Specifically, I talked a couple of times about binomial regression ([here](http://www.deeplytrivial.com/2018/04/g-is-for-glm-function.html) and [here](http://www.deeplytrivial.com/2018/04/z-is-for-z-scores-and-standardizing.html)), which is used to predict (read: recreate with a set of variables significantly related to) a binary outcome. The data example I used involved my dissertation data and the binary outcome was verdict: guilty or not guilty. A regression model returns the linear correction applied to the predictor variables to reproduce the outcome, and will highlight whether a predictor was significantly related to the outcome or not. But a big question you may be asking of your binomial model is: how well does it predict the outcome? Specifically, how can you examine whether your regression model is correctly classifying cases?

We’ll start by loading/setting up the data and rerunning the binomial regression with interactions.

dissertation<-read.delim("dissertation\_data.txt",header=TRUE)  
dissertation<-dissertation[,1:44]  
predictors<-c("obguilt","reasdoubt","bettertolet","libertyvorder",  
 "jurevidence","guilt")  
dissertation<-subset(dissertation, !is.na(libertyvorder))  
  
dissertation[45:50]<-lapply(dissertation[predictors], function(x) {  
 y<-scale(x, center=TRUE, scale=TRUE)  
 }  
)  
  
pred\_int<-'verdict ~ obguilt.1 + reasdoubt.1 + bettertolet.1 + libertyvorder.1 +   
 jurevidence.1 + guilt.1 + obguilt.1\*guilt.1 + reasdoubt.1\*guilt.1 +  
 bettertolet.1\*guilt.1 + libertyvorder.1\*guilt.1 + jurevidence.1\*guilt.1'  
model<-glm(pred\_int, family="binomial", data=dissertation)  
summary(model)

##   
## Call:  
## glm(formula = pred\_int, family = "binomial", data = dissertation)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.6101 -0.5432 -0.1289 0.6422 2.2805   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -0.47994 0.16264 -2.951 0.00317 \*\*   
## obguilt.1 0.25161 0.16158 1.557 0.11942   
## reasdoubt.1 -0.09230 0.20037 -0.461 0.64507   
## bettertolet.1 -0.22484 0.20340 -1.105 0.26899   
## libertyvorder.1 0.05825 0.21517 0.271 0.78660   
## jurevidence.1 0.07252 0.19376 0.374 0.70819   
## guilt.1 2.31003 0.26867 8.598 < 2e-16 \*\*\*  
## obguilt.1:guilt.1 0.14058 0.23411 0.600 0.54818   
## reasdoubt.1:guilt.1 -0.61724 0.29693 -2.079 0.03764 \*   
## bettertolet.1:guilt.1 0.02579 0.30123 0.086 0.93178   
## libertyvorder.1:guilt.1 -0.27492 0.29355 -0.937 0.34899   
## jurevidence.1:guilt.1 0.27601 0.36181 0.763 0.44555   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 490.08 on 354 degrees of freedom  
## Residual deviance: 300.66 on 343 degrees of freedom  
## AIC: 324.66  
##   
## Number of Fisher Scoring iterations: 6

The predict function, which I introduced [here](https://www.deeplytrivial.com/2018/04/y-is-for-ys-y-hats-and-residuals.html), can also be used for the binomial model. Let’s have R generate predicted scores for everyone in the dissertation sample:

dissertation$predver<-predict(model)  
dissertation$predver

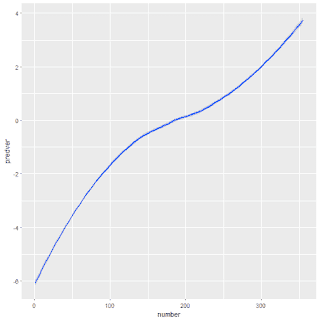
## [1] 0.3907097456 -4.1351129605 2.1820478279 -2.8768390246 2.5804618523  
## [6] 0.4244692909 2.3065468369 -2.7853434926 0.3504760502 -0.2747339639  
## [11] -1.8506160725 -0.6956240161 -4.7860574839 -0.3875950731 -2.4955679446  
## [16] -0.3941516951 -4.5831011509 1.6185480937 0.4971923298 4.1581842900  
## [21] -0.6320531052 -4.8447046319 -2.3974890696 1.8566258698 0.0360685822  
## [26] 2.2151040131 2.3477149003 -2.4493726369 -0.2253481404 -4.8899805287  
## [31] 1.7789459288 -0.0978703861 -3.5541042186 -3.6009218603 0.1568318789  
## [36] 3.7866003489 -0.6371816898 -0.7047761441 -0.7529742376 -0.0302759317  
## [41] -0.1108055330 1.9751810033 0.2373614802 0.0424471071 -0.4018757856  
## [46] 0.0530272726 -1.0763759980 0.0099577637 0.3128581222 1.4806679691  
## [51] -1.7468626219 0.2998282372 -3.6359162016 -2.2200774510 0.3192366472  
## [56] 3.0103216033 -2.0625775984 -6.0179845235 2.0300503627 2.3676828409  
## [61] -2.8971753746 -3.2131490026 2.1349358889 3.0215336139 1.2436192890  
## [66] 0.2885535375 0.2141821004 1.9480686936 0.0438751446 -1.9368013875  
## [71] 0.2931258287 0.5319938265 0.0177643261 3.3724920900 0.0332949791  
## [76] 2.5935500970 0.7571810150 0.7131757400 2.5411073339 2.8499853550  
## [81] 2.8063291084 -0.4500738791 1.4700679077 -0.8659309719 0.0870492258  
## [86] 0.5728074322 0.1476797509 2.4697257261 2.5935500970 -2.2200774510  
## [91] -0.0941827753 1.3708676633 1.4345235392 -0.2407209578 2.4662700339  
## [96] -1.9687731888 -6.7412580522 -0.0006224018 -4.4132951092 -2.8543032695  
## [101] 1.2295635352 2.8194173530 0.1215689324 -3.8258079371 1.8959803882  
## [106] -4.5578801595 2.3754402614 0.0826808026 1.5112359711 -3.5402060466  
## [111] 0.2556657363 0.7054183194 1.4675797244 -2.3974890696 2.6955929822  
## [116] -0.3123518919 -4.8431862346 -2.0132721372 0.4673405434 -2.3053405270  
## [121] 1.9498822386 -0.5164183930 -1.8277820872 -0.0134750769 -2.3013547136  
## [126] -0.2498730859 -4.4281010683 -0.0134750769 -0.2604532514 0.1476797509  
## [131] -2.3392939519 -2.0625775984 -3.5541042186 1.5087477879 -4.6453051124  
## [136] 2.0616474606 -3.2691362859 -7.3752231145 -1.6666447439 1.0532964013  
## [141] -2.0625775984 -0.3355312717 2.2481601983 -2.2200774510 -4.3276959075  
## [146] 0.8685972087 -0.7727065311 1.7511589809 -0.4774548995 0.0008056357  
## [151] 1.7022334970 -0.4202625135 -0.2902646169 2.4409712692 0.0008056357  
## [156] 0.0008056357 -3.6009218603 -0.8567788439 -0.4528474822 0.3517462520  
## [161] 0.1307210605 -3.7843118182 -2.8419024763 -3.5191098774 -0.1460684795  
## [166] 1.8809888141 2.8194173530 -2.4656469123 1.0589888029 0.1659840070  
## [171] 1.4345235392 2.3676828409 1.5749534339 -0.1681557545 2.6406620359  
## [176] 0.1476797509 -2.2135177411 1.9168260534 -3.4993205379 0.4557086940  
## [181] -3.8136089417 -0.1121510987 -3.9772095600 1.3849234171 0.3504760502  
## [186] 2.3807710856 -3.0667307601 2.3040586537 1.7599138086 -0.2083894500  
## [191] 0.6844579761 -0.3552635652 -1.9459392035 -0.6075281598 -2.1663310490  
## [196] 2.3676828409 -1.9205271122 -2.2334295071 -4.4265826710 -1.0117771483  
## [201] -0.0161530548 -0.3072233074 -0.0161530548 -0.7451676752 -7.0351269313  
## [206] 2.6406620359 -3.7523234832 -0.2498730859 2.0222929422 3.2886316225  
## [211] -1.6221457956 2.4749949634 1.7570711677 0.0904873650 -4.7332807307  
## [216] 0.1568318789 -0.0302759317 0.5127229828 1.3097316594 -6.9309218514  
## [221] 0.0515992352 -0.4514194447 -0.2253481404 -4.7652690656 -0.4279866041  
## [226] -4.4136563866 -3.7618312672 0.0156676181 -0.2590252139 2.6076058507  
## [231] 1.6420333133 -3.9985172969 -6.2076483227 0.1632104039 0.1829426974  
## [236] -4.7652690656 -4.4212844958 1.6001906117 0.8579971472 -3.8699110198  
## [241] 0.3022779567 -0.1679979189 1.9421248181 0.6592738895 1.6132788564  
## [246] -0.0366544567 -3.4818233673 -3.9422152187 -0.3473613776 0.4321933815  
## [251] 0.7480288869 -0.2498730859 -1.9861068488 -2.2297920164 -0.7621263656  
## [256] 1.2966434147 0.1632104039 0.2048721368 1.7789459288 0.4926393080  
## [261] 0.4096285430 -1.7794744955 -2.5822853071 2.0413250624 -6.6574350219  
## [266] -0.1277642235 -2.1972434657 -2.5075677545 -0.4482774141 -0.6943740757  
## [271] -0.7821891015 6.3289445390 0.1568318789 0.1165981835 1.4781797859  
## [276] -4.2287015488 -3.6157278195 -0.1511970641 -0.7047761441 2.0935344484  
## [281] -3.8258079371 -4.4231102471 1.3097316594 3.4081542651 -0.4996175382  
## [286] -2.0534397824 0.9783975145 -2.2562634924 3.7196170683 1.1110084017  
## [291] 2.1661785291 -4.2138955896 1.9421248181 2.3065468369 -0.7139282722  
## [296] -4.1431023472 -2.0854115837 2.9389399956 1.7711269214 -0.0302759317  
## [301] -2.6458711124 0.5856241187 -0.1199576611 1.8566258698 -2.2383553905  
## [306] 2.3807710856 -0.2838860920 3.1176953128 2.8499853550 2.8063291084  
## [311] 0.0034011417 -0.4683781352 -3.0377484314 -1.3833686805 1.7764577456  
## [316] 1.7842151661 3.4081542651 0.1165981835 -4.6988069009 -2.6013721641  
## [321] 2.0616474606 -0.2498730859 -4.2207121622 4.1705330009 5.2103776377  
## [326] -4.5406977837 -1.5080855068 -2.5232652805 -5.7259789038 2.5211393933  
## [331] -0.3487069432 -2.5035573312 -2.2764097339 -5.8364854607 -1.8694684539  
## [336] 1.3402996614 0.5728074322 0.3663267540 -0.1603491921 -2.1690805453  
## [341] -1.4105339689 3.0768201201 -5.1065624241 -4.5966850670 -4.5498907729  
## [346] -1.3078399029 -1.0882592824 0.3128581222 -0.3644156933 0.3100845191  
## [351] 2.4774831467 -1.0763759980 2.2151040131 -0.0952748801 -4.6864864366

