USING NEWTON-RAPHSON AND LOGISTIC-LASSO FOR PREDICTING BREAST CANCER STATUS

A PREPRINT

Xinyi Lin
xl2836

Junting Ren
jr3755

Christian Pascual
cbp2128

March 9, 2019

1 Introduction

The diagnosis of breast cancer at an early stage is an important goal of screening. While many different screening tests exist today, there is still room for improvement. A promising avenue for breast cancer detection is prediction based off of breast cancer tissue images. For example, a study by Kim et al. demonstrated that regression models could be helpful in diagnosing breast cancer using Logistic LASSO and stepwise logistic regression. [1]

This report seeks to validate their findings and use regression methods to predict breast cancer status from image data. The two techniques of interest will be a logistic model using the Newton-Raphson optimization and another using coordinate descent in a LASSO context.

2 Methods

2.1 The data set

Our data set contains 569 observations and 33 rows. The response is a binary variable, $diagnosis$, that takes either M for malignant tissue or B for benign tissue. We take "malignant" to take on the 1 value. There are 30 potential predictors that are derived from the image data, including the mean, standard deviation and largest values of various features in the images. Important examples of features included in the data set include cell nuclei radius, texture (derived from gray-scale values), and concavity.

The data was centered and scaled before use in any model to ensure comparability between models and to prevent undue influence from differences in magnitude between predictors.

2.2 Full model

Due to the highly correlated nature of the data set, we have chosen a subset of 12 predictors of the original 31 to represent a full model. This set of predictors includes: mean radius, mean texture, mean perimeter, mean smoothness, mean compactness, mean concavity, mean concave points, mean fractal dimension, mean symmetry, standard error of radius, perimeter, and symmetry. This model will be used as the reference against the Newton-Raphson model and the Logistic-LASSO model.

2.3 Newton-Raphson model

For logistic regression, the probability of a malignant tissue is given by:

$$P(Y_i = 1|X_i) = p_i = \frac{e^{\beta_0 + X_i\boldsymbol{\beta}}}{1 + e^{\beta_0 + X_i\boldsymbol{\beta}}}$$

With n observations, we can derive the likelihood $L(y; \boldsymbol{\beta})$ and the log-likelihood $l(y; \boldsymbol{\beta})$ for data $(\boldsymbol{x_1}, Y_1), \ldots, (\boldsymbol{x_n}, Y_n)$:

$$L(\boldsymbol{\beta}) = \prod [(\frac{e^{\boldsymbol{x_i^T \beta}}}{1 + e^{\boldsymbol{x_i^T \beta}}})^{Y_i} (\frac{1}{1 + e^{\boldsymbol{x_i^T \beta}}})^{1-Y_i}]$$

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} (Y_i(\beta_0 + X_i \boldsymbol{\beta}) - log(1 + e^{\beta_0 + X_i \boldsymbol{\beta}})$$

From the log-likelihood, we can derive both the gradient $\nabla l(y; \beta)$ and the Hessian $\nabla^2 l(y; \beta)$

$$\nabla l(y; \beta) = \begin{bmatrix} \sum_{i=1}^{n} (Y_i - p_i) \\ \sum_{i=1}^{n} x_{i1}(Y_i - p_i) \\ \cdot \\ \cdot \\ \cdot \\ \sum_{i=1}^{n} x_{in}(Y_i - p_i) \end{bmatrix}$$

$$\nabla^2 l(y; \beta) = \begin{bmatrix} \sum_{i=1}^{n} p_i(1 - p_i) & \sum_{i=1}^{n} X_i^T p_i(1 - p_i) \\ \sum_{i=1}^{n} X_i p_i(1 - p_i) & \sum_{i=1}^{n} X_i X_i^T p_i(1 - p_i) \end{bmatrix}$$

With these components, we can use the Newton-Raphson algorithm to calculate a set of β coefficients with the following equation:

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

for which at each iteration, the log-likelihood will be recalculated. The Newton-Raphson algorithm requires an initial guess for the β vector. For our initial guess, we used a vector of size 13, with each element equal to 0.001. We defined convergence as when the difference between the current log-likelihood and the last iteration's log-likelihood reached below 1^-5.

2.4 Logistic-LASSO Model

For this model, we sought to minimize the objective function, derived from the quadratic Taylor approximation to the binomial log-likelihood function:

$$\underset{(\beta_0, \beta_\mathbf{1})}{min} \left(\frac{1}{2n} \sum_{i=1}^{n} \omega_i (z_i - \beta_0 - \mathbf{x}_i^T \beta_\mathbf{1})^2 + \lambda \sum_{j=0}^{p} |\beta_j| \right)$$

ω_i is the working weight, z_i is the working response, β_0 is the intercept and $\beta_\mathbf{1}$ is the set of β coefficients.

This objective function was minimized using coordinate descent. The intercept term was not penalized. Each β_k was optimized using the following equation:

$$\tilde{\beta_j} = \frac{S(\sum_i \omega_i x_{i,j} (y_i - \tilde{y}_i^{(-j)}), \gamma)}{\sum \omega_i x_{i,j}^2}$$

where S is the weighted soft threshold function, y^{-j} is the response omitting β_j and γ is the threshold to be used on all the β_k.

Similar to our Newton-Raphson algorithm, our Logistic-LASSO defined convergence to occur when the Frobenius norm between the calculated β and the β from the last iteration to drop below 1^{-5}. The same 0.001 vector will also be used to initialize the algorithm.

Table 1: Comparison of model coefficient estimates

Coefficient	Newton-Raphson	Logistic-LASSO
Intercept	-0.8814618	0.372583
Mean Radius	5.3048266	0.2884114
Mean Texture	1.7317848	0.0863283
Mean Perimeter	-2.2869860	-0.1923504
Mean Smoothness	0.7641516	0.0230698
Mean Compactness	-0.3487548	0.0142313
Mean Concavity	1.1529339	0239956
Mean Concave Points	2.6332369	0.2106158
Mean Fractal Dimension	-0.3600730	-0.0287683
Mean Symmetry	1.0065376	0.0348996
SE Radius	0.9500865	0.1044217
SE Perimeter	-0.7418374	-0.1139348
SE Symmetry	-0.8810955	-0.0214862

2.5 Assessing optimal parameters and predictive ability

5-fold cross validation will be used to find an optimal λ value for the Logistic-LASSO model. The optimal λ will be defined as the λ that minimizes the average test MSE across the tested λ values.

