**Section 2.2.1 – Parameter estimation**

Maximum likelihood estimation is used to estimate the parameters of the model. As shown previously in the notes, the likelihood function is



but now



in the above expression. Similar to Chapter 1, we can find the log likelihood function:



Taking derivatives with respect to β0, …, βp, setting them equal to 0, and then solving for the parameters lead to the 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.

Appendix B gives a simple example of the Newton-Raphson procedure, one of these iterative numerical procedures, for finding the MLE of π corresponding to Chapter 1 topics.

We will use a procedure called iteratively reweighted least squares (IRLS) to find the maximum likelihood estimates.

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

If the new estimates, say  …, , are very close to  …, , the iterative numerical procedure is said to “converge” and  …,  are used as the MLEs  …, . If the new 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.

The glm() function from the stats package does the computations and finds the MLEs.

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

Example: Placekicking (Placekick.R, Placekick.csv)

This example is motivated by the work that I did for my MS report and Bilder and Loughin (*Chance*, 1998).

The purpose of this and future examples involving the data is to estimate the probability of success for a placekick. Below are the explanatory variables to be considered:

* week: week of the season
* distance: Distance of the placekick in yards
* change: Binary variable denoting lead-change (1) vs. non-lead-change (0) placekicks; successful lead-change placekicks are those that *change* which team is winning the game.
* elap30: Number of minutes remaining before the end of the half with overtime placekicks receiving a value of 0
* PAT: Binary variable denoting the type of placekick where a point after touchdown (PAT) is a 1 and a field goal is a 0
* type: Binary variable denoting dome (0) vs. outdoor (1) placekicks
* field: Binary variable denoting grass (1) vs. artificial turf (0) placekicks
* wind: Binary variable for placekicks attempted in windy conditions (1) vs. non-windy conditions (0); I define windy as a wind stronger than 15 miles per hour at kickoff in an outdoor stadium

The response variable is referred to as good in the data set. It is a 1 for successful placekicks and a 0 for failed placekicks.

There are 1,425 placekick observations from one NFL season that are within this data set. Below is how the data is read into R:

> placekick <- read.csv(file = "C:\\data\\Placekick.csv")

> head(placekick)

week distance change elap30 PAT type field wind good

1 1 21 1 24.7167 0 1 1 0 1

2 1 21 0 15.8500 0 1 1 0 1

3 1 20 0 0.4500 1 1 1 0 1

4 1 28 0 13.5500 0 1 1 0 1

5 1 20 0 21.8667 1 0 0 0 1

6 1 25 0 17.6833 0 0 0 0 1

For now, we are only going to use the distance to estimate the probability of a successful placekick. Thus, our logistic regression model is



where Y is the good response variable and x1 denotes the distance in yards for the placekick. Less formally, we will also write the model as



Below is how we estimate the model with the glm() function:

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

binomial(link = logit), data = placekick)

> mod.fit

Call: glm(formula = good ~ distance, family = binomial(link = logit), data = placekick)

Coefficients:

(Intercept) distance

5.812 -0.115

Degrees of Freedom:1424 Total (i.e. Null); 1423 Residual

Null Deviance: 1013

Residual Deviance:775.7 AIC: 779.7

The estimated logistic regression model is



Note that the function gets its name from “generalized linear model”. This is a general class of linear models which includes logistic regression models. At the end of this chapter, I will formally define this general class.

Question: What happens to the estimated probability of success as the distance increases?

There is actually much more information stored within the mod.fit object than showed so far. Through the use of the names() function, we obtain the following list of items:

> names(mod.fit)

[1] "coefficients" "residuals"

[3] "fitted.values" "effects"

[5] "R" "rank"

[7] "qr" "family"

[9] "linear.predictors" "deviance"

[11] "aic" "null.deviance"

[13] "iter" "weights"

[15] "prior.weights" "df.residual"

[17] "df.null" "y"

[19] "converged" "boundary"

[21] "model" "call"

[23] "formula" "terms"

[25] "data" "offset"

[27] "control" "method"

[29] "contrasts" "xlevels"

> mod.fit$coefficients

(Intercept) distance

5.8120798 -0.1150267

To see a summary of the information in mod.fit, we can use the summary() function:

> summary(object = mod.fit)

Call: glm(formula = good ~ distance, family = binomial(link = logit), data = placekick)

Deviance Residuals:

Min 1Q Median 3Q Max

-2.7441 0.2425 0.2425 0.3801 1.6092

Coefficients:

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

(Intercept) 5.812080 0.326277 17.81 <2e-16 \*\*\*

distance -0.115027 0.008339 -13.79 <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: 1013.43 on 1424 degrees of freedom

Residual deviance: 775.75 on 1423 degrees of freedom

AIC: 779.75

Number of Fisher Scoring iterations: 6

Now is a good time for a reminder of why R is often referred to as an “object oriented language”. As discussed in Appendix A, every object in R has a class associated with it. The classes for mod.fit are:

> class(mod.fit)

[1] "glm" "lm"

Functions are typically designed to operate on only one or very few classes of objects. However, some functions, like summary(), are *generic*, in the sense that essentially different versions of them have been constructed to work with different classes of objects.

