A study of optimization algorithms on a breast cancer diagnosis dataset

Xinlei Chen, Guojing Wu and Yujing Yao

March 14, 2019

Abstract

This report discusses a study of optimization algorithms on breast cancer diagnosis dataset. Our goal is to build a predictive model based on logistic regression to facilicate cancer diagnosis, and we compared estimation method including Newton Raphson, Gradient Decent and Lasso. Our result shows that. . . ..

Background

The goal of the exerise is to build a predictive model based on logistic regression to facilicate cancer diagnosis.

Dataset

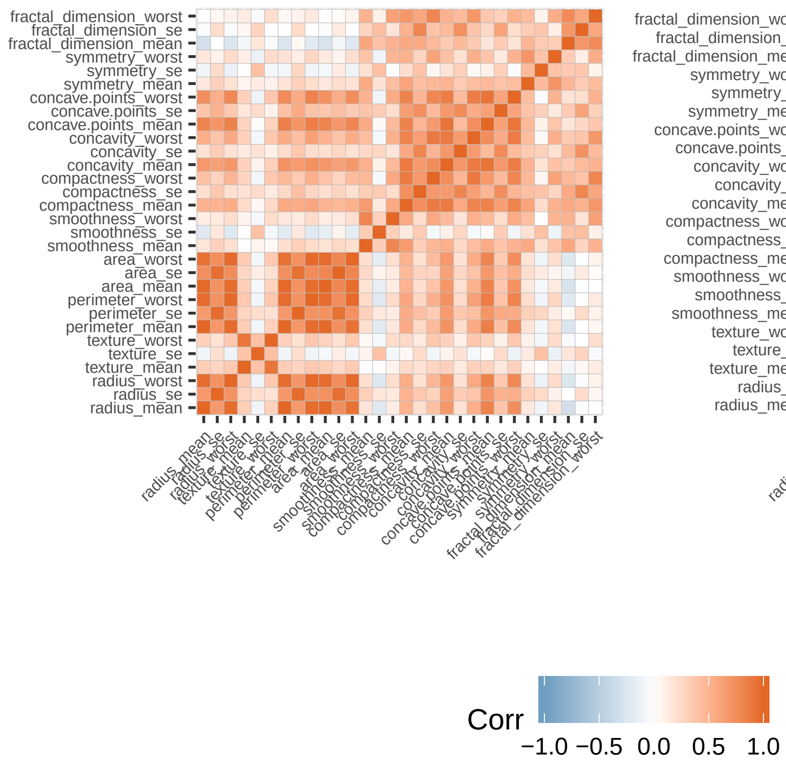
The data *breast-cancer.csv* have 569 row and 33 columns. The first column **ID** lables individual breast tissue images; The second column **Diagnonsis** indentifies if the image is coming from cancer tissue or benign cases (M=malignant, B = benign). There are 357 benign and 212 malignant cases. The other 30 columns correspond to mean, standard deviation and the largest values (points on the tails) of the distributions of the following 10 features computed for the cellnuclei.

- 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 ($perimeter^2/area$ - 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)

Variable selection

2. check multicolinearity -one plot

Figure 1: multicolinearity plot of the dataset



3. report final input of model - descripive mean(sd)

Method

Logistic Regression

Let y be the vector n response random variable, X denote the $n \times p$ design matrix(letX_i denote the ith row) and β denote the $p \times 1$ coefficient. The likelihood of logistic regression is:

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

Maximizing the likelihood is equivalent to maximizing the log likelihood:

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

$$= < X\beta, Y > - \sum_{i=1}^{n} \log(1 + \exp(X_i\beta))$$

Let p, a vector of n denote $p = \frac{\exp(X\beta)}{1+\exp(X\beta)}$. The gradient of this function is:

$$\nabla f(\beta) = X^T(y - p)$$

The Hessian is given by:

$$\nabla^2 f(\beta) = -X^T W X$$

where $W = diag(p_1(1 - p_1), p_2(1 - p_2), \cdots, p_n(1 - p_n))$ Hessian matrix is negative definite, well behaved.

