

# 200301-EDA\_and\_model-yuqi

Yuqi Miao ym2771

3/1/2020

## data and manipulation

```
cancer = read_csv("breast-cancer.csv") %>%
  mutate(diagnosis = as.numeric(factor(diagnosis, levels = c("B","M"), labels = c(0,1))) - 1) %>%
  mutate(radius_mean = scale(radius_mean))
predictor_scale = as.tibble(scale(cancer[3:12]))
cancer_package = cancer %>% select(contains("mean"), diagnosis)
cancer_scale = cbind(rep(1,569), predictor_scale, cancer$diagnosis)
names(cancer_scale) = c("ones", names(cancer)[3:12], "response")

## standardize data!!
```

$$\log\left(\frac{\pi_i}{1 - \pi_i}\right) = \mathbf{x}_i\boldsymbol{\beta}$$

```
hessian_gradient_log = function(data, beta_vec){
  y = as.matrix(data%>% select(last_col()))
  x = as.matrix(data[,1:dim(data)[2]-1])
  theta = x%*%beta_vec
  pi = exp(theta)/(1+exp(theta))
  loglikelihood = sum(y*theta-log(1+exp(theta)))
  gradient = t(x)%*%(y-pi)
  pi_matrix = matrix(0, nrow = dim(data)[1], ncol = dim(data)[1])
  diag(pi_matrix) = pi*(1-pi)
  hessian = -t(x)%*%pi_matrix%*%(x)
  return(list(loglikelihood = loglikelihood, gradient = gradient, hessian = hessian))
}

a = hessian_gradient_log(cancer_scale, rep(0.02,11))
```

```
NR = function(data, beta_start, tol = 1e-10, max = 200){
  i = 0
  cur = beta_start
  stuff = hessian_gradient_log(data, cur)
  curlog = stuff$loglikelihood
  res = c(i = 0, curlog = curlog, cur = cur, step = step)
  prevlog = -Inf
  while((i<=max)&&(abs(curlog-prevlog)>tol)){
    step = 1
    i = i+1
    prevlog = stuff$loglikelihood
    eigen = eigen(stuff$hessian)
    if(sum(eigen$values)==0){
      hessian = stuff$hessian
    }
    else{
      hessian = stuff$hessian - max(eigen$values)
    }
    prev = cur
  }
```

```

cur = prev - rep(step,length(prev))*(solve(hessian)%*%stuff$gradient)
stuff = hessian_gradient_log(data,cur)
curlog = stuff$loglikelihood
while(curlog<prevlog){
  step = step/2
  cur = prev - rep(step,length(prev))*(solve(hessian)%*%stuff$gradient)
  stuff = hessian_gradient_log(data,cur)
  curlog = stuff$loglikelihood
}
names(cur) = names(data)[-12]
res = rbind(res, c(i=i, curlog = curlog, cur = cur,step = step))
}
return(res)
}
beta_start = rep(0.02,11)
names(beta_start) = names(cancer_scale)[-12]

NR_result = NR(cancer_scale, beta_start)

NR_coef = NR_result[dim(NR_result)[1], -1:-2]

```

## validation using glm

```

cancer_fit = glm(response~., data = cancer_scale,family = binomial(link = "logit"))
summary(cancer_fit)

```

```

##
## Call:
## glm(formula = response ~ ., family = binomial(link = "logit"),
##      data = cancer_scale)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.95590  -0.14839  -0.03943   0.00429   2.91690
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    0.48702    0.56432   0.863  0.3881
## ones           NA           NA      NA      NA
## radius_mean    -7.22185    13.09494  -0.551  0.5813
## texture_mean     1.65476     0.27758   5.961 2.5e-09 ***
## perimeter_mean  -1.73763    12.27499  -0.142  0.8874
## area_mean       14.00485     5.89090   2.377  0.0174 *
## smoothness_mean  1.07495     0.44942   2.392  0.0168 *
## compactness_mean -0.07723     1.07434  -0.072  0.9427
## concavity_mean   0.67512     0.64733   1.043  0.2970
## `concave points_mean` 2.59287     1.10701   2.342  0.0192 *
## symmetry_mean    0.44626     0.29143   1.531  0.1257
## fractal_dimension_mean -0.48248     0.60406  -0.799  0.4244
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 751.44  on 568  degrees of freedom
## Residual deviance: 146.13  on 558  degrees of freedom
## AIC: 168.13
##
## Number of Fisher Scoring iterations: 9

```

```
# same results with NR
```

questions or modify:

1. normalize or standardize?
2. how to standardize easily?

```
auc = function(yi,pi_hat){
  auc = c()
  for (i in seq(dim(pi_hat)[2])){
    c = tibble(pi_hat = pi_hat[,i],yi = yi)
    m = sum(yi==1)
    n = length(yi)-m
    c =
      c %>%
      arrange(pi_hat) %>%
      mutate(order = seq_along(pi_hat)) %>%
      group_by(pi_hat) %>%
      mutate(mean_order = mean(order))
    pos_order = c %>% filter(yi == 1) %>% pull(mean_order)
    auc = c(auc,(sum(pos_order)-m*(m+1)/2)/(m*n))
  }

  return(auc)
}

yi = c(0,1,0,1,0,0,0,1)
pi = matrix(rep(rep(c(0.2,0.000001,0.3,0.4),2),2),ncol = 2 )
auc(yi,pi)
```

```
## [1] 0.7 0.7
```

```
#lambda_max = max(abs(NR_coef)) ## warm start
tuning_grid = seq(3, 0.01,length = 100) ## tuning seq
## function of lasso coordinate descent algorithm
# input:
# data: y -- binary response;
#       x -- scaled predictors
# k: # fold for CV
# beta_vec: initial beta guess
# tune_grid: seq of lambda used in variable selection
# tol: tolerance for stop iteration
# max: max iteration times

# output: res--result list containing:
# beta -- tibble:
#   rows: lambdas
#   columns:
#   beta1-beta10: beta estimation for every lambda
#   targ: target function value for every lambda iteration
#   times: times of iteration in each lambda iteration
# coeff -- list of final coefficient
# best_tune -- best tuning parameter lambda
# MSE_te -- test MSE results among cv
```

```
lasso_co_des = function(data, beta_vec, k = 5, tune_grid,tol = 1e-10, max = 200){
  ## create fold
  folds = createFolds(data$response,k = k, returnTrain = T)
  cv_result = c(k = 0,best_lambda = 0, beta_vec = beta_vec, auc_te = 0, g.stat_tr = Inf, g.stat_te = Inf,MSE_test =
```

```

## lasso coord des for train data
for (i in 1:k) {
  tr_rows = folds[[i]]
  train = data[tr_rows,]
  test = data[-tr_rows,]
  n_tr = dim(train)[1]
  x_tr = as.matrix(train[,1:dim(train)[2]-1]) # dim = (0.8*569, 11)
  y_tr = as.matrix(train[,dim(train)[2]]) # dim = (0.8*569, 1)
  x_te = as.matrix(test[,1:dim(test)[2]-1]) # dim = (0.2*569, 11)
  y_te = as.matrix(test[,dim(test)[2]]) # dim = (0.2*569, 1)
  res = c()
  ## for every lambda
  for (lambda in sort(tune_grid, decreasing = T)) {
    theta_vec = x_tr%*%beta_vec # dim = (455,1), theta for every obs
    pi_vec = exp(theta_vec)/(1+exp(theta_vec)) # dim = (455,1), pi for every obs
    w_vec = pi_vec*(1-pi_vec)+rep(1e-10,length(pi_vec)) # dim = (455,1), weight for every obs
    z_vec = theta_vec + (y_tr - pi_vec)/w_vec # dim = (455,1), working response for every obs
    w_res_vec = sqrt(w_vec)*(z_vec - theta_vec)
    #w_res_vec = w_res_vec[!is.nan(w_res_vec)]
    cur_target = (1/2*n_tr)*t((w_res_vec)%*%(w_res_vec) + lambda*sum(abs(beta_vec[-1])))
    pre_target = -Inf
    time = 0
    names(beta_vec) = paste("beta_",0:10, sep = "")
    res = rbind(res,c(k = k,lambda = lambda,time = time, cur_target = cur_target, beta_vec))
    while((abs(pre_target-cur_target)>tol) & time < max){
      time = time+1
      zero_index = seq(1,length(beta_start))
      pre_target = cur_target
      beta_pre = beta_vec
      for (j in 1:dim(train)[2]-1) {
        x_tr_s = x_tr[,zero_index]
        beta_pre_s = beta_pre[zero_index] ## sparse updating
        w_z_vec_j = as.vector(sqrt(w_vec)*(z_vec - x_tr_s%*%beta_pre_s + as.vector(x_tr[,j]*beta_pre[j])))
        #w_z_vec_j = w_z_vec_j[!is.nan(w_z_vec_j)]
        # dim(455,1),working response with out jth predictor
        w_x_tr_j = as.vector(as.vector(sqrt(w_vec)) * x_tr[,j])
        #w_x_tr_j = w_x_tr_j[!is.nan(w_x_tr_j)]
        ## dim(455,1), weighted obs on jth row
        tmp = as.numeric(t(w_x_tr_j) %*% w_z_vec_j)
        lambda_n = lambda*dim(x_tr)[1]
        if(j == 1){
          beta_pre[j] = tmp/sum(w_vec)
        }

        if (j>=2) {
          if (abs(tmp)>lambda_n) {
            if(tmp > 0) {tmp = tmp - lambda_n}
            else {tmp = tmp + lambda_n}
          }else{
            tmp = 0
            zero_index = zero_index[zero_index!=j]}
        }
        beta_pre[j] = tmp/(t(w_x_tr_j) %*% w_x_tr_j)
      }
      beta_vec = beta_pre
      theta_vec = x_tr%*%beta_vec # dim = (455,1), theta for every obs
      pi_vec = exp(theta_vec)/(1+exp(theta_vec)) # dim = (455,1), pi for every obs
      w_vec = pi_vec*(1-pi_vec)+rep(1e-10,length(pi_vec)) # dim = (455,1), weight for every obs
      z_vec = theta_vec + (y_tr - pi_vec)/w_vec
      # dim = (455,1), working response for every obs
      w_res_vec = sqrt(w_vec)*(z_vec - theta_vec)
    }
  }
}

