Breast Cancer Diagnosis using Optimization

Margaret Gacheru (mg3861), Alyssa Vanderbeek (amv2187), Jack Yan (xy2395), Mengyu Zhang (mz2777)

Introduction

In this project, we build a model to classify breast tissue images into malignant cancer or benign tissue. We are interested in observing how the certain features for individual breast tissue are influential in determining cancer diagnosis. Prediction algorithms are potentially useful in this area as a way to aid in the diagnosis of breast cancer.

Our data is taken from tissue imaging from 569 breast cancer patients. Our goal was to build a prediction algorithm from the 30 variables provided. First, we built a full logistic model with all 30 features and utilized a Newton-Raphson algorithm in order to estimate the model coefficients. Furthermore, we built a logistic-lasso model to perform variable selection in order to identify the best subset of tissue characteristics that play a major role in prediction.

Methods

The dataset contains 569 individuals with 30 variables describing the tissue images. These variables correspond to mean, standard deviation, and largest value of 10 tissue characteristics, including, radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension.

Logistic Regression Model

The logistic regression model can be defined as

$$\log(\frac{\pi}{1-\pi}) = X\beta$$

where the link function is $\log(\frac{\pi}{1-\pi})$.

The likelihood function is

$$L(\beta; X, y) = \prod_{i=1}^{n} \left\{ \left(\frac{\exp(X_i\beta)}{1+\exp(X_i\beta)} \right)^{y_i} \left(\frac{1}{1+\exp(X_i\beta)} \right)^{1-y_i} \right\}$$

where $y_i \sim bin(1, \pi_i)$,

$$y_i = \begin{cases} 1, & malignant\ tissue \\ 0, & benign\ tissue \end{cases}$$

Consequently, the log-likelihood function is

$$l(\beta) = \sum_{i=1}^{n} \{ y_i (X_i\beta) - \log(1+\exp(X_i\beta)) \}$$

Maximizing log-likelihood function is equivalent to maximizing likelihood function, so the gradient and Hessian matrix will be

$$\nabla l(\boldsymbol{\beta}) = \begin{pmatrix} \sum_{i=1}^{n} y_i - p_i \\ \sum_{i=1}^{n} \mathbf{x}_i (y_i - p_i) \end{pmatrix}_{(p+1)\times 1}$$

and

$$\begin{aligned} \nabla^2 l(\boldsymbol{\beta}) &= -\sum_{i=1}^{n} \begin{pmatrix} 1 \\ \mathbf{x}_i \end{pmatrix} \begin{pmatrix} 1 & \mathbf{x}_i^T \end{pmatrix} p_i (1-p_i) \\ &= -\begin{pmatrix} \sum p_i(1-p_i) & \sum \mathbf{x}_i^T p_i (1-p_i) \\ \sum \mathbf{x}_i p_i (1-p_i) & \sum \mathbf{x}_i \mathbf{x}_i^T p_i (1-p_i) \end{pmatrix} \end{aligned}$$

where $p_i = P(Y_i = 1|\mathbf{x}_i) = \frac{\exp(\beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}_1)}{1+\exp(\beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}_1)}$.

With p = 30 predictors, we obtain a 31 $\times$ 1 gradient vector and 31 $\times$ 31 Hessian matrix

$$\nabla l\ (\boldsymbol{\beta}) = \boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{\pi}),\ \boldsymbol{\pi} = \frac{e^{\boldsymbol{X}\boldsymbol{\beta}}}{1 + e^{\boldsymbol{X}\boldsymbol{\beta}}}$$

where $\boldsymbol{X}$ is the design matrix

$$H = \boldsymbol{X}^T \boldsymbol{\Lambda} \boldsymbol{X}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix with $var(y_i),\ i = 1, ..., n$ as the diagonal elements

Algorithms

Newton-Raphson Algorithm

Newton-Raphson algorithm is a method to search for solutions to the system of equations $\nabla l(\boldsymbol{\beta}) = 0$. At each step, given the current point $\boldsymbol{\beta}_0$, the gradient $\nabla l(\boldsymbol{\beta})$ for $\boldsymbol{\beta}$ near $\boldsymbol{\beta}_0$ may be approximated by

$$\nabla l(\boldsymbol{\beta}_0) + \nabla^2 l(\boldsymbol{\beta}_0)(\boldsymbol{\beta} - \boldsymbol{\beta}_0)$$

