Predictive Model Based on Logistic Regression for Cancer Diagnosis

Yijing Tao; Renjie Wei; Jibei Zheng; Haolin Zhong; Anyu Zhu

3/27/2022

Introduction

Background and Objective

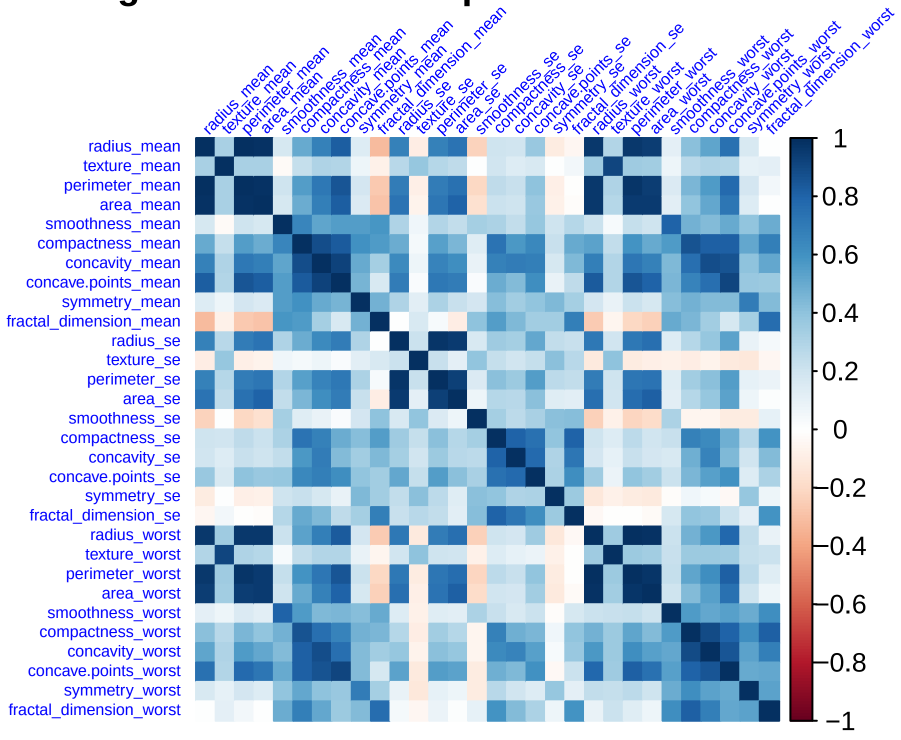
Breast cancer mainly occurs in middle-aged and older women. The median age at the time of breast cancer diagnosis is 62. This means half of the women who developed breast cancer are 62 years of age or younger when they are diagnosed. The goal of the project is to build a predictive model based on logistic regression to facilitate cancer diagnosis. We first build a logistic model to classify the images, then developed a Newton-Raphson algorithm to estimate the parameters of the logistic model. Then, we built a logistic-LASSO model to select features. Finally, we applied 5-fold cross-validation to select the best λ for the logistic-LASSO model.

Data Preprocessing

The dataset 'breast-cancer'we used contains 569 rows and 32 columns. The variable `diagnosis` identifies if the image is coming from cancer tissue or benign. We labeled `malignant` as 1 and `benign` as 0. In total there are 212 malignant cases and 357 benign cases. There are 30 variables corresponding to mean, standard deviation and the largest values (points on the tails) of the distributions of 10 features: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension.

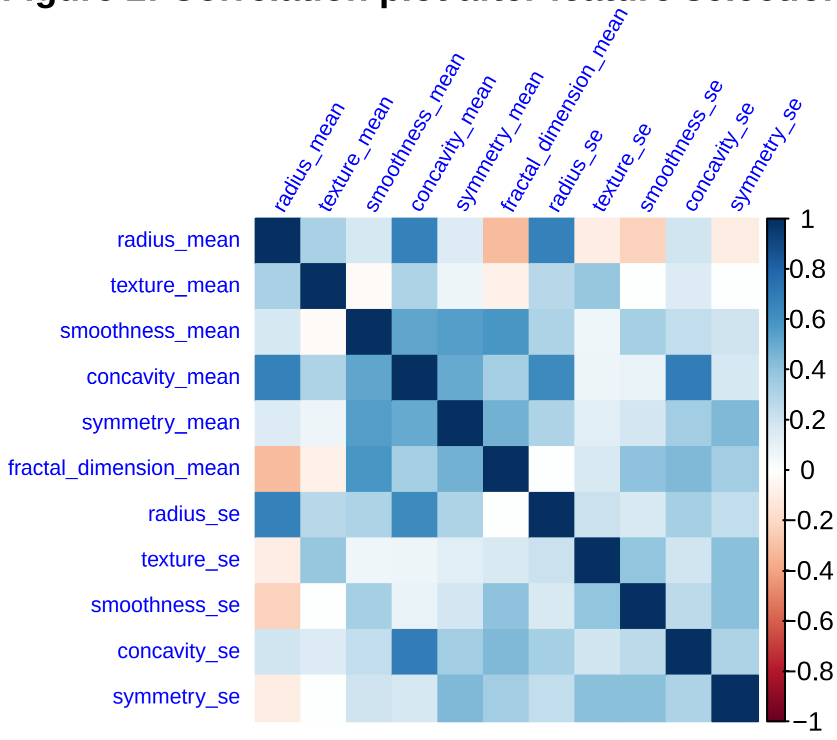
Figure q displays the correlation between variables. We can see the correlation coefficient is large between several pairs of variables, which will potentially cause the problem of not converging in Newton-Raphson algorithm and Logistic Lasso model.

Figure 1: Correlation plot of all variables



To reduce the multicolinearity effect, we conducted feature selection by removing variables with correlation coefficient > 0.7 and keep the rest 11 variables. After the adjustment, the correlation plot between variables change to:

Figure 2: Correlation plot after feature selection



We standardized the data by the `scale()` function in R, take the first 80% of observations as training dataset, and the rest 20% of observations as testing dataset for model comparison.

