# Comparing Full Methods

## Amy Pitts

## 3/20/2022

## Data import + Cleaning

Cleaning involves

- removing the id variable
- re-factoring the outcome variable `diagnosis` to be 0 if B and `if M. Then this variable is re-names toy`.

## Test + Train

Next we will break the data into the test and train sets. The split will be 80-20.

```
partition <- function(p, data){
  set.seed(5)
  # generating a probability value
  part_p = runif(nrow(data), min = 0, max = 1)
  # assigning partition id based on probability value
  # parameter p sets proportion of train vs test
  part_id = ifelse(part_p <= p, "train", "test")
  # appending to data set
  data_new = cbind(data, part_id)

  return(data_new)
}
# here training proportion is set to 0.8
part_data <- partition(p = 0.8, data = dat)

trn_data <-
  part_data %>%
  filter(part_id == "train") %>%
  select(-part_id)

tst_data <- # this will come back into play for test error
  part_data %>%
  filter(part_id == "test") %>%
  select(-part_id)
```

I want to only check 3 variables

```
cor(trn_data$texture_mean, trn_data$radius_mean)
```

```
## [1] 0.3249025
```

```
cor(trn_data$radius_mean, trn_data$smoothness_mean)
```

```
## [1] 0.1527862
```

```r
cor(trn_data$smoothness_mean, trn_data$texture_mean)
```

```
## [1] -0.02904833
```

```r
# trn_data = trn_data %>%
#   select(y, texture_mean, radius_mean, smoothness_mean)
# head(trn_data)
```

```r
cor_dat <- data.frame(cor(dat)) %>%
  mutate(variable_left = names(data.frame(cor(dat)))) %>%
  relocate(variable_left) %>%
  pivot_longer(
    c(y:fractal_dimension_worst),
    names_to = "variables_top",
    values_to = "cor"
  ) %>%
  mutate(cor = abs(cor)) %>%
  arrange(desc(cor)) %>%
  filter(cor != 1) %>%
  filter(cor > 0.8)
head(cor_dat)
```

```
## # A tibble: 6 x 3
##   variable_left   variables_top      cor
##   <chr>           <chr>            <dbl>
## 1 radius_mean     perimeter_mean   0.998
## 2 perimeter_mean  radius_mean      0.998
## 3 radius_worst    perimeter_worst  0.994
## 4 perimeter_worst radius_worst     0.994
## 5 radius_mean     area_mean        0.987
## 6 area_mean       radius_mean      0.987
```

## Comparing all Methods

We will be comparing:

- Waveley's Method
- Amy's Method
- base R glm
- glmnet (lambda $= 0$)
- Jimmy's LASSO function with $\lambda = 0$

### Waveley's Method

```r
rep_col <- function(x, n){
  matrix(rep(x, each = n), ncol = n, byrow = TRUE)
}

logistic_stuff <- function(dat, beta){

  x <- dat[[1]] %>% unname() %>% as.matrix()
  y <- dat[[2]] %>% unname() %>% as.matrix()

  x_with_1 <- cbind(1, x)
```

```r
  u <- x_with_1 %*% beta
# return(u)

  expu <- exp(u)

  loglik <- sum(y*u - log(1 + expu))

  p <- expu/(1 + expu)
#   return(p)
# return(p)
  grad <- t(x_with_1) %*% (y - p)

  i_mat <- diag(nrow(p))
  diag(i_mat) <- p*(1 - p)

  hess <- -(t(x_with_1) %*% i_mat %*% x_with_1)

  return(
    list(
    loglik = loglik,
    grad = grad,
    hess = hess
  ))
}

NewtonRaphson_w <- function(dat, func, start, tol = 1e-8, maxiter = 200) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf

  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol && !is.na(stuff$loglik)) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
  #   newhess <- ((stuff$hess + t(stuff$hess))/2)
    newhess <- stuff$hess

  #   if (!is.negative.definite(newhess)) { # redirection
  #     while (!is.negative.definite(newhess)) {
        # subtracts identity matrix until a negative definite matrix is achieved
  #       newhess1 <- newhess - diag(nrow(newhess))
        # sanity check print("changing ascent direction")
  #       newhess <- ((newhess1 + t(newhess1))/2)
  #     }
  #   }

    cur <- prev - solve(newhess) %*% stuff$grad
    stuff <- func(dat, cur)

    if (stuff$loglik < prevloglik) {  # back tracking (half-step)
      j = 1
```

