**Logistic regression models**

The purpose of the next two sub-sections is to examine how to use logistic and multinomial regression models to obtain a probability that an observation belongs to a particular population. The logistic regression model will be used for the two population case, and the multinomial regression model will be used for the more than two population case. These models are discussed extensively in my Categorical Data Analysis course. For this reason, I will only go through the basics of these models so that we can focus more on our purpose for their use here–classification.

For more in depth information about these models, please see my Categorical Data Analysis course website at [www.chrisbilder.com/categorical](http://www.chrisbilder.com/categorical) or my *Categorical Data Analysis with R* book.

**Logistic regression basics**

Review of normal linear regression models

I will be more formal here with random variables (capital letters) and their observed values (lowercase letters).

Yr = β0 + β1xr1 + β2xr2 + … + βpxrp + εr

where εr ~ independent N(0, σ2) and r = 1, …, N

Note that

E(Yr) = β0 + β1xr1 + β2xr2 + … + βpxrp

E(Yr) is what one would expect Yr to be on average for a set of xr1, xr2, …, xrp values. Also, one can show that Var(Yr) = σ2. Thus, Yr ~ N(β0 + β1xr1 + β2xr2 + … + βpxrp, σ2) for r = 1, …, N.

If βk (one of the β’s above) is equal to 0, this says there is no linear relationship between the corresponding independent (explanatory) variable xk and the dependent (response) variable. If βk > 0, there is a positive relationship, and if βk < 0, there is a negative relationship. All of these statements are with respect to the other independent variables in the model remaining constant.

We can also express the model in a matrix form as

E(**Y**)= **Xβ**

where



 is the estimate of **β**

Logistic regression model

Let Yr be a binary response variable for r = 1, …, N, where a 1 denotes a success and a 0 denotes a failure. We let Yr ~ Bernoulli(πr) where E(Yr) = πr and Var(Yr) = πr(1 – πr). Notice how the variance could be potentially different for each Yr, unlike for normal linear regression.

The logistic regression model is expressed in terms of the E(Yr):



Notice that , so that the numerator is always less than the denominator. Thus, 0 < πr < 1, which is the reason for using this model rather than simply setting E(Yr) equal to β0 + β1xr1 + … + βpxrp. Other transformations of β0 + β1xr1 + … + βpxrp are sometimes used to guarantee 0 < πr < 1, but the one used with logistic regression is the most frequently used in practice. The source of this transformation is the inverse of the logistic cumulative distribution function.

The logistic regression model can also be written as



through using some algebra. Most often, you will see log(πr/(1-πr)) written instead as logit(πr). Thus, the “logit transformation” refers to the log of πr/(1-πr).

We can write the model without the r subscript when we want to state the model in general:

 and 

Questions:

* Suppose β1 > 0. What does this say about the relationship between x1 and the response Y?
* Suppose β1 < 0. What does this say about the relationship between x1 and the response Y?

One can more succinctly quantify the relationship between the independent variable x1 and the Y response by using “odds ratios”. This is discussed in the Categorical Data Analysis course.

Example: Plot of π vs. x (PiPlot.R)

When there is only one independent variable, β0 = 1, and β1 = 0.5 (or -0.5), a plot of π vs. x1 looks like the following:

A graph of a function

Description automatically generated with medium confidence

We can make the following generalizations:

* 0 < π < 1
* When β1 > 0, there is a positive relationship between x1 and π.
* When β1 < 0, there is a negative relationship between x1 and π.
* The shape of the curve is somewhat similar to the letter s. This is referred to as a sigmoidal shape.
* The shape of the curve is exactly the same if it is rotated 180 degrees about π = 0.5; thus, the shape is “rotationally symmetric”.
* The slope of the curve is dependent on the value of x1. We can show this mathematically by taking the derivative with respect to x1: 

R code:

> par(mfrow = c(1,2))

> beta0 <- 1

> beta1 <- 0.5

> curve(expr = exp(beta0+beta1\*x)/(1+exp(beta0+beta1\*x)),

xlim = c(-15, 15), col = "black", main = expression(beta[1] == 0.5), xlab = expression(x[1]), ylab = expression(pi))

> beta0 <- 1

> beta1 <- -0.5

> curve(expr = exp(beta0+beta1\*x)/(1+exp(beta0+beta1\*x)),

xlim = c(-15, 15), col = "black", main = expression(beta[1] == -0.5), xlab = expression(x[1]), ylab = expression(pi))

Questions:

* What happens to the β1 = 0.5 plot when β1 is increased?
* What happens to the β1 = 0.5 plot when β1 is decreased to be close to 0?
* Suppose a plot of logit(π) vs. x1 was made. What would the plot look like?

