Logistic-LASSO Breast Cancer Classification Task

Ngoc Duong - nqd2000

3/25/2020

Data import and cleaning

```
breast_cancer_data = read.csv("./breast-cancer-1.csv")

bcdf = breast_cancer_data %>%
  mutate(diagnosis = ifelse(diagnosis == "M",1,0)) %>%
  dplyr::select(diagnosis, everything()) %>%
  dplyr::select(-id, -X)
```

Standardize design matrix (because although logistic is scale-invariant, LASSO is not, this is to ensure comparability of estimates by these different models)

```
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)
```

Investigate multicollinearity problem

```
p1 <- bcdf_fin %>% dplyr::select(-diagnosis) %>%  #filter only numeric variables
  cor() %>%
  #compute correlation matrix
  ggcorrplot(.,ggtheme = ggplot2::theme_gray,
             colors = c("#6D9EC1", "white", "#E46726"),
             tl.cex = 6)
ggsave("plot1.pdf", p1, width = 8.3, height = 6.3)
#We can see that there are some very strong correlations between certain variables
```

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 > 0.85) %>%
    slice(which(row_number() %% 2 == 0)) %>%
    pivot_longer(x:y) %>% dplyr::select(-r,-name) %>% distinct(value)
```

```r
#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

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

```
##
## Call:
## glm(formula = diagnosis ~ ., family = "binomial", data = full_data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.12112  -0.28937  -0.05735   0.19911   2.55257
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -1.52775    0.20654  -7.397 1.39e-13 ***
## smoothness_mean           1.57247    0.40552   3.878 0.000105 ***
## symmetry_mean            -0.11873    0.30203  -0.393 0.694245
## fractal_dimension_mean   -4.50615    0.53498  -8.423  < 2e-16 ***
## texture_se                0.80683    0.23051   3.500 0.000465 ***
## smoothness_se            -0.83043    0.29501  -2.815 0.004880 **
## compactness_se            0.39537    0.42498   0.930 0.352201
## concavity_se             -0.02998    0.34701  -0.086 0.931160
## concave.points_se         2.28564    0.36494   6.263 3.78e-10 ***
## symmetry_se              -0.20394    0.33909  -0.601 0.547557
## fractal_dimension_se     -0.75510    0.55886  -1.351 0.176651
## smoothness_worst          0.78588    0.48781   1.611 0.107174
## symmetry_worst            1.00913    0.43083   2.342 0.019166 *
## fractal_dimension_worst   2.83456    0.64071   4.424 9.68e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 751.44  on 568  degrees of freedom
## Residual deviance: 264.20  on 555  degrees of freedom
## AIC: 292.2
##
## Number of Fisher Scoring iterations: 7
```

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

Task 1

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

```r
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

```r
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)
  }
```

Test on dataset

```
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 = 0.3,
            y = full_data$diagnosis,
            X = as.matrix(Xmat_int),
            betavec = rep(1, ncol(Xmat_int)))
```

```
##     [,1]         [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## res    0 1.000000e+06    1    1    1    1    1    1    1     1     1     1
##        1 6.611112e-01    0    0    0    0    0    0    0     0     0     0
##        2 5.000000e-01    0    0    0    0    0    0    0     0     0     0
##        3 5.000000e-01    0    0    0    0    0    0    0     0     0     0
##     [,13] [,14] [,15] [,16]
## res     1     1     1     1
##         0     0     0     0
##         0     0     0     0
##         0     0     0     0
```

Check for convergence

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

Plot the path

```
p2 = path_df %>%
  pivot_longer(2:ncol(path_df),
               names_to = "Predictors",
               values_to = "estimate") %>%
  ggplot(aes(x = log(Tunegrid), y = estimate, group = Predictors, col = Predictors)) +
  geom_line() +
  labs(x = "Log(lambda)",
```

```
       y = "Coefficient Estimate")
ggsave("plot2.pdf", p2, width = 8.3, height = 6.3)
```

Cross validation

