regression is a type of regression where the *outcome*, or *dependent variable* is related to the probability of categorical variable being true (in our case, whether a patient was a readmission or not). The output is a model that predicts, for our example, readmission.

```
#show variable names in analytic data.frame
colnames(trainSet)
```

```
##  [1] "patientid"      "Event_ID"       "encounter_type" "outcome"
##  [5] "Admit_date"     "Discharge_date" "Admit_source"   "indexadmit"
##  [9] "Readmit30"      "LengthOfStay"   "LScore"         "AScore"
## [13] "EScore"         "MyoComorbid"    "DCComorbid"     "CScore"
## [17] "LACE"
```

```r
#run a simple logistic regression model just using LScore and Ascore
#we can cast AScore as categorical data using factor()
laModel <- glm(Readmit30 ~ LScore + AScore, family = "binomial", data=trainSet)
```

Interpreting Logistic Regression Models

Let's look at the output of our model. This gives us the coefficients on the logit scale.

```r
#Summarize the model
summary(laModel)
```

```
##
## Call:
## glm(formula = Readmit30 ~ LScore + AScore, family = "binomial",
##     data = trainSet)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9115  -0.6532  -0.4769  -0.3727   2.4264
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.147185   0.047902  -65.70   <2e-16 ***
## LScore       0.257597   0.009259   27.82   <2e-16 ***
## AScore3      0.680471   0.033095   20.56   <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: 26084  on 31078  degrees of freedom
## Residual deviance: 24872  on 31076  degrees of freedom
## AIC: 24878
##
## Number of Fisher Scoring iterations: 5
```

```r
#grab coefficients themselves
coef(laModel)
```

```
## (Intercept)      LScore     AScore3
##  -3.1471845   0.2575968   0.6804705
```

We note that both of our predictors (`LScore` and `AScore`) are significant terms in our model, which indicates that they are useful predictors in our model. For example, our L-score p-value is less than 2 x 10^-16, which is highly significant.

How can we use these coefficients? You cannot interpret the Logistic model coefficients strictly in terms of probabilities, because the logistic model is non-linear in terms of probabilities.

However, the coefficients in the model can be interpreted in terms of Odds Ratio. What is the Odds Ratio? Remember that odds can be expressed as `numberCases:numberNonCases`. For example, an odds of 5:1 (win:lose) means that 5 times out of 6, we expect to win, and 1 times out of 6 we expect not to win. The odds ratio (OR) is just `numberCases/numberNonCases`. In the case of 5:1 odds, the OR is 5/1 = 5. The probability of winning in this case would be `numberCases / (numberCases + numberNonCases)` or 5/(1+5) = 0.833333.

So mathematically, the Odds Ratio for our model using our `LScore` as an independent variable can be calculated as:

$$OddsRatio = \frac{prob(readmit = TRUE)}{prob(readmit = FALSE)} = \frac{numberReadmits}{numberNonReadmits}$$

Since

$$oddsRatio(Readmit = 1) = \frac{prob(Readmit = 1)}{prob(Readmit = 0)} = \frac{prob(Readmit = 1)}{1 - prob(Readmit = 1)}$$

because

$$prob(Readmit = 0) = 1 - prob(Readmit = 1)$$

by definition. So, we can define our logistic model as:

$$log(OddRatio(Readmit = TRUE)) = Constant + CoefL * LScore + CoefA * Ascore$$

We call `log(OddsRatio(Readmit=TRUE))` the logit. Notice that the logit has a linear relation to our `LScore` and our `AScore`. Our model parameters are a `Constant`, `CoefL` is the fitted model coefficient for our `LScore`, and `CoefA` is the fitted model coefficient for our `AScore`.

if we exponentiate both sides, remembering that `exp(A+B) = exp(A) * exp(B)`:

$$OddsRatio(Readmit = TRUE) = exp(Constant + CoefL * LScore + CoefA * AScore)$$

Moving things around, we get:

$$OddsRatio(Readmit = TRUE) = exp(\frac{prob(Readmit = TRUE)}{1 - prob(Readmit = TRUE)}) = exp(Constant)*exp(CoefL*LScore)*exp(CoefA*A$$

