Predictive Model Based on Logistic Regression for Cancer Diagnosis

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

3/27/2022

Contents

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.

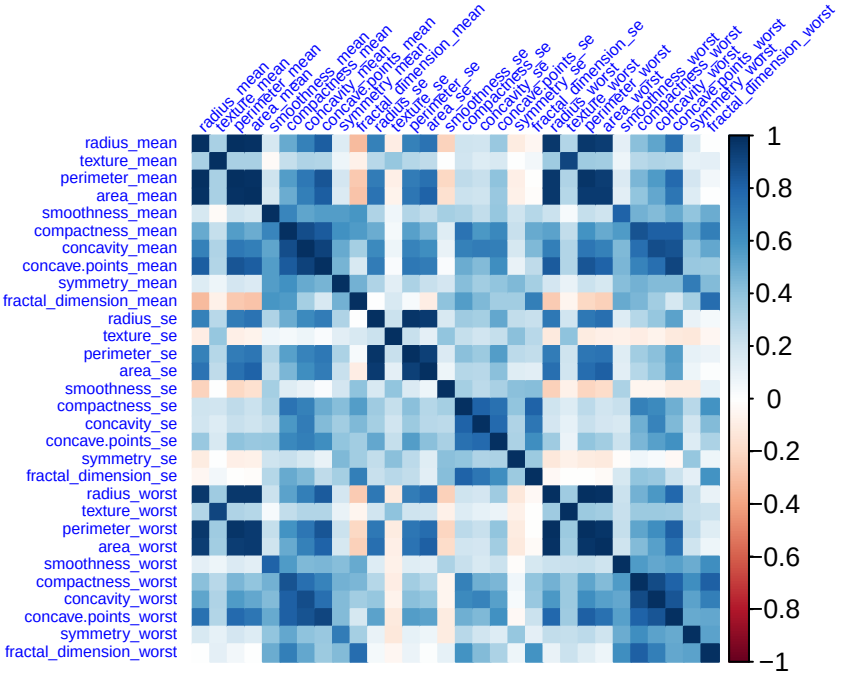


Figure 1. Correlation plot of all variables

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:

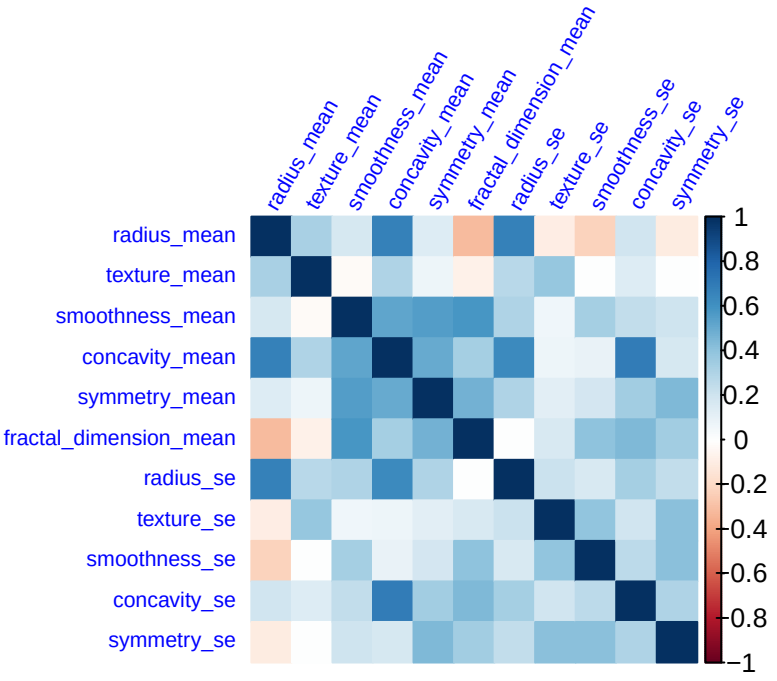


Figure 2. Correlation plot of after feature selection

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.

Methods

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:

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

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

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

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

Then maximize the log likelihood:

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

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

$$\nabla l(\beta) = \sum_{i=1}^{n} (y_i - \pi_i)\, x_i = X^T (Y - \pi)$$

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.

Newton-Raphson Algorithm

Our target is to find the β that maximizes the log-likelihood $l(\beta)$ of our logistic regression, that is, the solution to the equation $\nabla l(\beta) = 0$. At the same time, the Hessian matrix $\nabla^2 l(\beta)$ needs to be negative definite to ensure β to be a local maximum rather than a local minimum or a saddle point. As for logistic regression,

$$H = -X^T \operatorname{diag}\left(\pi_i \left(1 - \pi_i\right)\right) X = -\sum_{i=1}^{p} a_i^2 x_{ii}^2 \pi_i (1 - \pi_i),$$

for any a,

$$\mathbf{a}^T H \mathbf{a} = -\sum_{i=1}^{p} a_i^2 x_{ii}^2 \pi_i (1 - \pi_i) < 0,$$

thus the Hessian matrix is negative definite and the log-likelihood is concave at any parameter value. In this case, $\nabla l(\beta) = 0$ guarantees the global maximum.

The basic idea is that given a current point θ_{i-1}, we choose an ascent direction d and an appropriate step length λ to travel and locate the next θ_i. Specially, the Newton-Raphson algorithm makes use of the information from the second partial derivatives to find the direction with the highest convergence efficiency. The ith step is given by

$$\theta_i = \theta_{i-1} - \lambda [\nabla^2 f(\theta_{i-1})]^{-1} \nabla f(\theta_{i-1})$$

Normally, we need to check if the Hessian matrix is negative definite. If not, we should replace it with a negative definite matrix such as $\nabla^2 f(\theta_{i-1}) - \gamma I$ where γ is the largest eigenvalue, or with negative Fisher information matrix, or simply with $-I_{p\times p}$ (as in the Gradient Descent algorithm). Again in this study, the Hessian matrix is always negative definite so this step is not necessary. Next, we add step halving modification in each iteration. After each time we update the parameter point, we check if the log-likelihood has increased. If not, we backtrack and cut the current λ in half and recalculate the next parameter point, until the log-likelihood is increasing.