In order to assess predictive ability, we will use 10-fold cross validation to derive an average test MSE for each of the 2 models (Newton-Raphson, and Logistic LASSO). The best model will be defined as the one with the lowest average test MSE among the folds.

3 Results

3.1 Model coefficient estimates

The estimated coefficients for each model are listed in Table 1. The estimates for the Newton-Raphson algorithm match what is estimated in the glm implementation of logistic regression. However, the estimates for the Logistic-LASSO do not match up against the implementation in $glmnet$. We attribute these errors to needing a better initial β vector, a stricter convergence tolerance or slight differences in algorithm implementation.

Figure 1 illustrates a path of solutions along increasing λ's. The constant line represents the intercept since we chose not to penalize it in our implementation. The graph demonstrates that 2 parameters (mean radius and mean concave points) have a large effect on the log odds of classifying as a malignant image.

3.2 Optimal lambda for Logistic-LASSO

After the 5-fold cross validation, we found that a λ value of 0.29 had the lowest MSE between the range of e^{-7} to e^{5} Figure 2 shows how the average test MSE among the 5 folds changed as a function of $log(\lambda)$, the minimum is depicted at the horizontal red line.

3.3 Model with best predictive ability

Figure 3 graphs the distribution of test MSEs created in the 10-fold cross validation. The distribution of test MSEs from the Logistic-LASSO are significantly less than those of the Newton-Raphson model. Therefore with our data set, the Logistic-LASSO model has better predictive ability.

4 Conclusion

This report sought to explore and compare how two different models are able to predict cancer malignancy given various image data. The optimization of these models requires the use of the Newton-Raphson and coordinate descent algorithms in a logistic regression context. We used cross validation to optimize the regularization parameter λ and to judge the predictive ability of both models. In the end, Logistic-LASSO produced a model that was more effective