Maximum likelihood estimation is used to estimate the parameters of the model. The likelihood function is



where



in the above expression. Thus, we can write  now as . We can find the log likelihood function as:



Taking derivatives with respect to β0, …, βp, setting them equal to 0, and then solving for the parameters leads to the maximum likelihood estimates (MLEs). These parameter estimates are denoted by  …, . Corresponding estimates of π are



Unfortunately, there are no closed form expressions that can be written out for  …,  except in very simple cases. The MLEs instead are found through using iterative numerical procedures. Most software packages use an iterative numerical procedure known as iteratively reweighted least squares (IRLS) to find the MLEs for logistic regression models.

Without going into all of the details behind IRLS, initial estimates for the parameters, say  …, , are found. Weighted least squares estimation (topic in a regression analysis course; weights are based on ) is used to find a “better” set of parameter estimates.

If the new parameter estimates, say  …, , are very close to  …, , the iterative numerical procedure is said to “converge” and  …,  are used as the MLEs  …, . If the new parameter estimates  …,  are not very close to  …,  weighted least squares estimation is used again with new weights. This iterative process continues until convergence or a prior-specified maximum number of iterations is reached.

Question: If the prior-specified maximum number of iterations limit is reached, should the last set of parameter estimates be used as  …, ?

Example: Placekicking data (PlacekickLogisticReg.R, Placekick.csv, valid.csv)

The purpose here is to show how to estimate the model



using the glm() function. Note that it is o.k. to be a little less formal and replace the x’s with their corresponding variable names:



> placekick <- read.csv(file = "Placekick.csv")

> valid <- read.csv(file = "valid.csv")

> mod.fit <- glm(formula = good ~ week + distance + change + elap30 + PAT + type + field + wind, data = placekick,

family = binomial(link = logit))

> mod.fit

Call: glm(formula = good ~ week + distance + change + elap30 + PAT + type + field + wind, family = binomial(link = logit), data = placekick)

Coefficients:

(Intercept) week distance change

4.781039 -0.024774 -0.085899 -0.349698

elap30 PAT type field

0.004795 1.245801 0.309970 -0.216276

wind

-0.630237

Degrees of Freedom: 1424 Total (i.e. Null); 1416 Residual

Null Deviance: 1013

Residual Deviance: 753.5 AIC: 771.5

> names(mod.fit)

[1] "coefficients" "residuals"

[3] "fitted.values" "effects"

<OUTPUT EDITED>

[29] "contrasts" "xlevels"

> mod.fit$coefficients

(Intercept) week distance change

4.781038874 -0.024774073 -0.085899002 -0.349697755

elap30 PAT type field

0.004794629 1.245801150 0.309970162 -0.216275715

wind

-0.630236532

> class(mod.fit)

[1] "glm" "lm"

> methods(class = glm)

[1] add1 addterm anova

[4] coerce confint cooks.distance

[7] deviance drop1 dropterm

[10] effects ellipse3d extractAIC

[13] family formula gamma.shape

[16] influence initialize logLik

[19] model.frame nobs predict

[22] print profile residuals

[25] rstandard rstudent show

[28] slotsFromS3 summary vcov

[31] weights

see '?methods' for accessing help and source code

> methods(class = lm)

[1] add1 addterm alias

[4] anova boxcox case.names

[7] coerce confint cooks.distance

[10] deviance dfbeta dfbetas

[13] drop1 dropterm dummy.coef

[16] effects ellipse3d extractAIC

[19] family formula hatvalues

[22] influence initialize kappa

[25] labels logLik logtrans

[28] model.frame model.matrix nobs

[31] plot plot3d predict

[34] print proj qr

[37] residuals rstandard rstudent

[40] show simulate slotsFromS3

[43] summary variable.names vcov

see '?methods' for accessing help and source code

> summary(mod.fit)

Call:

glm(formula = good ~ week + distance + change + elap30 + PAT + type + field + wind, family = binomial(link = logit), data = placekick)

Deviance Residuals:

Min 1Q Median 3Q Max

-3.0283 0.1614 0.1871 0.4722 1.4888

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) 4.781039 0.558442 8.561 < 2e-16 \*\*\*

week -0.024774 0.019528 -1.269 0.20456

distance -0.085899 0.011228 -7.650 2e-14 \*\*\*

change -0.349698 0.195358 -1.790 0.07345 .

elap30 0.004795 0.010552 0.454 0.64955

PAT 1.245801 0.385085 3.235 0.00122 \*\*

type 0.309970 0.279175 1.110 0.26687

field -0.216276 0.254247 -0.851 0.39496

wind -0.630237 0.343864 -1.833 0.06683 .

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 1013.43 on 1424 degrees of freedom

Residual deviance: 753.52 on 1416 degrees of freedom

AIC: 771.52

Number of Fisher Scoring iterations: 6

> library(car)

> Anova(mod.fit)

Analysis of Deviance Table (Type II tests)

Response: good

LR Chisq Df Pr(>Chisq)

week 1.617 1 0.2035650

distance 69.122 1 < 2.2e-16 \*\*\*

change 3.175 1 0.0747900 .

elap30 0.207 1 0.6493575

PAT 11.117 1 0.0008554 \*\*\*

type 1.247 1 0.2640669

field 0.734 1 0.3915089

wind 3.224 1 0.0725444 .

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Comments:

* R models the probability that the good variable is equal to 1 by default. If we instead had the good variable equal to “success” and “failure” values (needs to be of a factor type), R would model the probability of “success”. This results from how R recognizes the order of values within a variable. Simply, R always models the probability of the last level in a numerical/alphabetical ordering (0, 1, 2, …, 9, …, a, A, b, B, …, z, Z) of the levels within the variable (see the DA notes for another discussion on this topic).
* The estimated logistic regression model is



* Questions:
  + What happens to the estimated probability of success as the distance increases?
  + What happens to the estimated probability of success for a lead-change (1) as opposed to a non-lead-change (0) placekicks?
  + Which variables have β parameters significantly different from 0?

**Logistic regression for prediction**

Again, the purpose of us examining logistic regression models is to use the model to predict if an observation is from one of two populations. This prediction begins by examining the estimated probability success



for each observation. One commonly uses the following criteria to classify an observation:

If the estimated probability of success is greater than 0.5, we can classify the observation into the Y = 1 population; otherwise, we classify the observation into the Y = 0 population.

While using 0.5 as the “cut-off” probability value is intuitive, it is not necessary. One can sometimes achieve a better overall classification accuracy by using a cut-off which is different than 0.5. To decide where to make this cut-off, a receiver operating characteristic (ROC) curve can be examined.

ROC curves

In order to understand what a ROC curve represents, we need to discuss what the “sensitivity” and “specificity” represent for a classification method.

Example: HIV testing (partially from the DA notes)

Suppose a clinical trial is being conducted on a new HIV test. The test measures a number of different variables related to the presence of HIV. Using a classification method, a rule is developed to classify the subjects as negative or positive. Suppose an older, more expensive test (a “gold standard”) can be used to determine if someone is really HIV positive or not. Below are the possible outcomes:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | HIV test results | |
|  |  | Negative | Positive |
| HIV actual | No | Correct | Error |
| Yes | Error | Correct |

The sensitivity of the classification method is the proportion of true (actual) positives correctly classified as positive. In a notational form,



The specificity of the classification method is the proportion of true (actual) negatives correctly classified as negatives. In a notational form,



For example, suppose the following results occurred in a clinical trial:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | HIV test results | |
|  |  | Negative | Positive |
| HIV actual | No | 195 | 5 |
| Yes | 1 | 99 |

The sensitivity is 99/100 = 0.99, and the specificity is 195/200 = 0.975.

Of course, the larger the sensitivity and specificity are, the better the classification method is. Conversely, one often works with the compliments of these accuracy measures.

The false negative rate (1-sensitivity) of the classification method is the proportion of true (actual) positives incorrectly classified as negative. In a notational form,

.

The false positive rate (1-specificity) of the classification method is the proportion of true (actual) negatives incorrectly classified as positives. In a notational form,

.

For this example, the false negative rate is 1/100 = 0.01 and the false positive rate is 5/200 = 0.025. Of course, we would like these measures to be as small as possible.

