A study of optimization algorithms on a breast cancer diagnosis dataset

Ruoyuan Qian rq2166, Yimeng Shang ys3298, Zongchao Liu zl2860

March 20, 2020

Introduction

Cancer detection based on typical clinical features is being increasingly critical to population health. It is a desirable way to help clinicial practitioners enhance diagnosis accuracy and prevent faults. In this report, we built several predictive models to facilitate cancer diagnosis. Our goal is to compare optimization methods including the Newton-Raphson with classic logistic regression and pathwise coordinate-wise with LASSO logistic regression. The two methods were compared based on the correct classification rate attained through a 10-fold cross-validation process.

Method

Dataset

Our data set consists of 569 observations and 33 rows. The response of the following model is a binary variable, diagnosis, in which M represents malignant tissues and B represents benign tissues. For analysis, we recoded malignant as 1, benign as 0.

There are 30 potential predictors in the data set corresponding to mean, standard deviation and the largest values (points on the tails) of the distributions of the features computed for the cell nuclei. For our analysis, we used both the "full data" and a "subset" of the data. The subset of the data was attained by picking up some of the predictors. In the subset data, the selected predictors were:

- radius_mean: mean of distances from center to points on the perimeter
- texture_mean: mean of gray-scale values
- perimeter_mean: mean size of the core tumor
- area_mean: mean area of the core tumor
- smoothness_mean: local variation in radius lengths
- compactness_mean: $perimeter^2$ / area - 1.0
- concavity_mean: severity of concave portions of the contour
- concave_points_mean: average number of concave portions of the contour
- symmetry_mean: symmetry of the tumor
- radius_se: variation of distances from center to points on the perimeter
- perimeter_se: variation of the sizes of the core tumor

- symmetry_se: variation of symmetry

Before using the data, we standardized it by subtracting the mean and divided by the square root of sum of the vector to make the norm of it equals to one. Moreover, to randomly divided the data into training and testing data(for cross-validation), we used random() function from R to randomize the dataset.

Evaluation metric

For prediction ability, we chose the following criteria:

Correct classification rate, which is defined as:

$$correct\ classifcation\ rate = \frac{1}{n}\sum_{i=1}^{n}|\widehat{y}_i - y_i|$$

Newton-Raphson model

Let y be the vector of n response random variable, X denote the $n \times p$ design matrix (X_i denote the ith row) and β denote the $p \times 1$ coefficient. Let $\eta = E(y) = X\beta$ and given the link function as $g(\eta) = \log\frac{\eta}{1-\eta}$, we have the logistic regression model written as:

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

The likelihood of this logistic regression is:

$$L(\beta) = \prod_{i=1}^{n}[(\frac{e^{X_i^T\beta}}{1+e^{X_i^T\beta}})^{y_i}(\frac{1}{1+e^{X_i^T\beta}})^{1-y_i}]$$

Maximizing the likelihood is equivalent with maximizing the log likelihood

$$f(\beta) = \sum_{i=1}^{n}(Y_i(X_i^T\beta) - \log(1+e^{X_i^T\beta}))$$

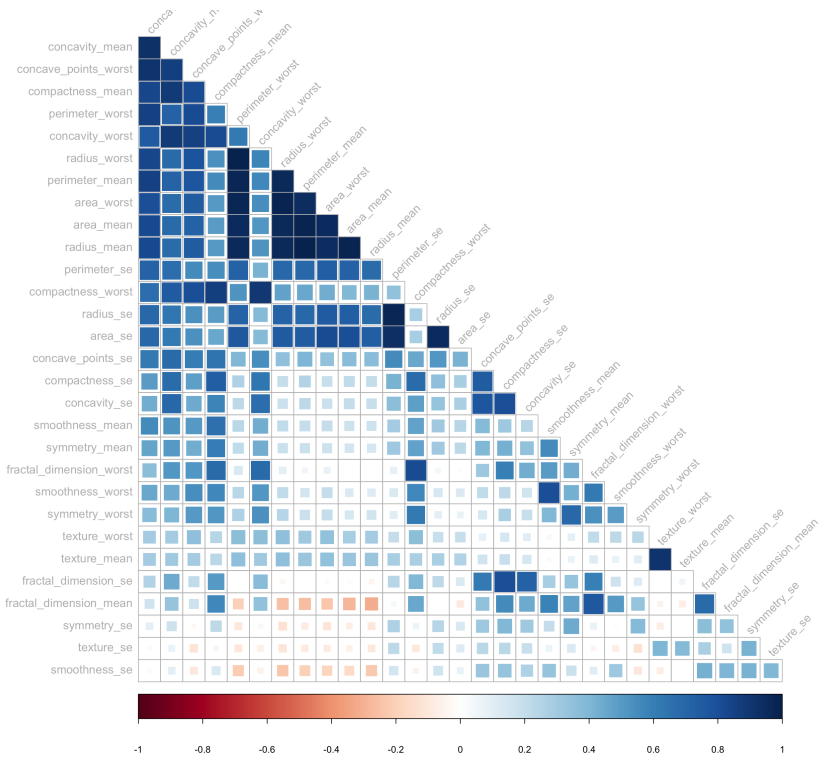
The gradient is given by:

$\nabla f(\beta) = X^T(Y-P)$, where X is design matrix, Y is a vector of y_i, P is a vector and is equal to $P = \frac{e^{X^T\beta}}{1+e^{X^T\beta}}$

Then we get the Hessian matrix:

$\nabla^2 f(\beta) = -X^TWX$, where W is diagonal matrix and is equal to $W = diag(p_1(1-p_1), p_2(1-p_2), ..., p_n(1-p_n))$, $p_i = \frac{e^{X_i^T\beta}}{1+e^{X_i^T\beta}}$

From the correlation plot, it's clearly that there are several highly correlated covariates. Before fitting the logistic model, we firstly select some non-correlated predictors. We then used Newton-Raphson algorithm to optimize the coefficients. The initial guess for Newton-Raphson we choose is that all coefficients equal to 1. For each iteration, we guarantee that the hessian matrix is negative definite by checking it's max eigenvalue and furtherly we checked add a λ for each step to make sure that the likelihood is increasing at each step. We defined convergence as when the difference between the current log-likelihood and the log-likelihood of the last iteration reached below 10^{-3}.