So we find that the `OddsRatio(Readmit = TRUE)` is calculated by multiplying `exp(Constant)`, `exp(CoefL * LScore)` and `exp(CoefA * AScore)`, which is a nice result. This means that in order to interpret the coefficients in terms of odds, we need to exponentiate them. We can then interpret these transformed coefficients in terms of an associated increase in the Odds Ratio. So let's first transform our coefficients by exponentiate them.

```
coefs <- coef(laModel)
expCoefs  <- exp(coefs)
expCoefs
```

```
## (Intercept)      LScore     AScore3
##  0.04297295  1.29381709  1.97480670
```

Looking at the exponentiated coefficient for `LScore`, this means that for a 1 unit increase in `LScore`, the `OddsRatio(Readmit score)` is increased by 29.3817094 percent. This means that if you want to interpret the coefficients in the model in terms of increases in units, you need to first multiply the unit increase by the coefficient and then exponentiate. For example, going from 1 to 6 in our LScore (a 5 unit increase), our odds ratio increases by `exp(5 * CoefL)` or 3.6254709.

The interpretation of the `AScore` is different because we're treating it as a categorical variable (remember, if *no admission through ER* = 0, and *admission through ER* = 3). If there is an admission through emergency room, there is a 97.4806695 percent increase in our predicted Odds Ratio.

What happens to the model if we don't treat `AScore` as a categorical variable (you can cast this variable as a number by using `as.numeric()`)? Try it out!

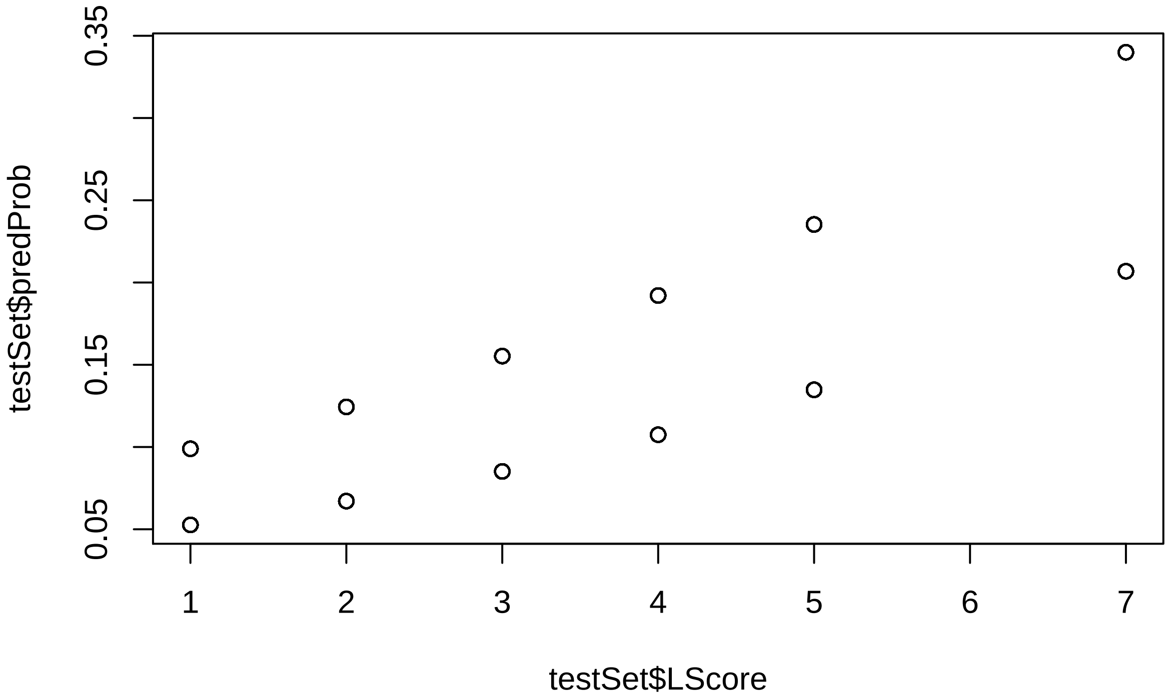
Using models for prediction on our test set

Now you have built a model. What next? If you want to see the values that the model predicts for the dependent variable, you can use `predict()`. This command will return two types of values, based on the arguments we pass it. Either 1) *predicted probabilities* or 2) the *Log(Odds Ratio)*.

