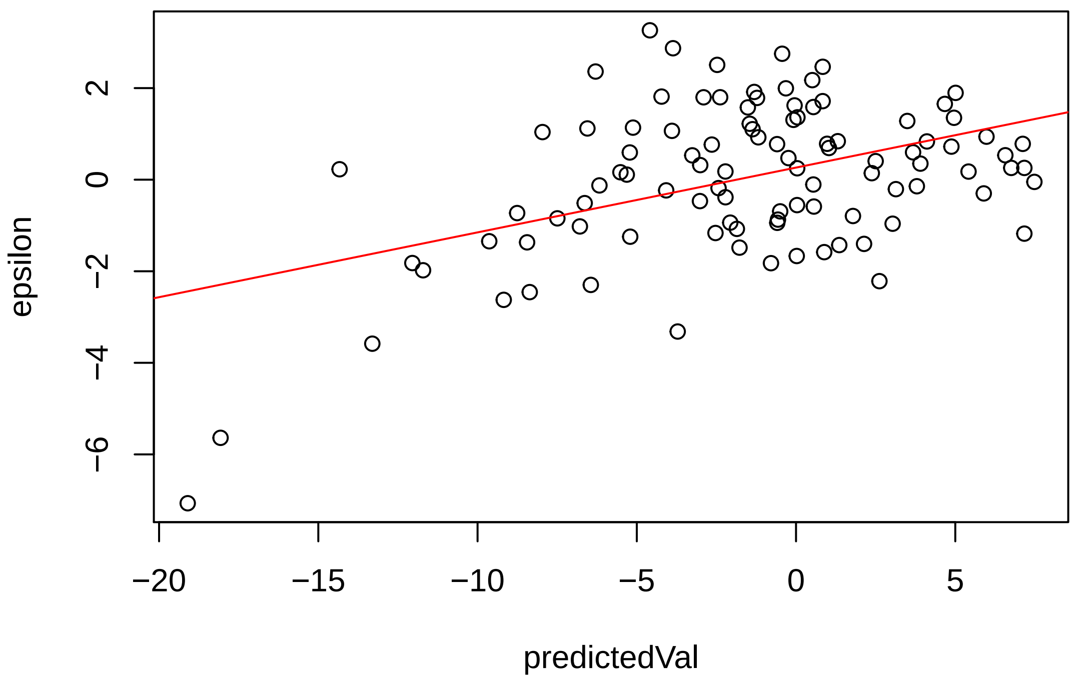
W203_Homework_14_I

Natarajan Shankar

August 9, 2016

```r
library(MASS); library(gvlma); library(sandwich) ; library(stats)
suppressMessages(library(lmtest)); suppressMessages(library(boot))
suppressMessages(library(QuantPsyc))

# Set a seed for repeatable results, when needed
set.seed(123456)

# Select number of data points, a general setting
N <- 100

# Creating independent variables
# first, the covariance matrix for our multi-variate normal
# This determines the correlation of our independent variables
sig <- matrix(c(2, .5, .25, .5, 1, 0, .25, 0, 1), nrow=3)

# Now the variables:
predictorVar = as.data.frame(mvrnorm(n = N, mu = rep(1,3), Sigma = sig))

# For convenience, name the predictor variables
# THe third will not be used in this assignment but leave it in there
colnames(predictorVar) <- c("X1", "X2", "X3")

# Set the coefficients as per the homework handout
b0 = 1 ; b1 = 2 ; b2 = -5

# The function that helps create heteroskedasticity
hskdTrigger <- function(x) {
  # use one of the predictor variables
  # Reference: StackForge
  1.0 + 0.4*predictorVar$X2
}

# Use the function that helps create heteroskedasticity to generate data
# Suppress warnings that call out presence of NaN values.
epsilon = suppressWarnings(rnorm(N, 0, hskdTrigger(x)))

# Implement the linear predictor equation, use the error term
predictedVal = b0 + b1*predictorVar$X1 + b2 * predictorVar$X2 + epsilon

# Save an X dataframe for part 2 of this asignment
X <- predictorVar
X$X3 <- NULL # remove the X3 column, we will not be using it

# For running models, keep the Y data together with the predictor data
predictorVar <- cbind(predictedVal, predictorVar)

# Ensure that the generated graph indicates heteroskedasticity
# Plot residuals against ptedicted values
plot(predictedVal, epsilon)

# Using a Least Squares fit, draw a reference line so that a
# general orientation of slope is available on the plot
abline(lsfit(predictedVal, epsilon), col = "red")
```



```r
# Next, fit a linear model to the data
modellm <- lm(predictedVal ~ X1 + X2 , data = predictorVar)

# test for Heterskedsticity by running  the Breusch Pagan test
bpresults <- bptest(modellm)
bpresults
```

```
##
##  studentized Breusch-Pagan test
##
## data:  modellm
## BP = 15.955, df = 2, p-value = 0.0003431
```

