Project 2: Breast Cancer Prediction Model

Group 7: Melanie Mayer, Yuqi Miao, Sibei Liu, Xue Jin

March 31, 2020

Introduction

Breast cancer is the most common invasive cancer and the second leading cause of death from cancer in women worldwide, marked by the uncontrolled growth of breast cells. Non-cancerous breast tumors do not metastasize and are usually not life-threatening, while malignant tumors are cancerous, aggressive and deadly. Therefore, it's important to have breast lumps accurately diagnosed so that decision with regard to medical treatment, rehabilitation and personal matters can be made appropriately.

Objectives

The main objective of this project is to build an accurate predictive model based on logistic regression that classifies between malignant and benign images of breast tissue. Using the Breast Cancer Diagnosis dataset, logistics model and logistic-LASSO model will be implemented to predict the diagnosis. A Newton-Raphson algorithm and Pathwise Coordinate optimization will be developed to estimate the logistic model and the lasso model respectively. We aim to find the model with the best performance in terms of predicting when breast tissue is malignant.

Breast Cancer Diagnosis Dataset

The Breast Cancer Diagnosis dataset, with 569 observations, contains the diagnosis and a set of 10 features capturing the characteristics of the cell nuclei present in the digitized image of breast mass. The ten features collected for each cell nucleus are:

- 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 (perimeterˆ2 / 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)

The mean, standard deviation (SD) and largest values of these features are computed for each image, resulting in 30 possible predictive variables. We will analyze the predictive ability for diagnosis of malignant or benign cases of these covariates. The mean of each feature will be included in our models and presented in the result section. We will also create models with all 30 predictors however, these results will be reported in the supplement section.

Methods

Model Parameters

For logistic regression we assume the response variable Y_i for the i_{th} observation follows a binary distribution:

$$Y_i \sim Bin(\pi_i)$$

where π_i denotes the probablity that the i_{th} observation's tissue is malignant. We can assume all observations are independent from one another, hence the likelihood function for the vector $\boldsymbol{\pi}$ can be written as:

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

For logistic regression, the logit link function is used:

$$\log(\frac{\pi_i}{1-\pi_i}) = \boldsymbol{\beta}^T \boldsymbol{x}_i = \theta_i$$

where $\boldsymbol{x}_i^T = \begin{bmatrix} 1 & x_{1i} & x_{2i} & ... & x_{pi} \end{bmatrix}$ and $\boldsymbol{\beta}^T = \begin{bmatrix} \beta_0 & \beta_1 & \beta_2 & ... & \beta_p \end{bmatrix}$. One can solve for $\pi_i = \frac{e^{\theta_i}}{1+e^{\theta_i}}$. We aim to find the best estimate of the vector of coefficients, $\boldsymbol{\beta}$. The log-likelihood function for this vector can be written as:

$$l(\boldsymbol{\theta}) = logL(\boldsymbol{\pi}) = \sum_{i=1}^{n}(Y_i log\frac{\pi_i}{1-\pi_i} + log(1-\pi_i)) = \sum_{i=1}^{n}(Y_i\theta_i - log(1+e^{\theta_i}))$$

The maximum likelihood is thus achieved when the gradient is equal to zero and the Hessian is negative definite. The gradient can be found to be:

$$\nabla l(\boldsymbol{\theta}|\boldsymbol{X}) = \sum_{i=1}^{n}(Y_i - \pi_i)\boldsymbol{x}_i = \boldsymbol{X}^T(\boldsymbol{Y} - \boldsymbol{\pi})$$

The Hessian matrix is thus:

$$\nabla^2 l(\boldsymbol{\theta}|\boldsymbol{X}) = -\sum_{i=1}^{n} \pi_i(1-\pi_i)\boldsymbol{x}_i\boldsymbol{x}_i^T = -\boldsymbol{X}^T diag(\pi_i(1-\pi_i))\boldsymbol{X}$$

Full model

In order to estimate $\boldsymbol{\beta}$ we need to maximize the loglikelihood function. There is no closed form, hence we turn to numerical methods. The Newton-Raphson method is used to fit the full logistic model. To find the maximum likelihood estimate of each element of $\boldsymbol{\beta}$, an iterative process is set as follows:

$$\theta_{i+1} = \theta_i - \delta(\nabla^2 l(\theta_i|\boldsymbol{X}) - \gamma I)^{-1}\nabla l(\theta_i|\boldsymbol{X})$$

This is the Newton-Raphson method with two modifications. δ is included in the process to accomplish the step-halving modification, the step coefficient ensures the likelihood is always increasing in order to achieve quicker convergence. Once the likelihood approaches convergence the steps become smaller until convergence is reached. γ is the modification coefficient to ensure the aescent direction of the iteration vector at θ_i.