The next step in the algorithm is determined by solving the system of linear equations

$$\nabla l(\boldsymbol{\beta}_0) + \nabla^2 l(\boldsymbol{\beta}_0)(\boldsymbol{\beta} - \boldsymbol{\beta}_0) = \mathbf{0}$$

and the next "current point" is set to be the solution, which is

$$\boldsymbol{\beta}_1 = \boldsymbol{\beta}_0 - \left[\nabla^2 l(\boldsymbol{\beta}_0)\right]^{-1} \nabla l(\boldsymbol{\beta}_0)$$

The ith step is given by

$$\boldsymbol{\beta}_i = \boldsymbol{\beta}_{i-1} - \left[\nabla^2 l(\boldsymbol{\beta}_{i-1})\right]^{-1} \nabla l(\boldsymbol{\beta}_{i-1})$$

Coordinate-wise descent with regularized logistic regression

The logistic-lasso can be written as a penalized weighted least-squares problem

$$\min_{(\beta_0,\boldsymbol{\beta}_1)} L(\beta_0,\boldsymbol{\beta}_1,\lambda)=\left\{-\ell(\beta_0,\boldsymbol{\beta}_1)+\lambda\sum_{j=0}^{p}|\beta_j|\right\}$$

When the p, the number of parameters, is large, the optimization could be challenging. Therefore, a coordinate-wise descent algorithm will be applied. The objective function is

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

Minimizing $f(\beta_j)$ w.r.t β_j while having $\widetilde{\beta}_k$ fixed, we have weighted beta updates

$$\widetilde{\beta}_j(\gamma)\leftarrow\frac{S\left(\sum_i w_i x_{i,j}\left(y_i-\tilde{y}_i^{(-j)}\right),\gamma\right)}{\sum_i w_i x_{i,j}^2}$$

where $\tilde{y}_i^{(-j)}=\sum_{k\neq j}x_{i,k}\widetilde{\beta}_k$.

If we Taylor expansion the log-likelihood around "current estimates" $\left(\widetilde{\beta}_0,\tilde{\beta}_1\right)$, we have a quadratic approximation to the log-likelihood

$$f(\beta_0,\boldsymbol{\beta}_1)\approx\ell(\beta_0,\boldsymbol{\beta}_1)=-\frac{1}{2n}\sum_{i=1}^{n}w_i\left(z_i-\beta_0-\mathbf{x}_i^T\boldsymbol{\beta}_1\right)^2+C\left(\widetilde{\beta}_0,\widetilde{\boldsymbol{\beta}}_1\right)$$

where

$$z_i=\widetilde{\beta}_0+\mathbf{x}_i^T\widetilde{\boldsymbol{\beta}}_1+\frac{y_i-\widetilde{p}_i(\mathbf{x}_i)}{\widetilde{p}_i(\mathbf{x}_i)(1-\widetilde{p}_i(\mathbf{x}_i))}$$

$$w_i=\widetilde{p}_i(\mathbf{x}_i)(1-\widetilde{p}_i(\mathbf{x}_i))$$

$$\widetilde{p}_i=\frac{\exp\left(\widetilde{\beta}_0+\mathbf{x}_i^T\widetilde{\boldsymbol{\beta}}\right)}{1+\exp\left(\widetilde{\beta}_0+\mathbf{x}_i^T\widetilde{\boldsymbol{\beta}}_1\right)}$$

Therefore, the algorithm's steps are

Step 1. Find λ_{max} such that all the estimated β are zero;

Step 2. Define a fine sequence $\lambda_{max}\geq\lambda_1\geq...\geq\lambda_{min}\geq 0$;

Step 3. Defined the quadratic approximated objective function $L(\beta_0,\boldsymbol{\beta}_1,\lambda)$ for λ_k using the estimated parameter at λ_{k-1} $(\lambda_{k-1}>\lambda_k)$.

Step 4. Run a coordinate-wise descendent algorithm to optimize the function defined in Step 3.

Results

This optimal model was estimated using the above steps in combination with 5-fold cross-validation. To find the optimal value of λ, we fit the model for a range of values between 0.01 and 54.5 (equal to -4 to 4 on the log scale, Figures 1 and 2). The optimal λ was found to be 1.51 ($log(\lambda) = 0.41$). Note that at this value that we see the path of solutions begin to diverge (Figure 1). The model with this $\lambda = 1.51$ contains 16 predictors, from the 30 original covariates in the dataset (Table 1). We use the AUC to assess predictive performance, which in this case is equal to 0.9946 (Figure 2).