Method

Logistic Model

Take Y_i as the response of i_{th} observation and follows binary distribution $Bin(\pi_i)$. π_i is the probability of i_{th} observation being malignant. By applying the logit link:

$$g(\mu) = \operatorname{logit}(\mu) = \log \frac{\mu}{1-\mu}$$

we have the logistic regression model:

$$\log \frac{\pi_i}{1-\pi_i} = X_i \beta$$

Thus we have the likelihood function of logistic regression

$$L(\pi) = \prod_{i=1}^{n} f\left(y_i\right) = \prod_{i=1}^{n} \pi_i^{y_i} \left(1-\pi_i\right)^{1-y_i}$$

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

Then maximize the log likelihood:

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

By taking derivative with respect to β, the gradient is:

$$\nabla l(\beta) = \sum_{i=1}^{n} \left(y_i - \pi_i\right) \boldsymbol{x}_i = X^T (Y - \boldsymbol{\pi})$$

where $\pi_i = \frac{e^{\beta_i}}{1+e^{\beta_i}}$

By taking the second derivative, the Hessian matrix can be represented by:

$$\nabla^2 l(\beta) = -X^T \operatorname{diag}\left(\pi_i \left(1-\pi_i\right)\right) X$$

i = 1, … n. Hessian matrix is negative definite.

5-Fold Cross Validation

The test data is taken from the same data, but is not involved in the training, so that the model can be evaluated relatively objectively to match the data outside the training set. The evaluation of the model in the test data is commonly done by cross-validation. It divides the original data into K groups (K-Fold), and makes each subset of data into a test set separately, and the rest of the K-1 subset data as the training set, so that K models will be obtained. These K models are evaluated separately in the test set, and the final error MSE (Mean Squared Error) is summed and averaged to obtain the cross-validation error.

In order to find the best lambda, we use 5-fold cross validation. The dataset is divided into five subdatasets by using the *crossv_kfold* function. We combined 4 of them as the training data set, and the rest $\frac{1}{5}$ of them as the test data set each time. The optimal coefficients is then found five times by running the logit-LASSO on the training data set, leaving a different subset out each time. The subset left out is then used to estimate the model performance. This is done for all lambdas in the pre-defined sequence in order to search for the lambda with the highest average predictive ability. We use AUC as the criteria to choose the best tuning

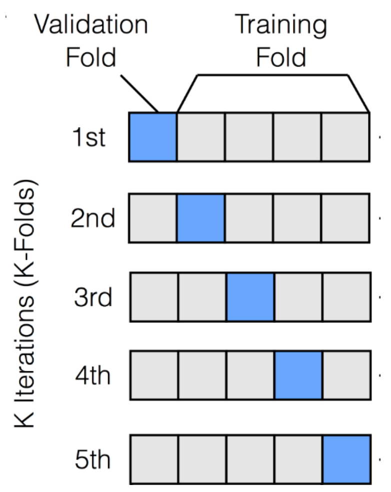


Figure 1: Caption for the picture.

parameter and corresponding model obtained from train data since the response of this data set is binary. We then calculating AUC to evaluate the model performance in test dataset and choose the best lambda whose AUC is the biggest.

After the 5-fold cross validation, we found that the best lambda is about $e^{-1.273} = 0.280$ in the optimal model when set.seed(2022).

