Final Report

Ngoc Duong, Cui Sitong, Xinru Wang, Jin Ge

4/14/2020

Objective

Breast cancer is one of the most common cancers in women. However, early diagnoses of breast cancer can aid in reducing the mortatlity rate. Additionally, advances in imaging technologies and statistical methodologies have allowed for higher-quality data and novel models that could improve the precision of breast cancer diagnoses. The purpose of our project is to build and compare different models in classifying breast cancer tumor as benign or malignant based on image-based predictors. Specifically, we are look to build a logistic regression model using Newton-Raphson method, and a logistic-LASSO model using coordinate-wise optimization algorithm.

Dataset

There were 569 images collected independently from patients, 212 of whom had malignant tumor and 357 were benign cases. The images were broken down into 30 predictors, corresponding to the mean, standard deviation, and largest values (points on the tails) of the following 10 features: 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, and fractal dimension ("coastline approximation" -1)

Data cleaning

As shown in the pairwise correlation plot **Figure 1**, we can observe the presence of some strong multi-collinearity among the predictors. For instance, the radius_mean variable has almost perfect correlation of 1 and 0.99 with perimeter_mean and area_mean variables, respectively. We then decided to leave out variables that showed pairwise correlations of more than 85%. The final dataset contained 13 predictors. **Figure 2** shows the decreased amount of correlation between variables in this reduced dataset.

Next, considering the LASSO is not scale-invariant, we standardized the design matrix. This is to ensure comparability of estimates by the logistic-LASSO model and Newton-Raphson/logistic regression model. The standardization formula is as follows:

$$standardized(x_{ij}) = \frac{x_{ij} - \bar{x_j}}{std(x_j)}$$

for $i = 1, 2, ...30$ and $j = 1, 2, ..., 569$

Finally, we recoded the response variable such that "malignant" = 1, and "benign" = 0.

Newton-Raphson model

We used logistic regression to classify the malignancy of tissue. Malignancy corresponds to response variable being 1 ($y^{(i)} = 1$).

Log likelihood is

$$l(y; \beta) = \sum_{i=1}^{n} \{y_{(i)} log\mu_{(i)} + (1 - y_{(i)}) log(1 - \mu_{(i)})\}$$

Its gradient is given by

$$g : \bigtriangledown l(y; \beta) = \sum_{i=1}^{n} (y_{(i)} - \mu_{(i)}) x_{(i)} = X^T (y - \mu)$$

Its Hessian matrix is given by

$$H : \bigtriangledown^2 l(y; \beta) = -\sum_{i=1}^{n} \mu_{(i)} (1 - \mu_{(i)}) x_{(i)} (x_{(i)})^T = -X^T S X$$

where $S = diag(\mu_{(i)}(1 - \mu_{(i)}))$, and

$$\mu_{(i)} = p_\theta(y = 1|x) = \frac{e^{X_i \beta}}{1 + e^{X_i \beta}}$$

Since we have several predictors, we want to optimize several likehood functions simultaneously. This is equivalent to solving a system of log-likelihood equations $\bigtriangledown l(y; \beta_j) = 0$ where $j = 1, 2, ...13$. To achieve this, we used the Newton Raphson algorithm.

Newton-Raphson Algorithm

Starting at a current point β_i, we can expand the log-likelihood function around this point using Taylor's expansion, which gives a neighborhood of β_i containing β_{i+1} which increases the likelihood. The equation below can be used to iteratively update β_i until the sequence converges and $\bigtriangledown l(y; \beta_j) = 0$ is satisfied:

$$\beta_{i+1} = \beta_i - [\bigtriangledown^2 l(\beta_i)]^{-1} \bigtriangledown l(\beta_i)$$

Modifications to Newton-Raphson

When implementing Newton-Raphson, we need to check at every step, that the updating direction (for β_{i+1}) is heading to a maximum, and that the point is moving sufficient distances towards the maximum so we do not miss it. Therefore, we also implemented some modifications, specifically gradient descent and step-halving.

- For step-halving, we modified the updating function for β_{i+1} as follows:

$$\beta_{i+1} = \beta_i - \lambda [\bigtriangledown^2 l(y; \beta_i)]^{-1} \bigtriangledown l(y; \beta_i)$$

,

where $\lambda = 1$ until $l(\beta_{i+1}) \leq l(\beta_i)$, which means the new point would have gone too far. Then, we can search for a value λ such that $l(\beta_{i+1}, \lambda) \geq l(\beta_i)$. At this step, we can cut the step (λ) in half for each sub-iteration.

- For gradient descent, at every iteration, we checked whether $\bigtriangledown^2 l(y; \beta)$ is negative definite (signifying the point is moving in the right direction). If at some iteration, $\bigtriangledown^2 l(y; \beta)$ is not negative definite, we replace it with a similar negative definite matrix, such as $\bigtriangledown^2 l(y; \beta) - \gamma I$ where γ is chosen such that the resulting matrix is negative definite. Naturally, this γ must be greater than any of the elements of the diagonal matrix D obtained by eigendecomposition: $\bigtriangledown^2 l(y; \beta) = P^T D P$.