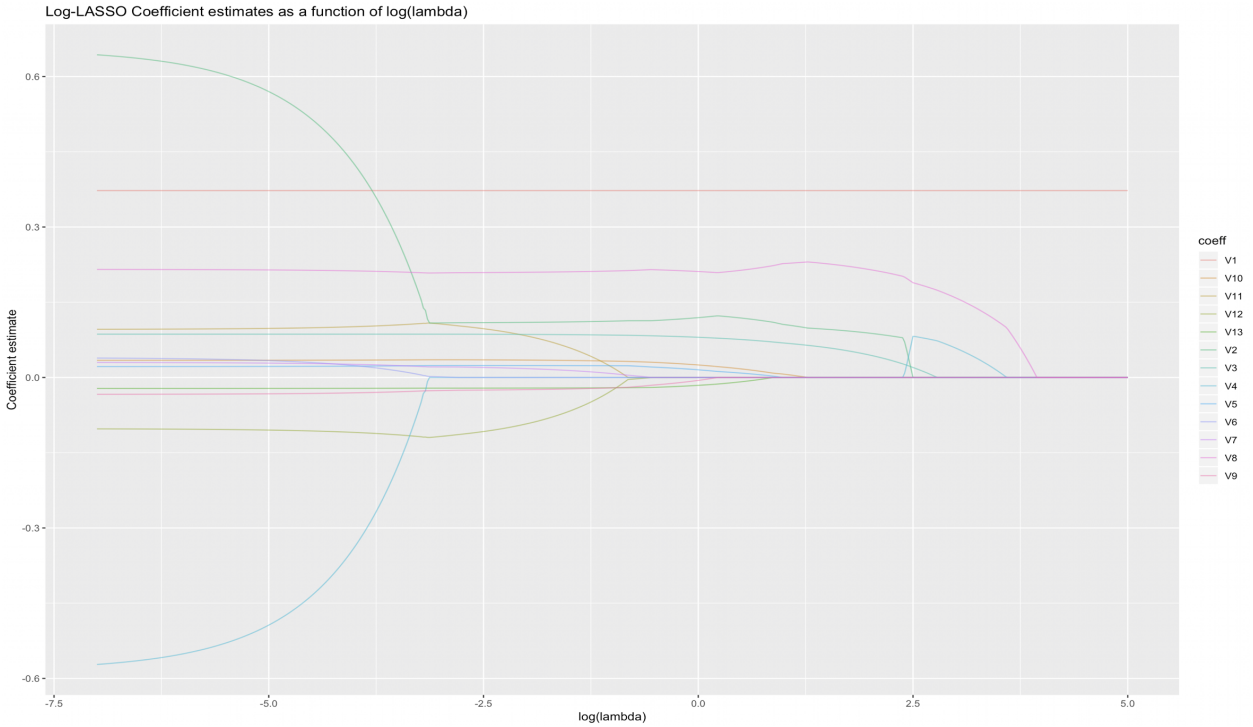

Figure 1: Comparing average test MSE against λ

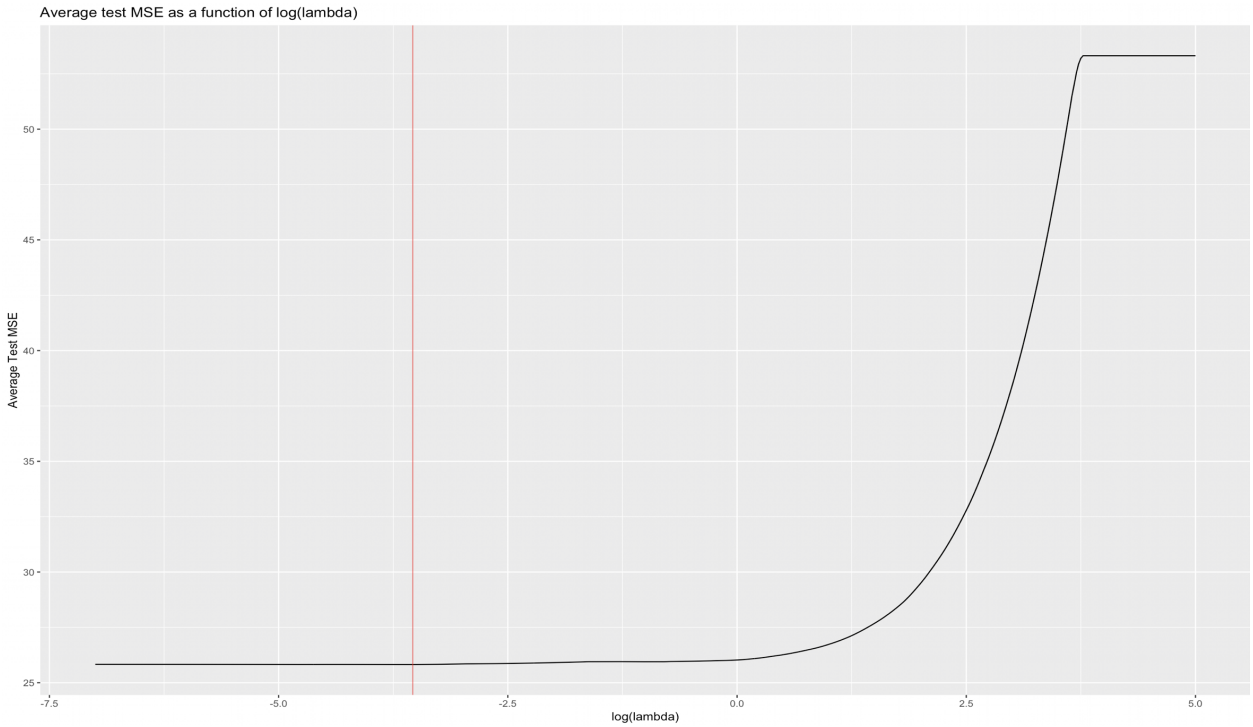

Figure 2: Comparing average test MSE against λ

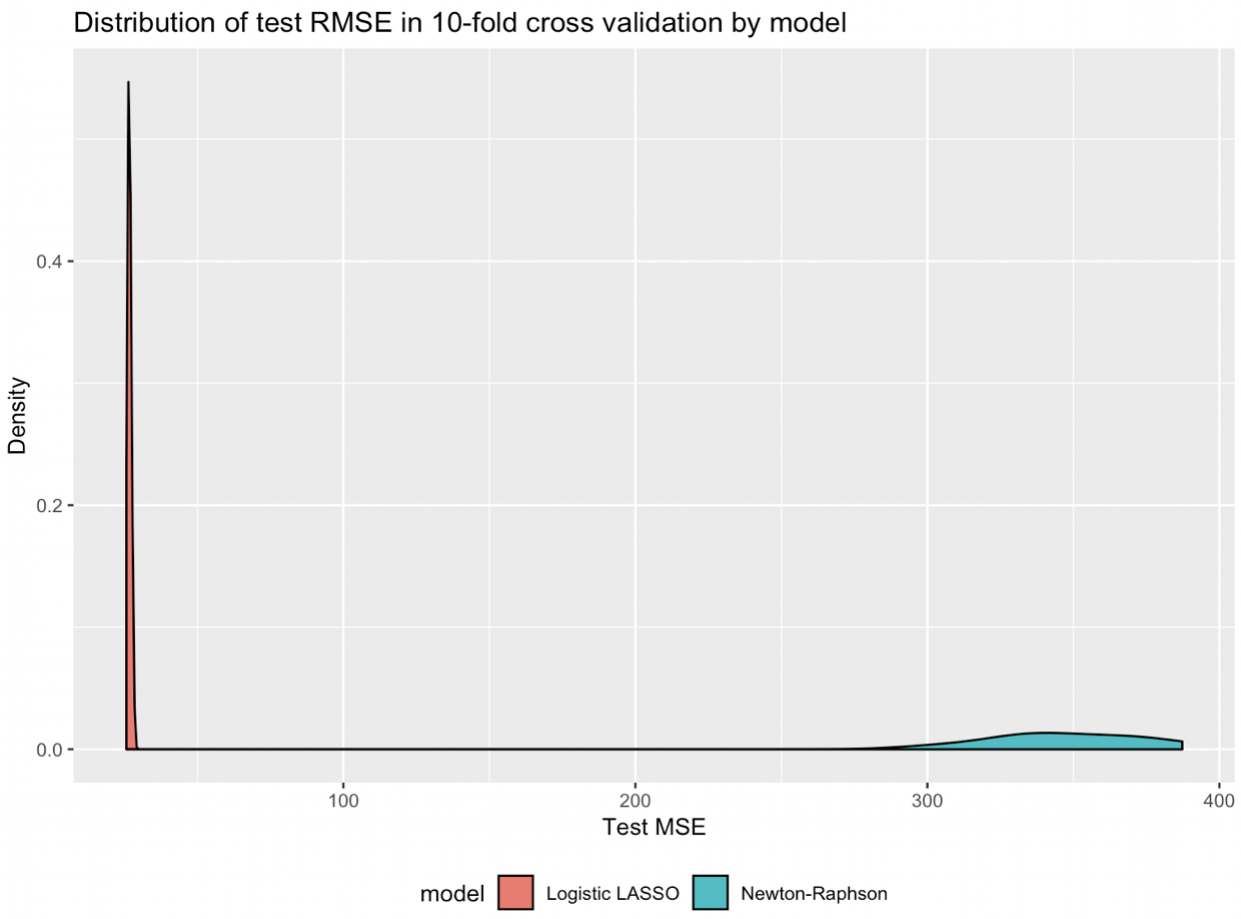

Figure 3: Comparing average test MSE against λ

at predicting cancer malignancy in the data. These types of models have promise in improving healthcare through improved diagnostics.

References

[1] Kim, Sun Mi et al. Logistic LASSO regression for the diagnosis of breast cancer using clinical demographic data and the BI-RADS lexicon for ultrasonography *Ultrasonography (Seoul, Korea)* vol. 37,1 (2017): 36-42.