If you look at the help entry for `predict.glm` it mentions that by setting the option of `type` to be `response`, you can directly get the predicted probabilities (that is, `prob(readmit=TRUE)`) from the model. We'll use these later when we compare the misclassification rates in our model.

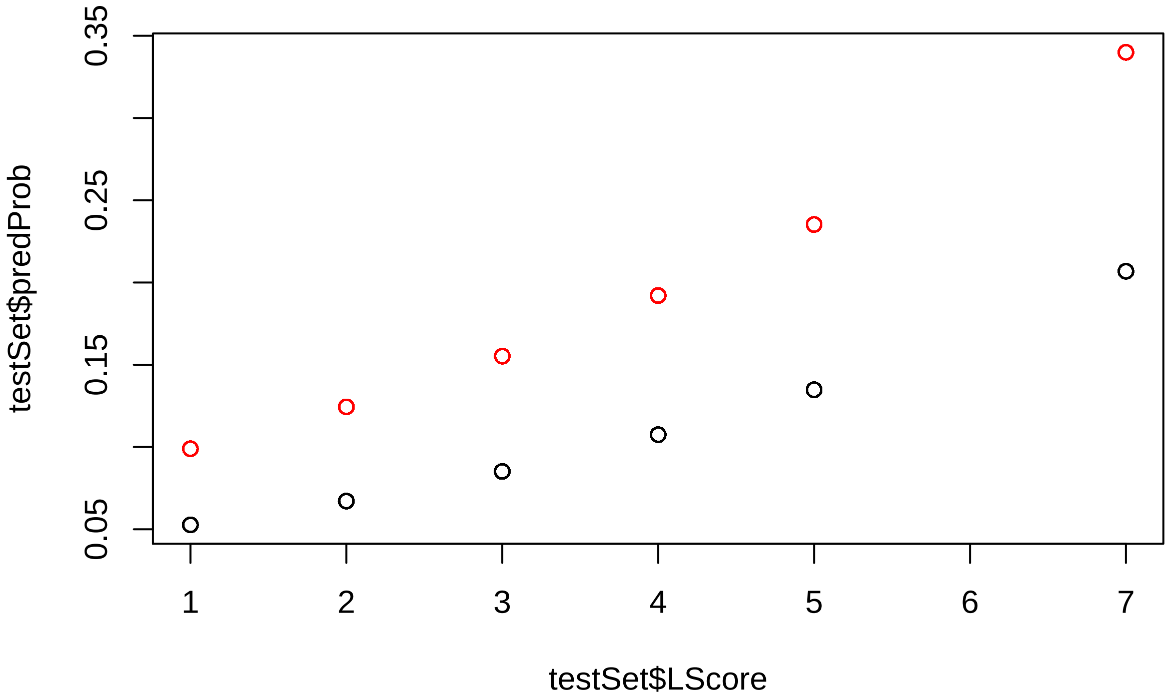
The interpretation of the `AScore` is different because we're treating it as categorical data (remember, if *no admission through ER* = 0, and *admission through ER* = 3). If there is an admission through emergency room, there is a 97.4806695 percent increase in our predicted Odds Ratio.

```r
modelPredictedProbabilities <- predict(laModel, newdata=testSet, type = "response")

##add the modelPredictedProbabilities as a column in testSet

testSet <- data.frame(testSet, predProb=modelPredictedProbabilities)

plot(testSet$LScore, testSet$predProb)
```



Looking at this plot, we can see two things: our predicted probabilities are in two groups of points. The bottom set of points are the ones for which our `AScore` is 0, and the top set of points are where `AScore` is 3. This becomes more obvious if we color the points according by `AScore`.

