logistic

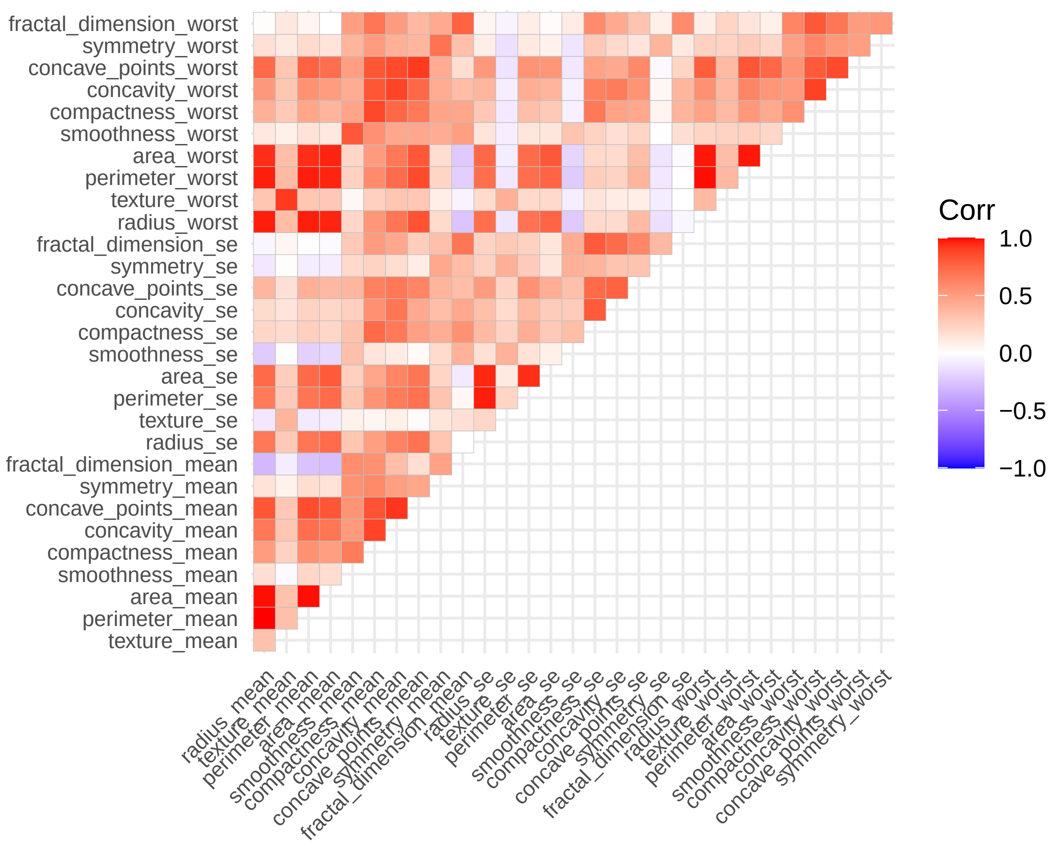
Xinran Sun, Haotian Wu, Lin Yang, Shengzhi Luo

3/17/2022

```
ggplot2::theme_set(theme_minimal() + theme(legend.position = "bottom"))
```

