# P8160 Group Project 2: Breast Cancer Diagnosis and Optimizations

Wenhan Bao | Tianchuan Gao | Jialiang Hua | Qihang Wu | Paula Wu

3/25/2022

## Objective

The main objective of our project is to build an accurate predictive model based on logistic regression that classifies malignant and benign images of breast tissue. Using the Breast Cancer Diagnosis dataset, we will implement logistic model and logistic-LASSO model to predict the diagnosis. A modified Newton-Raphson algorithm and a Pathwise Coordinate optimization will be developed to estimate the logistic model and the lasso model, respectively. 5-fold cross-validation will be applied to find the optimal value for the tuning parameter $\lambda$. Our aim is to find the model with the best performance in predicting the diagnosis of breast tissue images.

## Background & Methods

### Background

Breast cancer, which affects 1 in 7 women worldwide, is the most common invasive cancer in women around the world. It can start from different parts of the breast and is marked by the uncontrolled growth of breast cells. Benign breast tumors do not metastasize and are usually not life-threatening, while malignant tumors are aggressive and deadly. Nowadays, substantial support for breast cancer awareness and research funding has created great advances in the diagnosis and treatment of breast cancer. The prognosis of the disease has been greatly improved once it is detected and treated early. Therefore, it's important to have breast lumps accurately diagnosed so that timely clinical intervention can be conducted.

The dataset we used contains 569 observations and 33 columns, including 357 benign and 212 malignant cases. 30 out of the 33 columns are predictors that contain the mean, standard deviation, and the largest value of the distributions of 10 distinct features listed below:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter$\hat{2}$ / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

Our goal is to build a predictive model based on logistic regression to facilitate the breast cancer diagnosis.

# Methods

## 1. Exploratory Data Analysis (EDA)

We imported and cleaned the data and conducted EDA. We plotted the correlation plot **(Fig 1)** of all the predictors. From that, we find that there is a high correlation between many of the predictors. This is because our predictors include mean, standard deviation and the largest values of the distributions of 10 features, which means that some predictors can be calculated from others. Then, we made the feature plots **(Fig 2)** to explore the relationship between binary diagnosis outcome and all predictors to determine the potential effective predictors. Features such as the mean of "compactness", "radius", "concavity", "perimeter", "area", and "concave points" clearly distinguish benign breast cancer images from the malignant ones - benign tissues tend to have lower mean values on the scale compared to the malignant tissues.

