Computing Group Project 3

Muhire Kwizera, Quinton Neville, Zelos Zhu

April 2019

1 Introduction

With the declining costs of microarrays and advanced improvements in processing power, large datasets are becoming more abundant than ever to study the impact of genetic mutations on various disease states. In humans, an extra copy of chromosome 21 can lead to down syndrome, a commonly studied genetic disorder that causes developmental and cognitive delays. In this analysis, we investigate the relationship between down syndrome in mice and protein modifications that produces detectable signals in the nuclear fraction of the cortex. We built and compared the pathwise coordinate-descent optimization algorithm with LASSO regularization and a bootstrap smoothing method proposed by Efron[1] for the estimation of logistic regression covariates to classify diseased and non-diseased individuals.

2 Methods

2.1 Data

The data in this study consisted of down syndrome diagnoses for 1080 mice and 79 variables. The outcome of interest, down syndrome diagnosis, was binary (diseased or non-diseased) and the predictors (continuous) were 77 various proteins' expression levels. The last variable was a unique identification number for each mouse.

Three observations, mouse ID's: 3426_13, 3426_14, 3426_15 were removed from the analysis due to too much missing data, >50% observations, (See Figure 1). Nine proteins (BCL2, H3MeK4, BAD, EGR1, H3AcK18, pCFOS, Bcatenin, MEK, ELK) were removed for similar reasons but each one of these proteins was highly correlated with *at least one* other protein that was retained in the final dataset (See Figure 2). Our final dataset included 1077 observations and 70 predictor variables, 68 of which were proteins. The data was then scaled and randomly split into 862 observations for the training set and 215 observations

[1]Bradley Efron (2014) Estimation and Accuracy After Model Selection, Journal of the American Statistical Association, 109:507, 991-1007, DOI: 10.1080/01621459.2013.823775

for the testing set, reflecting an approximate eighty-twenty partition of the data, and were used throughout all following statistical analyses.

2.2 Statistical Analysis

We initially explored associations between protein expressions in mice and the observance of down-syndrome, employing boxplots, inter-correlation visualizations, and summary statistics to investigate these data. Additionally, visualizations were created to investigate distributions for each protein expression variable and examine the high levels of inter-correlation between these protein expression covariates.

Ultimately, the large degree of correlation in these data (See Figures 1-3) motivated the need for a variable selection algorithm to minimize the effect of or remove certain protein expressions, in order to build a robust binary classification model for down-syndrome by protein expression. Here, we utilized a pathwise-coordinate descent LASSO logistic regression algorithm, both hand-built and built-in **R**, as well as a bootstrap smoothing method for estimation of the coefficients and subsequent standard error from the LASSO regression. With correct predictive classification of down syndrome as the goal, these models were optimized and their respective diagnostic capabilites were compared through misclassification rate, ROC AUC, and cost analyses.

Furthermore, the bootstrap-smoothing method allowed for the construction of confidence intervals around the coefficient estimates, motivating analogous inference to the usual frequentist confidence intervals. If the smoothed interval, essentially a 95% confidence interval, contained zero it would be considered insignificant. All computations were performed in the R statistical software.

2.3 Models

The following logistic models (1) - (2), were fit to the 862 training observations, eliciting predicted probabilities of binary classification, which were ultimately utilized with an optimized probability threshold to categorize these sample of mice into controls (no down-syndrome) or cases (down-syndrome). Y_i and x_i were the i^{th} mouse's disease state and vector of predictors respectively.

$$Y_i \sim \text{Bernoulli}(p_i) \tag{1}$$

$$F(\mathbf{x_i}^T\beta) = P(Y_i = 1|\beta) = \frac{\mathrm{e}^{\mathbf{x}_i^T\beta}}{1 + \mathrm{e}^{\mathbf{x}_i^T\beta}} = p_i \tag{2}$$

$$\mathcal{L}(Y_i|\beta) = \sum_{i=1}^{n} \left(Y_i \left(\mathbf{x}_i^T\beta \right) - log \left(1 + \mathrm{e}^{\mathbf{x}_i^T\beta} \right) \right) \tag{3}$$

We implemented the pathwise coordinatewise optimization algorithm, built in the previous project, which optimized model estimates one-by-one while holding all the others fixed. The algorithm minimized an objective function which was the additive inverse of the log likelihood (3). Convergence was defined as

both a small euclidean distance between successive objective functions and a small euclidean distance between successive model estimates, within tolerance. Tolerance was preset to the square root of the machine's floating-point precision.

We included a LASSO constraint to perform regularization in the pathwise coordinatewise optimization algorithm, and the resulting objective function was

$$g(\beta) = -\mathcal{L}(\beta_0, ..., \beta_p) + \lambda \sum_{j=1}^{p} |\beta_j| \tag{4}$$

where $\mathcal{L}(\beta_0, ..., \beta_p)$ was the log likelihood in (3), λ was a tuning parameter, and p was the number of predictors in the model. We obtained a path of solutions for λ values between 0 and the largest estimate from the full model, and we determined the optimal λ value by 5-fold CV based on misclassification rate.

Furthermore we built and implemented a bootstrap-smoothing approach to estimate the model parameters, $(\beta_0, ..., \beta_p)$, one for each protein/predictor, β_j; a standard deviation for each parameter was calculated using methods proposed by Efron so that we could construct the *smoothed interval*:

$$\beta_j \pm 1.96\tilde{sd}_j$$

For each bootstrap, we applied a logistic lasso model using 5-fold cross validation from the *glmnet* package, providing us with bootstrap coefficient estimates β_j^*'s for each protein; this was done for 1000 bootstraps. The mean among the 1000 bootstraps of each β_j^* provide us with our desired final parameter estimate, β_j.

The standard deviation of each parameter estimate is as follows:

$$\tilde{sd}_j = [\sum_{l=1}^{n} \hat{cov}_l^2]^{\frac{1}{2}}$$

where

$$cov_l = \sum_{k=1}^{n} (Y_{kl}^* - Y_{\cdot l}^*)(t_k^* - t_\cdot^*)/1000$$

where Y_{kl}^* is a vector corresponding to the number of times each observation was chosen during each individual bootstrap sample. $Y_{\cdot l}^*$ corresponds to the mean of Y_{kl}^* across the total number of bootstraps. t_k^* corresponds to the kth bootstrap of the statistic of interest, so each β_j^* bootstrap estimate. $t_\cdot^*$ corresponds to the mean among the t_k^*, so in this case our β_j.

We compared the optimal model obtained by the LASSO regularization and the bootstrap-smoothing model in terms of their estimated model parameters using the training set and in terms of their prediction accuracy (quantified by their misclassification rate), predicted probability distributions, Receiver Operator Curves, and Cost Analysis; utilizing the test data set. We also investigated the use of the deviance metric for choosing the optimal built-in LASSO and bootstrap-smoothing models. The smoothed interval provided from the bootstrap gave us insight to whether or not a particular parameter estimate was statistically significant.

3 Results

3.1 Prediction

Drawing motivation from the exploratory analysis visualizations, we were able to identify roughly 16 (correlated) protein expressions that could potentially be driving associations with down-syndrome in these mice, shown in Figure 3. Considering these original data, 507 cases of mice were observed to meet clinical criterion for down syndrome classification out of 1077 (47%).

The optimal model estimates from the built-in R LASSO, coordinate descent LASSO and bootstrap-smoothing methods are in figure 4. They are based on minimizing the cross-validated misclassification error rate. The plot also displays 95% confidence intervals based on the bootstrap-smoothing method. Figure 5 shows the proportions of bootstraps for which variables were shrunk to 0 while figure 6 displays the distributions of optimal LASSO tuning parameters based on bootstrap-smoothing (the red curve). The optimal LASSO tuning parameters for the built-in R LASSO and coordinate descent LASSO methods were $\lambda = 0.0022$ and $\lambda = 0.0011$ respectively. The built-in R LASSO, coordinate descent LASSO, and bootstrap-smoothing methods had respective test misclassification rates of 4.1%, .6%, and 3.7%.

The bootstrap-smoothing (or bagging) method with **R** packages *caret* & *glmnet* LASSO models elicited substantially different point estimates for the associated coefficient estimates and optimal LASSO tuning parameter values by differing optimization schemes (Figure 6). , The former used accuracy as a metric and the latter used the area under the receiver operating curve (ROC).

We observed higher separation between low vs. high probability prediction estimates of disease in the smoothing method versus the LASSO. This induced a differential optimal misclassification rate probability threshold between the models, with $p = 46\%, 59\%$ in the smoothing versus LASSO methods respectively (Figure 8). With respect to the diagnostic ability of the models, both elicited extremely sharp Receiver Operator Curves with the area under the curves estimated at approximately 99%, meaning as diagnostics neared a 100% true positive rate there was still a negligible false positive rate (Figure 6). Furthermore, with respect to subjective cost analysis, utilizing a False Positive and False Negative adverse matrix respectively, the smoothing method reached optimal threshold at a 1% higher threshold than the LASSO for the former, while the elicited identical results in the later scenario (Figures 7, 8).

4 Conclusion

Based on the misclassification rate optimization criterion the bootstrap-smoothing method performed better than both the built-in R LASSO and coordinate descent LASSO algorithms.

Considering the results the bootstrap smoothing interval estimation and analogous hypothesis testing, further investigation into scientific literature ap-

peared to corroborate our statistical findings for some of the predictors. It is a well supported hypothesis that ITSN1 is a highly *clinically* significant positively associated predictor of down syndrome, as this was a commonly found result across literature[2]. Similarly, published research was found for APP[3] and ERK[4], providing further evidence to support the significant positive and negative associations with down syndrome we observed, respectively. No conclusive supporting evidence was found to support our findings regarding TRKA, however there has been conjectured linkages between degeneration of this protein process and other illnesses in mice, and may speculatively be an indicator of reduced nervous system functioning processes rather than down syndrome specifically.

The built-in LASSO and bootstrap-smoothing results based on the misclassification rate and area optimization under the ROC curve were substantially different. We reported optimal estimates and confidence based on misclassification rate for the built-in LASSO, coordinate descent LASSO and bootstrap-smoothing methods as this metric was simpler to compute for the coordinate descent algorithm. We also reported the comparisons of the optimal deviance-based built-in LASSO and bootstrap-smoothing models.

The main limitation of this project was that since the coordinate LASSO was very slow it was not used with bootstrap-smoothing. The results comparing the bootstrap-smoothing to coordinate descent algorithms may, therefore, not reflect the complete truth. Furthermore there were convergence issues with both the coordinate LASSO and bootstrap-smoothing algorithms that might have affected the coordinate descent LASSO method more than the bootstrap-smoothing method. Finally, we did not compute the optimal number of bootstraps for the bootstrap-smoothing method and our standard error estimates were not bias-corrected.

[2]Hunter MP, Nelson M, Kurzer M, et al. Intersectin 1 contributes to phenotypes in vivo: implications for Down's syndrome. Neuroreport. 2011;22(15):767–772. doi:10.1097/WNR.0b013e32834ae348

[3]Trazzi S, Fuchs C, Valli E, Perini G, Bartesaghi R, Ciani E. Further, the amyloid precursor protein (APP) triplicated gene impairs neuronal precursor differentiation and neurite development through two different domains in the Ts65Dn mouse model for Down syndrome. J Biol Chem. 2013;288(29):20817–20829. doi:10.1074/jbc.M113.451088

[4]Cramer N, Galdzicki Z. From abnormal hippocampal synaptic plasticity in down syndrome mouse models to cognitive disability in down syndrome. Neural Plast. 2012;2012:101542. doi:10.1155/2012/101542

5 Appendix

5.1 Figures

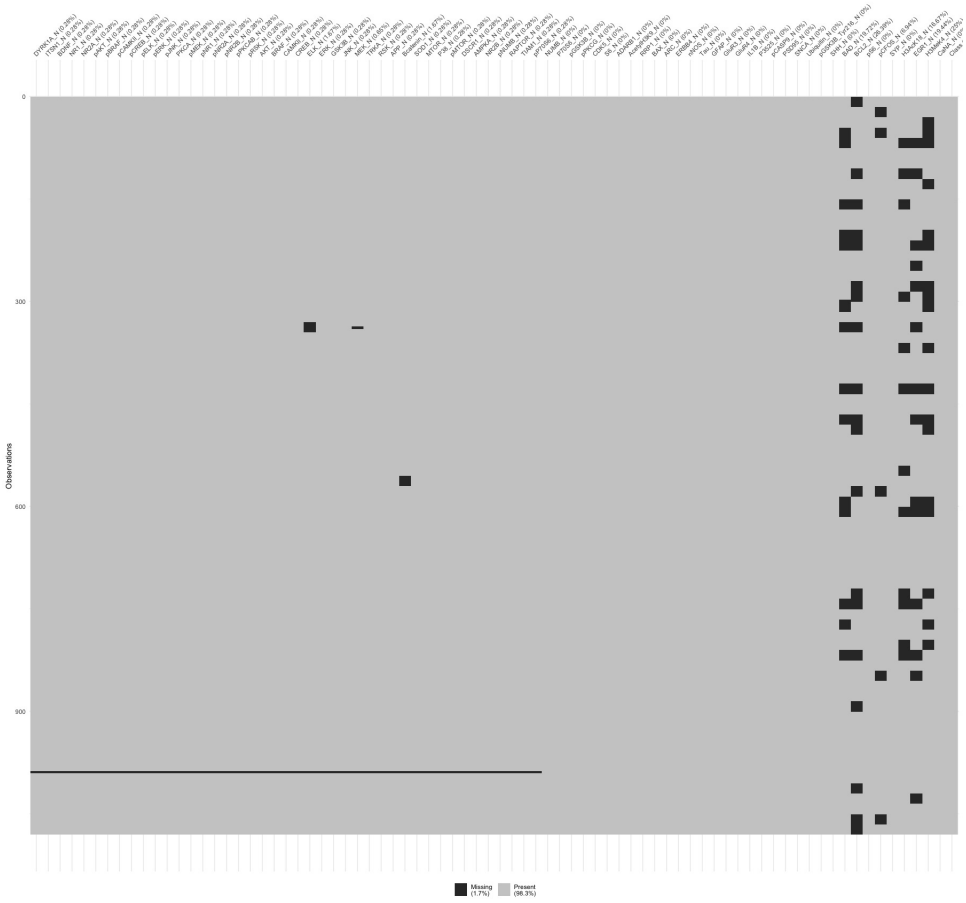


Figure 1: Graphical display of missing data. Rows correspond to each mouse, and each columns corresponds to a protein.