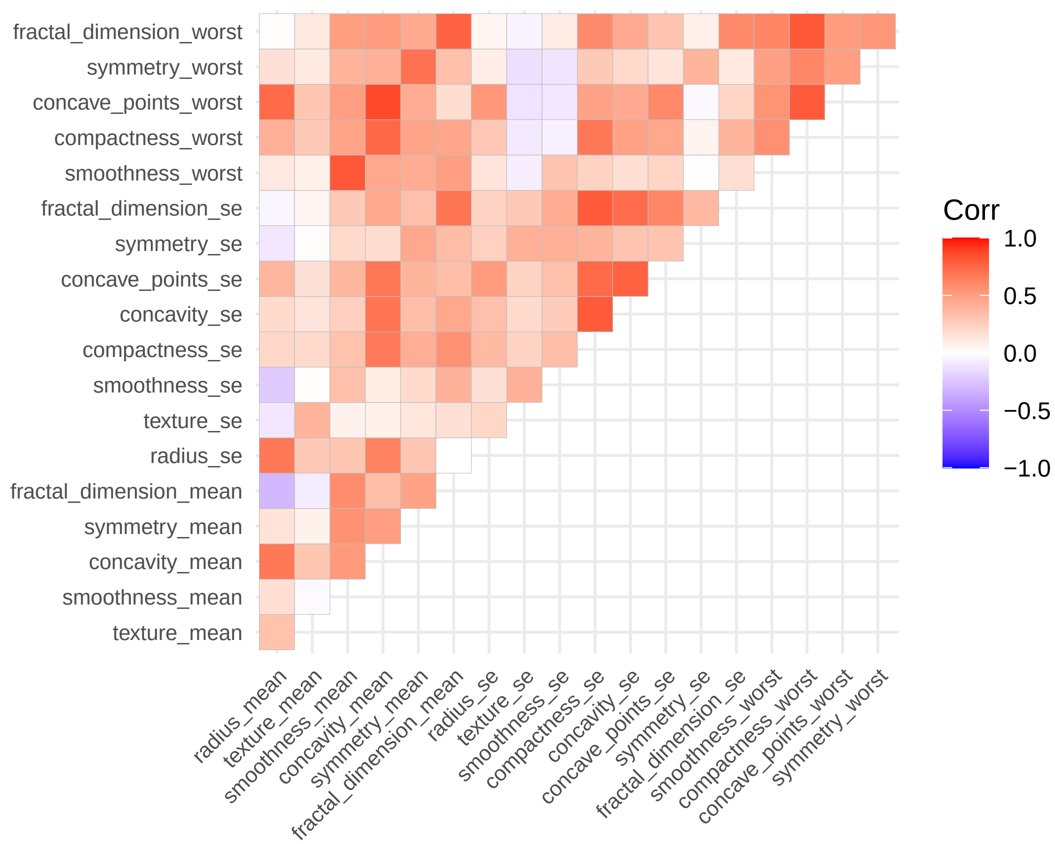
data import and data clean

```
#load the data
breast = read.csv("breast-cancer.csv") %>%
  janitor::clean_names() %>%
  dplyr::select(-1, -33) %>% #drop id and NA columns
  mutate(diagnosis = recode(diagnosis, "M" = 1, "B" = 0))
#check collinearity
corr = breast[2:31] %>%
  cor()
ggcorrplot(corr, type = "upper", tl.cex = 8)
```




```
#remove some highly correlated variables
breast_dat <- breast %>% dplyr::select(-area_se, -perimeter_se, -area_worst, -perimeter_mean, -perimete

corr1 = breast_dat[2:20] %>%
  cor()
ggcorrplot(corr1, type = "upper", tl.cex = 8)
```




```r
#partition data into training and test data
set.seed(2022)
trainRows <- createDataPartition(y = breast_dat$diagnosis, p = 0.8, list = FALSE)
breast_train <- breast_dat[trainRows, ]
breast_test <-  breast_dat[-trainRows, ]

head(breast_dat, 5)
```

```
##   diagnosis radius_mean texture_mean smoothness_mean concavity_mean
## 1         1       17.99        10.38         0.11840         0.3001
## 2         1       20.57        17.77         0.08474         0.0869
## 3         1       19.69        21.25         0.10960         0.1974
## 4         1       11.42        20.38         0.14250         0.2414
## 5         1       20.29        14.34         0.10030         0.1980
##   symmetry_mean fractal_dimension_mean radius_se texture_se smoothness_se
## 1        0.2419                0.07871    1.0950     0.9053      0.006399
## 2        0.1812                0.05667    0.5435     0.7339      0.005225
## 3        0.2069                0.05999    0.7456     0.7869      0.006150
## 4        0.2597                0.09744    0.4956     1.1560      0.009110
## 5        0.1809                0.05883    0.7572     0.7813      0.011490
##   compactness_se concavity_se concave_points_se symmetry_se
## 1        0.04904      0.05373           0.01587     0.03003
## 2        0.01308      0.01860           0.01340     0.01389
## 3        0.04006      0.03832           0.02058     0.02250
## 4        0.07458      0.05661           0.01867     0.05963
## 5        0.02461      0.05688           0.01885     0.01756
```

```
##   fractal_dimension_se smoothness_worst compactness_worst concave_points_worst
## 1             0.006193           0.1622            0.6656               0.2654
## 2             0.003532           0.1238            0.1866               0.1860
## 3             0.004571           0.1444            0.4245               0.2430
## 4             0.009208           0.2098            0.8663               0.2575
## 5             0.005115           0.1374            0.2050               0.1625
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
```

```r
r = dim(breast_dat)[1] #row number
c = dim(breast_dat)[2] #column number
var_names = names(breast_dat)[-c(1,2)] #variable names

standardize = function(col) {
  mean = mean(col)
  sd = sd(col)
  return((col - mean)/sd)
}
stand_df = breast_dat %>%
  dplyr::select(radius_mean:fractal_dimension_worst) %>%
  map_df(.x = ., standardize) #standardize
X = stand_df #predictors
y = breast_dat[,1]#response
```

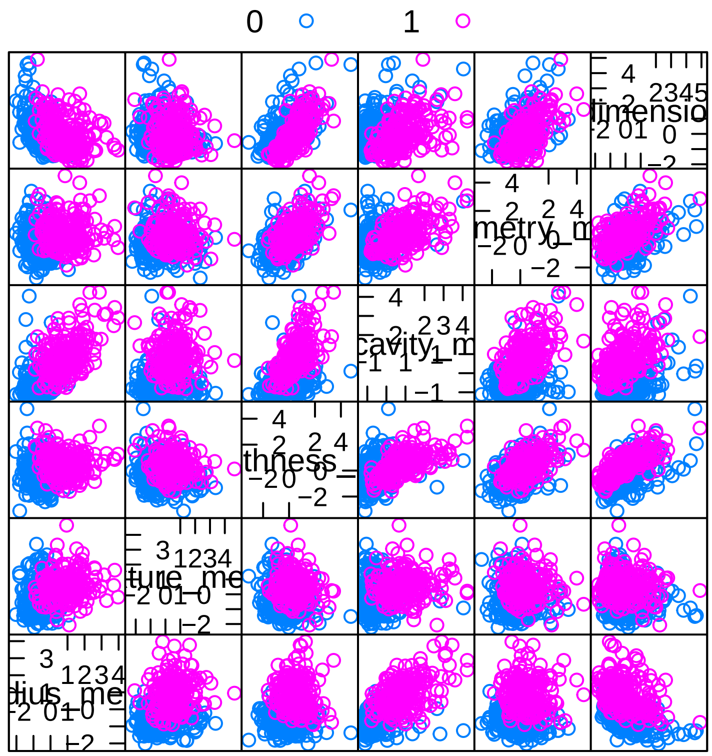
```r
x_train <- breast_train[2:20] #predictors
y_train <- breast_train[1] #response
x_train_stan <- cbind(rep(1, nrow(x_train)), scale(x_train))

x_test <- breast_test[2:20]
x_test_stan <- cbind(rep(1, nrow(x_test)), scale(x_test))
```

