Prediction & Logistic Regression

Melinda K. Higgins, PhD.

February 13, 2017

Logistic Regression Example

The following code and data analysis example comes from Chapter 7, section 7.2 on logistic regression in the "Practical Data Science with R" book by Nina Zumel and John Mount (aka, textbook for this course).

Dataset

The dataset we'll use is the "CDC" data available from the book Github repository at https://github.com/WinVector/zmPDSwR/tree/master/CDC. We'll use the RData file "NatalRiskData.rData" which has 26313 observations and 15 variables. In this dataset, there is a variable "ORIGRANDGROUP" which has uniformly distributed random numbers ranging from 0 to 10. The `R` code chunk below divides the dataset into a "test" and a "training" dataset based on these random numbers: if <= 0.5, the observation is assigned to the "train" dataset and if > 0.5, the observation is assigned to the "test" dataset.

```
load("NatalRiskData.rData")
train <- sdata[sdata$ORIGRANDGROUP <= 5,]
test <- sdata[sdata$ORIGRANDGROUP > 5,]
```

The variables included that we will use in our logistic regression model are provided in the following table:

Variable	Type	Description
DEMOGRAPHICS		
"PWGT"	Numeric	Mother's prepregnancy weight
"UPREVIS"	Numeric (integer)	Number of prenatal medical visits
"CIG_REC"	Logical	TRUE if smoker; FALSE otherwise
"GESTREC3"	Categorical	Two categories: <37 weeks (premature) and >= 37 weeks
"DPLURAL"	Categorical	Birth plurality, three categories: single/twin/triplet+
COMPLICATIONS		
"ULD_MECO"	Logical	TRUE if moderate/heavy fecal straining of amniotic fluid
"ULD_PRECIP"	Logical	TRUE for unusually short labor (< 3 hrs)
"ULD_BREECH"	Logical	TRUE for breech (pelvis first) birth position
RISK FACTORS		
"URF_DIAB"	Logical	TRUE if mother is diabetic
"URF_CHYPER"	Logical	TRUE if mother has chronic hypertension
"URF_PHYPER"	Logical	TRUE if mother has pregnncy-related hypertension
"URF_ECLAM"	Logical	TRUE if mother experienced eclampsia; pregnancy related seizures
OUTCOME		
"atRisk"	Logical	TRUE if 5-minute Apgar score < 7; FALSE otherwise

Generalized Linear Model (`glm`) for Logistic Regression

As we saw in the previous lesson on linear regression, the R function used was `lm()` for linear model with models that:

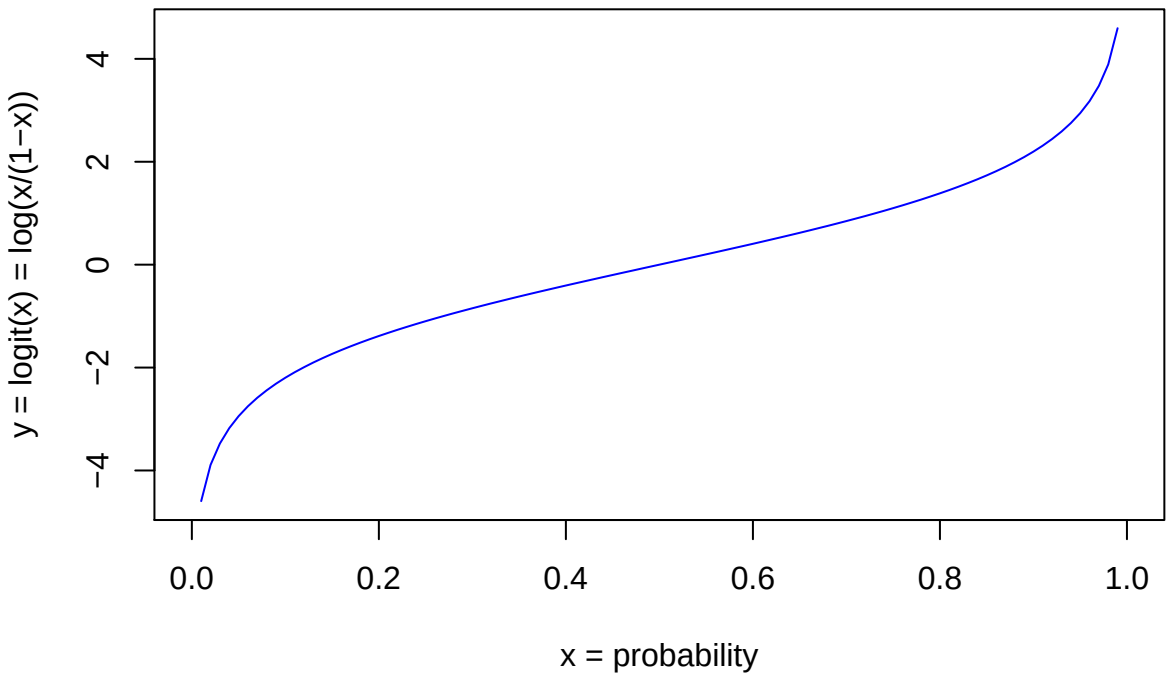
1. have "additive" effects (`Y = intercept + X1 + X2 + ...`)
2. are assumed to have "linear" association with the outcome of interest
3. have an outcome with a normal distribution.