Logistic-LASSO model

If we Taylor expansion the log-likelihood around "current estimates", we get a quadratic approximation to the log-likelihood of logistic regression, which shows that logistic regression can be viewed as a weighted linear regression:

$$f(\beta_0, \beta_{\mathbf{1}}) \approx l(\beta_0, \beta_{\mathbf{1}}) = -\frac{1}{2n}\sum_{i=1}^{n}\omega_i(z_i - \beta_0 - \mathbf{x}_i^T\beta_{\mathbf{1}})^2 + C(\tilde{\beta}_0, \tilde{\beta}_{\mathbf{1}})$$

where we have the working response:

$$z_i = \tilde{\beta}_0 + \mathbf{x}_i^T\tilde{\beta}_{\mathbf{1}} + \frac{y_i - \tilde{p}_i(x_i)}{\tilde{p}_i(x_i)(1-\tilde{p}_i(x_i))}$$

the working weights:

$$\omega_i = \tilde{p}_i(x_i)(1-\tilde{p}_i(x_i))$$

and the probability given the current parameters:

$$\tilde{p}_i(x_i) = \frac{e^{\tilde{\beta}_0 + \mathbf{x}_i^T\beta_{\mathbf{1}}}}{1 + e^{\tilde{\beta}_0 + \mathbf{x}_i^T\beta_{\mathbf{1}}}}$$

The coordinate descent algorithm to solve the penalized weighted least squares problem aiming to minimize the objective function:

$$\min_{\beta\in\mathbb{R}^{p+1}}\{-f(\beta) + \lambda P(\beta)\}$$

To minimize the objective function, we apply coordinate descent algorithm, where the coefficients except intercept term was penalized by tuning parameter λ times l1 norm. Each $beta_k$ was optimized using the following equation, where S() is the soft threshold of LASSO regression:

$$\tilde{\beta}_j^{lasso}(\lambda) \leftarrow \frac{S(\sum_{i=1}^n \omega_i x_{i,j}(z_i - \tilde{z_i}^{(-j)}), \lambda)}{\sum_{i=1}^n \omega_i x_{i,j}^2}$$

In order to choose a best penalty parameter λ, we conducted a range of λ and calculate the pathway of solution. Using 5 fold cross validation, we can get the correct classification rate for each λ. The λ with highest correct classification rate is the best λ we chose.

Model comparasion method

To compare the two models, we use cross validation from 1 to 10 folds to get the mean of correct classification rate of each model. The one with higher correct classification rate is the better one.

Result

Logistics regression

The Newton-Raphson algorithm was used for updating parameters of the logistic regression model. The parameters obtained through Newton-Raphson algorithm are very close to what are estimated by implementing the glm( ) function in the 'stat' package (Table 1). The slight difference in coefficients might be due to the setting of convergence tolerance, iteration count and initial betas.

LASSO logistic regression

Coefficients

A pathwise coordinate-wise optimization algorithm was implemented to obtain a path of solutions with a descending sequence of λ's. The trace plots (**Fig. 1, Fig. 2**) of the LASSO logistic regression model show that the variable estimates of parameters were penalized by λ. The estimates shrink to 0 accordingly when λ increases. As a result of the penalty term λ, when it is extremely large, all the estimates of the parameters will go to 0. The estimated parameters of LASSO logistic regression are shown in Table 1, compared with classic logistic regression.