```

```

w_res_vec = w_res_vec[!is.nan(w_res_vec)]
cur_target = (1/2*n_tr)*t((w_res_vec))%%(w_res_vec) +lambda*sum(abs(beta_vec[-1]))
res = rbind(res,c(k = k,lambda = lambda,time = time, cur_target = cur_target, beta_vec))
}
}
## choose lambda
res = as.tibble(res)
beta_lambda = res %>%
  group_by(lambda) %>%
  filter(cur_target == max(cur_target)) %>%
  dplyr::select(contains("beta"))
beta_lambda_m = as.matrix(beta_lambda[2:dim(beta_lambda)[2]]) # dim = (194, 11)
## use train dataset to choose lambda
pi_hat = exp(x_tr %*% t(beta_lambda_m))/(1+exp(x_tr %*% t(beta_lambda_m)))
residual_lambda = rep(y_tr,dim(beta_lambda_m)[1]) - pi_hat
MSE_tr = as.vector(rowSums(t(residual_lambda^2)))
## auc calculation
auc_tr = auc(y_tr,pi_hat = pi_hat)
g.res = (rep(y_tr,dim(beta_lambda_m)[1]) - pi_hat)/sqrt(pi_hat*(1-pi_hat))
g.stat_tr = as.vector(rowSums(t(g.res^2)))
best_result = as.tibble(cbind(beta_lambda,g.stat_tr = g.stat_tr, auc_tr = auc_tr, MSE_tr = MSE_tr)) %>%
  filter(auc_tr == max(auc_tr)) %>% filter(lambda == min(lambda))
best_result = best_result[1,]
# use test datasets to evaluate
best_result_m = as.matrix(best_result[2:(dim(best_result)[2]-3)])
pi_hat_te = exp(x_te %*% t(best_result_m))/(1+exp(x_te %*% t(best_result_m)))
residual_test = y_te - pi_hat_te
MSE_test = as.vector(rowSums(t(residual_test^2)))
auc_te = auc(y_te,pi_hat_te)
g.res_te = (y_te - pi_hat_te)/sqrt(pi_hat_te*(1-pi_hat_te))
g.stat_te = as.vector(rowSums(t(g.res_te^2)))

cv_result = rbind(cv_result, c(k = i,best_lambda = best_result$lambda, beta_vec = best_result[2:dim(best_result)
#cv_result = as.tibble(cv_result)
}

return(cv_result)
}
x = lasso_co_des(data = cancer_scale, beta_vec = rep(0.02,11), k = 5,tune_grid = tuning_grid,max = 200)

knitr::kable(x)

```

|           | k | best_lambda | beta_vec.lambda | beta_vec.beta_0    | beta_vec.beta_1   | beta_vec.beta_2   | beta_vec.beta_3   |
|-----------|---|-------------|-----------------|--------------------|-------------------|-------------------|-------------------|
| cv_result | 0 | 0           | 0.02            | 0.02               | 0.02              | 0.02              | 0.02              |
|           | 1 | 0.01        | 0.01            | -0.641325351498482 | 2.00490580294464  | 0.96659993964192  | 0                 |
|           | 2 | 0.01        | 0.01            | -0.635976660876175 | 1.64938907491227  | 1.04242992558731  | 0.29018081246531  |
|           | 3 | 0.01        | 0.01            | -0.681991726573321 | 0.259343655049853 | 0.420948748142398 | 0.597291896107064 |
|           | 4 | 0.01        | 0.01            | -0.668832346798737 | 1.17263206368449  | 0.875000215937718 | 1.01679089311367  |
|           | 5 | 0.01        | 0.01            | -0.526584086068486 | 0.814117533052765 | 0.441300815527579 | 0                 |

```

# cleaning the above x
library(sjmisc)
y=as.data.frame(x)
y_y=rotate_df(y)
names(y_y)=c("Enter", "Fold1", "Fold2", "Fold3", "Fold4", "Fold5")
knitr::kable(y_y)