We compare this to the "full model", which is a standard logistic model estimated with Newton-Raphson that uses all 30 covariates in the dataset. The model is also trained using 5-fold cross-validation. Its AUC is 0.9394.

Table 1: Coefficient estimates for the 16 covariates in the optimal model, selected by LASSO and using pathwise coordinate-wise optimization

	Optimal model coefficients
texture_mean	0.1492035
concave.points_mean	0.9665191
fractal_dimension_mean	-0.0549282
radius_se	2.1766218
texture_se	-0.1323202
smoothness_se	0.1717834
compactness_se	-0.5951974
fractal_dimension_se	-0.2184259
radius_worst	1.5752823
texture_worst	1.3400344
area_worst	2.7737454
smoothness_worst	0.5564578
concavity_worst	0.8848153
concave.points_worst	1.1901389
symmetry_worst	0.4808392

Figure 1: A path of solutions with a sequence of descending lambda's

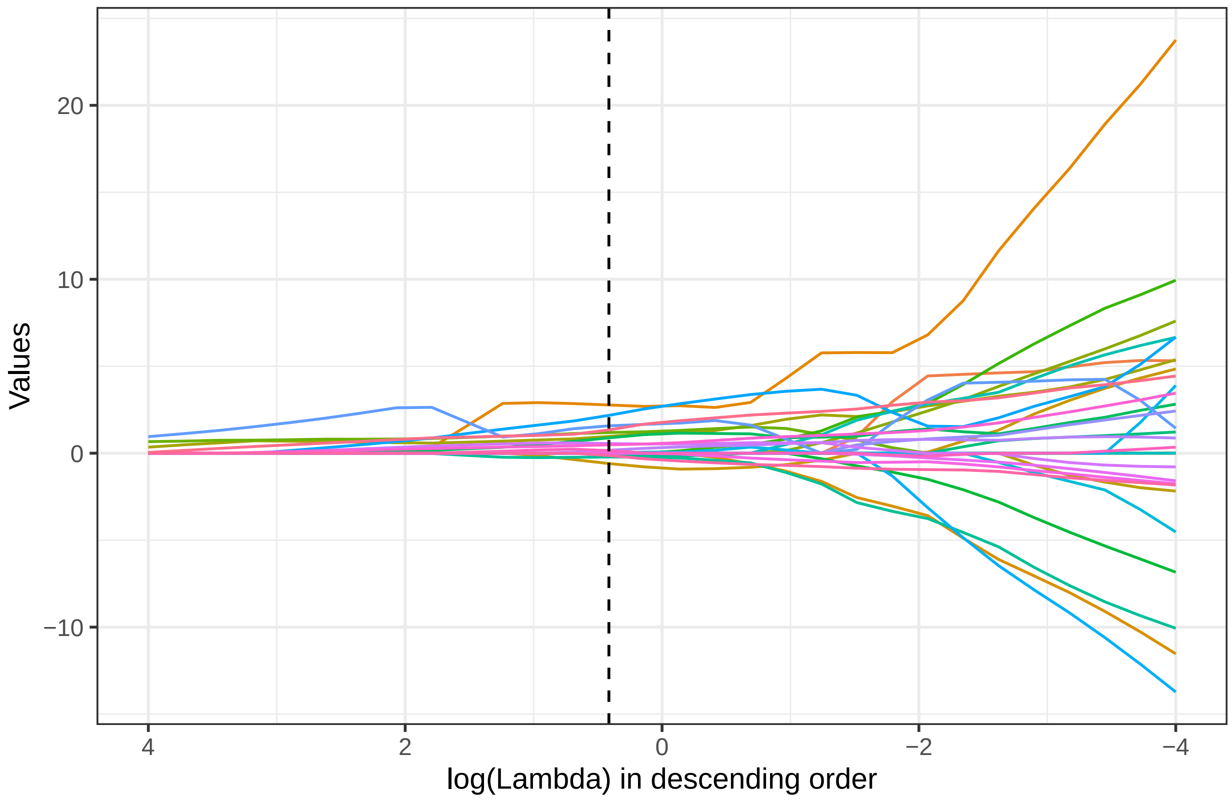
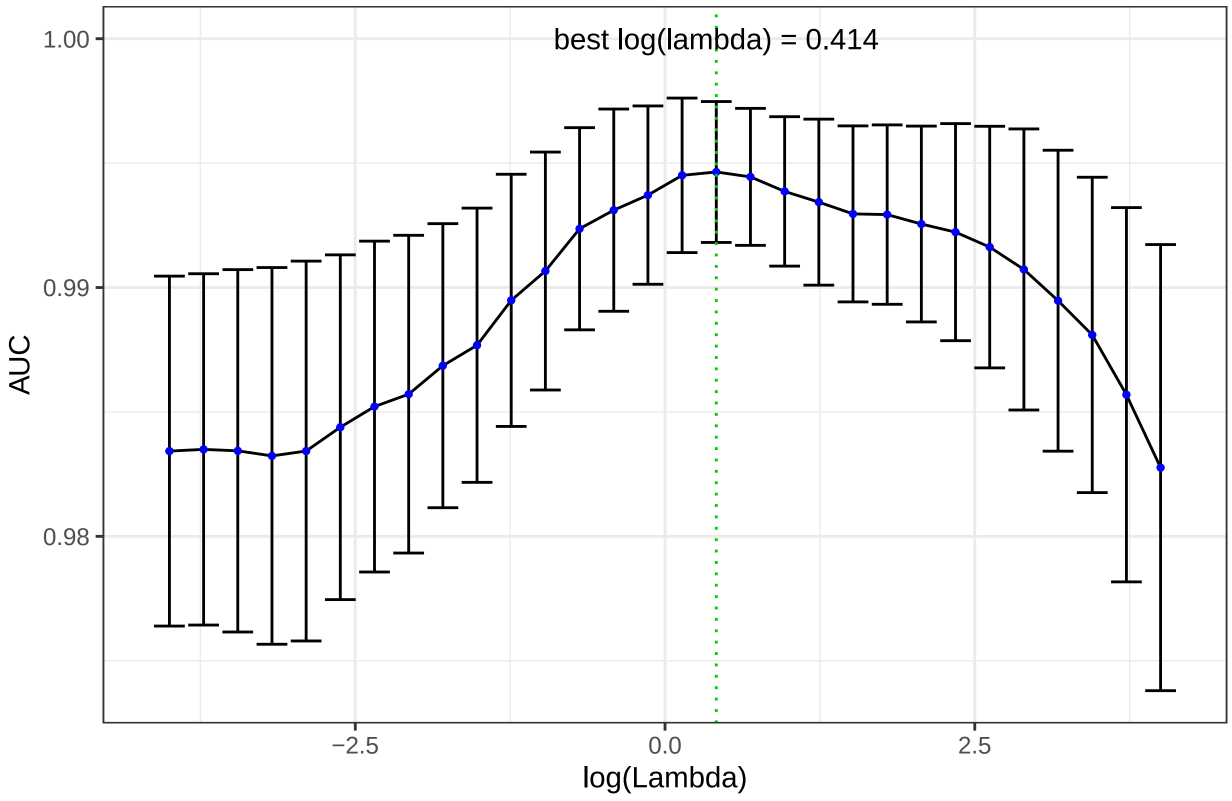