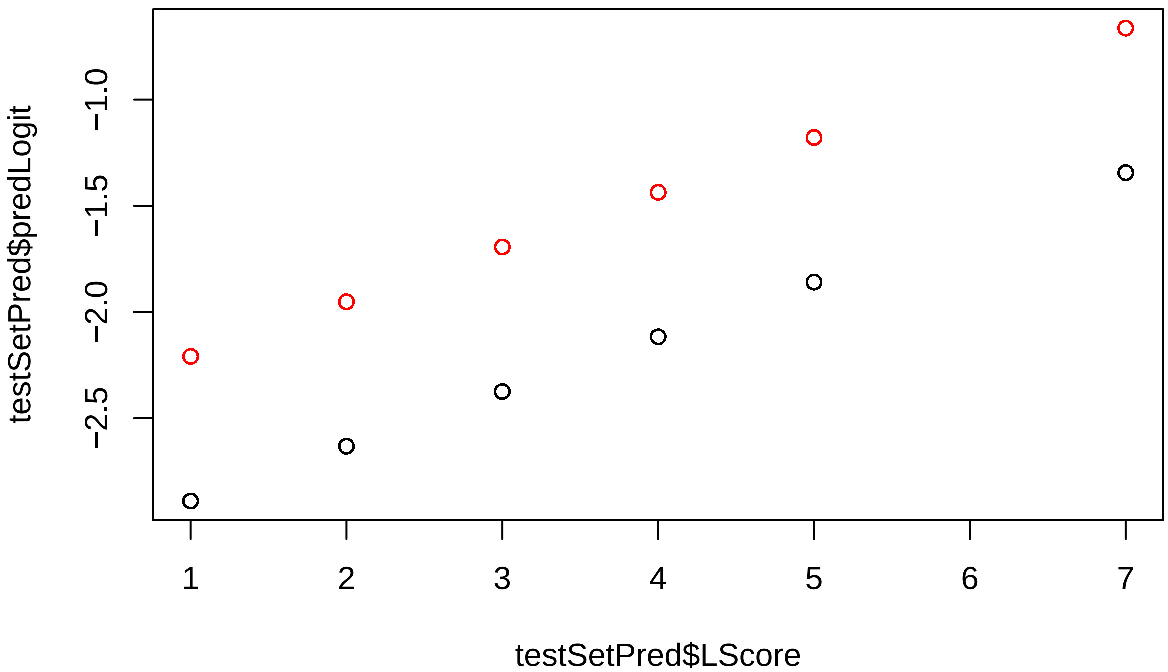
```
plot(testSet$LScore, testSet$predProb, col=testSet$AScore)
```



There are two other things to notice: our predicted probabilities are not that high (the maximum is 0.3) and that the relation between the predicted probabilities and `LScore` isn't linear.

So, let's visualize the logit instead. If you do not specify the `type` parameter, `predict()` returns the logit, or `log(OddsRatio)`. That means that in order to get the predicted Odds Ratio, you will need to exponentiate the output using `exp()`.