Figure 1 Trace plot of LASSO-logistic fit with all data

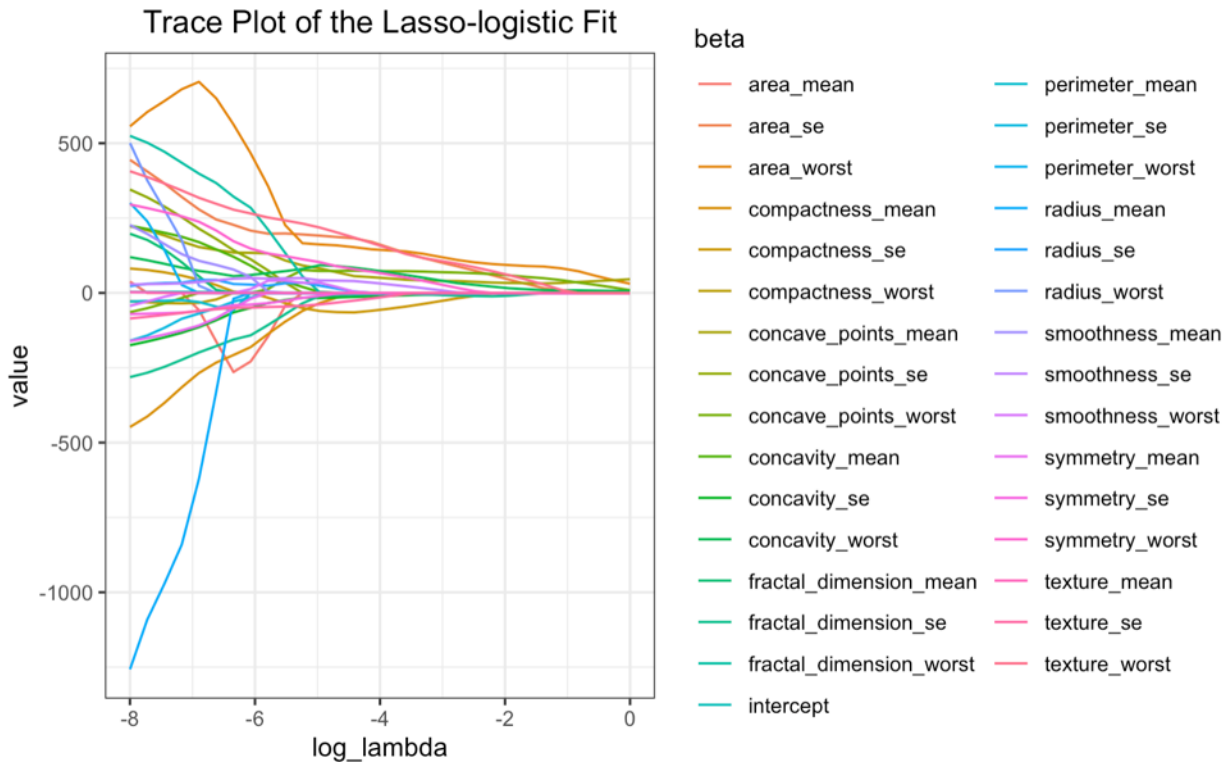
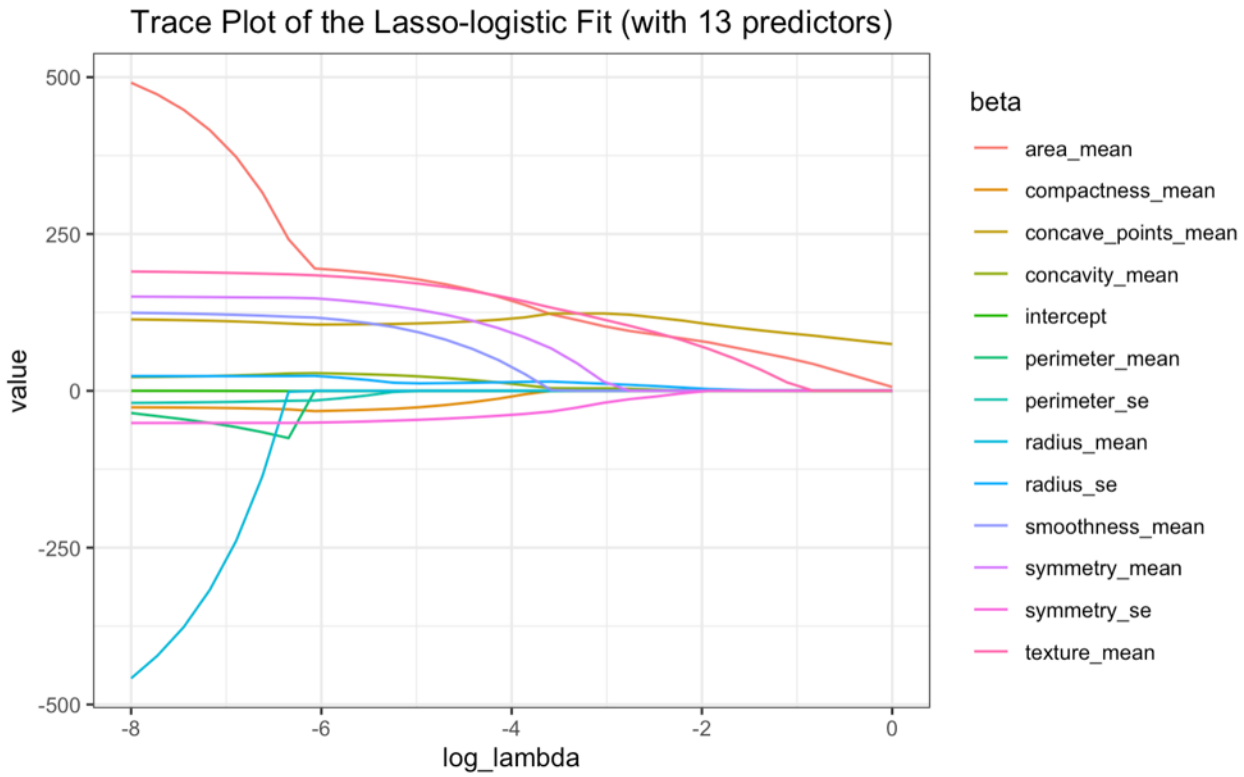