Figure 2: LASSO−logistic regression by 5 fold cross validation to find the o



Conclusions

The goal of this project was to develop a model to predict whether a breast tissue sample was benign or malignant cancer. We estimated both an optimal model using LASSO variable selection and pathwise coordinate-wise optimization, and compare this to a logistic model using all available covariates estimated from a Newton-Raphson algorithm.

There are several benefits to using LASSO selection and pathwise coordinate-wise optimization approach to this classification problem. First, the predictive performance is high and favorable compared to a full logistic regression model (AUC of 0.9946 vs. 0.9394). Secondly, the Newton-Raphson estimates for the full model were highly dependent on the chosen starting value. This makes it difficult to determine what the global maximum is, and can therefore affect the accuracy of the estimates. Finally, the optimal has the advantage of parsimony.

In sum, we found that using LASSO variable selection and pathwise coordinate-wise optimization algorithm for model estimation produced a highly effective prediction model for diagnosis of cancer from breast tissue images.

Appendix

```r
breast_cancer <- data.frame(read_csv("breast-cancer.csv")[,-33])

cancer = breast_cancer %>%
  mutate(diagnosis = recode(diagnosis, "M" = 1, "B" = 0))
cancer_ls = list(x = cancer[,3:32], y = cancer$diagnosis)
x = sapply(cancer_ls$x, function(x) as.numeric(unlist(x)))
cancer_ls = list(x = x, y = cancer$diagnosis)
```

```r
logisticstuff <- function(dat, betavec){
  x<- dat$x
  xm <- cbind(rep(1, nrow(x)), scale(x)) # standardize the data
  u <- xm %*% betavec
  expu <- exp(u)
  j <- 1
  loglik_temp <- matrix()
  p <- matrix()

  for(j in 1:nrow(xm)){
    loglik_temp[j] <- ifelse(u[j] > 10,
                             sum(dat$y[j] * u[j]  - u[j]),
                             sum(dat$y[j] * u[j]  - log(1 + expu[j])))
    p[j] <- ifelse(u[j] > 10, 1, expu[j] / (1 + expu[j]))
  j <- j+1
  }

  loglik <- sum(loglik_temp)
  grad <- matrix(colSums(xm * as.vector(dat$y - p))) # gradient at betavec
  Hess <- 0
  i = 1
  for (i in 1:nrow(xm)) {
    tt <- xm[i,]
    dd <- t(tt)
    Hess <- Hess - tt %*% dd * p[i] * (1 - p[i])
    i <- i + 1
  }
  return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

```
NewtonRaphson <- function(dat, func, start, tol=1e-10, maxiter = 20000) {
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf      # To make sure it iterates
  while(i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    cur <- prev - ginv(stuff$Hess, 2.34406e-18) %*% stuff$grad
    prevstuff <- stuff
    stuff <- func(dat, cur)        # log-lik, gradient, Hessian
    gamma <- 0.01

    while(max(eigen(stuff$Hess)$value)>0){
      stuff$Hess = stuff$Hess - diag(31) * gamma
      gamma <- gamma + 0.01
    }

    if (stuff$loglik > prevloglik)
    {
        res <- rbind(res, c(i, stuff$loglik, cur))# Add current values to results matrix
    } else
    {
      lambda <- 1
      while (stuff$loglik < prevloglik) {
        lambda <- lambda / 2 # step-halving
        cur <- prev - lambda * ginv(prevstuff$Hess, 2.34406e-18) %*% prevstuff$grad
        stuff <- func(dat, cur)        # log-lik, gradient, Hessian
      }
        res <- rbind(res, c(i, stuff$loglik, cur))# Add current values to results matrix
    }
  }
  return(res)
}
```

```
NewRes <- NewtonRaphson(dat = cancer_ls, func = logisticstuff, start = rep(0, 31))
NewRes
#res1 = NewtonRaphson(dat = cancer_ls, func = logisticstuff, start = rep(1, 31))


glm(diagnosis ~ ., family = binomial("logit"), data  = cancer[, -1], start = rep(0, 31))

check <- tail(NewRes)
check
```

```
Sfunc <- function(beta,lambda) {

  if ((abs(beta)-lambda) > 0)
  {
    return (sign(beta) * (abs(beta)-lambda))
    }
  else {
    return (0)
    }
}


coordlasso <- function(lambda, data_ls, start, tol=1e-10, maxiter = 2000){
  x<- data_ls$x
  xm <- cbind(rep(1, nrow(x)), scale(x)) # standardize the data
  i <- 0
  pp <- length(start)
  n <- length(data_ls$y)
  betavec <- start
  loglik <- 0
  res <- c(0, loglik, betavec)
  prevloglik <- -Inf # To make sure it iterates
  while (i < maxiter && abs(loglik - prevloglik) > tol && loglik < Inf) {
    i <- i + 1
    prevloglik <- loglik
    for (j in 1:pp) {
      u <- xm %*% betavec
      expu <- exp(u)
      prob <- expu / (expu + 1)
      w <- prob * (1 - prob) # weights
      # avoid coeffcients diverging in order to achieve fitted  probabilities of 0 or 1.
      w <- ifelse(abs(w-0) < 1e-5, 1e-5, w)
      z <- u + (data_ls$y - prob) / w
      # calculate noj
      z_j <- xm[,-j] %*% betavec[-j]
      betavec[j] <- Sfunc(sum(w * (xm[,j]) * (z - z_j)), lambda) / (sum(w * xm[,j] * xm[,j]))
    }
    loglik <- sum(w * (z - xm %*% betavec)^2) / (2 * n) + lambda * sum(abs(betavec))
    res <- rbind(res, c(i, loglik, betavec))}
  return(res)
}
```

```
CorRes <- coordlasso(lambda = 0.8, cancer_ls, start = rep(0, 31) ,maxiter = 2000)
CorRes

#logmod <- glmnet(cancer_ls$x, y=cancer_ls$y, alpha=1, family="binomial",lambda = 0.8)
#coef.glmnet(logmod)
```

