**STAT 414 Practical**

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

As discussed in Chapter 1, points can be scored in American football by a placekicker

kicking a ball through a target area at an end of the field. A success occurs when the

football is kicked over the crossbar and between the two uprights of the goal posts.

The placekicker’s team receives 1 or 3 points, where a point after touchdown (PAT)

receives 1 point and a field goal receives 3 points. A placekick that is not successful

receives 0 points. Bilder and Loughin (1998) use logistic regression to estimate the probability of

success for a placekick (PAT or field goal) in the National Football League (NFL).

They examine a number of explanatory variables, including:

* 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;
* lead-changing placekicks are those that have the potential to change which

team is winning the game (for example, if a field goal is attempted by a team that

is losing by 3 points or less, they will no longer be losing if the kick is successful)

* 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 PAT attempt is a 1

and a field goal attempt is a 0

* type: Binary variable denoting outdoor (1) vs. dome (0) 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); we define windy as a wind stronger than 15 miles per hour at

kickoff in an outdoor stadium

* good: Binary variable denoting successful (1) vs. failed (0) placekicks; this is our

response variable

There are 1,425 placekick observations collected during the 1995 NFL season that are available in the data set. Below is how the data are read into R (see Appendix A.7.1 for more information on reading data into R):

R codes

1. placekick <- read . table ( file = "C :\\ data \\ Placekick . csv", header = TRUE , sep = " ,")

The data are saved in a comma-delimited (.csv) format, so we use the read.table() function to read it in. The file argument specifies where the file is located on our computer. Note that double back slashes, not single back slashes, are required to separate folders and file names. The header = TRUE argument value instructs R to use the names in the first row of the file as the variable names. The sep = "," argument value states that the file is comma-delimited. After reading the data set into R, it is saved into a data frame called placekick, and the first six observations are printed using the head() function.

1. head ( placekick )
2. mod.fit <- glm( formula = good ~ distance , family =binomial ( link = logit ), data = placekick )

* The results from glm() are saved into an object that we call mod.fit, which is a

shortened version of “model fit” (we could choose a different name). The arguments

within glm() are:

* formula — Specifies the model with a ~ separating the response and explanatory

variables

* family — Gives the type of model to be fit where binomial states the response

type (remember that Bernoulli is a special case of the binomial) and logit is the

function on the left side of the model (we will discuss “link” functions in Section 2.3)

* data — Name of the data frame containing the variables

1. mod.fit

Additional information is printed in the output, but there are actually many more items, called components, that are stored within the mod.fit object. Through the use of the names() function, we obtain the following list of these components:

1. names (mod.fit )

A summary of what is inside mod.fit is obtained from the summary() function:

1. summary (mod.fit )

The class of the mod.fit object is glm. Associated with this classare a number of method functions that help to summarize the information within objects or that complete additional useful calculations. To see a list of these method functions, the methods() function is used:

1. class (mod.fit )

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

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

The estimated variance-covariance matrix (hereafter referred to as a “covariance matrix”

for simplicity) for a vector of parameter estimates and can be found in R using the vcov() function. This matrix is found by inverting a matrix of the second partial derivatives (often referred to as a “Hessian matrix”) of the log-likelihood function evaluated at the parameter estimates and multiplying this resulting matrix by -1. This covariance matrix has the same form as the one resulting from weighted least squares estimation in a linear model. The individual elements of the matrix do not have a simple form, so we do not present them here. Below is the coefficients table again from the summary(mod.fit) output:

1. round ( summary (mod. fit) $coefficients ,4)

The purpose of this example is to show how to obtain the estimated covariancematrix for the regression parameter estimators. We use the mod.fit object from the model with distance as the only explanatory variable.

1. vcov (mod.fit )

We can extract the estimated variance for beta\_1 by specifying the (2,2) element of the matrix.

1. vcov (mod.fit )[2 ,2] # Var -hat (beta - hat\_1 )
2. summary (mod.fit ) $coefficients [2 ,2]^2

For readers interested in the actual matrix calculations, we provide an example of how the covariance matrix is calculated

14. pi.hat <- mod. fit$fitted . values

15. V <- diag (pi. hat \*(1 - pi.hat))

1. X <- cbind (1, placekick$distance )

We create the diagonal matrix V by using the diag() function, where we first extract pi\_1 is from mod.fit. Next, we create the X matrix by using the cbind() function which puts a column of 1’s before a column of the explanatory variable values. Continuing, the t() function finds the transpose of the X matrix, the %\*% syntax instructs R to perform matrix multiplication, and the solve() function finds the inverse of the matrix. The resulting matrix is practically the same as that produced by vcov(), where differences are due to rounding error.

