Lab 9: ROC and Cross-validation

TA: Isabel Laterzo

March 23, 2021

Lab adapted from text and code by Simon Hoellerbauer and Chelsea Estancona.

Today we will learn about various ways to assess model fit. We will examine AIC, BIC, ROC, AUC (so many acronyms!), and the likelihood ratio test.

First, we'll use data from the site below to construct a model with a binary dependent variable: whether or not a student is admitted to graduate school.

We will also create a second model to compare it to. In this second model, we introduce an additional predictor that is pretty uninformative - we'll call it `gre_fake`

```
mydata <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
m1 <- glm(admit ~ gre + gpa + rank, data = mydata, family = "binomial")
summary(m1)
```

```
##
## Call:
## glm(formula = admit ~ gre + gpa + rank, family = "binomial",
##     data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5802  -0.8848  -0.6382   1.1575   2.1732
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.449548   1.132846  -3.045  0.00233 **
## gre          0.002294   0.001092   2.101  0.03564 *
## gpa          0.777014   0.327484   2.373  0.01766 *
## rank        -0.560031   0.127137  -4.405 1.06e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 459.44  on 396  degrees of freedom
## AIC: 467.44
##
## Number of Fisher Scoring iterations: 4
```

```
mydata$gre_fake <- seq(1:400)
m2 <- glm(admit~ gre_fake + gpa, data = mydata, family = "binomial")
summary(m2)
```

```
##
## Call:
## glm(formula = admit ~ gre_fake + gpa, family = "binomial", data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1581  -0.8934  -0.7571   1.3115   1.9406
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.4669515  1.0531982  -4.241 2.22e-05 ***
## gre_fake     0.0005560  0.0009463   0.588 0.556799
## gpa          1.0502709  0.2988955   3.514 0.000442 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 486.62  on 397  degrees of freedom
## AIC: 492.62
##
## Number of Fisher Scoring iterations: 4
```

AIC and BIC

First, let's explore these models with some simple checks. One approach is to use the Akaike Information Criterion and Bayesian Information Criterion, which allow us to compare any two models with the same dependent variable.

```
## smaller values indicate a better fitting model
AIC(m1)
```

```
## [1] 467.4418
```

```
AIC(m2)
```

```
## [1] 492.622
```

```
BIC(m1)
```

```
## [1] 483.4076
```

```
BIC(m2)
```

```
## [1] 504.5964
```

AIC is an estimate of the constant plus the relative distance between the unknown TRUE likelihood function of the data and the FITTED likelihood function of the model. A lower AIC indicates a model is closer to the truth. BIC is an estimate of a function of the posterior probability of a model being true. A lower BIC also means a model is closer to the true model. Unsurprisingly, our model without the fake predictor performs better among these tests!

The Likelihood Ratio test (or Wilks test)

Another approach to compare models is the likelihood ratio test. Unlike AIC and BIC, likelihood ratio tests can only compare nested models, so we can't compare the same model estimated with different link functions

- if you're ever comparing probit and logit models for example, you cannot use this test.

The LRT compares the more complex model (with more variables) to the "nested" model, or the one that is simpler. It ultimately tells us if it is beneficial to add parameters to our model or not.

Our null hypothesis is we should use the simpler (or "nested") model. Our alternative hypothesis is we should use the complex model. A common significance level to reject the null is 0.05, indicating we should uses the more complex model.

```
#generate a "nested" or simpler model
m1_b <- glm(admit ~ gre + gpa, data = mydata, family = "binomial")
summary(m1_b)
```

```
##
## Call:
## glm(formula = admit ~ gre + gpa, family = "binomial", data = mydata)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2730  -0.8988  -0.7206   1.3013   2.0620
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.949378   1.075093  -4.604 4.15e-06 ***
## gre          0.002691   0.001057   2.544   0.0109 *
## gpa          0.754687   0.319586   2.361   0.0182 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 480.34  on 397  degrees of freedom
## AIC: 486.34
##
## Number of Fisher Scoring iterations: 4
```

```
## likelihood ratio test by hand aka what is the canned function doing?
## answer: 2 * (log-likelihood(m2) - log-likelihood(m1))
Lik_nest <- logLik(m1_b) #get the log likelihood
Lik_nest
```

```
## 'log Lik.' -240.172 (df=3)
```

```
Lik_complex <- logLik(m1)
Lik_complex
```

```
## 'log Lik.' -229.7209 (df=4)
```

```
LR <- -2 * (as.numeric(Lik_nest) - as.numeric(Lik_complex))
#formula for the likelihood ratio
LR
```

```
## [1] 20.90222
```

```
## this likelihood ratio follows a chi-squared distribution. How is the
## chi-squared distribution parameterized?
p_val <- pchisq(LR, df = 1, lower.tail = FALSE) # where does the value for
                                                #degrees of freedom come from?
```

```
p_val
```

```
## [1] 4.83335e-06
```

```
#You can ignore the stuff around the p-value. The issue is that logLik returns
#an object of class logLik, and this gets inherited by all operations on this
#object, because we are working in an object-oriented programming language.
class(p_val)
```

```
## [1] "numeric"
```

```
## using lrtest from the lmtest package
library(lmtest)
lrtest(m1, m1_b)
```

```
## Likelihood ratio test
##
## Model 1: admit ~ gre + gpa + rank
## Model 2: admit ~ gre + gpa
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1   4 -229.72
## 2   3 -240.17 -1 20.902  4.833e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#same as what we got above
```

