279 Option 2

Ben Miller

2023-12-14

Table of Contents

[Linear helper functions 1](#_Toc153891018)

[Logistic helper functions 4](#_Toc153891019)

[Poisson helper functions 6](#_Toc153891020)

[The my\_glm2 function 8](#_Toc153891021)

[Final Product + Examples 9](#_Toc153891022)

## Linear helper functions

#This function is to be used within my function for linear models. It calculates the standard errors, t\_scores, and p-values for each coefficient.  
  
linear\_sig2 <- function(X,beta,n,y){   
 yhats <- X%\*%beta  
 sigma2 <- (1/(n-ncol(X)))\*(sum((y-yhats)^2))  
 Diag <- diag(sigma2\*solve(t(X) %\*% X))  
 se <- sqrt(Diag) #calculate SE's from the above info  
 t\_score <- beta/se #calc t\_scores  
 t\_score1 <- abs(t\_score) #need absvalue when calculating pvalues or the p vals will be wrong  
 p\_value <- 2\*pt(t\_score1,(n-ncol(X)),lower.tail=F)  
 coeff\_df <- data.frame(beta,se,t\_score,p\_value) #turns into data frame for output  
 return(coeff\_df)  
}

Linear with simulated data

# Simulate some data  
set.seed(279)  
n <- 100  
x1 <- runif(n)  
x2 <- rnorm(n)  
y <- rnorm(n, 1 + x1 + x2, 2)  
  
# create the data  
data <- data.frame(y, x1, x2)  
  
#Below is a helper function for linear models. It calculates coefficients and then uses the helper function linear\_sig2 to calculate SEs, t\_scores, and p-values. It also tracks the models deviance. The output is a list containing the model's coefficient table, deviance, the number of iterations needed to create the coefficients (always 1 with linear models), and the models family name.   
  
myglm\_linear2 <- function(formula,family,df){  
 X <- model.matrix(formula, df) #create design matrix from df and formula  
 response <- all.vars(formula)[1]   
 y <- df[[response]] #this and the above line extract the response variable's values and store it as y  
 beta <- solve(t(X) %\*% X) %\*% t(X) %\*% y #creates coefficents   
 mean\_vector <- X%\*%beta  
 deviance <- sum((y - mean\_vector)^2) #calculates deviance  
 n <- nrow(df) #calculates n of df... to be used as input in linear\_sig2  
 coeff\_df <- linear\_sig2(X,beta,n,y)  
 return(list(coefficients = coeff\_df, deviance = deviance,iterations = 1,family = family))  
}  
  
  
m1 <- myglm\_linear2(y ~ x1 + x2,"guassian",data)  
m1$coefficients

## beta se t\_score p\_value  
## (Intercept) 1.1182111 0.4321904 2.587311 1.115660e-02  
## x1 1.4126586 0.7134387 1.980070 5.052892e-02  
## x2 0.9552916 0.1878524 5.085330 1.788912e-06

m1$deviance

## [1] 397.8476

m1$iterations

## [1] 1

m1$family

## [1] "guassian"

Linear with real data

library(palmerpenguins)

## Warning: package 'palmerpenguins' was built under R version 4.2.3

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.3

## Warning: package 'readr' was built under R version 4.2.3

## Warning: package 'purrr' was built under R version 4.2.3

## Warning: package 'dplyr' was built under R version 4.2.3

## Warning: package 'forcats' was built under R version 4.2.3

## Warning: package 'lubridate' was built under R version 4.2.3

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.2 ✔ readr 2.1.4  
## ✔ forcats 1.0.0 ✔ stringr 1.5.0  
## ✔ ggplot2 3.4.2 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
## ✔ purrr 1.0.1   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts to become errors

data <- penguins |>  
 drop\_na()  
  
  
m1 <- myglm\_linear2(body\_mass\_g ~ flipper\_length\_mm + bill\_length\_mm,"guassian",data)  
m1$coefficients

## beta se t\_score p\_value  
## (Intercept) -5836.298732 312.603503 -18.669972 1.341791e-53  
## flipper\_length\_mm 48.889692 2.034204 24.033815 1.737931e-74  
## bill\_length\_mm 4.958601 5.213505 0.951107 3.422461e-01

m1$deviance

## [1] 51071963

m1$family

## [1] "guassian"

m1$iterations

## [1] 1

## Logistic helper functions

#The below function is a helper function used to take initialized logit coefficients, among other inputs, and use Fisher scoring to update those coefficients 'max\_iter' number of times or until the difference in deviances is < .001. The model's final coefficients, its deviance, and the number of Fisher iterations it took to converge are then returned. The function also calculates and returns the coefficients SE's, z\_scores, and p-values.   
  