Logit-lasso Pathwise Coordinate-wise Update Algorithm

In the case of large dimensionality or multicollinearity of predictors it can be beneficial to perform a regularization method which shrinks coefficients and can perform variable selection. Here we implement the Least Absolute Shrinkage and Selection Operator (LASSO) method for logistic regression with a path-wise coordinate-wise optimization algorithm to select variables from the full model, increase the prediction efficiency and avoid overfitting. With the ten predictor model, LASSO may not be as beneficial compared to the 30 predictor model where we have 30 predictors describing 10 features hence are likely to experience multicollinearity and expect LASSO to outperform the classical logistic regression.

In LASSO we add an L1 penalization term to the loss function, such that we try to find the coefficients to minimize:

$$min_{(\boldsymbol{\beta})}\{-l(\boldsymbol{\beta}) + \lambda \sum_{j=0}^{p} |\beta_j|\}$$

With the pathwise coordinate-wise update algorithm, we find the likelihood to be:

$$l(\boldsymbol{\beta}) = -\frac{1}{2n}\sum_{i=1}^{n} \omega_i (z_i - \boldsymbol{X_i \beta})^2$$

where,

$$\pi_i = \frac{exp(\boldsymbol{X_i\beta})}{1 + exp(\boldsymbol{X_i\beta})}$$

$$\omega_i = \pi_i(1-\pi_i)$$

$$z_i = \boldsymbol{X_i\beta} + \frac{y_i - \pi_i}{\pi_i(1-\pi_i)}$$

One must pre-define the tuning parameter sequence $\{\lambda_1, ..., \lambda_s\}$ and a starting vector $\boldsymbol{\beta_{start=}}\{\beta_0^{(0)}, ..., \beta_p^{(0)}\}$. When choosing the sequence of lambdas, it is best to define $max(\boldsymbol{\lambda}) = max(\boldsymbol{\beta})$, where $\boldsymbol{\beta}$ refers to the estimated coefficients from the logistic regression, such that all coefficients will shrink to zero. The optimal process for each λ_u is then found by using the optimal $\boldsymbol{\beta_{u-1}}$ from the previous iteration as a warm start in order to reach the optimal values quicker. Within every iteration for each lambda, the optimal $\boldsymbol{\beta}$ is searched using coordinate-wise updating, that is it is minimized over one parameter at a time while keeping all others fixed.

For β_j in t_{th} iteration

$$\beta_j^{(t)} = \begin{cases} \frac{\sum_{i=1}^{n} \omega_i (z_i - \sum_{j=1}^{p} \boldsymbol{X_i}\beta_j)}{\sum_i^n \omega_i}, & j = 0 \\ \frac{s(\beta_j^{(t*)}, \lambda_u n)}{\sum_{i=1}^{n} \omega_i x_{ij}^2}, & j = 1, 2, ..., p \end{cases}$$

$$\beta_j^{(t*)} = \sum_{i=1}^{n} \omega_i x_{ij} z_{ij}^*$$

$$z_{ij}^* = z_i - \sum_{\substack{k=0 \\ \beta_k \neq 0}}^{j-1} \beta_k^{(i)} x_{ik} - \sum_{\substack{k=j+1 \\ \beta_k \neq 0}}^{p} \beta_k^{(i-1)} x_{ik}$$

Cross Validation

In order to find the optimal lambda, we use 5-fold cross validation. The dataset is divided into five sub-datasets. The optimal coefficients is then found five times by running the logit-LASSO on a combination of four of the five subsets, 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. The statistics we use to compare predictive ability is SSE and pearson chi-square statistic. SSE is defined as:

$$SSE = \sum_{i=1}^{n}(y_i - \widehat{\pi}_i)^2$$

where $\widehat{\pi}_i = log\frac{exp(\boldsymbol{X}_i\beta)}{1+exp(\boldsymbol{X}_i\beta)}$. Pearson chi-square statistic is defined as:

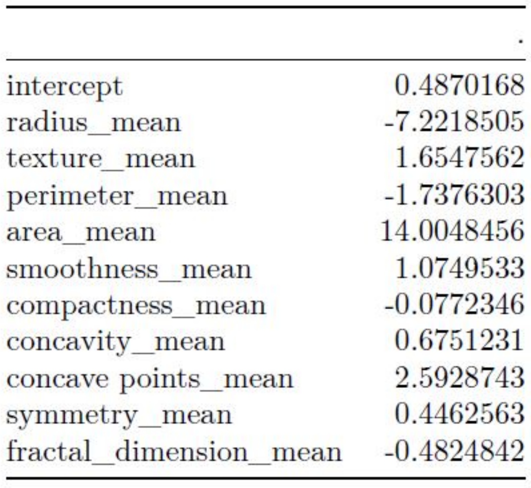
$$G = \sum_{i=1}^{n}\frac{y_i - \widehat{\pi}_i}{\widehat{\pi}_i(1-\widehat{\pi}_i)}$$

We want both of these statistics to be minimized. By taking the average of the above statistics over the value found for each of the five folds, we get a value to evaluate the model fit for each lambda.