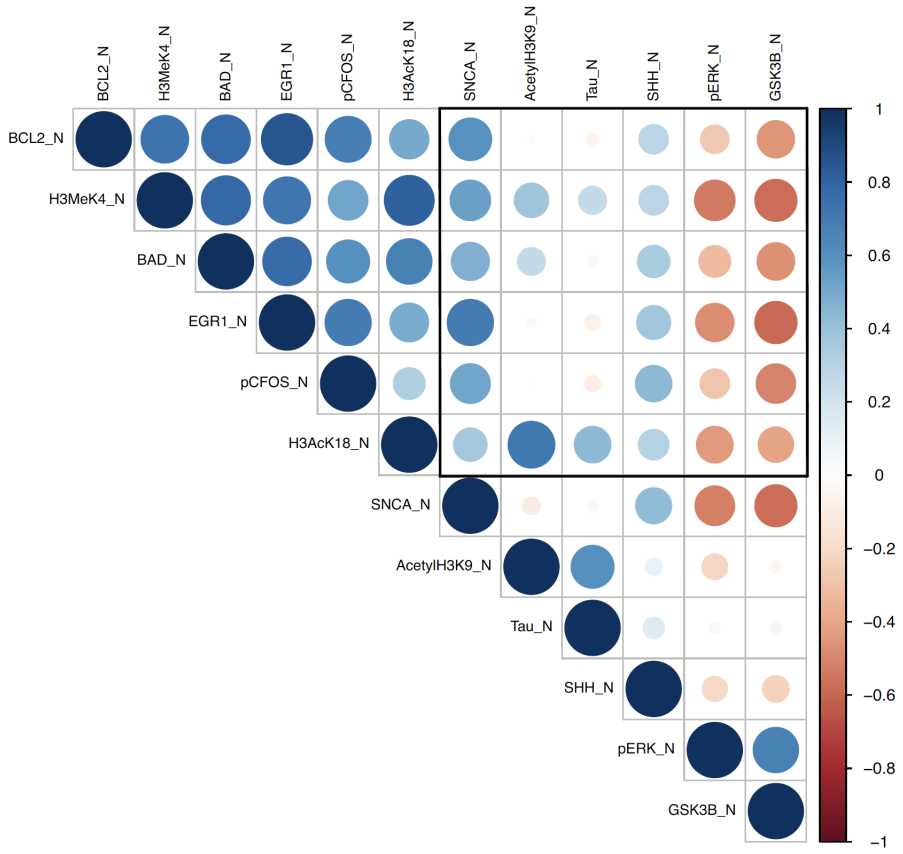


Figure 2: Correlation plot showing a group of proteins that were highly correlated with the proteins we removed

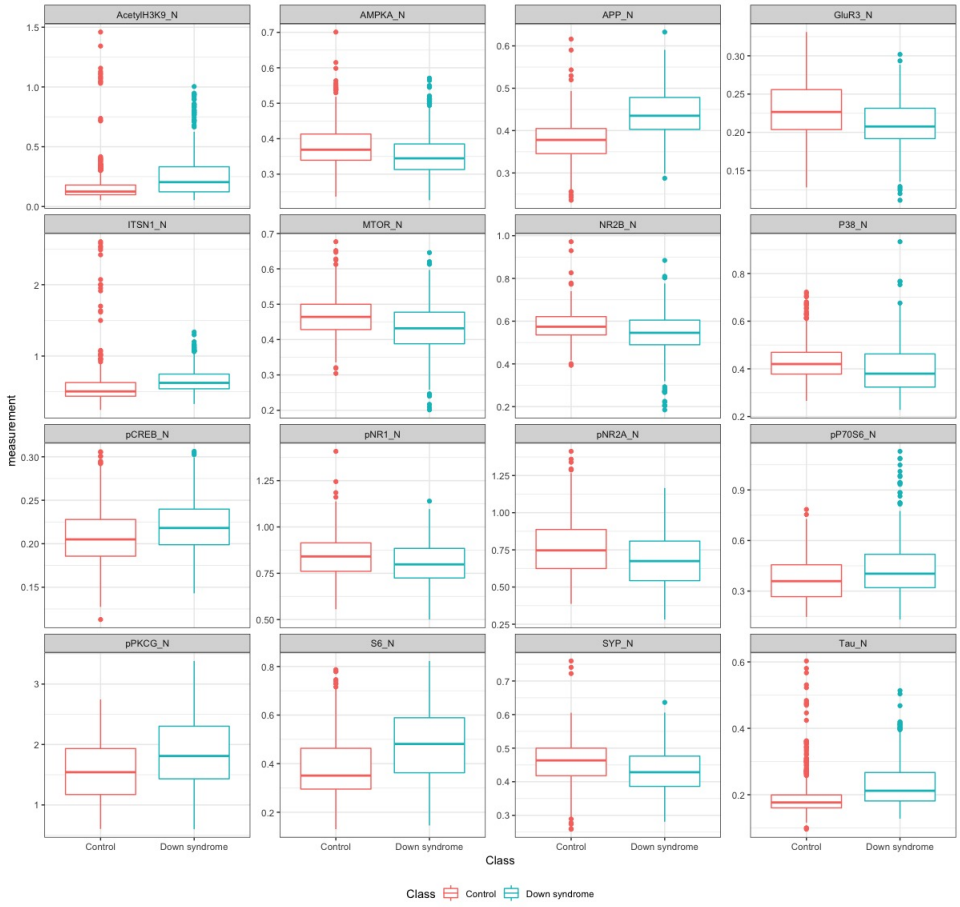


Figure 3: Top 16 most significantly different proteins between diseased and non-diseased based on p-value

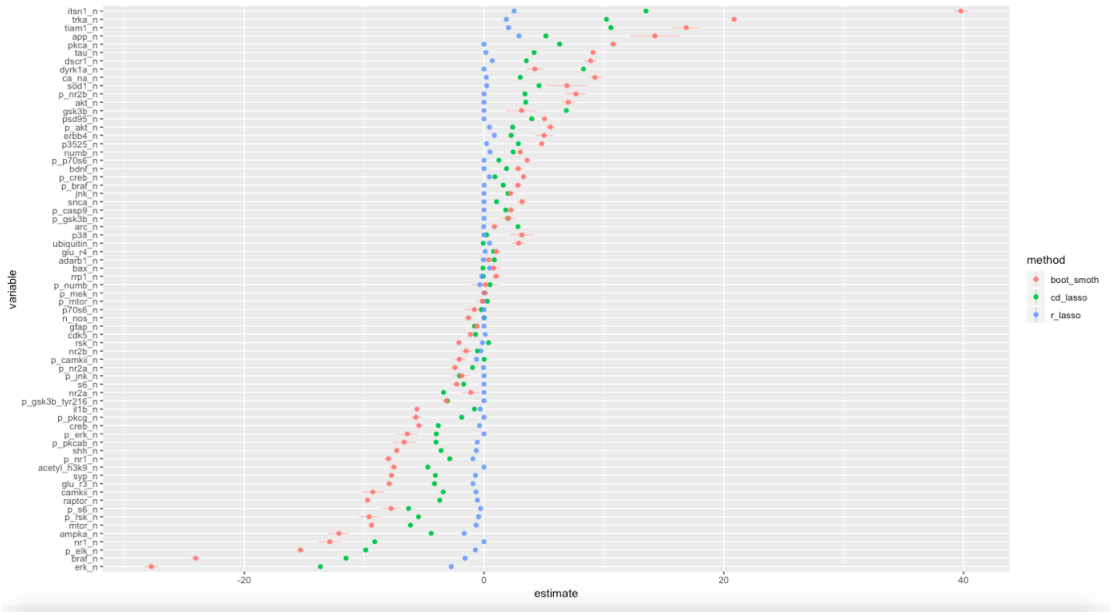


Figure 4: Parameter estimates for the built-in LASSO (blue), coordinate descent LASSO (green) and bootstrap smoothing (red). The lines represent 95% confidence intervals. The results are based on the misclassification error rate.

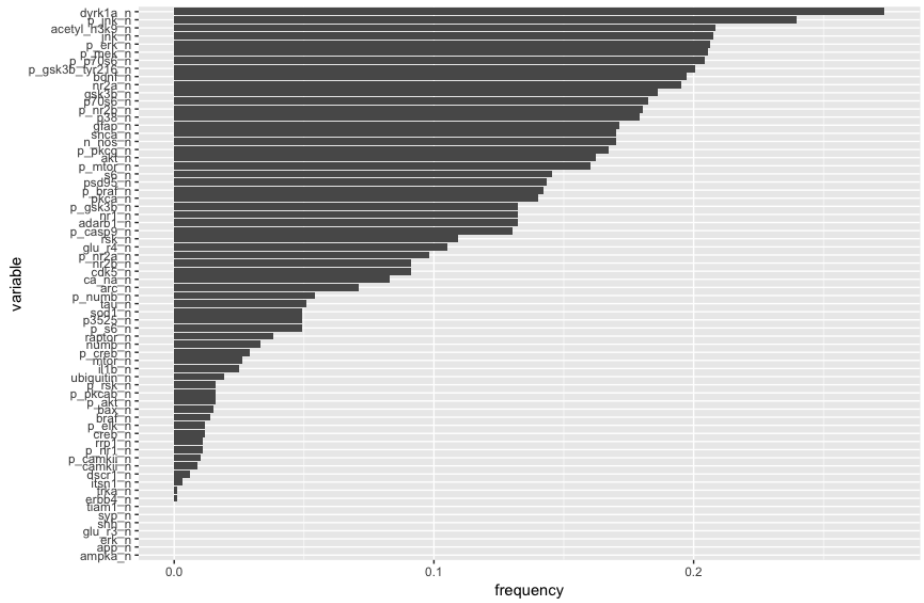


Figure 5: Proportion of bootstraps for which variables were shrunk to 0

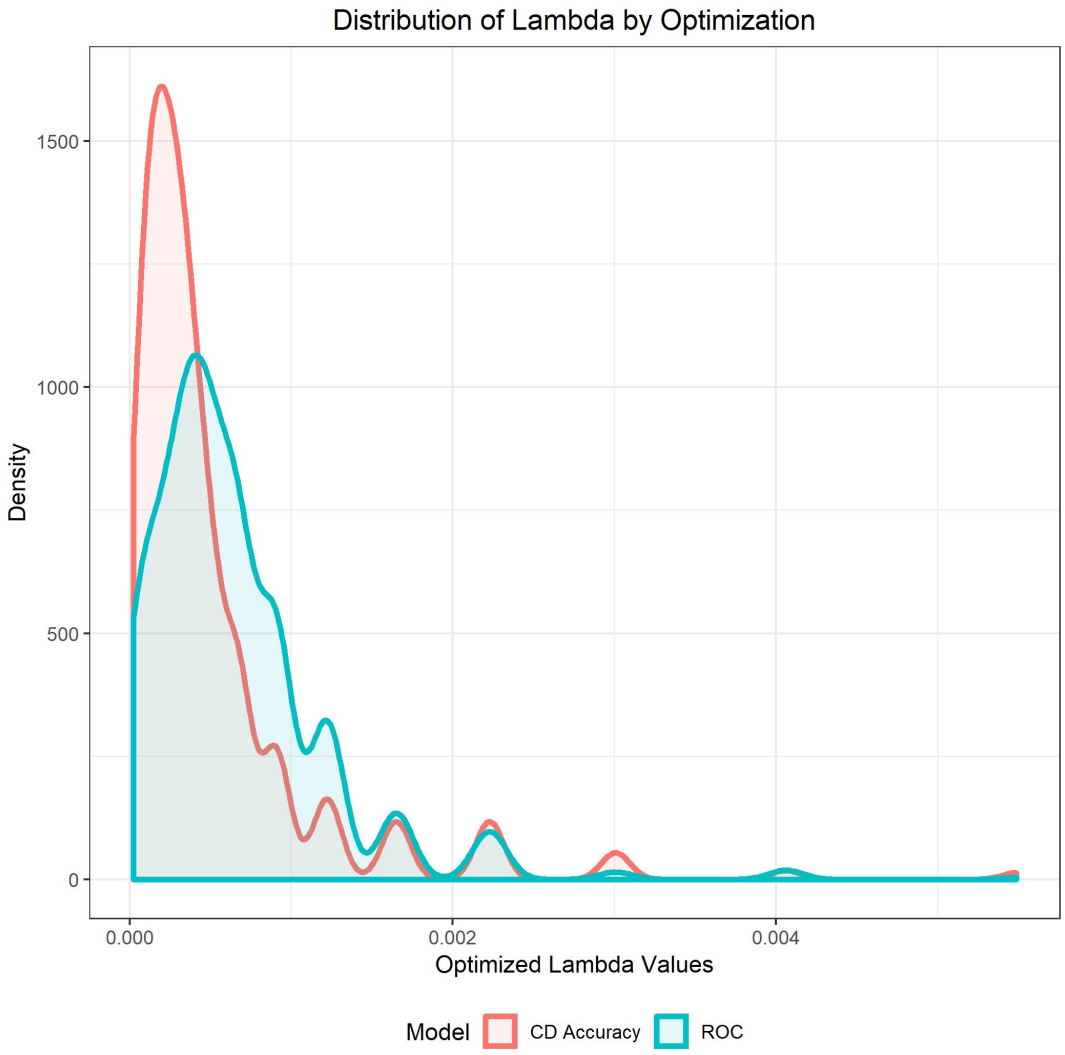


Figure 6: Lambda values across smoothing estimates for accuracy optimization versus ROC