Logistic regression has similar assumptions (additive effects and linear association with the outcome), except the outcome of interest is NOT a continuous number with a normal distribution. Instead, the outcome is a category - does the subject have the outcome of interest or not - in this case was the newborn classified as "at risk" or not? So, the logistic regression model is predicting the PROBABILITY that a given birth (observation) was "at risk".

Without going into all of the math, the "logit" is a function of the probability that the outcome of interest is `TRUE` (i.e. baby is "at risk"). The *logit* is `log(p/(1-p))` where `p` is the probability and `p/(1-p)` is the odds that the outcome is `TRUE`. So, `log(p/(1-p))` is the *log-odds* of the probability for the outcome you're interested in. So, when running a logistic regression instead of creating a model that predicts `y`, a logistic regression model predicts `logit(y)`. The `logit(y)` is the function of the outcome that is supposed to have a linear relationship to the predictors (`x`'s).

The plot below shows what the `logit(x)` function looks like, where `y = log(x/(1-x))`.

```
x <- seq(0,1,.01)
y <- log(x/(1-x))
plot(x,y,col='blue',type="l",xlab="x = probability",
     ylab="y = logit(x) = log(x/(1-x))")
```



The function to fit a logistic regression is `glm` which stands for fitting a "generalized linear model", which basically indicates that we are fitting a model for an outcome that has been "generalized" (through some kind of mathematical function) that "links" the outcome of interest to the predictors such that it is linearly related to the predictors. The outcome, however, can be in a variety of formats. In the case of logistic regression, the outcome has a binary distribution (2 possible outcomes - usually yes vs no) with a `logit` "link" function.

When using the `glm()` to fit a "General Linear Model", you need to know which "family" of models you will use to model your outcome. In R, the following "family objects" are available for `glm()` modeling (run `help(family)` and `help(glm)`):

```
binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")
```

NOTE: If we use `gaussian(link = "identity")` family with the `glm()` function, we're essentially fitting a "normal" linear regression model.

For our example here, we will use the `binomial(link = "logit")` family to perform a logistic regression. It is worth noting, that for "count-based" data (like "number of comorbidities", "number of children", "number of symptoms"), the `poisson(link = "log")` family can be used. `glm.nb()` (from the `MASS` package) which models the "negative binomial" family also works well for count-based outcomes.

Logistic Regression Model

Use the following code chunk to define the predictors (`x`'s) and outcome variable (`y`) for the model. The code below uses the `paste()` command to help manipulate putting the variable names together with `+` additive effects among the `x`'s and adding the `~` tilde for modeling `y` by the `x`'s.

```
# define the outcome variable
y <- "atRisk"

# define the input variable list
x <- names(sdata[,1:12])

# use the paste() command to put a + in between
# each of the x variables and put the ~ in between the
# outcome y and the x variables
fmla <- paste(y, paste(x, collapse="+"), sep="~")
```

The final model formula is atRisk~PWGT+UPREVIS+CIG_REC+GESTREC3+DPLURAL+ULD_MECO+ULD_PRECIP+

Develop the model using the training data

This is the formula that we'll use in the `glm()` model function call. The model is created using the `train` data.

```
model <- glm(fmla,
             data=train,
             family=binomial(link="logit"))
```

Model summary

```
summary(model)
```

```
##
## Call:
## glm(formula = fmla, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9732  -0.1818  -0.1511  -0.1358   3.2641
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -4.412189   0.289352 -15.249  < 2e-16 ***
## PWGT                      0.003762   0.001487   2.530 0.011417 *
## UPREVIS                  -0.063289   0.015252  -4.150 3.33e-05 ***
## CIG_RECTRUE               0.313169   0.187230   1.673 0.094398 .
## GESTREC3< 37 weeks        1.545183   0.140795  10.975  < 2e-16 ***
## DPLURALtriplet or higher  1.394193   0.498866   2.795 0.005194 **
## DPLURALtwin               0.312319   0.241088   1.295 0.195163
## ULD_MECOTRUE              0.818426   0.235798   3.471 0.000519 ***
## ULD_PRECIPTRUE            0.191720   0.357680   0.536 0.591951
## ULD_BREECHTRUE            0.749237   0.178129   4.206 2.60e-05 ***
## URF_DIABTRUE             -0.346467   0.287514  -1.205 0.228187
## URF_CHYPERTRUE            0.560025   0.389678   1.437 0.150676
## URF_PHYPERTRUE            0.161599   0.250003   0.646 0.518029
## URF_ECLAMTRUE             0.498064   0.776948   0.641 0.521489
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2698.7  on 14211  degrees of freedom
## Residual deviance: 2463.0  on 14198  degrees of freedom
## AIC: 2491
##
## Number of Fisher Scoring iterations: 7
```