Appendix: Code

Christian Pascual, Xinyi Lin, Junting Ren

3/7/2019

```
library(tidyverse)
library(glmnet)
library(modelr)
library(matrixcalc)
set.seed(8160)
```

Data preparation

```
standardize = function(col) {
  mean = mean(col)
  stdev = sd(col)
  return((col - mean)/stdev)
}

# just standardize the covariates
standardized.data = read.csv(file = "breast-cancer-1.csv") %>%
  dplyr::select(radius_mean:fractal_dimension_worst) %>%
  map_df(.x = ., standardize)

# add back in the response and ids
data = cbind(read.csv(file = "breast-cancer-1.csv") %>% dplyr::select(diagnosis), standardized.data) %>%
  mutate(diagnosis = ifelse(diagnosis == "M", 1, 0))

needed.cols = c("diagnosis", "radius_mean", "texture_mean", "perimeter_mean",
                "smoothness_mean", "compactness_mean", "concavity_mean",
                "concave.points_mean", "fractal_dimension_mean",
                "symmetry_mean", "radius_se", "perimeter_se", "symmetry_se")

short.data = data %>%
  dplyr::select(which(colnames(data) %in% needed.cols))
```

Data setup

```
# Christian's data preparation
cbp.X = data %>%
  dplyr::select(radius_mean, texture_mean, perimeter_mean, smoothness_mean,
                compactness_mean, concavity_mean, concave.points_mean,
                fractal_dimension_mean, symmetry_mean, radius_se,
                perimeter_se, symmetry_se)

# Xinyi's data preparation
xl.x = data %>%
```

```
  mutate(intercept = 1) %>%
  dplyr::select(intercept, radius_mean, texture_mean, perimeter_mean,
                smoothness_mean, compactness_mean, concavity_mean,
                concave.points_mean, fractal_dimension_mean, symmetry_mean,
                radius_se, perimeter_se, symmetry_se) %>%
  as.matrix(.)

cbp.y = data$diagnosis
xl.y = as.matrix(data$diagnosis)

# Coefficients from regular glmnet
f = formula(diagnosis ~ radius_mean + texture_mean + perimeter_mean + smoothness_mean + compactness_mean

full.fit = glm(f, family = binomial(link = "logit"), data = data)
```

Functions needed for analysis

```
soft.threshold = function(beta, gamma) {
  new.beta = beta
    if (abs(beta) > gamma && beta > 0) {
      new.beta = beta - gamma
    } else if (abs(beta) > gamma && beta < 0) {
      new.beta = beta + gamma
    } else {
      new.beta = 0
    }
  return(new.beta)
}

calc.cur.p = function(intercept, data, betas) {
  # return n x 1 array of current probabilities evaluated with given betas
  return(
    exp(intercept * rep(1, nrow(data)) + data %*% betas) / (1 + exp(intercept * rep(1, nrow(data)) + da
  )
}

calc.working.resp = function(intercept, data, resp, betas, p) {
  # return n x 1 array of working responses evaluated with given betas
  return(
    intercept * rep(1, nrow(data)) + data %*% betas + (resp - p) / (p * (1 - p))
  )
}

calc.working.weights = function(p) {
  # return n x 1 array of working weights for the data
  return(p * (1 - p))
}

calc.obj = function(data, weights, w.resp, intercept, betas, lambda) {
  # return the objective function of data and current params
  return(
```

```
    log.lik = 1/(2 * nrow(data)) *
      sum((weights * (w.resp - intercept * rep(1, nrow(data)) - data %*% betas))^2) + lambda * sum(abs(
  )
}
```

Logistic LASSO implementation

```
LogLASSO.CD = function(X, y, beta, lambda, tol = 1e-5, maxiter = 1000) {

  ### Parameters: ##############################################################
  # X : design matrix                                                          #
  # y : response variable (should be binary)                                   #
  # beta : starting beta coefficients to start from                            #
  # lambda : constraining parameter for LASSO penalization                     #
  # tol : how precise should our convergence be                                #
  # maxiter : how many iterations should be performed before stopping          #
  ##############################################################################

  # Turn the betas into their own matrix
  X = as.matrix(X)

  # exclude the intercept from the input betas
  beta = as.matrix(beta[2:length(beta)])

  # Initialize important parameters before starting the coordinate descent
  beta0 = 1/length(y) * sum(y - X %*% beta)
  p = calc.cur.p(intercept = beta0, data = X, betas = beta)
  z = calc.working.resp(intercept = beta0, data = X, resp = y,
                        betas = beta, p = p)
  omega = calc.working.weights(p)
  obj = calc.obj(data = X, weights = omega, w.resp = z,
                 intercept = beta0, betas = beta, lambda = lambda)

  # Initialize the row for tracking each of these parameters
  path = c(iter = 0, intercept = beta0, obj = obj, beta)

    for (j in 1:maxiter) {

      prev.beta = beta

      # Coordinate descent
      for (k in 1:length(beta)) {
        r = y - (X %*% beta) + (X[,k] * beta[k])
        threshold.val = sum(omega * X[,k] * r)
        beta[k] = (soft.threshold(threshold.val, gamma = lambda)) / sum(omega * X[,k]^2)
      }

      # With new betas, recalculate the working parameters
      beta0 = mean(y) - sum(colMeans(X) * beta)
      p = calc.cur.p(intercept = beta0, data = X, betas = beta)
      z = calc.working.resp(intercept = beta0, data = X, resp = y,
```

```
                        betas = beta, p = p)
      omega = calc.working.weights(p)
      obj = calc.obj(data = X, weights = omega, w.resp = z,
                     intercept = beta0, betas = beta, lambda = lambda)

      # Append it to tracking matrix
      path = rbind(path, c(iter = j, intercept = beta0, beta, obj = obj))

      # Break the loop if the diff between likelihoods is below tolerance
      if (
        norm(prev.beta - beta, "F") < tol
        ) { break }
    }

  return(list(
    path = as.tibble(path),
    coefficients = c(beta0, beta))
    )
  }
```