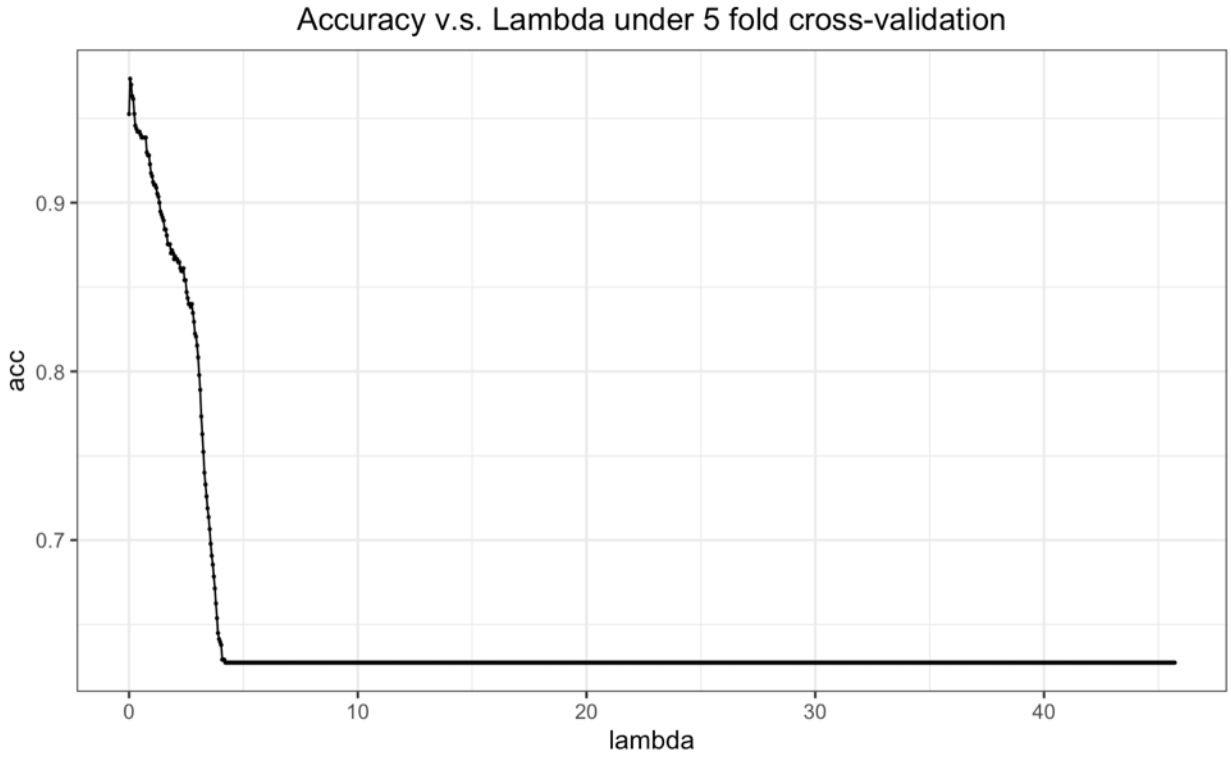
Figure 2 Trace plot of LASSO-logistic fit with 13 predictors



Lambda for lasso logistic regression

To select the best tuning parameter λ of the LASSO model, we conducted a 5-fold cross-validation under the scenario of a total number of 1000 λ's. For each λ, we calculated the average correct classification rate of the test sets and selected the λ that corresponds to the highest correct classification rate as the final penalty term. The resulting λ is 0.074. **Fig. 3** shows the range of λ and its corresponding correct classification rate.

Figure 3 Correct classification rate across different lambda



Model comparison

Coefficient comparison

We compared these methods: classic logistic Regression via Newton-Raphson Algorithm, classic logistic regression via R package (stats), LASSO logistic regression via pathwise coordinate-wise descent (CD) algorithm and LASSO logistic regression via R package (glmnet). **Table 1** shows all the estimates of parameters within each method. We see that the estimated parameters of LASSO logistic regression via CD were different from the parameters of the model fitted by R package. This might be because the R package does not penalize the intercept but our CD method does. The LASSO model obtained through CD was simpler than the model built by the R package.

Table 1. Coefficient estimates for logistic regression model

Beta	Estimation			
	LASSO-logistic(Newton-Raphson)	Logistic (by package)	LASSO-logistic (Coordinate Descent)	LASSO-logistic (by package)
intercept	0.3171993	0.3171996	-0.4310572	-0.4639651
radius_mean	-685.1867076	-685.1871312	0	0
texture_mean	193.4665868	193.466615	97.672251	161.4163433
perimeter_mean	-108.4934723	-108.493329	0	0
area_mean	669.8811206	669.8813162	92.1327588	169.2110091
smoothness_mean	124.0695786	124.0695929	0	90.5777635
compactness_mean	-21.0068543	-21.0068588	0	-13.2058936
concavity_mean	17.069388	17.0693856	2.4576969	17.8549767
concave_points_mean	121.6540579	121.654078	118.7520697	94.4988156
symmetry_mean	149.5601328	149.560158	0	118.5511138
radius_se	18.7009291	18.7009294	8.4386991	11.6445167
perimeter_se	-16.0289194	-16.0289205	0	0
symmetry_se	-51.0861786	-51.0861847	-10.940971	-41.6421708

Prediction performance comparison

We also compared the prediction performance on the "full data" and the "sub-data" by using 10 fold cross-validation. The average correct classification rate (1 - misclassification rate) was used as the criterion of performance. While we are building the models based on the subset of the data (with only 12 predictors rather than 31), the average correct classification rate for the classic logistic model is 93.1%, lower than the LASSO with 97.0% (**Table 2**). Since LASSO has been considered as an effective way for feature selection, we also applied our LASSO model in the full data(with 31 predictors) to see if it did well in predicting among excessive variables. Surprisingly, we found that the LASSO model built on the full dataset had a higher prediction accuracy (93.8%) than the classic logistic regression model originally built on the sub-dataset (**Table 2**). This implied that if we had not done a feature selection work before, the LASSO model would be a more effective tool to help select the optimal combination of predictors.The result might also suggest that the sub-dataset with the 12 predictors was not the best subset of variables for prediction.