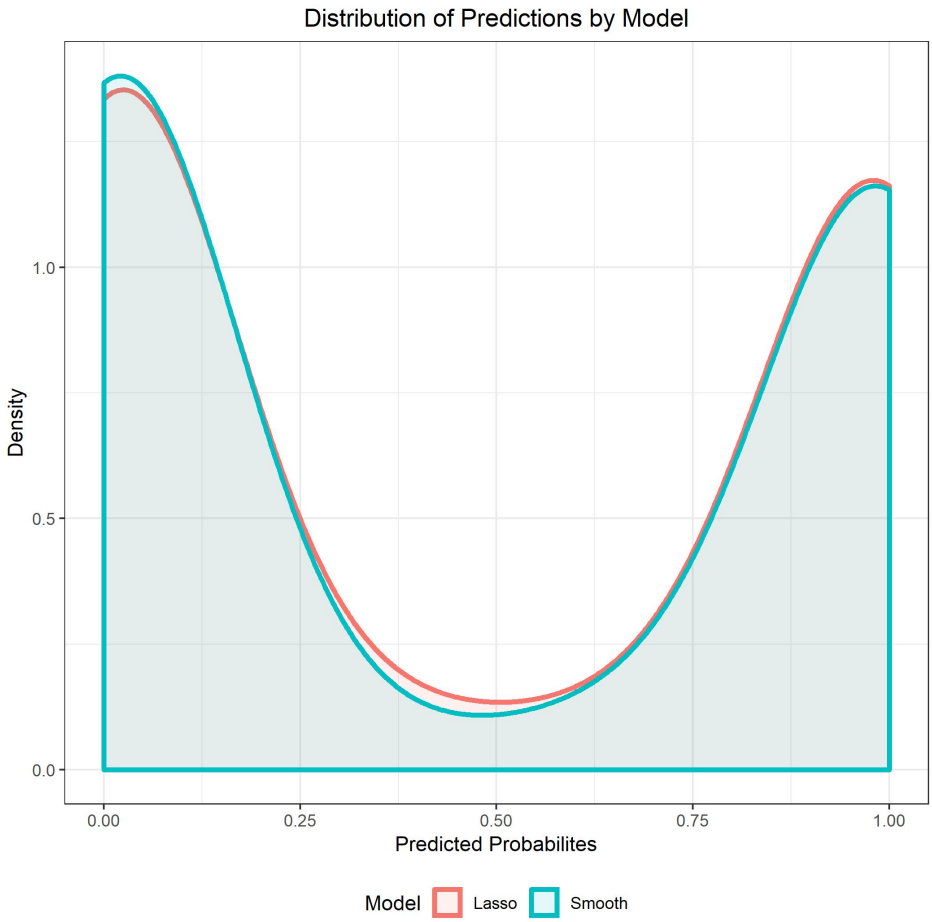


Figure 7: Distribution of predicted probabilities by model

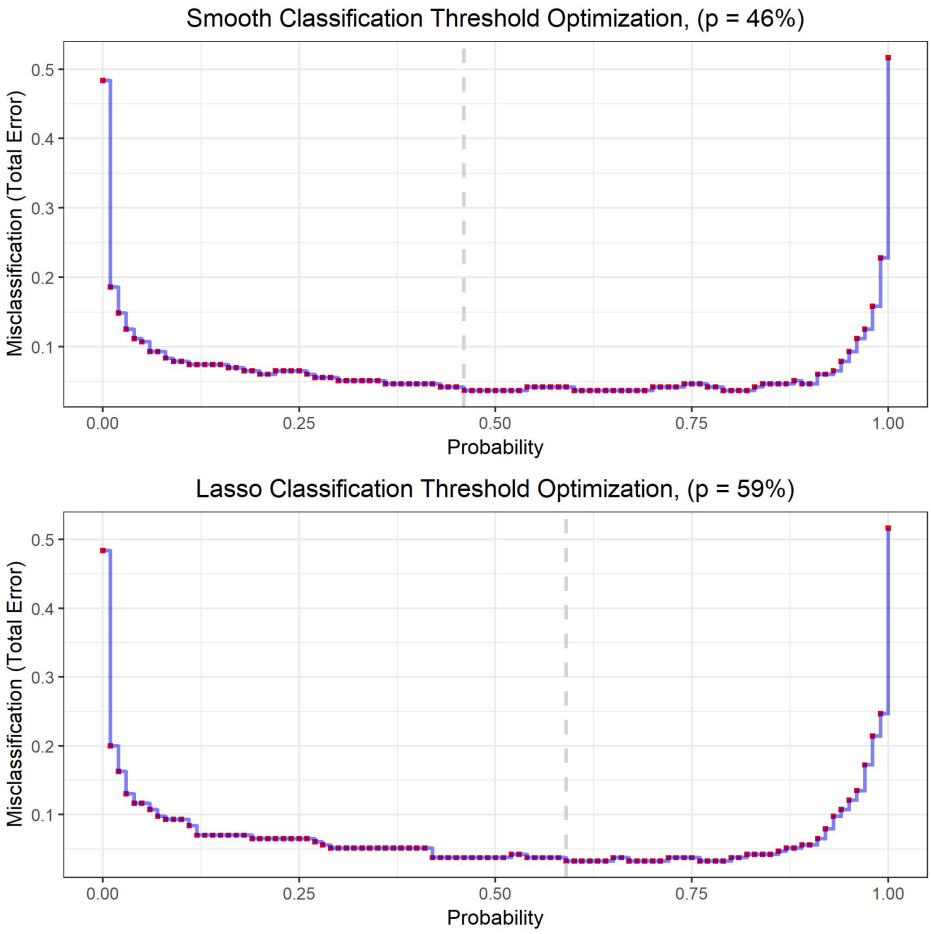


Figure 8: Optimal probability thresholds by model from misclassificaition

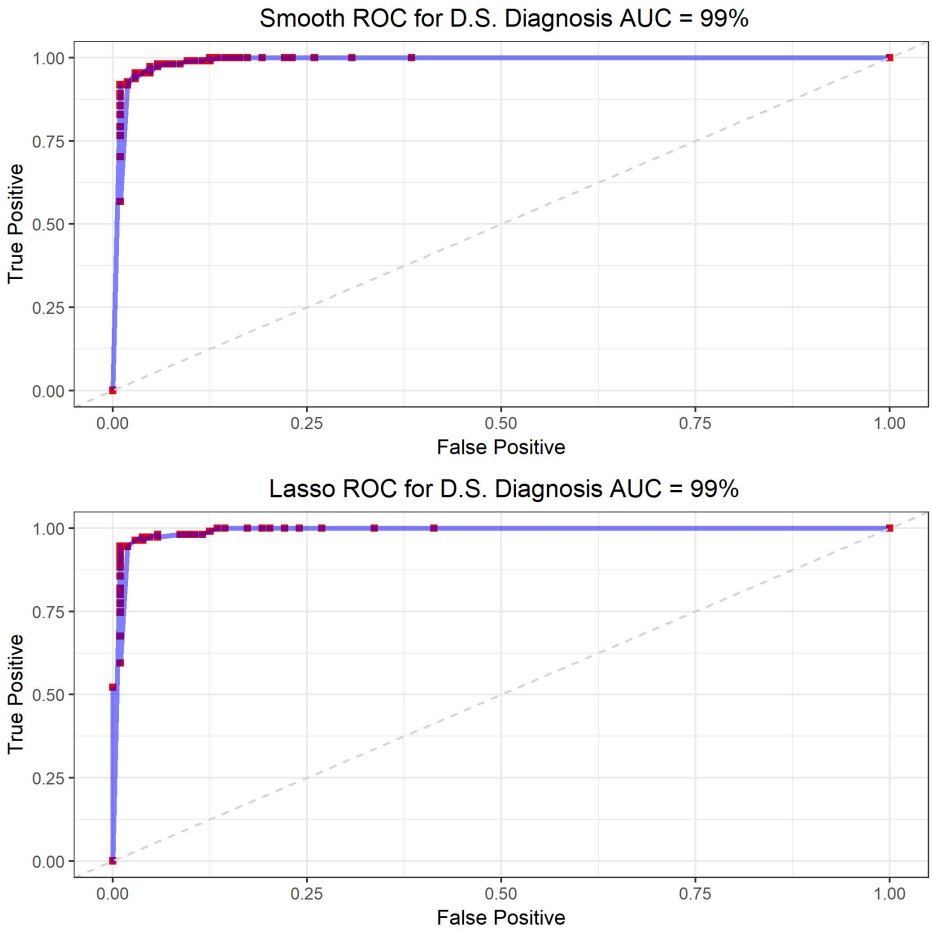


Figure 9: Receiver Operator Curves and associated AUC values by model

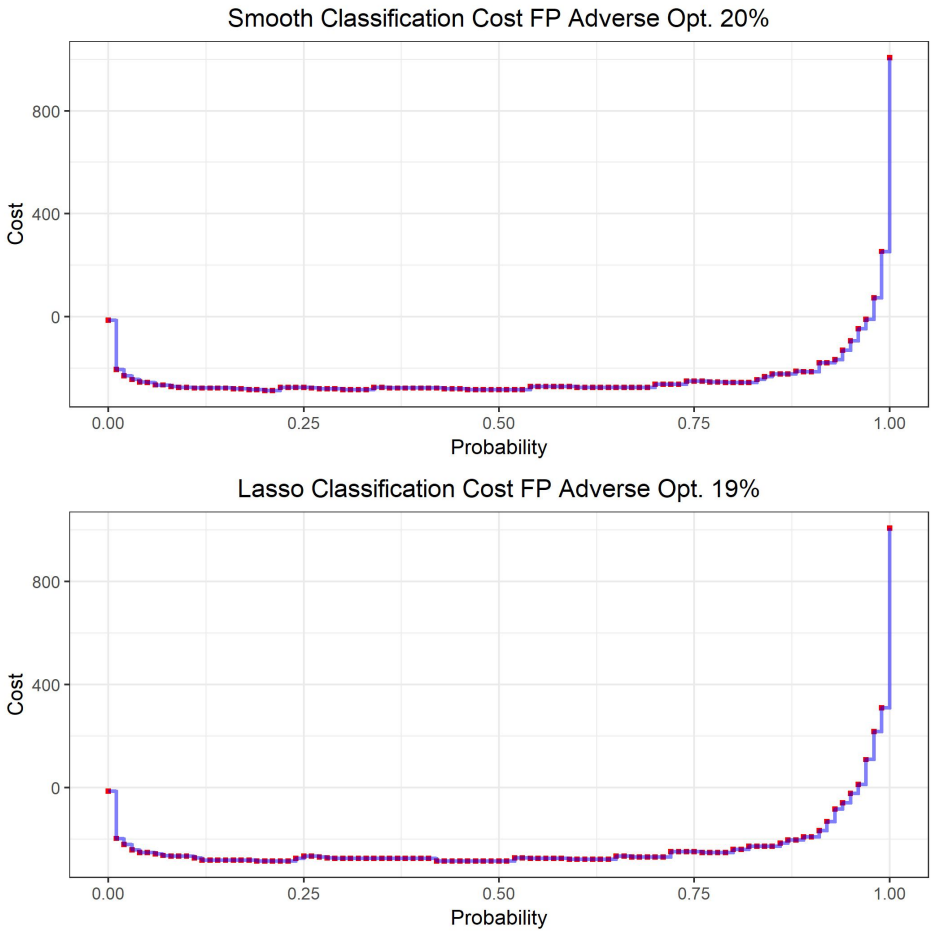


Figure 10: Optimal probability threshold for a false positive-adverse cost matrix