Predictions

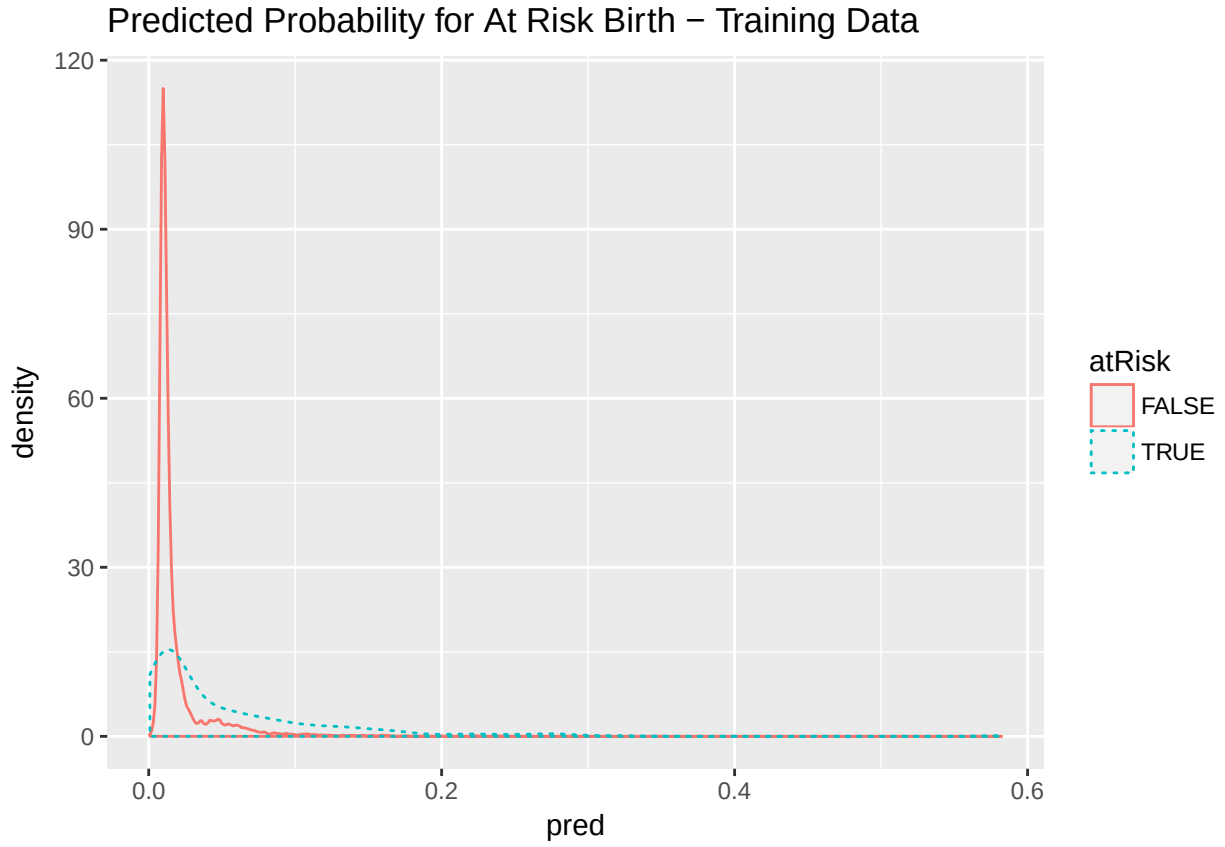
Now that we've created the model from the `train` data. Let's make predictions using the model and the `train` and the `test` datasets. To learn more about the options, run `help(predict.glm)`. The code chunk below adds `pred` as a new column in both the `train` and `test` datasets.

Note: the `predict()` function is a "generic function" that really calls `predict.glm()` since the input object is the output model from the `glm()` function. Run `help(predict)` to learn more.

```
train$pred <- predict(model,
                      newdata=train,
                      type="response")
test$pred <- predict(model,
                      newdata=test,
                      type="response")
```

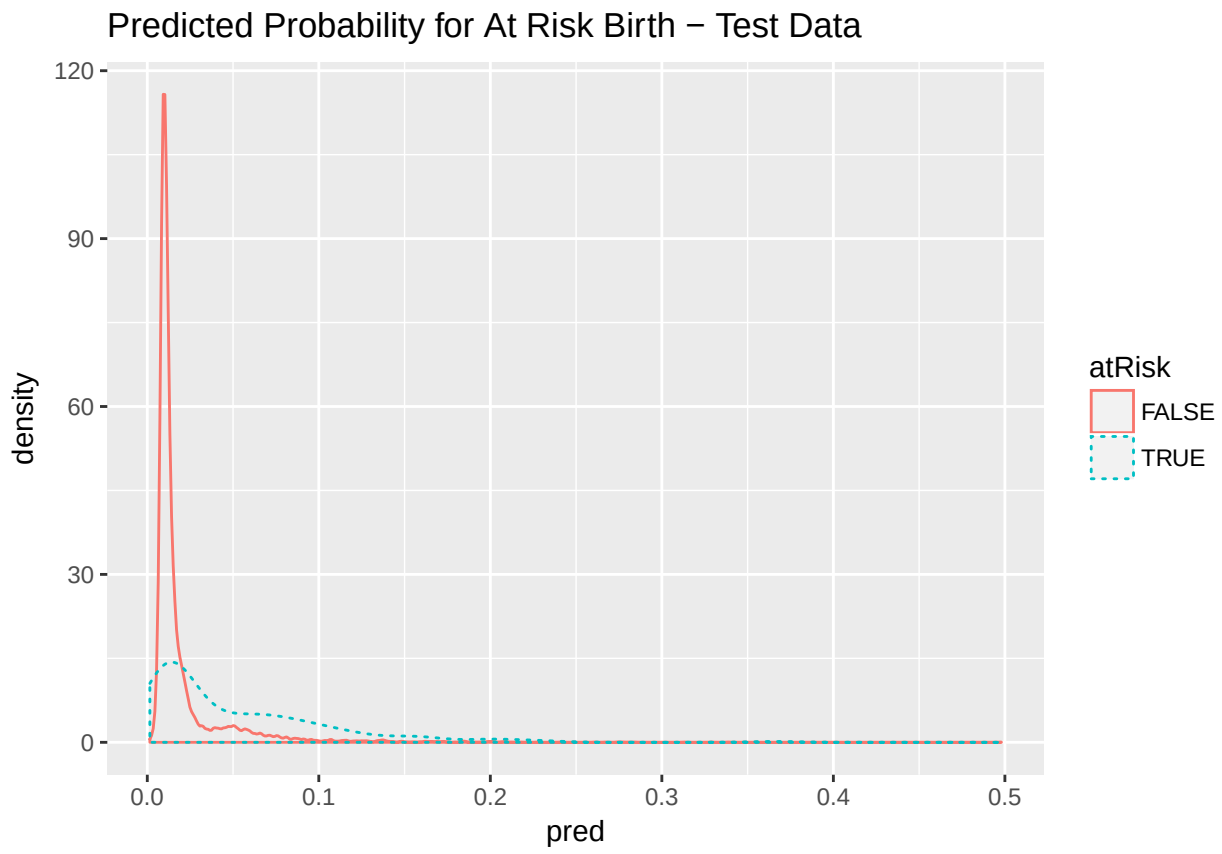
Plot of Predicted Probabilities - Training Data

```
library(ggplot2)
ggplot(train,
       aes(x=pred, color=atRisk, linetype=atRisk)) +
  geom_density() +
  ggtitle("Predicted Probability for At Risk Birth - Training Data")
```