The stopping criteria is when the absolute value of difference in consecutive log-likelihood values is less than the tolerance of 10^{-10} or iteration times reaches 1000. Since a unique global maximum is guaranteed, variety of initial θ_0 should result in the same final estimation. After testing different inputs, we get the same convergence indeed. The estimations using the training dataset with the selected predictors are shown in the results section.

Logitstic LASSO Coordinate Descent Algorithm

Regularization is the common variable selection approaches for high-dimensional covariates. One of the best known Regularization is called LASSO, which is to add $L1$-penalty to the objective function. In this project, we consider a logistic regression, with penalty term, we are aiming to maximize the penalized log likelihood:

$$\max_{\beta \in \mathbb{R}^{p+1}} \frac{1}{n} \sum_{i=1}^{n} \{y_i (X_i \beta) - \log(1 + \exp(X_i \beta))\} - \lambda \sum_{j=0}^{p} |\beta_j|$$

for some $\lambda \geq 0$. Here the $x_{i,j}$, which are our data, are standardized so that $\sum_i x_{i,j}/n = 0$ and $\sum_i x_{i,j}^2/n = 1$.

If the current estimate of the parameter is $\tilde{\beta}$, we can form a quadratic approximation to the negative log likelihood by Taylor expansion around the current estimate, which is:

$$f(\beta) = -\frac{1}{2n}\sum_{i=1}^{n} w_i(z_i - \sum_{j=0}^{p} x_{i,j}\beta_j)^2 + C(\tilde{\beta})$$

where

$$z_i = \tilde{\beta}_0 + x_i^T\tilde{\beta} + \frac{y_i - \tilde{p}(x_i)}{\tilde{p}(x_i)(1-\tilde{p}(x_i))}, \text{working response}$$
$$w_i = \tilde{p}(x_i)(1-\tilde{p}(x_i)), \text{working weights}$$

and $\tilde{p}(x_i)$ is evaluated at the current parameters, the last term is constant.

The coordinate descent algorithm for logistic lasso regressions finds the iteratively re-weighted least squares (IRLS) solution for the penalized Taylor approximation of the log-likelihood of the logistic regression model, which is

$$\min_{\beta\in\mathbb{R}^{p+1}} \{-f(\beta) + \lambda\sum_{j=0}^{p}|\beta_j|\}$$

The coordinate descent procedure for the logistic regression follows the same steps as the least squares procedure, with the exception that the coordinate update formula is modified to account for the weights, that is, for any $\lambda \in \mathbb{R}_+$:

$$\tilde{\beta}_j^{\text{lasso}} \leftarrow \frac{S(\sum_{i=1}^n w_i x_{ij}(z_i - \sum_{j\neq k}\beta_j x_{ij}), \lambda)}{\sum_{i=1}^n w_i x_{ij}^2}$$

where S is the soft-threshold (or *shrinkage*) function, defined as $S(\hat{\beta}, \lambda) = sign(\hat{\beta})(|\hat{\beta}| - \lambda)_+$.

The above amounts to a sequence of nested loops:

- outer loop: start with λ that all the coefficients are forced to be zero (in application, $\lambda_{\max} = \frac{\max|X^T y|}{n}$), then decrements λ;
- middle loop: update the quadratic $f(\beta)$ using the current estimates of parameters;
- inner loop: run the coordinate descent algorithm on the penalized weighted least square problem.

In our problem, care is taken to avoid coefficients diverging in order to achieve fitted probabilities of 0 or 1 which is the warning message by the R package. When a working weight get closely enough to 0 (10^{-5} in application) , we set it to a threshold value (10^{-5}).

The implementation of algorithm can be shown in the **Algorithm 1**.

5-Fold Cross Validation

In the k-fold cross validation,the original data will be divided 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.

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.

Algorithm 1 Coordinate Descent Logistic Lasso

for $\lambda \in \lambda_{max} \geq \lambda_1 \geq ... \lambda_{min} \geq 0$ **do**
  for $iteration = 1, 2, ..., k$ **do**
    Run the coordinate descent algorithm until it converges
    for $j = 1, 2, ..., p + 1$ **do**
      Update $\tilde{\beta}_j^{\text{lasso}} \leftarrow \frac{S(\sum_{i=1}^{n} w_i x_{ij}(z_i - \sum_{j \neq k} \beta_j x_{ij}), \lambda)}{\sum_{i=1}^{n} w_i x_{ij}^2}$
    end for
    Check the difference between previous loss and current loss. If the difference is smaller than predefined tolerance(10^{-10}), then stop updating.
  end for
  Use the final coefficient estimates for each value of λ_t as the initial value for the subsequent λ_{t+1}
end for

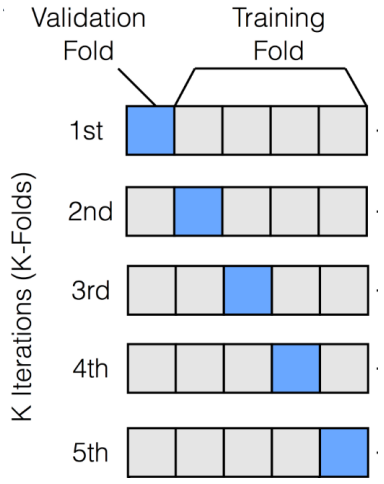


Figure 1: Principle of k-fold CV

We use AUC as the criteria to choose the best tuning 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 average AUC is the biggest.

