Breast Cancer Diagnosis

Mar 20, 2022

Data import

```r
# --- Import data ---
breast_cancer =
  read_csv("./data/breast-cancer.csv") %>%
  dplyr::select(-c(1, 33)) %>%
  janitor::clean_names() %>%
  # add extra row
  add_row(diagnosis = 'B', radius_mean = 7.76, texture_mean = 24.54,
          perimeter_mean = 47.92, area_mean = 181, smoothness_mean = 0.05263,
          compactness_mean = 0.04362, concavity_mean = 0,
          concave_points_mean = 0, symmetry_mean = 0.1587,
          fractal_dimension_mean = 0.05884, radius_se = 0.3857,
          texture_se = 1.428, perimeter_se = 2.548, area_se = 19.15,
          smoothness_se = 0.007189, compactness_se = 0.00466, concavity_se = 0,
          concave_points_se = 0, symmetry_se = 0.02676,
          fractal_dimension_se = 0.002783, radius_worst = 9.456,
          texture_worst = 30.37, perimeter_worst = 59.16, area_worst = 268.6,
          smoothness_worst = 0.08996, compactness_worst = 0.06444,
          concavity_worst = 0, concave_points_worst = 0,
          symmetry_worst = 0.2871, fractal_dimension_worst = 0.07039) %>%
  mutate(diagnosis =
           as.numeric(as.factor(recode(diagnosis, `M` = 1, `B` = 0))) - 1) %>%
  mutate_each_(funs(scale(.)), c(2:31))


# --- Data partition (8:2) ---
set.seed(2022)
trRows <- createDataPartition(breast_cancer$diagnosis, p = 0.8, list = FALSE)

# --- `mean` predictors for Newton-Raphson method ---
pred_1 <- as.tibble(breast_cancer[2:11])
bc_scale1 <- as.matrix(cbind(intercept = rep(1, nrow(breast_cancer)),
                             pred_1, outcome = breast_cancer$diagnosis))

# train
bc_scale1 <- bc_scale1[trRows, ]
bc_scale1_x <- bc_scale1[, -12]
bc_scale1_y <- bc_scale1[, 12]

# test
t_bc_scale1_x <- bc_scale1[-trRows, -12]
t_bc_scale1_y <- bc_scale1[-trRows, 12]
```

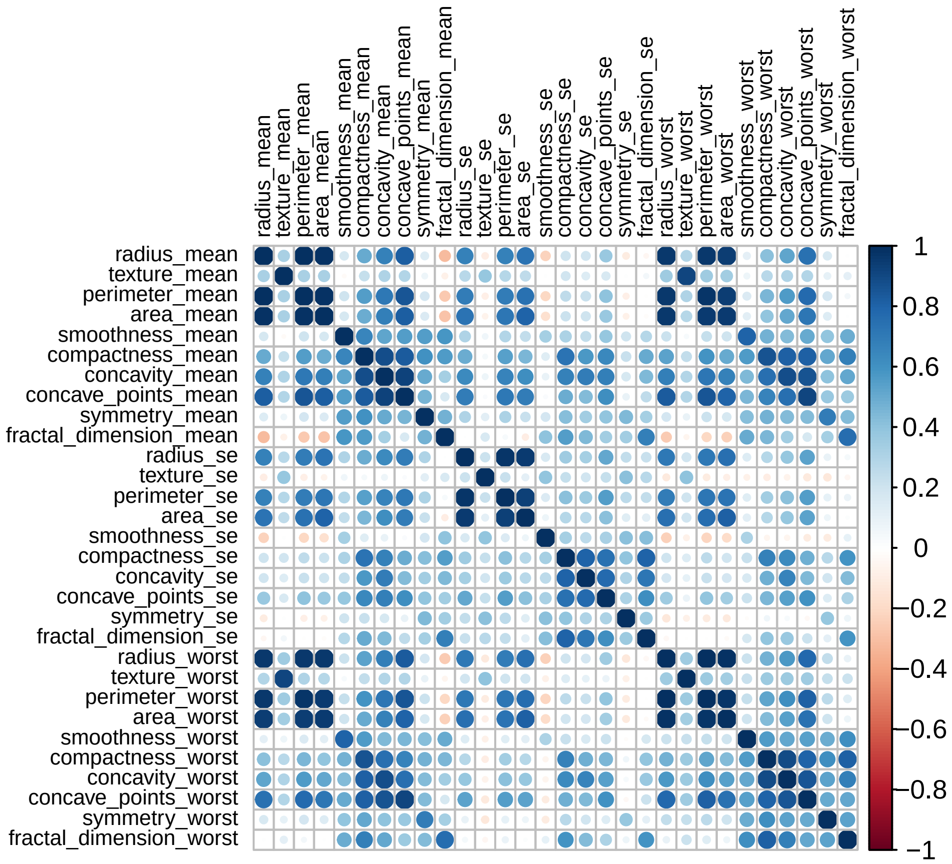
```r
# --- All predictors for lasso & CV ---
pred_2 <- as.tibble(breast_cancer[2:31])
bc_scale2 <- as.matrix(cbind(intercept = rep(1, nrow(breast_cancer)),
                             pred_2, outcome = breast_cancer$diagnosis))

# train
bc_scale2 <- bc_scale2[trRows, ]
bc_scale2_x <- bc_scale2[, -32]
bc_scale2_y <- bc_scale2[, 32]

# test
t_bc_scale2_x <- bc_scale2[-trRows, -32]
t_bc_scale2_y <- bc_scale2[-trRows, 32]
```