Logistic-LASSO model

The LASSO method aims to minimizes the following equation with a penalty term. This is a quardratic approximation to the negative log likelihood by Taylor expanding around the current estimate, which is:

$$f(\beta) = \frac{1}{2n}\Sigma_{i=1}^{n} w_i (z_i - \Sigma_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda \Sigma_{j=1}^{p} |\beta_j|, \lambda \geq 0$$

, where

- w_i are the working weights, defined as $\tilde{p}(x_i)(1-\tilde{p}(x_i))$, and $\tilde{p}(x_i)$ is the probability of event for each observation, and is evaluated at the current parameters,
- z_i are the working response $= \tilde{\beta}_0 + x_i^T \tilde{\beta} + \frac{y_i - \tilde{p}(x_i)}{\tilde{p}(x_i)(1-\tilde{p}(x_i))}$

Then, each β_i is optimized using the folllowing equation:

$$\tilde{\beta}_i = \frac{S(\Sigma_i w_i x_{i,j}(y_i - \tilde{y}_i^{(-j)}), \lambda)}{\Sigma_i w_i x_{i,j}^2}$$

where $S(\hat{\beta}, \gamma)$ is called soft-threshold and is defined as $S(\hat{\beta}, \lambda) = sign(\hat{\beta})(|\hat{\beta}| - \lambda)_+$

Results

The coefficient estimates can be found in **Table 1**. Newton-Raphson algorithm gives quite similar estimates to those in the logistic regression model produced by GLM package. For the logistic-LASSO model, we can see the coefficient estimates are approximately close to the ones produced by GLMnet with 5-fold cross-validation. They do not exactly match, however, potentially due to some dissimilarities in the setup conditions during our implementation and theirs.

The path of solution could be found in **Figure 3**, and the distribution of cross-validated MSEs produced by the hand-built logistic-LASSO model could be found in **Figure 4**. Five-fold cross-validation suggested the best λ is 0.0045, which corresponds to the lowest cross-validated MSE. A similar distribution of cross-validated MSEs produced by the GLMNet package in R can be found in **Figure 5**. Here, 5-fold cross-validation suggested the best λ is 0.0037.

Lastly, we wanted to compare the prediction performance using MSE as a criteria. We first compared the MSE obtained from fitting the models given the calculated coefficients on the whole dataset. The results are displayed in **Table 2**. We then examined this under the 5-fold cross-validation condition applied to all models. The distributions of 5-fold cross-validated MSE can be found in **Figure 6**. We noticed that the cross-validated MSE of both Newton-Raphson and logistic-LASSO are similarly distributed, although the latter seems to perform slightly less well. Nonetheless, the errors were all in close proximity with one another so we are confident they all offer good discriminatory power.

Conclusions

The report aimed to explore how different models perform at the same task of classifying breast cancer tumors into benign and malignant types using various predictors derived from the tumor images. Since we have eliminated most multicollinearity at the beginning, it is reasonable to expect Newton-Raphson and logistic-LASSO to have quite similar discriminatory performance. On the other hand, we would expect to see logistic-LASSO to perform better than Newton Raphson in terms of predictive ability in the presence of higher-dimensional data and highly correlated predictors. All in all, the models we explored in this report perform decently in giving correct classification of breast cancer types.