```
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
```

```
## # A tibble: 1 x 1
##      grid
##     <dbl>
## 1 0.00454
```

```
log(best.ll.lambda)
```

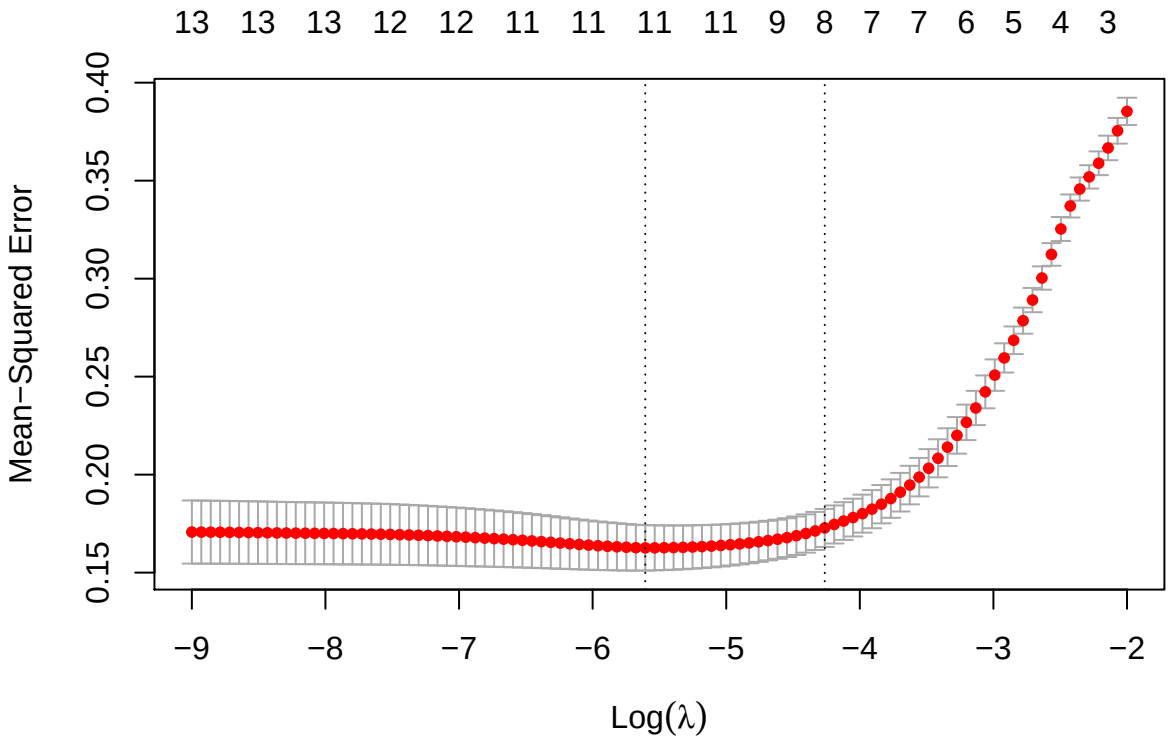
```
## # A tibble: 1 x 1
##    grid
##   <dbl>
## 1 -5.39
```

Visualize CV RMSE

```
p3 = cv_res %>% ggplot(aes(x = log(cv_res$grid), y = cv_res$mse)) +
  geom_errorbar(aes(ymin = cv_res$mse-cv_res$rmse.std.error,
                    ymax = cv_res$mse+cv_res$rmse.std.error),col = 1) +
  geom_line() + geom_point(size = 0.8, col = 4) +
  labs(x = "Log(lambda)", y = "Mean-Squared Error") +
  geom_vline(xintercept = as.numeric(log(best.ll.lambda)), col = 2) +
  geom_text(aes(x=as.numeric(log(best.ll.lambda)), label="log(lambda) = -5.4", y=0.22), col = 2, vjust =

ggsave("plot3.pdf", p3, width = 8.3, height = 6.3)
```

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)))

plot(cv.lasso)
```



```
cv.lasso$lambda.min
```

```
## [1] 0.00367552
```

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

```
## [1] -5.606061
```

```
#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

```
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

```
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
```

```
## [1] 0.07053378
```

Logistic-Lasso's MSE

```
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
```

```
## [1] 0.07272212
```

GLMNet's MSE

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

```
## [1] 0.07193173
```

Summary table

```
log_lasso = c(error = round(loglasso_error,4))
newton_raphson = c(error = round(newton_raph_error,4))
glmnet = c(error = round(glmnet_error,4))

table2 = cbind(newton_raphson, log_lasso, glmnet)

rownames(table2) <- c("MSE")
colnames(table2)[1:3] <- c("Newton-Raphson","Logistic LASSO","GLMnet")
knitr::kable(table2, escape = FALSE)
```