Results

Logistic Regression Model

The results from the logistic model estimated using the modified Newton-Raphson algorithm with step-halving and gradient ascent can be found in Table 1. Here we are showing the results when only using the means of the ten features as predictors. We found average area had the largest coefficient. For a one unit increase in the average area of the tissue, we expect to see a change in the log odds of having a malignant tutor of 14, holding all other predictors constant. The predictors were all standardized so caution must be used when analyzing these coefficients however, this seems to be very high however. The smallest coefficient, -7.22, was estimated for average radius. The results from logistic regression using the mean, SD and largest values of the 10 features (30 predictors) are shown in Supplementary Table 1.



	.
intercept	0.4870168
radius_mean	-7.2218505
texture_mean	1.6547562
perimeter_mean	-1.7376303
area_mean	14.0048456
smoothness_mean	1.0749533
compactness_mean	-0.0772346
concavity_mean	0.6751231
concave points_mean	2.5928743
symmetry_mean	0.4462563
fractal_dimension_mean	-0.4824842

Table 1. Estimated coefficients of logistic regression with 10 mean features

Logistic-LASSO Model

The sequence of λ we compared ranged from three to zero, with a length of 100. The pre-defined values for all β coefficients including the intercept was 0.02. The optimal λ was selected based on the minimum SSE, maxmimum AUC and minimum g-statistics estimated on the test dataset. The results from each of the five folds can be found in Table 2.

	Enter	Fold1	Fold2	Fold3	Fold4	Fold5
k	0.00	1.0000000	2.0000000	3.0000000	4.0000000	5.0000000
best_lambda	0.00	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
beta_vec1	0.02	-0.5851898	-0.5921021	-0.6782398	-0.6305423	-0.5250743
beta_vec2	0.02	1.9370600	1.8598428	1.8706748	1.0162875	1.7805325
beta_vec3	0.02	0.8620696	0.9438123	0.9124855	0.8276538	0.9191474
beta_vec4	0.02	-0.0027197	0.0110750	0.0517034	1.0505263	-0.0067178
beta_vec5	0.02	-0.0045932	-0.0077140	-0.0160055	-0.0010837	-0.0105529
beta_vec6	0.02	0.4154545	0.4039963	0.3839284	0.3662432	0.5432791
beta_vec7	0.02	-0.0406285	0.0146759	-0.0267084	0.0795153	-0.0452543
beta_vec8	0.02	0.1820403	0.1956254	0.3122696	0.2686984	0.1299537
beta_vec9	0.02	2.0666740	1.9595822	1.9458032	1.8233049	2.3142211
beta_vec10	0.02	0.0725989	0.1110710	0.1140780	0.1066402	0.1031025
beta_vec11	0.02	-0.1413081	-0.1589079	-0.1699873	-0.1369779	-0.1387135
g.stat_tr	Inf	137.3082360	135.9975379	116.9462145	135.4900541	114.5090024
auc_te	0.00	0.9913435	0.9923154	0.9849530	0.9826870	0.9743770
g.stat_te	Inf	22.6159703	22.1514224	48.1975784	34.5067570	48.0076306
SSE_test	Inf	4.2783563	4.1138120	4.9735803	5.9718400	6.8409613

Table 2. 5-fold cross validation coefficients of logistic-LASSO model and optimal lambda

We find the results are consistant across all five folds based on all three statistics. The optimal λ appears to be zero. This implies the L1 penalization term added to the loss function is not effecting our model. Using the test data of each fold, the AUC ranges from 0.99 to 0.97, the g-statistic ranges from 22.6 to 48.0, while SSE ranges from 4.1 to 6.8.