Tables and Figures

Figure 1. Pairwise correlation plot for predictors

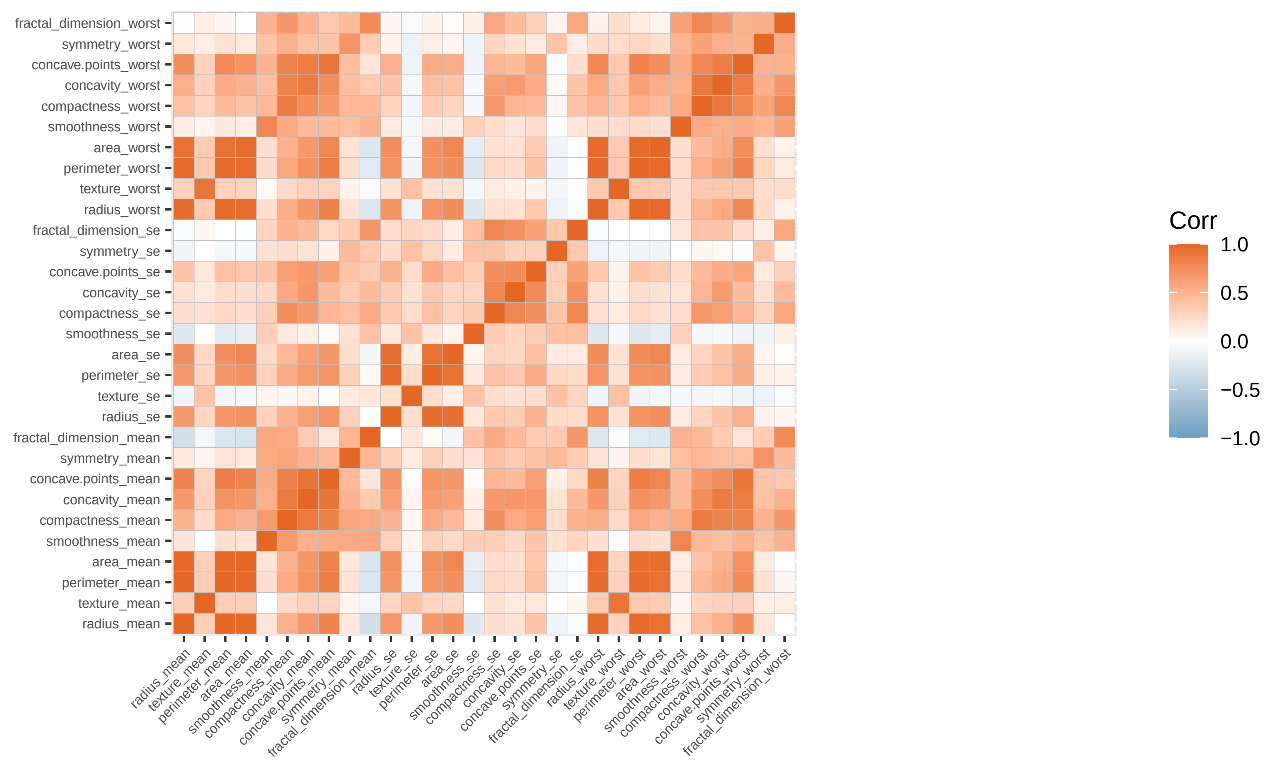
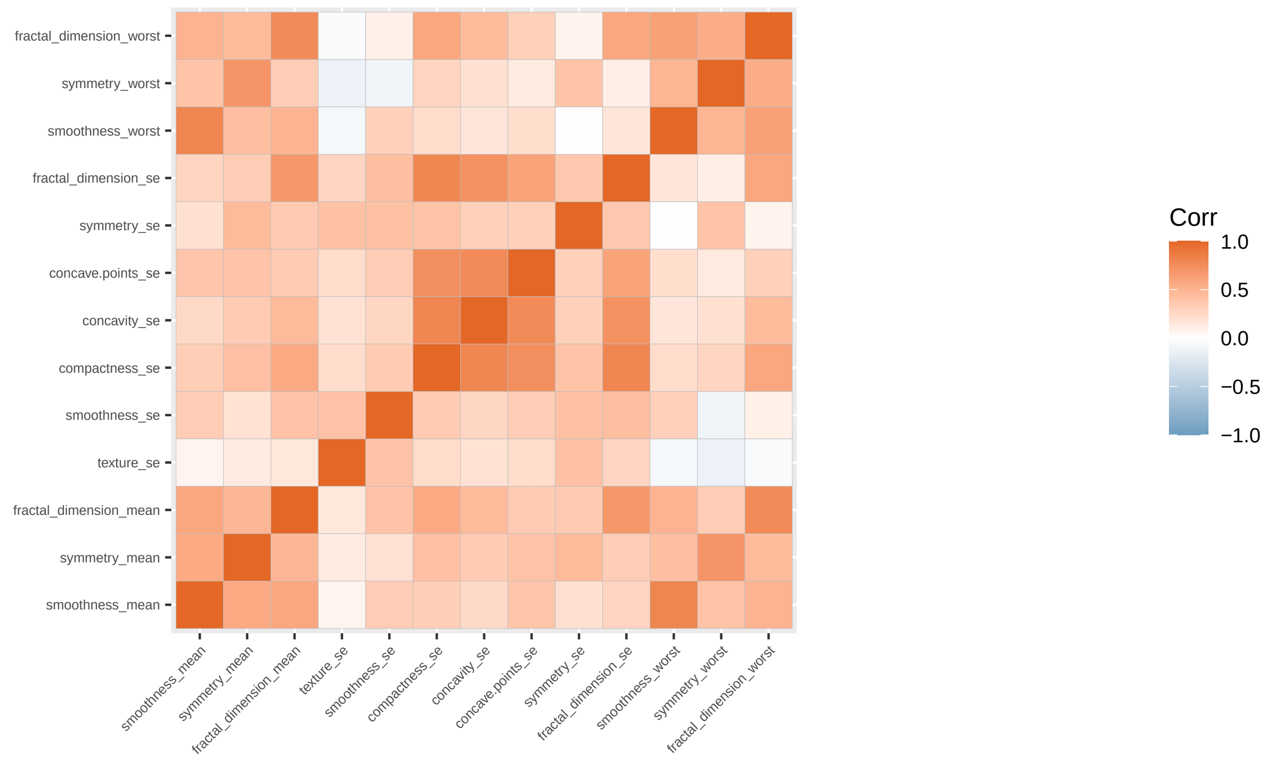


Figure 2. Pairwise correlation plot after removing some highly correlated variables