Newton Raphson

Gradient Decent

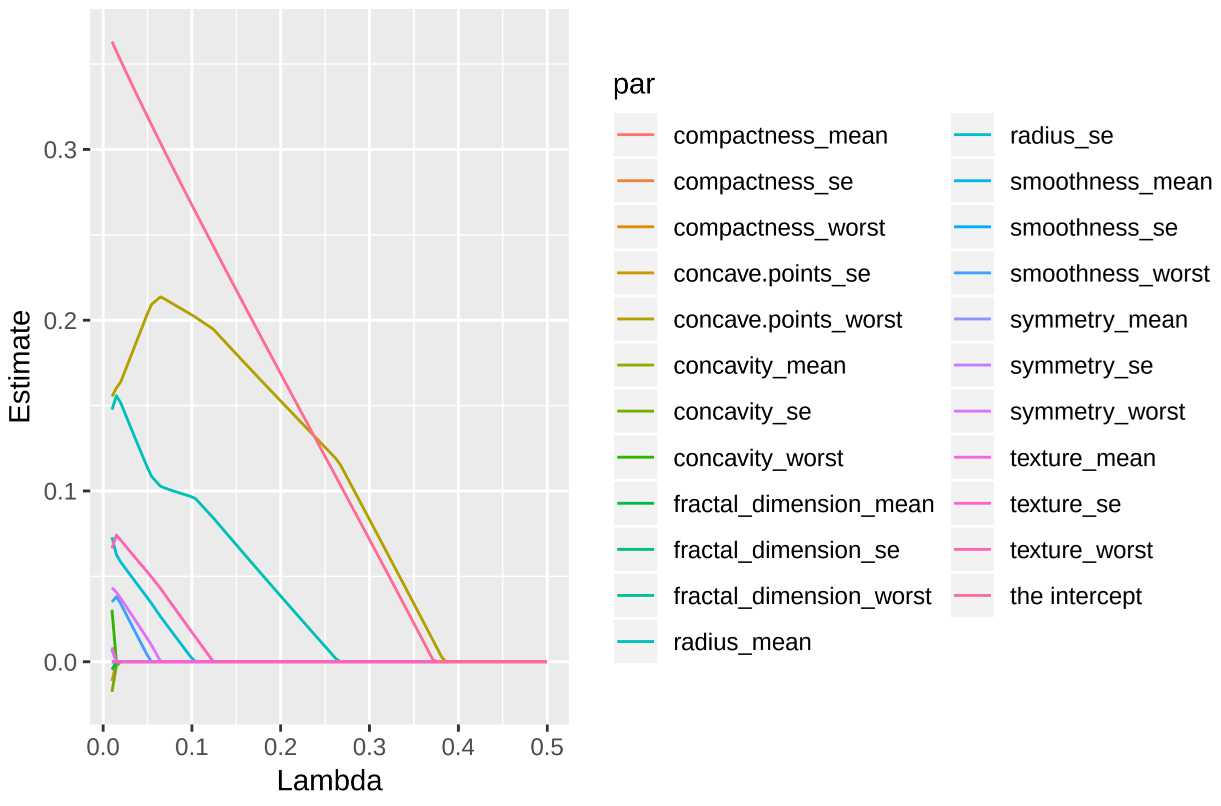
Lasso

problem 3&4

Results

Eestimation path for Lasso

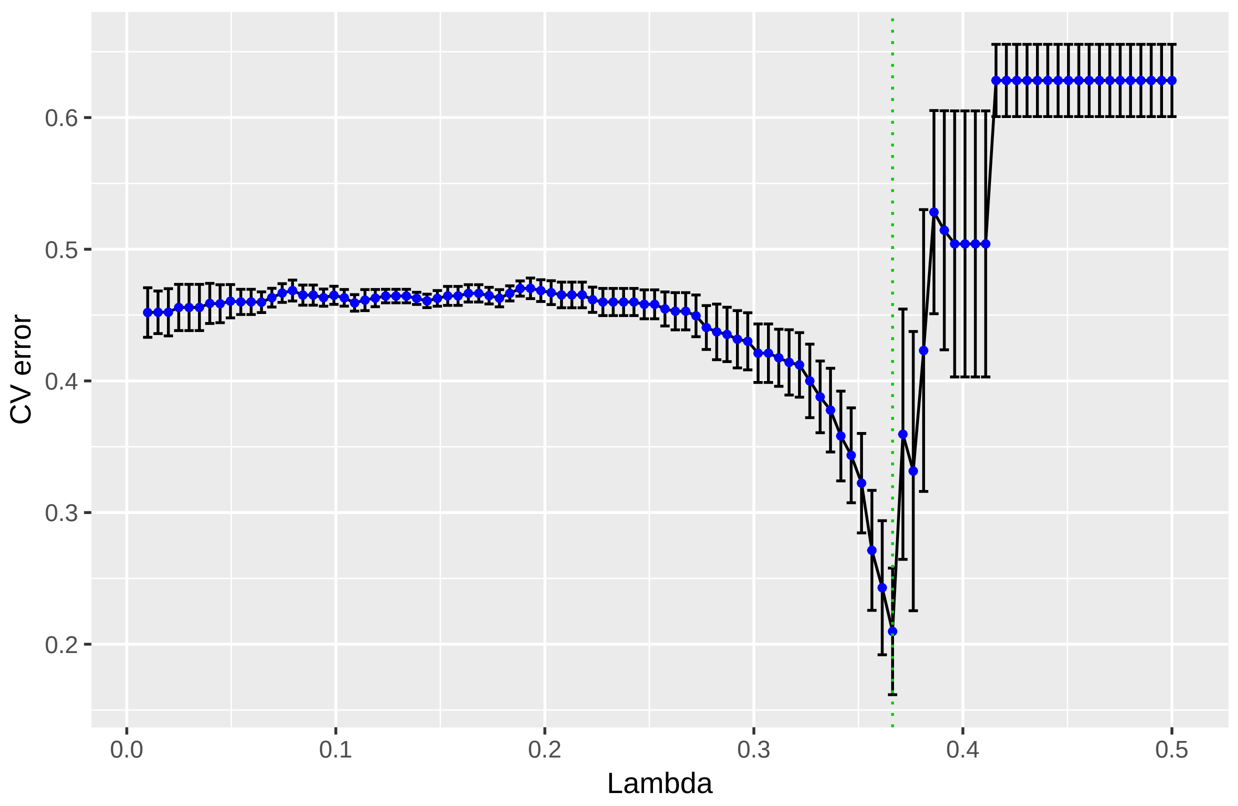
Figure 2: a path of solutions with a sequence of descending lambda's



Cross validation for lasso

one plot for beta one plot for lambda

Figure 3: Lasso regression by 5 fold cross validation



Model comparison

comparison table

Table 1: The comparison of performance for estimation algorithms and models

	GLM package	Newton Raphson	Gradient Decent	Logistic Lasso
iteration times	NA	14	201	3
prediction error	0.01	0.01	0.02	0.2

[a] Dataset: Breast Cancer Diagnosis

Conclusions

References

1 Bender, Ralf, Thomas Augustin, and Maria Blettner. "Generating survival times to simulate Cox proportional hazards models." Statistics in medicine 24.11 (2005): 1713-1723.

2 Austin, Peter C. "Generating survival times to simulate Cox proportional hazards models with time-varying covariates." Statistics in medicine 31.29 (2012): 3946-3958.

3 Morina, David, and Albert Navarro. "The R package survsim for the simulation of simple and complex survival data." Journal of Statistical Software 59.2 (2014): 1-20.