Plot of Predicted Probabilities - Test Data

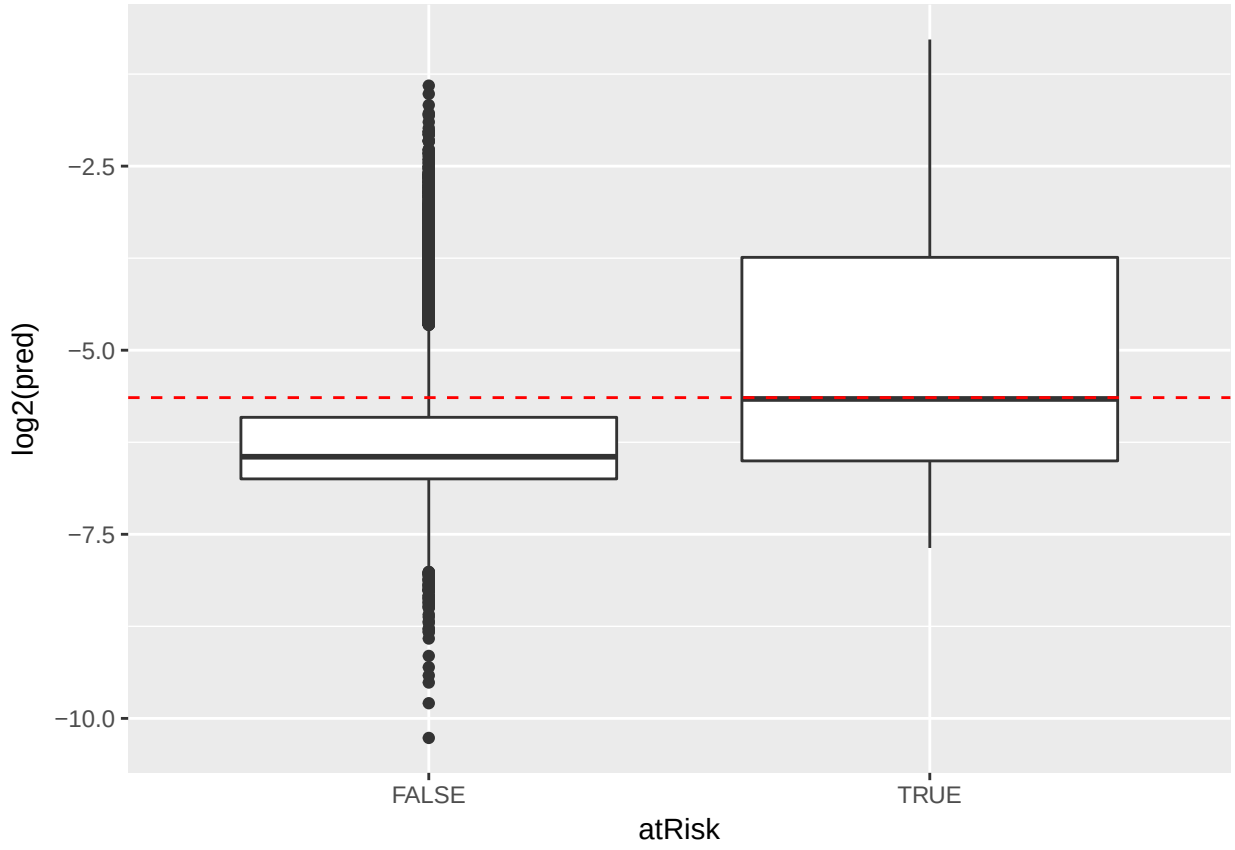
```
ggplot(test,
       aes(x=pred, color=atRisk, linetype=atRisk)) +
  geom_density() +
  ggtitle("Predicted Probability for At Risk Birth - Test Data")
```



Discrimination Plots - boxplot of probabilities by outcome

Another way to evaluation how well the 2 outcomes were separated is to look at the difference between the average predicted probability between the 2 "classes". We can visualize that using boxplots. Sine the probabilities were so skewed (long tails to right), I took the `log2()` and then made the plot.

```
# since the densities were so skewed, we'll
# plot the log2(pred) by class
# draw a line for the 0.02 threshold
ggplot(train, aes(atRisk, log2(pred))) +
  geom_boxplot() +
  geom_hline(yintercept=log2(0.02),
             col="red", lty=2)
```



Choosing a "Decision" Threshold

After looking at the plots above, what probability do you think will "best" separate the 2 outcomes ("at risk" versus "not at risk")?