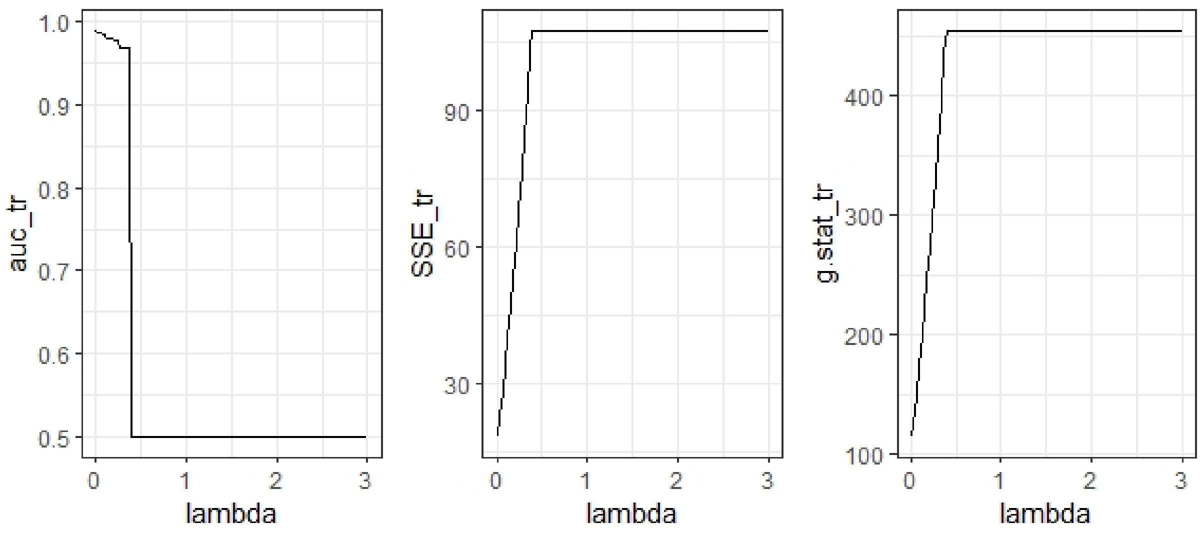


Fig 1. Predictive ability criteria over range of λ

Figure 1 shows the trend in AUC, SSE and g-statistic over the range of lambdas tested. With the increase of λ, both SSE and g-statistics have a dramatic increase and become constant around 0.4. AUC decreases drastically from 1 to 0.4. All of them indicate that the bigger the λ is, the worse the model performs.

Since the optimal $\lambda = 0$, none of the coefficients of the logistic-lasso is shrunk to zero, as seen in Table 3. The coefficient terms are much smaller compared to those estimated by the classic logistic regression however.

beta_0	beta_1	beta_2	beta_3	beta_4	beta_5	beta_6	beta_7	beta_8	beta_9	beta_10
-0.53551	2.1002	0.87201	0.08757	-0.0195	0.81993	0.0722	0.41069	1.27974	0.07145	-0.15123

Table 3. Coefficients of final logistic-LASSO model

The results when all 30 predictors are considered, and their corresponding Cross Validation results in each fold, are shown in Supplementary Table 2. The optimal λ under that scenario is 0.03, which shrinks about 20 coefficients to 0, as presented in Supplementary Table 3.

Discussion

The full logistic model using the ten predictors resulted in an AUC of 0.9879 while the LASSO-Logistic model using the same predictors resulted in an AUC of 0.99, thus as expected the logistic-LASSO regression is a more optimal model between these two options. The difference is very small however and if one is interested in interpreting the coefficients the full logistic model may still be preferred. When attempting to estimate a binary outcome, there are many other models which could have been examined as well. We could have compared our results to other generalized linear models using a probit or complementary log-log link function. It could be of interest to perform these models in addition to the logistic link.

The logistic-LASSO model appears to perform very well on this data, achieving a very high AUC (found in Table 2). This is very important because it implies a high sensitivity and specificity, thus often properly diagnosing breast tissue. This is most likely a result of our predictors being highly associated with whether or not tissue is malignant. When all 30 predictors are included in our model, the AUC only increases marginally (found in Supplementary Table 2). However when LASSO is performed, many predictors based on mean are shrunk to zero and instead predictors based on largest (worst) value of the features remain in the model. This could be due to largest value being better predictors but could also be due to the predictors being highly correlated with one another and thus LASSO is picking largest value to remain in the model at random. This is a potential limitation of LASSO regression.

Supplement

	coef
cur.ones	10.9788932
cur.radius_mean	-156.8293242
cur.texture_mean	2.4425562
cur.perimeter_mean	14.5634941
cur.area_mean	131.2208078
cur.smoothness_mean	8.6614671
cur.compactness_mean	-23.4558454
cur.concavity_mean	14.5243749
cur.concave.points_mean	11.9608850
cur.symmetry_mean	-4.3787895
cur.fractal_dimension_mean	1.9332019
cur.radius_se	-1.0039330
cur.texture_se	-3.2262845
cur.perimeter_se	-15.6575530
cur.area_se	51.9843423
cur.smoothness_se	-0.3215738
cur.compactness_se	14.6159803
cur.concavity_se	-20.2182318
cur.concave.points_se	19.2175444
cur.symmetry_se	-5.9974445
cur.fractal_dimension_se	-23.1327888
cur.radius_worst	73.9636466
cur.texture_worst	7.1661910
cur.perimeter_worst	19.0033108
cur.area_worst	-62.7328247
cur.smoothness_worst	-4.6440407
cur.compactness_worst	-11.0534416
cur.concavity_worst	16.8479478
cur.concave.points_worst	-2.8942002
cur.symmetry_worst	9.6950239
cur.fractal_dimension_worst	14.7267503
step	0.0000000

Supplementary Table 1: Estimated coefficients of 30 features under Newton-Raphson method

