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!!
```

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

```
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
```

```r
    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_coeff =  NR_result[dim(NR_result)[1], -1:-2]
```

validation using glm

```r
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_coeff)) ## warm start
tuning_grid = seq(3, 0,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, g.stat_tr = Inf,auc_te = 0,  g.stat_te = Inf,MSE_test =
```

```r
## 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)
```

```r
        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)))
  result_tr = as.tibble(cbind(beta_lambda,g.stat_tr = g.stat_tr,auc_tr = auc_tr,MSE_tr = MSE_tr))
  best_result =  result_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(list(cv_result = unnest(as.tibble(cv_result)),res = res, result_tr= result_tr , best_result = best_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$cv_result,digits = 3)
```

k	best_lambda	beta_vec1	beta_vec2	beta_vec3	beta_vec4	beta_vec5	beta_vec6	beta_vec7	beta_vec8	beta_vec9
0	0	0.020	0.020	0.020	0.020	0.020	0.020	0.020	0.020	0.020
1	0	-0.686	2.470	1.542	0.109	0.607	1.083	-0.522	1.089	2.192
2	0	-0.685	2.268	1.646	0.131	0.770	0.984	-0.335	1.250	1.963
3	0	-0.520	2.444	1.696	0.093	2.069	1.333	-1.566	2.319	1.621
4	0	-0.711	1.668	1.540	1.191	0.899	0.977	-0.251	1.570	1.467
5	0	-0.490	2.955	1.892	0.155	0.823	1.670	-0.810	0.964	2.703