For this dataset, the density of prediction probabilities for both classes are clustered on the left with a lot of overlap (i.e. these classes are hard to separate). This makes sense since "at risk" births are rare - looking back at the original full `sdata` dataset, the % of "at risk" births were really low `sum(sdata$atRisk)*100/length(sdata$atRisk)` = 1.8317942 %.

In order to use the model as a classifier, we need to choose a threshold probability where `p < threshold` is "false" and `p > threshold` is "true". When choosing a threshold you are trying to balance the *"precision"* of the classifier (predicted positives that are true positives) with *"recall"* (how many of the true positives are found). One way to look at the tradeoff between these is to look at the *"enrichment rate"* which is the ratio of the classifier precision to the average rate of positives. (see pp 161-164 in "Practical Data Science with R"). The code chunk below produces a plot of the *"enrichment rate"* (TOP) across a range of threshold and the *"recall"* (BOTTOM) for thr training dataset.

```
library(ROCR)
```

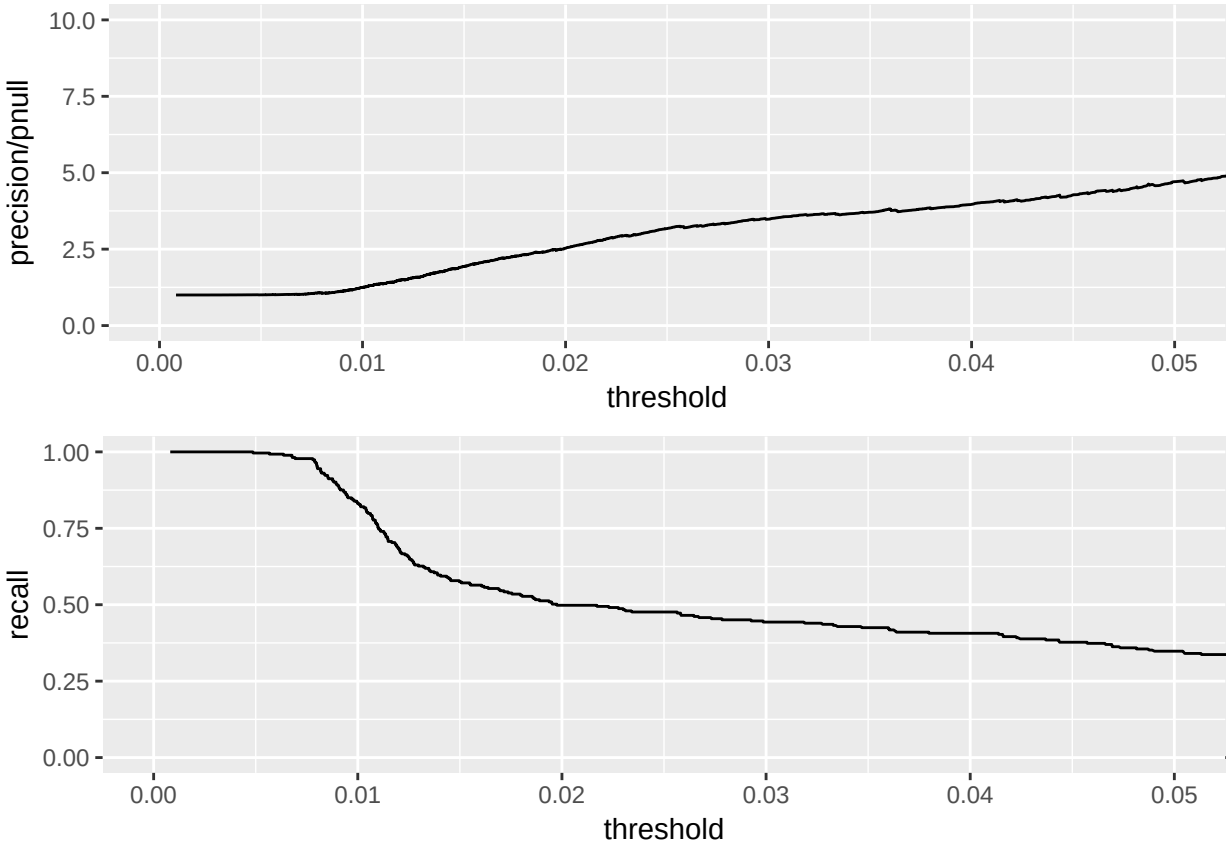
```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

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

