HW5 - DATA 609

Thomas Hill

October 31, 2021

```
library(knitr)
```

Ex. 1 – Carry out the logistic regression (Example 22 on Page 94) in R using the data

x	0.1	0.5	1.5	2	2.5
y	0.0	1.0	1.0	1	0.0

The formula is:

$$y(x) = \frac{1}{1+e^{-(a+bx)}}$$

```
p_i <- function(a_i = 1, b_i = 1, x_val) {
  fxn <- 1/(1+exp(-1*(a_i + b_i*(x_val)))) #formula
  return(fxn)
}

p_i(x_val = x)
```

```
## [1] 0.7502601 0.8175745 0.9241418 0.9525741 0.9706878
```

Above are the values for P_i according to binary logistic regression

```
log_lik <- function(x_i, y_i) {
  p_xi <- p_i(x_val = x_i)
  fxn <- y_i * log(p_xi) + (1-y_i) * log(1-p_xi)
  return(fxn)
}

log_lik(x_i = x ,y_i = y) #L_i
```

```
## [1] -1.38733533 -0.20141328 -0.07888973 -0.04858735 -3.52975042
```

```
sum(log_lik(x_i = x ,y_i = y)) #log-likelihood objective, assuming a_i = b_i = 1
```

```
## [1] -5.245976
```

Additionally, here is L_i and log-likelihood objective.

Using the equation that logistic function $S(x)$ has derivative $S'(x) = S(x)(1 - S(x))$, the Jacobian can quickly be derived as:

$$\begin{bmatrix} S(x)(1-S(x)) \\ -x*S(x)(1-S(x)) \end{bmatrix}$$

Using code from the previous module to carry out the Newton method for estimating parameters.

```
a_0 <- 1
b_0 <- 1
initial_params <- matrix(a_0,b_0, nrow = 2)


drda <- p_i(x_val = x) * (1 - p_i(x_val = x))
drdb <- -x * p_i(x_val = x) * (1 - p_i(x_val = x))

ex1_j <- cbind(drda,drdb) #initial jacobian
print(ex1_j)
```

```
##            drda        drdb
## [1,] 0.18736988 -0.01873699
## [2,] 0.14914645 -0.07457323
## [3,] 0.07010372 -0.10515557
## [4,] 0.04517666 -0.09035332
## [5,] 0.02845302 -0.07113256
```

```
ex1_resid <- matrix(y - p_i(x_val = x)) #initial residuals
print(ex1_resid)
```

```
##             [,1]
## [1,] -0.75026011
## [2,]  0.18242552
## [3,]  0.07585818
## [4,]  0.04742587
## [5,] -0.97068777
```

Where $\frac{dR}{da} = -\frac{e^{-(a+bx)}}{(1+e^(-(a+bx)))^2}$ and $\frac{dR}{da} = -\frac{xe^{-(a+bx)}}{(1+e^(-(a+bx)))^2}$

```
initial_input <- list(initial_params, ex1_resid, ex1_j)
ex1_solve <- function(list_input = initial_input) {
  params <- list_input[[1]]
  R <- list_input[[2]]
  J <- list_input[[3]]


  newton <- params - solve(t(J) %*% J) %*% t(J) %*% R #gauss-newton algorithm
  ex1_solve.result <- newton #t + 1 iteration of parameter estimates
  p_xhat <- p_i(a_i = ex1_solve.result[1], b_i = ex1_solve.result[2], x_val = x) #calculate new
 S(x) with new parameters

  ex1_solve.resid <- y - p_xhat #calculate new residual
  ex1_solve.drda <-  p_xhat * (1 - p_xhat) #update parameters in dRda
  ex1_solve.drdb <- -x * p_xhat * (1 - p_xhat) #update parameters in dRdb
  ex1_solve.jacobian <- cbind(ex1_solve.drda,ex1_solve.drdb) #recalculate jacobian using above d
erivatives
    return(list(ex1_solve.result,ex1_solve.resid, ex1_solve.jacobian)) #return list to allow for
multiple iterations
}


first_iteration <- ex1_solve()
print(first_iteration)
```

```
## [[1]]
##          [,1]
## drda 3.060204
## drdb 1.022345
##
## [[2]]
## [1] -0.95939605  0.02734817  0.01001386  0.00603040 -0.99637426
##
## [[3]]
##      ex1_solve.drda ex1_solve.drdb
## [1,]    0.038955271   -0.003895527
## [2,]    0.026600252   -0.013300126
## [3,]    0.009913581   -0.014870372
## [4,]    0.005994035   -0.011988069
## [5,]    0.003612591   -0.009031477
```