**Fig 1 Correlation Plot**

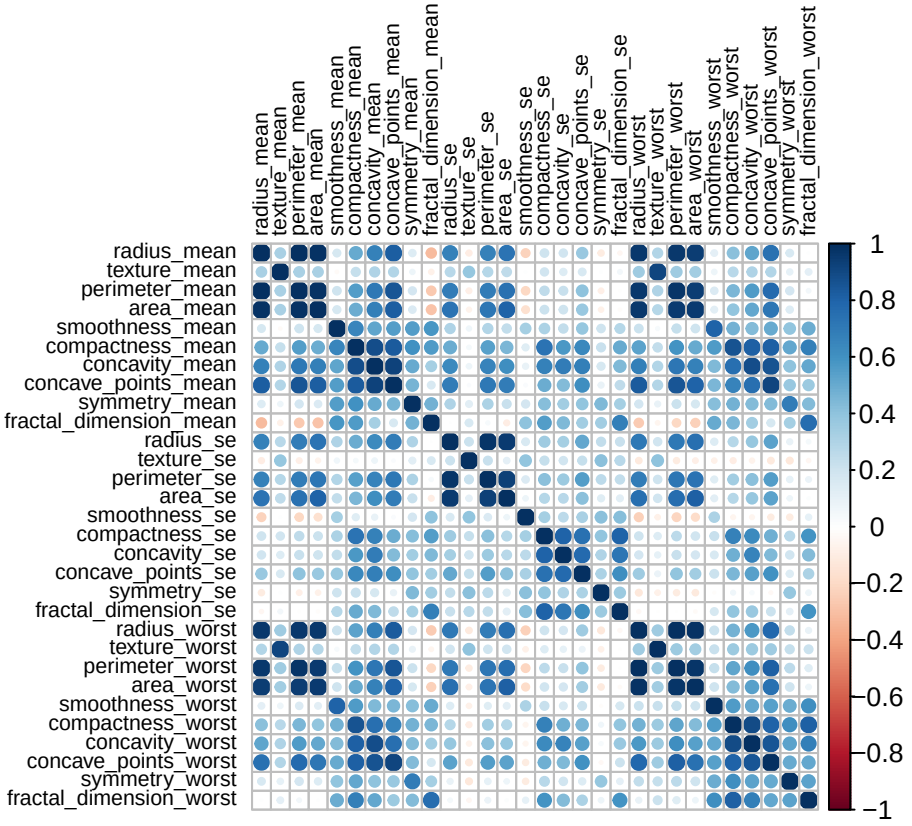


**Fig 2.1 Response and mean of features**

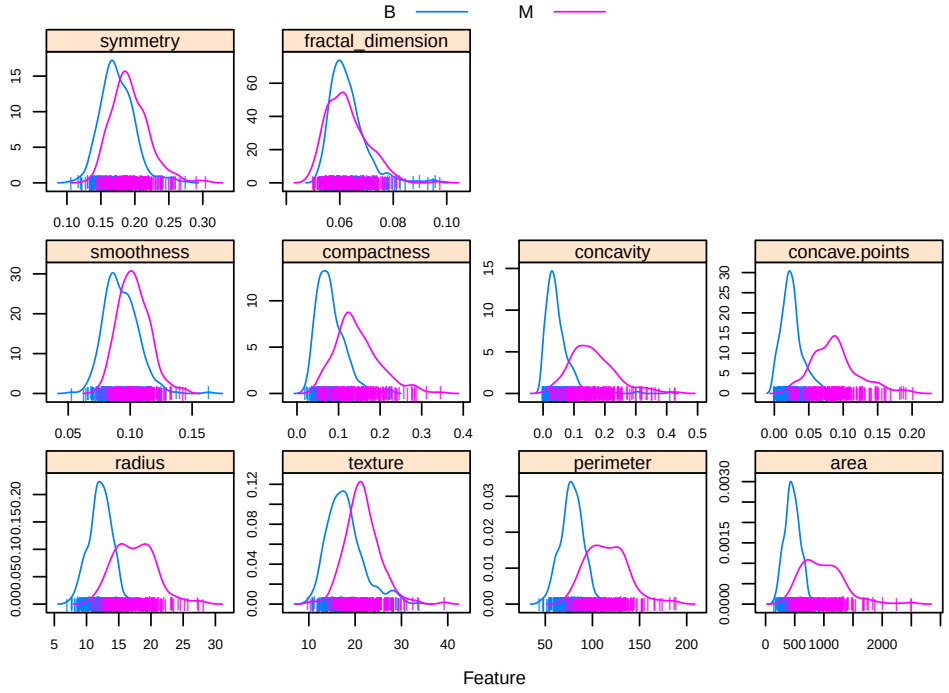



**Fig 2.2 Response and SD of features**

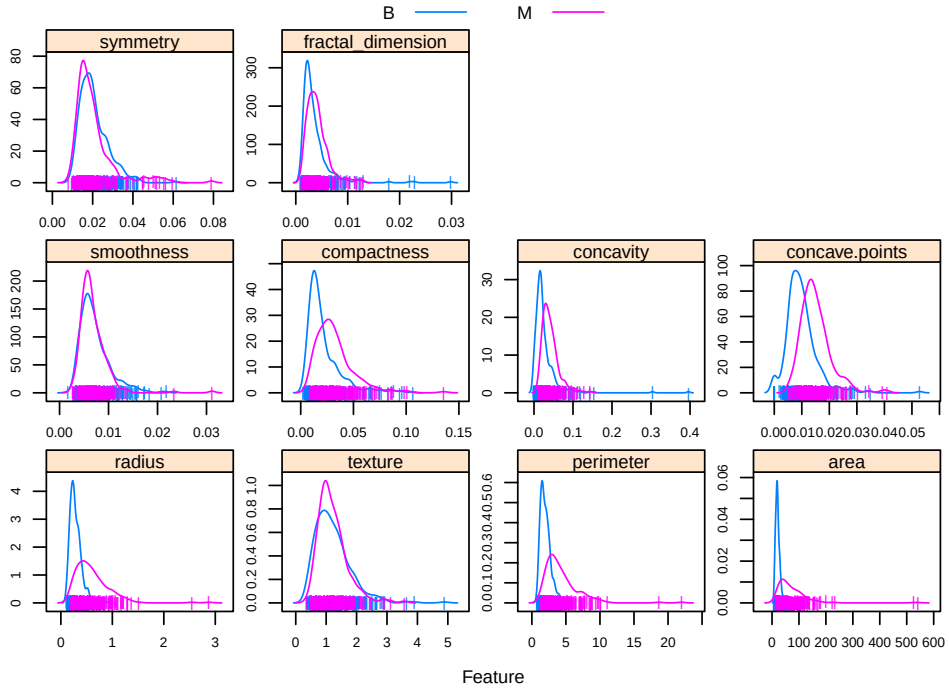
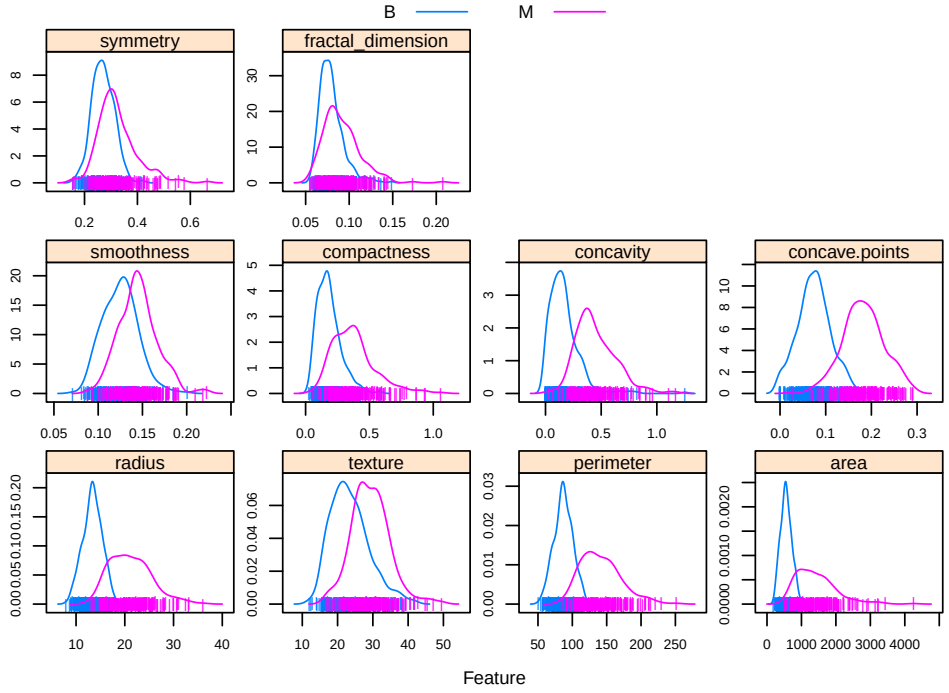


**Fig 2.3 Response and largest value of features**




## 2. Modified Newton-Raphson Algorithm

***Model Parameters***

The logistic model can be defined as:

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

$\pi_i$ stands for the probability that the tissue is malignant in the $i_{th}$ observation and it can be written as:

$$\pi_i = \frac{e^{\mathbf{X}\boldsymbol{\beta}}}{1+e^{\mathbf{X}\boldsymbol{\beta}}}$$