The Breusch Pagan test confirms that based upon the statistical confidence of `3.43094\times 10^{-4}`, the Null hypothesis of Homoskedasticity can be rejected

```r
# Look at the data from the lm model
summary(modellm)
```

```
##
## Call:
```

```
## lm(formula = predictedVal ~ X1 + X2, data = predictorVar)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.5695 -1.0107  0.1028  1.2227  3.5516
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.1878     0.2361   5.032 2.23e-06 ***
## X1            2.0974     0.1328  15.794  < 2e-16 ***
## X2           -5.2665     0.1754 -30.033  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.673 on 97 degrees of freedom
## Multiple R-squared:  0.904,  Adjusted R-squared:  0.902
## F-statistic: 456.6 on 2 and 97 DF,  p-value: < 2.2e-16
```

The lm model fits the data very well as shown by the adjusted R-squared value. 90% of the Response variable can be explained by the results of the LM model

```r
# Check confidence intervals of the coefficients
# Get the standard beta values
lm.beta(modellm)
```

```
##         X1         X2
##  0.5489519 -1.0438748
```

Coefficient of X1 contributes in positive manner to the Response variable, The coefficient of X2 contributes in negative manner to the response variable

```r
# Perform a coeftest to get the robust standard errors
coeftest(modellm)
```

```
##
## t test of coefficients:
##
##             Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)  1.18779    0.23606   5.0317 2.235e-06 ***
## X1           2.09741    0.13280  15.7939 < 2.2e-16 ***
## X2          -5.26645    0.17535 -30.0334 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Set up the Bootstrap function
# Mimic code from the textbook
bootReg <- function(formula, data, indices){
  d<- data[indices,]
  fit <- lm(formula, data = d)
  return(coef(fit))
}

# Use bootstrap to estimate the model, use 2000 samples as recommended in the text
bootResults <- boot(statistic = bootReg,
              formula =  predictedVal ~ X1 + X2, data = predictorVar, R = 2000)
# Summarize the results of bootstrap
bootResults
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = predictorVar, statistic = bootReg, R = 2000, formula = predictedVal ~
##     X1 + X2)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1*  1.187794  0.0017920906   0.1904614
## t2*  2.097409 -0.0005449895   0.1609334
## t3* -5.266454  0.0091632382   0.2374794
```

Bootstrap: Coeffficients of approximately 1.xx, 2.xx and -5.xx are almost the same as in the LM model.

```r
# Obtain bootstrap confidence levels for the Intercept and the 2 other Coefficients
ci_1 <- boot.ci(bootResults, type="bca", index=1)
ci_1$bca[4:5]
```

```
## [1] 0.8129949 1.5494954
```

```r
ci_2 <- boot.ci(bootResults, type="bca", index=2)
ci_2$bca[4:5]
```

```
## [1] 1.852572 2.518470
```

```r
ci_3 <- boot.ci(bootResults, type="bca", index=3)
ci_3$bca[4:5]
```

```
## [1] -5.816809 -4.869990
```

```r
# Look at confidence intervals from the LM model
confint(modellm)
```

```
##                  2.5 %    97.5 %
## (Intercept)  0.7192719  1.656317
## X1           1.8338406  2.360976
## X2          -5.6144812 -4.918426
```

Bootstrap: Confidence levels seen with Bootstrap are close in value to those seen with the LM model. The data distributional requirements are well compensated for by the bootstrap method

```r
# Run the coeftest test to get Robust Standard Errors
coeftest(modellm, vcov = vcovHC)
```

```
## 
## t test of coefficients:
## 
##             Estimate Std. Error  t value  Pr(>|t|)    
## (Intercept)  1.18779    0.19060   6.2318 1.192e-08 ***
## X1           2.09741    0.17065  12.2906 < 2.2e-16 ***
## X2          -5.26645    0.25111 -20.9724 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sandwich/Coeftest: Coeffficients b0, b1, b2 are associated with a very high level of statistical significance