feature plot

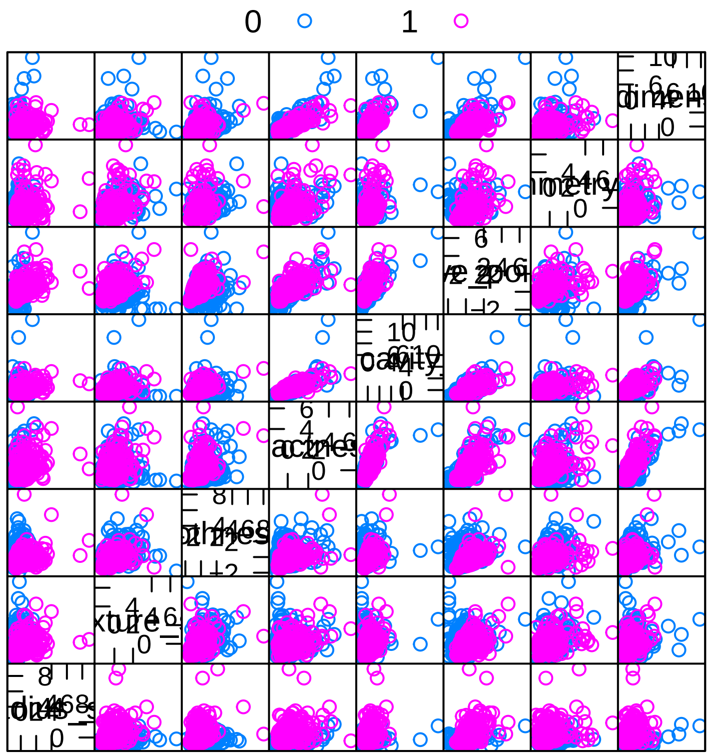
```r
data = cbind(y,X)

featurePlot(x = data[, 2:7],
            y = factor(data$y),
            plot = "pairs",
            auto.key = list(columns = 2)
)
```



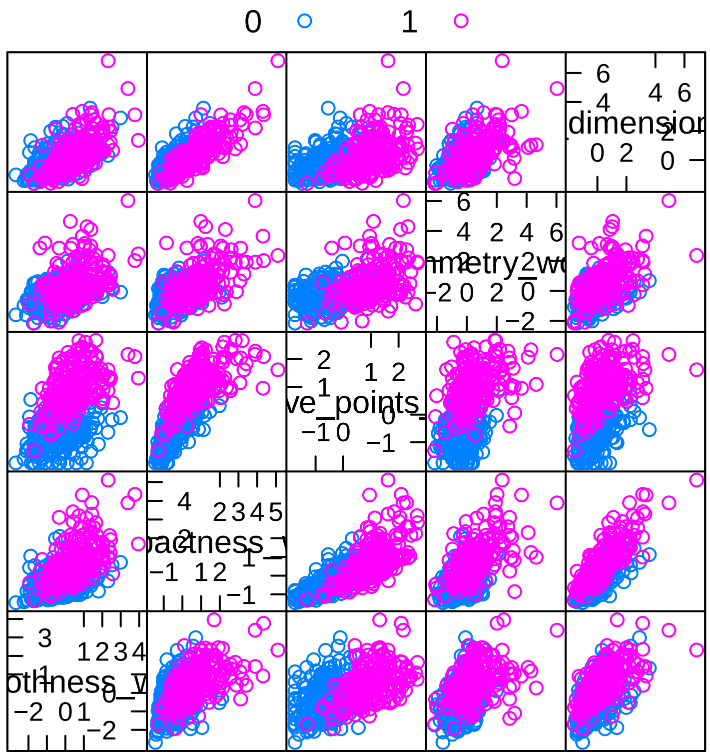
Scatter Plot Matrix

```r
featurePlot(x = data[, 8:15],
            y = factor(data$y),
            plot = "pairs",
            auto.key = list(columns = 2)
)
```



Scatter Plot Matrix

```r
featurePlot(x = data[, 16:20],
            y = factor(data$y),
            plot = "pairs",
            auto.key = list(columns = 2)
)
```



Scatter Plot Matrix

```
mean_data = breast_dat %>%
  group_by(diagnosis) %>%
  summarise(across(radius_mean: fractal_dimension_worst, ~ mean(.x, na.rm = TRUE)))
mean_data
```

```
## # A tibble: 2 x 20
##   diagnosis radius_mean texture_mean smoothness_mean concavity_mean
##       <dbl>       <dbl>        <dbl>           <dbl>          <dbl>
## 1         0        12.1         17.9          0.0925         0.0461
## 2         1        17.5         21.6          0.103          0.161
## # ... with 15 more variables: symmetry_mean <dbl>,
## #   fractal_dimension_mean <dbl>, radius_se <dbl>, texture_se <dbl>,
## #   smoothness_se <dbl>, compactness_se <dbl>, concavity_se <dbl>,
## #   concave_points_se <dbl>, symmetry_se <dbl>, fractal_dimension_se <dbl>,
## #   smoothness_worst <dbl>, compactness_worst <dbl>,
## #   concave_points_worst <dbl>, symmetry_worst <dbl>,
## #   fractal_dimension_worst <dbl>
```