```r
x
```

```
## $cv_result
## # A tibble: 6 x 17
##       k best_lambda beta_vec1 beta_vec2 beta_vec3 beta_vec4 beta_vec5 beta_vec6
##   <dbl>       <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1     0           0     0.02       0.02      0.02    0.02        0.02      0.02
## 2     1           0    -0.686      2.47      1.54    0.109       0.607     1.08
## 3     2           0    -0.685      2.27      1.65    0.131       0.770     0.984
```

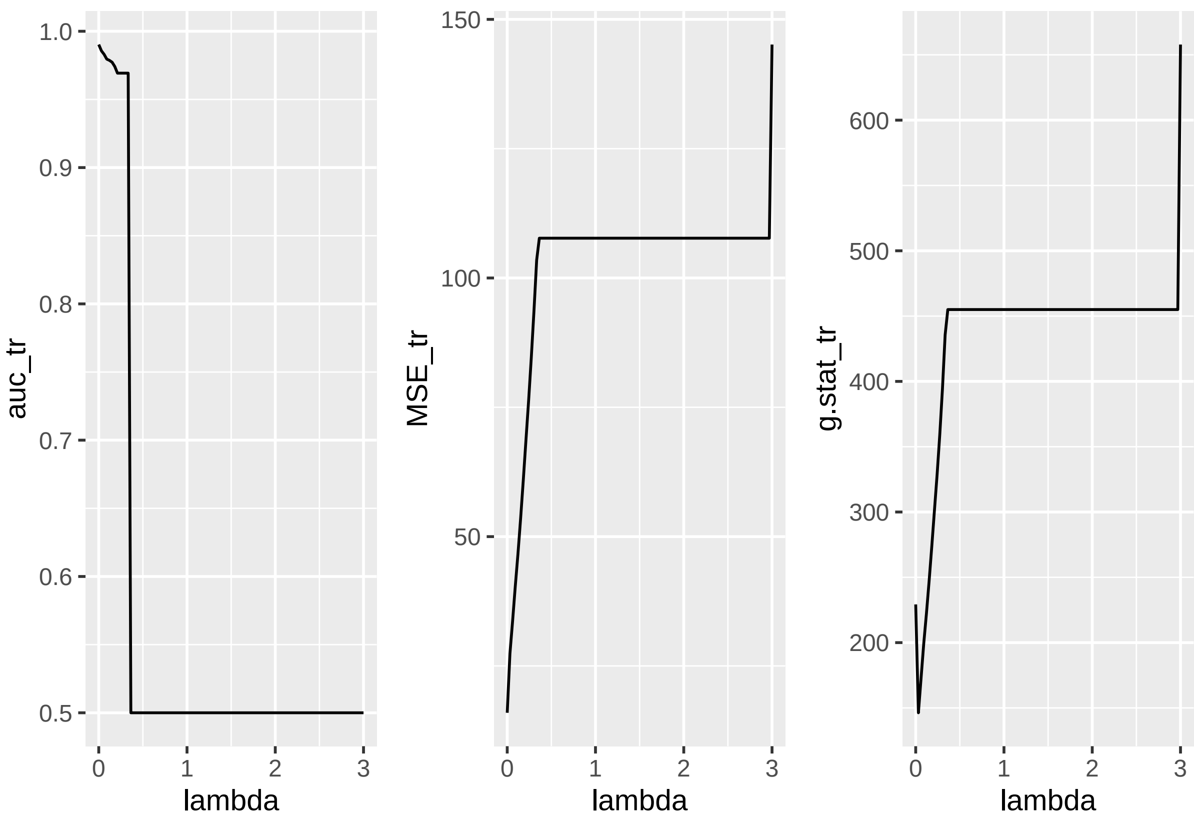
```
## 4     3           0    -0.520      2.44      1.70    0.0934     2.07      1.33
## 5     4           0    -0.711      1.67      1.54    1.19       0.899     0.977
## 6     5           0    -0.490      2.95      1.89    0.155      0.823     1.67
## # ... with 9 more variables: beta_vec7 <dbl>, beta_vec8 <dbl>, beta_vec9 <dbl>,
## #   beta_vec10 <dbl>, beta_vec11 <dbl>, g.stat_tr <dbl>, auc_te <dbl>,
## #   g.stat_te <dbl>, MSE_test <dbl>
##
## $res
## # A tibble: 1,031 x 15
##        k lambda  time cur_target beta_0 beta_1 beta_2 beta_3 beta_4 beta_5
##    <dbl>  <dbl> <dbl>      <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1     5   3        0     53549.  0.436  -9.85   1.52   1.45   13.9   1.01
##  2     5   3        1    149674.  0.715   0      0      0       0     0
##  3     5   3        2    106969. -0.586   0      0      0       0     0
##  4     5   3        3    103470. -0.468   0      0      0       0     0
##  5     5   3        4    103512. -0.470   0      0      0       0     0
##  6     5   3        5    103512. -0.470   0      0      0       0     0
##  7     5   3        6    103512. -0.470   0      0      0       0     0
##  8     5   3        7    103512. -0.470   0      0      0       0     0
##  9     5   2.97     0    103512. -0.470   0      0      0       0     0
## 10     5   2.97     1    103512. -0.470   0      0      0       0     0
## # ... with 1,021 more rows, and 5 more variables: beta_6 <dbl>, beta_7 <dbl>,
## #   beta_8 <dbl>, beta_9 <dbl>, beta_10 <dbl>
##
## $result_tr
## # A tibble: 186 x 15
##    lambda beta_0 beta_1 beta_2 beta_3 beta_4 beta_5 beta_6 beta_7 beta_8 beta_9
##     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1   3     0.715      0      0      0      0      0      0      0      0      0
##  2   2.97 -0.470      0      0      0      0      0      0      0      0      0
##  3   2.97 -0.470      0      0      0      0      0      0      0      0      0
##  4   2.94 -0.470      0      0      0      0      0      0      0      0      0
##  5   2.94 -0.470      0      0      0      0      0      0      0      0      0
##  6   2.91 -0.470      0      0      0      0      0      0      0      0      0
##  7   2.91 -0.470      0      0      0      0      0      0      0      0      0
##  8   2.88 -0.470      0      0      0      0      0      0      0      0      0
##  9   2.88 -0.470      0      0      0      0      0      0      0      0      0
## 10   2.85 -0.470      0      0      0      0      0      0      0      0      0
## # ... with 176 more rows, and 4 more variables: beta_10 <dbl>, g.stat_tr <dbl>,
## #   auc_tr <dbl>, MSE_tr <dbl>
##
## $best_result
## # A tibble: 1 x 15
##   lambda beta_0 beta_1 beta_2 beta_3 beta_4 beta_5 beta_6 beta_7 beta_8 beta_9
##    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1      0 -0.490   2.95   1.89  0.155  0.823   1.67 -0.810  0.964   2.70  0.546
## # ... with 4 more variables: beta_10 <dbl>, g.stat_tr <dbl>, auc_tr <dbl>,
## #   MSE_tr <dbl>
```

```r
g1 = x$result_tr %>%
  ggplot(aes(x = lambda, y = auc_tr))+
  geom_line()

g2 = x$result_tr %>%
  ggplot(aes(x = lambda, y = MSE_tr))+
  geom_line()