	Newton-Raphson	Logistic LASSO	GLMnet
MSE	0.0705	0.0727	0.0719

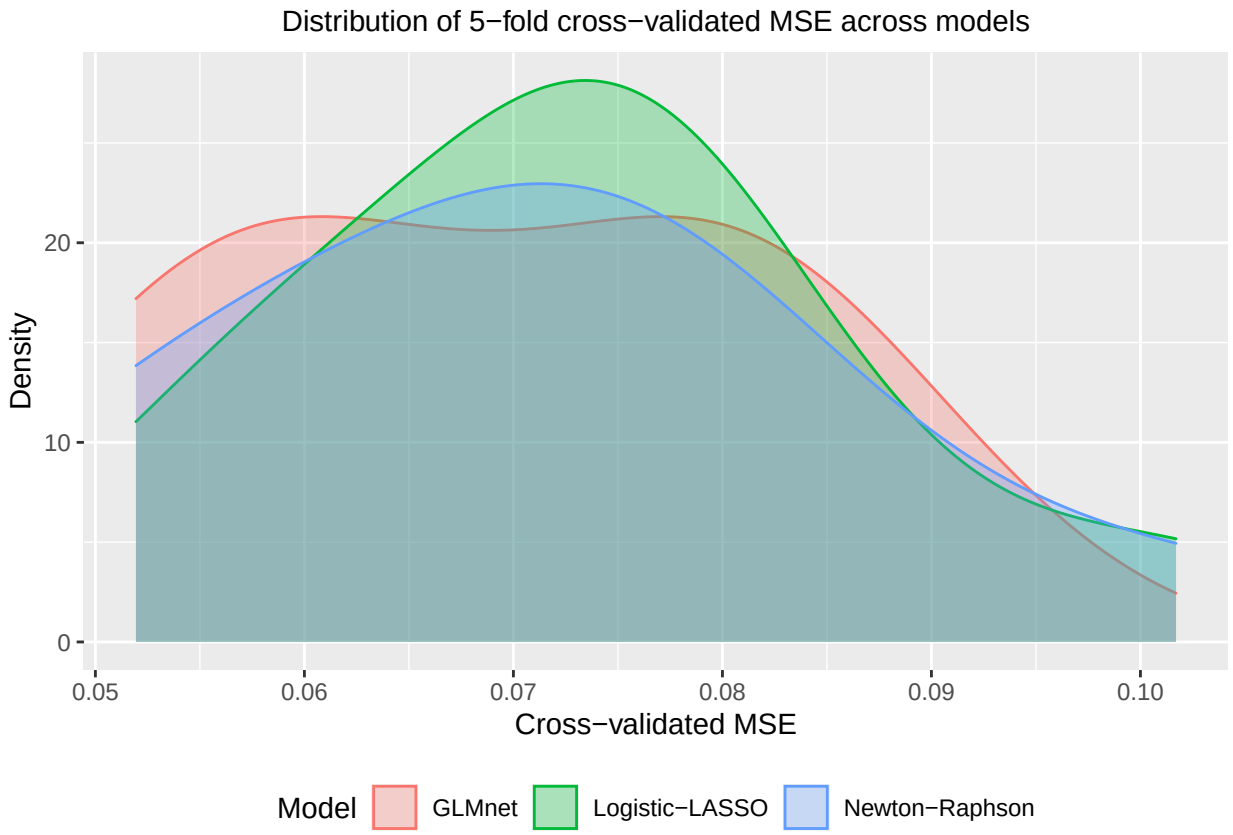
Cross-validation

```
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)
```