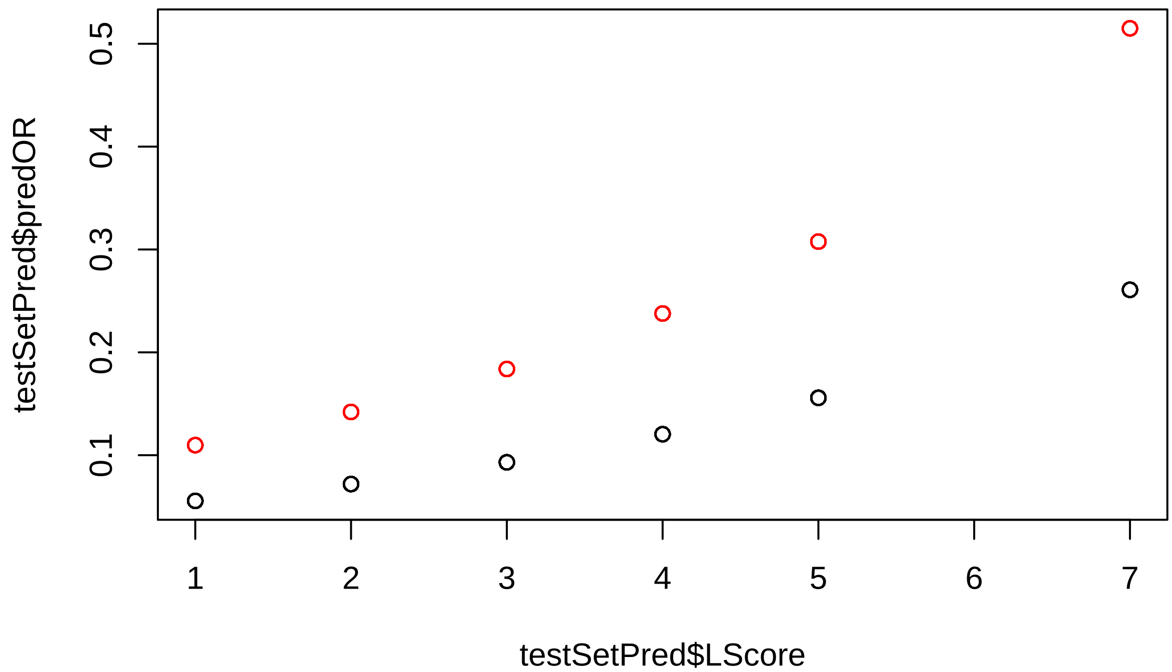
```
#predict the logit instead for our testSet
modelPredictedLogOddsRatio <- predict(laModel,newdata = testSet)

#add as another column in our table
testSetPred <- data.frame(testSet, predLogit = modelPredictedLogOddsRatio)

#plot the LScore versus logit (coloring by AScore)
plot(testSetPred$LScore, testSetPred$predLogit, col=testSetPred$AScore)
```




```r
#transform the logit to the predictedOdds ratio
modelPredictedOddsRatio <- exp(modelPredictedLogOddsRatio)
modelPredictedOddsRatio[1:10]
```

```
##      16075      14259      31319       4735      25510      33710
## 0.10979754 0.26079864 0.23779968 0.26079864 0.15579717 0.26079864
##      30382       4032      18861       4838
## 0.05559913 0.09307087 0.15579717 0.07193511
```

```r
#add as column in our table
testSetPred <- data.frame(testSetPred, predOR = modelPredictedOddsRatio)

#plot Odds ratio versus LScore
plot(testSetPred$LScore, testSetPred$predOR, col=testSetPred$AScore)
```




```r
#
exp(coef(laModel))
```

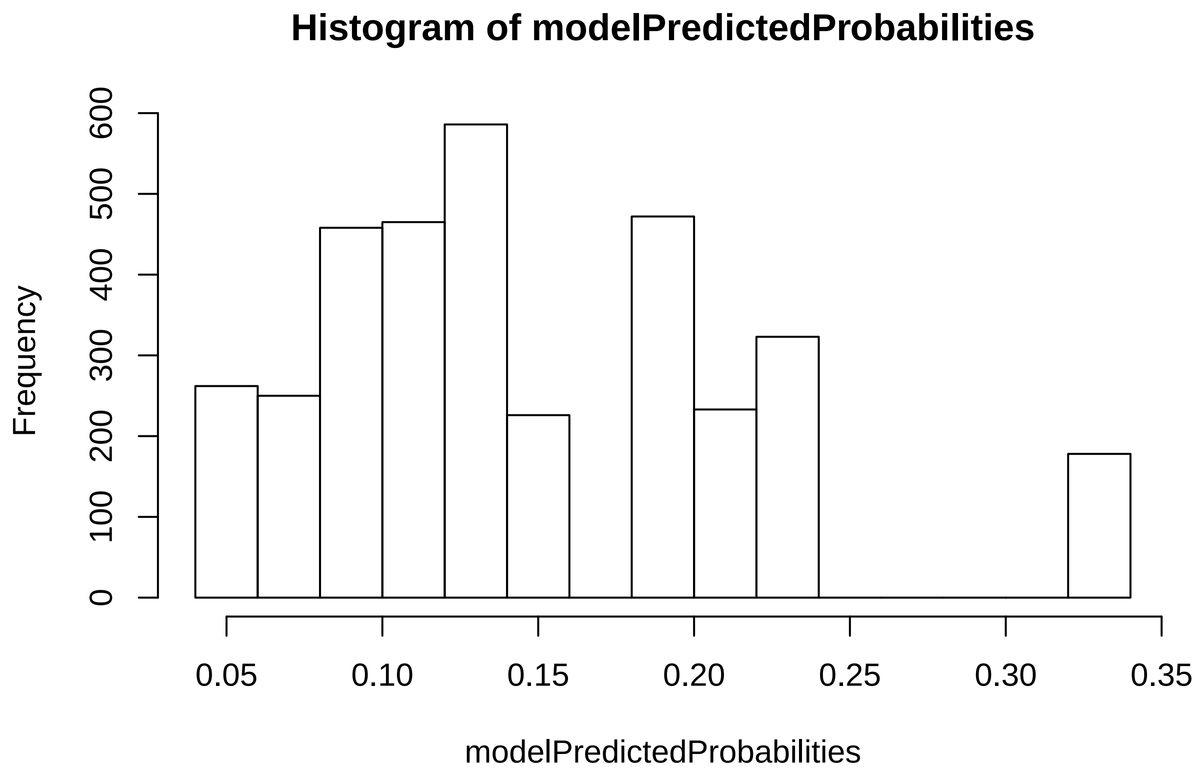
```
## (Intercept)      LScore     AScore3
##  0.04297295  1.29381709  1.97480670
```