g3 = x$result_tr %>%
  ggplot(aes(x = lambda, y = g.stat_tr))+
  geom_line()
```

```r
g1+g2+g3
```




```r
# cleaning the above x
library(sjmisc)
y=as.data.frame(x$cv_result)
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	0.00	1.0000000	2.0000000	3.0000000	4.0000000	5.0000000
best_lambda	0.00	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
beta_vec1	0.02	-0.6859925	-0.6846651	-0.5202080	-0.7108130	-0.4904983
beta_vec2	0.02	2.4700201	2.2680094	2.4442414	1.6678653	2.9547735
beta_vec3	0.02	1.5423244	1.6459899	1.6960535	1.5396684	1.8918639
beta_vec4	0.02	0.1086057	0.1309746	0.0933700	1.1913073	0.1548061
beta_vec5	0.02	0.6066107	0.7695696	2.0689442	0.8991037	0.8233475
beta_vec6	0.02	1.0825592	0.9841494	1.3327553	0.9768157	1.6699939
beta_vec7	0.02	-0.5217764	-0.3350776	-1.5663942	-0.2512495	-0.8098719
beta_vec8	0.02	1.0885347	1.2501449	2.3188669	1.5697889	0.9636958
beta_vec9	0.02	2.1922951	1.9632482	1.6208382	1.4671648	2.7033725
beta_vec10	0.02	0.4242812	0.5513924	0.5472317	0.5297819	0.5455744
beta_vec11	0.02	-0.4955606	-0.6031343	-0.0903676	-0.4954602	-0.2339802
g.stat_tr	Inf	238.9887331	270.9964837	166.3436015	344.1339201	229.2555310
auc_te	0.00	0.9934211	0.9916472	0.9811912	0.9844183	0.9747280
g.stat_te	Inf	20.9787985	22.6434852	241.7250075	55.0375829	119.9943143
MSE_test	Inf	3.5422803	3.9095202	5.6402349	5.6143326	8.2576478

instead of using MSE, using pearson chi-square

validation

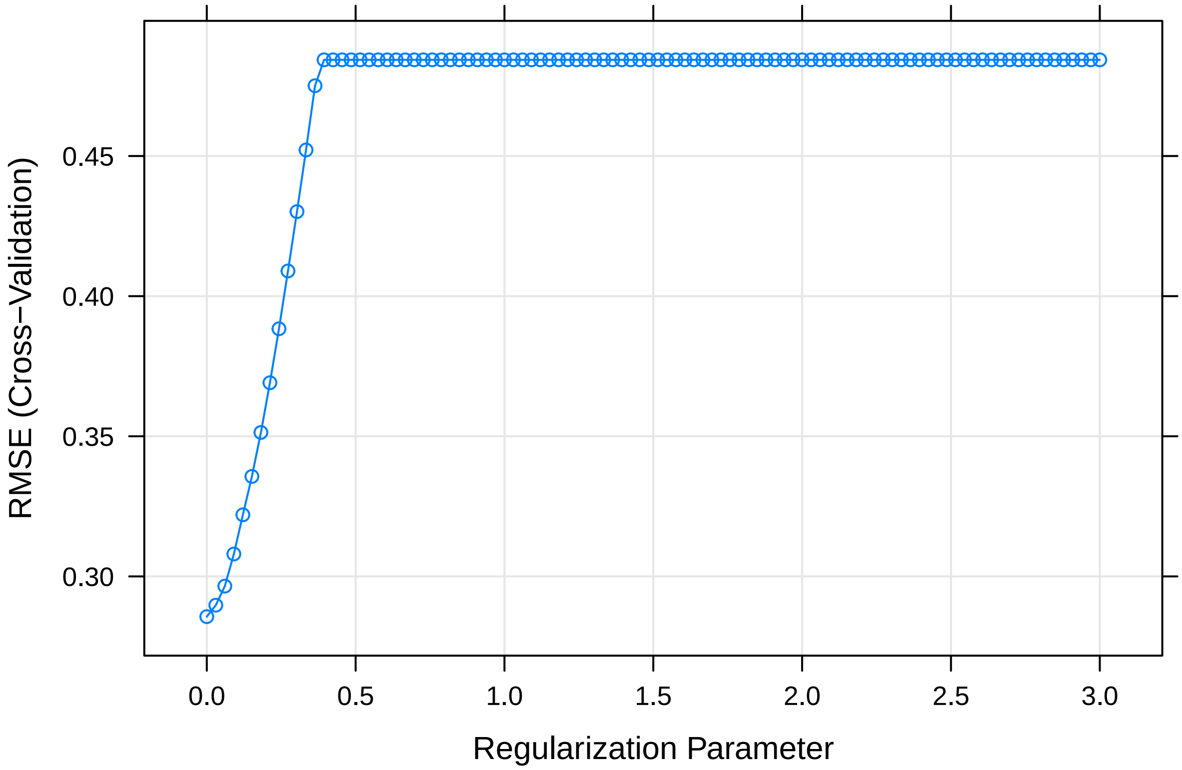
```r
x.mat <- model.matrix(diagnosis~., cancer_package[-1])[,-1]
y.class <- cancer_package$diagnosis
```

```r
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
```

```
##   alpha lambda
## 1     1      0
```

```r
plot(lasso.fit)
```




```r
# 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()
```