```
library(grid)

predObj <- prediction(train$pred, train$atRisk)

precObj <- performance(predObj, measure="prec")
recObj <- performance(predObj, measure="rec")

precision <- (precObj@y.values)[[1]]
prec.x <- (precObj@x.values)[[1]]
recall <- (recObj@y.values)[[1]]

rocFrame <- data.frame(threshold=prec.x,
                       precision=precision,
                       recall=recall)

nplot <- function(plist){
  n <- length(plist)
  grid.newpage()
  pushViewport(viewport(layout=grid.layout(n,1)))
  vplayout=function(x,y){
    viewport(layout.pos.row=x, layout.pos.col=y)
  }
  for (i in 1:n){
    print(plist[[i]],
          vp=vplayout(i,1))
  }
}

pnull <- mean(as.numeric(train$atRisk))

p1 <- ggplot(rocFrame, aes(x=threshold)) +
  geom_line(aes(y=precision/pnull)) +
  coord_cartesian(xlim=c(0,0.05), ylim=c(0,10))

p2 <- ggplot(rocFrame, aes(x=threshold)) +
  geom_line(aes(y=recall)) +
  coord_cartesian(xlim=c(0,0.05))

nplot(list(p1,p2))
```

```
## Warning: Removed 1 rows containing missing values (geom_path).
```



Pick up a threshold - Review Confusion Matrix

Given the plots above, we want to find a value where the TOP plot ("precision/pnull" enrichment rate is maximized) while ALSO maximizing the BOTTOM plot ("recall"). So, we want to strike a balance between the two.

Threshold = 0.02

The code below creates the "confusion matrix" which compares the true positives and negatives with those predicted by the model. For this, we use the `test` dataset.

```
# test a classifier with a threshold > 0.02
ctab.test <- table(pred=test$pred>0.02, atRisk=test$atRisk)
ctab.test
```

```
##        atRisk
## pred    FALSE TRUE
##   FALSE  9487   93
##   TRUE   2405  116
```

```
# compute precision = true positives / predicted true
precision <- ctab.test[2,2]/sum(ctab.test[2,])

# compute recall = true positives / actual true
recall <- ctab.test[2,2]/sum(ctab.test[,2])
```

This classifier is low-precision with a precision = 4.6 % and recall of 55.5 %.

Threshold = 0.01

Let's run this again for threshold = 0.01. Setting the threshold lower increased the number of true positives predicted correctly, but we now have a much higher error rate for predicting the falses.

```
# test a classifier with a threshold > 0.01
ctab.test <- table(pred=test$pred>0.01, atRisk=test$atRisk)
ctab.test
```

```
##        atRisk
## pred    FALSE TRUE
##   FALSE  3999   35
##   TRUE   7893  174
```

```
# compute precision = true positives / predicted true
precision <- ctab.test[2,2]/sum(ctab.test[2,])

# compute recall = true positives / actual true
recall <- ctab.test[2,2]/sum(ctab.test[,2])
```

This classifier is low-precision with a precision = 2.16 % and recall of 83.25 %. So, the *recall* is higher, but *precision* is lower.

Threshold = 0.03

Let's run this again for threshold = 0.03. Now, we've done a better job predicting the falses but did much worse predicting the true "at risk" births.

```
# test a classifier with a threshold > 0.03
ctab.test <- table(pred=test$pred>0.03, atRisk=test$atRisk)
ctab.test
```

```
##        atRisk
## pred    FALSE  TRUE
##   FALSE 10341   114
##   TRUE   1551    95
```

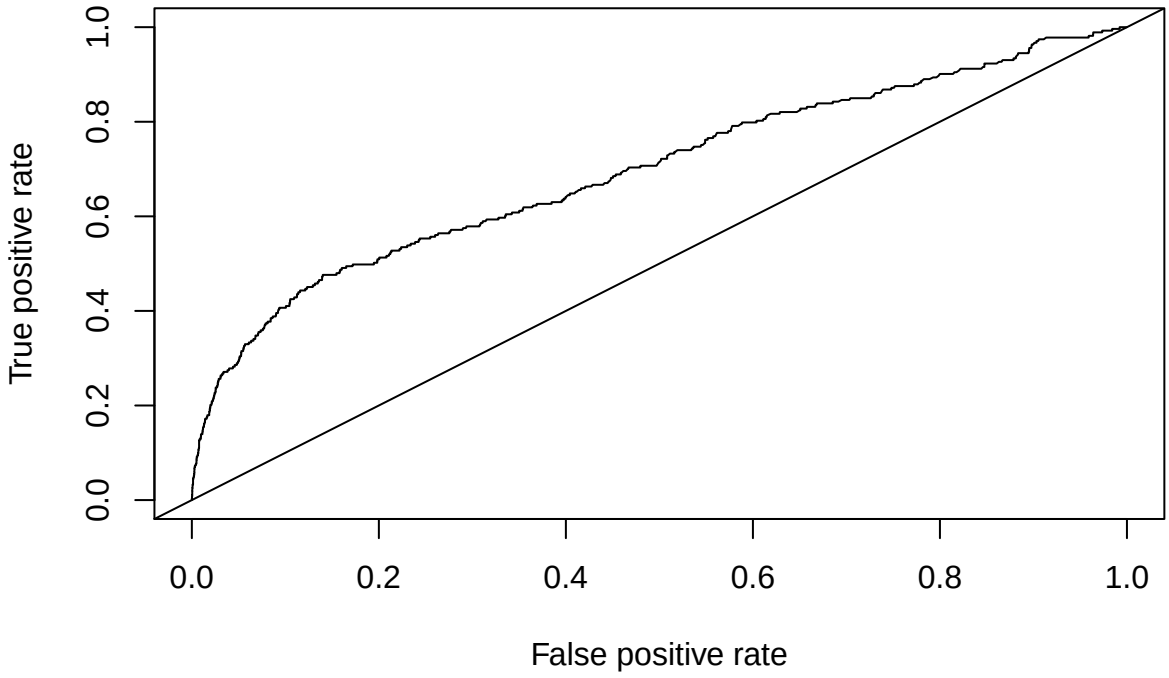
```
# compute precision = true positives / predicted true
precision <- ctab.test[2,2]/sum(ctab.test[2,])