After trying several iterations of this, my build of the algorithm does not find adequate parameters for minimizing the sum of the residuals. Next, I'll try using the *glm* function built into R.

```
ex1_glm <- glm(y ~ x, family = binomial(link = 'logit'))
print(ex1_glm)
```

```
##
## Call:  glm(formula = y ~ x, family = binomial(link = "logit"))
##
## Coefficients:
## (Intercept)            x
##     0.35127      0.04116
##
## Degrees of Freedom: 4 Total (i.e. Null);  3 Residual
## Null Deviance:       6.73
## Residual Deviance: 6.728     AIC: 10.73
```

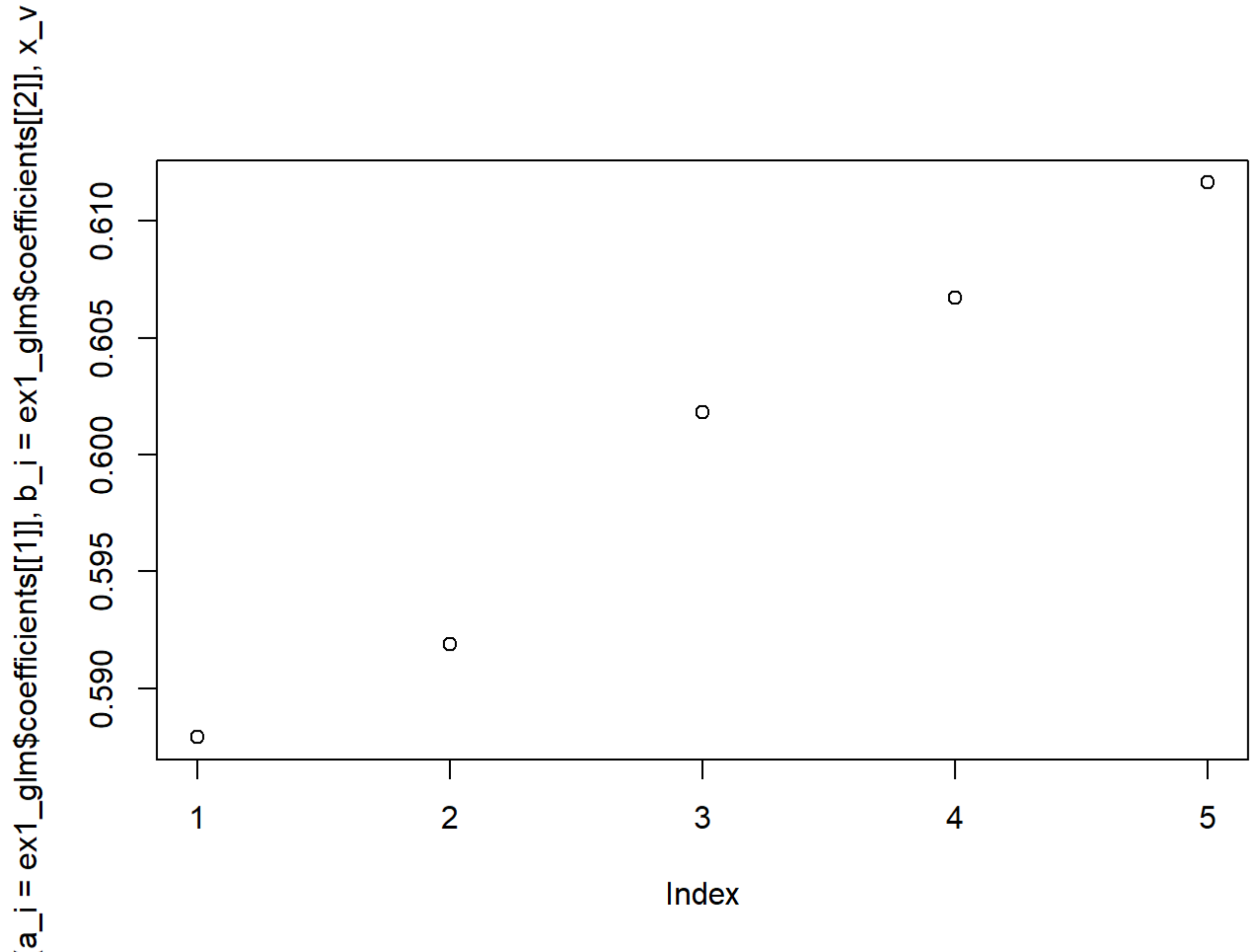
```
log_lik_2 <- function(x_i, y_i) {
  p_xi <- p_i(a_i = ex1_glm$coefficients[[1]], b_i = ex1_glm$coefficients[[2]], x_val = x_i)
  fxn <- y_i * log(p_xi) + (1-y_i) * log(1-p_xi)
  return(fxn)
}

log_lik_2(x, y)
```

```
## [1] -0.8865435 -0.5244083 -0.5078140 -0.4996694 -0.9457999
```

```
sum(log_lik_2(x_i = x, y_i = y))
```

```
## [1] -3.364235
```

```
plot(p_i(a_i = ex1_glm$coefficients[[1]], b_i = ex1_glm$coefficients[[2]], x_val = x))
```