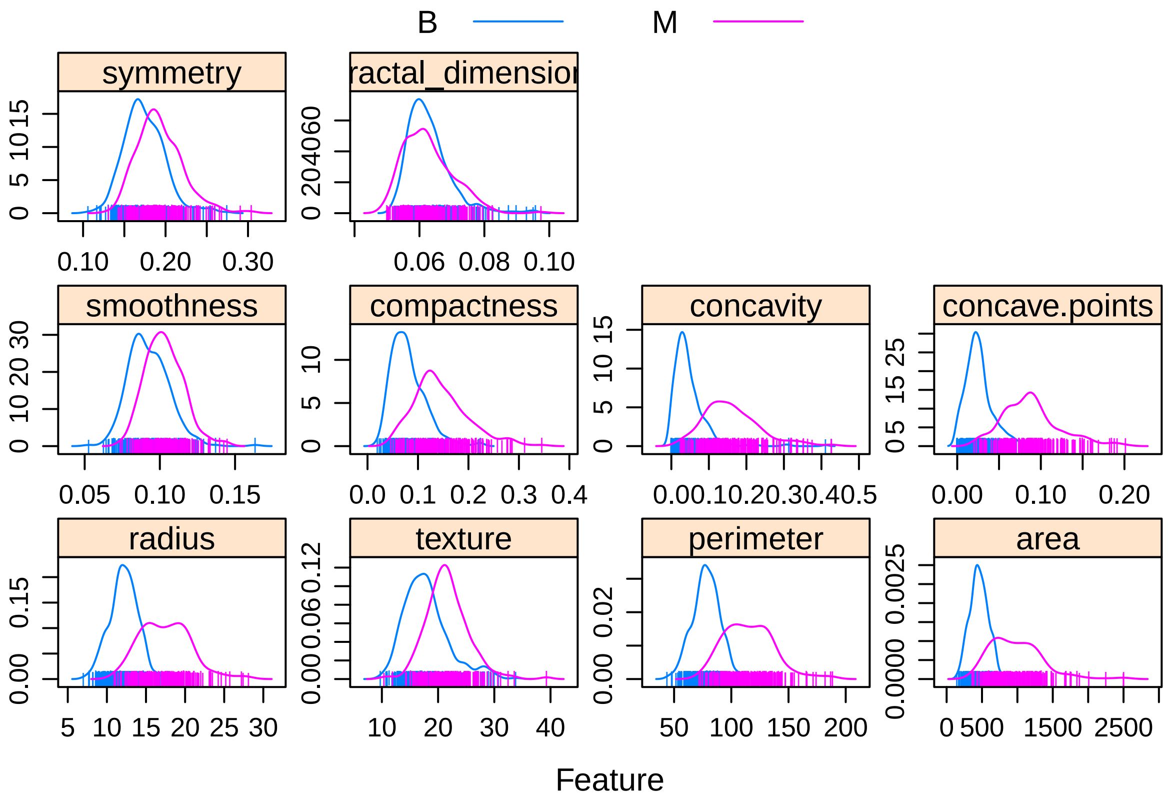
Exploratory Data Analysis

```r
# --- Correlation plot ---
corrplot::corrplot(cor(breast_cancer[2:31]), type = "full",
                   tl.cex = .7, tl.col = "black")
```