Table 2. Prediction Performance (using the subset)

Test Set	correct classification rate		
	Classic Logistic Regression	LASSO (full data)	LASSO (sub-data)
fold1	0.9107143	0.9821429	0.9285714
fold2	0.9821429	0.9642857	0.9821429
fold3	0.9821429	0.9821429	0.9821429
fold4	0.8928571	0.9285714	0.8928571
fold5	0.9285714	0.9464286	0.9642857
fold6	0.9464286	1	0.9642857
fold7	0.875	0.9464286	0.875
fold8	0.9285714	0.9642857	0.9285714
fold9	0.9107143	1	0.9107143
fold10	0.9538462	0.9846154	0.9538462
Average	0.9310989	0.9698901	0.9382418

Conclusion

In this report, logistic regression with Newton-Raphson optimization and LASSO via CD were conducted to predict the outcome of breast cancer. A five-fold cross validation was used to select the best λ from a sequence of λ's by the highest correct classification rate. Then we compared the coefficient estimation for logistic regression with Newton-Raphson optimization, logistic regression in R package, LASSO via CD and LASSO from in R package. The coefficient estimates for two logistic regressions were similar while the model of LASSO via CD was simpler than the LASSO model attained by the R package. As for comparison in prediction performance, logistic regression based on subset and LASSO via CD based on subset and full data were compared based on correct classification rate. LASSO with full data performed best.

Discussion

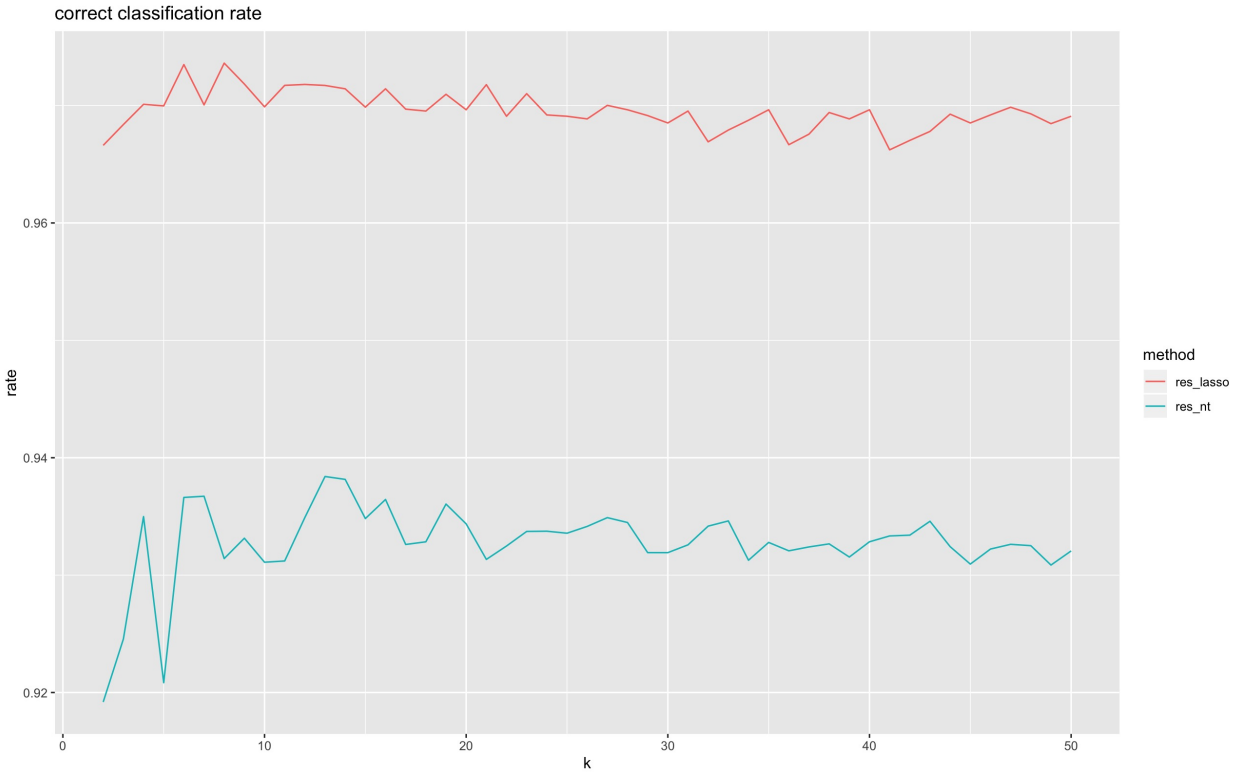
Randomization

The method of randomization we used to implement cross-validation is a little different from the traditional way. The observations in dataset are randomized and are split into k folds based on its new order instead of randomly sampling k folds from original dataset. Therefore, randomization is done as well in our method but just in a different way.

k-folds cross validation

We change the number of folds k in cross validation to check the relation between k and mean correct classification rate in different methods. It turns out that there is no trend between k and mean correct classification rate, and the mean correct classification rate of LASSO is always higher than logistic regression with modified Newton-Raphson optimization **(Fig 4)**.

Figure 4 Correct classification rate across different number of folds



Prospect

The original dataset has 10 features with 3 measurements for each, which may result to high collinearity. Unlike LASSO, logistic regression is unable to do variable selection, so we select predictors for it based on the correlation matrix, more effort could be done to find a better way of variable selection for logistic regression.