Full logistic model

```
glm.fit <- glm(diagnosis ~ .,
               data = breast_train,
               family = binomial)
```

```r
summary(glm.fit)$coefficients %>% knitr::kable()
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-61.1480991	22.7005353	-2.6936853	0.0070667
radius_mean	0.4087734	0.5194231	0.7869758	0.4312960
texture_mean	0.7991309	0.2960648	2.6991761	0.0069511
smoothness_mean	112.0773037	108.6087742	1.0319360	0.3021021
concavity_mean	81.0558072	35.4365649	2.2873494	0.0221754
symmetry_mean	-74.1111829	39.6817182	-1.8676405	0.0618122
fractal_dimension_mean	-344.5973095	228.6957280	-1.5067938	0.1318635
radius_se	39.6728660	14.0741918	2.8188379	0.0048198
texture_se	-0.4026481	1.5678789	-0.2568107	0.7973249
smoothness_se	442.3410192	418.9499688	1.0558326	0.2910447
compactness_se	380.5961088	185.4664743	2.0521019	0.0401598
concavity_se	-74.9595207	51.3448406	-1.4599231	0.1443112
concave_points_se	-210.2627737	404.8257177	-0.5193909	0.6034882
symmetry_se	-486.7748560	225.4609542	-2.1590207	0.0308486
fractal_dimension_se	-3184.3013759	1568.2807496	-2.0304409	0.0423117
smoothness_worst	-41.9855490	75.0498013	-0.5594358	0.5758643
compactness_worst	-72.5516143	28.7121732	-2.5268590	0.0115088
concave_points_worst	144.8910643	66.8810152	2.1664005	0.0302806
symmetry_worst	80.0311702	29.5678265	2.7066978	0.0067956
fractal_dimension_worst	480.1713745	207.0899338	2.3186611	0.0204134

```r
glm.fit %>% predict(breast_test, type = "response")
```

```
##            14            21            27            28            30            43 
## 4.435159e-01 1.115467e-08 1.000000e+00 1.000000e+00 9.999140e-01 1.000000e+00 
##            50            52            60            62            68            71 
## 1.085158e-01 2.823488e-06 8.386401e-10 2.385735e-07 1.174097e-06 1.000000e+00 
##            75            87            88            90            98            99 
## 5.368857e-05 9.796329e-01 1.000000e+00 9.793390e-01 8.378933e-09 3.892338e-06 
##           100           108           109           116           125           126 
## 9.542554e-01 1.290556e-06 1.000000e+00 2.117744e-04 5.907608e-09 9.177100e-08 
##           128           135           141           149           152           164 
## 9.999965e-01 1.000000e+00 3.389340e-13 6.584321e-02 2.765556e-08 6.483175e-05 
##           165           171           180           183           192           196 
## 1.000000e+00 1.121857e-07 6.303818e-10 9.999949e-01 3.618526e-08 1.294629e-06 
##           198           199           212           213           217           222 
## 9.997821e-01 9.999741e-01 6.334345e-07 1.000000e+00 4.787718e-06 9.813174e-06 
##           237           241           244           249           250           258 
## 1.000000e+00 9.670721e-05 4.672950e-04 4.301725e-03 2.809702e-06 1.000000e+00 
##           261           264           265           275           284           292 
## 1.000000e+00 2.147741e-03 1.000000e+00 9.999649e-01 9.999801e-01 7.142970e-01 
##           294           300           312           317           320           323 
## 8.644211e-06 1.585715e-10 8.593789e-06 1.689966e-12 1.011243e-13 6.278814e-05 
##           324           325           327           332           333           343 
## 1.000000e+00 3.471332e-07 5.700384e-08 8.543291e-05 7.596965e-11 2.641163e-05 
##           349           354           356           357           358           360 
## 6.191597e-06 1.000000e+00 3.071796e-05 3.943388e-05 7.153699e-07 1.343892e-03 
##           364           377           386           394           398           408
```

```
## 1.615436e-02 7.813589e-08 9.993107e-01 1.000000e+00 3.386318e-06 4.994989e-07
##          413          418          421          434          439          440
## 9.538364e-11 1.000000e+00 3.659893e-06 1.000000e+00 2.240033e-05 2.328171e-07
##          441          444          453          456          458          459
## 1.268173e-03 2.220446e-16 6.978071e-05 8.146849e-01 2.043044e-04 1.468687e-04
##          461          478          479          481          482          484
## 1.000000e+00 7.212287e-09 8.951327e-07 3.716606e-08 3.215714e-02 1.219353e-04
##          491          495          496          519          520          525
## 1.468055e-03 1.425063e-06 1.498459e-03 1.100079e-02 2.512633e-04 1.379163e-07
##          528          538          540          543          546          558
## 5.401006e-06 2.400588e-02 3.067226e-07 8.691250e-02 3.067696e-03 7.608324e-06
##          559          564          565          568          569
## 2.577921e-04 1.000000e+00 1.000000e+00 1.000000e+00 1.033328e-08
```

```r
pred <- predict(glm.fit, breast_test, type = "response")
y_test <- factor(breast_test$diagnosis)
auc_full <- auc(y_test, pred)
auc_full
```

```
## Area under the curve: 0.994
```

Newton-Raphson algorithm

```r
## logistic stuff
#logisticstuff = function(dat, betavec){
#
#  x = dat[, -1] %>% as.matrix()
#  x = cbind(rep(1, nrow(x)), scale(x))
#  y = as.matrix(dat[, 1])
#  theta = x %*% betavec
#  p = exp(theta) / (1 + exp(theta))
#
#  loglik = sum(y * theta - log(1 + exp(theta)))
#  grad = t(x) %*% (y - p) # gradient
#
# # w = p * (1 - p)
# # w = diag(as.vector(w), nrow = nrow(w))
# # print(w)
#  hess = -(t(x) %*% diag(as.vector(p * (1 - p))) %*% x) # hessian matrix
#
#  return(list(loglik = loglik, grad = grad, hess = hess))
#}
#beta = rep(1, 20)
#test = logisticstuff(breast_train, betavec = beta)
#test$loglik
```

```r
#newtonraphson = function(dat, func, start, tol = 1e-10, maxiter = 200){
#  i = 0
#  curbeta = start
#  stuff = func(dat, curbeta)
#  res = c(0, stuff$loglik, curbeta)
```