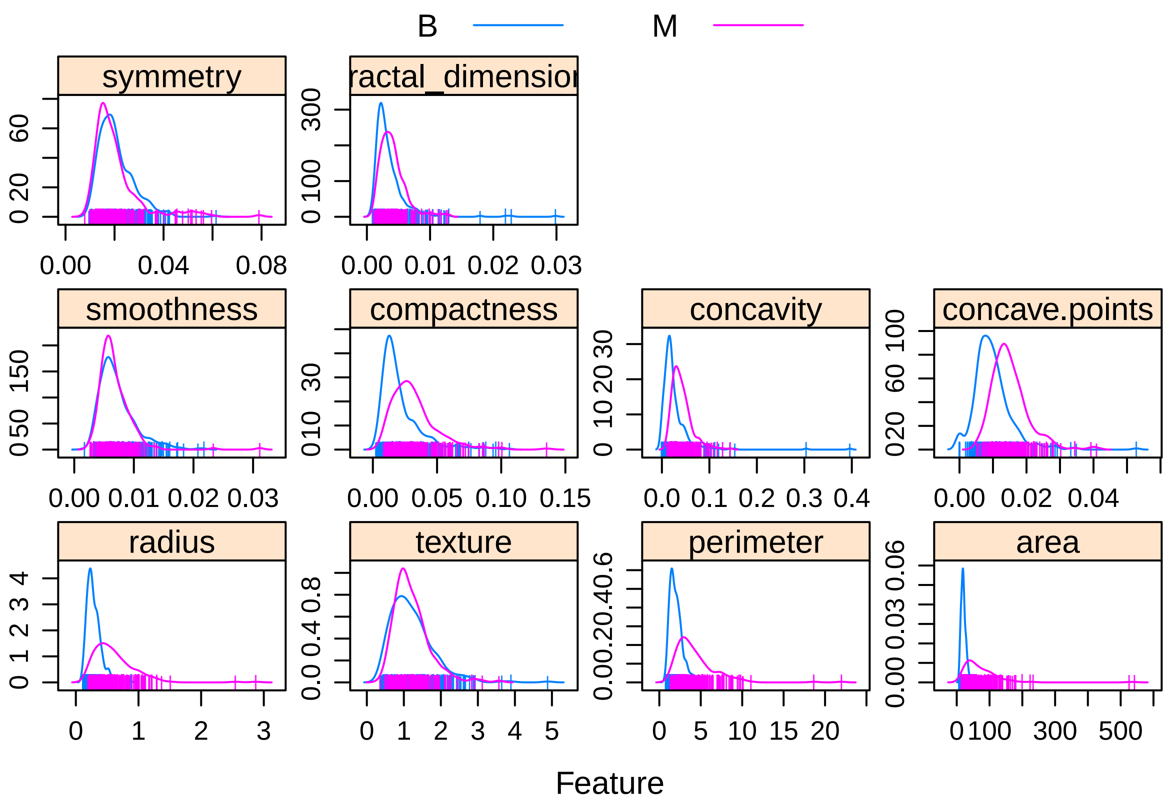


```r
# --- Fea
bc = read.csv("./data/breast-cancer.csv") %>%
  mutate(diagnosis = factor(diagnosis))
```