In application, one can usually increase the sensitivity of a classification method, but at a cost of increasing the false positive rate. Conversely, one can usually increase the specificity of a classification method, but at a cost of increasing the false negative rate. This can make it difficult to decide then on how to implement a particular classification method.

Example: Placekicking data (PlacekickLogisticReg.R, Placekick.csv, valid.csv)

In words, what would the sensitivity and specificity represent for this example? Suppose “positive” corresponds to good = 1, and “negative” corresponds to good = 0.

The estimated probability of success can be found as follows:

> head(mod.fit$fitted.values)

1 2 3 4 5 6

0.9434752 0.9577942 0.9876288 0.9248228 0.9877382 0.9366464

> pi.hat <- predict(object = mod.fit, type = "response")

> head(pi.hat)

1 2 3 4 5 6

0.9434752 0.9577942 0.9876288 0.9248228 0.9877382 0.9366464

> #Validation data

> pi.hat.v <- predict(object = mod.fit, newdata = valid, type = "response")

> head(pi.hat.v)

1 2 3 4 5 6

0.9433265 0.6635789 0.4760922 0.9877546 0.9866572 0.9849546

Using a 0.5 cut-off probability, we obtain the following accuracy levels:

> cutoff <- 0.5

> pred.good <- ifelse(test = pi.hat > cutoff, yes = 1, no =

0)

> pred.good.v <- ifelse(test = pi.hat.v > cutoff, yes = 1, no

= 0)

> summarize.class <- function(original, classify) {

class.table <- table(original, classify)

numb <- rowSums(class.table)

prop <- round(class.table/numb,4)

overall <- round(sum(diag(class.table)) /

sum(class.table), 4)

list(class.table = class.table, prop = prop,

overall.correct = overall)

}

> summarize.class(original = placekick[,9], classify =

pred.good) #Resubstitution

$class.table

classify

original 0 1

0 18 145

1 13 1249

$prop

classify

original 0 1

0 0.1104 0.8896

1 0.0103 0.9897

$overall.correct

[1] 0.8891

> summarize.class(original = valid[,9], classify =

pred.good.v) #Validation data

$class.table

classify

original 0 1

0 3 35

1 8 236

$prop

classify

original 0 1

0 0.0789 0.9211

1 0.0328 0.9672

$overall.correct

[1] 0.8475

The sensitivity for the original data using resubstitution is 0.9897, but the specificity is only 0.1104.

The overall accuracy values are somewhat similar to what we have seen before. What if we chose a different cut-off probability? Using the same code as before except changing cut-off to 0.7, this is what I obtain:

> summarize.class(original = placekick[,9], classify =

pred.good) #Resubstitution

$class.table

classify

original 0 1

0 79 84

1 109 1153

$prop

classify

original 0 1

0 0.4847 0.5153

1 0.0864 0.9136

$overall.correct

[1] 0.8646

> summarize.class(original = valid[,9], classify =

pred.good.v) #Validation data

$class.table

classify

original 0 1

0 23 15

1 29 215

$prop

classify

original 0 1

0 0.6053 0.3947

1 0.1189 0.8811

$overall.correct

[1] 0.844

Large improvements occur with the specificity, but with a cost of lower sensitivity. Depending on the situation, this may be preferred.

Cross-validation here involves removing one observation at a time, re-fit the model with N – 1 observations, and then obtain the removed observation’s estimate of π. This process is repeated over all observations in the data set. The resulting estimates of π then would be used to obtain the classifications for good = 0 or 1 in the same way as we have already done for resubstitution and the validation data set. Because the sample size is large here, cross-validation should obtain very similar classification accuracy values to those for resubstitution.

There is not a specific function that will allow one to automatically perform cross-validation. Below is a simple function that I wrote to obtain the estimates of π:

> cv <- function(model, data.set) {

N <- nrow(data.set)

pi.hat.cv <- numeric(length = N)

for(r in 1:N) {

mod.fit <- glm(formula = model, data = data.set[-r,],

family = binomial(link = logit))

pi.hat.cv[r] <- predict(object = mod.fit, newdata =

data.set[r,], type = "response")

#print(r) #Uncomment and turn off buffered output

(MISC > BUFFERED OUTPUT) if want to see print(r)

}

pi.hat.cv

}

> save.cv <- cv(model = good ~ week + distance + change +

elap30 + PAT + type + field + wind, data.set =

placekick)

> cutoff <- 0.5

> pred.good.cv <- ifelse(test = save.cv > cutoff, yes = 1, no = 0)

> summarize.class(original = placekick[,9], classify =

pred.good.cv) #cross-validation

$class.table

classify

original 0 1

0 17 146

1 14 1248

$prop

classify

original 0 1

0 0.1043 0.8957

1 0.0111 0.9889

$overall.correct

[1] 0.8877

Indeed, the resubstitution and cross-validation accuracy measures are very similar. For this reason, I will not use the cross-validation classifications when I continue to work with this data.