Appendix

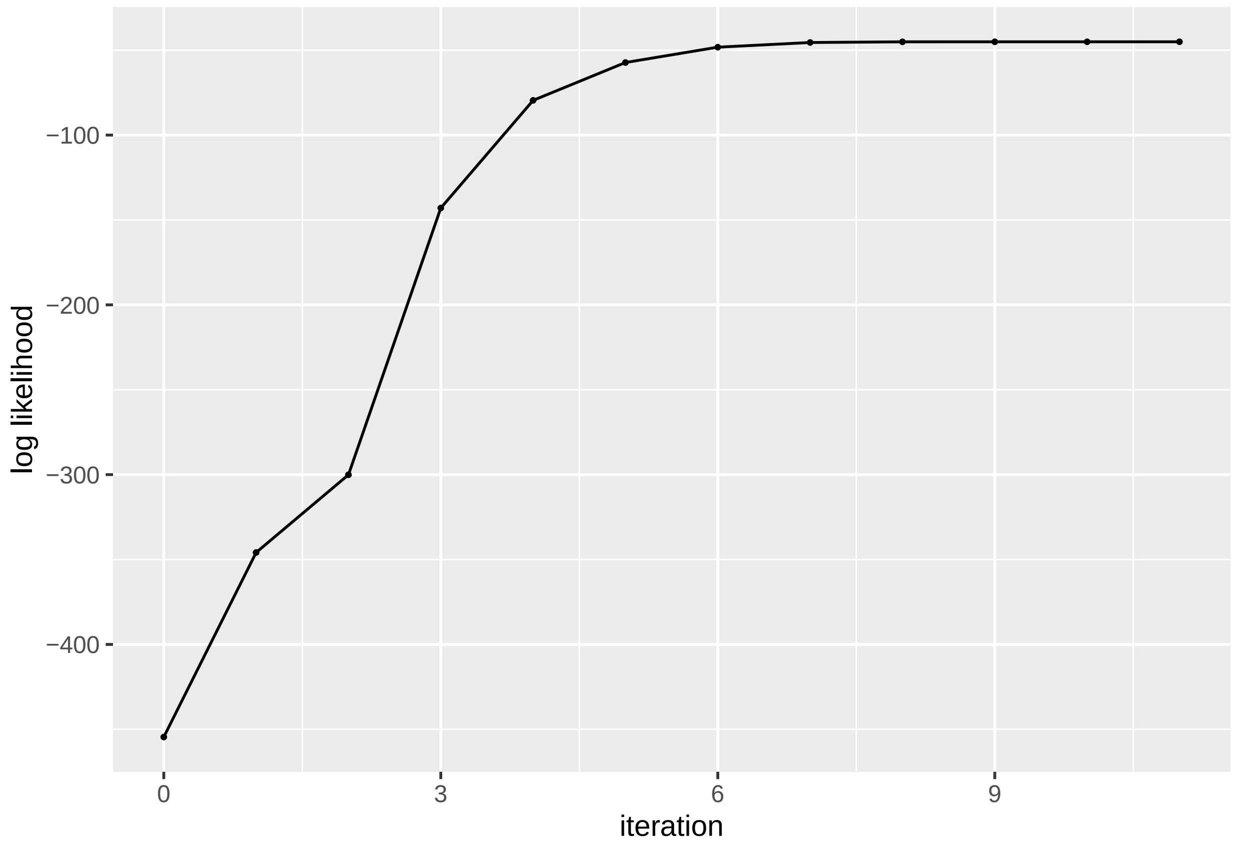
```
# data preprocessing
normalize <- function(col){
    mean = mean(col)
    std = sd(col)
    return((col - mean)/std)
}
cancer = read.csv("breast-cancer.csv") %>%
  mutate(diagnosis = as.numeric(factor(diagnosis)) - 1) %>%
  dplyr::select(-id)

y = as.matrix(cancer$diagnosis)
x = cancer %>%
  dplyr::select(-diagnosis, -perimeter_mean, -area_mean, -concave.points_mean, -radius_worst,
         -area_se, -perimeter_worst, -area_worst, -concave.points_worst, -texture_worst,
         -smoothness_worst, -compactness_se, -compactness_mean, -compactness_worst, -concavity_worst,
         -fractal_dimension_worst, -perimeter_se, -concave.points_se, -fractal_dimension_se, -symmetry_
    map_df(.x = ., normalize) %>% as.matrix()
colMeans(x)
```

```
##           radius_mean           texture_mean        smoothness_mean
##          -1.383450e-16           6.151104e-17           1.620945e-16
##         concavity_mean          symmetry_mean fractal_dimension_mean
##           3.883768e-17           1.686096e-16           4.814025e-16
##             radius_se             texture_se          smoothness_se
##           3.704199e-17          -1.065620e-16           1.291348e-16
##           concavity_se            symmetry_se
##          -5.963912e-17           8.751893e-17
```

```r
n = dim(x)[1]
n_train = floor(n*0.8)

y_train = as.matrix(y[1:n_train,], ncol = 1)
y_test = as.matrix(y[(n_train + 1):n,], ncol = 1)
x_train = x[1:n_train,]
x_test = x[(n_train + 1):n,]

x_train_b0 = cbind(rep(1,nrow(x_train)),x_train)
x_test_b0 = cbind(rep(1,nrow(x_test)), x_test)
```

```r
logisticstuff <- function(x, y, betavec) {

  x = cbind(1, x)
  colnames(x)[1] = "intercept"

  u <- x %*% betavec
  expu <- exp(u)
  # loglikelihood
  loglik <- t(u) %*% y - sum(log(1 + expu))
  p <- expu / (1 + expu)
  # gradient
  grad <- t(x) %*% (y - p)
  # hessian
  Hess <- -t(x) %*% diag(as.vector(p * (1-p))) %*% x

  return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

```r
# newton raphson
NewtonRaphson <- function(x, y, func, start, tol=1e-10, maxiter = 1000) {
  i = 0
  cur = start
  stuff = func(x, y, 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

    # redirection
    hess = stuff$Hess
    hess.eigen = eigen(stuff$Hess)$values

    if (sum(hess.eigen) > 0){
      g = max(hess.eigen) + 1
      hess = hess - g*diag(1, dim(hess))
    }

    cur = prev - solve(hess) %*% stuff$grad

    # step halving
    lambda = 1
    while (func(x, y, cur)$loglik < prevloglik) {
      lambda = lambda / 2
      cur = prev - lambda * solve(hess) %*% stuff$grad
    }

    stuff = func(x, y, cur) # log-lik, gradient, Hessian
    res = rbind(res, c(i, stuff$loglik, cur))
    # Add current values to results matrix
  }
  colnames(res) = c("i", "loglik", paste0("beta", 0:(ncol(res)-3)))
  return(res)
}

nr = NewtonRaphson(x_train, y_train, logisticstuff, rep(1,12))