The parameters of $a = 0.35127, b = 0.04116$ appear to maximize the sum log-likelihood around approximately -3.364.

Ex. 2 – Using the motor car database (mtcars) of the built-in datasets in R, carry out the basic principal component analysis and explain your result.

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
summary(prcomp(mtcars, scale = TRUE))
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6    PC7
## Standard deviation     2.5707 1.6280 0.79196 0.51923 0.47271 0.46000 0.3678
## Proportion of Variance 0.6008 0.2409 0.05702 0.02451 0.02031 0.01924 0.0123
## Cumulative Proportion  0.6008 0.8417 0.89873 0.92324 0.94356 0.96279 0.9751
##                            PC8    PC9    PC10   PC11
## Standard deviation     0.35057 0.2776 0.22811 0.1485
## Proportion of Variance 0.01117 0.0070 0.00473 0.0020
## Cumulative Proportion  0.98626 0.9933 0.99800 1.0000
```

mtcars is a dataframe that contains characteristics of 32 different cars and their impact on several indicators of vehicle performance. Using the *prcomp* function returns two major lists: the first shows the contribution to to variance that each principal component adds. The 11 components are ranked by size of the standard deviation, with the first component being the largest. The easiest way to interpret the importance of the component is looking at the proportion of variance it explains. For instance, the first component explains 60% of variance between car features. Beyond this, we can also see that the first five components explain nearly 95% of variance. This is is a promising finding, as it means of the 11 original variables, there are some that could be omitted in a preliminary model to explain the relationship between car design and performance. Next, lets look what the components are made of.