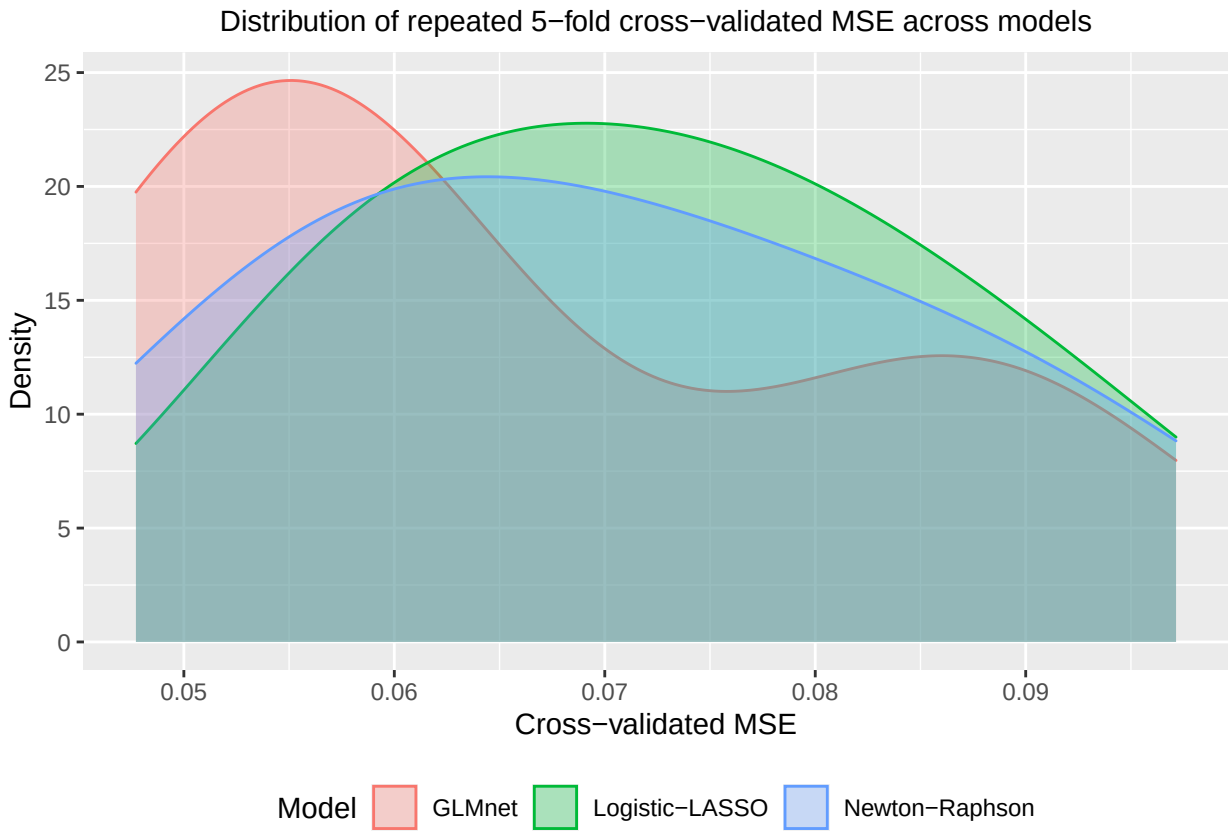
```
  glmnet_p = glmnet_expu/(1+glmnet_expu)
  glmnet_mse = mean((y_test - glmnet_p)^2) #cross-validated MSE for glmnet
  glmnet_mses = rbind(glmnet_mses, glmnet_mse)
  }
  res = tibble(`GLMnet`= glmnet_mse, `Logistic-LASSO` = ll_mses, `Newton-Raphson` = nr_mses)
  return(res)}

#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)
```

Visualize error comparisons

```
mse_comp_df %>%
  pivot_longer(1:3, names_to = "Model", values_to = "MSE") %>%
  ggplot(aes(x = MSE, col = Model, fill = Model)) +
  geom_density(adjust = 1.5, alpha = 0.3) +
  labs(title = "Distribution of 5-fold cross-validated MSE across models",
       x = "Cross-validated MSE",
       y = "Density") +
  theme(legend.position = "bottom",
        plot.title = element_text(size= 11, hjust = 0.5))
```




```
rep_mse_comp_df %>%
  pivot_longer(1:3, names_to = "Model", values_to = "MSE") %>%
  ggplot(aes(x = MSE, col = Model, fill = Model)) +
  geom_density(adjust = 1.5, alpha = 0.3) +
  labs(title = "Distribution of repeated 5-fold cross-validated MSE across models",
       x = "Cross-validated MSE",
       y = "Density") +
  theme(legend.position = "bottom",
        plot.title = element_text(size= 11, hjust = 0.5))
```



All coefficients by all model

```
table1 = cbind(glm_coeff_tib, t(nr_coeff_tib), glmnet_coeff_tib, loglasso_coeff_tib) %>% rename_at(2, ~

knitr::kable(table1, escape = FALSE)
```

	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

ROC curves

```
nr_pred_prob <- predict(log.mod, newdata = full_data, type = "response")

roc.glm <- roc(full_data$diagnosis, nr_pred_prob)
plot(roc.glm, legacy.axes = TRUE, print.auc = TRUE)
```