The likelihood function for vector $\beta$ is:

$$L(\beta) = \prod_{i=0}^{n}(\frac{e^{\mathbf{X}\beta}}{1+e^{\mathbf{X}\beta}})^{y_i}(\frac{1}{1+e^{\mathbf{X}\beta}})^{1-y_i}$$

and thus we can calculate the log-likelihood function, gradient and Hessian matrix respectively.:

$$l(\beta) = \sum_{i=0}^{n}(y_i * X\beta - log(1+e^{X\beta}))$$

$$\nabla l(\beta) = \begin{pmatrix} \sum_{i=1}^{n}(y_i - \pi_i) \\ \sum_{i=1}^{n} x_{i1} \times (y_i - \pi_i) \\ . \\ . \\ . \\ \sum_{i=1}^{n} x_{in} \times (y_i - \pi_i) \end{pmatrix}$$

$$\nabla^2 f(\beta) = -\sum_{i=1}^{n} \begin{pmatrix} 1 \\ x_i \end{pmatrix} \begin{pmatrix} 1 & x_i \end{pmatrix} \pi_i (1-\pi_i)$$

$$= -\begin{pmatrix} \sum \pi_i (1-\pi_i) & \sum x_i \pi_i (1-\pi_i) \\ \sum x_i \pi_i (1-\pi_i) & \sum x_i^2 \pi_i (1-\pi_i) \end{pmatrix}$$

***Model Explanation***
With those parameters above, we can apply the Newton-Raphson algorithm to calculate $\beta$ with the following equation in each iteration:

$$\boldsymbol{\beta_{i+1}} = \boldsymbol{\beta_i} - \left[\nabla^2 l\left(\boldsymbol{\beta_i}\right)\right]^{-1} \nabla l\left(\boldsymbol{\beta_i}\right)$$

We also apply two modifications: step-halving and re-direction. Both modification are meant to ensure the likelihood is increasing the the ascent direction at each iteration.

## 3. Logistic-LASSO

***Least Absolute Shrinkage and Selection Operator (LASSO)*** To estimate coefficients through Newton-Raphson method, it is necessary to compute the corresponding inverse of Hessian Matrix $[\nabla^2 f(\theta_{i-1})]^{-1}$. However, the computational burden of calculation will increase as the dimension of predictors increase. On top of that, the collinearity could also be a problem. Therefore, we use a regularization method, LASSO, to shrink coefficients and perform variable selections. For regression lasso, the objective function is:

$$f(\beta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda\sum_{i=1}^{p}|\beta_j|,$$

where the first term is residual sum of squares (RSS) and the second term is the lasso l1 penalty. Noted that the $x_{i,j}$ needs to be standardized before LASSO so that the penalty will be equally applied to all predictors. For each single predictor, the LASSO solution is like:

$$\hat{\beta}^{lasso}(\lambda) = S(\hat{\beta}, \lambda) = \begin{cases} \hat{\beta} - \lambda, \text{ if } \hat{\beta} > 0 \text{ and } \lambda < |\hat{\beta}| \\ \hat{\beta} + \lambda, \text{ if } \hat{\beta} < 0 \text{ and } \lambda < |\hat{\beta}| \\ 0, \text{ if } \lambda > |\hat{\beta}|, \end{cases}$$

where $S(\hat{\beta}, \lambda)$ is called soft threshold. The basic idea of this function is to shrink all $\beta$ coefficients between $-\lambda$ and $\lambda$.