Newton-Raphson implementation

```
logisticstuff <- function(x, y, betavec) {
  u <- x %*% betavec
  expu <- exp(u)
  loglik = vector(mode = "numeric", nrow(x))
  for(i in 1:nrow(x))
    loglik[i] = y[i]*u[i] - log(1 + expu[i])
  loglik_value = sum(loglik)
  # Log-likelihood at betavec
  p <- expu / (1 + expu)
  # P(Y_i=1|x_i)
  grad = vector(mode = "numeric", length(betavec))
  #grad[1] = sum(y - p)
  for(i in 1:13)
    grad[i] = sum(t(x[,i])%*%(y - p))
  #Hess <- -t(x)%*%p%*%t(1-p)%*%x
  Hess = hess_cal(x, p)
  return(list(loglik = loglik_value, grad = grad, Hess = Hess))
}

hess_cal = function(x, p) {
  len = length(p)
  hess = matrix(0, ncol(x), ncol(x))
  for (i in 1:len) {
    unit = x[i,] %*% t(x[i,]) * p[i] *(1 - p[i])
    #unit = t(x[i,])%*%x[i,]*p[i]*(1-p[i])
    hess = hess + unit
  }
  return(-hess)
}
```

```r
NewtonRaphson <- function(x, y, logisticstuff, start, tol = 1e-5, maxiter = 200) {
  i <- 0
  cur <- start
  stuff <- logisticstuff(x, y, cur)
  res = c(0, cur)
  #res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf      # To make sure it iterates
  #while(i < maxiter && abs(stuff$loglik - prevloglik) > tol && stuff$loglik > -Inf)
  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    i <- i + 1
    prevloglik <- stuff$loglik
    print(prevloglik)
    prev <- cur
    cur <- prev - solve(stuff$Hess) %*% stuff$grad
    stuff <- logisticstuff(x, y, cur)        # log-lik, gradient, Hessian
    res = rbind(res, c(i, cur))
    #res <- rbind(res, c(i, stuff$loglik, cur))
    # Add current values to results matrix
    }
  return(res)
}
modified <- function(x, y, logisticstuff, start, tol=1e-5, maxiter = 200){
  i <- 0
  cur <- start
  beta_len <- length(start)
  stuff <- logisticstuff(x, y, cur)
  res = c(0, cur)
  #res <- c(0, stuff$loglik,cur)
  prevloglik <- -Inf # To make sure it iterates
  while(i <= maxiter && abs(stuff$loglik - prevloglik) > tol)
  #while(i <= maxiter && abs(stuff$loglik - prevloglik) > tol && stuff$loglik > -Inf)
    { i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    lambda = 0
    while (is.negative.definite(stuff$Hess-lambda*diag(beta_len)) == FALSE) {
      lambda = lambda + 1
    }
    cur <- prev - solve(stuff$Hess-lambda*diag(beta_len)) %*% stuff$grad
    #cur <- prev + (diag(beta_len)/10)%*%(stuff$grad)
    #cur = prev + t(stuff$grad)%*%(stuff$grad)
    stuff <- logisticstuff(x, y, cur) # log-lik, gradient, Hessian
    res = rbind(res, c(i, cur))
    #res <- rbind(res, c(i, stuff$loglik, cur))
    }
  return(round(res,2))
}
```

Visualizations

betas vs lambda

```
# Generate data to visualize how the coefficients change with the logistic LASSO
lambda.seq = exp(seq(-7, 5, length = 500))
start.betas = rep(0.001,13)

coeff.path = NULL
for (l in 1:length(lambda.seq)) {
  fit = LogLASSO.CD(X = cbp.X, y = cbp.y, beta = start.betas, lambda = lambda.seq[l])
  coeff.path = rbind(coeff.path, c(lambda = lambda.seq[l], fit$coefficients))
  print(paste("Iter", l, "done", sep = " ")) # progress bar
}

#colnames(coeff.path) = c("lambda", paste("V", 1:13, sep = ""))
tidy.lambda = as.tibble(coeff.path) %>%
  gather(., key = "coeff", value = "coeff_est", V1:V13) %>%
  mutate(
    log.lambda = log(lambda)
  )

ggplot(data = tidy.lambda, aes(x = log.lambda, y = coeff_est, color = coeff, group = coeff)) +
  geom_line(alpha = 0.5) +
  theme(legend.position = "right") +
  labs(
    title = "Log-LASSO Coefficient estimates as a function of log(lambda)",
    x = "log(lambda)",
    y = "Coefficient estimate"
  )
```

Cross-validation to find the best lambda

```
avg.rmses = NULL
start.betas = rep(0.001, 13)

# Set up the datasets for cross-validation
folds = crossv_kfold(short.data, k = 5)

for (l in lambda.seq) {
  rmses = NULL
  for (k in 1:nrow(folds)) {

    train.idx = folds[k,1][[1]][[toString(k)]]$idx

    train = short.data[train.idx,]
    test = short.data[-train.idx,]

    train.X = train %>%
      dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
      dplyr::select(-diagnosis)
```

```
    train.y = train$diagnosis

    test.X = test %>%
      dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
      dplyr::select(-diagnosis)
    test.y = test$diagnosis

    LogLASSO = LogLASSO.CD(X = train.X, y = train.y,
                           beta = start.betas, lambda = l)
    LL.coefs = LogLASSO$coefficients
    rmse = sum(sqrt((test.y - as.matrix(cbind(1 * rep(1, nrow(test.X)), test.X)) %*% LL.coefs)^2))
    rmses = cbind(rmses, rmse)
  }
  avg.rmses = cbind(avg.rmses, mean(rmses))
  print(paste("iter: lambda = ", l, "done"))
}

plot.lambda = tibble(
  lambdas = lambda.seq,
  avg.test.MSE = c(avg.rmses),
  log.lambdas = log(lambda.seq)
) %>%
  arrange(-log.lambdas)

min.RMSE = min(plot.lambda$avg.test.MSE)
min.lambda = plot.lambda[which(plot.lambda$avg.test.MSE == min.RMSE),]$log.lambdas

ggplot(data = plot.lambda, aes(x = log.lambdas, y = avg.test.MSE)) +
  geom_line() +
  geom_vline(xintercept = min.lambda, alpha = 0.5, color = "red") +
  labs(
    title = "Average test MSE as a function of log(lambda)",
    x = "log(lambda)",
    y = "Average Test MSE"
  )
```