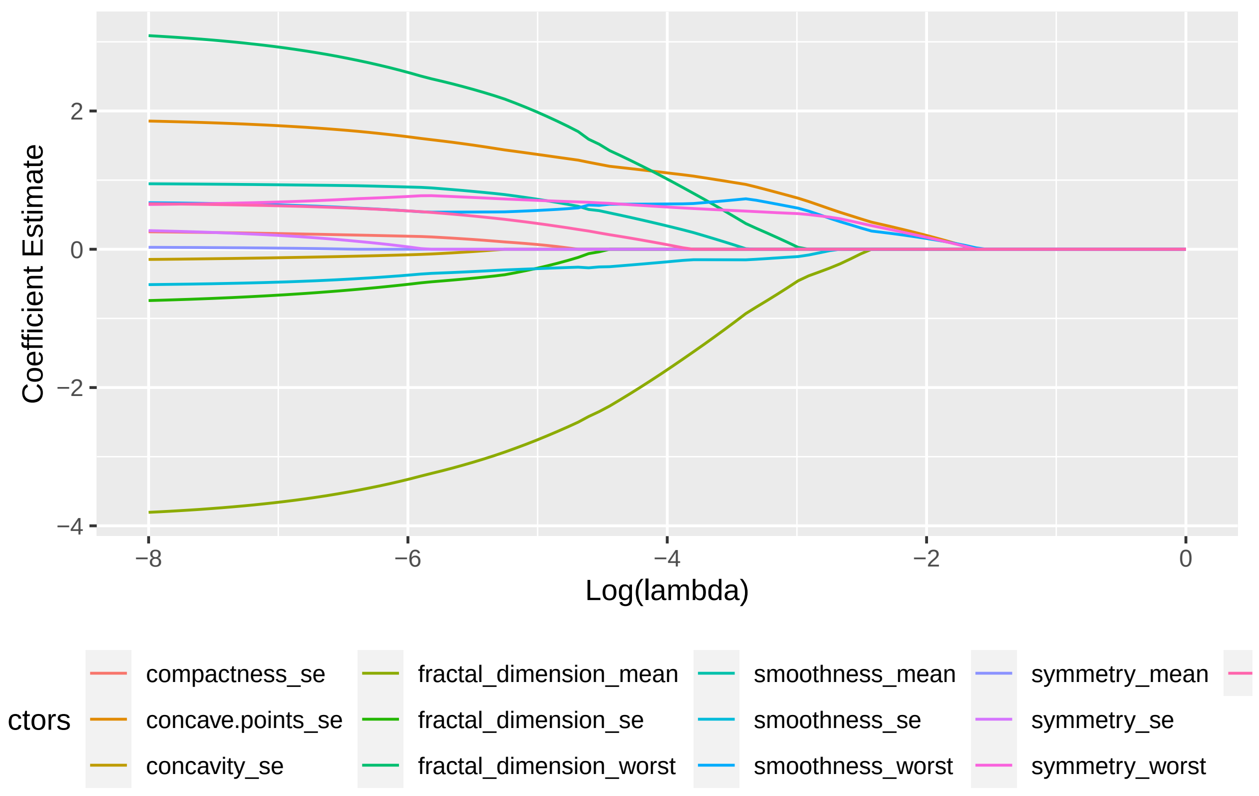


Figure 3. Solution path of coordinatewise optimization for logistic−LASSO

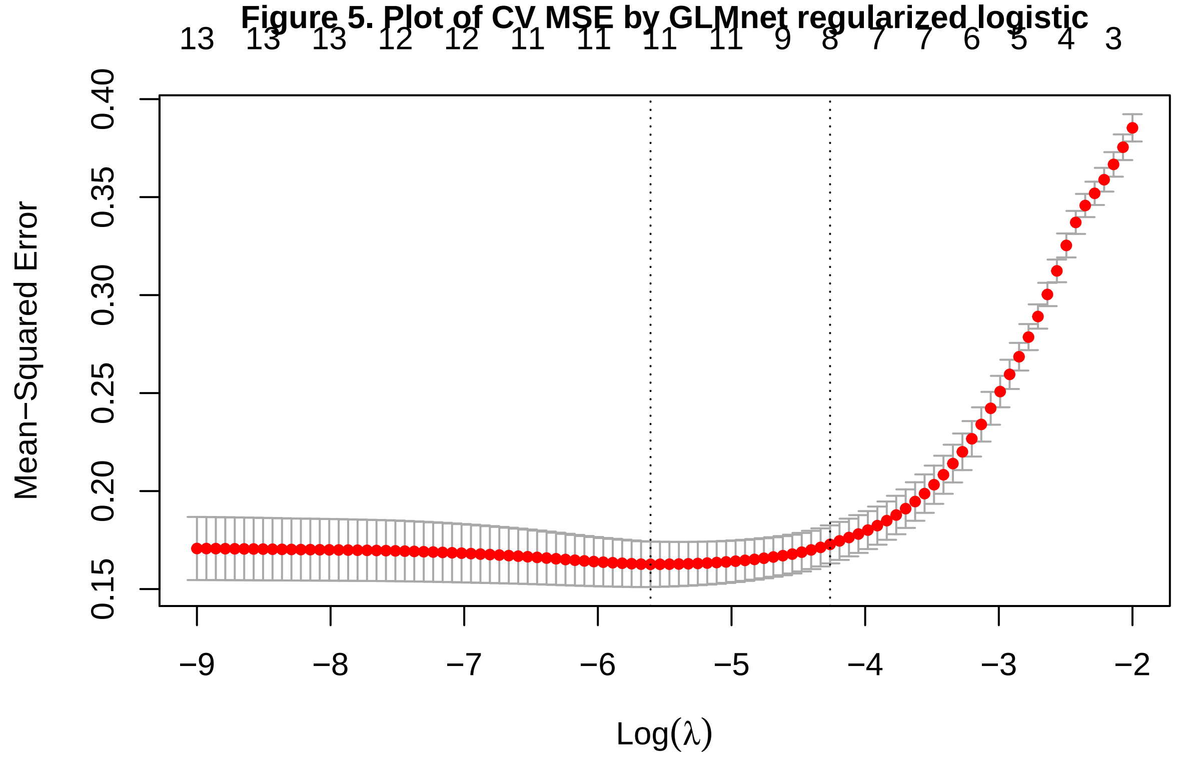


Figure 5. Plot of CV MSE by GLMnet regularized logistic

Figure 4. Plot of CV MSE by hand−built logistic−LASSO

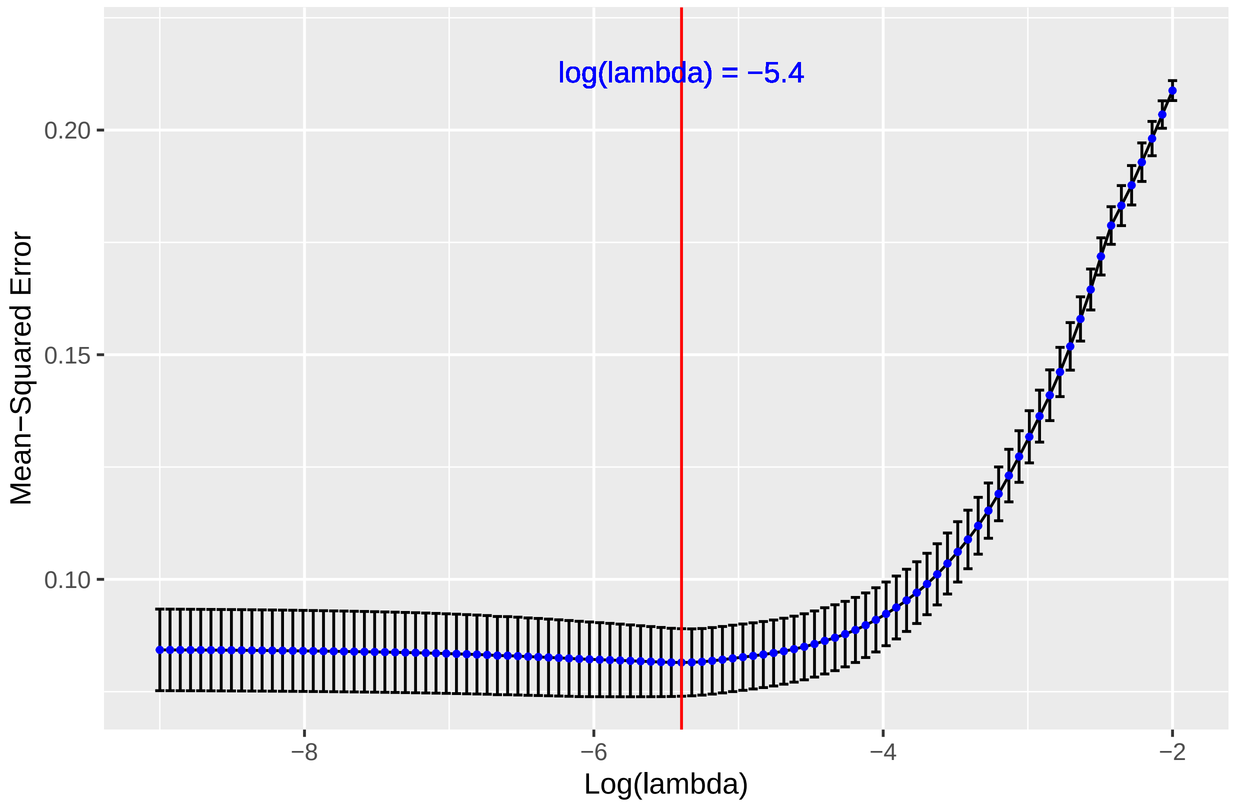


Figure 6. Distribution of 5−fold cross−validated MSE across models

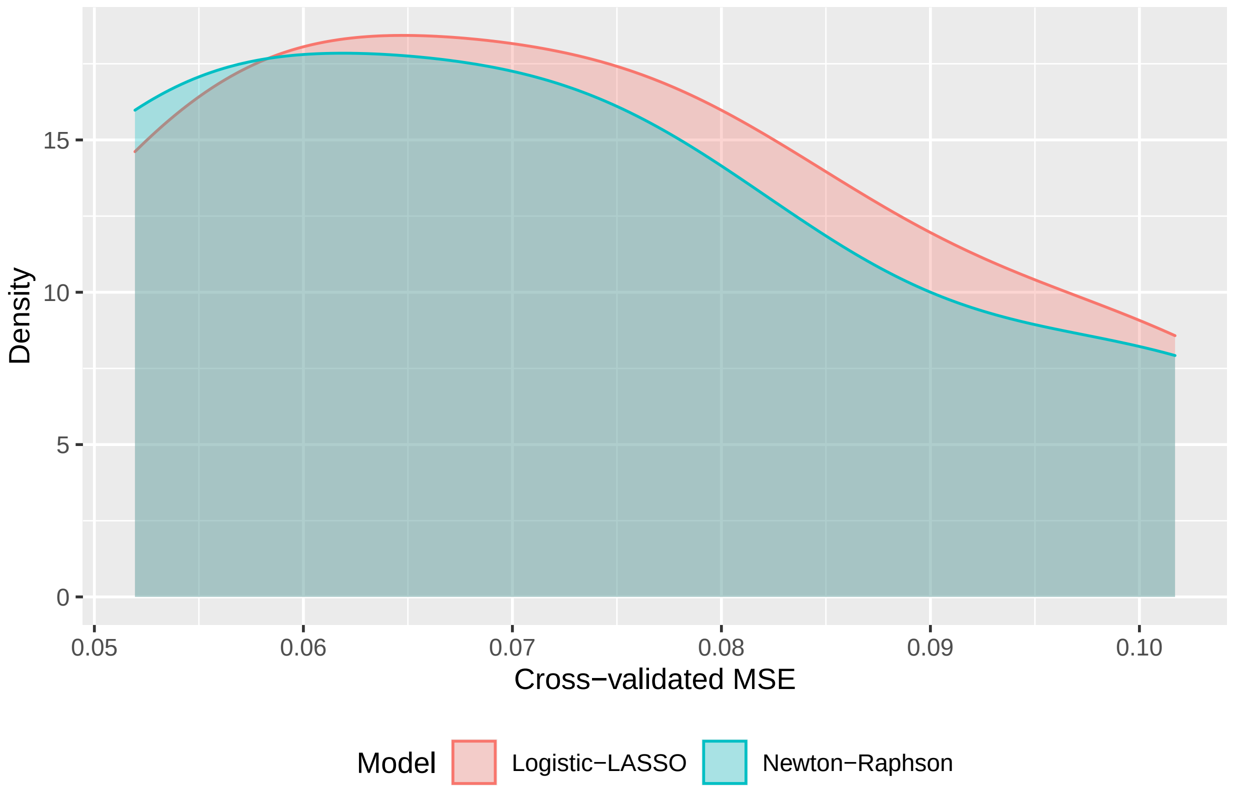


Table 1. Coefficient estimates by each model

	GLM binomial	Newton-Raphson	GLMnet	Logistic-LASSO
smoothness_mean	1.5725	1.5725	1.1176	1.0229
symmetry_mean	-0.1187	-0.1187	0.0000	0.0000
fractal_dimension_mean	-4.5061	-4.5061	-3.5738	-3.3703
texture_se	0.8068	0.8068	0.5009	0.4466
smoothness_se	-0.8304	-0.8304	-0.6757	-0.6253
compactness_se	0.3954	0.3954	0.1787	0.1392
concavity_se	-0.0300	-0.0300	0.0000	0.0000
concave.points_se	2.2856	2.2856	1.8469	1.7472
symmetry_se	-0.2039	-0.2039	-0.0997	-0.0615
fractal_dimension_se	-0.7551	-0.7551	-0.3363	-0.2821
smoothness_worst	0.7859	0.7859	0.8685	0.8577