Table 2: The comparison of performance for estimation algorithms and models

	GLM package	Newton Raphson	Gradient Decent	Logistic Lasso
radius_mean	6.26	6.26	1.95	0.00
texture_mean	-1.16	-1.16	-0.20	0.00
smoothness_mean	2.59	2.59	0.49	0.00
compactness_mean	-7.32	-7.32	-1.23	0.00
concavity_mean	8.70	8.70	1.96	0.00
symmetry_mean	-1.52	-1.52	-0.24	0.00
fractal_dimension_mean	0.89	0.89	-0.39	0.00
radius_se	10.78	10.78	1.82	0.00
texture_se	-3.04	-3.04	-0.62	0.00
smoothness_se	1.82	1.82	0.39	0.00
compactness_se	5.97	5.97	0.40	0.00
concavity_se	-6.13	-6.13	-1.41	0.00
concave.points_se	4.17	4.17	0.96	0.00
symmetry_se	-2.18	-2.18	-0.30	0.00
fractal_dimension_se	-11.08	-11.08	-1.01	0.00
texture_worst	6.04	6.04	1.61	0.00
smoothness_worst	-1.24	-1.24	-0.06	0.00
compactness_worst	-6.86	-6.86	-1.08	0.00
concavity_worst	4.64	4.64	1.23	0.00
concave.points_worst	2.27	2.27	0.57	0.02
symmetry_worst	3.33	3.33	0.96	0.00
fractal_dimension_worst	8.33	8.33	1.28	0.00
intercept	-0.03	-0.03	-0.72	0.01