Tabulation

Assemble all models

```
# Fit all three models
start.betas = rep(0.001,13)
NR.fit = NewtonRaphson(xl.x, xl.y, logisticstuff, start.betas)
LL.fit = LogLASSO.CD(X = cbp.X, y = cbp.y,
                     beta = start.betas,
                     lambda = exp(min.lambda))
```

```
coeff.table = tibble(
  `Coefficient` = c("Intercept", "Mean radius", "Mean texture", "Mean perimeter",
                    "Mean smoothness", "Mean compactness", "Mean concavity",
                    "Mean concave points", "Mean fractal dimension", "Mean symmetry",
```

```
                    "Standard error radius", "Standard error perimeter", "Standard error symmetry"),
  `Newton-Raphson` = NR.fit[nrow(NR.fit), 2:ncol(NR.fit)],
  `Logistic-LASSO` = LL.fit$coefficients
)
knitr::kable(coeff.table)
```

Evaluating predictive ability

```
NR.coefs = glm(diagnosis ~ ., family = binomial(link = "logit"), data = short.data)$coefficients
```

```
start.betas = rep(0.001, 13)

# Set up the datasets for cross-validation
folds = crossv_kfold(short.data, k = 10)

NR.rmses = NULL
LL.rmses = NULL
for (k in 1:nrow(folds)) {
  train.idx = folds[k,1][[1]][[toString(k)]]$idx

  train = short.data[train.idx,]
  test = short.data[-train.idx,]

  train.X = train %>%
    dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
    dplyr::select(-diagnosis)
  train.y = train$diagnosis

  test.X = test %>%
    dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
    dplyr::select(-diagnosis)
  test.y = test$diagnosis

  LogLASSO = LogLASSO.CD(X = train.X, y = train.y, beta = start.betas, lambda = exp(min.lambda))
  LL.coefs = LogLASSO$coefficients

  LL.rmse = sum(sqrt((test.y - as.matrix(cbind(1 * rep(1, nrow(test.X)), test.X)) %*% LL.coefs)^2))
  LL.rmses = cbind(LL.rmses, LL.rmse)

  NR.rmse = sum(sqrt((test.y - as.matrix(cbind(1 * rep(1, nrow(test.X)), test.X)) %*% NR.coefs)^2))
  NR.rmses = cbind(NR.rmses, NR.rmse)
  }
```

```
a = tibble(
  `Logistic LASSO` = c(LL.rmses),
  `Newton-Raphson` = c(NR.rmses)
) %>%
  gather(., key = "model", value = "test.RMSE", `Logistic LASSO`:`Newton-Raphson`)

a %>%
  ggplot(data = ., aes(x = test.RMSE, fill = model)) +
  geom_density() +
```

```
  theme(legend.position = "bottom") +
  labs(
    title = "Distribution of test RMSE in 10-fold cross validation by model",
    x = "Test MSE",
    y = "Density"
  )
```

Appendix: Code

Christian Pascual, Xinyi Lin, Junting Ren

3/7/2019

```
library(tidyverse)
library(glmnet)
library(modelr)
library(matrixcalc)
set.seed(8160)
```

Data preparation

```
standardize = function(col) {
  mean = mean(col)
  stdev = sd(col)
  return((col - mean)/stdev)
}

# just standardize the covariates
standardized.data = read.csv(file = "breast-cancer-1.csv") %>%
  dplyr::select(radius_mean:fractal_dimension_worst) %>%
  map_df(.x = ., standardize)

# add back in the response and ids
data = cbind(read.csv(file = "breast-cancer-1.csv") %>% dplyr::select(diagnosis), standardized.data) %>%
  mutate(diagnosis = ifelse(diagnosis == "M", 1, 0))

needed.cols = c("diagnosis", "radius_mean", "texture_mean", "perimeter_mean",
                "smoothness_mean", "compactness_mean", "concavity_mean",
                "concave.points_mean", "fractal_dimension_mean",
                "symmetry_mean", "radius_se", "perimeter_se", "symmetry_se")

short.data = data %>%
  dplyr::select(which(colnames(data) %in% needed.cols))
```

Data setup

```
# Christian's data preparation
cbp.X = data %>%
  dplyr::select(radius_mean, texture_mean, perimeter_mean, smoothness_mean,
                compactness_mean, concavity_mean, concave.points_mean,
                fractal_dimension_mean, symmetry_mean, radius_se,
                perimeter_se, symmetry_se)

# Xinyi's data preparation
xl.x = data %>%
```

```
  mutate(intercept = 1) %>%
  dplyr::select(intercept, radius_mean, texture_mean, perimeter_mean,
                smoothness_mean, compactness_mean, concavity_mean,
                concave.points_mean, fractal_dimension_mean, symmetry_mean,
                radius_se, perimeter_se, symmetry_se) %>%
  as.matrix(.)

cbp.y = data$diagnosis
xl.y = as.matrix(data$diagnosis)

# Coefficients from regular glmnet
f = formula(diagnosis ~ radius_mean + texture_mean + perimeter_mean + smoothness_mean + compactness_mean

full.fit = glm(f, family = binomial(link = "logit"), data = data)
```

Functions needed for analysis

```
soft.threshold = function(beta, gamma) {
  new.beta = beta
    if (abs(beta) > gamma && beta > 0) {
      new.beta = beta - gamma
    } else if (abs(beta) > gamma && beta < 0) {
      new.beta = beta + gamma
    } else {
      new.beta = 0
    }
  return(new.beta)
}

calc.cur.p = function(intercept, data, betas) {
  # return n x 1 array of current probabilities evaluated with given betas
  return(
    exp(intercept * rep(1, nrow(data)) + data %*% betas) / (1 + exp(intercept * rep(1, nrow(data)) + da
  )
}

calc.working.resp = function(intercept, data, resp, betas, p) {
  # return n x 1 array of working responses evaluated with given betas
  return(
    intercept * rep(1, nrow(data)) + data %*% betas + (resp - p) / (p * (1 - p))
  )
}

calc.working.weights = function(p) {
  # return n x 1 array of working weights for the data
  return(p * (1 - p))
}

calc.obj = function(data, weights, w.resp, intercept, betas, lambda) {
  # return the objective function of data and current params
  return(
```

```
    log.lik = 1/(2 * nrow(data)) *
      sum((weights * (w.resp - intercept * rep(1, nrow(data)) - data %*% betas))^2) + lambda * sum(abs(
  )
}
```