3

```r
      while (stuff$loglik < prevloglik & (!is.na(stuff$loglik))) {
          halfstep = 1/(2^j)
          cur <- prev - halfstep*solve(newhess) %*% stuff$grad
          stuff <- func(dat, cur)
        # sanity check print("backtracking")
          j = j + 1
      }
    }
    res <- rbind(res, c(i, stuff$loglik, cur))
  }
  return(res)
}


rerun_NR <- function(beta_init){
  # calling the function
  ans_w <- NewtonRaphson_w(
        list(x = trn_data %>% select(-y) %>% as.matrix(),
          y = trn_data %>% select(y) %>% as.matrix()),
          logistic_stuff,
          beta_init)

  # organixing the results
  if (sum(is.na(ans_w[nrow(ans_w),])) > 0) {
    beta_est <- ans_w[nrow(ans_w) - 1, -c(1,2)]
  }

  if (sum(is.na(ans_w[nrow(ans_w),])) == 0) {
    beta_est <- ans_w[nrow(ans_w), -c(1,2)]
  }
  # results
  waveley_est<- tibble(beta_subscript = seq(0, (ncol(trn_data)-1)),
                      beta_estimates = beta_est) #%>% knitr::kable()
  return(waveley_est)
}

beta_init <- rep(0.0001, ncol(trn_data)) %>% as.matrix()


beta_init <- rep(0.0001, ncol(trn_data)) %>% as.matrix()
w_0.0001 <- rerun_NR(beta_init)$beta_estimates

beta_init <- rep(0.001, ncol(trn_data)) %>% as.matrix()
w_0.001 <- rerun_NR(beta_init)$beta_estimates
```

**Amy's Method**

```r
loglike_func <- function(dat, betavec){
  # setting up an intercept
  dat_temp = dat %>%
    rename(intercept = y) %>%
    mutate(intercept = rep(1, nrow(dat) ))
  dat_x = unname(as.matrix(dat_temp)) # creating the x matrix
```

```r
  # finding the pi values
  u = dat_x %*% betavec
  pi <- exp(u) / (1+exp(u))

  # loglikelihood
  loglik <- sum(dat$y*u - log(1 + exp(u)))

  #gradient
  grad <- t(dat_x)%*%(dat$y - pi)

  # Hessian
  W = matrix(0, nrow = dim(dat)[1], ncol = dim(dat)[1])
  diag(W)= pi*(1-pi)
  hess = -(t(dat_x)%*% W %*% (dat_x))

  return(list(loglik = loglik, grad = grad, hess = hess))

}
#loglike_func(dat, betavec = c( rep(0.03, 31)))  # test!


NewtonRaphson_a <- function(dat,  start, tol = 1e-8, maxiter = 200){
  i = 0
  cur = start
  stuff = loglike_func(dat, cur)
  res <- c(i=0, "loglik" = stuff$loglik,  "step" = 1, cur)
  prevloglik <- -Inf # To make sure it iterates
  while(i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    step = 1
    i <- i + 1
    prevloglik <- stuff$loglik

    # checking negative definite
    eigen_vals = eigen(stuff$hess)
    if(max(eigen_vals$values) <= 0 ){ # check neg def, if not change
      hess = stuff$hess
    } else{ # if it is pos def then need to adjust
      hess = stuff$hess - (max(eigen_vals$values) + 0.1)*diag(nrow(stuff$hess))
    }


    prev  <- cur
    cur   <- prev - rep(step, length(prev))*(solve(stuff$hess) %*% stuff$grad)
    stuff <- loglike_func(dat, cur) # log-lik, gradient, Hessian

    # doing half stepping
    while(stuff$loglik < prevloglik){
      stuff <- loglike_func(dat, prev)
      step  <- step / 2 # this is where half steping happens
      cur   <- prev - step*(solve(stuff$hess) %*% stuff$grad)
      stuff <- loglike_func(dat, cur)
    }
    # Add current values to results matrix
```

```
    res <- rbind(res, c(i, stuff$loglik, step, cur))
  }

  colnames(res) <- c("i", "loglik",  "step", "intercept", names(dat[,-1]))
  return(res)
}
#Running the algorithm with random starting values

betavec = c(rep(0.0001, ncol(trn_data)))
ans0.0001 <- NewtonRaphson_a(trn_data, betavec)

# beta values
amy_est_0.0001 <- data.frame(ans0.0001) %>%
  select(-step, -loglik, -i) %>%
  filter(row_number() == n()) %>%
  pivot_longer(
    cols = everything(),
    names_to = "term",
    values_to = "a_0.0001"
  ) %>%
  mutate(`a_0.0001` = round(`a_0.0001`,3))
```