***Coordinate-wise Descent Algorithm*** Another approach to solve the complex computation of inverse Hessian Matrix is coordinate descent approach, which starts with an initial guess of parameters $\theta$, optimizes one parameter at a time based on the best knowledge of other parameters, and uses the obtained results as the start values for the next iteration. We repeat the above steps until convergence. Finally, this approach tries to minimize the following objective function when considering a lasso penalty:

$$f(\beta_j) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \sum_{k\neq j} x_{i,j}\tilde{\beta}_k - x_{i,j}\beta_j)^2 + \lambda\sum_{k\neq j}|\tilde{\beta}_k| + \lambda|\beta_j|,$$

where $\tilde{\beta}$ represents the current estimates of $\beta$ and therefore constants. If we also consider a weight $\omega_i$ is associated with each observation, then the updated $\beta_j$ in this case is:

$$\tilde{\beta}_j(\lambda) \leftarrow \frac{S(\sum_i \omega_i x_{i,j}(y_i - \tilde{y}_i^{(-j)}), \lambda)}{\sum_i \omega_i x_{i,j}^2},$$

where $\tilde{y}_i^{(-j)} = \sum_{k\neq j} x_i\tilde{\beta}_k$. From here, we use the Taylor expansion. So the log-likelihood around "current estimate" is:

$$l(\beta) = -\frac{1}{2n}\sum_{i=1}^{n}\omega_i(z_i - X_i\beta)^2,$$

where working weights $\omega_i = \tilde{\pi}_i(1-\tilde{\pi}_i)$, working response $z_i = X_i\tilde{\beta} + \frac{y_i - \tilde{\pi}_i}{\tilde{\pi}_i(1-\tilde{\pi}_i)}$, and $\tilde{\pi}_i = \frac{exp(X_i\tilde{\beta})}{1+exp(X_i\tilde{\beta})}$.
Finally, similar to regression lasso, the logistic lasso can be written as a penalized weighted least-squares problem like this:

$$min_\beta L(\beta) = -l(\beta) + \lambda\sum_{j=0}^{p}|\beta_j|$$

.

***Pathwise Coordinate-wise Algorithm*** The difference between pathwise coordinate-wise method and coordinate-wise method is that a sequence value of lambda is required for input. There are some steps taken as followed:

- Select a $\lambda_{max}$ for all the estimate $\beta = 0$ which is the inner product ($max_l \langle X_l, y \rangle$).
- Compute the solution with a sequence of descending $\lambda$ from maximum to zero ($\lambda_{max} > \lambda_k > \lambda_{k-1} ... > 0$).
- Initialize coordinate descent algorithms for $\lambda_k$ by the calculated estimate $\beta$ from previous $\lambda_{k+1}$ as a warm start
- By repeating the two steps above, a sequence of optimal coefficients $\beta$ for each descending $\lambda$. When objective function is taken the minimum value ($min_\beta L(\beta)$), the best $\lambda$ could be identified and optimal coefficients $\beta$ could be selected under this best $\lambda$

***5-fold Cross-validation*** In a nutshell, a 5-fold cross-validation first splits the shuffled training data (80% of the whole dataset) into 5 parts and takes one group as the hold-out set (validation set) while fitting the model using the remaining 4 training sets. After a model is fitted, we evaluate and retain the model performance, namely AUC and RSS, and move on to the next iteration. After 5 iterations, we calculated the mean RSS and the mean AUC with standard deviations. Our goal is to find an optimal $\lambda$ that maximizes the mean AUC.

In our implementation, we initially choose 30 equally spaced $\lambda$ from $e^{(-4,3)}$. The reason why the $\lambda$'s are widespread is that we would like to determine the rough range, instead of the precise value, of the optimum of this tuning parameter first. For each $\lambda$, we ran the 5-fold cross-validation. Within each fold, we use Coordinate-wise optimization to update the coefficients vectors $\boldsymbol{\beta}$ and evaluate the performance by calculating AUC and RSS. Among 30 $\lambda$, the optimal $\lambda$ that maximizes the AUC equals 2.3681. We then narrow down the range to (3, 0) and choose 150 equally spaced $\lambda$ in between. In this case, the best $\lambda$ selected by cross-validation equals 1.5705.

# Results

## Newton-Raphson Methods

We apply the modified Newton-Raphson algorithm on the first 10 features(mean) since the result of all 30 features is not good.