```

|   | Enter | Fold1 | Fold2 | Fold3 | Fold4 | Fold5 |
|---|-------|-------|-------|-------|-------|-------|
| k | 0     | 1     | 2     | 3     | 4     | 5     |

|                    | Enter | Fold1              | Fold2              | Fold3              | Fold4              | Fold5           |
|--------------------|-------|--------------------|--------------------|--------------------|--------------------|-----------------|
| best_lambda        | 0     | 0.01               | 0.01               | 0.01               | 0.01               | 0.01            |
| beta_vec.lambda    | 0.02  | 0.01               | 0.01               | 0.01               | 0.01               | 0.01            |
| beta_vec.beta_0    | 0.02  | -0.641325351498482 | -0.635976660876175 | -0.681991726573321 | -0.668832346798737 | -0.526584086068 |
| beta_vec.beta_1    | 0.02  | 2.00490580294464   | 1.64938907491227   | 0.259343655049853  | 1.17263206368449   | 0.8141175330527 |
| beta_vec.beta_2    | 0.02  | 0.96659993964192   | 1.04242992558731   | 0.420948748142398  | 0.875000215937718  | 0.4413008155275 |
| beta_vec.beta_3    | 0.02  | 0                  | 0.29018081246531   | 0.597291896107064  | 1.01679089311367   | 0               |
| beta_vec.beta_4    | 0.02  | 0.0133529652373485 | 0.0484707378078863 | 0                  | 0.0378341623885164 | 0               |
| beta_vec.beta_5    | 0.02  | 0.32670996983099   | 0.265865681362969  | 0                  | 0.207638745339431  | 0               |
| beta_vec.beta_6    | 0.02  | 0                  | 0                  | 0                  | 0                  | 0               |
| beta_vec.beta_7    | 0.02  | 0.0917876099116541 | 0.242067059495091  | 0                  | 0.53791592632816   | 0               |
| beta_vec.beta_8    | 0.02  | 2.21196517334801   | 1.99835020021248   | 1.82524888522456   | 1.73750089255878   | 2.0230230352962 |
| beta_vec.beta_9    | 0.02  | 0.0754333811390448 | 0.160127684252132  | 0                  | 0.156340291925062  | 0               |
| beta_vec.beta_10   | 0     | 0                  | 0                  | 0                  | 0                  | 0               |
| beta_vec.g.stat_tr | Inf   | 145.172270554506   | 144.378997062778   | 137.928322464409   | 144.748286643231   | 129.57235040108 |
| auc_te             | Inf   | 0.992036011080332  | 0.99198128967591   | 0.985266457680251  | 0.981648199445983  | 0.9743769743769 |
| g.stat_tr          | Inf   | 145.172270554506   | 144.378997062778   | 137.928322464409   | 144.748286643231   | 129.57235040108 |
| g.stat_te          | 0     | 22.1139370275807   | 22.8895393351494   | 40.1676991492816   | 41.6094246512014   | 38.977546958474 |
| MSE_test           | 0     | 4.22683743241178   | 4.15181974978863   | 6.7079943143784    | 6.28990827099095   | 7.0252779009262 |

instead of using MSE, using pearson chi-square

validation

```
# x.mat <- model.matrix(diagnosis~., cancer_package[-1])[, -1]
# y.class <- cancer_package$diagnosis
#
# ctrl1 <- trainControl(method = "cv", number = 5)
# lasso.fit <- train(x.mat, y.class,
#                   method = "glmnet",
#                   tuneGrid = expand.grid(alpha = 1,
#                                           lambda = tuning_grid),
#                   # preProc = c("center", "scale"),
#                   trControl = ctrl1)
#
# lasso.fit$bestTune
# plot(lasso.fit)
# min(lasso.fit$results$RMSE)
# co=coef(lasso.fit$finalModel, lasso.fit$bestTune$lambda)
# co2=co@x
#
# names(co2)=co@Dimnames[[1]]
# co2 %>% as.data.frame() %>% knitr::kable()
```