ans = data.frame(nr)
colnames(ans) = c("iteration", "log likelihood")
ggplot(ans, aes(x = iteration, y = `log likelihood`)) +
  geom_point(size = 0.5) +
  geom_line()
```



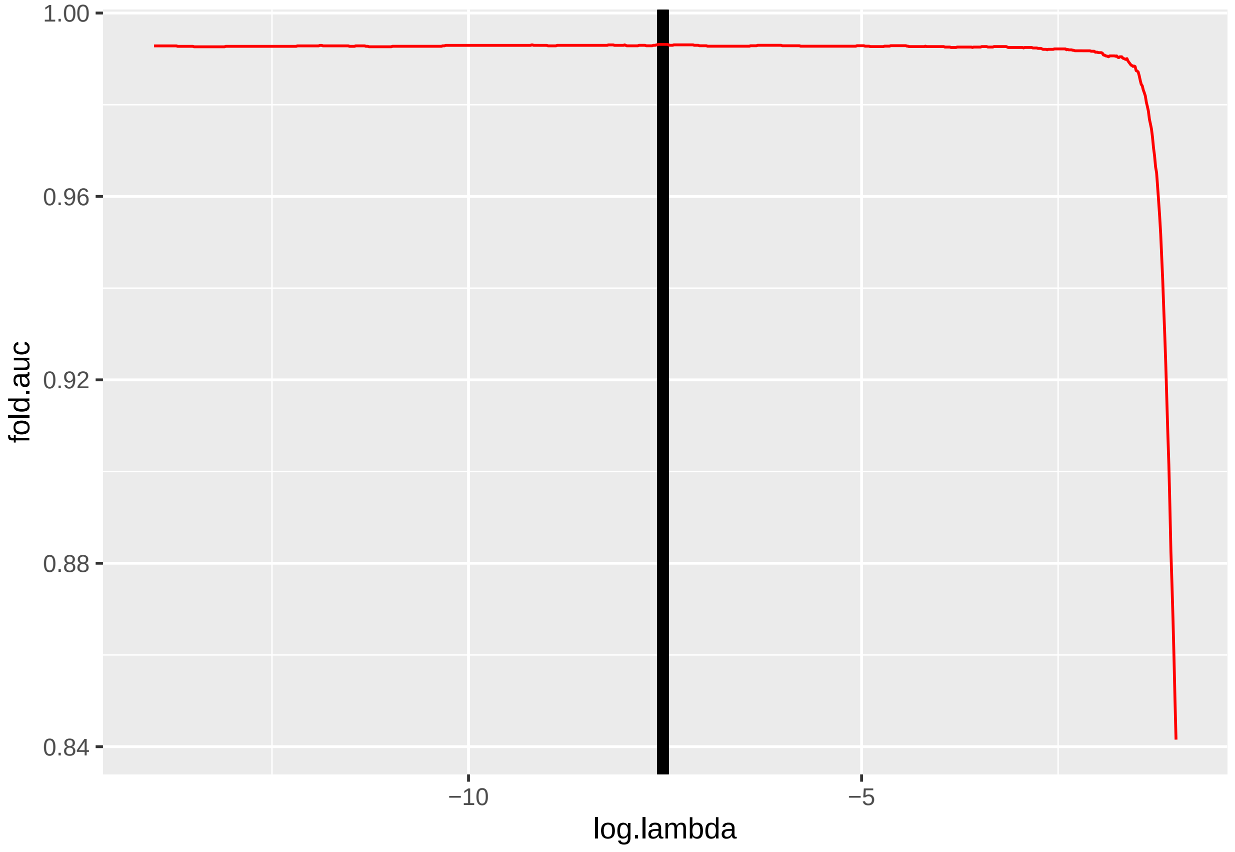
```
# coordinate lasso
soft_threshold <- function(beta, lambda) {
    sign(beta) * max(abs(beta) - lambda, 0)
}

getP <- function(X, betavec){
    Px <- 1/(1 + exp(-(X %*% betavec)))
    return(Px)
}

getW <- function(Px){
    W <- Px*(1-Px)
    return(Px)
}

getZ <- function(X, y, betavec, Px, W){
    Z <- X %*% betavec + (y - Px)/W
    return(Z)
}


lassoCD <- function(
        X, y, lambda, init_beta, max_iter = 500, tol = 1e-8
){
    betavec <- init_beta
    N <- length(y)
    i <- 0
    loss <- 1e5
    prevloss <- Inf
    res <- c(0, loss, betavec)
    cont <- TRUE
    while(i <= max_iter & cont){
        i <- i + 1
        prevloss <- loss
        for(j in 1:length(betavec)){
            Px <- getP(X, betavec)
            W <- getW(Px)
            W <- ifelse(abs(W-0) < 1e-5, 1e-5, W)
            Z <- getZ(X, y, betavec, Px, W)
            Zresj <- X[,-j] %*% betavec[-j]
            betaj <-
                soft_threshold(sum(W * X[,j] * (Z - Zresj)), lambda)/sum(W * X[,j] * X[,j])
            betavec[j] <- betaj
            loss <- (1/(2*N))*sum(W * (Z - X %*% betavec)^2) + lambda * sum(abs(betavec))
        }
        #print(loss)
        if(abs(prevloss - loss) < tol || loss < Inf){
            cont <- FALSE
        }
        res <- rbind(res, c(i, loss, betavec))
    }
    return(res)
}
```

```
# auc cv
model.glm <- glm.fit(x_train, y_train, family = binomial(link = "logit"))

lambda.cv.lassoCD = function(lambdas, x, y, k) {
  set.seed(2022)
  data = as.data.frame(cbind(x, y))
```

```r
  folds = crossv_kfold(data, k = k)

  start = rep(5, ncol(x))
  fold.auc <- vector()
  #fold.se <- vector()

  for (j in 1:length(lambdas)) {
    fold.errors <- vector()
    for (i in 1:k) {
      trainrow= folds[i,1][[1]][[toString(i)]]$idx
      testrow = folds[i,2][[1]][[toString(i)]]$idx

      train.X = x[trainrow,]
      test.X = x[testrow,]

      train.y = y[trainrow,]
      test.y = y[testrow,]

      # Perform the logistic-LASSO
      fit = lassoCD(train.X, train.y, lambda = lambdas[j], init_beta = start)
      betas = fit[nrow(fit),3:ncol(fit)]
      u = test.X %*% betas
      expu = exp(u)
      prob = expu / (1 + expu)
      # Calculate the test MSE for the fold
      fold.errors[i] = mean(auc(test.y, prob))
    }
    start = betas
    fold.auc[j] = mean(fold.errors)
    #fold.se[j] = sqrt(var(fold.errors)/k)
  }
  return(cbind(log.lambda = log(lambdas), fold.auc))
}

lambda.seq = exp(seq(-1,-14, length=1000))

cv.path.lassoCD = lambda.cv.lassoCD(lambda.seq, x_train_b0, y_train, 5)
cv.path.lassoCD = as.data.frame(cv.path.lassoCD)
max.auc.lassoCD = max(cv.path.lassoCD$fold.auc)

max.auc.lassoCD
```

```
## [1] 0.9931619
```

```r
best.lambda.lassoCD = cv.path.lassoCD[which(cv.path.lassoCD$fold.auc == max.auc.lassoCD),]$log.lambda

cv.path.lassoCD %>%
  ggplot(data = ., aes(x = log.lambda, y = fold.auc)) +
  geom_vline(xintercept = best.lambda.lassoCD) +
  geom_line(color = "red")
```