	Enter	Fold1	Fold2	Fold3	Fold4	Fold5
k	0	1.0000000	2.0000000	3.0000000	4.0000000	5.0000000
best_lambda	0	0.0000000	0.0000000	0.0000000	0.0000000	0.0303030
beta_vec1	1	-0.6663464	-0.8606693	-0.6244876	-0.5355721	-0.7737072
beta_vec2	1	1.4446157	1.7172558	2.1313547	2.2414343	0.2165391
beta_vec3	1	1.1141379	1.0516070	1.0438591	1.0085630	0.3121205
beta_vec4	1	0.1525774	0.0710993	0.3134996	0.1776248	0.0000000
beta_vec5	1	0.3582305	0.2507639	-0.0270073	0.1110899	0.0000000
beta_vec6	1	0.7406111	0.6795368	1.0789549	1.0204367	0.0000000
beta_vec7	1	-0.7592081	-0.5953038	-0.0628564	-0.3732903	0.0000000
beta_vec8	1	0.9333178	0.7112050	1.0380966	0.3601083	0.0000000
beta_vec9	1	1.1557369	1.4566499	0.2751937	1.9059881	0.8541841
beta_vec10	1	0.0243545	0.3555338	0.0343354	0.1857253	0.0000000
beta_vec11	1	-0.2790475	-0.4709215	-0.5075943	-0.5182740	0.0000000
beta_vec12	1	1.4844467	1.1900859	1.1684542	1.1049459	0.0000000
beta_vec13	1	-0.1769061	-0.0938746	-0.0395756	-0.1096598	0.0000000
beta_vec14	1	1.3698537	1.2170041	1.0613571	0.9619370	0.0000000
beta_vec15	1	1.6500516	1.3190013	1.2700602	1.1353677	0.0000000
beta_vec16	1	-0.1667238	-0.1415278	-0.3170986	-0.4389349	0.0000000
beta_vec17	1	-0.5206480	-0.3403789	-0.3967853	-0.3717366	0.0000000
beta_vec18	1	-0.3255053	-0.1431664	-0.2770359	-0.1385741	0.0000000
beta_vec19	1	-0.4716457	-0.2749817	-0.3331025	-0.2050350	0.0000000
beta_vec20	1	-0.6296226	-0.3529685	-0.3045117	-0.3653631	0.0000000
beta_vec21	1	-0.4037133	-0.2866930	-0.3612959	-0.3119301	0.0000000
beta_vec22	1	4.0435322	2.7400581	3.7517734	2.3336327	2.0672393
beta_vec23	1	0.8868274	1.1107406	1.2166616	1.0809459	0.7069871
beta_vec24	1	4.4380960	2.9678262	3.7811780	2.3000294	1.8912901
beta_vec25	1	2.4154698	1.4236061	2.0741866	1.1815997	0.3085923
beta_vec26	1	1.0240182	0.9518878	0.6785581	0.6943120	0.1512226
beta_vec27	1	0.4841746	0.6177355	0.7418958	0.6275855	0.0000000
beta_vec28	1	0.9079662	0.7588550	1.1239071	0.7332618	0.1695023
beta_vec29	1	2.9778771	3.5427933	4.0128312	3.8603744	2.4497217
beta_vec30	1	1.0674250	1.2512243	0.9738290	0.9026693	0.4915037
beta_vec31	1	0.4788639	0.5340397	0.6037901	0.4549507	0.0000000
g.stat_tr	Inf	NaN	NaN	NaN	NaN	59.6412968
auc_te	0	0.9986486	0.9993243	0.9983897	0.9920077	0.9705285
g.stat_te	Inf	NaN	NaN	NaN	NaN	1868.0016983
SSE_test	Inf	2.5853338	2.0644662	3.2451899	4.0137240	3.8903373

Supplementary Table 2: Estimated coefficients of 30 features in each fold of Cross Validation

	coef
beta_0	-0.7737072
beta_1	0.2165391
beta_2	0.3121205
beta_3	0.0000000
beta_4	0.0000000
beta_5	0.0000000
beta_6	0.0000000
beta_7	0.0000000
beta_8	0.8541841
beta_9	0.0000000
beta_10	0.0000000
beta_11	0.0000000
beta_12	0.0000000
beta_13	0.0000000
beta_14	0.0000000
beta_15	0.0000000
beta_16	0.0000000
beta_17	0.0000000
beta_18	0.0000000
beta_19	0.0000000
beta_20	0.0000000
beta_21	2.0672393
beta_22	0.7069871
beta_23	1.8912901
beta_24	0.3085923
beta_25	0.1512226
beta_26	0.0000000
beta_27	0.1695023
beta_28	2.4497217
beta_29	0.4915037
beta_30	0.0000000

Supplementary Table 3: Estimated coefficients of 30 features in final model in Cross Validation

