# 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() +
```

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