Now, remember that the outcome variable is not guilty (0) and guilty (1), so you might be wondering – what’s with these predicted values? Why aren’t they 0 or 1?

Binomial regression is used for nonlinear outcomes. Since the outcome is 0/1, it’s nonlinear. But binomial regression is based on the general linear model. So how can we apply the general linear model to a nonlinear outcome? Answer: by transforming scores. Specifically, it transforms the outcome into a [log odds ratio](https://www.deeplytrivial.com/2017/09/statistics-sunday-everyone-loves-log.html); the log transform makes the outcome variable behave somewhat linearly and symmetrically. The predicted outcome, then, is also a log odds ratio.

ordvalues<-dissertation[order(dissertation$predver),]  
ordvalues<-ordvalues[,51]  
ordvalues<-data.frame(1:355,ordvalues)  
colnames(ordvalues)<-c("number","predver")  
library(ggplot2)  
ggplot(data=ordvalues, aes(number,predver))+geom\_smooth()

## `geom\_smooth()` using method = 'loess'

[](https://i0.wp.com/3.bp.blogspot.com/-t68kCtjcevo/WueGreJepKI/AAAAAAAALbs/QsSzDQL-QjgO27QRZkuhkXsZrqP9GRTiQCLcBGAs/s1600/unnamed-chunk-3-1.png?ssl=1)

Log odds ratios are great for analysis, but when trying to understand how well your model is predicting values, we want to convert them into a metric that’s easier to understand in isolation and when compared to the observed values. We can convert them into probabilities with the following equation:

dissertation$verdict\_predicted<-exp(predict(model))/(1+exp(predict(model)))

This gives us a value ranging from 0 to 1, which is the probability that a particular person will select guilty. We can use this value in different ways to see how well our model is doing. Typically, we’ll divide at the 50% mark, so anyone with a probability of 0.5 or greater is predicted to select guilty, and anyone with a probability less than 0.5 would be predicted to select not guilty. We then compare this new variable with the observed results to see how well the model did.

dissertation$vpred\_rounded<-round(dissertation$verdict\_predicted,digits=0)  
library(expss)