```
mean(exp((best.lambda.lassoCD)))
```

```
## [1] 0.000539252
```

```
best.lambda.lassoCD <- mean(exp((best.lambda.lassoCD)))
```

```
best_beta.lassoCD <- lassoCD(x_train_b0, y_train,mean(exp(best.lambda.lassoCD)), init_beta = rep(4,dim(
```

```
# caret package
set.seed(9543)

ctrl <- trainControl(method = "cv",
                     number = 5,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)

glmnGrid <- expand.grid(alpha = 1, lambda = lambda.seq)

model.glmn <- train(x = x_train,
                   y = as.factor(ifelse(y_train == 1, "pos", "neg")),
                   method = "glmnet",
                   tuneGrid = glmnGrid,
                   metric = "ROC",
                   trControl = ctrl)

model.glmn$bestTune
```

```
##      alpha      lambda
## 656     1 0.004184046
```

```
# coeficients
# newton rapson
NR.coef = ans[nrow(ans), 3:ncol(ans)] %>% t()

# logistic lasso
NR.lasso.ans = lassoCD(x_train_b0, y_train, lambda = exp(best.lambda.lassoCD), init_beta = rep(0, ncol(
NR.lasso.coef = NR.lasso.ans[nrow(NR.lasso.ans), 3:ncol(NR.lasso.ans)]

# glm
fit.glm = glm(y_train ~ x_train, family = binomial(link = "logit"))
```

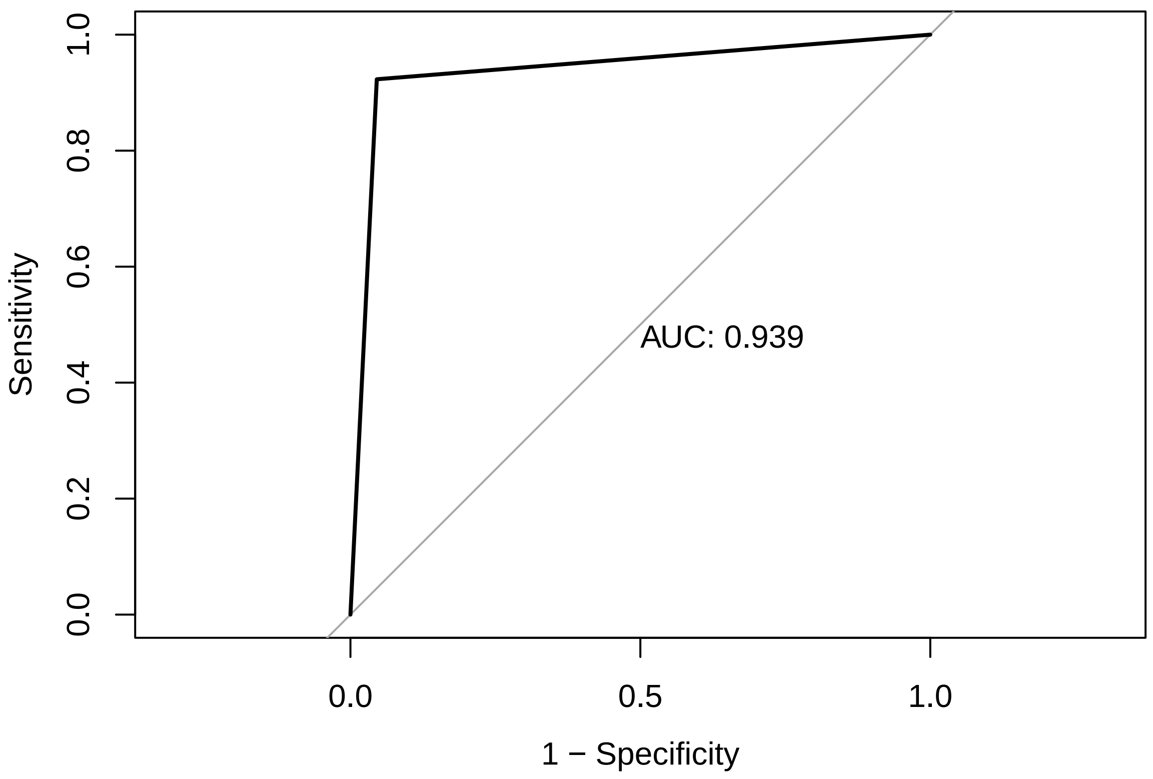
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.coef = fit.glm$coefficients %>% as.data.frame()

glmn.coef = coef(model.glmn$finalModel, model.glmn$bestTune$lambda) %>% as.matrix() %>% as.data.frame()

logistics.coef = cbind(glmn.coef, glm.coef, NR.coef, NR.lasso.coef)
colnames(logistics.coef) = c("caret-glmnet", "glm package", "Newton-Raphson", "Logistic-LASSO")

logistics.coef %>%
  select("glm package", "Newton-Raphson", "Logistic-LASSO", "caret-glmnet") %>%
  knitr::kable()
```