- beta_0 refers to the intercept
- beta_1 - beta_10 refers to feature means
- beta_11 - beta_20 refer to feature standard error
- beta_21 - beta_30 refer to the largest value of the feature

Code

```
knitr::opts_chunk$set(echo = TRUE,message = FALSE,warning = FALSE)

require(survival)
require(quantreg)
require(glmnet)
require(MASS)
require(pROC)
```

```r
library(tidyverse)
library(caret)
library(patchwork)
```

Read in Data and Manipulation:

```r
cancer = read_csv("breast-cancer.csv") %>%
  mutate(diagnosis = as.numeric(factor(diagnosis, levels = c("B","M"), labels = c(0,1)))-1) %>%
  mutate(radius_mean = scale(radius_mean))

predictor_scale = as_tibble(scale(cancer[3:12]))
cancer_package = cancer %>% select(contains("mean"), diagnosis)
cancer_scale = cbind(rep(1,569), predictor_scale, cancer$diagnosis)
names(cancer_scale) = c("ones", names(cancer)[3:12], "response")


predictor_scale2 = as_tibble(scale(cancer[3:32]))
cancer_scale2 = cbind(rep(1,569), predictor_scale2, cancer$diagnosis)
names(cancer_scale2) = c("ones", names(cancer)[3:32], "response")

## standardize data!!
```

Find likelihood, gradient, and Hessian matrix for logistic model:

$$\log(\frac{\pi_i}{1-\pi_i}) = \mathbf{x}_i\boldsymbol{\beta}$$

```r
hessian_gradient_log = function(data, beta_vec){
  y = as.matrix(data %>% select(last_col()))
  x = as.matrix(data[, 1:dim(data)[2]-1])
  theta = x%*%beta_vec
  pi = exp(theta)/(1 + exp(theta))
  loglikelihood = sum(y*theta - log(1 + exp(theta)))
  gradient = t(x)%*%(y - pi)
  pi_matrix = matrix(0, nrow = dim(data)[1],ncol = dim(data)[1])
  diag(pi_matrix) = pi*(1 - pi)
  hessian = -t(x)%*%pi_matrix%*%(x)
  return(list(loglikelihood = loglikelihood, gradient = gradient, hessian = hessian))
}

a = hessian_gradient_log(cancer_scale, rep(0.02,11)) #Example of function
a = hessian_gradient_log(cancer_scale2, rep(0.02,31)) #Example of function
```

Function for Newton-Raphson algorithm:

```r
NR = function(data, beta_start, tol = 1e-10, max = 200){
  i = 0
  cur = beta_start
  stuff = hessian_gradient_log(data, cur)
  curlog = stuff$loglikelihood
  res = c(i = 0,curlog = curlog,cur = cur, step = 1)
  prevlog = -Inf
  while((i <= max) && (abs(curlog - prevlog) > tol)){
    step = 1
    i = i+1
    prevlog = stuff$loglikelihood

    #Check ascent directions
    eigen = eigen(stuff$hessian)
    if(max(eigen$values) <= 0){ #Check if negative definite
      hessian = stuff$hessian
    }
    else{ #Create a similar negative definite matrix
      hessian = stuff$hessian - (max(eigen$values) + 0.1)*diag(dim(stuff$hessian)[1])
    }
    prev = cur
    cur = prev - rep(step,length(prev))*(solve(hessian)%*%stuff$gradient)
    stuff = hessian_gradient_log(data,cur)
    curlog = stuff$loglikelihood

    while(curlog < prevlog){
      stuff = hessian_gradient_log(data, prev)
      step = step/2 #For step halving
      cur = prev - rep(step, length(prev))*(solve(hessian)%*%stuff$gradient)
      stuff = hessian_gradient_log(data,cur)
      curlog = stuff$loglikelihood
    }
    names(cur) = names(data)[-dim(data)[2]]
    res = rbind(res, c(i=i, curlog = curlog, cur = cur,step = step))
  }
  return(res)
}

#run NR algorithm, only mean values as predictors
beta_start = rep(0.02,11)
names(beta_start) = names(cancer_scale)[-12]

NR_result = NR(cancer_scale, beta_start)
#extract final coefficients from NR
NR_coeff =  NR_result[dim(NR_result)[1], c(-1,-2, -dim(NR_result)[2])]

#compare results to glm package logistic regression:
cancer_fit = glm(response ~ ., data = cancer_scale[, -1], family = binomial(link = "logit"))
round(as.vector(cancer_fit$coefficients), 3) == round(as.vector(NR_coeff), 3)
# same results as NR

roc(cancer_scale$response ~ predict(cancer_fit, type = "response"))

#run NR algorithm, all measures as predictors
beta_start2 = rep(0.001, 31)
names(beta_start2) = names(cancer_scale2)[-32]

#run NR algorithm
NR_result2 = NR(cancer_scale2, beta_start2)
#extract final coefficients from NR
```

```
NR_coeff2 =  NR_result2[dim(NR_result2)[1], c(-1,-2, -dim(NR_result2)[2])]

round(as.vector(NR_coeff2), 3)
```