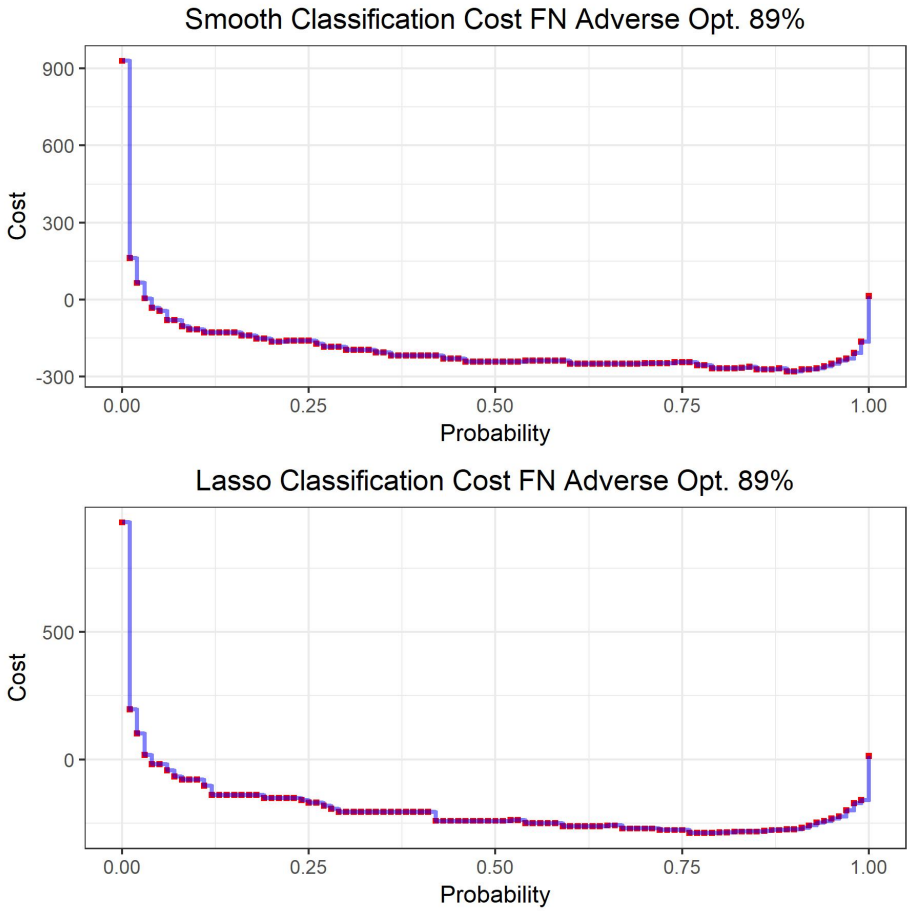


Figure 11: Optimal probability threshold for a false negative-adverse cost matrix

6 Code

EDA_group_proj3

Muhire Kwizera, Quinton Neville, Zelos Zhu

4/16/2019

EDA

```
library(tidyverse)
library(knitr)
library(corrplot)
library(naniar)
library(multtest)

down.df <- read_csv("Down.csv") %>%
  filter(., !(MouseID %in% c("3426_13", "3426_14", "3426_15"))) %>%
  dplyr::select(., -c("BCL2_N", "H3MeK4_N", "BAD_N", "EGR1_N", "H3AcK18_N", "pCFOS_N", "Bcatenin_N", "M

# 1077 observations --- #IDs - "3426_13", "3426_14", "3426_15" are missing a lot
nobs = down.df %>%
  nrow()

# 79 variables - 1 is ID, 1 is disease status, several are highly correlated with one another, remove t
#BCL2_N, H3MeK4_N , BAD_N , EGR1_N , H3AcK18_N , pCFOS_N , Bcatenin_N , MEK_N , ELK_N -- 68 after
nvars = down.df %>%
  ncol()

# IDs are unique
nid = down.df %>%
  pull(MouseID) %>%
  unique()

#consider scrapping IDs: 3426_13, 3426_14, 3426_15
down.df %>%
  dplyr::select(-c(MouseID, Class)) %>%
  apply(., 1, function(x) sum(is.na(x))) %>%
  View()

#NA's per column - BCL2_N, H3MeK4_N, BAD_N, EGR1_N, H3AcK18_N, pCFOS_N each have a lot of missing value
#Removing Bcatenin_N, MEK_N, ELK_N makes our data super clean, rid of NA's
#Each of these genes have a respective multicollinear variable associated to it

down.df %>%
  dplyr::select(-c(MouseID, Class)) %>%
  apply(., 2, function(x) sum(is.na(x))) %>%
  View()

#Show pair-wise correlations between genes -- these should be shrunk by lasso
down.df %>%
  dplyr::select(-c(MouseID, Class)) %>%
  cor(., use = "pairwise.complete.obs") %>%
  as.data.frame %>%
```

```
  rownames_to_column(var = 'var1') %>%
  gather(var2, value, -var1) %>%
  filter(., abs(value) > .50 & var1 != var2) %>%
  arrange(desc(value)) %>%
  View()

vis_miss(down.df %>% dplyr::select(-c(MouseID)))

#Since we removed these columns, reread in with them
read_csv("Down.csv") %>%
  dplyr::select(BCL2_N, H3MeK4_N , BAD_N , EGR1_N,  pCFOS_N, H3AcK18_N, SNCA_N, AcetylH3K9_N, Tau_N, SH
  cor(method = "spearman", use = "pairwise.complete.obs") %>%
  corrplot(type = "upper", order = "hclust", tl.col = "black", tl.cex = 0.75)
```

```
# mean
meanvals = down.df %>%
  dplyr::select(-c(mouse_id, class)) %>%
  apply(., 2, function(x) mean(x, na.rm = TRUE))

# std dev
sdvals = down.df %>%
  dplyr::select(-c(mouse_id, class)) %>%
  apply(., 2, function(x) sd(x, na.rm = TRUE))

# descriptive stats
mean_sd = cbind(meanvals, sdvals) %>%
  as.tibble %>%
#  mutate(interpretation_factor = sdvals - meanvals) %>%
  round(., 3) %>%
  mutate(variable = names(down.df)[2:69]) %>%
  dplyr::filter(variable %in% c("itsn1_n","erk_n","app_n","trka_n"))

mean_sd %>%
  head() %>%
  kable()

#Shows data needs to be scaled
boxplot(down.df[,-c(1,70)])
boxplot(scale(down.df[,-c(1,70)]))
```

```
# proportion of malignant breast cancer diagnosis
prop_malign = down.df %>%
  mutate(Class = ifelse(Class == "Down syndrome", 1, 0)) %>%
  pull(Class) %>%
  # sum()
  mean()

#which proteins should be best indicators
multtest_structure <- t(as.matrix(down.df[,-c(1,70)]))
indicator <- down.df[,70] %>% pull(Class) %>% as.factor %>% as.numeric()
indicator <- indicator - 1

teststat <- mt.teststat(multtest_structure, indicator, test = "t", nonpara = "n")
names(teststat) <- names(down.df[,2:69])
```

```r
rawp = 2 * (1 - pnorm(abs(teststat)))
top15_sig_proteins <- names(sort(rawp))[1:16]

down.df %>%
  gather(., key = "protein", value = "measurement", vars) %>%
  filter(protein %in% top15_sig_proteins) %>%
  ggplot(., aes(x = Class, y = measurement, col = Class)) +
  geom_boxplot() +
  facet_wrap(~protein, ncol = 4, scales = "free_y")
```

Project 3 P8160 - Lasso, Smoothing, and Hypothesis Testing

Muhire Kwizera, Quinton Neville, Zelos Zhu

April 18, 2019

1. Load Data

Cleaning, removing NA's, ordering factors, and scaling the data.

```
down.df <- read_csv("./Down.csv") %>%
  filter(., !(MouseID %in% c("3426_13", "3426_14", "3426_15"))) %>%
  dplyr::select(., -c("BCL2_N","H3MeK4_N","BAD_N","EGR1_N","H3AcK18_N","pCFOS_N","Bcatenin_N","MEK_N","
  janitor::clean_names() %>%
  mutate(
    class = as.factor(class) %>%
      fct_recode("Case" = "Down syndrome") %>%
      fct_relevel("Control", "Case")
  )

#Scale the data
scale.df <- down.df %>% dplyr::select(-c(mouse_id, class)) %>% scale() %>% as.tibble()
down.df  <- bind_cols(down.df %>% dplyr::select(mouse_id), scale.df, down.df %>% dplyr::select(class))
```

2. GLMNET Lasso Functions

```
#Test and Train
set.seed(2019)
train <- sample(1:nrow(down.df), 862)
test.df <- down.df[-train, ]
train.df <- down.df[train, ]

#Fit Lasso Mod Function
lasso <- function(data, accuracy = TRUE) {

  #Snag the index for matrices
  index <- which(names(data) == "class")

  #Response
  Y <- as.matrix(data[, index])

  #Design matrix
  X <- data[ ,-c(1, index)] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(.,data)
```

```
  X <- X[,-1]

  #Switch for Accuracy vs. ROC penalty optimization
  if (accuracy == TRUE) {
  #For Accuracy
  mod.lasso <- train(X, Y %>% as.factor(), method = "glmnet", trControl = trainControl(method = "cv", nu
                             tuneGrid = expand.grid(alpha = 1,lambda = 10^seq(-5, -2, length = 24)))
  } else {
  #For ROC
  mod.lasso <- train(X, Y %>% as.factor(), method = "glmnet",
                     trControl = trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunc
                     metric = "ROC", tuneGrid = expand.grid(alpha = 1,lambda = 10^seq(-5, -2, length = 
  }

  #Return list of coefficients, opt. lambda, and number of variables shrunk to zero
  return(list(beta = coef(mod.lasso$finalModel, mod.lasso$bestTune$lambda) %>% as.matrix(), num.zero = 
}

#test
#lasso(train.df)$beta

glmnet.predict <- function(model, data) {
  #Snag the index for matrices
  index <- which(names(data) == "class")
  #Design matrix
  X1 <- data[ ,-c(1, index)] %>%
    names() %>%
    paste(., collapse = "+") %>%
    paste("~ ", .) %>%
    formula() %>%
    model.matrix(., data)
  X1 <- X1[,-1]
  return(predict(model, newdata = X1, type = "prob")$Case)
}

#Test
set.seed(2019)
lasso.mod <- lasso(train.df, accuracy = TRUE)
#lasso.mod$lambda
preds <- ifelse(glmnet.predict(lasso.mod$model, test.df) > 0.5, "Case", "Control")
#mean(test.df$class != preds) #Misclassification
#lasso.mod$num.zero
#lasso.mod$beta
```

2. Smoothing and Parallelizing w/GLMNET

Boot strap coefficient estimates...

```
boot.lasso <- function(data, n.boot) {
coefficients <- c() #t(y_i*) in boot sampling
boot.selected <- c() #Y_ij * in boot smothing
lambda <- c()
```

```
n <- nrow(data)


for (i in 1:n.boot) {
  boot.df <- data[sample(1:n, replace = TRUE), ]
  selected <- bind_rows(data %>% dplyr::select(mouse_id),
            boot.df %>% dplyr::select(mouse_id)) %>%
      group_by(mouse_id) %>%
    summarize(
      count = n() - 1
    )
  boot.selected <- cbind(boot.selected, selected$count)
  result <- lasso(boot.df, accuracy = FALSE)
 # print(result$lambda)
  coefficients <- cbind(coefficients, result$beta)
  lambda <- c(lambda, result$lambda)
  }
return(list(coefficients = t(coefficients), lambda = lambda, boot.selected = t(boot.selected)))
}

#Test
#obj <- boot.lasso(train.df, 10)
#head(obj[[1]])
#head(obj[[2]][, 0:50])
```

Parallelize...

```
nCores <- 7
taskFun <- boot.lasso

registerDoParallel(nCores)

combine <- function(x, ...) {
  mapply(rbind, x, ..., SIMPLIFY = FALSE)
}

#100 X 10 = 1000 iterations
result.glmnet <- foreach(i = 1:1000, .combine = combine, .multicombine = TRUE,
                         .packages = c("dplyr", "glmnet", "caret", "elasticnet")) %dopar% {
  outSub <- taskFun(train.df, n.boot = 1)
  list(coefficients = outSub$coefficients, lambda = outSub$lambda, boot.selected = outSub$boot.selected
}
result.glmnet[[3]] <- result.glmnet[[3]] %>% t() #Right dimensions for matrix mult. later

write.csv(result.glmnet$coefficients, "glmnet_coef_roc_result.csv")
write.csv(result.glmnet$boot.selected, "glmnet_boot_roc_select.csv")
write.csv(result.glmnet$lambda, "lambda_roc.csv")
```