symmetry_worst	1.0091	1.0091	0.7511	0.7050
fractal_dimension_worst	2.8346	2.8346	2.1066	1.9924
intercept	-1.5278	-1.5278	-1.2742	-1.1624

Table 2. MSE by each model tested on whole dataset

	Newton-Raphson	Logistic LASSO	GLMnet
MSE	0.0705	0.0727	0.0719

Appendix

Codes used for the project

Standardize design matrix

```
pred_names = bcdf %>% dplyr::select(-diagnosis) %>% names() %>% as.vector()
bcdf_x = NULL

for (i in pred_names) {
col = (bcdf[,i] - mean(bcdf[,i]))/sd(bcdf[,i])
bcdf_x = cbind(bcdf_x , col)
}

colnames(bcdf_x) <- c(pred_names)

bcdf_fin = cbind(bcdf[1], bcdf_x)
```

Find correlation pairs that are above 0.85 to leave out of the dataset

```
#obtain list of variables that are correlated with one another whose correlation is at least 0.85
cor_var = bcdf_x %>%
    correlate() %>%
    stretch() %>%
    arrange(desc(r)) %>%
    filter(r > 0.9) %>%
    slice(which(row_number() %% 2 == 0)) %>%
    pivot_longer(x:y) %>% dplyr::select(-r,-name) %>% distinct(value)

#full data with response variable and predictors
```