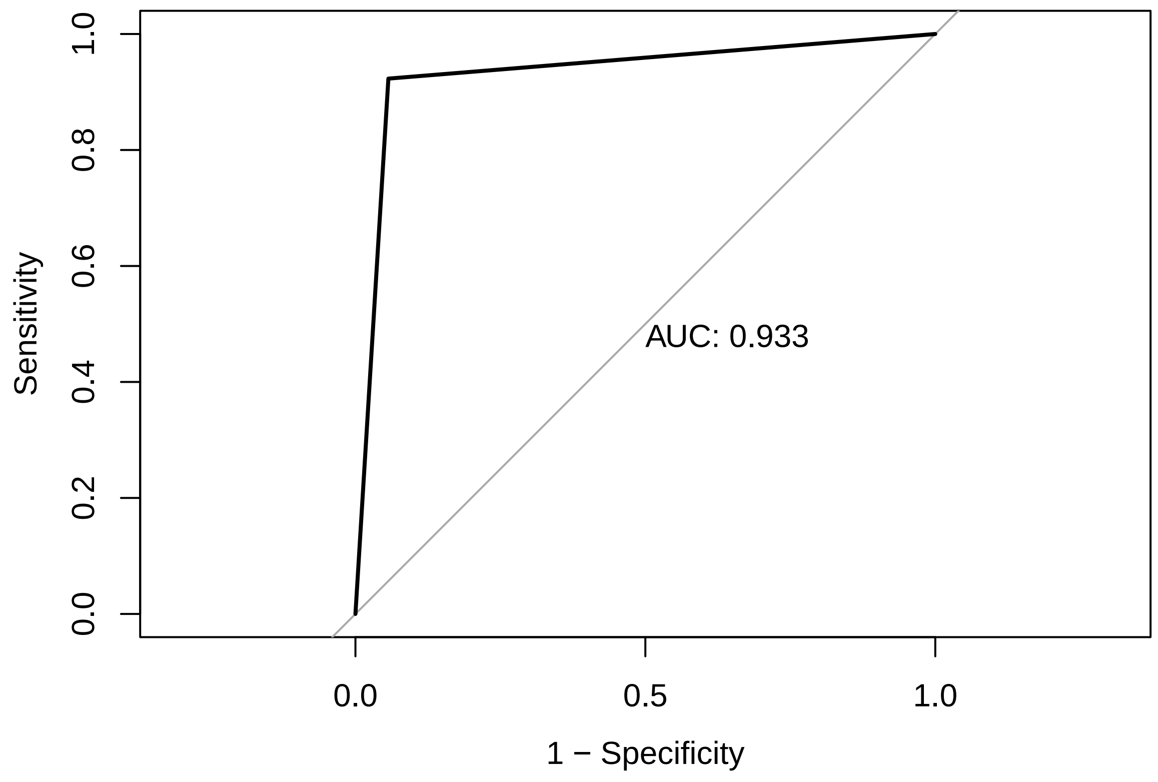
	glm package	Newton-Raphson	Logistic-LASSO	caret-glmnet
(Intercept)	0.0654858	0.0654858	-0.1780196	-0.1635653
radius_mean	2.2932120	2.2932120	0.7817191	2.3018939
texture_mean	2.3360195	2.3360195	0.3828877	1.6203391
smoothness_mean	1.0236278	1.0236278	0.3554685	0.8470807
concavity_mean	5.6067035	5.6067035	0.2635717	2.8889899
symmetry_mean	0.4361524	0.4361524	0.1120210	0.1402152
fractal_dimension_mean	-0.6928074	-0.6928074	-0.0279319	-0.0567243
radius_se	1.7676645	1.7676645	0.1404886	0.9250945
texture_se	-0.8230016	-0.8230016	-0.0510010	-0.3910476
smoothness_se	-0.0306181	-0.0306181	-0.0427242	-0.0772251
concavity_se	-2.7175298	-2.7175298	0.0343491	-1.2376771
symmetry_se	-0.4841348	-0.4841348	-0.0419582	-0.2633245

```
# prediction
lassoCD.predict <- function(betavec, X_new, y){
    Py <- 1/(1 + exp(-(X_new %*% betavec)))
    y.pred = rep(0, nrow(y))
    y.pred[Py > 0.5] = 1
    return(y.pred)
}

y.pred.log.lasso = lassoCD.predict(NR.lasso.coef, x_test_b0, y_test)
y.pred.log = lassoCD.predict(NR.coef, x_test_b0, y_test)
y.pred.glm = lassoCD.predict(as.matrix(glm.coef), x_test_b0, y_test)
y.pred.glmn = lassoCD.predict(as.matrix(NR.lasso.coef), x_test_b0, y_test)
```