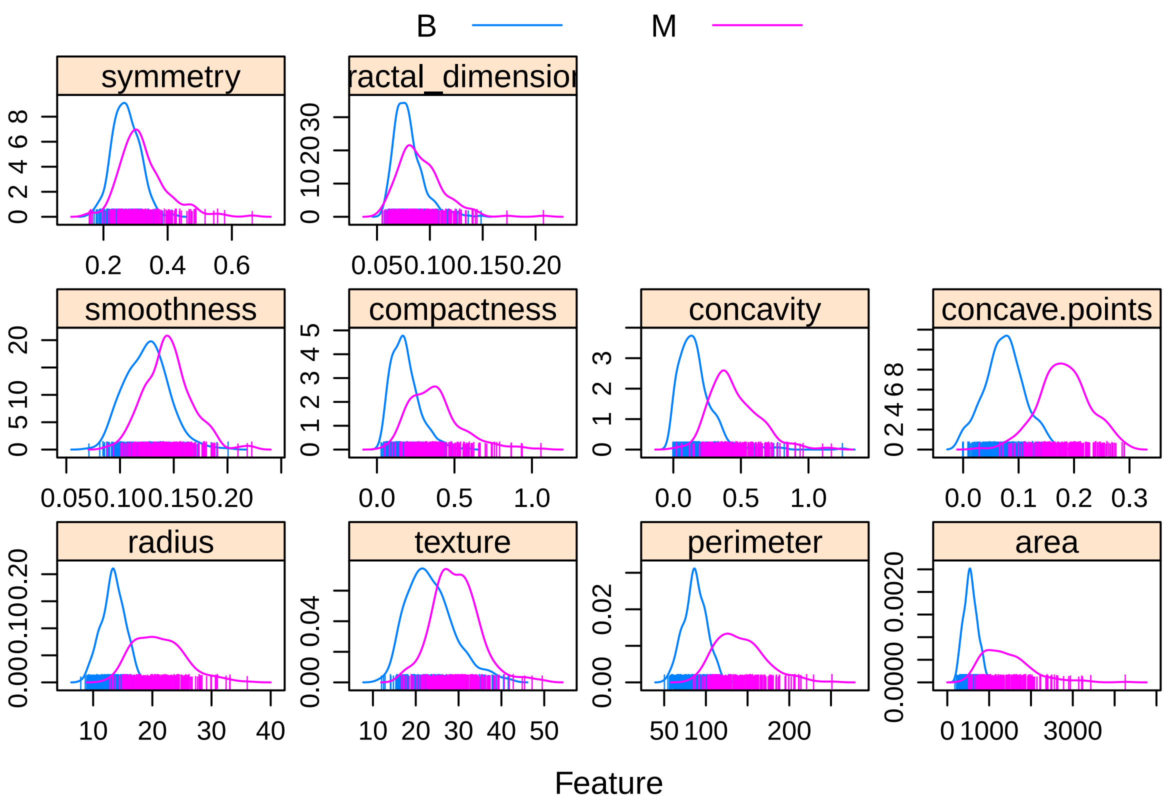
```r
#mean
cancer_mean = bc[, 3:12] %>% as_tibble()
colnames(cancer_mean) = gsub("_mean", "", colnames(cancer_mean))
featurePlot(x = cancer_mean,
            y = bc$diagnosis,
            scales = list(x = list(relation = "free"),
                          y = list(relation = "free")),
            plot = "density", pch = "|",
            auto.key = list(columns = 2))
```



```r
#se
cancer_se = bc[, 13:22] %>% as_tibble()
colnames(cancer_se) = gsub("_se", "", colnames(cancer_se))
featurePlot(x = cancer_se,
            y = bc$diagnosis,
            scales = list(x = list(relation = "free"),
                          y = list(relation = "free")),
            plot = "density", pch = "|",
            auto.key = list(columns = 2))
```



```
#largest
cancer_max = bc[, 23:32] %>% as_tibble()
colnames(cancer_max) = gsub("_worst", "", colnames(cancer_max))
featurePlot(x = cancer_max,
            y = bc$diagnosis,
            scales = list(x = list(relation = "free"),
                          y = list(relation = "free")),
            plot = "density", pch = "|",
            auto.key = list(columns = 2))
```