Logistic LASSO implementation

```
LogLASSO.CD = function(X, y, beta, lambda, tol = 1e-5, maxiter = 1000) {

  ### Parameters: ##############################################################
  # X : design matrix                                                          #
  # y : response variable (should be binary)                                   #
  # beta : starting beta coefficients to start from                            #
  # lambda : constraining parameter for LASSO penalization                     #
  # tol : how precise should our convergence be                                #
  # maxiter : how many iterations should be performed before stopping          #
  ##############################################################################

  # Turn the betas into their own matrix
  X = as.matrix(X)

  # exclude the intercept from the input betas
  beta = as.matrix(beta[2:length(beta)])

  # Initialize important parameters before starting the coordinate descent
  beta0 = 1/length(y) * sum(y - X %*% beta)
  p = calc.cur.p(intercept = beta0, data = X, betas = beta)
  z = calc.working.resp(intercept = beta0, data = X, resp = y,
                        betas = beta, p = p)
  omega = calc.working.weights(p)
  obj = calc.obj(data = X, weights = omega, w.resp = z,
                 intercept = beta0, betas = beta, lambda = lambda)

  # Initialize the row for tracking each of these parameters
  path = c(iter = 0, intercept = beta0, obj = obj, beta)

    for (j in 1:maxiter) {

      prev.beta = beta

      # Coordinate descent
      for (k in 1:length(beta)) {
        r = y - (X %*% beta) + (X[,k] * beta[k])
        threshold.val = sum(omega * X[,k] * r)
        beta[k] = (soft.threshold(threshold.val, gamma = lambda)) / sum(omega * X[,k]^2)
      }

      # With new betas, recalculate the working parameters
      beta0 = mean(y) - sum(colMeans(X) * beta)
      p = calc.cur.p(intercept = beta0, data = X, betas = beta)
      z = calc.working.resp(intercept = beta0, data = X, resp = y,
```

```
                        betas = beta, p = p)
      omega = calc.working.weights(p)
      obj = calc.obj(data = X, weights = omega, w.resp = z,
                     intercept = beta0, betas = beta, lambda = lambda)

      # Append it to tracking matrix
      path = rbind(path, c(iter = j, intercept = beta0, beta, obj = obj))

      # Break the loop if the diff between likelihoods is below tolerance
      if (
        norm(prev.beta - beta, "F") < tol
        ) { break }
    }

  return(list(
    path = as.tibble(path),
    coefficients = c(beta0, beta))
    )
  }
```

Newton-Raphson implementation

```
logisticstuff <- function(x, y, betavec) {
  u <- x %*% betavec
  expu <- exp(u)
  loglik = vector(mode = "numeric", nrow(x))
  for(i in 1:nrow(x))
    loglik[i] = y[i]*u[i] - log(1 + expu[i])
  loglik_value = sum(loglik)
  # Log-likelihood at betavec
  p <- expu / (1 + expu)
  # P(Y_i=1|x_i)
  grad = vector(mode = "numeric", length(betavec))
  #grad[1] = sum(y - p)
  for(i in 1:13)
    grad[i] = sum(t(x[,i])%*%(y - p))
  #Hess <- -t(x)%*%p%*%t(1-p)%*%x
  Hess = hess_cal(x, p)
  return(list(loglik = loglik_value, grad = grad, Hess = Hess))
}

hess_cal = function(x, p) {
  len = length(p)
  hess = matrix(0, ncol(x), ncol(x))
  for (i in 1:len) {
    unit = x[i,] %*% t(x[i,]) * p[i] *(1 - p[i])
    #unit = t(x[i,])%*%x[i,]*p[i]*(1-p[i])
    hess = hess + unit
  }
  return(-hess)
}
```

```r
NewtonRaphson <- function(x, y, logisticstuff, start, tol = 1e-5, maxiter = 200) {
  i <- 0
  cur <- start
  stuff <- logisticstuff(x, y, cur)
  res = c(0, cur)
  #res <- c(0, stuff$loglik, cur)
  prevloglik <- -Inf      # To make sure it iterates
  #while(i < maxiter && abs(stuff$loglik - prevloglik) > tol && stuff$loglik > -Inf)
  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    i <- i + 1
    prevloglik <- stuff$loglik
    print(prevloglik)
    prev <- cur
    cur <- prev - solve(stuff$Hess) %*% stuff$grad
    stuff <- logisticstuff(x, y, cur)        # log-lik, gradient, Hessian
    res = rbind(res, c(i, cur))
    #res <- rbind(res, c(i, stuff$loglik, cur))
    # Add current values to results matrix
    }
  return(res)
}
modified <- function(x, y, logisticstuff, start, tol=1e-5, maxiter = 200){
  i <- 0
  cur <- start
  beta_len <- length(start)
  stuff <- logisticstuff(x, y, cur)
  res = c(0, cur)
  #res <- c(0, stuff$loglik,cur)
  prevloglik <- -Inf # To make sure it iterates
  while(i <= maxiter && abs(stuff$loglik - prevloglik) > tol)
  #while(i <= maxiter && abs(stuff$loglik - prevloglik) > tol && stuff$loglik > -Inf)
    { i <- i + 1
    prevloglik <- stuff$loglik
    prev <- cur
    lambda = 0
    while (is.negative.definite(stuff$Hess-lambda*diag(beta_len)) == FALSE) {
      lambda = lambda + 1
    }
    cur <- prev - solve(stuff$Hess-lambda*diag(beta_len)) %*% stuff$grad
    #cur <- prev + (diag(beta_len)/10)%*%(stuff$grad)
    #cur = prev + t(stuff$grad)%*%(stuff$grad)
    stuff <- logisticstuff(x, y, cur) # log-lik, gradient, Hessian
    res = rbind(res, c(i, cur))
    #res <- rbind(res, c(i, stuff$loglik, cur))
    }
  return(round(res,2))
}
```