# compute recall = true positives / actual true
recall <- ctab.test[2,2]/sum(ctab.test[,2])
```

This classifier is low-precision with a precision = 5.77 % and recall of 45.45 %. Now *precision* is higher, but *recall* is now lower.

The "Concordance" or "C"-statistic (e.g. AUC for the ROC)

The "concordance" or "c"-statistic is the "area under the curve" for the receiver operating curve (ROC). This curve is a plot of the tradeoffs between the false positive rate and true positive rate across a range of threshold probabilities. For this we'll use the `ROCR` package.

```
library(ROCR)
ROCRpred <- prediction(train$pred, train$atRisk)
ROCRperf <- performance(ROCRpred, 'tpr','fpr')
plot(ROCRperf)
abline(0,1)
```



Another option is to plot the Sensitivity versus the Specificity, which can be done using the `pROC` package. This package also computes the AUC or "c"-statistic. *NOTE: A "perfect" AUC = 1 whereas AUC=0.5 indicates the model predictions are 50/50 - not better than flipping a coin. So, we want the AUC > 0.5 and as close to 1 as possible.*

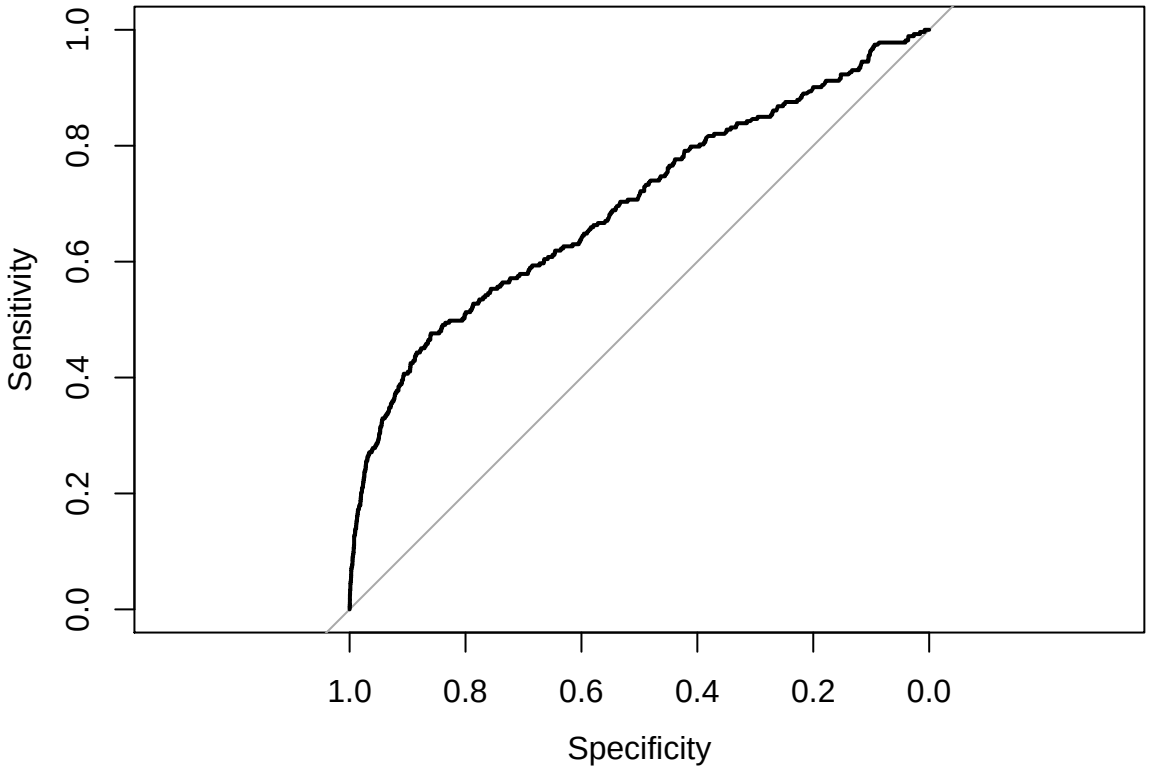
```
library(pROC)
```

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

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

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