```
#  prevloglik = -Inf
#
#  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol && (!is.na(stuff$loglik)) ) {
#    i = i + 1
#    prevloglik = stuff$loglik
#    prev = curbeta
#    curbeta  = prev - solve(stuff$hess) %*% stuff$grad
#    stuff = func(dat, curbeta)
#
#    eigen_vals = eigen(stuff$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))*diag(nrow(stuff$hess))
#    }
#
#    curbeta  = prev - solve(stuff$hess) %*% stuff$grad
#    stuff = func(dat, curbeta)
#
#    j = 1
#    # half step
#      while (stuff$loglik < prevloglik && (!is.na(stuff$loglik)) ) {
#       # stuff <- func(dat, curbeta)
#        j  <- j / 2
#        curbeta   <- prev - j * solve(stuff$hess) %*% stuff$grad
#        stuff <- func(dat, curbeta)
#      }
#    curbeta = prev - solve(stuff$hess) %*% stuff$grad
#    stuff = func(dat, curbeta)
#    #redirection
#    j = 1
#    while (stuff$loglik < prevloglik) {
#
#
#      if (!all(eigen(stuff$hess)$values) < 0) {
#      #gamma = max(eigen(stuff$hess)$values)
#      new_hess = stuff$hess - 0.1*diag(20)
#      curbeta = prev - solve(new_hess) %*% stuff$grad
#      }
#      else {
#      j = j/2
#      curbeta = prev - j * solve(stuff$hess) %*% stuff$grad
#      }
#
#    }
#
#  #  #redirection
#  #  j = 1
#  #  while (stuff$loglik < prevloglik) {
#  #
#  #    if (!all(eigen(stuff$hess)$values) < 0) {
#  #    #gamma = max(eigen(stuff$hess)$values)
#  #    new_hess = stuff$hess - diag(31)
```

```
# #    curbeta = prev - solve(new_hess) %*% stuff$grad
# #    }
# #    else {
# #    j = j/2
# #    curbeta = prev - j * solve(stuff$hess) %*% stuff$grad
# #    }
# #  }
# #
# #  stuff = func(dat, curbeta)
#    res = rbind(res, c(i, stuff$loglik, curbeta))
#  }
#  return(res)
#}
#res = newtonraphson(breast_train, logisticstuff, beta)
```

coordinate-wise optimization of a logistic-lasso model

```
x_train <- breast_train[2:20] #predictors
y_train <- breast_train[1] #response
x_train_stan <- cbind(rep(1, nrow(x_train)), scale(x_train))
x_test <- breast_test[2:20]
y_test <- breast_test[1]
```

```
#soft threshold
sfxn <- function(beta, lambda) {
  if (abs(beta) > lambda) {
    return(sign(beta) * (abs(beta) - lambda))
  }
  else {
    return(0)
  }
}
```

```
#coordinate-wise optimization function
coordwise_lasso <- function(lambda, x, y, betastart, tol = exp(-10), maxiter = 5000) {
  i <- 0
  n <- length(y)
  pnum <- length(betastart)
  betavec <- betastart
  loglik <- 0
  res <- c(0, loglik, betavec)
  prevloglik <- -Inf
  while (i < maxiter & abs(loglik - prevloglik) > tol & loglik < Inf) {
    i <- i + 1
    prevloglik <- loglik
    for (j in 1:pnum) {
      theta <- x %*% betavec
      p <- exp(theta) / (1 + exp(theta)) #probability of malignant cases
      w <- p*(1-p) #working weights
      w <- ifelse(abs(w-0) < 1e-5, 1e-5, w)
      z <- theta + (y - p)/w #working response
```

```r
      zwoj <- x[, -j] %*% betavec[-j]
      betavec[j] <- sfxn(sum(w*(x[,j])*(z - zwoj)), lambda) / (sum(w*x[,j]*x[,j]))
    }
    theta <- x %*% betavec
    p <- exp(theta) / (1 + exp(theta)) #probability of malignant cases
    w <- p*(1-p) #working weights
    w <- ifelse(abs(w-0) < 1e-10, 1e-10, w)
    z <- theta + (y - p)/w
    loglik <- sum(w*(z - theta)^2) / (2*n) + lambda * sum(abs(betavec))
    res <- rbind(res, c(i, loglik, betavec))
  }
  return(res)
}
#coordwise_res <- coordwise_lasso(lambda = 0.006, x_train_stan, y_train, betastart = rep(0, #20))
#coordwise_res[nrow(coordwise_res), ]
```

We need to calculate lambdamax first to define a sequence of lambda.