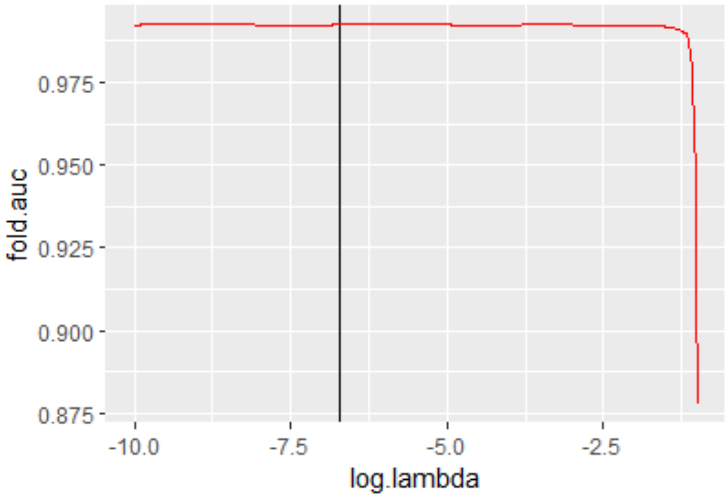


Figure 2: log.lambda with the Largest AUC

After the 5-fold cross validation, we found that the best lambda is about $e^{-6.236} = 0.00196$ in the optimal model when set.seed(2022) within e^{-1} to e^{-10}.

Results and Discussion

Table 1 showed model coefficient estimated by `glm` package, Newton-Raphson Method, `caret` package and Newton-Raphson Method with LASSO. Coefficients given by methods with LASSO were generated under the best lambda identified by 5-fold cross-validation. The Newton-Raphson method has given estimation identical to the glm package's result, while Newton-Raphson Method with LASSO method generated different results compared to coefficients given by the caret package.

	glm package	Newton-Raphson	Newton-Raphson-LASSO	caret-glmnet
(Intercept)	0.0654858	0.0654858	-0.0426412	-0.1640622
radius_mean	2.2932120	2.2932120	2.8124161	2.3027405
texture_mean	2.3360195	2.3360195	2.1686704	1.6190796
smoothness_mean	1.0236278	1.0236278	1.0493920	0.8468917
concavity_mean	5.6067035	5.6067035	3.6810934	2.8832435
symmetry_mean	0.4361524	0.4361524	0.2940973	0.1396835
fractal_dimension_mean	-0.6928074	-0.6928074	-0.0473727	-0.0553622
radius_se	1.7676645	1.7676645	1.2966335	0.9232047
texture_se	-0.8230016	-0.8230016	-0.6025411	-0.3902772
smoothness_se	-0.0306181	-0.0306181	-0.3360082	-0.0771703
concavity_se	-2.7175298	-2.7175298	-1.3849591	-1.2343725
symmetry_se	-0.4841348	-0.4841348	-0.4795127	-0.2628887

Table 1. Model Coefficient Estimated by Different Methods

Figure 4 and Figure 5 showed comparison among performances of models in terms of 5-fold cross-validated AUC on the training dataset and AUC on the test dataset. For calculating cross-validated AUC, previous folds used in the train process of the logistic LASSO model were used. Compared to the full logistic model, the optimal logistic LASSO model displayed better performance in the cross validation on the train dataset,

and showed the same AUC on the test dataset. Therefore, we can conclude that inclusion of LASSO penalty term in the model training process has slightly improved the model. Besides, the model trained by `caret`, although showed the best performance in the cross-validation, has a test AUC slightly smaller than other 2 models.