```
path <- function(data_ls,lambdas){
  start <- rep(0, 31)
  betas <- NULL
  for (x in 1:length(lambdas)) {
  coor_result <- coordlasso(lambda = lambdas[x],
                            data_ls = data_ls,
                            start= start,
                            maxiter = 2000)
  curbeta <- coor_result[nrow(coor_result),3:dim(coor_result)[2]]
```

```r
  start <- curbeta
  betas <- rbind(betas,c(curbeta))
  }
  return(data.frame(cbind(lambdas,betas)))
}

path_sol <- path(cancer_ls,lambdas = exp(seq(4,-4, length=30))) # 5 min
colnames(path_sol) <- c("lambdas","intercept", colnames(cancer_ls$x))
```

```r
# data manipulaiton
data_ma <- function(data){
  data_ls = list(x = data[,3:32], y = data$diagnosis)
  x = sapply(data_ls$x, function(x) as.numeric(unlist(x)))
  data_ls = list(x = x, y = data$diagnosis)
  return(data_ls)
}

#Randomly shuffle the data

cv <- function(org_data, lambdas, K){ #original data
  org_data <- org_data[sample(nrow(org_data)),]
  #Create K equally size folds
  folds <- cut(seq(1,nrow(org_data)),breaks = K,labels = FALSE)
  cv_AUC <- vector()
  cv_se <- vector()
  #Perform 10 fold cross validation
  for (j in 1:length(lambdas)) {
  cv_errors <- vector()
  for(i in 1:K){
    start <- rep(0, 31)
    #Segement your data by fold using the which() function
    testIndexes <- which(folds == i, arr.ind = TRUE)
    testData <- data_ma(org_data[testIndexes, ])
    xt <- cbind(rep(1, nrow(testData$x)), scale(testData$x)) # standardize the data
    testData <- list(x = xt, y = testData$y)

    trainData <- data_ma(org_data[-testIndexes, ])
    coor_result <- coordlasso(lambda = lambdas[j],
                              data_ls = trainData,
                              start= start)
    curbeta <- coor_result[nrow(coor_result),3:dim(coor_result)[2]]
    u <- as.matrix(testData$x) %*% curbeta
    expu <- exp(u)
    prob <- expu / (1 + expu)
    cv_errors[i] = AUC(prob, testData$y) #MSE
  }
  cv_AUC[j] <- mean(cv_errors)
  cv_se[j] <- sqrt(var(cv_errors)/K)
  }
  return(cbind(lambdas, cv_AUC, cv_se))
}

set.seed(123)
cv_res <- cv(org_data = cancer, lambdas = exp(seq(4,-4, length=30)), K = 5) #15 min
```

```r
best_lambda <- cv_res[which.max(cv_res[,2]),1]
cv_res <- data.frame(cv_res)
cv_res %>%
  filter(lambdas == best_lambda)
```

```r
cv_New <- function(org_data, func, K){ #original data
  org_data <- org_data[sample(nrow(org_data)),]
  #Create K equally size folds
  folds <- cut(seq(1,nrow(org_data)),breaks = K,labels = FALSE)
  cv_errors <- vector()
  #Perform 10 fold cross validation
  for(i in 1:K){
    start <- rep(0, 31)
    #Segement your data by fold using the which() function
    testIndexes <- which(folds == i, arr.ind = TRUE)
    testData <- data_ma(org_data[testIndexes, ])
    xt <- cbind(rep(1, nrow(testData$x)), scale(testData$x)) # standardize the data
    testData <- list(x = xt, y = testData$y)
    trainData <- data_ma(org_data[-testIndexes, ])

    New_result <- NewtonRaphson(dat = trainData,
                                func = func,
                                start= start)
    curbeta <- New_result[nrow(New_result),3:dim(New_result)[2]]
    u <- as.matrix(testData$x) %*% curbeta
    prob <- vector()
    for(j in 1:nrow(testData$x)){
    prob[j] <- ifelse(u[j] > 10, 1, exp(u[j]) / (1 + exp(u[j])))
    j <- j+1
    }

    cv_errors[i] = AUC(prob, testData$y) #AUC
  }
  cv_error <- mean(cv_errors)
  cv_se <- sqrt(var(cv_errors)/K)

  return(cbind(cv_error, cv_se))
}

set.seed(123)
cv_new_res <- cv_New(org_data = cancer, func = logisticstuff, K =5)
cv_new_res
```