```
prcomp(mtcars, scale = TRUE)
```

```
## Standard deviations (1, .., p=11):
##  [1] 2.5706809 1.6280258 0.7919579 0.5192277 0.4727061 0.4599958 0.3677798
##  [8] 0.3505730 0.2775728 0.2281128 0.1484736
##
## Rotation (n x k) = (11 x 11):
##             PC1         PC2         PC3          PC4         PC5         PC6
## mpg  -0.3625305  0.01612440 -0.22574419 -0.022540255  0.10284468 -0.10879743
## cyl   0.3739160  0.04374371 -0.17531118 -0.002591838  0.05848381  0.16855369
## disp  0.3681852 -0.04932413 -0.06148414  0.256607885  0.39399530 -0.33616451
## hp    0.3300569  0.24878402  0.14001476 -0.067676157  0.54004744  0.07143563
## drat -0.2941514  0.27469408  0.16118879  0.854828743  0.07732727  0.24449705
## wt    0.3461033 -0.14303825  0.34181851  0.245899314 -0.07502912 -0.46493964
## qsec -0.2004563 -0.46337482  0.40316904  0.068076532 -0.16466591 -0.33048032
## vs   -0.3065113 -0.23164699  0.42881517 -0.214848616  0.59953955  0.19401702
## am   -0.2349429  0.42941765 -0.20576657 -0.030462908  0.08978128 -0.57081745
## gear -0.2069162  0.46234863  0.28977993 -0.264690521  0.04832960 -0.24356284
## carb  0.2140177  0.41357106  0.52854459 -0.126789179 -0.36131875  0.18352168
##               PC7          PC8          PC9        PC10         PC11
## mpg   0.367723810 -0.754091423  0.235701617  0.13928524 -0.124895628
## cyl   0.057277736 -0.230824925  0.054035270 -0.84641949 -0.140695441
## disp  0.214303077  0.001142134  0.198427848  0.04937979  0.660606481
## hp   -0.001495989 -0.222358441 -0.575830072  0.24782351 -0.256492062
## drat  0.021119857  0.032193501 -0.046901228 -0.10149369 -0.039530246
## wt   -0.020668302 -0.008571929  0.359498251  0.09439426 -0.567448697
## qsec  0.050010522 -0.231840021 -0.528377185 -0.27067295  0.181361780
## vs   -0.265780836  0.025935128  0.358582624 -0.15903909  0.008414634
## am   -0.587305101 -0.059746952 -0.047403982 -0.17778541  0.029823537
## gear  0.605097617  0.336150240 -0.001735039 -0.21382515 -0.053507085
## carb -0.174603192 -0.395629107  0.170640677  0.07225950  0.319594676
```

This next call to *prcomp* offers a look at what each component is composed of. The first component relates many of the features together in correlations that would make sense in the real world. For example, miles per gallon (mpg) and weight (wt) are opposite signs, while number of cylinders (cyl) and engine size (disp) are same signs. Also notable to pick out are these variables have the largest weights relative to other features. In the second variable, note that mpg and cyl contribute far less rotation, with coefficients of 0.016 and 0.043. Instead, PC2 offers relationships between the complexity of the engine - transmission (am), gears, and carburetors (carb) are the largest contributors, while quarter mile time (qsec) is a large magnitude in the other direction.

Ex. 3 – Generate a random 4 x 5 matrix, and find its singular value decomposition using R.

```
random_matrix <- matrix(rnorm(20),nrow=4)

ex3_svd <- svd(random_matrix)
```

The *svd* function in R returns the three matrices that comprise the decomposition: for the starting matrix A, the function returns D, the diagonal matrix, as well as U and V, the left and right orthonormal matrices. They have the following equivalence:

$$A = UDV^T$$

We can confirm this by performing matrix math in R

```
D <-diag(ex3_svd$d) #make vector diagonal
U <-ex3_svd$u
V <-ex3_svd$v

U %*% D %*% t(V)
```

```
##            [,1]       [,2]      [,3]       [,4]        [,5]
## [1,] -0.1976013  0.8916718 1.3009164  0.7757679  1.77833463
## [2,] -0.9299211 -1.0985724 0.1406240 -0.6759909  0.02783712
## [3,] -1.2169035 -0.7583808 0.2140155 -0.2338809  0.56282008
## [4,] -0.9955845  1.1041622 0.8510487  0.8643016 -0.14368113
```

```
random_matrix
```

```
##            [,1]       [,2]      [,3]       [,4]        [,5]
## [1,] -0.1976013  0.8916718 1.3009164  0.7757679  1.77833463
## [2,] -0.9299211 -1.0985724 0.1406240 -0.6759909  0.02783712
## [3,] -1.2169035 -0.7583808 0.2140155 -0.2338809  0.56282008
## [4,] -0.9955845  1.1041622 0.8510487  0.8643016 -0.14368113
```

Ex. 4 – First try to simulate 100 data points for *y* using $y = 5x_1 + 2x_2 + 2x_3 + x_4$ where x_1, x_2 are uniformly distributed in [1,2] while x_3, x_4 are normally distributed with zero mean and unit variance. Then, use the principal component analysis (PCA) to analyze the data to find its principal components. Are the results expected from the formula?