Table 1: Newton-Raphson Results of Mean Features

| Features. | Coefficient |
|---|---|
| intercept | 0.4870168 |
| radius_mean | -7.2218505 |
| texture_mean | 1.6547562 |
| perimeter_mean | -1.7376303 |
| area_mean | 14.0048456 |
| smoothness_mean | 1.0749533 |
| compactness_mean | -0.0772346 |
| concavity_mean | 0.6751231 |
| concave_points_mean | 2.5928743 |
| symmetry_mean | 0.4462563 |
| fractal_dimension_mean | -0.4824842 |

We also compare this to the `glm` result and check if they match in the first three digits. The results come to all true and this model fits well.

## Pathwise Logistic-LASSO Model

For the Pathwise logistic-lasso model, we first select $\lambda_{max} = 218.13$ to maximize the inner product of response and all variables. The lasso solution path is shown in **(Fig 3)**. With a goal to minimize the objective function and have a relatively high AUC, we tried a sequence of $\lambda$ from 0 to 3 and found that the minimum value would be taken under when $\lambda = 1.57$ **(Fig 4)**, which means the optimal solution will be computed for this optimal $\lambda$ value.

**Fig 3 Lasso Solution Path**

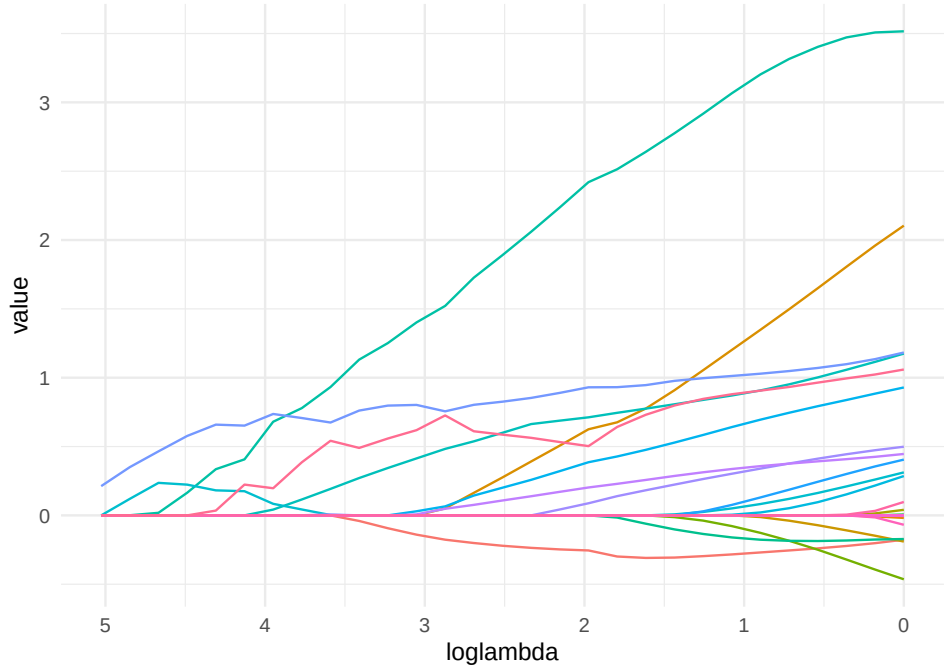



**Fig 4 Comparing the AUC for different lambda**

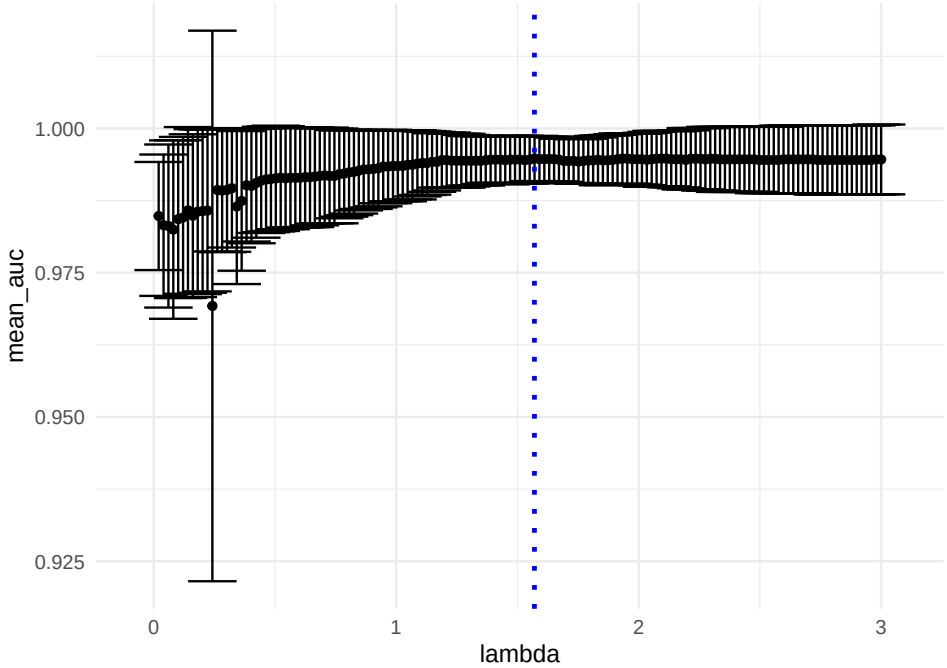


Table 2: Comparing the coefficients for optimal and full model

| beta | full_coefficients | optimal_coefficients |
|---|---|---|
| Intercept | -6.7120020 | -0.2307176 |
| radius_mean | 15.3615092 | 0.0000000 |
| texture_mean | 1.2214100 | 0.0000000 |
| perimeter_mean | 6.6009820 | 0.0000000 |
| area_mean | 2.0001412 | 0.0000000 |
| smoothness_mean | 3.8129595 | 0.0000000 |
| compactness_mean | 3.6996305 | 0.0000000 |
| concavity_mean | 6.3315880 | 0.1514531 |