We focus again on fitting the logistic regression model with only the distance

explanatory variable. Below we create a function logL() to calculate the log-likelihood

function for any given parameter values, explanatory variable values for x1, and binary

responses for Y :

1. logL <- function (beta , x, Y) { pi <- exp( beta [1] + beta [2]\* x)/(1 + exp( beta [1] + beta [2]\* x))

sum (Y\*log(pi) + (1-Y)\*log (1- pi)) }

1. logL ( beta = mod. fit$coefficients , x = placekick$distance , Y =placekick$good )
2. logLik (mod.fit )

To maximize the log-likelihood function and find the corresponding MLEs, we use

the optim() function. This is a very general optimization function that minimizes

an R function with respect to a vector of parameters. Because we want to maximize

the log-likelihood function instead, we use the control = list(fnscale = -1) argument

value within optim(), which essentially instructs R to minimize the negative

of logL(). Below is our code:

# Find starting values for parameter estimates

1. reg.mod <- lm( formula = good ~ distance , data = placekick )
2. reg. mod$coefficients
3. mod.fit. optim <- optim (par = reg. mod$coefficients , fn = logL , hessian = TRUE , x = placekick$distance , Y = placekick$good , control = list ( fnscale = -1) , method = " BFGS ")
4. names (mod.fit . optim )
5. mod.fit. optim$par
6. mod.fit. optim$value
7. mod.fit. optim$convergence
8. -solve (mod.fit . optim$hessian )

The optimization procedures within optim() need initial starting values for the regression

parameters. We simply fit a linear regression model to the data using the lm() function and take the corresponding parameter estimates as these starting values (see Appendix A.7 for another example of using lm()). Within the call to optim(), we specify the initial parameter estimates using the par argument, and we specify the function to be maximized using the fn argument. Note that the first argument in the function named in fn must correspond to the initial parameter estimates; this is why beta was given as the first argument in logL(). The TRUE value for the hessian argument instructs R to obtain a numerical estimate of the Hessian matrix for the parameters. In other words, it estimates a matrix of second partial derivatives for the log-likelihood function, which can then be inverted to obtain the estimated covariance matrix for the parameter estimates (see Appendix B.3.4).3 Finally, we specify the corresponding values of x and Y for the logL() function. The object produced by optim() is a list that we save as mod.fit.optim. Components within the list include the parameter estimates ($par), the log-likelihood function’s maximum value ($value), and the estimated covariance matrix ($hessian; we subsequently use solve() to find the estimated covariance matrix as the inverse of the Hessian matrix). All of these values are practically the same as those produced by glm(), where small differences are due to different convergence criteria for the iterative numerical procedures. The 0 value for mod.fit.optim$convergence means that

convergence was achieved. For this implementation of optim(), we used the method = "BFGS" optimization procedure (similar to a Newton-Raphson procedure), but other procedures are available within the function (see Gentle, 2009 or the help for optim() for details on various iterative numerical procedures).

The purpose of this example is to show how to reform the placekicking data to a binomial response format and to show that the model parameter estimates are the same for this format. We focus only on the distance explanatory variable for this example. The aggregate() function is used to find the number of successes and number of observations for each distance:

1. w <- aggregate ( formula = good ~ distance , data = placekick , FUN= sum )
2. n <- aggregate ( formula = good ~ distance , data = placekick , FUN= length )
3. w.n <- data . frame ( distance = w$distance , success = w$good , trials = n$good , proportion = round ( w$good /n$good ,4) )
4. head (w.n)

The formula argument within aggregate() is used the same way as within glm(). The FUN argument specifies how the response variable good will be summarized, where the sum() function adds the 0 and 1 responses for each distance and the length() function counts the number of observations for each distance. The final result is in the w.n data frame. For example, there are 2 successes out of 3 trials at a distance of 18 yards, which results in an observed proportion of successes of 2=3 \_ 0:6667. Note that the reason for the large number of observations at 20 yards is because most PATs are attempted from this distance. The logistic regression model is estimated by the glm() function with two changes to our previous code:

1. mod.fit.bin <- glm( formula = success / trials ~ distance , weights= trials , family = binomial ( link = logit ), data = w.n)
2. summary (mod.fit .bin)