```r
x.matrix <- scale(x_train) %>% as.matrix()
y.matrix <- as.matrix(y_train)
lambdamax <- max(abs(t(x.matrix) %*% y.matrix)) #/ nrow(y.matrix)
lambda_seq1 <- exp(seq(log(lambdamax), -5, length = 50))
lambda_seq2 <- exp(seq(log(lambdamax), -5, length = 50))
```

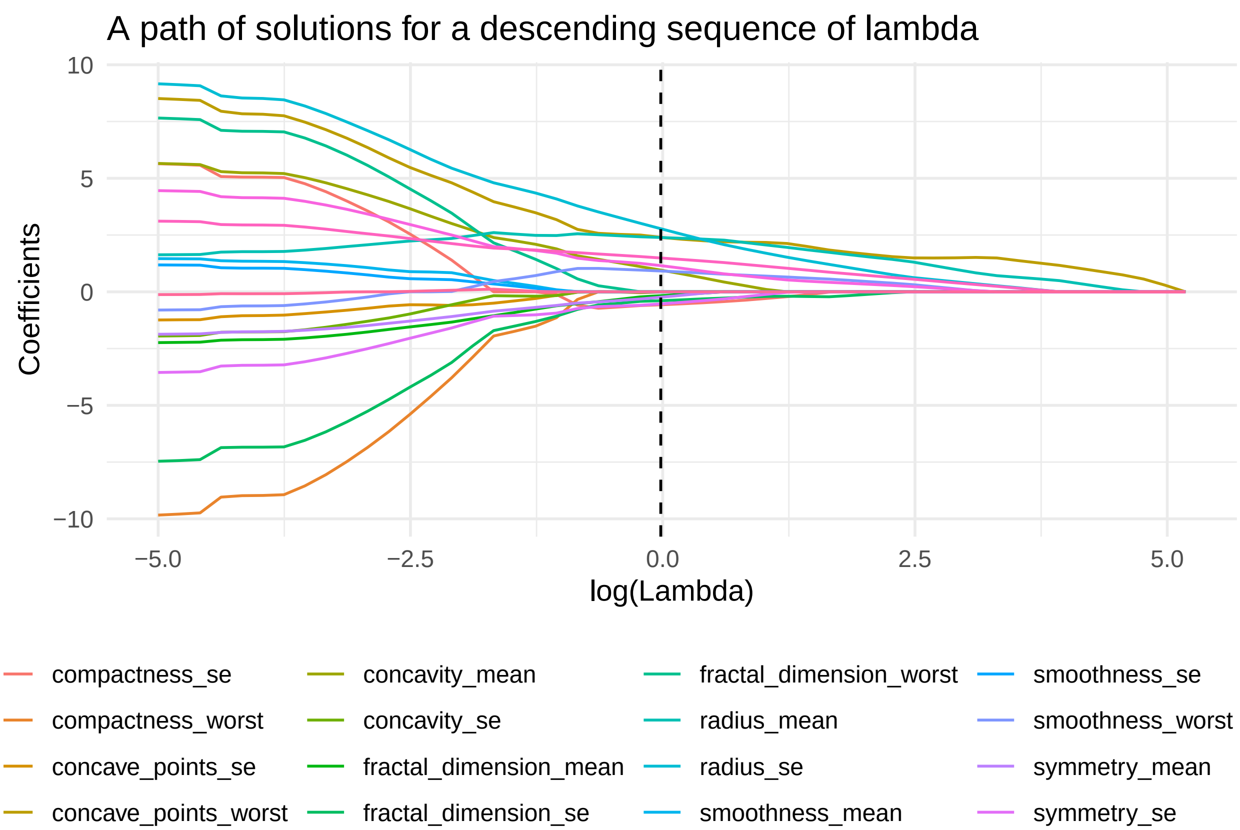
```r
#a path of solutions
pathwise <- function(x, y, lambda) {
  n <- length(lambda)
  betastart <- rep(0, 20)
  betas <- NULL
  for (i in 1:n) {
    coordwise_res <- coordwise_lasso(lambda = lambda[i],
                                     x = x,
                                     y = y,
                                     betastart = betastart)
    curbeta <- coordwise_res[nrow(coordwise_res), 3:22]
    betastart <- curbeta
    betas <- rbind(betas, c(curbeta))
  }
  return(data.frame(cbind(lambda, betas)))
}
pathwise_sol <- pathwise(x_train_stan, y_train, lambda_seq2)
round(pathwise_sol, 2) %>% knitr::kable()
```

lambda	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
177.83	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
144.47	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.30	0.00	0.00
117.37	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.56	0.00	0.00
95.35	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.74	0.00	0.00
77.46	0.00	0.22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.89	0.00	0.00
62.93	0.00	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.03	0.00	0.00
51.12	0.00	0.49	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.16	0.00	0.00

lambda	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
41.53	0.00	0.57	0.08	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.27	0.00	0.00
33.74	-0.05	0.64	0.16	0.00	0.00	0.00	0.00	0.18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.38	0.00	0.00
27.41	-0.11	0.71	0.24	0.00	0.00	0.00	0.00	0.27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.49	0.00	0.00
22.27	-0.17	0.83	0.32	0.00	0.00	0.00	0.00	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.00	1.51	0.05	0.00
18.09	-0.22	0.99	0.40	0.00	0.00	0.00	0.00	0.44	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.14	0.00	1.50	0.11	0.00
14.70	-0.25	1.15	0.48	0.00	0.00	0.00	0.00	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.00	1.49	0.17	0.00
11.94	-0.29	1.31	0.55	0.00	0.00	0.00	0.00	0.63	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.31	0.00	1.49	0.23	0.00
9.70	-0.32	1.43	0.63	0.00	0.00	0.00	0.00	0.76	0.00	0.00	0.00	0.00	0.00	0.00	-0.04	0.38	0.00	1.55	0.28	0.00
7.88	-0.34	1.53	0.71	0.00	0.00	0.00	0.00	0.91	0.00	0.00	0.00	0.00	0.00	0.00	-0.10	0.44	0.00	1.64	0.33	0.00
6.40	-0.35	1.63	0.79	0.00	0.00	0.00	0.00	1.06	0.00	0.00	0.00	0.00	0.00	0.00	-0.16	0.50	0.00	1.73	0.37	0.00
5.20	-0.37	1.74	0.86	0.00	0.00	0.00	0.00	1.21	0.00	0.00	-0.01	0.00	0.00	0.00	-0.22	0.56	0.00	1.84	0.42	0.00
4.22	-0.39	1.84	0.95	0.00	0.00	0.00	0.00	1.36	0.00	0.00	-0.12	0.00	0.00	0.00	-0.21	0.60	0.00	1.99	0.46	0.00
3.43	-0.40	1.96	1.04	0.00	0.00	0.00	0.00	1.52	0.00	0.00	-0.21	0.00	0.00	-0.02	-0.20	0.65	0.00	2.13	0.52	0.00
2.79	-0.41	2.06	1.12	0.00	0.10	0.00	0.00	1.70	0.00	0.00	-0.29	0.00	0.00	-0.11	-0.20	0.69	0.00	2.17	0.61	0.00
2.27	-0.41	2.17	1.20	0.00	0.27	0.00	0.00	1.88	0.00	0.00	-0.37	0.00	0.00	-0.21	-0.23	0.73	0.00	2.18	0.70	0.00