| concave points_mean | -4.5445495 | 0.9240514 |
| symmetry_mean | 6.0177074 | 0.1887078 |
| fractal_dimension_mean | -3.7150361 | 0.0000000 |
| radius_se | 0.8185762 | 0.0000000 |
| texture_se | -1.9716802 | 0.0000000 |
| perimeter_se | 8.6455676 | 0.0000000 |
| area_se | 0.7532164 | 0.0000000 |
| smoothness_se | -7.2858931 | 0.0000000 |
| compactness_se | -2.9384468 | 0.0000000 |
| concavity_se | 1.4394241 | 0.0000000 |
| concave points_se | -13.9518630 | 0.0000000 |
| symmetry_se | -3.8392690 | 0.0000000 |
| fractal_dimension_se | 3.4956263 | 0.0000000 |
| radius_worst | 18.2512163 | 0.0000000 |
| texture_worst | 6.7310378 | 0.1302448 |
| perimeter_worst | 19.3803433 | 3.7063624 |
| area_worst | 7.7215211 | 0.0000000 |
| smoothness_worst | 17.3699841 | 0.3410588 |
| compactness_worst | 2.7156803 | 0.0000000 |
| concavity_worst | 7.1459593 | 0.2560776 |
| concave points_worst | -3.1117336 | 0.0196025 |
| symmetry_worst | -4.6306316 | 0.0203774 |
| fractal_dimension_worst | 6.3252661 | 0.0000000 |

In the above table, we directly compare the coefficients estimates $\beta$ for all 30 variables and $\beta_0$ for intercept. In the optimal lasso model, there are only ten variables included while the rest predictors are all shrinked towards zero.

### Comparsion: "Optimal" model vs. "Full" model

As the name suggested, the "optimal" model has $\lambda = 1.5705$ as its tuning parameter while the "full" model has $\lambda = 0$. We evaluate both models and obtain their AUC scores: optimal model has an AUC of 0.9994 and the full model has an AUC of 1. Thus, we concluded that the full model has a better prediction, even though the prediction performance of these two models only differ by 0.0006.