Selecting a Probability Threshold

So you might notice that we have predicted probabilities, but we haven't actually predicted any values (whether a patient is a readmit risk or not). We can do this by choosing a *threshold probability*, that is, a cutoff value for our predicted probabilities that separates who we call as a readmit risk and who isn't.

How do we decide the threshold? One simple way to decide is to do a histogram of the *predicted probabilities*. We note that there is a gap between 0.20 and 0.25.

```r
hist(modelPredictedProbabilities)
```



What happens when we set our probability threshold at 0.225? We can use `ifelse()` to recode the probabilities using this threshold.

```
modelPredictions <- ifelse(modelPredictedProbabilities < 0.225, 0, 1)
modelPredictions[1:10]
```

```
## 16075 14259 31319  4735 25510 33710 30382  4032 18861  4838
##     0     0     0     0     0     0     0     0     0     0
```

We can do a crosstab between our predictions from our LA model and the truth (those we have identified as readmission risks) in our `testSet`.

```
truthPredict <- table(testSet$Readmit30, modelPredictions)
truthPredict
```

```
##    modelPredictions
##        0    1
##   0 2552  339
##   1  400  162
```

Looking at this 2x2 table, you might notice that we do kind of badly in terms of predicting readmission risk. Our accuracy can be calculated by calculating the total number of misclassifications (where predict does not equal truth). The misclassifications are where we predict 1, but the truth is 0 (false positives), and where we predict 0, but the truth is 1 (false negatives).

```r
totalCases <- sum(truthPredict)
misclassified <- truthPredict[1,2] + truthPredict[2,1]
misclassified
```

```
## [1] 739
```

```r
accuracy <- (totalCases - misclassified) / totalCases
accuracy
```

```
## [1] 0.7859832
```

For our `LScore + AScore` model, our prediction threshold of 0.225 has an accuracy of 78.5983203 percent.