```
full_data = as_tibble(bcdf_fin) %>% dplyr::select(-perimeter_mean, -radius_mean, -perimeter_worst, -radi

#design matrix without intercept
Xmat_no_int = full_data %>% dplyr::select(-diagnosis)

#design matrix with intercept
Xmat_int = Xmat_no_int %>% mutate(intercept = 1)
```

###Looking at logistic regression results by glm

```
log.mod = glm(diagnosis~., data = full_data, family = "binomial")
summary(log.mod)

glm_coeff_tib = tibble(`GLM binomial` = round(replace(log.mod$coeff %>% as.numeric(), c(1,2:14), log.moc
```

Function to return log-likelihood, gradient, and Hessian matrix of logistic regression

```
logisticstuff <- function(y, x, betavec) {
  u <- x %*% betavec
  expu <- exp(u)
  loglik.ind = NULL

  loglik = t(u) %*% y - sum((log(1+expu)))
  # Log-likelihood at betavec

  p <- expu / (1 + expu)
  # P(Y_i=1|x_i)

  grad = t(x) %*% (y-p)
   #gradient at betavec

    # Hessian at betavec
  hess <- -t(x) %*% diag(as.vector(p*(1-p))) %*% x
  return(list(loglik = loglik, grad = grad, Hess = hess))
}
```