1.84	-0.41	2.28	1.29	0.00	0.43	0.00	0.00	2.07	0.00	0.00	-0.43	0.00	0.00	-0.31	-0.28	0.77	0.00	2.20	0.79	0.00
1.49	-0.40	2.32	1.36	0.00	0.61	-0.06	-0.06	2.29	0.00	0.00	-0.48	0.00	0.00	-0.39	-0.31	0.82	0.00	2.25	0.91	0.00
1.21	-0.39	2.36	1.42	0.00	0.79	-0.15	-0.12	2.53	0.00	0.00	-0.53	0.00	0.00	-0.46	-0.35	0.87	0.00	2.31	1.03	0.00
0.99	-0.38	2.39	1.49	0.00	0.95	-0.24	-0.17	2.77	0.00	0.00	-0.57	0.00	-0.01	-0.53	-0.39	0.91	0.00	2.39	1.15	0.00
0.80	-0.36	2.42	1.55	0.00	1.11	-0.32	-0.22	3.02	0.00	0.00	-0.61	0.00	-0.04	-0.59	-0.42	0.96	0.00	2.50	1.25	0.00
0.65	-0.33	2.47	1.61	0.00	1.27	-0.38	-0.33	3.27	0.00	0.00	-0.67	0.00	-0.01	-0.63	-0.50	0.99	0.00	2.53	1.32	0.13
0.53	-0.30	2.51	1.66	0.00	1.43	-0.44	-0.44	3.52	0.00	0.00	-0.73	0.00	0.00	-0.65	-0.58	1.03	0.00	2.58	1.38	0.27
0.43	-0.25	2.56	1.72	0.00	1.59	-0.50	-0.52	3.78	0.00	0.01	-0.59	-0.02	-0.01	-0.74	-0.78	1.03	-0.34	2.75	1.48	0.57
0.35	-0.17	2.48	1.79	0.09	1.89	-0.60	-0.62	4.09	0.00	0.05	-0.12	-0.17	-0.16	-0.94	-1.08	0.89	-1.14	3.18	1.70	1.02
0.28	-0.11	2.49	1.84	0.24	2.09	-0.69	-0.76	4.35	0.02	0.15	0.00	-0.19	-0.29	-1.01	-1.31	0.72	-1.52	3.48	1.82	1.43
0.23	-0.07	2.54	1.88	0.37	2.25	-0.77	-0.91	4.58	0.07	0.25	0.00	-0.18	-0.40	-1.05	-1.51	0.58	-1.74	3.73	1.91	1.80

lambda	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
0.19	-0.02	2.61	1.93	0.50	2.39	-0.85	-1.05	4.80	0.12	0.35	0.00	-0.17	-0.50	-1.07	-1.71	0.45	-1.95	3.97	1.99	2.16
0.15	0.00	2.48	2.02	0.67	2.71	-0.97	-1.19	5.12	0.09	0.44	0.69	-0.37	-0.57	-1.33	-2.38	0.22	-2.88	4.39	2.24	2.81
0.12	0.00	2.35	2.12	0.84	3.01	-1.09	-1.34	5.44	0.07	0.53	1.39	-0.57	-0.60	-1.59	-3.11	0.02	-3.79	4.80	2.50	3.46
0.10	0.05	2.29	2.23	0.87	3.33	-1.19	-1.44	5.84	0.04	0.56	1.99	-0.78	-0.58	-1.82	-3.68	0.00	-4.61	5.13	2.73	4.01
0.08	0.13	2.24	2.35	0.89	3.66	-1.29	-1.55	6.27	0.02	0.58	2.55	-0.98	-0.57	-2.05	-4.20	0.00	-5.40	5.48	2.96	4.54
0.07	0.19	2.16	2.46	0.96	3.98	-1.39	-1.66	6.70	0.00	0.65	3.08	-1.15	-0.63	-2.28	-4.74	-0.09	-6.16	5.90	3.20	5.06
0.05	0.24	2.07	2.56	1.06	4.27	-1.48	-1.77	7.09	0.00	0.74	3.56	-1.30	-0.73	-2.50	-5.26	-0.23	-6.85	6.35	3.42	5.56
0.04	0.29	1.99	2.65	1.15	4.54	-1.57	-1.87	7.48	-0.01	0.83	4.00	-1.43	-0.81	-2.71	-5.73	-0.34	-7.48	6.76	3.63	6.02
0.04	0.34	1.91	2.76	1.22	4.80	-1.63	-1.96	7.85	-0.04	0.91	4.41	-1.56	-0.89	-2.91	-6.17	-0.45	-8.06	7.14	3.82	6.43
0.03	0.38	1.84	2.85	1.28	5.03	-1.69	-2.03	8.18	-0.07	0.98	4.76	-1.67	-0.96	-3.08	-6.54	-0.53	-8.55	7.47	3.98	6.77
0.02	0.42	1.78	2.93	1.33	5.21	-1.74	-2.09	8.46	-0.08	1.03	5.03	-1.76	-1.03	-3.22	-6.83	-0.61	-8.94	7.75	4.12	7.04
0.02	0.42	1.77	2.94	1.34	5.24	-1.76	-2.11	8.52	-0.08	1.04	5.05	-1.77	-1.05	-3.23	-6.84	-0.62	-8.97	7.82	4.14	7.07
0.02	0.43	1.77	2.95	1.35	5.24	-1.76	-2.11	8.54	-0.09	1.04	5.05	-1.77	-1.05	-3.24	-6.85	-0.62	-8.98	7.84	4.15	7.07
0.01	0.44	1.75	2.96	1.37	5.29	-1.79	-2.13	8.63	-0.08	1.06	5.08	-1.78	-1.10	-3.27	-6.86	-0.66	-9.04	7.95	4.19	7.11
0.01	0.50	1.64	3.09	1.45	5.60	-1.85	-2.22	9.07	-0.12	1.17	5.57	-1.92	-1.23	-3.52	-7.39	-0.79	-9.74	8.43	4.42	7.58
0.01	0.50	1.64	3.10	1.45	5.63	-1.86	-2.23	9.12	-0.12	1.18	5.61	-1.94	-1.23	-3.54	-7.43	-0.80	-9.79	8.47	4.44	7.62
0.01	0.51	1.63	3.11	1.46	5.65	-1.87	-2.24	9.16	-0.12	1.18	5.64	-1.95	-1.24	-3.55	-7.46	-0.80	-9.83	8.51	4.46	7.65