UpdateCoefficients <- function(X,y,beta,max\_iter){   
 CoeffVector <- X%\*%beta  
 ProbVector <- (exp(CoeffVector)/(1+exp(CoeffVector)))  
 Past\_Deviance <--2\*sum(dbinom(y, 1, ProbVector, log=T))  
 for(a in 1:max\_iter){ #Everything within this loop uses Fisher scoring to iteratively update the models coefficients  
 CoeffVector <- X%\*%beta  
 ProbVector <- (exp(CoeffVector)/(1+exp(CoeffVector)))  
 score <- (t(X) %\*% (y-ProbVector)) #calc score  
 diag\_vector <- rep(NA,length(ProbVector))  
 for(i in 1:length(ProbVector)){  
 diag\_vector[i] <- ProbVector[i]\*(1 - ProbVector[i])}  
 Diag <- diag(diag\_vector) #create diag matrix  
 info <- (t(X) %\*% Diag %\*% X) #calc info  
 beta <- (beta + solve(info) %\*% score) #update coeffs  
 CoeffVector <- X%\*%beta #The below code calculates difference in deviances to see if the break condition is met  
 ProbVector <- (exp(CoeffVector)/(1+exp(CoeffVector)))  
 Deviance <- -2\*sum(dbinom(y, 1, ProbVector, log=T))  
 Diff <- Past\_Deviance - Deviance   
 Past\_Deviance <- Deviance  
 if(Diff < .001){   
 break  
 }  
 }  
 diag\_vector <- rep(NA,length(ProbVector)) #The below code creates a data frame containing the model's coefficients and those coefficients standard errors, z\_scores, and p-values.  
 for(i in 1:length(ProbVector)){  
 diag\_vector[i] <- ProbVector[i]\*(1 - ProbVector[i])}  
 Diag <- diag(diag\_vector) #create diag matrix  
 DiagSE <- diag(solve(t(X) %\*% Diag %\*% X))  
 se <- sqrt(DiagSE)  
 z\_score <- beta/se  
 z\_score1 <- abs(z\_score)  
 p\_value <- 2\*pnorm(z\_score1,lower.tail = F)  
 coeff\_df <- data.frame(beta,se,z\_score,p\_value)  
 return(list(deviance = Deviance,coeff\_df = coeff\_df,iterations = a))  
}

Logistic with simulated data

set.seed(123)  
n <- 200  
x <- rnorm(n)  
p <- exp(1 + x)/(1 + exp(1 + x))  
y <- rbinom(n, 1, p)  
  
  
data <- data.frame(y,x)  
  
#Below is a helper function for logistic models. It initializes coefficients and then uses the helper function, UpdateCoefficients, to update coefficients and calculate SEs, t\_scores, p-values, the model's deviance, and how many Fisher iterations it took for the coefficients to converge. The output is a list containing the model's coefficient table, deviance, the number of Fisher iterations needed to create the coefficients, and the models family name.   
  
my\_glm\_log2 <- function(formula,family,max\_iter,df){   
 response <- all.vars(formula)[1]  
 y <- df[[response]] #145 and 144 extract the values of the formulas response variable  
 X <- model.matrix(formula, df) #creates design matrix from formula and df  
 BetaNum <- ncol(X)  
 beta <- rep(0,BetaNum)  
 ybar <- mean(y)  
 beta[1] <- log(ybar/(1-ybar)) #Initialize coeffs  
 model <- UpdateCoefficients(X,y,beta,max\_iter) #Helper function output is stored as model so its output can be extracted outside of UpdateCoefficients   
 return(list(coefficients = model$coeff\_df, family = family, iterations = model$iterations, deviance = model$deviance))  
}  
  
m1 <- my\_glm\_log2(y ~ x,"binomial",50,data)  
m1$coefficients

## beta se z\_score p\_value  
## (Intercept) 1.189801 0.1839126 6.469385 9.840288e-11  
## x 0.922279 0.2143541 4.302595 1.688094e-05

m1$family

## [1] "binomial"

m1$iterations

## [1] 4

m1$deviance

## [1] 208.6589

Logistic with real data

data <- penguins |>  
 drop\_na() |>  
 mutate(sex = ifelse(sex == "female", 1, 0))  
  
m1 <- my\_glm\_log2(sex ~ flipper\_length\_mm + bill\_length\_mm,   
 "binomial", 10, data)   
m1$coefficients