Modified Newton-Raphson method

```
library(MASS)
# Compute likelihood function, gradient, and Hessian matrix
compute_stat <- function(dat, betavec){
  x <- dat # include intercept
  y <- bc_scale1_y
  u <- x %*% betavec # n * 1 vector
  pi <- exp(u) / (1 + exp(u))
  # Log-likelihood
  loglik <- sum(y * u - log(1 + exp(u)))
  # Gradient -- (p + 1) * 1 vector
  grad <- t(x) %*% (y - pi)
  # Hessian -- (p + 1) * (p + 1) matrix
  hess <- -t(x) %*% diag(c(pi * (1 - pi))) %*% x

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

# Modified Mewton-Raphson method
NewtonRaphson <- function(dat, func, start, tol = 1e-15, maxier = 200) {
  i <- 0 # Iteration indicator
  cur <- start # Current position / beta's

  # Computate log-likelihood, gradient, and Hessian matrix
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur) # Store results
  prevloglik <- -Inf # Ensure iterations

  while (i < maxier && abs(stuff$loglik - prevloglik) > tol) {
    # --- Base case ---
    i <- i + 1
    step <- 1 # Original step size
    prevloglik <- stuff$loglik
    prev <- cur
    cur <- prev - solve(stuff$hess) %*% stuff$grad

    # Step halving and re-direction
    while (func(dat, cur)$loglik < stuff$loglik) {
      # Check if all eigenvalues are negative
      # IF NOT, redirection
      eig <- eigen(stuff$hess)$value
      if (sum(eig < 0) != length(eig)) {
        gamma <- max(eig)
        new_hess <- stuff$hess - gamma * diag(nrow = dim(dat)[1])
        cur <- prev - solve(new_hess) %*% stuff$grad
      }
      # Reduce step size by half
      else {
        step <- step / 2
        cur <- prev - step * solve(stuff$hess) %*% stuff$grad
      }
    }
    stuff <- func(dat, cur)

    # break if NA
    if (is.na(stuff$loglik)) {break}

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

NR results (both 10 means & all 30 predictors)

```
# NewtonRaphson(bc_scale1_x, compute_stat, rep(1, 11))
# NewtonRaphson(bc_scale2_x, compute_stat, rep(10, 31))

# The result of using 30 of predictors is not ideal
# due to some potential collinearity
```

Compare with `glm` reults

```
# Fit a logistic model
glm_fit <- glm(diagnosis ~., family = binomial(link = "logit"), data = breast_cancer[, 1:11])
glm_fit$coefficients
```

```
##            (Intercept)            radius_mean           texture_mean
##             0.48701675            -7.22185053             1.65475615
##         perimeter_mean              area_mean        smoothness_mean
##            -1.73763027            14.00484560             1.07495329
##       compactness_mean         concavity_mean    concave_points_mean
##            -0.07723455             0.67512313             2.59287426
##          symmetry_mean fractal_dimension_mean
##             0.44625631            -0.48248420
```

```
# Comparison
NR_result <- NewtonRaphson(bc_scale1_x, compute_stat, rep(10, 11))
round(tail(NR_result, 1)[, 3:13], 3) == round(as.vector(glm_fit$coefficients), 3) # return all TRUE
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Soft threshold and coor-wise Lasso

```
# Soft Threshold function
sf <- function(beta, lambda) {
  beta <- ifelse(lambda < abs(beta),
                 ifelse(beta > 0, beta - lambda, beta + lambda), 0)
  return(beta)
}


# Coordinate-wise LASSO with fixed lambda
y <- breast_cancer$diagnosis

cd_lasso <- function(dat, y, betavec, lambda, maxier = 2000, tol = 1e-10) {
  x <- dat # n * (p + 1) matrix, standardized
  i <- 0
  objfun <- 0
  prev_objfun <- -Inf # ensure iterations
  res <- c(0, objfun, betavec)

  while (i < maxier && abs(objfun - prev_objfun) > tol) {
    i <- i + 1
    prev_objfun <- objfun

    for (j in 1:length(betavec)) {
      u <- x %*% betavec
      pi <- exp(u) / (1 + exp(u))
      # working weights
      w <- pi * (1 - pi)
      w <- ifelse(w < 1e-5, 1e-5, w)
      # working response
      z <- x %*% betavec + (y - pi) / w
      z_dej <- x[, -j] %*% betavec[-j]
      betavec[j] <-
        sf(sum(w * x[, j] * (z - z_dej)), lambda) / (sum(w * x[, j]^2))
    }
    objfun <-
      sum(w * (z - x %*% betavec)^2) / (2 * dim(x)[1]) + lambda * sum(abs(betavec))
    if (is.na(objfun)) {break}

    res <- rbind(res, c(i, objfun, betavec))
  }

  return(res)
}
```