What does this p value suggest?

ROC and AUC

ROC curves (or Receiver-Operator Characteristic Curves) provide us with a graphical representation of the predictive power and accuracy of our models. Specifically, they are useful for binary classifier models (like being admitted or not to a school!). The ROC compares the predictions of our model to the outcomes in our data.

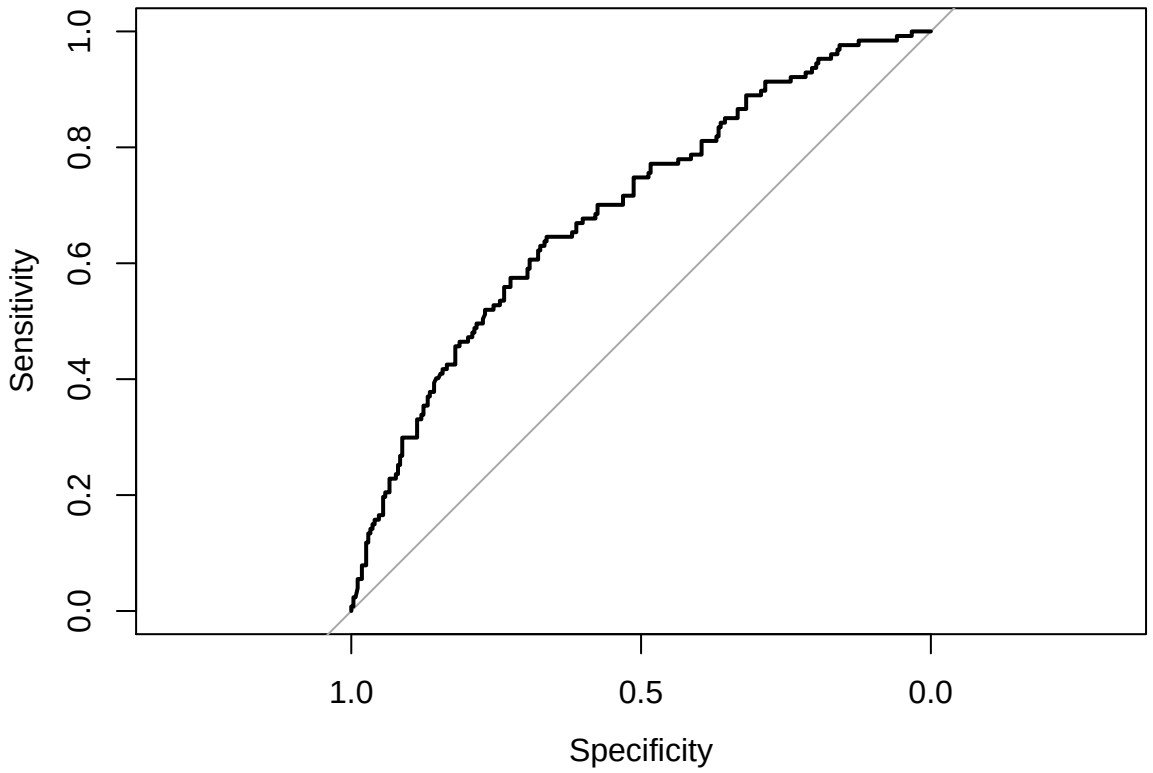
The curve is a plot of true positives against false positives, compared to a null line of 'no model'. We can think of the ROC curve as a measure of sensitivity (probability of detection) vs. specificity (probability of false alarm). It is essentially the plot of the power as a function of Type 1 error (the rejection of a true null hypothesis, or a false positive). ROC curves that are closer to the top left corner indicate better performance.