When a generic function is run with an object, R first checks the object's class type and then looks to find a *method* function with the name format <generic function>.<class name>. Below are examples for summary():

* summary(mod.fit) – The function summary.glm() summarizes the regression model
* summary(placekick) – The function summary.data.frame() summarizes the data frame’s contents
* summary.default() – R attempts to run this function if there is no method function for a class

Why is R set-up like this?

The purpose of generic functions is to use a familiar language set with any object. For example, we frequently want to summarize data or a model (summary()), plot data (plot()), and find predictions (predict()), so it is convenient to use the same language set no matter the application. This is why R is referred to as an object-oriented language. The object class type determines the function action. Understanding generic functions may be one of the most difficult topics for new R users!

To see a list of all generic functions associated with a class, use methods(class = <class name>). Here are the method functions for the glm class.

> methods(class = glm)

[1] add1 anova coerce

[4] confint cooks.distance deviance

[7] drop1 effects extractAIC

[10] family formula influence

[13] initialize logLik model.frame

[16] nobs predict print

[19] residuals rstandard rstudent

[22] show slotsFromS3 summary

[25] vcov weights

see '?methods' for accessing help and source code

To see a list of all method functions for a generic function, use methods(generic.function = <generic function name>). Below are the method functions associated with summary():

> methods(generic.function = "summary")

[1] summary.aov summary.aovlist\*

[3] summary.aspell\* summary.check\_packages\_in\_dir\*

[5] summary.connection summary.data.frame

[7] summary.Date summary.default

[9] summary.ecdf\* summary.factor

[11] summary.glm summary.infl\*

[13] summary.lm summary.loess\*

[15] summary.manova summary.matrix

[17] summary.mlm\* summary.nls\*

[19] summary.packageStatus\* summary.POSIXct

[21] summary.POSIXlt summary.ppr\*

[23] summary.prcomp\* summary.princomp\*

[25] summary.proc\_time summary.sockclientconn\*

[27] summary.srcfile summary.srcref

[29] summary.stepfun summary.stl\*

[31] summary.table summary.tukeysmooth\*

[33] summary.warnings

see '?methods' for accessing help and source code

We can find the estimated probability of success for a particular distance using



For example, the probability of success at a distance of 20 is 0.97:

> linear.pred <- mod.fit$coefficients[1] + mod.fit$coefficients[2]\*20

> linear.pred

(Intercept)

3.511547

> exp(linear.pred)/(1 + exp(linear.pred))

(Intercept)

0.9710145

> # Remove label

> as.numeric(exp(linear.pred)/(1 + exp(linear.pred)))

[1] 0.9710145

The estimated probability of success for a distance of 50 yards is 0.52 (code not in program):

> linear.pred <- mod.fit$coefficients[1] +

mod.fit$coefficients[2]\*50

> as.numeric(exp(linear.pred)/(1+exp(linear.pred)))

[1] 0.515182

Using this method to estimate the probability of success, we can now plot the model with the curve() function:

> curve(expr = exp(mod.fit$coefficients[1] +

mod.fit$coefficients[2]\*x) / (1 + exp(mod.fit$coefficients[1] + mod.fit$coefficients[2]\*x)), col = "red", xlim = c(18,

66), ylab = expression(hat(pi)), xlab = "Distance",

main = "Estimated probability of success for a

placekick", panel.first = grid())



If more than one explanatory variable is included in the model, the variable names can be separated by “+” symbols in the formula argument of glm(). For example, suppose we include change in addition to distance in the model:

> mod.fit2 <- glm(formula = good ~ change + distance, family = binomial(link = logit), data = placekick)

> summary(mod.fit2)

Call: glm(formula = good ~ change + distance, family = binomial(link = logit), data = placekick)

Deviance Residuals:

Min 1Q Median 3Q Max

-2.7061 0.2282 0.2282 0.3750 1.5649

Coefficients:

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

(Intercept) 5.893181 0.333184 17.687 <2e-16 \*\*\*

change -0.447783 0.193673 -2.312 0.0208 \*

distance -0.112889 0.008444 -13.370 <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: 1013.4 on 1424 degrees of freedom

Residual deviance: 770.5 on 1422 degrees of freedom

AIC: 776.5

Number of Fisher Scoring iterations: 6

> mod.fit2$coefficients

(Intercept) change distance

5.8931814 -0.4477832 -0.1128888

The estimated logistic regression model is



While we use the glm() function to find our MLEs for convenience, one could also find these estimates by programming in the log likelihood function and maximizing it ourselves. This is especially useful to know how for other “non-standard” problems that may occur in problems faced outside of this class. Please see my book for an example of how to find the MLEs for the  model without using glm(). The code uses a VERY nice function called optim(), which performs a number of different iterative numerical procedures. I have found this function to be very useful in my research!