Visualizations

betas vs lambda

```
# Generate data to visualize how the coefficients change with the logistic LASSO
lambda.seq = exp(seq(-7, 5, length = 500))
start.betas = rep(0.001,13)

coeff.path = NULL
for (l in 1:length(lambda.seq)) {
  fit = LogLASSO.CD(X = cbp.X, y = cbp.y, beta = start.betas, lambda = lambda.seq[l])
  coeff.path = rbind(coeff.path, c(lambda = lambda.seq[l], fit$coefficients))
  print(paste("Iter", l, "done", sep = " ")) # progress bar
}

#colnames(coeff.path) = c("lambda", paste("V", 1:13, sep = ""))
tidy.lambda = as.tibble(coeff.path) %>%
  gather(., key = "coeff", value = "coeff_est", V1:V13) %>%
  mutate(
    log.lambda = log(lambda)
  )

ggplot(data = tidy.lambda, aes(x = log.lambda, y = coeff_est, color = coeff, group = coeff)) +
  geom_line(alpha = 0.5) +
  theme(legend.position = "right") +
  labs(
    title = "Log-LASSO Coefficient estimates as a function of log(lambda)",
    x = "log(lambda)",
    y = "Coefficient estimate"
  )
```

Cross-validation to find the best lambda

```
avg.rmses = NULL
start.betas = rep(0.001, 13)

# Set up the datasets for cross-validation
folds = crossv_kfold(short.data, k = 5)

for (l in lambda.seq) {
  rmses = NULL
  for (k in 1:nrow(folds)) {

    train.idx = folds[k,1][[1]][[toString(k)]]$idx

    train = short.data[train.idx,]
    test = short.data[-train.idx,]

    train.X = train %>%
      dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
      dplyr::select(-diagnosis)
```

```
    train.y = train$diagnosis

    test.X = test %>%
      dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
      dplyr::select(-diagnosis)
    test.y = test$diagnosis

    LogLASSO = LogLASSO.CD(X = train.X, y = train.y,
                           beta = start.betas, lambda = l)
    LL.coefs = LogLASSO$coefficients
    rmse = sum(sqrt((test.y - as.matrix(cbind(1 * rep(1, nrow(test.X)), test.X)) %*% LL.coefs)^2))
    rmses = cbind(rmses, rmse)
  }
  avg.rmses = cbind(avg.rmses, mean(rmses))
  print(paste("iter: lambda = ", l, "done"))
}

plot.lambda = tibble(
  lambdas = lambda.seq,
  avg.test.MSE = c(avg.rmses),
  log.lambdas = log(lambda.seq)
) %>%
  arrange(-log.lambdas)

min.RMSE = min(plot.lambda$avg.test.MSE)
min.lambda = plot.lambda[which(plot.lambda$avg.test.MSE == min.RMSE),]$log.lambdas

ggplot(data = plot.lambda, aes(x = log.lambdas, y = avg.test.MSE)) +
  geom_line() +
  geom_vline(xintercept = min.lambda, alpha = 0.5, color = "red") +
  labs(
    title = "Average test MSE as a function of log(lambda)",
    x = "log(lambda)",
    y = "Average Test MSE"
  )
```

Tabulation

Assemble all models

```
# Fit all three models
start.betas = rep(0.001,13)
NR.fit = NewtonRaphson(xl.x, xl.y, logisticstuff, start.betas)
LL.fit = LogLASSO.CD(X = cbp.X, y = cbp.y,
                     beta = start.betas,
                     lambda = exp(min.lambda))
```

```
coeff.table = tibble(
  `Coefficient` = c("Intercept", "Mean radius", "Mean texture", "Mean perimeter",
                    "Mean smoothness", "Mean compactness", "Mean concavity",
                    "Mean concave points", "Mean fractal dimension", "Mean symmetry",
```

```
                    "Standard error radius", "Standard error perimeter", "Standard error symmetry"),
  `Newton-Raphson` = NR.fit[nrow(NR.fit), 2:ncol(NR.fit)],
  `Logistic-LASSO` = LL.fit$coefficients
)
knitr::kable(coeff.table)
```

Evaluating predictive ability

```
NR.coefs = glm(diagnosis ~ ., family = binomial(link = "logit"), data = short.data)$coefficients
```

```
start.betas = rep(0.001, 13)

# Set up the datasets for cross-validation
folds = crossv_kfold(short.data, k = 10)

NR.rmses = NULL
LL.rmses = NULL
for (k in 1:nrow(folds)) {
  train.idx = folds[k,1][[1]][[toString(k)]]$idx

  train = short.data[train.idx,]
  test = short.data[-train.idx,]

  train.X = train %>%
    dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
    dplyr::select(-diagnosis)
  train.y = train$diagnosis

  test.X = test %>%
    dplyr::select(which(colnames(short.data) %in% needed.cols)) %>%
    dplyr::select(-diagnosis)
  test.y = test$diagnosis

  LogLASSO = LogLASSO.CD(X = train.X, y = train.y, beta = start.betas, lambda = exp(min.lambda))
  LL.coefs = LogLASSO$coefficients

  LL.rmse = sum(sqrt((test.y - as.matrix(cbind(1 * rep(1, nrow(test.X)), test.X)) %*% LL.coefs)^2))
  LL.rmses = cbind(LL.rmses, LL.rmse)

  NR.rmse = sum(sqrt((test.y - as.matrix(cbind(1 * rep(1, nrow(test.X)), test.X)) %*% NR.coefs)^2))
  NR.rmses = cbind(NR.rmses, NR.rmse)
  }
```

```
a = tibble(
  `Logistic LASSO` = c(LL.rmses),
  `Newton-Raphson` = c(NR.rmses)
) %>%
  gather(., key = "model", value = "test.RMSE", `Logistic LASSO`:`Newton-Raphson`)

a %>%
  ggplot(data = ., aes(x = test.RMSE, fill = model)) +
  geom_density() +
```

```
  theme(legend.position = "bottom") +
  labs(
    title = "Distribution of test RMSE in 10-fold cross validation by model",
    x = "Test MSE",
    y = "Density"
  )
```