Try cd_lasso…

```
# cd_lasso(bc_scale2_x, bc_scale2_y, betavec = rep(1, 31), lambda = 1)
```

path-wise coordinate-wise for lasso solution path

```
# pathwise based on cd_lasso
pathwise_cd_lasso <- function(dat, y, betavec, lambda) {

  sort(lambda, decreasing = TRUE)
  pathwise_res <- NA

  for (j in 1:length(lambda)) {
    cd_results <- cd_lasso(dat, y, betavec, lambda[j])

    # warm start
    min_objfun_ind <- which.min(cd_results[-1, 2]) + 1
    betavec <- cd_results[min_objfun_ind, -c(1, 2)]
    objfun <- cd_results[min_objfun_ind, 2]

    pathwise_res <- rbind(pathwise_res,
                          c(j, lambda = lambda[j], obj = objfun,
                            beta = betavec))

  }

  return(pathwise_res[-1, ])
}
```

5-fold CV using cd_lasso

```
cv = function(raw_data, lambda_vec, kfolds = 5, betavec){

  fold_index = cut(seq(1,nrow(raw_data)),breaks = kfolds, labels = FALSE)
  raw_data = raw_data[sample(nrow(raw_data)), ]

  final_res <- c(lambda = 0, auc = 0, rss = 0)
  for (i in 1:length(lambda_vec)) {
    curlambda = lambda_vec[i]
    fold_auc = vector(); fold_rss = vector()

    for (k in 1:kfolds) {
      train = raw_data[fold_index != k, ]
      validation = raw_data[fold_index == k, ]

      train_x = train[, -ncol(raw_data)]
      train_y = train[, ncol(raw_data)]
      validation_x = validation[, -ncol(raw_data)]
      validation_y = validation[, ncol(raw_data)]

      # cd_lasso
      res = cd_lasso(train_x, train_y, betavec, curlambda)

      curbeta = res[dim(res)[1], 3:dim(res)[2]]
      u = validation_x %*% as.matrix(curbeta) # u: n * 1 vector
      prob = exp(u) / (1 + exp(u))

      fold_auc[k] = roc(as.factor(validation_y), prob)$auc[1]
      fold_rss[k] = sum((validation_y - prob)^2)
    }

    final_res = rbind(final_res, c(lambda = curlambda, mean_auc = mean(fold_auc),
                                   sd_auc = sd(fold_auc), rss = mean(fold_rss)))
  }

  return(final_res)
}

set.seed(2022)
cv_res <- cv(bc_scale2, lambda_vec = exp(seq(4, -3, length = 30)), betavec = rep(1, 31))
```

Compare optimal with full using test data

```
opt_lambda = cv_res[which.max(cv_res[, 2]), 1]

# Optimal lambda
optimal = cd_lasso(t_bc_scale2_x, t_bc_scale2_y, lambda = opt_lambda, betavec = rep(1,31))[,-c(1:2)] %>
  as.matrix()
u_opt <- t_bc_scale2_x %*% optimal[nrow(optimal),]
prob_opt <- as.matrix(exp(u_opt) / (1 + exp(u_opt)))
roc(as.factor(t_bc_scale2_y), prob_opt)$auc[1]
```

```
## [1] 0.9988981
```