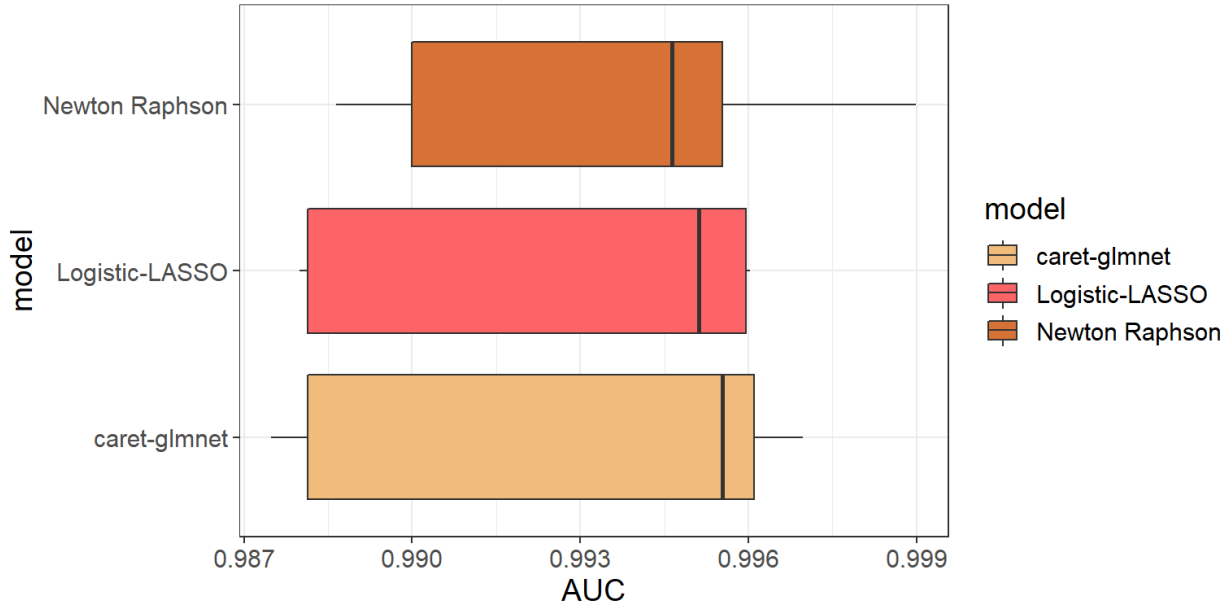


Figure 4. Model Coefficient Estimated by Different Methods

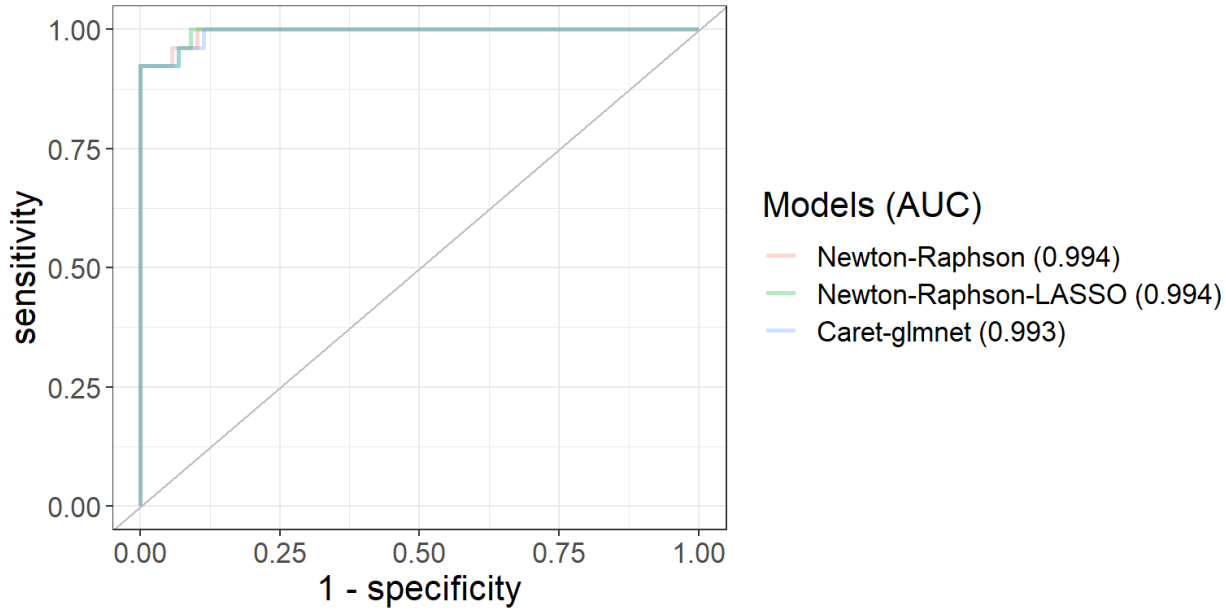


Figure 5. Model AUCs on the Test Dataset

This study has some limitations. At the data preprocessing stage, we removed variables with correlation coefficient larger than 0.7, which only left us 11 variables for modeling. For further study, we could try other criteria for feature selection and keep more information from the original dataset. In addition, we could try some other evaluation metrics in cross validation, like error rate.

Citations

1. Friedman, J., Hastie, T., Höfling, H. and Tibshirani, R. (2007). "Pathwise Coordinate Optimization." The Annals of Applied Statistics. Vol. 1, No. 2, pp. 302-332.
2. Friedman, J., Hastie, T., and Tibshirani, R. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." Journal of Statistical Software, Volume 33 Issue 1.
3. Fu, W. 1998. "Penalized regressions: the bridge vs. the lasso." Journal of Computational and Graphical Statistics, Volume 7, Issue 3, pp. 397-416.
4. Hastie, T., Tibshirani, R., and Friedman, J. 2009. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. New York City, NY: Springer Series in Statistics.
5. Tibshirani, R. 1996. "Regression Shrinkage and Selection via the Lasso." Journal of the Royal Statistical Society. Series B (Methodological), Volume 58 Issue 1, pp. 267-288.