It will also be interesting to look at the stability depending on initial values.

```
betavec = c(rep(0.001, ncol(trn_data)))
ans0.001 <- NewtonRaphson_a(trn_data, betavec)

# beta values
amy_est_0.001 <- data.frame(ans0.001) %>%
  select(-step, -loglik, -i) %>%
  filter(row_number() == n()) %>%
  pivot_longer(
    cols = everything(),
    names_to = "term",
    values_to = "a_0.001"
  ) %>%
  mutate(`a_0.001` = round(`a_0.001`,3))


betavec = c(rep(0.01, ncol(trn_data)))
ans0.01 <- NewtonRaphson_a(trn_data, betavec)

# beta values
amy_est_0.01 <- data.frame(ans0.01) %>%
  select(-step, -loglik, -i) %>%
  filter(row_number() == n()) %>%
  pivot_longer(
    cols = everything(),
    names_to = "term",
    values_to = "a_0.01"
  ) %>%
  mutate(`a_0.01` = round(`a_0.01`,3))
```

**Base R glm**

```r
# trn_data <- trn_data[c(1:455),]
glm_fit <- glm(y~., data=trn_data, family = "binomial")
result_glm <- summary(glm_fit)

glm_est <- glm_fit %>% broom::tidy() %>%
  select(term, estimate) %>%
  mutate(glm_est = round(estimate, 3)) %>%
  select(-estimate) %>%
  mutate(term = ifelse(term == "(Intercept)", "intercept", term))
```

**glmnet**

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-2
```

```r
xdat = as.matrix(trn_data %>% select(-y))
glmnet_fit <- glmnet(x = xdat, y = trn_data$y, family="binomial", lambda = 0)
glmnet_est <- as.vector(coef(glmnet_fit)) %>% round(3)
```

**Jimmy's Method (LASSO) but lambda = 0**

```r
# logistic function
logistic <- function(x) 1 / (1 + exp(-x))

# shrinkage function
S <- function(beta, gamma) {
  if(abs(beta) <= gamma) {
    0
  } else if(beta > 0) {
    beta - gamma
  } else {
    beta + gamma
  }
}

# probability adjustment function
p_adj <- function(p, epsilon) {
  if (p < epsilon) {
    0
  } else if(p > 1 - epsilon) {
    1
  } else {
    p
```

```r
  }
}

# weight adjustment function
w_adj <- function(p, epsilon) {
  if ((p < epsilon) | (p > 1 - epsilon)) {
    epsilon
  } else {
    p * (1 - p)
  }
}

set.seed(1)
epsilon <- 10^(-5)

n     <- nrow(trn_data)
X     <- trn_data[ , -c(1)]
X     <- as.matrix(cbind(rep(1, n), X))
y     <- trn_data$y

lambda <- 0 # (max(t(X) %*% y) / n)

# initialize parameters
beta <- rep(0, ncol(X))
p <- map_dbl(logistic(- X %*% beta), p_adj, epsilon)
w <- map_dbl(p, w_adj, epsilon)
z <- X %*% beta + (y - p) / w

terminate <- 0
iter <- 1
while(terminate < 1) {

  beta_old <- beta
  # initially go through all parameters
  K <- 1:ncol(X)
  #if(iter > 1) {
  #   K <- which(beta > 0)
  #}

  for(k in K) {
    x_k    <- X[ , k]
    x_notk <- X[ , -k]
    b_notk <- beta[-k]

    # un-penalized coefficient update
    b_k_temp <- sum(w * (z - x_notk %*% b_notk) * x_k) / sum(w * x_k^2)
    # shrinkage update
    b_k       <- S(b_k_temp, lambda / mean(w * x_k^2))

    # update beta vector along with other parameters
    beta[k] <- b_k
    #p <- map_dbl(logistic(- X %*% beta), p_adj, epsilon)
    #w <- map_dbl(p, w_adj, epsilon)
```

```r
    #z <- X %*% beta + (y - p) / w
  }

  iter <- iter + 1

  if(iter == 1000 | max(abs(beta - beta_old)) < 10^-10) {
    print(iter)
    terminate <- 1
  }

}
```

```
## [1] 1000
```

## Combining Results

```r
combine_res <- amy_est_0.0001 %>%
  full_join(amy_est_0.001) %>%
  full_join(amy_est_0.01) %>%
  full_join(glm_est) %>%
  mutate(glmnet_est= glmnet_est)
```