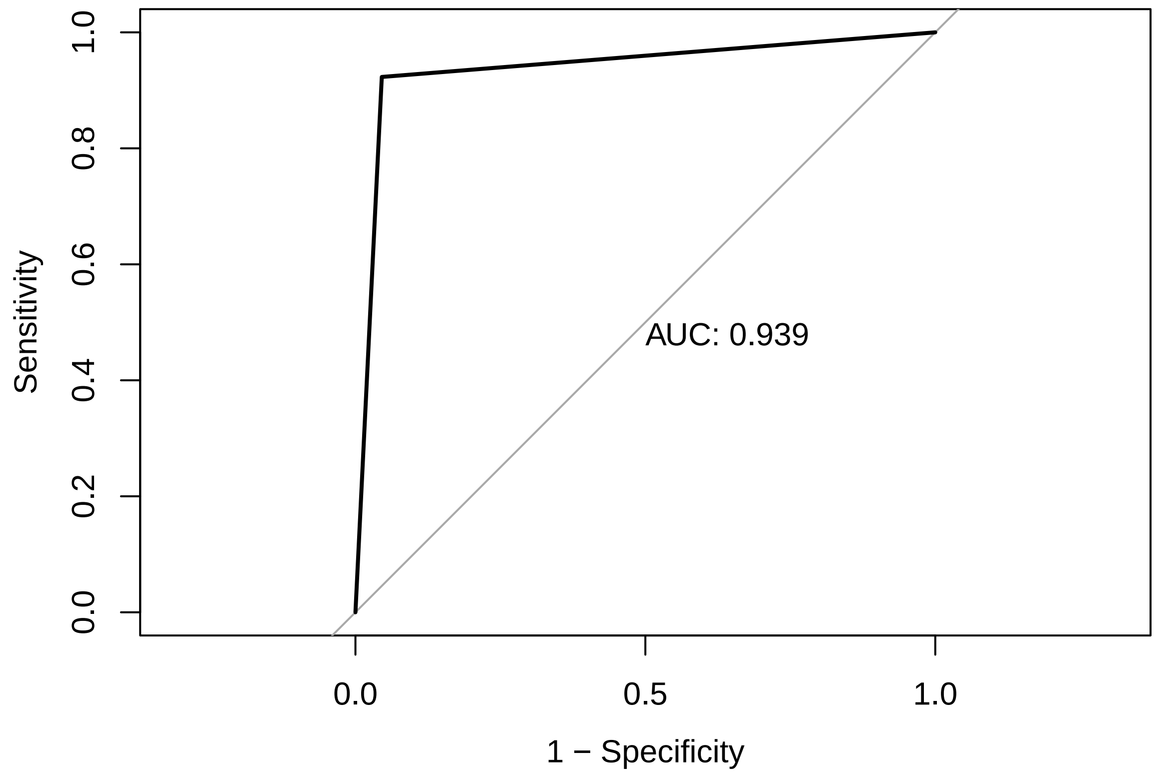
```r
roc.log.lasso <- roc(y_test, y.pred.log.lasso)
roc.log = roc(y_test, y.pred.log)
roc.glmn = roc(y_test, y.pred.glmn)
plot(roc.log.lasso, legacy.axes = TRUE, print.auc = TRUE)
```



```r
plot(roc.log, legacy.axes = TRUE, print.auc = TRUE)
```

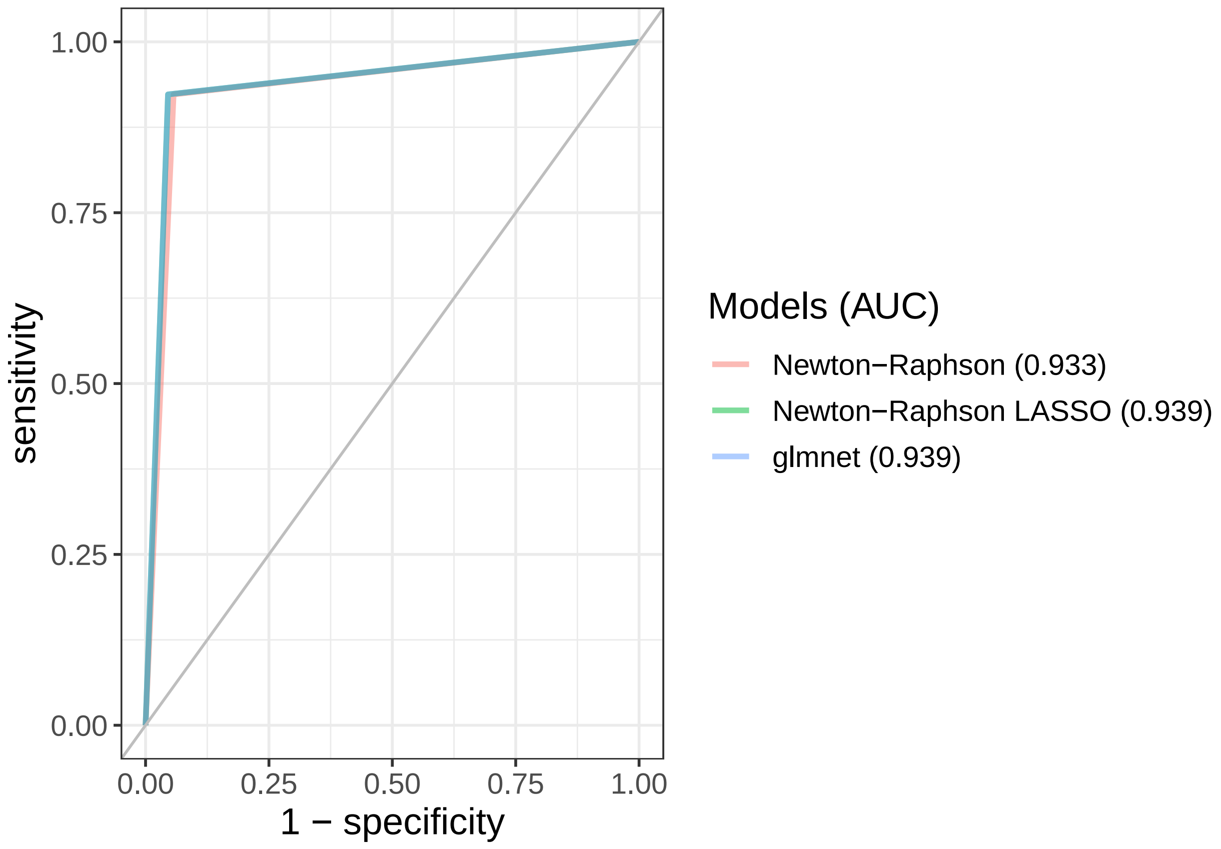


```
plot(roc.glmn, legacy.axes = TRUE, print.auc = TRUE)
```



```
auc <- c(roc.log$auc[1], roc.log.lasso$auc[1], roc.glmn$auc[1])

modelNames <- c("Newton-Raphson","Newton-Raphson LASSO","glmnet")
```

```r
ggroc(list(roc.log, roc.log.lasso, roc.glmn), legacy.axes = TRUE, size = 1, alpha = 0.5) +
  scale_color_discrete(labels = paste0(modelNames, " (", round(auc,3),")"),
                       name = "Models (AUC)") +
  geom_abline(intercept = 0, slope = 1, color = "grey") +
  theme_bw() +
  theme(text = element_text(size = 14))
```