View/Load Results from the bootstrap smoothing.

```
glmnet.boot.df <- list.files(path = "./data", pattern = "[r][e][s][u][l][t]+") %>% as.tibble() %>%   #P
  rename(data_csv_path = value) %>%                                                                  #Give appropri
  mutate(input_files = map(.x = data_csv_path, ~read_csv(paste0("./data/", .x)))) %>%    #Add list.col
  unnest() %>%
  dplyr::select(-c(`data_csv_path`, `X1`)) %>% as.matrix()
```

```
glmnet.select.df <- list.files(path = "./data", pattern = "[s][e][l][e][c][t]+") %>% as.tibble() %>%
  rename(data_csv_path = value) %>%                                          #Give appropri
  mutate(input_files = map(.x = data_csv_path, ~read_csv(paste0("./data/", .x)))) %>%    #Add list.col
  unnest() %>%
  dplyr::select(-`X1`) %>%
  nest('V1':'V100') %>%
  spread(data_csv_path, data) %>%
  unnest() %>%
  as.matrix() %>%
  unname()

dim(glmnet.boot.df)
dim(glmnet.select.df)
```

3. Hypothesis Test

```
#Remove non-convergence
val <- which(glmnet.boot.df[,1] == 0)
if (length(val) != 0) {
glmnet.boot.df <- glmnet.boot.df[-val, ]
glmnet.select.df <- glmnet.select.df[,-val]
}

#Smoothed Point Estimate
mu.tilde <- apply(glmnet.boot.df, 2, mean)

#Smoothed Interval Test
t.star <- mu.tilde[1]
t <- glmnet.boot.df[,1]
y.ij <- glmnet.select.df[1,]
y.star <- mean(y.ij)
cov.j <- mean(t(y.ij - y.star)*(t - t.star))
cov.j <- (y.ij - y.star) %*% (t - t.star)/1000

#Smooth Interval Cov/SD calculations
cov.j <- (glmnet.select.df - apply(glmnet.select.df, 1, mean)) %*% (glmnet.boot.df - mu.tilde) / 1000
sd.boot <- apply(cov.j^2, 2, function(x) sum(x)) %>% sqrt()  #Std. error

#Resulting 95 Intervals
smooth.95.df <- tibble(LB95 = mu.tilde - qnorm(0.975)*sd.boot,
       UB95 = mu.tilde + qnorm(0.975)*sd.boot,
       Estimate = mu.tilde, reject = ((0 > LB95 & 0 > UB95) | (LB95 > 0 & UB95 > 0)))

smooth.90.df <- tibble(LB95 = mu.tilde - qnorm(0.95)*sd.boot,
       UB95 = mu.tilde + qnorm(0.95)*sd.boot,
       Estimate = mu.tilde, reject = ((0 > LB95 & 0 > UB95) | (LB95 > 0 & UB95 > 0)))

mean(glmnet.boot.df == 0) #20.32% for ACC; 22.83 for ROC
#itsn1_n, erk_n, trka_n, app_n, tiam1_n,
which(smooth.95.df$reject == TRUE) #0 acc; 0
which(smooth.90.df$reject == TRUE) #25 one erk_n;
```

```r
rownames(lasso.mod$beta)[which(smooth.95.df$reject == TRUE) ]
rownames(lasso.mod$beta)[which(smooth.90.df$reject == TRUE)]

lasso.mod$beta
```

4. Prediction

Write smoothing predict funcition...

```r
predict.smooth <- function(test.df, mu.tilde) {
    preds <- cbind(1, test.df %>% dplyr::select(-c(mouse_id, class)) %>% as.matrix()) %*% mu.tilde
    probs <- exp(preds) / (1 + exp(preds))
      return(probs)
}
```

Optimal lambda for regular lasso with CV.

```r
sampleSize <- nrow(train.df)

lassoCV <- function(data.df, kfolds = 10){
  folds <- sample(1:kfolds, sampleSize, rep = T)
  lambda <- rep(0, kfolds)
  for (k in 1:kfolds) {
    train.df <- data.df[folds != k,]
    test.df <- data.df[folds == k,]

    lambda[k] <- lasso(train.df)$lambda
  }
  return(lambda)
}

#set grid
lambda.grid.lasso <- 10^seq(-4, -2, length = 50)
taskFun <- lassoCV

#100 X 10 = 1000 iterations
optimal.lambda <- foreach(i = 1:10, .combine = rbind, .packages = c("dplyr", "glmnet")) %dopar% {
  outSub <- taskFun(train.df, kfolds = 10)
}

optimal.lambda <- mean(optimal.lambda) #0.00102369

#(optimal.glmnet.lambda <- lassoCV(train.df, 10))
```

Probability Threshold Optimization

```r
#Prob sequence to search over
p <- seq(0, 1, by = 0.01)
#lasso.mod <- lasso(train.df)
smooth.preds <- predict.smooth(test.df, mu.tilde)
lasso.preds  <- glmnet.predict(lasso.mod$model, test.df)
smooth.error <- c()
lasso.error  <- c()
```

```
#Optimize threshold test/train or full data
for (i in 1:length(p)) {
        smooth.error <- c(smooth.error, (ifelse(smooth.preds > p[i], "Control", "Case") != test.df$clas
        lasso.error  <- c(lasso.error, (ifelse(lasso.preds  > p[i], "Case", "Control") != test.df$class
}

#Optimal p
opt.smooth.p <- cbind(p, smooth.error)[which.min(smooth.error), 1]  #First run 0.001 - 0.01 -- 46%
opt.lasso.p <- cbind(p, lasso.error)[which.min(lasso.error), 1]

#Visualize
plot.p <- function(smooth.error, lasso.error, charstring, opt.p) {
tibble(
  probability  = p,
  Smooth = smooth.error,
  Lasso  = lasso.error
) %>% gather(key = Model, value = Misclassification, Smooth:Lasso) %>%
  filter(Model == charstring) %>%
  ggplot() +
    geom_point(aes(probability, Misclassification), color = "red", size = 1, shape = 15) +
    geom_step(aes(probability, Misclassification), color = "blue", alpha = 0.5, size = 1) +
    geom_vline(xintercept = opt.p, colour = "lightgrey", linetype = 2, size = 1) +
  labs(
    y = "Misclassification (Total Error)",
    x = "Probability",
    title = sprintf("%s Classification Threshold Optimization, (p = %d%s)", charstring, round(opt.p * 1
  )
}


p.opt.gg <- plot.p(smooth.error, lasso.error, "Smooth", opt.smooth.p) /
  plot.p(smooth.error, lasso.error, "Lasso", opt.lasso.p)

p.opt.gg
ggsave("probability_optimization_roc.jpg", p.opt.gg)
```

Viz ROC/AUC

```
#################ROC AUC Functions#################################
Roc.Log <- function(result,y.pred.r0, charstring){
  probs <- seq(0,1,by = 0.01)
  roc.log <- matrix(0,nrow = length(probs),ncol=2)
  result <- ifelse(result == charstring, TRUE, FALSE)
  i <- 1
  for (p in probs){
    pred <- y.pred.r0 > p
    ##False positive rate
    false.pos <- with(test.df,
                      sum(!result & pred)/sum(!result))
    ##True postive rate
    true.pos <- with(test.df,
```

```
                      sum(result & pred)/sum(result))
    roc.log[i,] <- c(false.pos,true.pos)
    i <- i + 1
  }
  return(roc.log)
}

plotRoc <- function(roc.log,charstring){
  data.frame(FP = rev(roc.log[,1]),TP = rev(roc.log[,2])) %>%
    ggplot() +
    geom_point(aes(FP, TP), color = "red", size = 0.5) +
    geom_step(aes(FP, TP), color = "blue") +
    geom_abline(slope = 1, linetype = 2, colour = "lightgrey") +
    labs(
      x = "False Positive",
      y = "True Positive",
      title = paste0("ROC Curve for Down Syndrome Diagnosis with " ,charstring)
    )
}
auc <- function(roc){
  len <- nrow(roc)
  ##The "delta X" values
  delta <- roc[-1,1] - roc[-len,1]
  ##The "heights" the rectangle (drop the first or last).
  hgt <- roc[-1,2]
  ##The Riemann Sum
  sum(-delta*hgt)
}

#Do we shrink things very close to zero, to zero? or keep these very small effects??
#mu.tilde <- ifelse(abs(mu.tilde) < 0.05, 0, mu.tilde)

smooth.preds <- predict.smooth(test.df, mu.tilde)
#lasso.preds  <- glmnet.predict(lasso(train.df)$model, test.df)
smooth.roc   <- Roc.Log(test.df$class, smooth.preds, "Control")
lasso.roc    <- Roc.Log(test.df$class, lasso.preds, "Case")

plotRoc <- function(smooth.roc, lasso.roc, charstring = "Smooth", auc) {
roc.df <- tibble(FP = c(smooth.roc[ ,1], lasso.roc[ ,1]), TP = c(smooth.roc[ ,2], lasso.roc[ ,2]), Model
roc.df %>%
  filter(Model == charstring) %>%
    ggplot() +
    geom_point(aes(FP, TP), color = 'red' , size = 1.5, shape = 15) +
    geom_line(aes(FP, TP), colour = 'blue', size = 1.25, alpha = 0.5) +
    geom_abline(slope = 1, linetype = 2, colour = "lightgrey") +
    labs(
      x = "False Positive",
      y = "True Positive",
      title = sprintf(" %s ROC for D.S. Diagnosis AUC = %d%s", charstring,  round(auc * 100), "%")
    )
}

#AUC ~ approx same
```

```r
smooth.auc <- auc(smooth.roc)  #0.987 First scenario,
lasso.auc  <- auc(lasso.roc)   #0.992

#Plots
smooth.plot <- plotRoc(smooth.roc, lasso.roc, "Smooth", smooth.auc)
lasso.plot  <- plotRoc(smooth.roc, lasso.roc, "Lasso", lasso.auc)
roc.gg <- smooth.plot / lasso.plot
roc.gg
ggsave("roc_auc_roc.jpg", roc.gg)

#Refit predicitons
smooth.preds <- predict.smooth(test.df, mu.tilde)
#lasso.preds  <- glmnet.predict(lasso(train.df)$model, test.df)

#Correct Classification Rate
(ifelse(smooth.preds > opt.smooth.p, "Control", "Case") == test.df$class) %>% mean() #ACC cenario, both
(ifelse(lasso.preds > opt.lasso.p, "Case", "Control") == test.df$class)  %>% mean()  #Second scenario,
```

```r
#Refit predicitons
#smooth.preds <- predict.smooth(test.df, mu.tilde)
#lasso.preds  <- glmnet.predict(lasso(train.df)$model, test.df)

#Plot the predicitions
prediction.gg <- tibble(
  Probabilities = c(1 - smooth.preds, lasso.preds),
  Model = rep(c("Smooth", "Lasso"), each = length(smooth.preds))
) %>%
  ggplot() +
  geom_density(aes(x = Probabilities, colour = Model, fill = Model), size = 1.24, alpha = 0.1) +
  labs(
    x = "Predicted Probabilites",
    y = "Density",
    title = "Distribution of Predictions by Model"
  )
prediction.gg

ggsave("predictions_dist_roc.jpg", prediction.gg)

#Lambda distribution plots
lambda.gg <- tibble(
  Probabilities = rbind(lambda.acc, lambda.roc)$'V1',
  Model = rep(c("CD Accuracy", "ROC"), each = 1000)
) %>%
  ggplot() +
  geom_density(aes(x = Probabilities, colour = Model, fill = Model), size = 1.24, alpha = 0.1) +
  labs(
    x = "Optimized Lambda Values",
    y = "Density",
    title = "Distribution of Lambda by Optimization"
  )
lambda.gg

ggsave("lambda.jpg", lambda.gg)
```

```r
mean(lambda.roc$V1)
mean(lambda.acc$V1)
```

Cost Analysis (Subjective whether type 1 or type 2 error is worse)

Type 1 Adverse

```r
#Generate Cost Matrices
cost_t1 <- function(t, p){
    if(t & !p)
        return(10) #False positive weight
    if(!t & p)
        return(2) #False Negative
  if(t & p)
    return(-2) #True Positive prediction
    return(-1)  #True Negative
}

cost_t2 <- function(t, p){
    if(t & !p)
        return(2) #False positive weight
    if(!t & p)
        return(10) #False Negative
  if(t & p)
    return(-1) #True Positive prediction
    return(-2)  #True Negative
}

#Set grid sequence
seq <- 0.01
p <- seq(0, 1, by = seq)

cost <- function(cost_t1) {
smooth.cost.vec <- lasso.cost.vec <- c()
#Optimize threshold test/train or full data
for (i in 1:length(p)) {
  #i = 10
    smooth.pred <- ifelse(smooth.preds > p[i], TRUE, FALSE)
    lasso.pred  <- ifelse(lasso.preds  > p[i], TRUE, FALSE)
    smooth.df <- cbind(ifelse(test.df$class == "Control", TRUE, FALSE), smooth.pred)
    lasso.df  <- cbind(ifelse(test.df$class == "Case", TRUE, FALSE), lasso.pred)
    smooth.cost.vec <- c(smooth.cost.vec, sum(apply(smooth.df, 1, function(ls) cost_t1(ls[1], ls[2]))))
    lasso.cost.vec  <- c(lasso.cost.vec, sum(apply(lasso.df, 1, function(ls) cost_t1(ls[1], ls[2]))))
  }
  return(list(smooth.vec = smooth.cost.vec, lasso.vec = lasso.cost.vec))
}



#Visualize
cost.viz <- function(smooth.cost.vec, lasso.cost.vec, charstring, type, opt) {
tibble(
```