If you want a useful, but relatively monotone, video that breaks this down, I suggest this link:https://www.dataschool.io/roc-curves-and-auc-explained/

Now, with this simple model, we want to generate the predicted probability of getting into graduate school (we'll need these to assess the quality of our model predictions).

```
#install.packages("pROC")
library(pROC)
#install.packages("plotROC")
library(plotROC)

roc_prob <- predict(m1, type = ("response")) #generate predicted probabilities
                                             #for our observations
mydata$roc_prob <- roc_prob #bind them with the original data
```

```
plot(pROC::roc(admit ~ roc_prob, data = mydata)) #if all we want is a base plot...
```

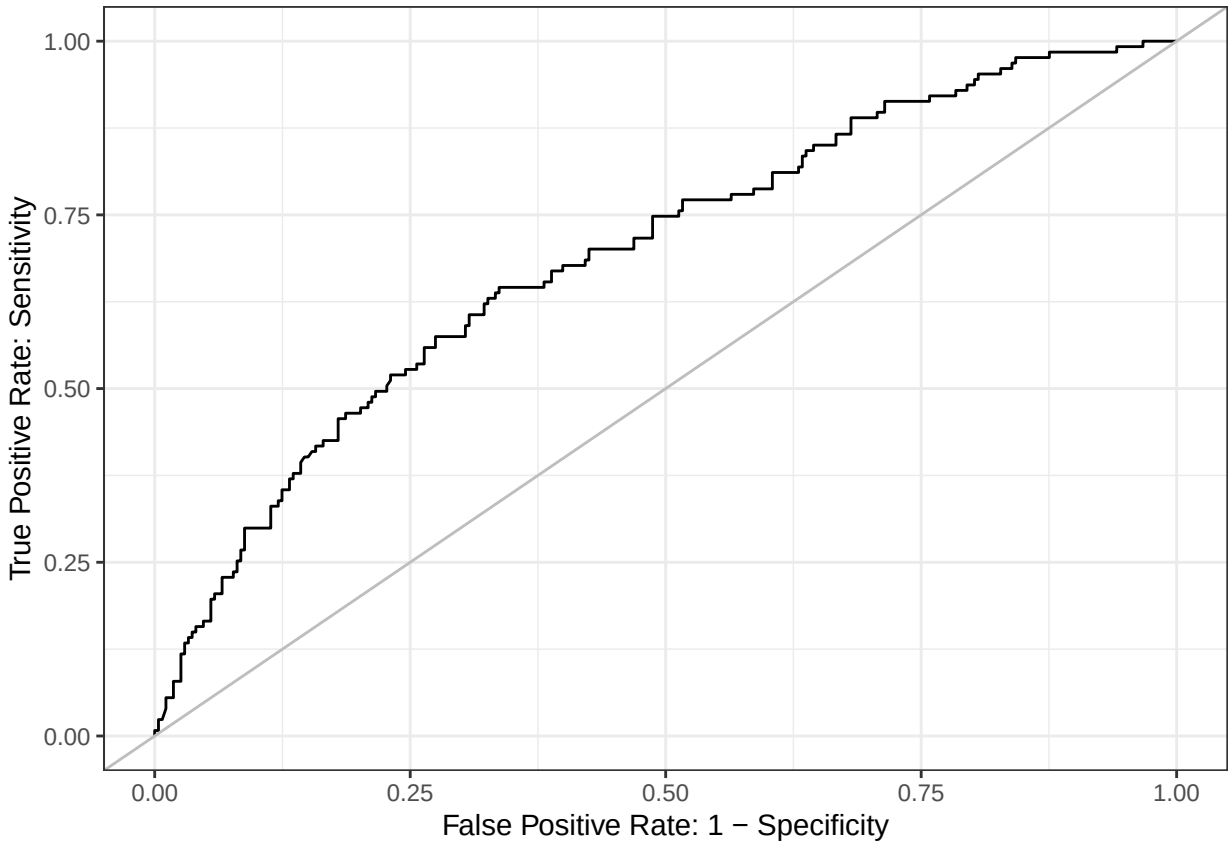



```
#note that there is an roc() function in the AUC package, so we have to specify here
```

Or with ggplot...

```
#install.packages("ROCR")
library(ROCR) #specific package for ROC plots in ggplot
library(ggplot2)

predicted <- predict(m1) #generate a prediction for each observation
prob <- prediction(predicted, mydata$admit) #makes our predicted probabilities
# and the dv the correct object type for this package
tprfpr <- performance(prob, "tpr", "fpr") #performance generates MANY measures of
#predictor evaluations, we want a true positive rate and a false positive rate

#below extracts the y values and x values we need
#we have to use slot and unlist because of the class of tprfpr
#the following will not work:
#tprfpr$y.values

tpr <- unlist(slot(tprfpr, "y.values"))
fpr <- unlist(slot(tprfpr, "x.values"))

#provide a data frame for ggplot
roc <- data.frame(tpr, fpr)

ROC_plot <- ggplot(roc) + geom_line(aes(x = fpr, y = tpr))+
  geom_abline(intercept = 0, slope = 1, colour = "gray")+
```