[a] Dataset: Breast Cancer Diagnosis

Appendix A

```r
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, comment = "")
library(tidyverse)
library(Hmisc)#for errvar
library(ggcorrplot) # for correlation heatmap
library(ggpubr) # common legend
library(matrixcalc) #is.negative.def
library(kableExtra) # table
set.seed(99999)
options(knitr.table.format = "latex")
###### data manipulation
#setwd("/Users/yujingyao/Desktop/")
mydata <- read.csv("breast-cancer-1.csv")
n <- dim(mydata)[1]
p <- dim(mydata)[2]
list <- c(3:(p - 1))
namesX <- names(mydata)[-c(1,2,p)]
# standardize
dataX <- do.call(cbind, lapply(list, function(x) (mydata[,x] - mean(mydata[,x]))/sd(mydata[,x])))
# design matrix
```

```r
X <- data.frame(dataX) %>%
  mutate(., intercept = 1)
# response
resp <- as.vector(ifelse(mydata[,2] == "M", 1, 0))
###### plot to check collinearity
colnames(dataX) <-namesX
colinearity.plot <- function(data){
  data.frame(data) %>%
  select(starts_with("radius"),
         starts_with("texture"),
         starts_with("perimeter"),
         starts_with("area"),
         starts_with("smooth"),
         starts_with("compact"),
         starts_with("concavity"),
         starts_with("concave"),
         starts_with("symmetry"),
         starts_with("fractal")) %>%
  cor() %>%
  ggcorrplot(.,ggtheme = ggplot2::theme_gray,
             colors = c("#6D9EC1", "white", "#E46726"),
             tl.cex = 6)}
g1 <- colinearity.plot(dataX)

###### variable selection
eig <- eigen(cor(dataX))$values
# found values very close to 0, multicolinearity exists
# function: find the maximum correlation between i and j
max.corr <- function(data){
  len = dim(data)[2]
  a <- 0.5
  for (i in 1:(len-1)) {
    for (j in (i+1):len) {
      if(abs(cor(data[,i],data[,j]))> a) a <- cor(data[,i],data[,j])
    }
  }
  return(round(a,3))
}
# function: update dataset according to several rules:eigenvalues and corr
selection <- function(data,eigen.tol,corr.tol){
  while (min(eigen(cor(data))$values) <= eigen.tol & max.corr(data) >= corr.tol){
       temp <- data
       data <- temp[,-(which(round(abs(cor(temp)),3) == max.corr(temp),arr.ind = TRUE)[1,1])]
      }
  return(data)}
newdataX <- selection(dataX, eigen.tol=1e-2,corr.tol = 0.8)
g2 <- colinearity.plot(newdataX)
eigpost <- eigen(cor(newdataX))$values
# look at the difference of the deleted colums
delnames <- setdiff(colnames(dataX),colnames(newdataX))
ggarrange(g1,g2, ncol=2, nrow=1, common.legend = TRUE, legend="bottom")

###### 0.logistic regression
```

```r
logfit.0 <- glm(resp~dataX, family = binomial(link = "logit"))
# algorithm didn't converge without delete colinearity
logdata <- cbind.data.frame(resp,newdataX)
logfit.1 <- glm(resp~., family = binomial(link = "logit"),data = logdata)
logit.beta <- coef(logfit.1)
##### 1. classical newton raphson
newX <- data.frame(newdataX) %>%
  mutate(., intercept = 1)
newdat <- list(y = resp, X = as.matrix(newX))
# function: calcualte loglik, gradient, hessian
logisticstuff <- function(dat, betavec){
  u <- dat$X %*% betavec
  expu <- exp(u)
  loglik <- t(u) %*% dat$y - sum((log(1 + expu))) # Log-likelihood at betavec
  prob <- expu / (1 + expu) # P(Y_i=1|x_i)
  grad <- t(dat$X) %*% (dat$y - prob)
  Hess <- -t(dat$X) %*% diag(as.vector(prob*(1 - prob))) %*% dat$X # Hessian at betavec
  return(list(loglik = loglik, grad = grad, Hess = Hess))}
NewtonRaphson <- function(dat, func, start, tol=1e-10, maxiter =200){
  i <- 0
  cur <- start
  stuff <- func(dat, cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf
  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol && stuff$loglik > -Inf) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    cur <- prev - solve(stuff$Hess) %*% stuff$grad
    # # step halving
    #     lambda <- 1
    #     curtemp <- prev - solve(stuff$Hess) %*% stuff$grad