## beta se z\_score p\_value  
## (Intercept) 7.005985900 1.72762038 4.0552809 5.007409e-05  
## flipper\_length\_mm -0.007736914 0.01108136 -0.6981915 4.850574e-01  
## bill\_length\_mm -0.124374714 0.02952819 -4.2120671 2.530444e-05

m1$family

## [1] "binomial"

m1$deviance

## [1] 419.9377

m1$iterations

## [1] 3

## Poisson helper functions

#The below function is a helper function used to take initialized poisson coefficients, among other inputs, and use Fisher scoring to update those coefficients 'max\_iter' number of times or until the difference in deviances is < .001. The model's final coefficients, its deviance, and the number of Fisher iterations it took to converge are then returned. The function also calculates and returns the coefficients SE's, z\_scores, and p-values.  
  
update\_pois\_coeffs <- function(X,y,beta,max\_iter){  
 CoeffVector <- X%\*%beta  
 ProbVector <- (exp(CoeffVector)/(1+exp(CoeffVector)))  
 Past\_Deviance <- 2\*(sum(dpois(y, y, log=T)) - sum(dpois(y, ProbVector, log=T)))  
 for(a in 1:max\_iter){ #Everything within this loop uses Fisher scoring to iteratively update the models coefficients  
 CoeffVector <- X%\*%beta  
 ProbVector <- exp(CoeffVector) #get means vector  
 score <- (t(X) %\*% (y-ProbVector)) #calc score  
 diag\_vector <- rep(NA,length(ProbVector))  
 for(i in 1:length(ProbVector)){  
 diag\_vector[i] <- ProbVector[i]}  
 Diag <- diag(diag\_vector) #create diag matrix  
 info <- (t(X) %\*% Diag %\*% X) #calc info  
 beta <- (beta + solve(info) %\*% score)#update coeffs  
 CoeffVector <- X%\*%beta #The below code calculates difference in deviances to see if the break condition is met  
 ProbVector <- exp(CoeffVector)  
 Deviance <- 2\*(sum(dpois(y, y, log=T)) - sum(dpois(y, ProbVector, log=T)))  
 Diff <- Past\_Deviance - Deviance   
 Past\_Deviance <- Deviance  
 if(Diff < .001){  
 break  
 }  
 }   
 diag\_vector <- rep(NA,length(ProbVector)) #The below code creates a data frame containing the model's coefficients and those coefficients standard errors, z\_scores, and p-values.  
 for(i in 1:length(ProbVector)){  
 diag\_vector[i] <- ProbVector[i]}  
 Diag <- diag(diag\_vector) #create diag matrix  
 DiagSE <- diag(solve(t(X) %\*% Diag %\*% X))  
 se <- sqrt(DiagSE)  
 z\_score <- beta/se  
 abs\_z <- abs(z\_score)  
 p\_value <- 2\*pnorm(abs\_z,lower.tail = F)  
 coeff\_df <- data.frame(beta,se,z\_score,p\_value)  
 return(list(deviance = Deviance,coeff\_df = coeff\_df,iterations = a))  
}

Poisson with simulated data

set.seed(214)  
n <- 300  
x1 <- rbinom(n, 1, 0.5)  
x2 <- runif(n)  
x3 <- runif(n)  
y <- rpois(n, exp(0.5 - x1 + x2 - 0.5\*x3))  
  
data <- data.frame(y,x1,x2,x3)  
  
#Below is a helper function for poisson models. It initializes coefficients and then uses the helper function, update\_pois\_coeffs, to update coefficients and calculate SEs, t\_scores, p-values, the model's deviance, and how many Fisher iterations it took for the coefficients to converge. The output is a list containing the model's coefficient table, deviance, the number of Fisher iterations needed to create the coefficients, and the models family name.   
  
my\_glm\_poiss2 <- function(formula,family,max\_iter,df){  
 X <- model.matrix(formula, df) #create design matrix from formula and df  
 response <- all.vars(formula)[1]  
 y <- df[[response]] #the above code extracts the values of the formula's response variable   
 BetaNum1 <- ncol(X)  
 beta <- rep(0,BetaNum1)  
 ybar <- mean(y)  
 beta[1] <- log(ybar) #initializes coefficients  
 model <- update\_pois\_coeffs(X,y,beta,max\_iter) #Helper function output is stored as model so its output can be extracted outside of update\_pois\_coeffs  
 return(list(coefficients = model$coeff\_df, family = family, deviance = model$deviance,iterations = model$iterations))  
}  
  
  
m1 <- my\_glm\_poiss2(y ~ x1 + x2 + x3,"poisson",12,data)  
m1$coefficients

## beta se z\_score p\_value  
## (Intercept) 0.4096576 0.1288543 3.179232 1.476660e-03  
## x1 -0.9375865 0.1000650 -9.369779 7.268426e-21  
## x2 1.0644028 0.1686682 6.310630 2.779011e-10  
## x3 -0.3281246 0.1686991 -1.945029 5.177144e-02

m1$family

## [1] "poisson"

m1$deviance

## [1] 358.0896

m1$iterations

## [1] 4

## The my\_glm2 function

#Finally, I use all of the above helper functions to create my\_glm2. The function takes inputs of a formula (y ~ x1 + x2 +... xn), the desired models family name (guassian, binomial, or poisson), a default max iterations fo 50 for Fisher scoring, and a data frame. Depending on the input family name, my\_glm2 uses the corresponding helper function to return a list containing the model's coefficient table (with t/z scores, SE's, and p-vals), the model's deviance, the model's family name, and the iterations required for the coefficients to converge.   
  