```r
y_matrix <- function(seed=1234) {
  set.seed(seed)
  x_1 <- runif(100, min = 1, max = 2)
  x_2 <- runif(100, min = 1, max = 2)
  x_3 <- rnorm(100, mean = 0, sd = 1)
  x_4 <- rnorm(100, mean = 0, sd = 1)
  y_func <- cbind(5*x_1, 2*x_2, 2*x_3, x_4)

  return(y_func)
}

ex4_y <- y_matrix(1028)
colnames(ex4_y) <- c('5x_1', '2x_2', '2x_3', 'x_4')
```

```r
summary(ex4_y)
```

```
##       5x_1            2x_2            2x_3               x_4
##  Min.   :5.041   Min.   :2.001   Min.   :-5.6042   Min.   :-1.7775
##  1st Qu.:6.186   1st Qu.:2.423   1st Qu.:-1.2634   1st Qu.:-0.6808
##  Median :7.886   Median :2.931   Median :-0.1838   Median : 0.1044
##  Mean   :7.668   Mean   :2.918   Mean   :-0.2489   Mean   : 0.1036
##  3rd Qu.:9.143   3rd Qu.:3.433   3rd Qu.: 1.0010   3rd Qu.: 0.7353
##  Max.   :9.983   Max.   :3.997   Max.   : 4.6198   Max.   : 2.3147
```

```r
sd(ex4_y[,1])
```

```
## [1] 1.572789
```

```r
sd(ex4_y[,2])
```

```
## [1] 0.5819646
```

```r
sd(ex4_y[,3])
```

```
## [1] 1.845164
```

```r
sd(ex4_y[,4])
```

```
## [1] 0.9983244
```

To begin, lets make sure the statistics for each column make sense. Columns for x_3 and x_4 are approximately centered around zero as expected. Standard deviation for x_4 is pretty close to one, and the expected standard deviations for $2x_3$ should be approximately $Var(2x_3) = 2^2 Var(x_3)$ or 2. The expected mean and standard deviations of x_1 and x_2 are 1.5 and ~0.289, values for $5x_1$ and $2x_2$ are approximately 5 and 2 times that.

```
summary(prcomp(ex4_y, scale = TRUE))
```

```
## Importance of components:
##                          PC1    PC2    PC3    PC4
## Standard deviation     1.133 0.9821 0.9490 0.9226
## Proportion of Variance 0.321 0.2411 0.2251 0.2128
## Cumulative Proportion  0.321 0.5621 0.7872 1.0000
```

After scaling, PCA indicates that there are in fact four features of approximately equal importance. Proportion of variance for each is between 0.2121 - 0.321, which means each random variable is approximately 25% contributor to variance.

```
prcomp(ex4_y)
```

```
## Standard deviations (1, .., p=4):
## [1] 1.8745813 1.5484787 0.9861738 0.5738024
##
## Rotation (n x k) = (4 x 4):
##             PC1         PC2         PC3         PC4
## 5x_1 -0.26998634 -0.96119570  0.04228371  0.03771314
## 2x_2 -0.04843510 -0.02675898 -0.02734377 -0.99809334
## 2x_3 -0.95662569  0.27416103  0.09146524  0.03656671
## x_4   0.09812595  0.01491655  0.99453432 -0.03240800
```

Looking at the unscaled rotation matrix, PC1 is made up primarily of $2x_3$, which has the highest calculated standard deviation. The other three variables follow in the exact expected order as well.