## Warning: package 'expss' was built under R version 3.4.4

dissertation<- apply\_labels(dissertation,  
 verdict = "Actual Verdict",  
 verdict = c("Not Guilty" = 0,  
 "Guilty" = 1),  
 vpred\_rounded = "Predicted Verdict",  
 vpred\_rounded = c("Not Guilty" = 0,  
 "Guilty" = 1)  
)  
cro(dissertation$verdict,list(dissertation$vpred\_rounded, total()))

|  | **Predicted Verdict** | |  | **#Total** |
| --- | --- | --- | --- | --- |
|  | **Not Guilty** | **Guilty** |  |  |
| **Actual Verdict** | | | | |
| Not Guilty | 152 | 39 |  | 191 |
| Guilty | 35 | 129 |  | 164 |
| #Total cases | 187 | 168 |  | 355 |

One thing we could look at regarding this table, which when dealing with actual versus predicted categories is known as a confusion matrix, is how well the model did at correctly categorizing cases – which we get by adding together the number of people with both observed and predicted not guilty, and people with observed and predicted guilty, then dividing that sum by the total.

accuracy<-(152+129)/355  
accuracy

## [1] 0.7915493

Our model correctly classified 79% of the cases. However, this is not the only way we can determine how well our model did. There are a [variety of derivations](https://en.wikipedia.org/wiki/Sensitivity_and_specificity#Definitions) you can make from the confusion matrix. But two you should definitely include when doing this kind of analysis are sensitivity and specificity. Sensitivity refers to the true positive rate, and specificity refers to the true negative rate.

When you’re working with confusion matrices, you’re often trying to diagnose or identify some condition, one that may be deemed positive or present, and the other that may be deemed negative or absent. These derivations are important because they look at how well your model identifies these different states. For instance, if most of my cases selected not guilty, I could get a high accuracy rate by simply predicting that *everyone* will select not guilty. But then my model lacks sensitivity – it only identifies negative cases (not guilty) and fails to identify any positive cases (guilty). If I were dealing with something even higher stakes, like whether a test result indicates the presence of a condition, I want to make certain my classification is sensitive to those positive cases. And vice versa, I could keep from missing any positive cases by just classifying everyone as positive, but then my model lacks specificity and I may subject people to treatment they don’t need (and that could be harmful).

Just like accuracy, sensitivity and specificity are easy to calculate. As I said above, I’ll consider not guilty to be negative and guilty to be positive. Sensitivity is simply the number of true positives (observed and predicted guilty) divided by the sum of true positives and false negatives (people who selected guilty but were classified as not guilty).

sensitivity<-129/164  
sensitivity

## [1] 0.7865854

And specificity is the number of true negatives (observed and predicted not guilty) divided by the sum of true negatives and false positives (people who selected not guilty but were classified as guilty).

specificity<-152/191  
specificity

## [1] 0.7958115

So the model correctly classifies 79% of the positive cases and 80% of the negative cases. The model could be improved, but it’s functioning equally well across positive and negative cases, which is good.

It should be pointed out that you can select any cutpoint you want for your probability variable. That is, if I want to be very conservative in identifying positive cases, I might want there to be a higher probability that it is a positive case before I classify it as such – perhaps I want to use a cutpoint like 75%. I can easily do that.

dissertation$vpred2[dissertation$verdict\_predicted < 0.75]<-0  
dissertation$vpred2[dissertation$verdict\_predicted >= 0.75]<-1  
dissertation<- apply\_labels(dissertation,  
 vpred2 = "Predicted Verdict (0.75 cut)",  
 vpred2 = c("Not Guilty" = 0,  
 "Guilty" = 1)  
)  
cro(dissertation$verdict,list(dissertation$vpred2, total()))

|  | **Predicted Verdict (0.75 cut)** | |  | **#Total** |
| --- | --- | --- | --- | --- |
|  | **Not Guilty** | **Guilty** |  |  |
| **Actual Verdict** | | | | |
| Not Guilty | 177 | 14 |  | 191 |
| Guilty | 80 | 84 |  | 164 |
| #Total cases | 257 | 98 |  | 355 |

accuracy2<-(177+84)/355  
sensitivity2<-84/164  
specificity2<-177/191  
accuracy2

## [1] 0.7352113

sensitivity2

## [1] 0.5121951

specificity2

## [1] 0.9267016

Changing the cut score improves specificity but at the cost of sensitivity, which makes sense, because our model was predicting equally well (or poorly, depending on how you look at it) across positives and negatives. In this case, a different cut score won’t improve our model. We would need to go back and see if there are better variables to use for prediction. And to keep us from fishing around in our data, we’d probably want to use a training and testing set for such exploratory analysis.