Appendix - R code

```
################Content#########################
#########################################
# 1. Functions preparation
# 2. Data manipulation
# 3. Call functions and solving problems
# 4. Code for the results and conclusion
#########################################
```

1. Function preparation

normalization

```
normal = function(x){
  for (i in 2:ncol(x)) {
    x[,i] = ((x[,i] - mean(x[,i]))/sqrt(sum(x[,i]^2)))}
  return(x)
}
```

Newton-Raphson

```
improved_NR = function(x, y, func, gradient, hessian, tol){
  theta = matrix(rep(1, ncol(x)), nrow = ncol(x))
  theta0 = theta
  iter_count = 0
  first_ite = TRUE
  while (first_ite | abs(func(theta, x ,y) - func(theta0, x ,y)) > tol & iter_count < 6000) {
    first_ite = FALSE
    lambda = 1
    f0 = func(theta, x ,y)
    gamma = max(eigen(hessian(theta, x ,y))$values) # get the max eigenvalues of hessian
    if (gamma < 0) { # if all eigenvalues < 0, it's finite definite
      direc = solve(hessian(theta, x ,y)) %*% gradient(theta, x ,y)
    }
    else {#If there exists any eigenvalue > 0, it's not definite.
      #We subtract the max eigenvalue plus 0.5 from all eigenvalue to make it finite definite
      n = nrow(theta)
      direc = solve(hessian(theta, x ,y) - (gamma + 0.5)*(diag(1,n,n))) %*% gradient(theta, x ,y)
    }
    theta_in = theta - lambda * direc
    for (i in 1:10 ) {
      if (func(theta_in, x ,y) < f0) {
        labmda = lambda/2
        theta_in = theta - lambda * direc
      }
      else{break}
    }
    theta0 = theta
```

```
    theta = theta_in

    iter_count = iter_count + 1
    #result = paste(iter_count, theta)
    #print(result)
#    if(iter_count %% 100 == 0) {
#      write.csv(theta, "/Users/yimeng/2020Spring/Computing/HW/Project2/proj2_sym")
#    }

  }
  #print(iter_count)
  return(theta)
}
```

Computation of loglikelihood hessian

```
loglikelihood = function(theta, x ,y) {
  eta = x %*% theta
  h = exp(eta)
  return(sum( y * eta - log(1 + h)))
}


gradlikelihood = function(theta, x, y) {
  eta = x %*% theta
  h = exp(eta)
  p = h / (1 + h)
  return(t(x) %*% (y - p))
}

hessianlog = function(theta, x, y) {
  eta = x %*% theta
  h = exp(eta)
  p = h / (1 + h)

  temp = t(x)
  for (i in 1:nrow(temp)) {
    temp[i,] = temp[i,] * p * (1 - p)
  }
  return(-temp %*% x)
}
```

Weighted related functions

```
weights_w = function(x, y, beta) {
  p = exp(x %*% beta)/(1 + exp(x %*% beta))
  #beta are coefficients
  w = p * (1 - p)
  return(w)
}
```

```
resp_z = function(x, y, beta){
  p = exp(x %*% beta) / (1 + exp(x %*% beta))
  z = x %*% beta + (y - p)/(p * (1 - p))
  return(z)
}


LogLassoLoss = function(x, y, lambda, beta) {
  z = resp_z(x, y, beta)
  w = weights_w(x, y, beta)
  log_lik =  (1/(2*nrow(x)) * sum(w * (z - x %*% beta)^2)) #### - or + ???
  return(log_lik)
}

soft_threshold  = function(beta, lambda) {
  if (beta > 0 & lambda < abs(beta)) {
    return(beta - lambda)
  }
  else if (beta < 0 & lambda < abs(beta)) {
    return(beta + lambda)
  }
  else {
    return(0)
  }
}
```

Logistic-Lasso functions

```
CoorWiseLasso = function(loss, lambda, x, y, tol=0.01) {
  max_iter = 200
  iter_count = 0

  beta = as.matrix(rep(1, ncol(x)))
  beta0 = beta

  while (iter_count == 0 |  iter_count < max_iter) {
    iter_count = iter_count + 1
    beta0 = beta
    for (j in 1:ncol(x)) {
      xbeta = x %*% beta
      p = exp(xbeta)/(1 + exp(xbeta))
      w = p * (1 - p)
      wz = w * xbeta + y - p

      numerator = sum(wz * x[,j] - w * x[,j] * (x[,-j] %*% as.matrix(beta[-j])))
      denominator = (x[,j]^2 %*% w)
      beta[j] = soft_threshold(numerator, lambda) / denominator
    }
  }
  return(beta)
}
```

```
# change lambda to get a dataframe
change_lambda = function(x, y, loss, tol, n) {
  lambda_max = max(CoorWiseLasso(loss, lambda = 0, x, y, tol))
  #lambda_max = 1
  lambda = exp(seq(from = -8, to = 0, length.out = n))
  #lambda = seq(from = lambda_max, to = 0, length.out = n)
  df_total = data.frame()
  for (i in 1:length(lambda)) {
    beta = CoorWiseLasso(loss, lambda[i], x, y, tol)
    result = data.frame(lambda[i], t(beta))
    df_total = rbind(df_total, result)
  }
  return(df_total)
}
```