```
## Joining, by = "term"
## Joining, by = "term"
## Joining, by = "term"
```

```r
combine_res %>%
  mutate(
    glm_est = round(glm_est,3),
    w_0.0001 = round(w_0.0001,3),
    w_0.001 = round(w_0.001, 3),
    #j_est = round(beta,3)
  ) %>%
  #mutate(term = c(0:(ncol(trn_data)-1))) %>%
  knitr::kable()
```

| term | a_0.0001 | a_0.001 | a_0.01 | glm_est | glmnet_est | w_0.0001 | w_0.001 |
|---|---|---|---|---|---|---|---|
| intercept | -443.746 | -208.587 | -217.456 | -1165.854 | -394.803 | -440.216 | -176.105 |
| radius_mean | -82.398 | -25.090 | -26.594 | -226.537 | -27.666 | -81.695 | -23.478 |
| texture_mean | -1.083 | -0.907 | -1.182 | -2.968 | 0.551 | -1.074 | -0.854 |
| perimeter_mean | 12.062 | 4.584 | 5.694 | 33.514 | 0.013 | 11.957 | 4.495 |
| area_mean | 0.197 | 0.018 | -0.051 | 0.531 | 0.229 | 0.195 | -0.009 |
| smoothness_mean | 689.557 | 248.111 | 258.230 | 1944.755 | -25.726 | 683.657 | 233.946 |
| compactness_mean | -1173.391 | -551.774 | -537.999 | -3261.548 | -1021.478 | -1163.188 | -458.713 |
| concavity_mean | -564.588 | -348.851 | -400.338 | -1344.234 | -387.748 | -560.612 | -307.720 |
| concave.points_mean | 1858.833 | 1348.717 | 1411.895 | 4655.386 | 2238.209 | 1844.502 | 1112.090 |
| symmetry_mean | -1161.107 | -760.142 | -781.333 | -3092.685 | -850.444 | -1151.316 | -609.451 |
| fractal_dimension_mean | 3515.343 | 2156.209 | 2004.182 | 9771.135 | 3071.874 | 3484.983 | 1695.822 |
| radius_se | 763.574 | 486.470 | 529.362 | 1960.594 | 438.904 | 757.534 | 421.324 |
| texture_se | -11.605 | -9.954 | -10.952 | -29.993 | -6.644 | -11.519 | -8.909 |
| perimeter_se | -21.247 | -14.016 | -16.195 | -50.759 | -9.283 | -21.107 | -13.984 |
| area_se | -4.094 | -2.450 | -2.782 | -10.755 | -1.580 | -4.060 | -2.123 |
| smoothness_se | -3947.158 | -2961.395 | -2961.740 | -10030.701 | -4492.049 | -3916.295 | -2466.660 |
| compactness_se | 719.542 | 311.800 | 392.120 | 2087.573 | -1035.439 | 713.470 | 378.441 |
| concavity_se | -631.281 | -292.923 | -294.554 | -1798.657 | 265.090 | -625.920 | -275.294 |
| concave.points_se | 6698.376 | 4844.060 | 5137.757 | 16699.060 | 5971.022 | 6648.329 | 4172.128 |
| symmetry_se | -1171.617 | -816.787 | -787.496 | -2943.672 | -1026.699 | -1162.130 | -613.508 |
| fractal_dimension_se | -7303.521 | -5367.256 | -6069.727 | -18435.908 | -1387.765 | -7249.560 | -5089.317 |
| radius_worst | -45.210 | -49.079 | -55.321 | -109.899 | -9.988 | -44.875 | -44.414 |
| texture_worst | 6.260 | 4.386 | 4.602 | 16.077 | 4.823 | 6.210 | 3.690 |
| perimeter_worst | 1.664 | 1.191 | 1.435 | 3.385 | 1.024 | 1.656 | 1.371 |
| area_worst | 0.421 | 0.488 | 0.547 | 1.046 | 0.215 | 0.417 | 0.424 |
| smoothness_worst | 396.284 | 325.164 | 330.644 | 964.315 | 672.007 | 393.308 | 266.992 |
| compactness_worst | -415.650 | -299.911 | -328.047 | -1045.701 | -82.531 | -412.484 | -268.221 |
| concavity_worst | 522.362 | 323.819 | 339.607 | 1334.798 | 326.429 | 518.293 | 274.807 |
| concave.points_worst | 13.607 | -53.989 | -112.800 | 158.346 | -76.691 | 12.834 | -85.613 |
| symmetry_worst | 814.391 | 552.328 | 550.390 | 2115.083 | 707.835 | 807.668 | 435.517 |
| fractal_dimension_worst | 437.947 | 497.054 | 711.435 | 825.123 | -597.772 | 435.993 | 570.389 |