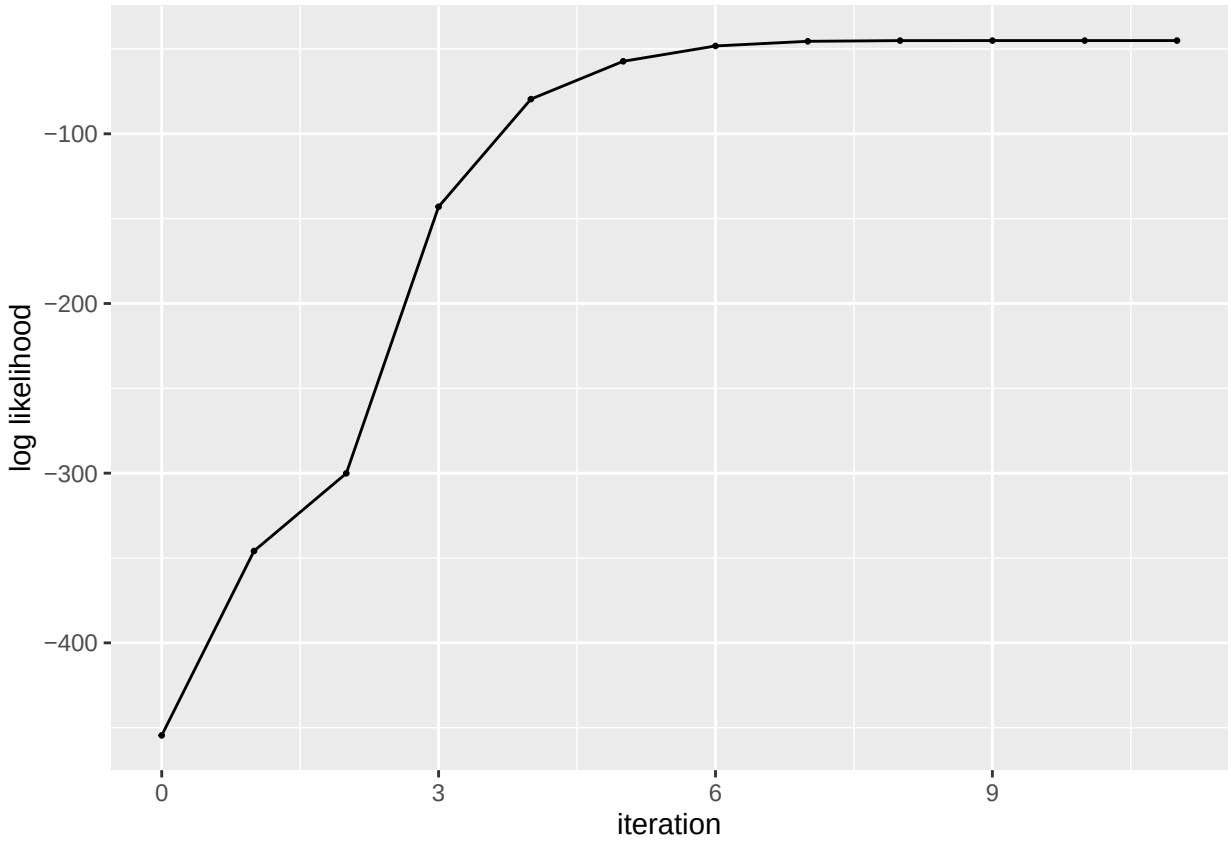
Appendix

```
# 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)
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)
```

```
# newton raphson
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))
}

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()
```

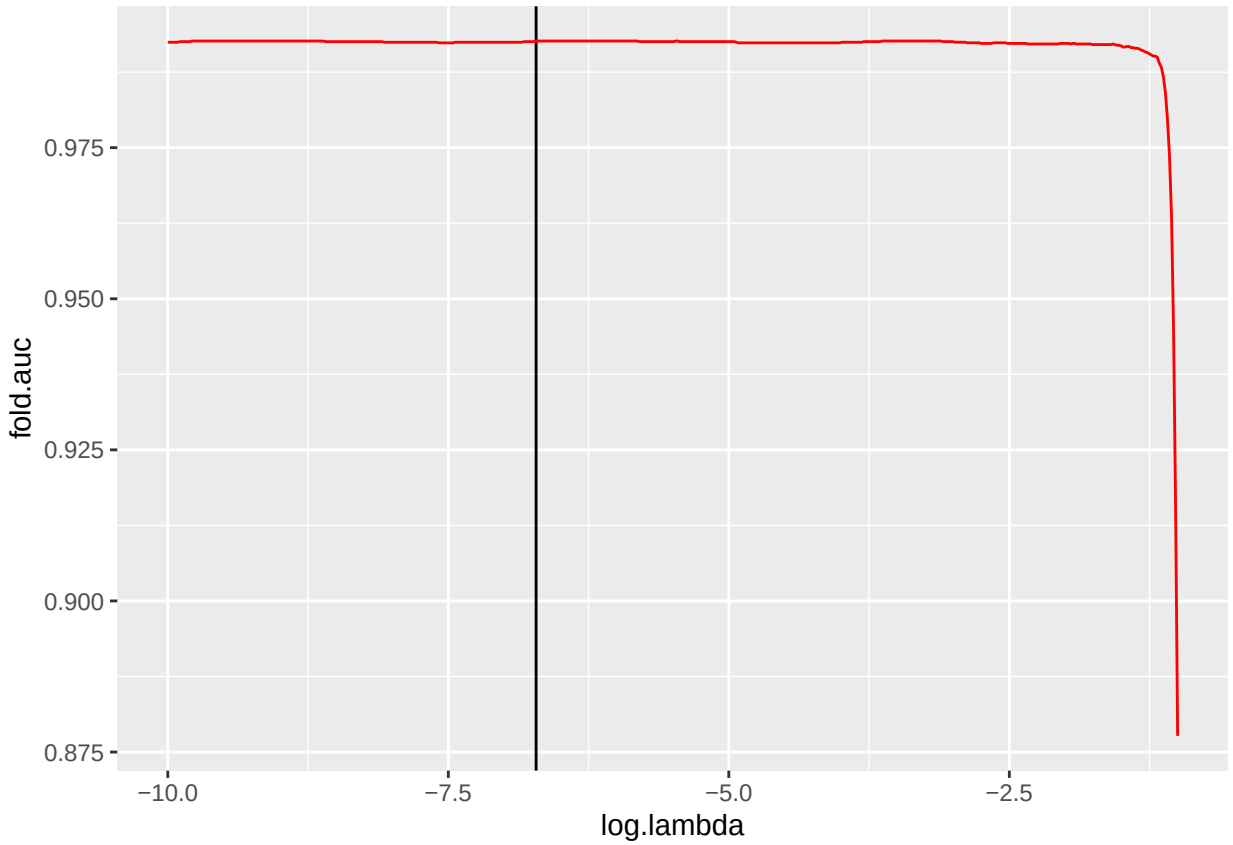



```
# coordinate lasso
soft_threshold <- function(beta, lambda) {
    sign(beta) * max(abs(beta) - lambda, 0)
}

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))
        }

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

  start = rep(1, ncol(x))
  fold.auc <- 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)

      fold.errors[i] = mean(auc(test.y, prob))
    }
    start = betas
    fold.auc[j] = mean(fold.errors)
  }
  return(cbind(log.lambda = log(lambdas), fold.auc))
}

lambda.seq = exp(seq(-1,-10, length=500))

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

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

best.lambda.lassoCD

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

best.lambda = best.lambdas[length(best.lambdas)]


ind = which(cv.path.lassoCD$log.lambda == best.lambda)

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




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

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