```
roccurve <- roc(train$atRisk ~ train$pred)
plot(roccurve)
```




```
##
## Call:
## roc.formula(formula = train$atRisk ~ train$pred)
##
## Data: train$pred in 13939 controls (train$atRisk FALSE) < 273 cases (train$atRisk TRUE).
## Area under the curve: 0.6956
```

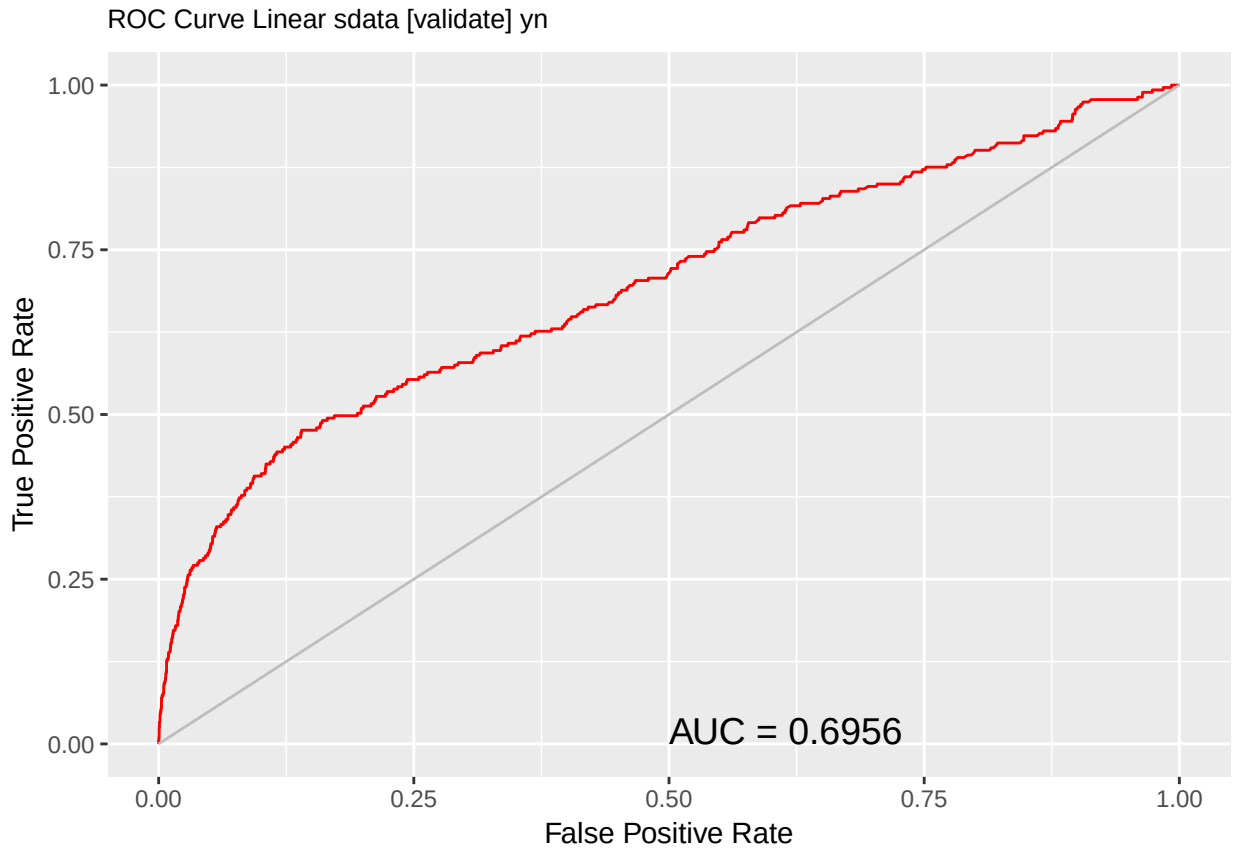
```
# pull out just the AUC statistic
auc(roccurve)
```

```
## Area under the curve: 0.6956
```

Finally, the code below is a 3rd option for making these plots based on the code generated by the `rattle` package.

```
#=======================================================
# Rattle timestamp: 2017-02-14 00:00:34 x86_64-w64-mingw32

# Evaluate model performance.

# ROC Curve: requires the ROCR package.

#library(ROCR)

# ROC Curve: requires the ggplot2 package.

#library(ggplot2, quietly=TRUE)
```

```
# Generate an ROC Curve for the glm model on sdata [validate].

#pr <- predict(model, type="response", newdata=train)
pr <- train$pred
train$yn <- as.numeric(train$atRisk)

# Remove observations with missing target.

no.miss   <- na.omit(train$yn)
miss.list <- attr(no.miss, "na.action")
attributes(no.miss) <- NULL

if (length(miss.list))
{
  pred <- prediction(pr[-miss.list], no.miss)
} else
{
  pred <- prediction(pr, no.miss)
}

pe <- performance(pred, "tpr", "fpr")
au <- performance(pred, "auc")@y.values[[1]]
pd <- data.frame(fpr=unlist(pe@x.values), tpr=unlist(pe@y.values))
p <- ggplot(pd, aes(x=fpr, y=tpr))
p <- p + geom_line(colour="red")
p <- p + xlab("False Positive Rate") + ylab("True Positive Rate")
p <- p + ggtitle("ROC Curve Linear sdata [validate] yn")
p <- p + theme(plot.title=element_text(size=10))
p <- p + geom_line(data=data.frame(), aes(x=c(0,1), y=c(0,1)), colour="grey")
p <- p + annotate("text", x=0.50, y=0.00, hjust=0, vjust=0, size=5,
                  label=paste("AUC =", round(au, 4)))
print(p)
```




```
# Calculate the area under the curve for the plot.


# Remove observations with missing target.

no.miss   <- na.omit(train$yn)
miss.list <- attr(no.miss, "na.action")
attributes(no.miss) <- NULL

if (length(miss.list))
{
  pred <- prediction(pr[-miss.list], no.miss)
} else
{
  pred <- prediction(pr, no.miss)
}
performance(pred, "auc")@y.values
```

```
## [[1]]
## [1] 0.6956179
```