    #     while(func(dat, curtemp)$loglik <= func(dat,prev)$loglik){
    #       lambda <- lambda*(1/2)
    #       curtemp <- prev - solve(stuff$Hess) %*% stuff$grad
    #     }
    #     cur <- curtemp
    # # step halving
    stuff <- func(dat, cur)
    res <- rbind(res, c(i, stuff$loglik, cur))}
  return(res)
}
newres <- NewtonRaphson(newdat,logisticstuff, start = rep(0, dim(newdat$X)[2]))
# check convergence
check <- tail(newres)[,1:2]
newton.beta <- newres[nrow(newres),3:dim(newres)[2]]
##### 2. gradient descent
gradient <- function(dat, func, start, tol=1e-10,maxiter =200){
  i <- 0
  cur <- start
  pp <- length(start)
  stuff <- func(dat, cur)
  hess <- t(dat$X)%*%(dat$X)# double check
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf
  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol && stuff$loglik > -Inf) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    cur <- prev + solve(hess) %*% (stuff$grad)
    stuff <- func(dat, cur)
    res <- rbind(res, c(i, stuff$loglik, cur))}
  return(res)
}
gradres <- gradient(newdat, logisticstuff, start = rep(0, dim(newdat$X)[2]),maxiter =200)
# check convergence
check <- tail(gradres)[,1:2]
grad.beta <- gradres[nrow(gradres),3:dim(gradres)[2]]
##### 3. coordinate-wise logistic lasso
sfun <- function(beta,lambda) sign(beta) * max((abs(beta) - lambda), 0)
coordinatelasso <- function(lambda, dat, start, tol=1e-10, maxiter = 200) {
  i <- 0
  pp <- length(start)
  n <- length(dat$y)
  betavec <- start
  u <- (dat$X) %*% betavec
  expu <- exp(u)
  prob <- expu/(expu+1)
  # weighted updates
  w <- prob*(1-prob)
  z <- u + (dat$y-prob)/w
  loglik <- t(u) %*% dat$y - sum(t(log(1 + expu))) - sum(lambda * abs(betavec))
  # quardratic approximation to the log likelihood
  # loglik <- 1/2*(1/n)*sum(w*(z-u)^2) - sum(lambda * abs(betavec))
  res <- c(0, loglik, betavec)
  prevloglik <- -Inf # To make sure it iterates
  while(i < maxiter && abs(loglik - prevloglik) > tol && loglik > -Inf) {
    i <- i + 1
    prevloglik <- loglik
    prev <- betavec
    for (j in 1:pp){
      ynoj <- dat$X[,-j] %*% betavec[-j]
      betavec[j] <- sfun(sum(w*(dat$X[,j])*(dat$y - ynoj))/(sum(w*(dat$X[,j])^2)), lambda)
    }
    u <- dat$X %*% betavec
    expu <- exp(u)
    loglik <- t(u) %*% dat$y - sum(t(log(1 + expu))) - sum(lambda * abs(betavec))
    # quardratic approximation to the log likelihood
    # loglik <- 1/2*(1/n)*sum(w*(z-u)^2) - sum(lambda * abs(betavec))
    res <- rbind(res, c(i, loglik, betavec))}
  return(res)
}
corres <- coordinatelasso(lambda = 0.5, newdat, start = rep(0, dim(newdat$X)[2]))
check <- tail(corres)[,1:2]
cor.beta <- corres[nrow(corres),3:dim(corres)[2]]
# impletement the pathwise coordinatewise optimization algorithm to obtain a path of solutions
```

```r
path <- function(inputx,inputy,grid){
  start <- rep(0, dim(inputx)[2])
  betas <- NULL
  for (x in 1:100) {
  cv.errors <- vector()
    cor.result <- coordinatelasso(lambda = grid[x],
                                  dat = list(X=as.matrix(inputx),y=inputy),
                                  start= start)
    lasbeta <- cor.result[nrow(cor.result),3:dim(cor.result)[2]]
    start <- lasbeta
    betas <- rbind(betas,c(lasbeta))
  }
  return(data.frame(cbind(grid,betas)))
}
path.out <- path(newX,resp,grid=seq(0.5, 1e-2, length=100))
colnames(path.out) <- c("grid",colnames(newdataX),"the intercept")
# plot a path of solutions
path.plot <- path.out %>%
  gather(key = par, value = estimate, c(2:dim(path.out)[2])) %>%
  ggplot(aes(x = grid, y = estimate, group = par, col = par)) +
  geom_line()+
  ggtitle("Figure 2: a path of solutions with a sequence of descending lambda's")+
  xlab("Lambda") + ylab("Estimate")
path.plot
##### 5-fold cross-validation and pathwise coordinatewise optimization algorithm
# when lambda = 0.5, all the betas go to 0
cvresult <- function(inputx,inputy,grid,K){