```r
# 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
```

```r
extract.coef = function(lambdas, x, y, k) {
  set.seed(2022)
  data = as.data.frame(cbind(x, y))
  folds = crossv_kfold(data, k = k)
```

```
  start = rep(1, ncol(x))

  for (j in 1:length(lambdas)) {
    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)]
    }
    start = betas
  }
  return(betas)
}

NR.lasso.coef = extract.coef(lambda.seq[1:ind], x_train_b0, y_train, 5)
```

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

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

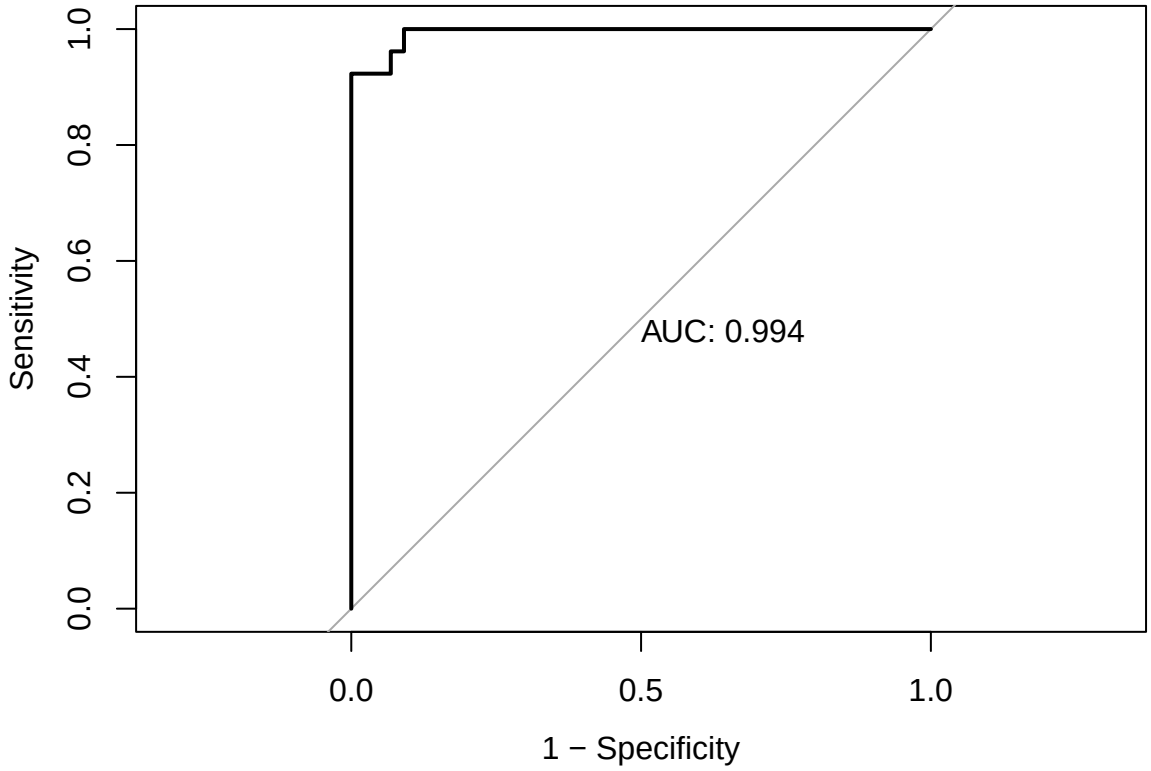
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
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()
```