Cross-Validation

```
## CV for select model
CV_model = function(k, data, pred_fun, tol, Newton = TRUE, lambda = 1) {
  result = c()
  ## split the data into k folds
  cvSplits = list()
  for (i in 0:(k - 2)) {
    size = as.integer(nrow(data)/k)
    cvSplits[[i + 1]] = data[(1 + i*size):(size + i*size), ]
  }
  last_size = nrow(data) - size*(k - 1)
  cvSplits[[k]] = data[(nrow(data) - last_size + 1):(nrow(data)), ]

  ## seperate train and test data
  for (i in 1:k) {
    test = data.frame()
    train = data.frame()
    # test data
    test = cvSplits[[i]]
    test_x = test[, 2:ncol(test)]
    test_y = test[, 1]
    # train data
    for (j in 1:k) {
      if (j != i) {
        train = rbind(train, cvSplits[[j]])
      }
    }
    train_x = train[, 2:ncol(train)]
    train_y = train[, 1]
    train_x = as.matrix(train_x)
    train_y = as.matrix(train_y)
    test_x = as.matrix(test_x)
    test_y = as.matrix(test_y)

    ## do regression
```

```
    if (Newton) {

      theta = improved_NR(train_x, train_y, loglikelihood, gradlikelihood, hessianlog, tol)
      pred_y = pred_fun(test_x, theta)

    } else {
      theta = CoorWiseLasso (LogLassoLoss, lambda, train_x, train_y, tol)
      pred_y = pred_fun(test_x, theta)
    }

    # predictive ability measure
    ## ? do we use MSE or log loss function
    ans = 1 - sum(abs(test_y - pred_y))/nrow(test_x)
    result = c(result, ans)
  }
  return(result)

}
```

```
## CV for select best lambda(n fold cv)
CV_lambda = function(k, data, pred_fun, lossfunc, tol, n) {
  result = c()


  y = data[,1]
  x = data[,2:ncol(data)]

  beta = as.matrix(rep(1, ncol(x)))
  lambda_max = max(CoorWiseLasso(lossfunc, lambda = 1, x, y, tol))
  lambda = seq(from = lambda_max, to = 0, length.out = n)

  res_cv = c(rep(0,n))
  for (lamda_index in 1:n) {
    lambda_inuse = lambda[lamda_index]
    ans = c(rep(0,k))

    ## split the data into k folds
    cvSplits = list()
    for (i in 0:(k - 2)) {
      size = as.integer(nrow(data)/k)
      cvSplits[[i + 1]] = data[(1 + i*size):(size + i*size), ]
    }
    last_size = nrow(data) - size*(k - 1)
    cvSplits[[k]] = data[(nrow(data) - last_size + 1):(nrow(data)), ]

    ## seperate train and test data
    for (i in 1:k) {
      test = data.frame()
      train = data.frame()
      # test data
      test = cvSplits[[i]]
      test_x = test[, 2:ncol(test)]
      test_y = test[, 1]
```

```
      # train data
      for (j in 1:k) {
        if (j != i) {
          train = rbind(train, cvSplits[[j]])
        }
      }
      train_x = train[, 2:ncol(train)]
      train_y = train[, 1]
      train_x = as.matrix(train_x)
      train_y = as.matrix(train_y)
      test_x = as.matrix(test_x)
      test_y = as.matrix(test_y)

      theta_inuse = CoorWiseLasso(lossfunc, lambda_inuse, train_x, train_y, tol)
      pred_y = pred_fun(test_x, theta_inuse)
      ans[i] = 1 - sum(abs(test_y - pred_y))/nrow(test_x)
    }
    res_cv[lamda_index] = mean(ans)
  }
  print(res_cv)
  max_index = which.max(res_cv)
  best_lambda = lambda[max_index]
  return(data.frame(acc = res_cv,
                    lambda = lambda))
}
```

Prediction

```
predict = function(x, theta) {
  eta = x %*% theta
  h = exp(eta)
  p = h/(1 + h)

  # change p to 0/1
  for (i in 1:length(p)) {
  if (p[i] > 0.5) {
    p[i] = 1

  } else {
    p[i] = 0
  }
}
  return(p)
}
```

2. Data manipulation

```
data = read_csv("breast-cancer.csv") %>%
  janitor::clean_names() %>% select(-x33)
```

```r
# random data
set.seed(1)
randOrder = sample(1:nrow(data))
data_ramdom = data[randOrder,]

# use all data to get designed marix
x_all = data_ramdom %>%
  #select(radius_mean:symmetry_mean, radius_se, perimeter_se, symmetry_se) %>%
  select(-id, -diagnosis) %>%
  mutate(intercept = 1) %>% select(intercept, everything()) # add intercept vector 1s
x_all = as.matrix(x_all)

# normalization data
x_all = normal(x_all)

# recode y as 0,1
y_all = data_ramdom %>% mutate(diagnosis = recode(diagnosis, "B" = "0", "M" = "1")) %>%
  mutate(diagnosis = as.numeric(diagnosis)) %>% select(diagnosis)
y_all = as.matrix(y_all)

data_all = cbind(y_all, x_all)

# select main predictors
x_selected = data_ramdom %>%
  select(radius_mean:symmetry_mean, radius_se, perimeter_se, symmetry_se) %>%
  mutate(intercept = 1) %>% select(intercept, everything()) # add intercept vector 1s
x_selected = as.matrix(x_selected)

# normalization data
x_selected = normal(x_selected)

# recode y as 0,1
y_selected = data_ramdom %>% mutate(diagnosis = recode(diagnosis, "B" = "0", "M" = "1")) %>%
  mutate(diagnosis = as.numeric(diagnosis)) %>% select(diagnosis)
y_selected = as.matrix(y_selected)

data_selected = cbind(y_selected,x_selected)
```

3. Call functions and solving problems

Problem 1+2

```r
tol = 0.001

# Newton
theta = improved_NR(x_selected, y_selected, loglikelihood, gradlikelihood, hessianlog, tol)
result = predict(x_selected, theta)

1 - sum(abs(y_selected - result))/nrow(x_selected)
```

Problem 3

```
pathway = change_lambda(x_selected, y_selected, LogLassoLoss, tol = 0.001, 30) %>%
  mutate(log_lambda = log(lambda.i.)) %>% select(-lambda.i.)

pathway_colnames = c(colnames(x_selected),"log_lambda")
colnames(pathway) = pathway_colnames

pathway =
  pathway %>%
  pivot_longer(
  1:13,
  names_to = "beta",
  values_to = "value")

# trace plot
ggplot(pathway, aes(x = log_lambda, y = value)) +
  #geom_point(aes(color = beta)) +
  #geom_point(aes(color = beta)) +
  geom_line(aes(color = beta)) +
  theme_bw() +
  labs(title = "Trace Plot of the Lasso-logistic Fit (with 13 predictors)") +
  theme(plot.title = element_text(hjust = .5))


# # In detail
# ggplot(pathway, aes(x = lambda.i., y = value)) +
#   #geom_point(aes(color = beta)) +
#   geom_line(aes(color = beta)) +
#   xlim(0, 0.15)
```