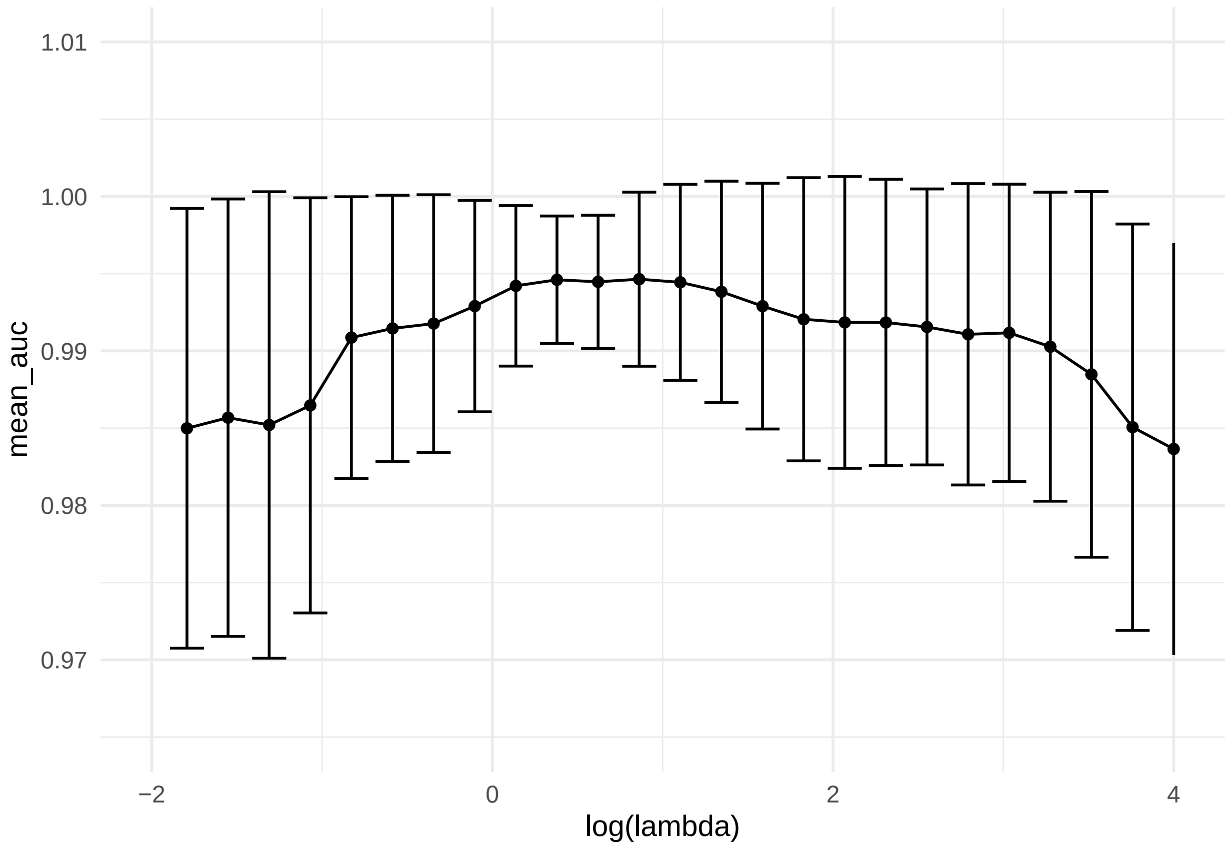
```
# Without lambda
full = cd_lasso(t_bc_scale2_x, t_bc_scale2_y, lambda = 0, betavec = rep(1,31))[,-c(1:2)] %>%
  as.matrix()
```

```r
u_full <- t_bc_scale2_x %*% full[nrow(full),]
prob_full <- as.matrix(exp(u_full) / (1 + exp(u_full)))
roc(as.factor(t_bc_scale2_y), prob_full)$auc[1]
```

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

5-fold cv mean auc & sd auc

```r
# mean & sd
as.tibble(cv_res[-1, ]) %>%
  ggplot(aes(x = log(lambda), y = mean_auc)) +
  geom_point() +
  geom_line() + xlim(c(-2, 4)) +
  geom_errorbar(aes(ymin = mean_auc - sd_auc, ymax = mean_auc + sd_auc), width = .2) +
  ylim(c(0.965, 1.01)) + theme_minimal()
```