```r
colnames(pathwise_sol) <- c("lambda", rownames(coef(summary(glm.fit))))
pathwise_sol %>%
  pivot_longer(
    3:21,
    names_to = "variables",
    values_to = "coefficients") %>%
  ggplot(aes(x = log(lambda), y = coefficients, group = variables, color = variables)) +
  geom_line() +
  geom_vline(xintercept = log(0.981), linetype = 2) +
  ggtitle("A path of solutions for a descending sequence of lambda") +
  xlab("log(Lambda)") +
  ylab("Coefficients") #+
```




```
#theme(legend.text = element_text(size = 6))
```

cross-validation

```
#{r} #set.seed(2022) #cv = function(data, lambda) { #  n <- nrow(data) #  data <- data[sample(n),
] #shuffle the data #  folds <- cut(seq(1, nrow(data)), breaks = 5, labels = FALSE)
#Create 5 equal size #folds # # mse <- data.frame() #a data frame storing mse results
#  #mse_lambda <- vector() #  #se <- vector() #a vector storing test errors #  res <-
lambda  #  #se <- vector() #a vectro storing test errors #   #    #Perform 5 fold cross
validation #  for (i in 1:5) { #    #partition the data into train and test data #    testRows
<- which(folds == i, arr.ind = TRUE) #    data_test <- data[testRows, ] #    data_train
<- data[-testRows, ] #    x_train <- data_train[2:20] #    x_train_stan <- cbind(rep(1,
nrow(x_train)), scale(x_train)) #    y_train <- data_train[1] #    x_test <- data_test[2:20]
#    #standardized test data #    x_test_stan <- cbind(rep(1, nrow(x_test)), scale(x_test))
#    y_test <- data_test %>% mutate(diagnosis = factor(diagnosis)) #    y_test <- y_test$diagnosis
#    #Use the test and train data partitions to perform lasso #    path_sol <- pathwise(x
= x_train_stan, #                          y = y_train, #                          lambda
= lambda) #    auc <- vector() #    for (j in 1:length(lambda)) { #      curbeta <-
as.numeric(path_sol[j, 2:21]) #      theta <- x_test_stan %*% curbeta #      p <- exp(theta)
/ (1 + exp(theta))  #      auc[j] <- auc(y_test, p) #      #y.pred <- ifelse(p > 0.5,
1, 0) #      #accuracy[j] <- mean(y.pred == y_test) #    } #    print(auc) #    res <-
cbind(res, auc) #    print(res) #  } #  return(res) #    #se[j] <- sqrt(var(error)/5)
#  #cv.auc.lambda <- rowMeans(mse) #  #return(cv.auc.lambda) #} #cv_test = cv(data =
breast_train, lambda_seq2) # #lll <- as.data.frame(cv_test) #colnames(c("auc1", "auc2",
"auc3", "auc4", "auc5")) #colnames(lll) <- c("res", "auc1", "auc2", "auc3", "auc4",
```

```
"auc5") #lll<-lll %>% dplyr::select(-1) #mean <- rowMeans(lll) #max(mean) # # #cv_res <-
as.data.frame(cv_test) #colnames(c("auc1", "auc2", "auc3", "auc4", #"auc5")) #colnames(cv_res)
<- c("res", "auc1", "auc2", "auc3", "auc4", "auc5") #cv_lambda <- cv_res[1] #mean_auc
<- cv_res %>% dplyr::select(-1) %>% rowMeans() #cv_auc <- cbind(cv_lambda, mean_auc)
#maxauc <- max(cv_auc$mean_auc) #bestlambda <- cv_auc[which(cv_auc$mean_auc == maxauc
),]$res #cv_auc %>%  #  ggplot(x = res, y = mean_auc ) + #  geom_line(aes(x = res, y =
mean_auc), col = "blue") + #  geom_vline(xintercept = bestlambda, linetype = "dashed",
col = "red") + #  labs(title = "Mean AUC vs. Lambda", #       x = "Lambda", #       y =
"Mean AUC") # # # ### Compare full model and lasso model #{r} ##corresponding betas of
best lambda #lasso_beta <- pathwise_sol[which(pathwise_sol$lambda == bestlambda ),][2:21]
%>% #as.numeric() # ##prediction performance function #predict <- function(x, y, betavec)
{ #  theta <- x %*% betavec #  p <- exp(theta) / (1 + exp(theta)) #  auc <- auc(y, p)}
# #auc_lasso <- predict(x_test_stan, y_test, lasso_beta) #auc_lasso # #cbind(auc_full,
auc_lasso) %>% knitr::kable() # #   #   #   #{r} ##coefficients of full and lasso models
#glm_beta <- glm.fit$coefficients %>% as.vector() #coefnames <- rownames(coef(summary(glm.fit)))
#cbind(coefnames, glm_beta, lasso_beta) %>% knitr::kable() #
```