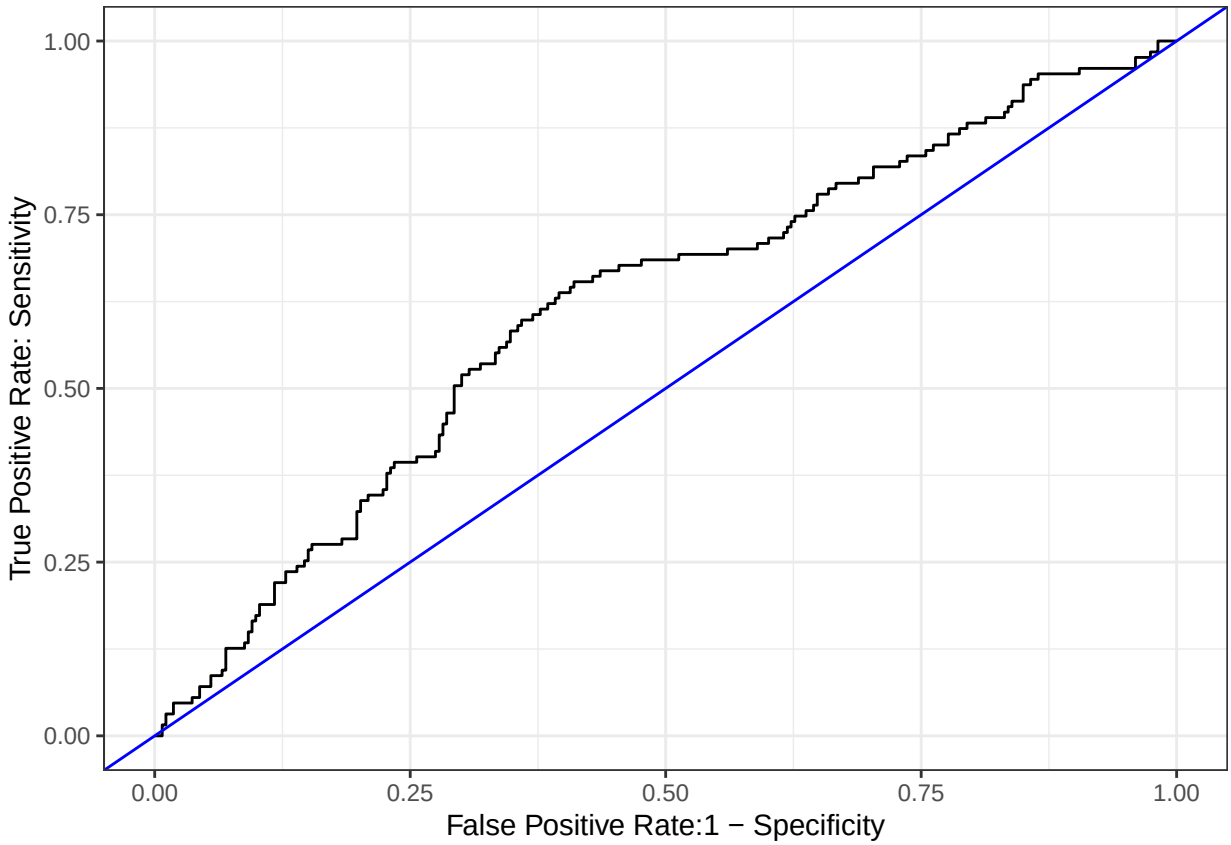
```
  ylab("True Positive Rate: Sensitivity") +
  xlab("False Positive Rate: 1 - Specificity") + theme_bw()

ROC_plot
```




```
#wow, a plot!
```

Now let's calculate the ROC of our model with the other (fake) predictor:

```
predicted_2 <- predict(m2)
prob_2 <- prediction(predicted_2, mydata$admit)
tprfpr_2 <- performance(prob_2, "tpr", "fpr")
tpr_2 <- unlist(slot(tprfpr_2, "y.values"))
fpr_2 <- unlist(slot(tprfpr_2, "x.values"))
roc_2 <- data.frame(tpr_2, fpr_2)

ROC_plot_2<- ggplot(roc_2) + geom_line(aes(x = fpr_2, y = tpr_2))+
  geom_abline(intercept = 0, slope = 1, colour = "blue") +  #to add the 'area'
  ylab("True Positive Rate: Sensitivity") +
  xlab("False Positive Rate:1 - Specificity") + theme_bw()

ROC_plot_2
```



How would you compare these two ROC curves? Which one is closer to the diagonal line (or suggestion of basically a random classifier)?