Newton-Raphson with gradient descent and step-halving

```
NewtonRaphson <- function(y, x, func, start, tol=1e-10, maxiter = 200) {
  i <- 0
  cur <- start
  x = as.matrix(x)
  colnames(x) = names(bcdf_x)
  stuff <- func(y, x , cur)
  res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf
  while(i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    grad <- stuff$grad
    hess <- stuff$Hess
    #gradient descent
    if(t(grad) %*% hess %*% grad > 0){#positive definite matrix
    inv.hess =
      solve(hess - (max(diag(hess))+100)*diag(nrow(hess)))} #make positive definite matrix negative def
    else
    {inv.hess <- solve(hess)}

    cur <- prev - inv.hess%*%grad
    stuff <- func(y, x, cur)

    #step-halving
    step = 0
    while (prevloglik > stuff$loglik){#moving too far -> halve step
    step = step + 1
    cur <- prev - (1/2)^step * inv.hess%*%grad
    stuff <- func(y, x, cur)
    }
  res <- rbind(res, c(i, stuff$loglik, cur))
  }
  return(res)
  }
```

Obtain estimated coefficients from modified Newton Raphson

```
newton_raph_res = NewtonRaphson(y = full_data$diagnosis, as.matrix(Xmat_int), logisticstuff, start = rep

#convert to data frame
newton_raph_coeff = newton_raph_res[c(nrow(newton_raph_res)),3:ncol(newton_raph_res)] %>% t() %>% as.dat

#assign names to coeffcieints
colnames(newton_raph_coeff) = colnames(Xmat_int)

#obtain final coeffcients
nr_coeff_tib = as_tibble(round(newton_raph_coeff,4))
```

Logistic-LASSO – Coordinate-wise descent algorithm

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

#soft_threshold = function(beta, lambda) {sign(beta)*ifelse(abs(beta)>lambda, abs(beta)-lambda,0)}

coord.lasso = function(lambda, y, X, betavec, tol = 1e-7, maxiter = 200){
  i = 0
  X = as.matrix(X)
  loglik = 1e6
  res = c(0, loglik, betavec)
  prevloglik = Inf
  while (i < maxiter && abs(loglik - prevloglik) > tol && loglik < Inf){
    i = i + 1
    prevloglik = loglik
    for (k in 1:length(betavec)){
      u = X %*% betavec
      expu = exp(u)
      p = expu/(1 + expu)
      weight = p*(1-p)

      #avoid coefficients from divergence to achieve final fitted probabilities of 0 or 1
      weight = ifelse(abs(weight-0) < 1e-6, 1e-6, weight)

      #calculate working responses
      resp = u + (y-p)/weight
      #r = z - X%*%betavec
      resp_without_j = X[,-k] %*% betavec[-k]

      #soft-threshold solution
      betavec[k] = soft_threshold(mean(weight*X[,k]*(resp-resp_without_j)),lambda)/(mean(weight*(X[,k]^

    #calculate new log-likelihood
  loglik = 1/(2*nrow(X))*sum(weight*(resp-X%*%betavec)^2) + lambda*sum(abs(betavec))
  res = rbind(res, c(i, loglik, betavec))
  }
  return(res)
}
```

```
coord.lasso(lambda = 0.1,
            y = full_data$diagnosis,
            X = as.matrix(Xmat_no_int),
            betavec = rep(0, ncol(Xmat_no_int)))
```

Compute the solution on a grid of lambdas – pathwise coordinate optimization to get path of solutions

```
path = function(X, y, tunegrid){
  coeff = NULL
  tunegrid = as.vector(tunegrid)
  for (nl in tunegrid){
    coord_res = coord.lasso(lambda = nl,
                            X = as.matrix(X),
                            y = y,
                            betavec = rep(0, ncol(X)))
    last_beta = coord_res[nrow(coord_res),3:ncol(coord_res)]
    betavec = last_beta
    coeff = rbind(coeff, c(last_beta))
  }
  return(cbind(tunegrid, coeff))
}

path_df = path(X = Xmat_no_int, y = bcdf_fin$diagnosis, tunegrid = exp(seq(0, -8, length = 100)))
colnames(path_df) = c("Tunegrid", colnames(Xmat_no_int))
path_df = as.data.frame(path_df)
```

Cross validation for logistic-LASSO

```
set.seed(2020)
mses = NULL
mse = NULL
rmse.std.error = NULL
grid = NULL
i = 0
crossval = function(X, y, tunegrid, fold_num){
  folds = sample(1:fold_num, nrow(X), replace = TRUE)
for(nl in tunegrid){
  i = i + 1
  for(k in 1:fold_num){
  #start = rep(1, ncol(X))
  x_train = as.matrix(X[folds != k,])
  y_train = y[folds != k]
  x_test = as.matrix(X[folds == k,])
  y_test = y[folds == k]
  start = rep(1, ncol(x_train))
  loglasso_res = coord.lasso(lambda = nl,
                             y = y_train,
                             X = x_train,
                             betavec = start)
  loglasso_coeff = loglasso_res[nrow(loglasso_res),3:ncol(loglasso_res)]
  expu = exp(x_test %*% loglasso_coeff)
  p = expu/(1+expu)
  mses[k] = mean((y_test-p)^2) #cross-validated MSE
  start = loglasso_coeff
  }
  mse[i] = mean(mses)
  rmse.std.error[i] = sqrt(var(mses)/fold_num)
  grid[i] = nl
  res = cbind(grid, mse, rmse.std.error)
}
  return(res)}

cv_res = crossval(X = Xmat_int, y = full_data$diagnosis, tunegrid = exp(seq(-9,-2,length = 100)), fold_
```