Function to find AUC:

```
auc = function(yi,pi_hat){
  auc = c()
  for (i in seq(dim(pi_hat)[2])){
    c = tibble(pi_hat = pi_hat[,i],yi = yi)
    m = sum(yi==1)
    n = length(yi)-m
    c =
      c %>%
      arrange(pi_hat) %>%
      mutate(order = seq_along(pi_hat)) %>%
      group_by(pi_hat) %>%
      mutate(mean_order = mean(order))
    pos_order = c %>% filter(yi == 1) %>% pull(mean_order)
    auc = c(auc,(sum(pos_order)-m*(m+1)/2)/(m*n))
  }
  return(auc)
}

yi = c(0,1,0,1,0,0,0,1)
pi = matrix(rep(rep(c(0.2,0.000001,0.3,0.4),2),2),ncol = 2 )
auc(yi,pi)
```

Build a logistic LASSO model with k-fold cross validation:

```
set.seed(2019)
#lambda_max = max(abs(NR_coeff)) ## warm start
tuning_grid = seq(3, -1, length = 100) ## tuning seq
## function of lasso coordinate descent algorithm
# input:
# data: y -- binary response;
#       x -- scaled predictors
# k: # fold for CV
# beta_vec: initial beta guess
# tune_grid: seq of lambda used in variable selection
# tol: tolerance for stop iteration
# max: max iteration times

# output: res--result list containing:
# beta -- tibble:
#     rows: lambdas
#     columns:
#     beta1-beta10: beta estimation for every lambda
#     targ: target function value for every lambda iteration
#     times: times of iteration in each lambda iteration
# coeff -- list of final coefficient
# best_tune -- best tuning parameter lambda
# SSE_te -- test SSE results among cv
```

```r
lasso_co_des = function(data, beta_vec,  k = 5, tune_grid, tol = 1e-10, max = 200){
  ## create fold
  folds = createFolds(data$response,k = k, returnTrain = T)
  cv_result = c(k = NULL, best_lambda = NULL, beta_vec = beta_vec,
                g.stat_tr = Inf, auc_te = 0,  g.stat_te = Inf, SSE_test = Inf)
  ## lasso coord des for train data
  for (i in 1:k) {
    tr_rows = folds[[i]]
    train = data[tr_rows, ]
    test = data[-tr_rows, ]
    n_tr = dim(train)[1]
    x_tr = as.matrix(train[ , 1:dim(train)[2] - 1])
    y_tr = as.matrix(train[ , dim(train)[2]])
    x_te = as.matrix(test[ , 1:dim(test)[2] - 1])
    y_te = as.matrix(test[, dim(test)[2]])
    res = NULL


    ## for performance measure for every lambda
    for (lambda in sort(tune_grid, decreasing = T)) {
      theta_vec = x_tr %*% beta_vec
      pi_vec = exp(theta_vec)/(1 + exp(theta_vec))
      w_vec = pi_vec*(1 - pi_vec) + rep(1e-10, length(pi_vec))
      z_vec = theta_vec + (y_tr - pi_vec)/w_vec
      w_res_vec = sqrt(w_vec)*(z_vec - theta_vec)
      cur_target = (1/2*(n_tr))*t((w_res_vec))%*%(w_res_vec) + lambda*sum(abs(beta_vec[-1]))
      pre_target = -Inf
      time = 0
      names(beta_vec) = paste("beta_",0:(dim(data)[2]-2), sep = "")
      res = rbind(res, c(k = k, lambda = lambda, time = time, cur_target = cur_target, beta_vec))
      while((abs(pre_target - cur_target) > tol) && time < max && cur_target > pre_target ){
        time = time + 1
        zero_index = seq(1, length(beta_start))
        pre_target = cur_target
        beta_pre = beta_vec
         for (j in 1:dim(train)[2] - 1) {
           x_tr_s = x_tr[1:dim(x_tr)[1], zero_index]
           dim(x_tr_s) = c(dim(x_tr)[1], length(zero_index))
           beta_pre_s = beta_pre[zero_index] ## sparse updating
           w_z_vec_j = as.vector(sqrt(w_vec)*(z_vec - x_tr_s%*%beta_pre_s + as.vector(x_tr[,j]*beta_pre
           #w_z_vec_j = w_z_vec_j[!is.nan(w_z_vec_j)]
           # dim(455,1),working response with out jth predictor
           w_x_tr_j = as.vector(as.vector(sqrt(w_vec)) * x_tr[,j])
           #w_x_tr_j = w_x_tr_j[!is.nan(w_x_tr_j)]