ROC Curves

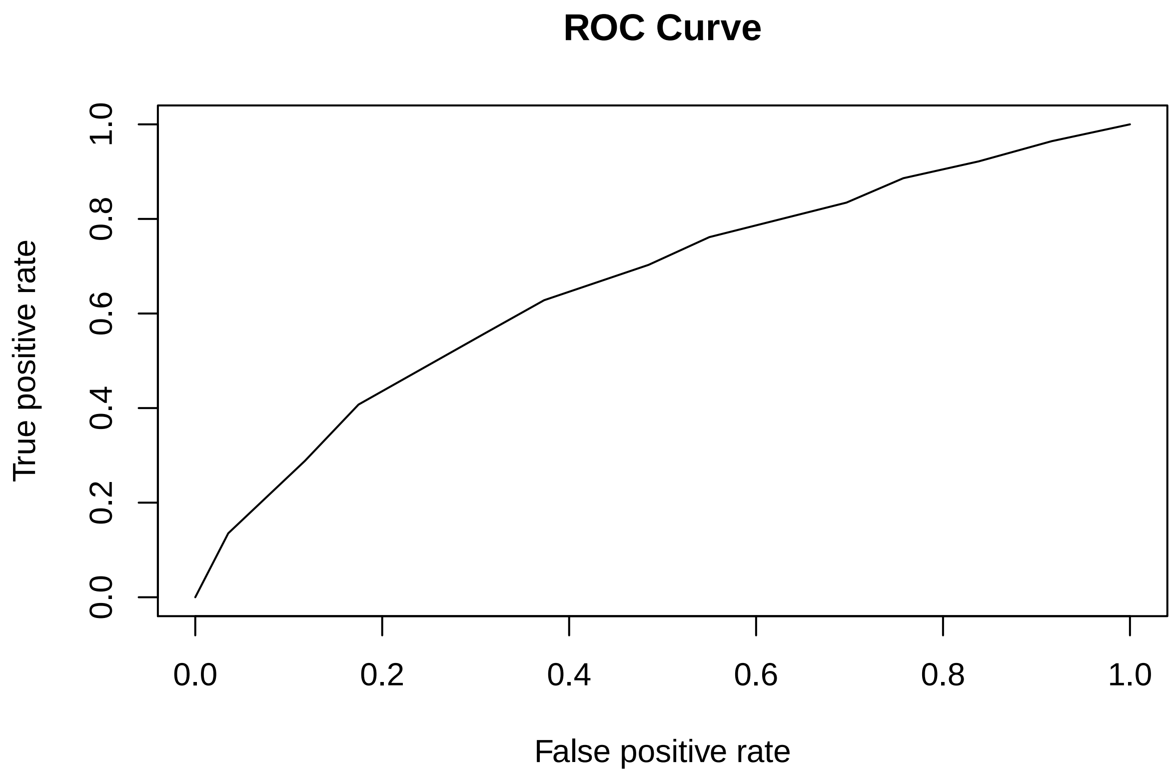
We can examine the impact of setting our probability threshold using the `ROCR` package (be sure to install it using `install.packages("ROCR")`).

An ROC curve (Receiver-Operator-Characteristic) is a way of assessing how our probability threshold affects our Sensitivity (our ability to detect true positives) and Specificity (our ability to detect true negatives). Any test has a sensitivity/specificity tradeoff. We can actually use an ROC curve to select a threshold based on whether we value Sensitivity or Specificity.

```r
library(ROCR)
```

```
## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```r
pr <- prediction(modelPredictedProbabilities, testSet$Readmit30)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf, main="ROC Curve")
```



The area under the ROC curve (AUC) is one way we can summarize our model performance. A model with perfect predictive ability has an AUC of 1. A random test (that is, a coin flip) has an AUC of 0.5.

```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.6627329
```

Our `LScore` + `AScore` model has an AUC of 0.6627329, which is not super great. Perhaps you can make it better?

More Info on Logistic Regression

Please note: this is a brief, non-comprehensive introduction to utilizing logistic regression for this class example. In addition to the R links at the end of this section, we highly recommend texts such as *Applied Logistic Regression* (Hosmer, Lemeshow and Sturidvant) for more detailed treatment of logistic regression, including topics such as model building strategies, diagnostics and assessment of fit as well as more complex designs which are beyond the scope of this assignment.

This page https://www.r-bloggers.com/evaluating-logistic-regression-models/ does a nice job explaining how to run logistic regressions and various ways to evaluate logistic regression models.

https://www.r-bloggers.com/how-to-perform-a-logistic-regression-in-r/ is also a good resource for understanding logistic regression models. This page http://www.ats.ucla.edu/stat/mult_pkg/faq/general/odds_ratio.htm is a good page for understanding odds ratios and predicted probabilities.