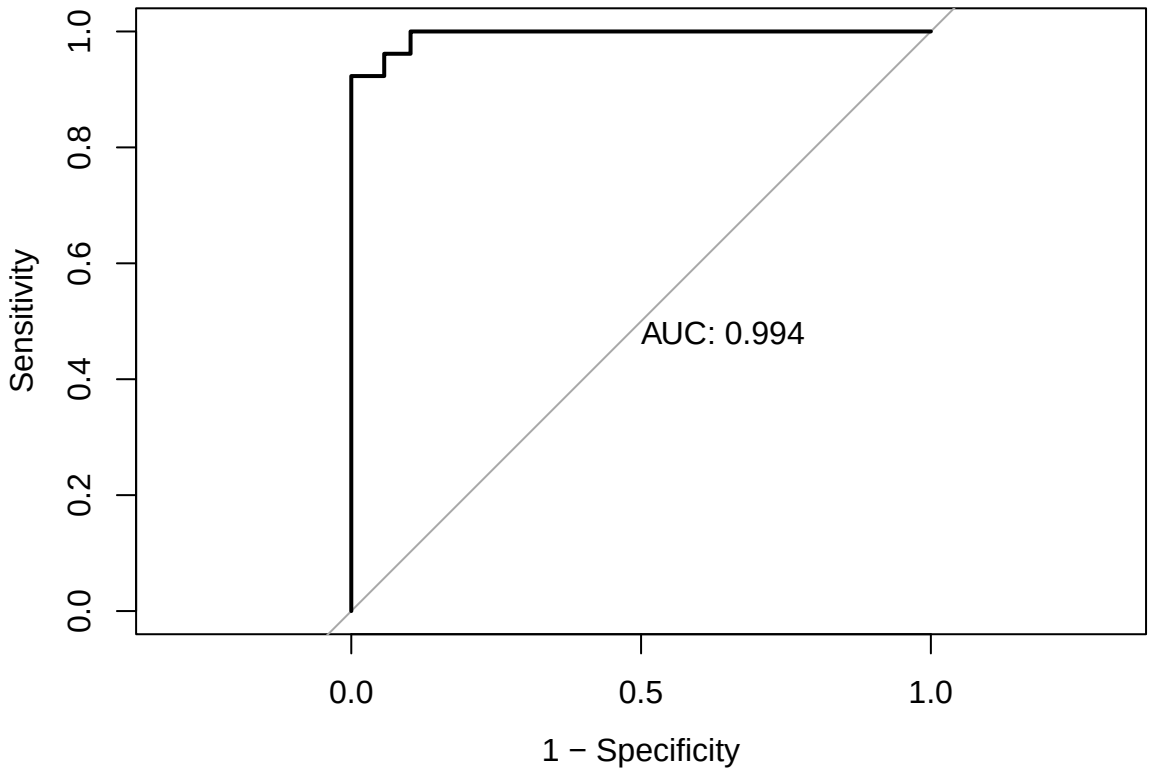
```
# prediction
lassoCD.predict <- function(betavec, X_new, y){
    Py <- 1/(1 + exp(-(X_new %*% betavec)))
    #y.pred = rep(0, nrow(y))
    #y.pred[Py > 0.5] = 1
    return(Py)
}

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(glmn.coef), x_test_b0, y_test)
```

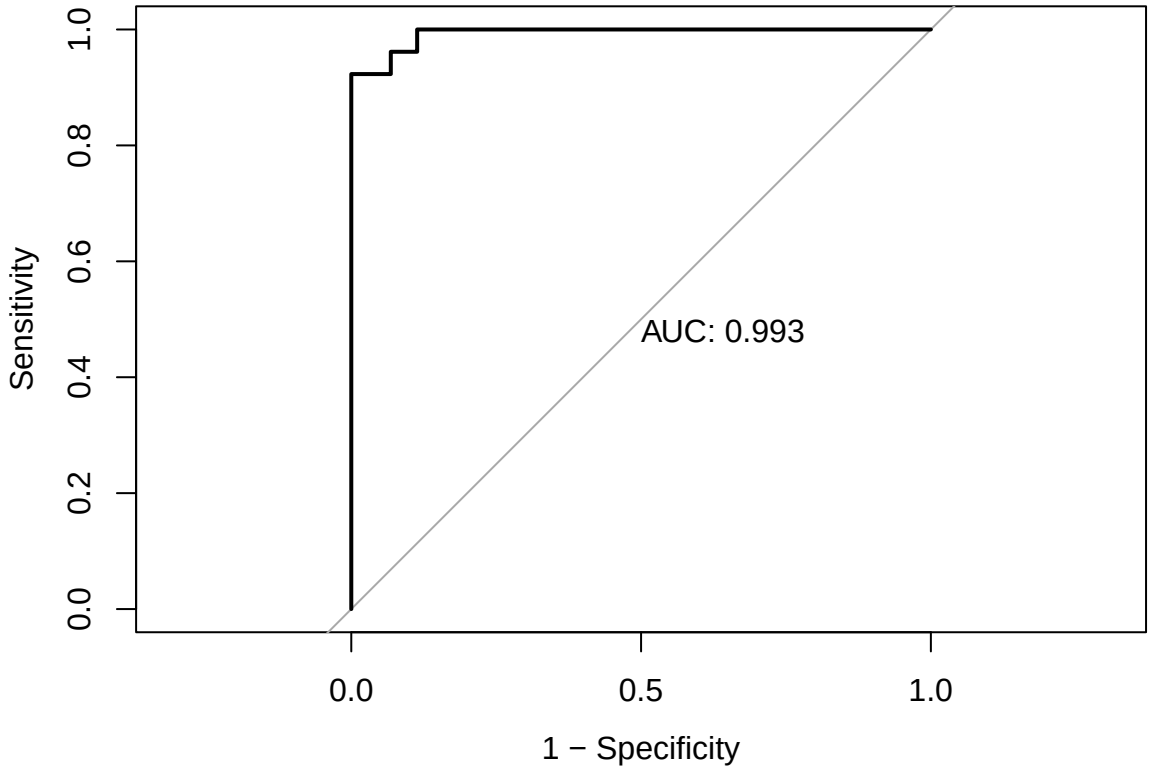
```
roc.log.lasso <- roc(y_test, y.pred.log.lasso)
roc.log = roc(y_test, y.pred.log)
roc.glmn = roc(y_test, y.pred.glmn)
plot(roc.log.lasso, legacy.axes = TRUE, print.auc = TRUE)
```