Logistic-LASSO

Muhire Kwizera, Quinton Neville, Zelos Zhu

4/11/2019

```
#Load necessary packages
library(tidyverse)
library(readr)
library(dplyr)
library(caret)
library(glmnet)
library(e1071)
library(naniar)
library(foreach)
```

** RESULTS BASED ON MISCLASSIFICATION RATE**

0. Load Data

```
# load original data: there's missing data.
# Look into this and how (or acknowledge) it might
# affect our results.
# We decided to remove variables that were correlated with
# others and which had more missing data. We also removed
# three mice with the most missing data
down.df <- read_csv("./Down.csv") %>%
  filter(!(MouseID %in% c("3426_13", "3426_14", "3426_15"))) %>%
  dplyr::select(-c("BCL2_N", "H3MeK4_N", "BAD_N","EGR1_N", "H3AcK18_N",
                   "pCFOS_N", "Bcatenin_N", "MEK_N","ELK_N")) %>%
  janitor::clean_names()

# This code scales predictors and adds intercept.
down.scaled.int = down.df %>%
  dplyr::select(-c(mouse_id, class)) %>%
  scale() %>%
  cbind(read_csv("./Down.csv") %>%
          filter(!(MouseID %in% c("3426_13", "3426_14", "3426_15"))) %>%
          dplyr::select(-c("BCL2_N", "H3MeK4_N", "BAD_N","EGR1_N", "H3AcK18_N",
                           "pCFOS_N", "Bcatenin_N", "MEK_N","ELK_N")) %>%
          janitor::clean_names() %>%
          dplyr::select(c(mouse_id, class))) %>%
  mutate(intercept = 1,
         class = ifelse(class == "Down syndrome", 1, 0)) %>%
  dplyr::select(mouse_id, class, intercept, everything())
# %>%
#   na.omit() # use this line for a complete-case analysis

# Split data into training and test sets: 80/20 split
set.seed(2019) # group's seed
```

```r
numTrain = round(nrow(down.scaled.int) * 0.80)
trainidx = sample(1:nrow(down.scaled.int), numTrain)

# training data
down.trainX = down.scaled.int %>%
  dplyr::select(-c(mouse_id, class)) %>%
  .[trainidx,]

down.trainY = down.scaled.int %>%
  dplyr::select(class) %>%
  .[trainidx,]

# Training data set for some of the built-in R functions
down.train = cbind(class = down.trainY,
                   down.trainX[,-1])

nparams = dim(down.train)[2] # the number of model parameters (includes intercept)

# test data
down.testX = down.scaled.int %>%
  dplyr::select(-c(mouse_id, class)) %>%
  .[-trainidx,]
down.testY = down.scaled.int %>%
  dplyr::select(class) %>%
  .[-trainidx,]
```

1. Coordinate Descent LASSO (CD-LASSO) Algorithm

Main Coordinate Descent Functions

Goal is to minimize the objective function $-\mathcal{L}(\beta; X, Y) + \lambda \sum_{j=0}^{p} |\beta_j|$ where $\mathcal{L}(\beta; X, Y)$ is the log likelihood and λ is the tuning parameter for the LASSO penalty.

```r
# analysis of probability of having malignant breast cancer
# unit increase in standardized varaible corresponds to increase in x by (sd - mean)

# Logistic with LASSO minimization function
log_lasso = function(betavec_now, x_now,
                     betavec_other, x_other,
                     beta_mult_now, beta_mult_other,
                     y, lambda = 0) {

  betavec_now = betavec_now %>% as.array()
  betavec_other = betavec_other %>% as.array()

  x_now = x_now %>% as.matrix()
  x_other = x_other %>% as.matrix()

  y = y

  mu <- x_now %*% betavec_now + x_other %*% betavec_other
```

```r
  # Maximizing log lik is same as minimizing neg log lik
  loglik_neg <- -sum(y*mu - log(1 + exp(mu)), na.rm = TRUE) # ignores missing values

  penalty = lambda * (abs(betavec_now * beta_mult_now) + sum(abs(betavec_other %*% beta_mult_other)))

  obj_fctn = loglik_neg + penalty

  return(obj_fctn)
}

# Optimize function in 1D
optimize1D = function(idx, dat, betavec, y, lambda = 0) {

  absbeta_multpr = c(0, rep(1, (length(betavec) - 1)))

  # Minimize objective function in 1-D while fixing everything else. Search interval is [-20, 20]
  min_obj = optimize(log_lasso,
                     -50:50, dat[,idx],
                     betavec[-idx], dat[,-idx],
                     absbeta_multpr[idx], absbeta_multpr[-idx],
                     y, lambda)

  return(list(idx = idx, minimum = min_obj$minimum, objective = min_obj$objective))
}

# function for updating betavec after each optimization
update_betas = function(idx, dat, betavec, y, lambda = 0) {
  min_obj = optimize1D(idx, dat, betavec, y, lambda)
  betavec[min_obj$idx] = min_obj$minimum

  return(list(betavec = betavec, objective = min_obj$objective))
}

# function coord_desc: Implements coordinate-descent algorithm.
# Inputs design matrix dat, response vector y, initial guesses
# for regression estimates betavec, and lasso penalty term
# lambda (default is 0). Outputs the solution for the specified
# lasso penalty.
coord_desc = function(dat, y, betavec, lambda = 0) {
   #--------------------Initialize-----------------------------
  tol = sqrt(.Machine$double.eps)
  differ_obj = 1
  differ_beta = 1
  mod = ncol(dat) + 1

  idx = 0
  cnt = 0

  obj_min = 0
  # solpath = NULL

  # loop through till convergence
  while (differ_obj > tol | differ_beta > tol | cnt < 15000) {
```

```
    cnt = cnt + 1
    if (cnt %% 1000 == 0) {print(cnt)} # for checking convergence

    idx = max(c(1, (idx + 1) %% mod)) # increment index

    old_betavec = betavec
    old_obj_min = obj_min
    # solpath = rbind(solpath, betavec)

    # print(betavec)
    # print(obj_min)

    updates = update_betas(idx, dat, betavec, y, lambda)

    betavec = updates$betavec
    obj_min = updates$objective

    differ_obj = abs(obj_min - old_obj_min)
    differ_beta = sum((betavec - old_betavec)^2)
}

# solution for current lambda value
#sol_curr_lambda = betavec #tail(solpath, 1) %>% as.array()

return(betavec)

}

# # test run of coord_desc function
# set.seed(seed = 2019)
# betavec = runif(nparams, min = -1, max = 1)
# coord_desc(dat = down.trainX, y = down.trainY, betavec, lambda = 0)

# function lasso_shrink_plot: illustrates lasso shrinkage.
# Inputs seed (default is 2019), length of vector of
# regression estimates length_betavec (default is nparams),
# training design matrix down.trainX, training response
# vector down.trainY, and the resolution of the shrinkage vector lambda_res.
# Outputs largest lasso penalty and lasso shrinkage plot.
lasso_shrink_plot = function(seed = 2019, length_betavec = nparams, down.trainX, down.trainY, lambda_res
 set.seed(seed = seed)
betavec = runif(length_betavec, min = -1, max = 1)
# # regression estimates without lasso penalty
# Rprof("cdprofile.out")
betas_nolasso = coord_desc(dat = down.trainX, y = down.trainY, betavec = betavec , lambda = 0)
# Rprof(NULL)
# summaryRprof(filename = "/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/
lambda_max = floor(max(abs(betas_nolasso))) # largest lambda value for cross validation

# # Fit coordinate descent for multiple lambda values
lambdas = seq(from = 0, to = lambda_max / 3, #to = 2 * lambda_max / 3,
              length.out = lambda_res) %>% as.array()
solutions = apply(lambdas, MARGIN = 1, FUN = coord_desc,
```

```
                  dat = down.trainX, y = down.trainY, betavec = betavec)

# Illustrating LASSO shrinkage
lasso_shr_plot = solutions %>%
  t() %>%
  .[,-1] %>%
  as.tibble() %>%
  mutate(lambda = lambdas) %>%
  # mutate(variable = replace(variable, c(paste0("V", 1:(ncol(dat) - 1))), full_vars)) %>%
  gather(variable, estimate, paste0("V", 1:(length_betavec - 1))) %>%
  # mutate(variable = replace(variable, c(paste0("V", 1:(ncol(dat) - 1))), full_vars)) %>%
  ggplot(aes(x = lambda, y = estimate, group = variable)) +
  geom_smooth(se = FALSE, color = "black", size = 0.75) +
  geom_vline(color = "blue", xintercept = 1.5,
             show.legend = TRUE) +
  geom_hline(color = "red",yintercept = 0)

return(list(lambda_max = lambda_max, plot = lasso_shr_plot))
}

# # test run of lasso_shrink_plot function
# lasso_shrink_plot(seed = 2019, length_betavec = nparams, down.trainX, down.trainY, lambda_res = 2)
```

Main Cross-validation Functions

```
# Compute misclassification rate for each lambda value
misscl_rate = function(dattest, solutions, s = 1,  ytest) {

  #  # ************************************************************
  # # this code was used to determine optimal probability cutoff
  #  # for predicting binary outcomes.
  #  # 0.5 was decent so kept it because it's fair
  #  # predict y
  # mupred = dat %*% solutions[, s]
  # probpred = exp(mupred) / (1 + exp(mupred))
  #
  # # get performance measures
  # pred.cd = prediction(mupred, y)
  # perf.cd = performance(pred.cd,"tpr","fpr")
  # plot(perf.cd)
  #
  # # looking at sensitivity and 1 - specificity for each cutoff
  # fpr_tpr_tbl = tibble(cutoff = perf.cd@alpha.values %>% unlist(),
  #                      fpr = perf.cd@x.values %>% unlist,
  #                      tpr = perf.cd@y.values %>% unlist)
  # # ********************************************************************

  # predict y
  mupred = dattest %*% solutions[, s]
  probpred = exp(mupred) / (1 + exp(mupred))
  ypred = ifelse(probpred > 0.5, 1, 0)
```

```
  # missclassification rate
  misscl = mean(ypred != ytest, na.rm = TRUE)

  return(misscl)
}

# implementing the heavy lifting of cross validation
implement_cv = function(folds, fold, y, dat, lambdas, betavec) {
  # split data into training and testing sets
ytest = y[folds == fold]
dattest = dat[folds == fold,] %>% as.matrix()
ytrain = y[folds != fold]
dattrain = dat[folds != fold,]

# implement coordinate descent algorithm for all lambda choices
solutions = apply(lambdas, 1, FUN = coord_desc, dat = dattrain, y = ytrain,  betavec = betavec)
msclass_rate = apply(1:ncol(solutions) %>% as.array, 1, FUN = misscl_rate,
      solutions = solutions, dattest = dattest, ytest = ytest)

return(msclass_rate)
}

# function cv_fctn: chooses best lasso shrinkage by
# cross validation function. Inputs number of folds k
# (default is 5), trainining response vector y, training
# design matrix dat, vector of candidate lasso shrinkages
# lambdas, and guesses of initial regression estimates betavec.
# Outputs the optimal lasso shrinkage best.lambda and the cross
# validation plot cvplot
cv_fctn = function(k = 5, y, dat, lambdas, betavec) {
  # cross validation
  # k = 2 #5
  set.seed(2019)
  folds = sample(1:k, nrow(y %>% as.array()), replace = TRUE)

  mscl_rates = apply(
  1:k %>% as.array,
  1,
  FUN = implement_cv,
  folds = folds,
  y = y,
  dat = dat,
  lambdas = lambdas,
  betavec = betavec
  )

  cv_errors = apply(mscl_rates, 1, mean)

  best.lambda = lambdas[which.min(cv_errors)]

  # Plot cross validation errors
  cvplot = tibble(lambda = lambdas,
  cv_error = cv_errors) %>%
```

```
  ggplot(aes(x = lambda, y = cv_errors)) + geom_smooth()

  return(list(best.lambda = best.lambda, cvplot = cvplot))
}

# # test run of cv_fctn
# cv.out = cv_fctn(k = 2, y = down.trainY, dat = down.trainX, lambdas = seq(from = 0, to = 0.1, length.
# # cv.out$cvplot
```