We can also create similar plots as in the DA and NNC notes that show the classifications. Below are PCA plots using resubstitution where a cut-off probability of 0.5 is used (see program for code):

A graph with red dots

Description automatically generated

A diagram of a graph

Description automatically generated

Compare these plots to those obtained with DA and NNC.

When the cut-off probability is changed to 0.7, the following plots are obtained:

A graph with red and black dots

Description automatically generated

A diagram of a graph

Description automatically generated

With a larger cut-off probability, the criteria to be classified as a success is more stringent. Thus, we see more placekicks classified as failures.

Below are the accuracy values for logistic regression with a cut-off probability of 0.7 (sensitivity and specificity are very similar to LDA in the notes):

> summarize.class(original = placekick[,9], classify =

pred.good) #Resubstitution

$class.table

classify

original 0 1

0 79 84

1 109 1153

$prop

classify

original 0 1

0 0.4847 0.5153

1 0.0864 0.9136

$overall.correct

[1] 0.8646

A receiver operating characteristic (ROC) curve plots the sensitivity vs. 1-specificity (false positive rate) for various cut-off probabilities used with a logistic regression model. This allows a researcher then to choose an appropriate level of sensitivity and false positive rate that they are comfortable with. Once chosen, the corresponding cut-off probability to these levels can then be used in application.

Please see the inset on p. 85 of Swets, Dawes, and Monahan (2000) for a nice demonstration on how to interpret a ROC curve. I recommend examining the left plot in the following order:

1. The 1st x and y axes (true positive probability and false positive probability, respectively)
2. The line for “chance” accuracy (probability of testing positive is the same no matter if actual is positive or negative!)
3. The ROC curve for the particular classification method (A number of cut-off probabilities are chosen, and the corresponding sensitivity and 1 – specificity are plotted to form the curve)
4. Choose a location on the ROC curve corresponding to the false positive rate that you can live with along with the desired sensitivity. Once you determine this, one could find the corresponding cut-off probability value (this cut-off cannot be found from by simply looking at this particular plot).
5. Examine the 2nd x and y axes (true negative probability and false negative probability, respectively).
6. Overall, the more area under the curve, the better the procedure does in classification.

What would the most ideal ROC curve look like?

Example: Placekicking data (PlacekickLogisticReg.R, Placekick.csv, valid.csv)

There are a number of packages available to construct ROC curves. Below is how I used the ROCR package:

> library(ROCR) #This is a S4 package

> save.pred <- prediction(predictions = pi.hat, labels =

placekick[,9])

> slotNames(save.pred)

[1] "predictions" "labels" "cutoffs" "fp"

[5] "tp" "tn" "fn" "n.pos"

[9] "n.neg" "n.pos.pred" "n.neg.pred"

> head(save.pred@fp[[1]]) #Number of false positives for a

cut-off

[1] 0 0 0 0 0 0

> tail(save.pred@fp[[1]]) #Number of false positives for a

cut-off

[1] 159 159 160 161 162 163

> head(save.pred@cutoffs[[1]]) #Corresponding cut-offs

70 72 112 27 93

Inf 0.9911915 0.9911545 0.9910281 0.9909890 0.9909675

> tail(save.pred@cutoffs[[1]]) #Corresponding cut-offs

977 1349 956 646 635 1176

0.3591345 0.3301465 0.2991858 0.2904308 0.2520078 0.2375404

> save.perf <- performance(prediction.obj = save.pred,

measure = "sens", x.measure = "fpr")

> slotNames(save.perf)

[1] "x.name" "y.name" "alpha.name"

[4] "x.values" "y.values" "alpha.values"

> plot(save.perf, main = "ROC curve for placekicking data")

> grid()