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

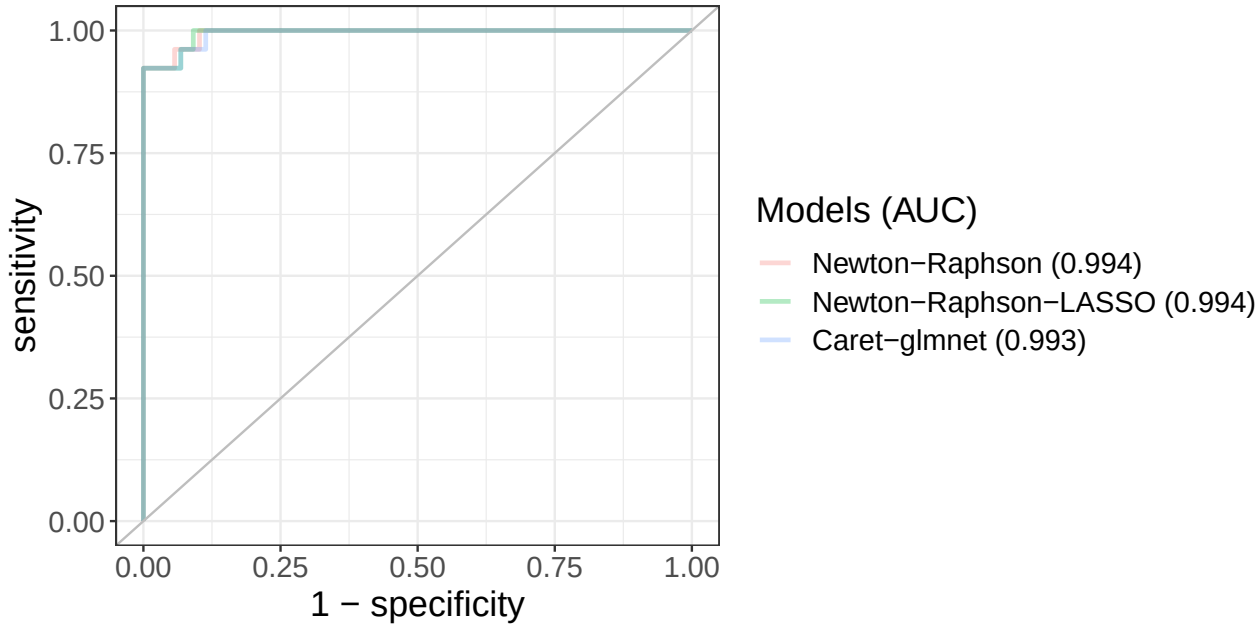



```
plot(roc.glmn, legacy.axes = TRUE, print.auc = TRUE)
```




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

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

```
ggroc(list(roc.log, roc.log.lasso, roc.glmn), legacy.axes = TRUE, size = 1, alpha = 0.3) +
  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 = 16))
```

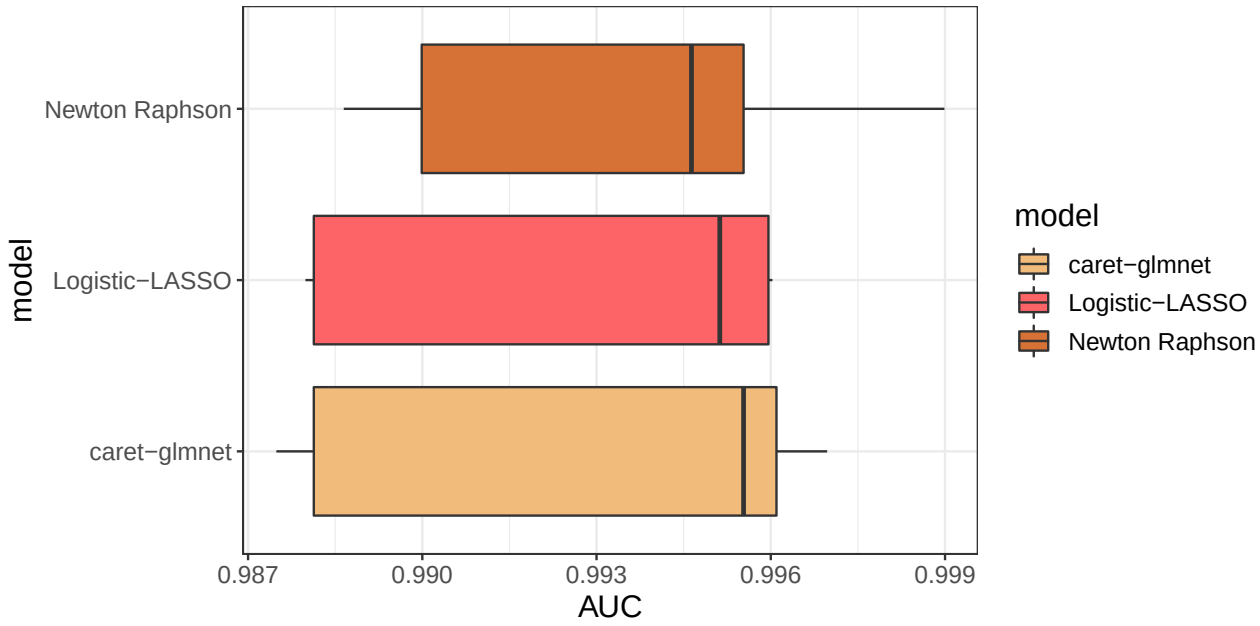



```
extract.cv.performance = function(x, y, k, betas) {
  set.seed(2022)
  data = as.data.frame(cbind(x, y))
  folds = crossv_kfold(data, k = k)

  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,]

    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))
  }
  return(fold.errors)
}
```

```
cv.roc.log.lasso = extract.cv.performance(x_train_b0, y_train, k = 5, as.vector(t(NR.lasso.coef)))
```

```
cv.roc.log = extract.cv.performance(x_train_b0, y_train, k = 5, as.vector(t(NR.coef)))
cv.roc.glmn = extract.cv.performance(x_train_b0, y_train, k = 5, as.vector(t(glmn.coef)))
```

```
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))
)

library(wesanderson)
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))
```




```
path <- function(X, y, lambdaseq){
    init_beta <- rep(1, 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(-1,-10, length=500)))
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()
```