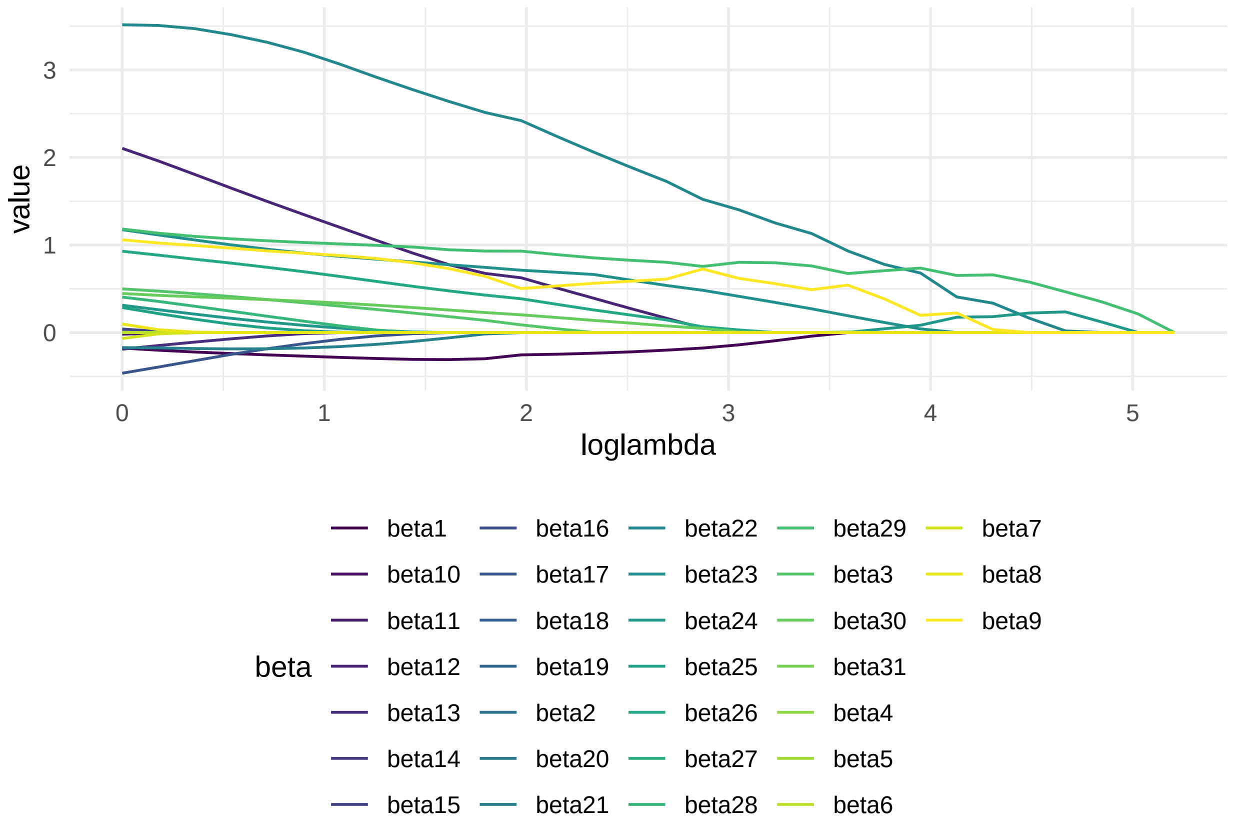
Lasso solution path

```r
# Computate maximum lambda shrinking every beta to 0
max_lambda2 <- max(t(bc_scale2_x) %*% bc_scale2_y)

# pathwise cd_lasso
```

```
pathwise = pathwise_cd_lasso(bc_scale2_x, bc_scale2_y, rep(1,31), exp(seq(log(max_lambda2), 0, length =
tidy.path = as.tibble(pathwise)[,-c(1,3)] %>%
  pivot_longer(2:32, names_to = "beta", values_to = "value") %>%
  mutate(loglambda = log(lambda))

ggplot(data = tidy.path, aes(x = loglambda, y = value, color = beta, group = beta)) +
  geom_line()
```


value
loglambda
beta
beta1
beta10
beta11
beta12
beta13
beta14
beta15
beta16
beta17
beta18
beta19
beta2
beta20
beta21
beta22
beta23
beta24
beta25
beta26
beta27
beta28
beta29
beta3
beta30
beta31
beta4
beta5
beta6
beta7
beta8
beta9