Problem 4

```
# CV for selecting best lambda

#best_lambda = CV_lambda(k = 5, data_all, predict, LogLassoLoss, tol, n = 1000)
#best_lambda
#best lambda = 0.074

CoorWiseLasso(LogLassoLoss,x = x_all ,y = y_all, lambda = 0.074)# best lambda is 0.074
CoorWiseLasso(LogLassoLoss,x = x_selected ,y = y_selected, lambda = 0.074)

library(glmnet)
fit.glmnet = cv.glmnet(scale(x_all[,-1], scale = T), y = y_all, family = "binomial", alpha = 1, standard
fit.glmnet$lambda.min
fit.lasso = glmnet(scale(x_all[,-1], scale = T), y = y_all, family = "binomial", alpha = 1,lambda = fit
coef(fit.lasso)


fit.glmnet = cv.glmnet(x_all[,-1], y = y_all, family = "binomial", alpha = 1, standardize = F)
fit.glmnet$lambda.min
```

```r
fit.lasso = glmnet(x_all[,-1], y = y_all, family = "binomial", alpha = 1,lambda = fit.glmnet$lambda.min
coef(fit.lasso)
```

```r
# CV for comparing two models

k = seq(from = 2, to = 50)

res_nt = rep(0,length(k))
res_lasso = rep(0,length(k))
for (i in 1:length(k)) {
  res_nt[i] = mean(CV_model(k = k[i], data_selected, predict, tol, Newton = RUE, #lambda = 1))
  res_lasso[i] = mean(CV_model(k = k[i], data_all, predict, tol, Newton = ALSE, #lambda = best_lambda))
}

# res_lasso
# res_nt

diff = res_lasso - res_nt
diff_k = data.frame(k,diff)
ggplot(diff_k) + geom_line(aes(x = k, y = diff)) + ggtitle("Lasso minus ewton #Raphson")

cv_res = data.frame(k, res_lasso, res_nt) %>% pivot_longer(
  res_lasso:res_nt,
  names_to = "method",
  values_to = "rate"
)

ggplot(cv_res) + geom_line(aes(x = k, y = rate, color = method)) + ggtitle("correct #classification rat
```

4. Code for the results and conclusion

The following chunks are for the results. Functions are somewhat modified.

```r
## CV for select best lambda(n fold cv)
CV_lambda = function(k, data, pred_fun, lossfunc, tol, n) {
  result = c()

  y = data[,1]
  x = data[,2:ncol(data)]

  beta = as.matrix(rep(1, ncol(x)))
  lambda_max = max(CoorWiseLasso(lossfunc, lambda = 1, x, y, tol))
  lambda = seq(from = 15, to = 0, length.out = n)

  res_cv = c(rep(0,n))
  for (lamda_index in 1:n) {
    lambda_inuse = lambda[lamda_index]
    ans = c(rep(0,k))

    ## split the data into k folds
    cvSplits = list()
```

```
    for (i in 0:(k - 2)) {
      size = as.integer(nrow(data)/k)
      cvSplits[[i + 1]] = data[(1 + i*size):(size + i*size), ]
    }
    last_size = nrow(data) - size*(k - 1)
    cvSplits[[k]] = data[(nrow(data) - last_size + 1):(nrow(data)), ]

    ## seperate train and test data
    for (i in 1:k) {
      test = data.frame()
      train = data.frame()
      # test data
      test = cvSplits[[i]]
      test_x = test[, 2:ncol(test)]
      test_y = test[, 1]
      # train data
      for (j in 1:k) {
        if (j != i) {
          train = rbind(train, cvSplits[[j]])
        }
      }
      train_x = train[, 2:ncol(train)]
      train_y = train[, 1]
      train_x = as.matrix(train_x)
      train_y = as.matrix(train_y)
      test_x = as.matrix(test_x)
      test_y = as.matrix(test_y)

      theta_inuse = CoorWiseLasso(lossfunc, lambda_inuse, train_x, train_y, tol)
      pred_y = pred_fun(test_x, theta_inuse)
      ans[i] = 1 - sum(abs(test_y - pred_y))/nrow(test_x)
    }
    res_cv[lamda_index] = mean(ans)
  }
  #print(res_cv)
  max_index = which.max(res_cv)
  best_lambda = lambda[max_index]
  return(data.frame(acc = res_cv,
                    lambda = lambda))
}


# For plotting the cross-validation process of lambdas
acc_lambda = CV_lambda(k = 5, data_all, predict, LogLassoLoss, tol, n = 1000)
acc_lambda %>%
  ggplot(aes(x = lambda, y = acc)) +
  geom_point(size = 0.2) +
  geom_line(size = 0.4) +
  theme_bw() +
  labs(title = "Accuracy v.s. Lambda under 5 fold cross-validation") +
  theme(plot.title = element_text(hjust =.5))
```

The following chunks are for the results.

```
mean(CV_model(k = 10, data_selected, predict, tol, Newton = TRUE, lambda = 1))
mean(CV_model(k = 10, data_all, predict, tol, Newton = FALSE, lambda = 0.074))
mean(CV_model(k = 10, data_selected, predict, tol, Newton = FALSE, lambda = 0.074))
```