Get Optimal model by 5-Fold Cross Validation for CD-LASSO

```
# # Fitting the optimal model
# optimal_sol = coord_desc(dat = down.trainX, y = down.trainY,
#                          betavec = runif(nparams, -1, 1) , cv.out$best.lambda) %>%
#   round(., 3)


# function cd_cv: implements coordinate descent algorithm,
# chooses optimal lasso shrinkage by cross validation, and
# estimates final optimal model using all the training data.
# Inputs number of folds k (default is 5), trainining response
# vector y, training design matrix dat, vector of candidate lasso
# shrinkages lambdas, and guesses of initial regression estimates
# betavec. Outputs the optimal lasso shrinkage best.lambda, and
# estimated optimal model coefficients optimal_sol
cd_cv = function(k = 5, y, dat, lambdas, betavec) {
  cv.out = cv_fctn(k, y, dat, lambdas, betavec)
  # cv.out$cvplot

  # Fitting the optimal model
  optimal_sol = coord_desc(dat = down.trainX, y = down.trainY,
                           betavec = runif(nparams, -1, 1) , cv.out$best.lambda)

  return(list(best.lambda = cv.out$best.lambda, optimal_sol = optimal_sol))
}

# # test run of cd_cv function
# # lambda sequence: 0.000000000 0.00111111 0.002222222 (chose 0) 0.003333333 0.004444444 0.005555556 0
# # Rprof("cdprofile.out") # profiling code
# log_lasso_out = cd_cv(k = 5, y = down.trainY, dat = down.trainX,
#                       lambdas = seq(from = 0, to = 0.001111111, length.out = 2) %>% as.array(),
#                       betavec = runif(nparams, -1, 1))
# # Rprof(NULL)
# # summaryRprof(filename = "/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homewor
#
# save(log_lasso_out, file = "log_lasso_out.RData")
load("/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/Group Project 3/boots
```

2. Bootstrap Smoothing Algorithm for Logistic LASSO (based on Efron (2014). JASA)

Get and Fit Optimal Model based on Built-in R Logistic LASSO (glmnet)

```
# function logistic_lasso_builtin: Implements built-in R
# lasso algorithm with train function and glmnet.
# Works under the complete-case scenario. Inputs training data
# down.train, training design matrix down.trainX, training
# response vector down.trainY, and largest lasso penalty lambda_max (default is 0.01).
# Outputs cross-validation plot cvplot and optimal lasso shrinkage best.lambda
logistic_lasso_builtin = function(down.train, down.trainX, down.trainY, lambda_max = 0.01, nlambdas = 1

# built-in R logistic-lasso algorithm.
# lambda = 0.3 seems best for complete-case analysis
# function train only works under complete-case scenario
# We chose lambda = seq(from = 0, to = 0.01, length.out = 10)
log.grid = expand.grid(alpha = 1, lambda = seq(0, lambda_max, length.out = nlambdas)) # 1 is for lasso

# train using train: misclassification rate
log.train.miscl <- train(class ~ .,
                  data = down.train %>% mutate(class = as.factor(class)),
                  method = "glmnet",
                  trControl = trainControl(method = "cv", number = 5), #, classProbs = TRUE),
                  family = "binomial",
                  tuneGrid = log.grid)

cvplot = plot(log.train.miscl)
best.lambda = log.train.miscl$bestTune$lambda

# # train using cv.glmnet: deviance
# lambda.grid.lasso <- seq(0, lambda_max, length = nlambdas) #
# log.train.dev <- cv.glmnet(down.trainX %>% as.matrix() %>% .[,-1], down.trainY %>% as.matrix(),
#                            alpha = 1,intercept = T, lambda = lambda.grid.lasso,
#                            family = "binomial", nfolds = 5, maxit = 100000)

cvplot = plot(log.train.dev)
best.lambda = log.train.dev$lambda.min

# fit optimal model with glmnet
glmnet.model = glmnet(down.trainX %>% as.matrix(), down.trainY, family = "binomial",
                      alpha = 1, lambda = best.lambda)

# Adding estimated intercept
betas_lasso_r = c(#glmnet.model$a0 %>% as.numeric(),
          glmnet.model$beta %>% as.vector())

# Adding estimated intercept
betas = c(#glmnet.model$a0 %>% as.numeric(),
          glmnet.model$beta %>% as.vector())

return(list(betas = betas, best.lambda = best.lambda, cv_plot = cvplot))
```

```
}

# # test run of logistic_lasso_builtin function
# log_lasso_r_out = logistic_lasso_builtin(down.train, down.trainX = down.trainX,
#                                          down.trainY = down.trainY, lambda_max = 0.01, nlambdas = 10)
#
# save(log_lasso_r_out, file = "log_lasso_built_in_once_misclassification.RData")
load("/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/Group Project 3/boots
```

Main Bootstrap Function with Parallel Computing

```
# function lasso_boot: fits built-in logistic lasso to bootstrapped
# data. Inputs iteration iter (default is 1), training data down.train,
# training design matrix down.trainX, training response vector down.trainY,
# and largest lasso penalty lambda_max (default is 0.01). Outputs
# logistic lasso output containing best lasso penalty best.lambda,
# the estimated parameters betas, and cross_validation plot.

lasso_boot = function(iter = 1, down.train, down.trainX, down.trainY, lambda_max = 0.01) {
  # sample from training data with replacement
  boot_idx = sample(1:nrow(down.train), replace = TRUE) #%>% sort() #%>% unique()

  # training data
  down.train.bootX = down.train %>%
    dplyr::select(-c(class)) %>%
    .[boot_idx,]

  down.train.bootY = down.train %>%
    dplyr::select(class) %>%
    .[boot_idx,]

  # Training data set for some of the built-in R functions
  down.train.boot = cbind(class = down.train.bootY,
                          down.train.bootX[,-1])

  # estimate model parameters
  log_lasso_r_out = logistic_lasso_builtin(down.train.boot, down.train.bootX,
                                           down.train.bootY, lambda_max, nlambdas = 10)

  # For confidence interval. Count how many times each observed response
  # appears in the bootstrapped data. Just take the sum of ones in the
  # bootstrapped dataset for this binary response case and the sum of zeroes
  multinom = NULL
  multinom[down.trainY == 1] = sum(down.train.bootY == 1)
  multinom[down.trainY == 0] = sum(down.train.bootY == 0)

  return(list(iter = iter, best.lambda = log_lasso_r_out$best.lambda,
              betas = log_lasso_r_out$betas, multinom = multinom))
  #c(iter = iter, best.lambda = log_lasso_r_out$best.lambda, betas = log_lasso_r_out$betas))
}

# # test run of lasso_boot function
```

```
# log_lasso_r_out = lasso_boot(iter = 1, down.train, down.trainX = down.trainX,
#                              down.trainY = down.trainY, lambda_max = 0.01)

# function boot_parallel: implements bootstrap with
# parallel computing: inputs original acui data,
# and number of bootstraps and outputs sampling distribution
boot_parallel = function(down.train, down.trainX, down.trainY, lambda_max = 0.01, B = 1:2) {
  # implement bootstrap with parrallel computing
  # boostrap B times
  # set.seed(2019)
  lasso_results = foreach(b = B, .combine = rbind) %do%
    lasso_boot(iter = b, down.train, down.trainX = down.trainX,
               down.trainY = down.trainY, lambda_max) %>%
    as.tibble()

  return(lasso_results)
}
```

Implement Bootstrap-Smoothing and Save Estimated Results for Future Use

```
#*********************************************************
# This section was commented out to speed-up computations
# The results from this section were saved in an R file and
# are loaded for computing confidence intervals
# *********************************************************

# # test run for boot_parallel
nboots = 200 # will take about
# # Rprof("cdprofile.out") # profiling code
# # lasso_results = boot_parallel(down.train, down.trainX, down.trainY, lambda_max = 0.01, B = 1:nboots
# set.seed(2019)
# repl_lasso_res = replicate(5, boot_parallel(down.train, down.trainX, down.trainY,
#                                             lambda_max = 0.01, B = 1:nboots))
# save(repl_lasso_res, file = "repl_lasso_res.RData")
# # Rprof(NULL)
# # summaryRprof(filename = "/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homewor
load("/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/Group Project 3/boots
```

Process the Saved Estimated Bootstrap-Smoothing Results for Standard error Computations

```
# # load saved results
# load("C:/Users/Studentlab/Desktop/Computing_Project3/repl_lasso_res.RData")
# load("/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/Group Project 3/boo

# function unlist_boot_res: unlists bootstrap results for further
# computations. Inputs ...
unlist_boot_res = function(lasso_results) {
  iterations = lasso_results[[1]] %>% unlist() # lasso_results[,1] %>% unlist() %>% as.array()
  best.lambdas = lasso_results[[2]] %>% unlist() # lasso_results[,2] %>% unlist() %>% as.array()

  # betas = multinom = NULL  # initialize
```

```
  # iter = 0
  # while (iter < length(iterations)) {
  #   iter = iter + 1
    betas = lasso_results[[3]] %>% as.data.frame() %>% t # rbind(betas, lasso_results[iter,3] %>% unlis
    multinom = lasso_results[[4]] %>% as.data.frame() %>% t # rbind(multinom, lasso_results[iter,4] %>%
  # }

  return(list(iterations = iterations, best.lambdas = best.lambdas,
              betas = betas, multinom = multinom))
}

# # ********************************************************************
# # This part processes the saved estimated results and puts them in a
# # desired form for standard error computations
# # ********************************************************************
#
# # extract each individual bootstrapped replication
# lasso_res1 = repl_lasso_res[,1]
# lasso_res2 = repl_lasso_res[,2]
# lasso_res3 = repl_lasso_res[,3]
# lasso_res4 = repl_lasso_res[,4]
# lasso_res5 = repl_lasso_res[,5]
#
# # unlist the replications
# # Rprof("cdprofile.out") # profiling code
# lasso_res1_unlist = unlist_boot_res(lasso_res1)
# # Rprof(NULL)
# # summaryRprof(filename = "/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homewor
#
# lasso_res2_unlist = unlist_boot_res(lasso_res2)
# lasso_res3_unlist = unlist_boot_res(lasso_res3)
# lasso_res4_unlist = unlist_boot_res(lasso_res4)
# lasso_res5_unlist = unlist_boot_res(lasso_res5)
#
# # combine the replications into one bootstrapped data set
# iterations = c((0 * nboots) + lasso_res1_unlist$iterations,
#                (1 * nboots) + lasso_res2_unlist$iterations,
#                (2 * nboots) + lasso_res3_unlist$iterations,
#                (3 * nboots) +  lasso_res4_unlist$iterations,
#                (4 * nboots) + lasso_res5_unlist$iterations)
#
# best.lambdas = c(lasso_res1_unlist$best.lambdas,
#                  lasso_res2_unlist$best.lambdas,
#                  lasso_res3_unlist$best.lambdas,
#                  lasso_res4_unlist$best.lambdas,
#                  lasso_res5_unlist$best.lambdas)
#
# betas = rbind(lasso_res1_unlist$betas,
#               lasso_res2_unlist$betas,
#               lasso_res3_unlist$betas,
#               lasso_res4_unlist$betas,
#               lasso_res5_unlist$betas)
#
```

```
# multinom = rbind(lasso_res1_unlist$multinom,
#                  lasso_res2_unlist$multinom,
#                  lasso_res3_unlist$multinom,
#                  lasso_res4_unlist$multinom,
#                  lasso_res5_unlist$multinom)
#
# # final bootstrap results. Used for standard error and
# # confidence interval computtaions
# lasso_results_all = list(iterations = iterations,
#                          best.lambdas = best.lambdas,
#                          betas = betas,
#                          multinom = multinom)
#
# # save the final bootsrapped data set
# save(lasso_results_all, file = "lasso_results_all.RData")
load("/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/Group Project 3/boots
```

Get Bootstrap-Smoothing Model and the Confidence Intervals

```
# get bootstrapped data of interest
betas = lasso_results_all$betas %>% .[-c(482,802),] # remove bootstraps 482 and 802 for convergence iss

multinom = lasso_results_all$multinom %>% .[-c(482,802),]
iterations = lasso_results_all$iterations %>% .[-c(482,802)]
best.lambdas = lasso_results_all$best.lambdas %>% .[-c(482,802)]

# estimated model parameters by bootstrap-smoothing.
# This is also t.* used for confidence intervals
mean_betas = betas %>%
  apply(., 2, mean) %>%
  as.vector()

# average multinomial variables in Efron (JASA 2014)
mean_multinom = multinom %>%
  apply(., 2, mean) %>%
  as.vector()

# computing (multinom - mean_multinom) and
# (betas - mean_betas)
diff_multinom = diff_betas = NULL
idx = 0 # 1:nrow(multinom) %>% as.array()
while (idx < nrow(multinom)) {
  idx = idx + 1
  diff_multinom = rbind(diff_multinom, multinom[idx,] - mean_multinom)
  diff_betas = rbind(diff_betas, betas[idx,] - mean_betas)
}

# function prod_mult_betas: computes and outputs the term
# cov_j from Efron's JASA 2014 paper which is used to get
# standard error for each beta. Inputs beta index beta_idx (default = 1)
prod_mult_betas = function(beta_idx = 1) {
  # initalize
```

```
  prods_diffs_mult_betas = NULL #0
  idx = 0

  # compute the term of interest
  while (idx < ncol(diff_multinom)) {
    idx = idx + 1
    prods_diffs_mult_betas = c(prods_diffs_mult_betas,
                               mean(diff_multinom[, idx] * diff_betas[, beta_idx]))
  }

  return(prods_diffs_mult_betas)
}

# # test run of prod_mult_betas function
# prods_diffs_mult_betas = prod_mult_betas(beta_idx = 1)

prods_diffs_mult_betas = apply(1:ncol(diff_betas) %>% as.array, 1, prod_mult_betas)
sd_boot = sqrt((prods_diffs_mult_betas / nboots)^2 %>% colSums()) # bootstrap-smoothing std dev

# confidence interval
lowci_95 = mean_betas - 1.96 * sd_boot
upci_95 = mean_betas + 1.96 * sd_boot
```