What are ways we could improve our ROC curve?

The ROC curve allows us to calculate the creatively named Area Under the Curve (AUC). Some people like to use the AUC as a measure of model fit because it is easy to understand and digest - it is just one number, after all, where higher numbers indicate higher predictive accuracy.

```
#using pROC
## for m1
#remember, from above: roc_prob <- predict(m1, type = ("response"))
pROC::auc(pROC::roc(admit ~ roc_prob, data = mydata))
```

```
## Area under the curve: 0.6921
```

```
## repeat for m2
roc_prob_2 <- predict(m2, type = ("response")) #generate predicted probabilities
                                               #for our observations
mydata$roc_prob_2 <- roc_prob_2 #bind them with the original data
pROC::auc(pROC::roc(admit ~ roc_prob_2, data = mydata))
```

```
## Area under the curve: 0.6162
```

```
#just confirms what the plot from above shows - adding in the new predictor hurts us

#which one of the above AUC values is higher?

#anoter way of doing this, fun!
```

```
#using AUC (pROC is better maintained, but plot looks nicer with AUC)
#install.packages("AUC")
library(AUC)
auc_ob <- AUC::roc(predict(m1, type = "response"),
         as.factor(model.frame(m1)[,1]))
auc(auc_ob)
```

```
## [1] 0.6921202
```

```
#REMEMBER TO BE CAREFUL ABOUT USING pROC AND AUC TOGETHER
```

Cross Validation

Another opportunity for us to test the predictive power of our model is to use cross-validation. This lets us divide our dataset into a 'training set' on which we run our model, and a 'validation set' on which we test the model's predictive ability.

There are two standard methods of cross-validation: k-fold (often 10-fold) techniques and leave-one-out techniques. The first uses a certain number of 'folds' or subsets of the data, where one fold is left out and the rest becomes the training set. The second uses n folds (where n is the number of observations in the data) and thus each observation is used individually as the validation set.

After fitting a model on to the training data, its performance is measured against each validation set and then averaged, gaining a better assessment of how the model will perform when asked to predict for new observations. We make choices about the number of partitions to construct based on a bias-variance trade off, where more partitions means less bias, but can also lead to more variance in our estimates of prediction error. Depending on our sample size, we may also need to make different decisions about k-fold vs. LOOCV.

Cross-validation serves as a way of getting at out-of-sample predictive accuracy, without having to designate a test-set (and thus allows us to use more of our data).

We can do both with cv.glm (there's also a cv.lm) from the 'boot' package:

First, we'll do K-fold cross validation:

```
#install.packages("boot")
library(boot)
set.seed(23450)

cost <- function(r, pi){
   mean(abs(r-pi) > 0.5)
}

#Here, r will be our vector of outcomes, and pi is our predicted outcomes.
#We want to penalize our model for predicting incorrectly.
#This cost function can be changed - for LM, it's MSE, for example.

model_test <- cv.glm(data = mydata[,(1:4)], glmfit = m1, cost = cost, K = 10)
#data, model, cost function, number of folds

model_test$delta #delta gives us an estimate of prediction error
```

```
## [1] 0.2950 0.2965
```

```
#1st is a raw estimate, 2nd is adjusted for fold choices

mydata2 <- mydata %>% select(admit, gre_fake, gpa) #only variables included in the model
```

```
model_test_2 <- cv.glm(data = mydata2, glmfit = m2, cost = cost, K = 10)
model_test_2$delta
```

```
## [1] 0.31750 0.31675
```

```
#ultimately, the first model produces lower prediction error.
```

Now, leave one out cross validation:

```
#LOOCV
model_test_LOOCV <- cv.glm(mydata[1:4], m1, cost = cost)
model_test_LOOCV$delta
```

```
## [1] 0.2975000 0.2980375
```

```
model_test_LOOCV2 <- cv.glm(mydata2, m2, cost = cost)
model_test_LOOCV2$delta
```

```
## [1] 0.3175000 0.3174938
```

```
#again, lower prediction error with the first model!
```

On your own

Use the voter turnout data from the Zelig package (see last lab and below) to generate two models. These models can be nested, or one can contain a randomly generated variable as per our m2 above. For each, plot the ROC curve with ggplot. Next, perform K-fold cross validation on each model. Compare the predictive errors. Perform leave-one-out cross validation. Why/when might you choose to use K-fold or LOOCV?

```
library(Zelig)
data(turnout)
```