Comments:

* R and its corresponding packages are written in a form very similar to the S programming language. Versions 3 and 4 of S are emulated by R. Version 3 (S3) is predominant, and this is what we have used so far in our course. For example, this allowed us to access components of a list by using $. Version 4 (S4) is used by the ROCR package. For our purposes, you just need to understand that
  + Components of an object are now called slots,
  + A list of components within an object is obtained by using slotNames() (rather than names()),
  + Slots are accessed by using the @ symbol (rather than $) for objects.
* The ROC curve is definitely above the “line for chance accuracy”. If you wanted to draw this line on the plot, how could you do it?
* The plot() method function here will not recognize panel.first = grid(), which we have used before to plot gridlines. I instead used the grid() function outside of the plot() function. The difference is that the gridlines are drawn AFTER the main plotting is done (gridlines are over the ROC curve – very difficult to see though, so not too important)

To find the 1 – specificity level for a particular sensitivity level or vice versa, we can examine items within save.perf. Unfortunately, this is not necessarily straightforward due to the structure of the items within the object. For example, suppose we want to have a 1 – specificity level of 0.5, and we want to find the corresponding sensitivity and cut-off probability:

> one.minus.spec <- save.perf@x.values[[1]]

> sens <- save.perf@y.values[[1]]

> save.perf@alpha.name #Says cut-off probability

[1] "Cutoff"

> cutoff <- save.perf@alpha.values[[1]]

> ck <- one.minus.spec > 0.495 & one.minus.spec < 0.505

> data.frame(sensitivity = sens[ck], one.minus.specificity

= one.minus.spec[ck], cutoff = cutoff[ck])

sensitivity one.minus.specificity cutoff

1 0.9057052 0.4969325 0.7113763

2 0.9064976 0.4969325 0.7108955

3 0.9072900 0.4969325 0.7103665

4 0.9080824 0.4969325 0.7097014

5 0.9080824 0.5030675 0.7087017

6 0.9088748 0.5030675 0.7084985

7 0.9096672 0.5030675 0.7063810

Thus, the corresponding sensitivity is 0.9081, and the cut-off probability is approximately 0.709. The objects save.perf@x.values, save.perf@y.values, and save.perf@alpha.values are lists without named components (look at these lists on your own). The numerical values are stored in the first component of the list, so I created nicely named objects to help identify them. Also, the save.perf@alpha.name gives a name of “Cutoff” meaning that the corresponding cut-off probabilities for the sensitivity and 1 – specificity values are stored in save.perf@alpha.values.

A more useful version of the ROC plot is shown below. The colorize and print.cuttoff.at arguments allow for information about the cut-off probabilities to be included on the plot.

> plot(save.perf, colorize = TRUE, print.cutoffs.at =

seq(from = 0.3, to = 0.9, by = 0.1), main = "ROC curve

for placekicking data")

> grid()



The ROC curve for the validation data is:

> save.pred.v <- prediction(predictions = pi.hat.v, labels = valid[,9])

> save.perf.v <- performance(prediction.obj = save.pred.v,

measure = "sens", x.measure = "fpr")

> plot(save.perf.v, main = "ROC curve for validation data",

colorize = TRUE, print.cutoffs.at = seq(from = 0.3, to

= 0.9, by = 0.1))

> grid()



Other plots can be created with using information obtained by the performance() function. For example, below is a plot of the sensitivity vs. the cut-off probabilities:

> save.perf2 <- performance(prediction.obj = save.pred,

measure = "sens", x.measure = "cutoff")

> slotNames(save.perf2)

[1] "x.name" "y.name" "alpha.name" "x.values"

[5] "y.values" "alpha.values"

> plot(save.perf2, main = "Sensitivity vs. cutoff

probabilities")

> grid()



Based on all of the calculations so far, what cut-off probability should be used with the placekicking data? I do not think there is one correct answer here.

There are often situations when you have more than one classification method, and you would like to compare their ROC curves to determine which is better. An overall measure of which classification method is best is the area underneath the ROC curves. The method with the most area corresponds to the “best” classification method (without taking into account specifically desired sensitivity or false positive rate levels). Why does the larger area correspond to the better method?

Example: Placekicking data (PlacekickLogisticreg.R, Placekick.csv, valid.csv)

The performance() function provides the area underneath the curve for the model with all of the variables in it.