3. Comparing CD-LASSO and Bootstrap-Smoothing

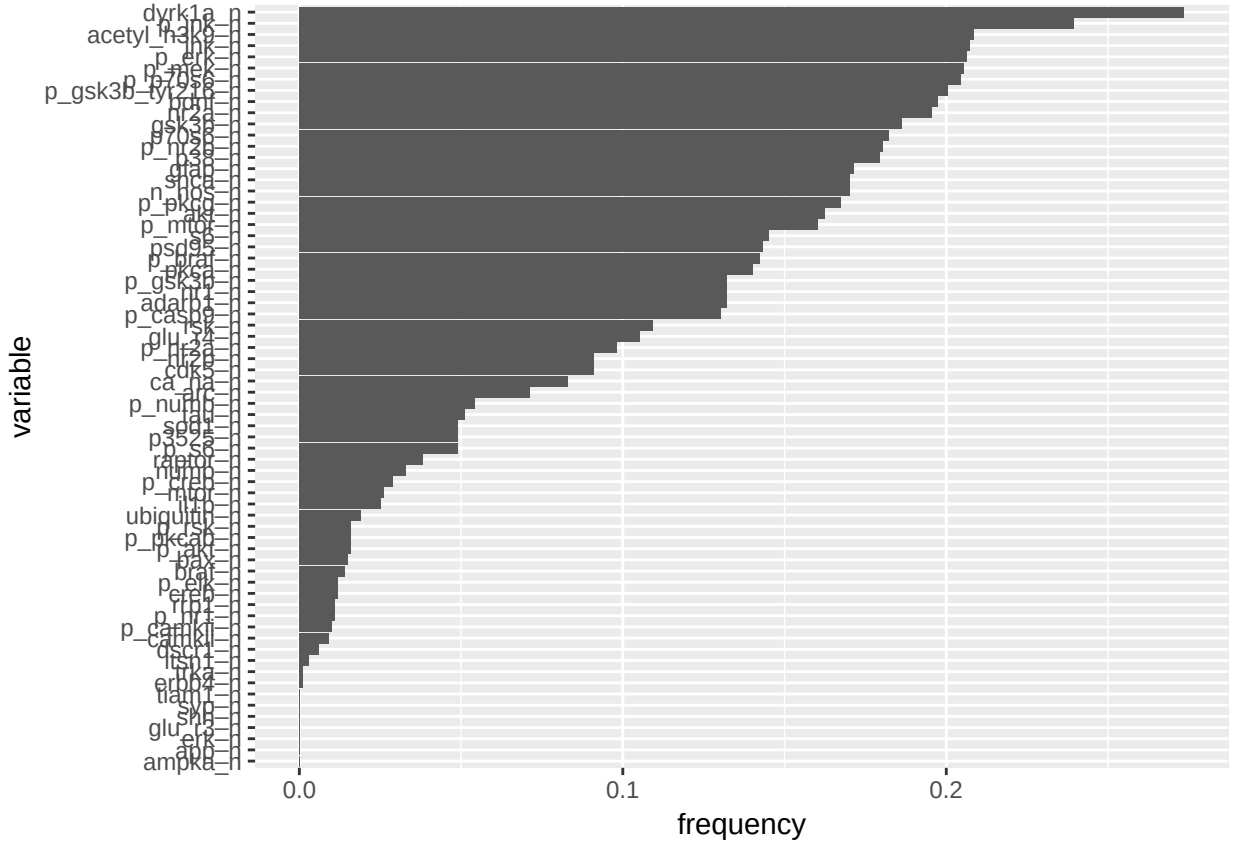
Predict Outcome and Get Misclassification Rate Based on CD-LASSO and Bootstrap-Smoothing

```
# function for predicting outcome and getting misclassification error
predicty = function(testX, betavec, testY) {
  # predict yvis_miss(down.df %>% select(-c(mouse_id))

  mupred = (testX %>% as.matrix()) %*% betavec
  probpred = exp(mupred) / (1 + exp(mupred))
  ypred = ifelse(probpred > 0.5, 1, 0)

  # misclassification rate
  misscl = mean(ypred != testY, na.rm = TRUE)

  return(misscl)
}

# misclassification rates for CD-lASSO
miscl_CD_LASSO = predicty(testX = down.testX, betavec = (log_lasso_out$optimal_sol %>% as.vector),
                          testY = down.testY)

# misclassification rates for built-in logistic-lASSO
# load("/Users/mhk/Desktop/Columbia/classes/Spring 2019/Advanced Computing/Homework/Group Project 3/boo
miscl_R_LASSO = predicty(testX = down.testX, betavec = (log_lasso_r_out$betas %>%  as.vector),
                          testY = down.testY)
```

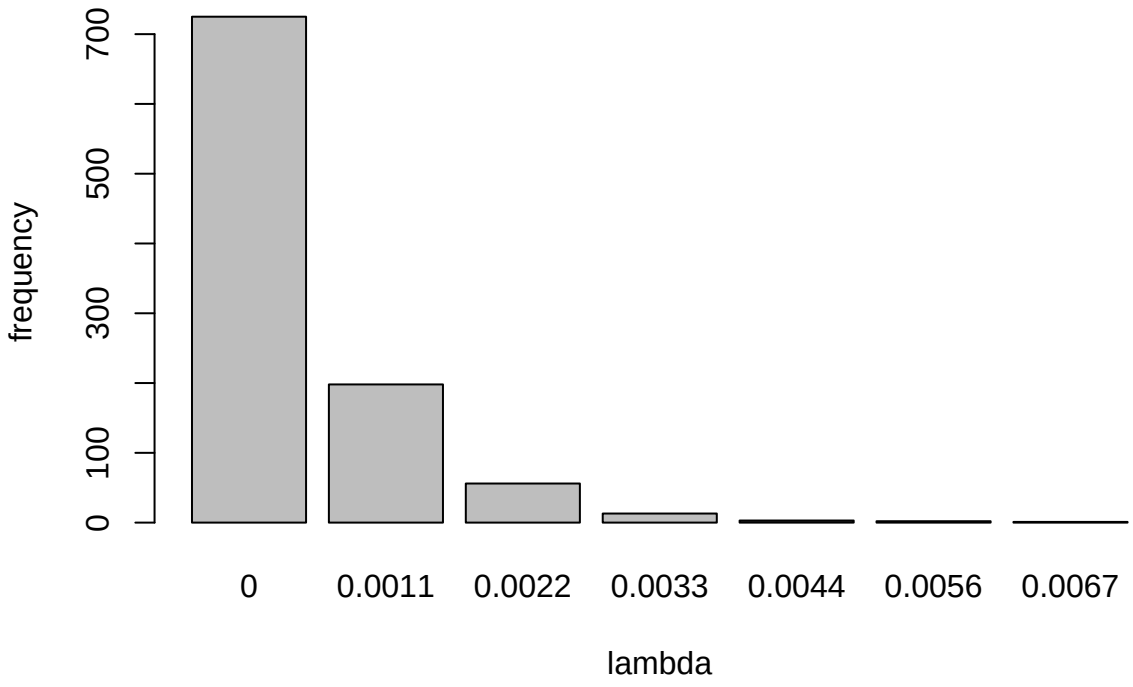
```
# misclassification rates for CD-lASSO. We lost the intercept along the way.
# We use the one from the built-in lasso (which was 0 in this case)
miscl_boot_smooth = predicty(testX = down.testX,
                             betavec = (c(log_lasso_r_out$betas[1], mean_betas) %>% as.vector),
                             testY = down.testY)
```

Count Proportion of Times an Estimate was Shrunk to 0 and a LASSO Tuning Parameter Chosen as Optimal

```
# proportion of times an estimate was shrunk to 0
tibble(variable = down.train %>% names() %>% .[-1],
       frequency = apply(betas == 0, 2, mean)) %>%
  mutate(variable = reorder(variable, frequency)) %>%
  ggplot(aes(x = variable, y = frequency)) +
  geom_bar(stat = "identity") +
  coord_flip()
```




```
# proportion of times a lasso
best.lambdas %>%
  round(.,4) %>%
  table() %>%
  barplot(xlab = "lambda", ylab = "frequency")
```



4. Plot Estimates and 95% CI's for the Included Variables

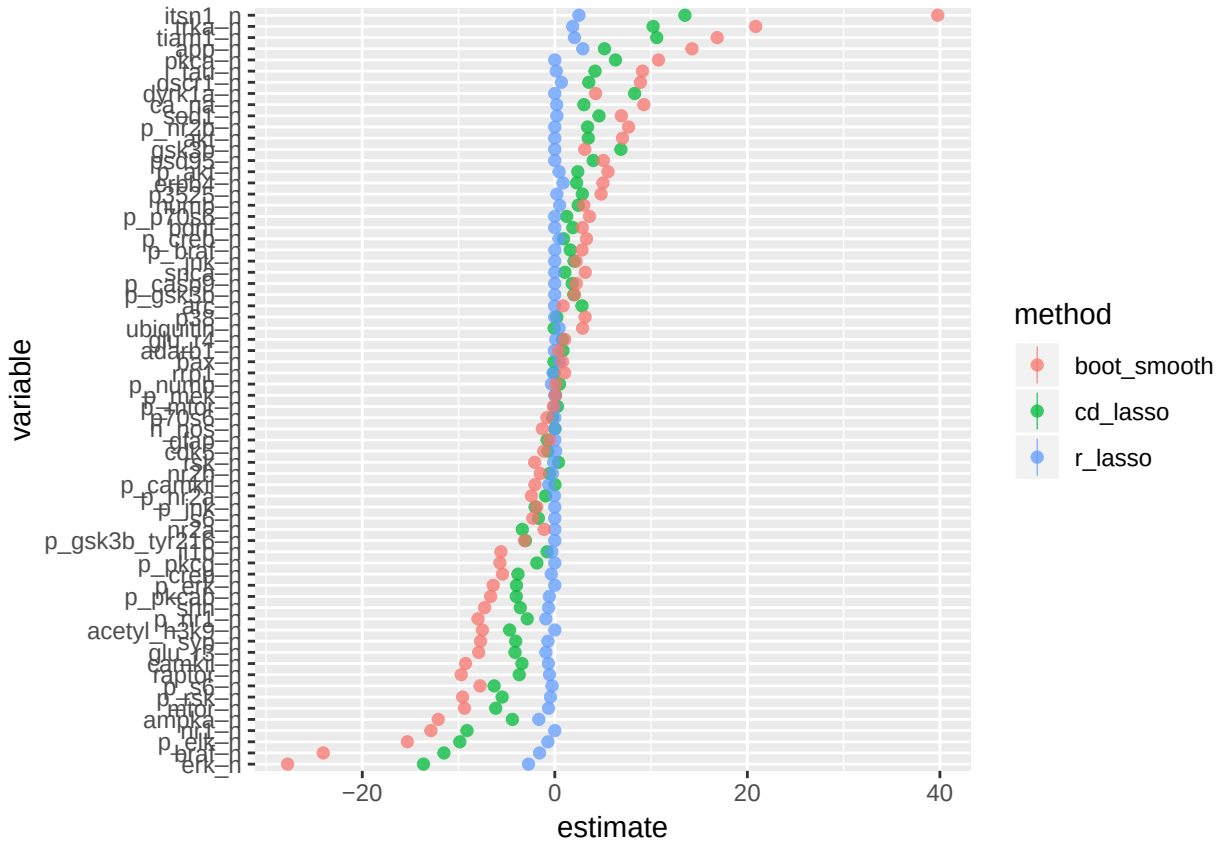
```
log_lasso_out$best.lambda
```

```
## [1] 0.001111111
```

```
log_lasso_r_out$best.lambda
```

```
## [1] 0.002222222
```

```
tibble(variable = down.train %>% names() %>% .[-1],
       estimate_cd_lasso = log_lasso_out$optimal_sol %>% .[-1],
       estimate_r_lasso = log_lasso_r_out$betas %>% .[-1],
       estimate_boot_smooth = mean_betas) %>%
  gather(method, estimate, estimate_cd_lasso:estimate_boot_smooth) %>%
  mutate(method = str_replace(method, "estimate_", ""),
         lowci_95 = ifelse(method == "boot_smoth", lowci_95, estimate),
         upci_95 = ifelse(method == "boot_smoth", upci_95, estimate)) %>%
  mutate(variable = reorder(variable, estimate)) %>%
  ggplot(aes(x = variable, y = estimate, color = method)) +
  geom_pointrange(aes(ymin = lowci_95, ymax = upci_95),
                  size = .25, alpha = .75) +
  coord_flip()
```


variable
estimate
−20
0
20
40
method
boot_smooth
cd_lasso
r_lasso