Comparing Full Methods

Amy Pitts

3/20/2022

Data import

```
partition <- function(p, data){
  set.seed(100)
  # 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)
```

```
intial_beta_val = 0.0001
```

```
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)

  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)

    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
      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)
}

beta_init <- rep(intial_beta_val, 31) %>% as.matrix()

ans_w <- NewtonRaphson_w(
      list(x = tst_data %>% select(-y) %>% as.matrix(),
       y = tst_data %>% select(y) %>% as.matrix()),
       logistic_stuff,
       beta_init)

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)]
}

waveley_est<- tibble(beta_subscript = seq(0, 30), beta_estimates = beta_est) #%>% knitr::kable()
```

```
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

  # 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(intial_beta_val, 31))
ans <- NewtonRaphson_a(tst_data, betavec)
```

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

```r
# tst_data <- tst_data[c(1:455),]
glm_fit <- glm(y~., data=tst_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))
```

```r
library(glmnet)
```

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

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

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

```
## Loaded glmnet 4.0-2
```

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

combining

```r
combine_res <- amy_est %>%
  full_join(glm_est) %>%
  mutate(glmnet_est= glmnet_est)
```

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

```r
combine_res %>%
  mutate(
    glm_est = round(glm_est,2),
    waveley_est = waveley_est$beta_estimates
  ) %>%
  knitr::kable()
```

term	amy_est	glm_est	glmnet_est	waveley_est
intercept	-500.639	-531.96	-328.388	-117.7728593
radius_mean	54.381	53.44	5.402	2.2709181
texture_mean	0.899	0.71	-0.017	-0.8275980
perimeter_mean	-5.612	-5.18	-0.005	0.0627462
area_mean	-0.025	-0.04	0.005	-0.0748486
smoothness_mean	-4371.611	-4367.47	-425.932	-652.2434499
compactness_mean	-783.025	-800.93	-154.697	-168.4436357
concavity_mean	-789.277	-808.73	48.627	-120.5004638
concave.points_mean	2550.294	2544.91	465.188	437.0851124
symmetry_mean	812.118	816.55	189.609	119.0231760
fractal_dimension_mean	6090.275	6258.99	1409.986	1074.6821466
radius_se	481.130	487.90	98.597	77.8192986
texture_se	19.266	19.04	-7.211	0.3245276
perimeter_se	-39.241	-39.87	-0.450	-5.0747669
area_se	-1.527	-1.53	-0.468	0.0107290
smoothness_se	-12889.794	-12946.28	-3681.983	-2365.5444886
compactness_se	633.706	798.62	269.835	102.8728899
concavity_se	-36.801	-51.68	138.117	33.7220095
concave.points_se	2922.737	2975.57	908.110	658.9992851
symmetry_se	4172.068	4197.00	776.000	827.9027921
fractal_dimension_se	-21017.063	-21921.47	-10935.789	-4233.8253979
radius_worst	-24.930	-25.76	4.594	-4.7091398
texture_worst	-1.201	-1.05	0.870	0.6747923
perimeter_worst	1.602	1.74	-0.024	0.5086267
area_worst	0.135	0.14	-0.020	0.0778434
smoothness_worst	3081.347	3100.19	854.070	554.2709590
compactness_worst	145.331	134.28	56.669	34.8128089
concavity_worst	143.531	147.73	-7.008	20.7123562
concave.points_worst	-166.634	-182.02	-130.643	-79.6679058
symmetry_worst	-729.976	-731.74	-135.525	-128.3210943
fractal_dimension_worst	565.925	626.68	270.973	124.0923821