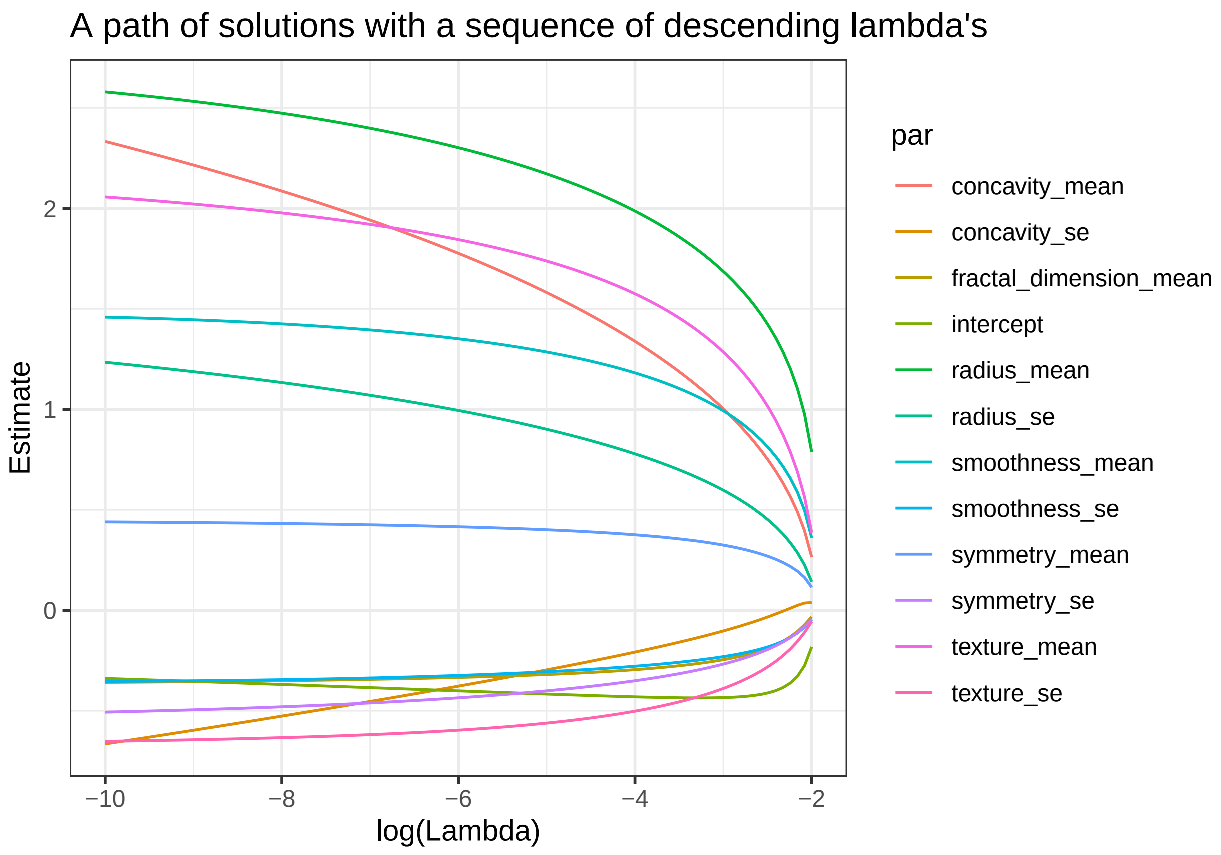


CV performance

```r
# cv.roc.log.lasso = lambda.cv.lassoCD(exp(best.lambda.lassoCD), x_train_b0, y_train, k = 5)
# cv.roc.log = lambda.cv.lassoCD(0, x_train_b0, y_train, k = 5)
# cv.roc.glmn = model.glmn$resample$ROC
```

```r
#df = tibble(
#  AUC = c(cv.roc.log, cv.roc.log.lasso, cv.roc.glmn),
#  model = c(rep("Newton Raphson", 5), rep("Logistic-LASSO", 5), rep("caret-glmnet",5))
#)

#df %>%
#  ggplot(aes(fill = model)) +
#  geom_boxplot(aes(x = AUC, y = model)) +
#  scale_fill_manual(values=wes_palettes$GrandBudapest1[c(1, 2, 4)]) +
#  theme_bw() +
#  theme(text = element_text(size = 14))
```

```r
path <- function(X, y, lambdaseq){
    init_beta <- rep(0, dim(X)[2])
    betas <- NULL
    for (i in 1:length(lambdaseq)) {
        cd.result = lassoCD(X, y,lambda = lambdaseq[i], init_beta)
        last_beta <- cd.result[nrow(cd.result),3:dim(cd.result)[2]]
        init_beta <- last_beta
        betas <- rbind(betas,c(last_beta))
        i <- i + 1
    }
    return(data.frame(cbind(lambdaseq,betas)))
}
path.out <- path(x_train_b0, y_train, lambdaseq = exp(seq(-2,-10, length=100)))
colnames(path.out) <- c("lambda","intercept",colnames(x_train_b0)[2:12])
# plot a path of solutions
path.plot <- path.out %>%
  gather(key = par, value = estimate, c(2:dim(path.out)[2])) %>%
  ggplot(aes(x = log(lambda), y = estimate, group = par, col = par)) +
  geom_line()+
  ggtitle("A path of solutions with a sequence of descending lambda's") +
  xlab("log(Lambda)") +
  ylab("Estimate") +
  theme(legend.position = "bottom",
        legend.text = element_text(size = 6))
path.plot + theme_bw()
```


A path of solutions with a sequence of descending lambda's
Estimate
log(Lambda)
par
concavity_mean
concavity_se
fractal_dimension_mean
intercept
radius_mean
radius_se
smoothness_mean
smoothness_se
symmetry_mean
symmetry_se
texture_mean
texture_se