  n <- dim(inputx)[1]
  folds <- sample(1:K, n, replace=TRUE)
  start <- rep(0, dim(inputx)[2])
  cv.error <- vector()
  cv.se <- vector()
  for (x in 1:100) {
  cv.errors <- vector()
  for (i in 1:K){
    cor.result <- coordinatelasso(lambda = grid[x],
                                  dat = list(X=as.matrix(inputx[folds!=i,]),y=inputy[folds!=i]),
                                  start= start)
    lasbeta <- cor.result[nrow(cor.result),3:dim(cor.result)[2]]
    u <- as.matrix(inputx[folds==i,])%*%lasbeta
    expu <- exp(u)
    prob <- expu / (1 + expu)
    pred <- ifelse(prob >=0.5,1,0)
    cv.errors[i] = mean((as.vector(inputy[folds==i])-as.vector(pred))^2) # MSE
    start <- lasbeta
  }
  cv.error[x] <- mean(cv.errors)
  cv.se[x] <- sqrt(var(cv.errors)/K)
  }
  return(cbind(grid,cv.error,cv.se))
}

result <- cvresult(newX,resp,grid=seq(0.5, 1e-2, length=100),K=5)
```

```r
# result <- cvresult(X,resp,grid=seq(0.5, 1e-2, length=100),K=5)
best.lambda <- result[which.min(result[,2]),1]
finlasso <- coordinatelasso(lambda = best.lambda, newdat, start = rep(0, dim(newdat$X)[2]))
check <- tail(finlasso)[,1:2]
lasso.beta <- finlasso[nrow(finlasso),3:dim(finlasso)[2]]

# plot for cross validation
result <- data.frame(result)
cv.plot <-
    ggplot(result, aes(x=result$grid, y=result$cv.error)) +
    geom_errorbar(aes(ymin=result$cv.error-result$cv.se, ymax=result$cv.error+result$cv.se),
                  colour=1) +
    geom_line() +
    geom_point(size=1,colour = 4) +
    ggtitle("Figure 3: Lasso regression by 5 fold cross validation")+
    xlab("Lambda") + ylab("CV error") +
    geom_vline(xintercept = best.lambda,col=3,lty=3)
cv.plot
######## compare prediction performance of all results
pred.fun <- function(outcome,input, beta){
    u <- as.matrix(input)%*%beta
    expu <- exp(u)
    prob <- expu / (1 + expu)
    pred <- ifelse(prob >=0.5,1,0)
    pred.error = mean((as.vector(outcome)-as.vector(pred))^2)
    return(pred.error)
}
# logistic regression by GLM
log.beta <- c(logit.beta[2:length(logit.beta)],logit.beta[1])
log.pred <- mean((resp-predict(logfit.1,type="response"))^2) # abs(mean(logfit.1$residuals))
# newton's method
newton.ite <- nrow(newres)
newton.beta <- newres[nrow(newres),3:dim(newres)[2]]
newton.pred <- pred.fun(resp,newX,newton.beta)
# gradient decent
grad.ite <- nrow(gradres)
grad.beta <- gradres[nrow(gradres),3:dim(gradres)[2]]
grad.pred <- pred.fun(resp,newX,grad.beta)
# lasso logistic
lasso.ite <- nrow(finlasso)
lasso.beta <- finlasso[nrow(finlasso),3:dim(finlasso)[2]]
lasso.pred <-  pred.fun(resp,newX,lasso.beta)

beta.res <- round(as.matrix(rbind(log.beta,newton.beta,grad.beta,lasso.beta)),2)
colnames(beta.res) <- colnames(newX)
rownames(beta.res) <- c("GLM package","Newton Raphson","Gradient Decent","Logistic Lasso")
perf.res <- matrix(rep(NA),ncol = 2, nrow = 4)
colnames(perf.res) <- c("iteration times","prediction error")
rownames(perf.res) <- c("GLM package","Newton Raphson","Gradient Decent","Logistic Lasso")
perf.res[1,1] <- "NA"
perf.res[1,2] <- round(log.pred ,2)
perf.res[2,1] <- newton.ite
perf.res[2,2] <- round(newton.pred,2)
perf.res[3,1] <- grad.ite
perf.res[3,2] <- round(grad.pred,2)
perf.res[4,1] <- lasso.ite
perf.res[4,2] <- round(lasso.pred,2)

# output-performace
kable(t(perf.res), "latex", caption = "The comparison of performance for estimation algorithms and model
  kable_styling(latex_options = c("hold_position", "scale_down")) %>%
  add_footnote(c("Dataset: Breast Cancer Diagnosis"),
               notation = "alphabet")

# output-beta
kable(t(beta.res), "latex", caption = "The comparison of performance for estimation algorithms and model
  kable_styling(latex_options = c("hold_position", "scale_down")) %>%
  add_footnote(c("Dataset: Breast Cancer Diagnosis"),
               notation = "alphabet")
```