my\_glm2 <- function(formula,family,max\_iter=50,df){  
 if(family == "guassian"){  
 return(myglm\_linear2(formula,family,df))  
 }  
 if(family == "binomial"){  
 return(my\_glm\_log2(formula,family,max\_iter=50,df))   
 }  
 if(family == "poisson"){  
 return(my\_glm\_poiss2(formula,family,max\_iter=50,df))  
 }  
   
}

## Final Product + Examples

Linear With Simulated Data

set.seed(279)  
n <- 100  
x1 <- runif(n)  
x2 <- rnorm(n)  
y <- rnorm(n, 1 + x1 + x2, 2)  
  
# create the data  
data <- data.frame(y, x1, x2)  
  
m1 <- my\_glm2(y ~ x1 + x2,"guassian",max\_iter=50,data)  
m1$coefficients

## beta se t\_score p\_value  
## (Intercept) 1.1182111 0.4321904 2.587311 1.115660e-02  
## x1 1.4126586 0.7134387 1.980070 5.052892e-02  
## x2 0.9552916 0.1878524 5.085330 1.788912e-06

m1$deviance

## [1] 397.8476

m1$iterations

## [1] 1

m1$family

## [1] "guassian"

Linear With Real Data

library(palmerpenguins)  
library(tidyverse)  
  
data <- penguins |>  
 drop\_na()  
  
m1 <- my\_glm2(body\_mass\_g ~ flipper\_length\_mm + bill\_length\_mm,"guassian",50,data)  
m1$coefficients

## beta se t\_score p\_value  
## (Intercept) -5836.298732 312.603503 -18.669972 1.341791e-53  
## flipper\_length\_mm 48.889692 2.034204 24.033815 1.737931e-74  
## bill\_length\_mm 4.958601 5.213505 0.951107 3.422461e-01

m1$deviance

## [1] 51071963

m1$iterations

## [1] 1

m1$family

## [1] "guassian"

Logistic With Simulated Data

set.seed(123)  
n <- 200  
x <- rnorm(n)  
p <- exp(1 + x)/(1 + exp(1 + x))  
y <- rbinom(n, 1, p)  
  
data <- data.frame(y,x)  
  
  
m1 <- my\_glm2(y ~ x,"binomial",50,data)  
m1$coefficients

## beta se z\_score p\_value  
## (Intercept) 1.189801 0.1839126 6.469385 9.840288e-11  
## x 0.922279 0.2143541 4.302595 1.688094e-05

m1$family

## [1] "binomial"

m1$iterations

## [1] 4

m1$deviance

## [1] 208.6589

Logistic With Real Data

data <- penguins |>  
 drop\_na() |>  
 mutate(sex = ifelse(sex == "female", 1, 0))  
  
m1 <- my\_glm2(sex ~ flipper\_length\_mm + bill\_length\_mm,   
 "binomial", 10, data)   
m1$coefficients

## beta se z\_score p\_value  
## (Intercept) 7.005985900 1.72762038 4.0552809 5.007409e-05  
## flipper\_length\_mm -0.007736914 0.01108136 -0.6981915 4.850574e-01  
## bill\_length\_mm -0.124374714 0.02952819 -4.2120671 2.530444e-05

m1$family

## [1] "binomial"

m1$deviance

## [1] 419.9377

m1$iterations

## [1] 3

Poisson With Simulated Data

set.seed(214)  
n <- 300  
x1 <- rbinom(n, 1, 0.5)  
x2 <- runif(n)  
x3 <- runif(n)  
y <- rpois(n, exp(0.5 - x1 + x2 - 0.5\*x3))  
  
data <- data.frame(y,x1,x2,x3)  
  
m1 <- my\_glm\_poiss2(y ~ x1 + x2 + x3,"poisson",16,data)  
m1$coefficients

## beta se z\_score p\_value  
## (Intercept) 0.4096576 0.1288543 3.179232 1.476660e-03  
## x1 -0.9375865 0.1000650 -9.369779 7.268426e-21  
## x2 1.0644028 0.1686682 6.310630 2.779011e-10  
## x3 -0.3281246 0.1686991 -1.945029 5.177144e-02

m1$family

## [1] "poisson"

m1$deviance

## [1] 358.0896

m1$iterations

## [1] 4