```r
vcovHC(modellm, type = "HC")
```

```
##              (Intercept)          X1           X2
## (Intercept)  0.033151898 -0.01277295 -0.005509922
## X1          -0.012772949  0.02551868 -0.023505140
## X2          -0.005509922 -0.02350514  0.055586039
```

The heteroskedasticity consistent covariance matrix shows very low covariance amongst the coefficients

```r
# Extract the R^2 values
sandwich_se <- diag(vcovHC(modellm, type = "HC"))^0.5
sandwich_se
```

```
## (Intercept)          X1          X2 
##   0.1820766   0.1597457   0.2357669
```

The LM model shows a high correlation towards the Response variable

Part 2

```r
# Generate data
# Assume that data from part 1 of this assignment form observations from the world
# Generate Y = 1 when predictedVal is positive and Y = 0 when predictedVal is negative
binomY <- ifelse(predictedVal < 0, 0, 1)

# Restore predictorVar from an earlier version of predictors
# We need a clean set up so that matrix multiplication can be done
predictorVar <- X

# Generate the complete data for Part 2 of this assignment, call it dataDF
dataDF <- cbind(binomY, predictedVal, predictorVar)
```

What is the likelihood that y_i is 1?

$$p^{y_i} * (1-p)^{(1-y_i)}$$

and when

$$y_i = 1$$

the likelihood that y_i = 1 is

$$p$$

What is the likelihood that y_i is 0?

$$p^{y_i} * (1-p)^{(1-y_i)}$$

and when

$$y_i = 0$$

the likelihood that y_i = 0 is

$$(1-p)$$

What is the sum of log likelihoods of observing all of y

$$LL = log[p_i^y * (1-p)^{(1-y_i)}]$$

$$LL = log[p_i^y] + log[(1-p)^{(1-y_i)}]$$

$$LL = y_i * log[p] + (1-y_i) * log[1-p]$$

now, given that

$$p = \frac{1}{1 + exp(-X * b)}$$

$$LL = -y_i * log[1 + exp(-X * b)] - (1 - y_i) * log[1 + exp(-X * b)]$$

The above equation is implemented in logit below

```
logit <- function(y, x, b)
{
  sum_loglh <- sum(-y*log(1 + exp(-(x%*%b))) - (1-y)*log(1 + exp(x%*%b)))
  return(abs(sum_loglh))
}

# Convert Y and X into matrices
Y <- as.matrix(binomY)
X <- as.matrix(predictorVar)

# Convert the coefficients into a matrix as well
b=as.matrix(c(b0, b1, b2))
X <- cbind(1, X) # get the constant term factored in


# Call the logit function with Y, x and b
logit(Y, X, b)
```

```
## [1] 20.42734
```

```
# Perform the numerical optimization using the optim() function
optim_result <- optim(b,fn=logit,y=Y,x=X, method='BFGS',hessian=TRUE)

# Check convergence to make sure that optim() did converge
optim_result$convergence
```

```
## [1] 0
```

```
# Check the parameters, report the b values
optim_result$par
```

```
##           [,1]
## [1,]  2.314090
## [2,]  2.167034
## [3,] -6.864022
```

```
# Per handout notes, get the standard error for each parameter
solve(optim_result$hessian)
```

```
##             [,1]       [,2]      [,3]
## [1,]  0.6757223  0.3106795 -1.174791
## [2,]  0.3106795  0.4098419 -1.012947
## [3,] -1.1747912 -1.0129474  3.136121
```

```
# Look for standard error of each parameter
s.e. <- sqrt(diag(solve(optim_result$hessian)))
s.e.
```

```
## [1] 0.8220233 0.6401890 1.7709095
```

```
# Compare with glm
glmData <- cbind(binomY, predictorVar)
glm_result <- glm(binomY ~ X1 + X2, data = glmData, family = binomial)
summary(glm_result)
```

```
##
## Call:
## glm(formula = binomY ~ X1 + X2, family = binomial, data = glmData)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.51968  -0.16717  -0.00875   0.11061   2.27813
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.3141     0.8220   2.815 0.004875 **
## X1            2.1669     0.6402   3.385 0.000712 ***
## X2           -6.8638     1.7708  -3.876 0.000106 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 134.602  on 99  degrees of freedom
## Residual deviance:  36.015  on 97  degrees of freedom
## AIC: 42.015
##
## Number of Fisher Scoring iterations: 8
```

Results from glm show coeffoicients that match the match the coefficients derived via optim()