           ## dim(455,1), weighted obs on jth row
           tmp = as.numeric(t(w_x_tr_j) %*% w_z_vec_j)
           lambda_n = lambda*dim(x_tr)[1]
           if(j == 1){
             beta_pre[j] = tmp/sum(w_vec)
           }

           if (j>=2) {
             if (abs(tmp)>lambda_n) {
```

```r
            if(tmp > 0) {tmp = tmp - lambda_n}
            else {tmp = tmp + lambda_n}
            }else{
            tmp = 0
            zero_index = zero_index[zero_index!=j]}
         }
         beta_pre[j] = tmp/(t(w_x_tr_j) %*% w_x_tr_j)
         }
      beta_vec = beta_pre
      theta_vec = x_tr%*%beta_vec # dim = (455,1), theta for every obs
      pi_vec = exp(theta_vec)/(1+exp(theta_vec)) # dim = (455,1), pi for every obs
      w_vec = pi_vec*(1-pi_vec)+rep(1e-10,length(pi_vec)) # dim = (455,1), weight for every obs
      z_vec = theta_vec + (y_tr - pi_vec)/w_vec
      # dim = (455,1), working response for every obs
      w_res_vec = sqrt(w_vec)*(z_vec - theta_vec)
      #w_res_vec = w_res_vec[!is.nan(w_res_vec)]
      cur_target = (1/(2*n_tr))*t((w_res_vec))%*%(w_res_vec) +lambda*sum(abs(beta_vec[-1]))
      res = rbind(res,c(k = k,lambda = lambda,time = time, cur_target = cur_target, beta_vec))
    }
  }
## choose lambda
res = as.tibble(res)
beta_lambda = res %>%
  group_by(lambda) %>%
  filter(cur_target == min(cur_target)) %>%
  dplyr::select(contains("beta"))
beta_lambda_m = as.matrix(beta_lambda[2:dim(beta_lambda)[2]]) # dim = (194, 11)
## use train dataset to choose lambda
pi_hat = exp(x_tr %*% t(beta_lambda_m))/(1+exp(x_tr %*% t(beta_lambda_m)))
residual_lambda = rep(y_tr,dim(beta_lambda_m)[1]) - pi_hat
SSE_tr = as.vector(rowSums(t(residual_lambda^2)))
## auc calculation
auc_tr = auc(y_tr,pi_hat = pi_hat)
g.res = (rep(y_tr,dim(beta_lambda_m)[1]) - pi_hat)/sqrt(pi_hat*(1-pi_hat))
g.stat_tr = as.vector(rowSums(t(g.res^2)))
result_tr = as.tibble(cbind(beta_lambda,g.stat_tr = g.stat_tr,auc_tr = auc_tr,SSE_tr = SSE_tr))
best_result =  result_tr %>%
  filter(auc_tr == max(auc_tr)) %>% filter(lambda == min(lambda))
best_result = best_result[1,]
# use test datasets to evaluate
best_result_m = as.matrix(best_result[2:(dim(best_result)[2]-3)])
pi_hat_te = exp(x_te %*% t(best_result_m))/(1+exp(x_te %*% t(best_result_m)))
residual_test = y_te - pi_hat_te
SSE_test = as.vector(rowSums(t(residual_test^2)))
auc_te = auc(y_te,pi_hat_te)
g.res_te = (y_te - pi_hat_te)/sqrt(pi_hat_te*(1-pi_hat_te))
g.stat_te = as.vector(rowSums(t(g.res_te^2)))

cv_result = rbind(cv_result, c(k = i,best_lambda = best_result$lambda, beta_vec = best_result[2:(dim(
#cv_result = as.tibble(cv_result)
}

return(list(cv_result = unnest(as.tibble(cv_result)),res = res, result_tr= result_tr , best_result = 
}

set.seed(2020)
x = lasso_co_des(data = cancer_scale, beta_vec = rep(0.02,11), k = 5, tune_grid = tuning_grid, max = 20

beta_start2 = rep(1,31)

beta_vec2 = beta_start2
x = lasso_co_des(data = cancer_scale2, beta_vec = beta_start2, k = 5,tune_grid = tuning_grid,max = 200)

knitr::kable(x$cv_result,digits = 3)

x

g1 = x$result_tr %>%
  ggplot(aes(x = lambda, y = auc_tr))+
  geom_line()+theme_bw()

g2 = x$result_tr %>%
  ggplot(aes(x = lambda, y = SSE_tr))+
  geom_line()+theme_bw()

g3 = x$result_tr %>%
  ggplot(aes(x = lambda, y = g.stat_tr))+
  geom_line()+theme_bw()


g1+g2+g3

x$res
```

```r
# cleaning the above x
library(sjmisc)
y=as.data.frame(x$cv_result)
y_y=rotate_df(y)
names(y_y)=c("Enter","Fold1","Fold2","Fold3","Fold4","Fold5")
knitr::kable(y_y)
```