```
#Find best lambda
best.ll.lambda = cv_res %>% filter(mse == min(cv_res$mse)) %>% dplyr::select(grid)
best.ll.lambda
log(best.ll.lambda)
```

Perform cross-validation logistic LASSO in glmnet (for comparison)

```
set.seed(2020)
cv.lasso <- cv.glmnet(as.matrix(Xmat_no_int), y = as.factor(full_data$diagnosis),
                      family="binomial",
                      type.measure = "mse",
                      nfolds = 5,
                      alpha = 1,
                      lambda = exp(seq(-9, -2, length=100)))

cv.lasso$lambda.min
```

```r
log(cv.lasso$lambda.min)

#coefficients
coeff = coef(cv.lasso, s=cv.lasso$lambda.min) %>% as.numeric()
glmnet_coeff = replace(coeff, c(1,2:14), coeff[c(2:14,1)])

#make tibble glmnet lasso coeff
glmnet_coeff_tib = tibble(`GLMnet` = round(glmnet_coeff,4))
```

Calculate MSE for Logistic-Lasso and Newton Raphson

```r
pred_error = function(y, X, betavec) {
  expu = exp(as.matrix(X) %*% betavec)
  p = expu/(1+expu)
  prediction_error = mean((as.vector(y)-p)^2)
  return(prediction_error)
}
```

Newton Raphon's MSE

```r
newton_raph_vec = newton_raph_res[c(nrow(newton_raph_res)),3:ncol(newton_raph_res)]

#nr_coeff_tib = tibble(`Newton-Raphson` = round(newton_raph_res[c(nrow(newton_raph_res)),3:ncol(newton_
newton_raph_error = pred_error(full_data$diagnosis, Xmat_int, newton_raph_vec)
newton_raph_error
```

Logistic-Lasso's MSE

```r
loglasso_betas = coord.lasso(lambda = as.numeric(best.ll.lambda),
            y = full_data$diagnosis,
            X = as.matrix(Xmat_int),
            betavec = rep(1, ncol(Xmat_int)))
#get coefficients at best lambda
loglasso_betas = loglasso_betas[nrow(loglasso_betas), 3:ncol(loglasso_betas)]

#make tibble
loglasso_coeff_tib = tibble(`Logistic-LASSO` = round(loglasso_betas, 4))

#calc error
loglasso_error = pred_error(full_data$diagnosis, Xmat_int, loglasso_betas)
loglasso_error
```

GLMNet's MSE

```r
glmnet_error = pred_error(full_data$diagnosis, Xmat_int, glmnet_coeff)
glmnet_error
```

Cross-validation for all models

```r
nr_mses = NULL
ll_mses = NULL
glmnet_mses = NULL
error_comp_df = NULL
set.seed(2020)

#k-fold cross-validation
cv_comp = function(X, y, fold_num){
  folds = sample(1:fold_num, nrow(X), replace = TRUE)
  for (k in 1:fold_num){
  #start = rep(1, ncol(X))
  x_train = as.matrix(X[folds != k,])
  y_train = y[folds != k]
  x_test = as.matrix(X[folds == k,])
  y_test = y[folds == k]

  ll_expu = exp(x_test %*% loglasso_betas)
  ll_p = ll_expu/(1+ll_expu)
  ll_mse = mean((y_test-ll_p)^2) #cross-validated MSE for logistic lasso
  ll_mses = rbind(ll_mses, ll_mse)

  nr_expu = exp(x_test %*% newton_raph_vec)
  nr_p = nr_expu/(1+nr_expu)
  nr_mse = mean((y_test - nr_p)^2) #cross-validated MSE for newton-raphson
  nr_mses = rbind(nr_mses, nr_mse)
  #glmnet_expu = exp(x_test %*% glmnet_coeff)
  #glmnet_p = glmnet_expu/(1+glmnet_expu)
  #glmnet_mse = mean((y_test - glmnet_p)^2) #cross-validated MSE for glmnet
  #glmnet_mses = rbind(glmnet_mses, glmnet_mse)
  }
  res = tibble(`Logistic-LASSO` = ll_mses, `Newton-Raphson` = nr_mses)
  return(res)}

#repeated cross-validation n times
rep_cv = function(X, y, fold_num, n){
  while (i <= n){
    i = i+1
    error_comp = cv_comp(X, y, fold_num)
    error_comp_df = rbind(error_comp_df, error_comp)
  }
  return(error_comp_df)
}
mse_comp_df = rep_cv(X = Xmat_int, y = full_data$diagnosis, fold_num = 5, n =1)
#rep_mse_comp_df = rep_cv(X = Xmat_int, y = full_data$diagnosis, fold_num = 5, n =5)
```