> #Resubstitution with original data

> save.auc <- performance(prediction.obj = save.pred, measure = "auc")

> slotNames(save.auc)

[1] "x.name" "y.name" "alpha.name" "x.values"

[5] "y.values" "alpha.values"

> save.auc@y.values

[[1]]

[1] 0.8471411

> #Validation data

> performance(prediction.obj = save.pred.v, measure = "auc")@y.values

[[1]]

[1] 0.839409

Both resubstitution with the original data set and the validation data set yield similar areas.

Next, I examine what happens when only distance or only elap30 is used for the logistic regression model.

> mod.fit.dist <- glm(formula = good ~ distance, data =

placekick, family = binomial(link = logit))

> round(summary(mod.fit.dist)$coefficients, 4) #Just part

of output to save space

Estimate Std. Error z value Pr(>|z|)

(Intercept) 5.8121 0.3263 17.8133 0

distance -0.1150 0.0083 -13.7937 0

> pi.hat.dist <- predict(object = mod.fit.dist, type =

"response")

> head(pi.hat.dist)

1 2 3 4 5 6

0.9675956 0.9675956 0.9710145 0.9303017 0.9710145 0.9496174

> save.pred.dist <- prediction(predictions = pi.hat.dist,

labels = placekick[,9])

> save.perf.dist <- performance(prediction.obj =

save.pred.dist, "sens", "fpr")

> mod.fit.elap30 <- glm(formula = good ~ elap30, data =

placekick, family = binomial(link = logit))

> round(summary(mod.fit.elap30)$coefficients, 4)

Estimate Std. Error z value Pr(>|z|)

(Intercept) 1.779 0.1352 13.1555 0.0000

elap30 0.023 0.0097 2.3708 0.0177

> pi.hat.elap30 <- predict(object = mod.fit.elap30, type =

"response")

> head(pi.hat.elap30)

1 2 3 4 5 6

0.9128199 0.8951287 0.8568474 0.8900480 0.9074497 0.8990290

> save.pred.elap30 <- prediction(predictions = pi.hat.elap30, labels = placekick[,9])

> save.perf.elap30 <- performance(prediction.obj =

save.pred.elap30, "sens", "fpr")

> plot(save.perf, print.cutoffs.at = seq(from = 0.3, to =

0.9, by = 0.1), main = "ROC curves")

> grid()

> plot(save.perf.dist, print.cutoffs.at = seq(from = 0.3,

to = 0.9, by = 0.1), add = TRUE, col = "red")

> plot(save.perf.elap30, print.cutoffs.at = seq(from = 0.3,

to = 0.9, by = 0.1), add = TRUE, col = "blue")

> legend(x = 0.5, y = 0.4, legend = c("All variables",

"Distance only", "Elap30 only"), col = c("black",

"red", "blue"), lty = c(1,1,1), bty = "n")



> #Area underneath the curve

> performance(prediction.obj = save.pred, "auc")@y.values

#All variables

[[1]]

[1] 0.8471411

> performance(prediction.obj = save.pred.dist,

"auc")@y.values #Only distance

[[1]]

[1] 0.8384952

> performance(prediction.obj = save.pred.elap30,

"auc")@y.values #Only elap30

[[1]]

[1] 0.5601028

The model with only distance performs about as well as the model with all of the variables in it! This helps to illustrate that just because a variable is “significant” does not mean it helps to improve the accuracy of a classification method.

The model with only elap30 in it does a poor job of predicting the correct classifications. In fact, the ROC curve for this model is barely above the line of chance accuracy (if it were drawn).

**Complete separation**

Convergence is not always obtained when estimating a logistic regression model. If you do not obtain convergence, R will print a warning message regarding it. The first thing you should do is try a larger number of iterations for the iterative numerical procedure. This can be done simply by specifying the maxit argument in glm() with a larger number than the default of 25.

If this does not work, there may be some fundamental problems with the data making the iterative numerical procedures not suitable. The most common problem occurs when an independent variable(s) perfectly separates the data between Y = 0 and 1 values; this is often referred to as *complete separation*. In addition to a convergence warning message, another warning message that glm() may give for this situation is

glm.fit: fitted probabilities numerically 0 or 1 occurred

This message corresponds to  = 0 or 1 for some observations. Please see the plot in the next example if it is not clear why this happens.