## Conclusion and Discussion

Here, we totally use three different optimization methods and try to compare the performance of different methods. Based on the results of coefficients, we could tell that Newton-Raphson has a relatively bad performance for this dataset and we guess the using of inverse of Hessian matrix is actually not accurate, especially when predictors are large so that we guess we could try alternatives to replace Hessian matrix.

When concentrating on the results of full and optimal model selected by pathwise coordinate-wise lasso model, it's quite interesting that the full model has a perfect classification result. One plausible reason could be that the data is well separated for benign and malignant breast cancer.

## Contributions

We contributed to this project evenly.

## Appendix

```
library(tidyverse)
library(ggplot2)
library(corrplot)
library(pROC)
library(caret)

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

options(
  ggplot2.continuous.colour = "viridis",
  ggplot2.continuous.fill = "viridis"
)

scale_colour_discrete = scale_colour_viridis_d
scale_fill_discrete = scale_fill_viridis_d

## Data import

# --- 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))
```

```r
# --- 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]

# --- 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

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

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

#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))

#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 <- breast_cancer$diagnosis
  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)

    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

# 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


## 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))

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

  return(res)
}

## path-wise coordinate-wise for lasso solution path

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

  pathwise_res <- NULL

  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)]
```

```
    pathwise_res <- rbind(pathwise_res,
                          c(j, lambda = lambda[j], beta = betavec))
  }

  return(pathwise_res[-1, ])
}

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)[, -1] %>%
  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(show.legend = FALSE) + scale_x_reverse()

## 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, gstat = 0)

  for (i in 1:length(lambda_vec)) {
    curlambda = lambda_vec[i]
    fold_auc = list(); fold_rss = list(); fold_gstat = list()

    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)
      fold_gstat[k] = sum((validation_y - prob)^2 / (prob * (1 - prob)))
    }

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

```
  }

  return(final_res)
}

set.seed(2022)
cv_res2 <- cv(bc_scale2, lambda_vec = exp(seq(log(max_lambda2), 0, length = 30)), betavec = rep(1, 31))

### Compare optimal with full using test data

opt_lambda = 1.57

# Optimal lambda
optimal = cd_lasso(t_bc_scale2_x, t_bc_scale2_y, lambda = opt_lambda, betavec = rep(0.5,31))[,-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]

# Without lambda
full = cd_lasso(t_bc_scale2_x, t_bc_scale2_y, lambda = 0, betavec = rep(0.5,31))[,-c(1:2)] %>%
  as.matrix()
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]


### 5-fold cv mean auc & sd auc

# mean & sd
as.tibble(cv_res2) %>%
  group_by(lambda) %>%
  summarise(mean_auc = mean(auc),
            sd = sd(auc)) %>%
  ggplot(aes(x = log(lambda), y = mean_auc)) +
  geom_point() +
  geom_errorbar(aes(ymin = mean_auc - sd, ymax = mean_auc + sd), width = .2,
                position = position_dodge(.05))



### Lasso Solution Path

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

# pathwise cd_lasso
res <- pathwise_cd_lasso(bc_scale1_x, bc_scale1_y, rep(1, 31), exp(seq(max_lambda2, 0, length = 50)))

as.tibble(res) %>%
  pivot_longer(4:dim(res)[2] - 1, names_to = "beta", values_to = "value") %>%
  ggplot(aes(x = log(lambda), y = value, color = beta)) +
  geom_line(show.legend = FALSE) + scale_x_reverse(limits = c(log(max_lambda2), 0))
```

```
result = read.csv("cv_results.csv")[-139,] %>%
    filter(lambda != 0) %>%
    ggplot(aes(x = lambda, y = mean_auc)) +
    geom_point() +
    geom_errorbar(aes(ymin = mean_auc - sd_auc, ymax = mean_auc + sd_auc), width = .2,
                  position = position_dodge(.05)) +
    geom_vline(xintercept = 1.57, linetype="dotted",
               color = "blue", size=1)
result
```

# Reference