It is important to note that the previous message of

glm.fit: fitted probabilities numerically 0 or 1 occurred

can be given in situations when there is not a problem. It could be that some estimated probabilities are just very close to 0 or 1. If you are unsure, check the frequency of these estimated probability values.

Example: Complete separation (Nonconvergence.R)

Consider a simple data set with one variable x1 that is less than 6 when y = 0 and greater than or equal to 6 when y = 1. Because x1 perfectly separates out the two possible values of y, complete separation occurs. Below is the corresponding R code and output:

> set1 <- data.frame(x1 = c(1,2,3,4,5,6,7,8,9,10), y =

c(0,0,0,0,0,1,1,1,1,1))

> set1

x1 y

1 1 0

2 2 0

<OUTPUT IS EDITED>

10 10 1

> mod.fit1 <- glm(formula = y ~ x1, data = set1, family = binomial(link = logit), trace = TRUE)

Deviance = 4.270292 Iterations - 1

Deviance = 2.574098 Iterations - 2

<OUTPUT IS EDITED>

Deviance = 7.864767e-10 Iterations - 25

Warning messages:

1: glm.fit: algorithm did not converge

2: glm.fit: fitted probabilities numerically 0 or 1 occurred

> mod.fit1$coefficients

(Intercept) x1

-245.84732 44.69951

R indicates that both convergence did not occur and at least some estimates of π are 0 or 1. Below is a plot of the data and the model at iteration #25 (left side):



Because there is a separation between the Y = 0 and 1 values, the slope of the line between x = 5 and 6 will continue to get larger as the iterations continue. Essentially, the β1 estimate is going to infinity with continued iterations. Notice this means the estimate is VERY biased.

Interestingly, R indicates “convergence” after 26 iterations if you increase maxit! However, the same

glm.fit: fitted probabilities numerically 0 or 1 occurred

message will occur. In fact, you can continue for a larger number of iterations and see the parameter estimates (values) will continue to change. One should not use logistic regression here because the parameter estimates will continue to change for a larger number of iterations.

To show the non-convergence, one also needs to change the epsilon argument value of glm(). This argument essentially controls how close the  values need to be from one iteration to the next to declare convergence. Smaller values of epsilon indicate that one needs  values to be closer. The default is epsilon = 10^(-8). My categorical data analysis course notes provide more details.

By reversing the Y values at x1 = 5 and 6, we obtain model convergence in 6 iterations (not shown here). The right plot above shows the data and the final model. The slope of the model is now not as great as it was before.

What can you do if complete separation occurs?

The most desirable options are

* Exact logistic regression – The exact distribution of the parameter estimators is used through the use of computational intensive algorithms.
* Modify the likelihood function – Because the likelihood function increases without bound during the iterative numerical procedure, this function can be modified to potentially prevent the problems from happening. One approach is to include a “penalty” in the likelihood function.

Additional comments:

* Complete separation is not necessarily bad because this says that one can obtain perfect classification accuracy. The problem is that the model estimated by maximum likelihood does not provide a good way to interpret the relationship between the response variable and the independent variables.
* It can be difficult to see complete separation graphically if there is more than one independent variable. And, there may be times even when the glm() function does not provide a warning. When parameter estimates are very large or very small with large estimated standard deviations, this is a sign that complete separation may exist. These types of parameter estimates can then lead to many observations with estimated probabilities of success close to 0 or 1.

**Additional considerations for logistic regression**

* There has been a lot of research on ROC curves, and we could spend a lot more time discussing them! For example, one can plot a confidence interval band around the ROC curve.
* Variable selection can be performed by standard methods as when working with regression models. For example, the “best” model can be thought of as the one with the smallest Akaike’s information criteria (AIC). However, this does not address the classification accuracy of the model.
* In addition to sensitivity and specificity, measures of accuracy often of interest are the positive predictive value (PPV) and the negative predictive value (NPV). In the context of the HIV example, the PPV is equal to



The NPV is equal to



Thus, we essentially are looking at P(B|A) now rather than P(A|B) as we were with the HIV example. To help emphasize the importance of these measures, suppose you took a test for a disease and your doctor has said that you tested positive. Which of these measures would be most important to you?

* When you are done trying all of classification methods, one should plot the accuracy levels to help choose between the methods. Below is a plot for the placekicking data set (PlotAccuracy.R):





Which method is best?