Predicting Down Syndrome From Protein Expression In Mice

Dayoung Yu (dry2115), Justin Hsie (2119), Christian Pascual (cbp2128)

5/18/2019

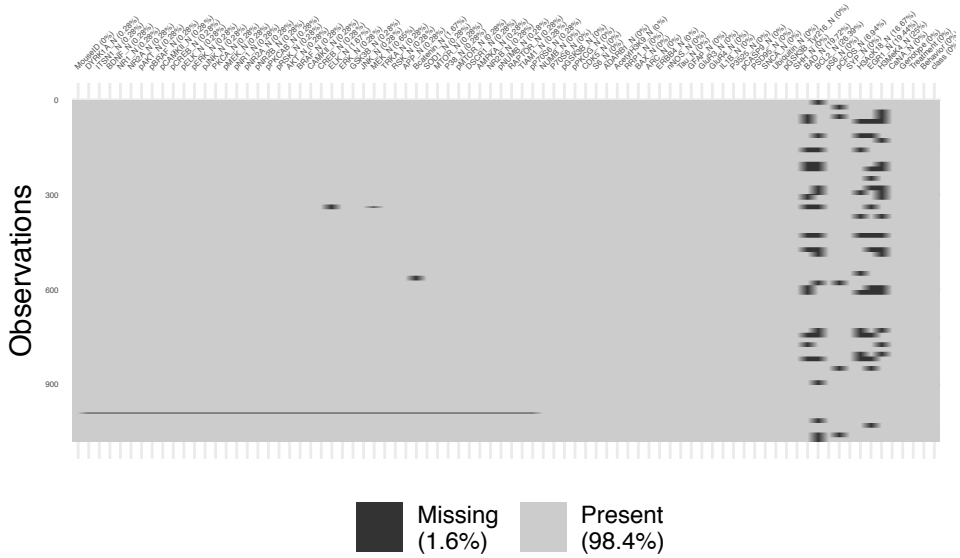
Introduction

Down Syndrome is a genetic disorder that originates from a full or partial extra copy of chromosome 21. The hallmarks of this syndrome include intellectual disability, developmental delays and distinct physical characteristics. Currently, there is no cure to Down Syndrome, but screening and diagnostic tests exist for the condition. Some diagnostic tests such as amniocentisis have been demonstrated to increase the risk of miscarriage, so prenatal screening is preferred. A systematic review by Yao et. al demonstrated that two prenatal screening tests in serum have sensitivities ranging from 77% to 93%. However, amniocentisis is much more predictive of Down Syndrome.

This disparity offers an interesting classification problem: given expression data, can we predict Down Syndrome with comparable accuracy to screening and diagnostic tests? We'll use the *Mice Protein Expression* dataset from Kaggle to explore this question. In its raw form, the dataset contains expression levels for 77 proteins in 1080 mice. These mice were either controls or trisomic, so we can use this dataset for classification purposes.

Data Cleaning

A quick check of the data reveals that it contains missing values. Below we'll visualize how the missing data is distributed.



Two key observations come from this visualization of the missing data: 1) There is a small subset of mice missing almost half of their data, and 2) there is a small subset of proteins that are missing in most mice. To pare down the dataset, we'll remove the mice that are missing most of their proteins and the proteins that are missing in most of the mice.

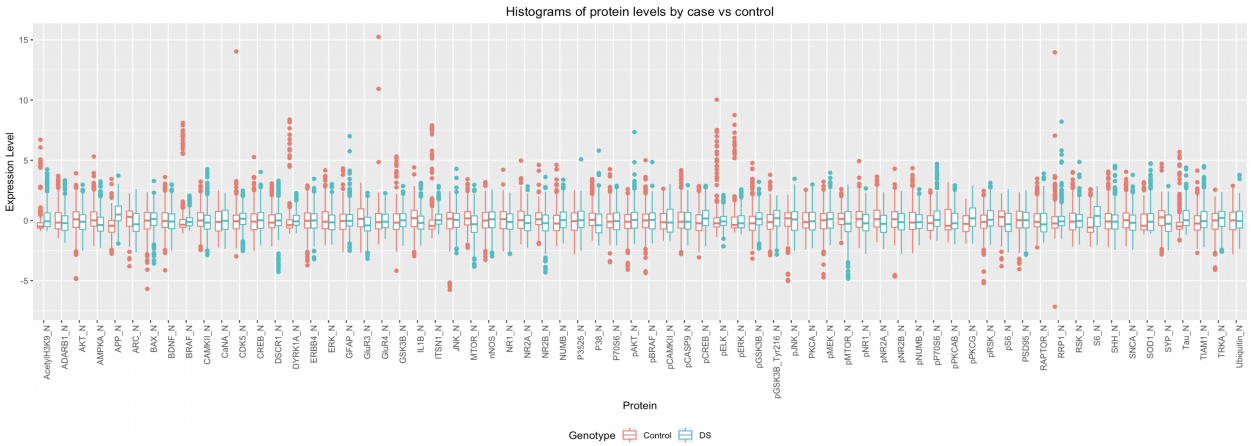


Figure 1: Distribution of protein expression levels by Down Syndrome status

The resulting dataset contains 68 proteins with complete information on 1077 mice. Each of the candidate proteins were also centered and scaled. Trisomic status was relabeled numerically to `0|1`.

Exploratory Data Analysis

A major challenge of this dataset is its high dimensionality. It's highly likely that only a small subset of the proteins are significant in predicting Down Syndrome status. Before we attempt to make predictions on our data, we'll explore and visualize it to inform any modeling we may want to do.

Looking for candidate proteins

The immediate first thing to check is if there are any proteins that help distinguish between cases and controls. Figure 1 lays out these differences.

The major takeaway from Figure 1 is that any differences that exist between the expression levels for each protein are small. There are notably more outliers for control mice for some proteins. No single protein seems to definitively classify cases against controls, so it might be helpful to at least see where the biggest differences are.

Out of the 68 remaining proteins, only 23 have more than a difference of 0.25 between the group means for expression levels. Only 9 have a difference greater than 0.4. This finding supports our suspicion that only a small subset of the proteins offer good discriminatory power.

Examining inter-protein correlation

Biologically, protein expression is done in cascades. Thus, we would expect groups of proteins to be highly correlated with each other, representing the interplay of up and down-regulation. If this is the case, we would be able to further pare down our candidate variables for use in modeling. Figure 3 contains the correlation plot for all the proteins in the dataset.

Figure 3 confirms that there is a high degree of positive correlation within the dataset. These proteins are likely to be within the same pathway, and thus having them in the same model won't help with classification. For our models, we need to use a form of dimension reduction.

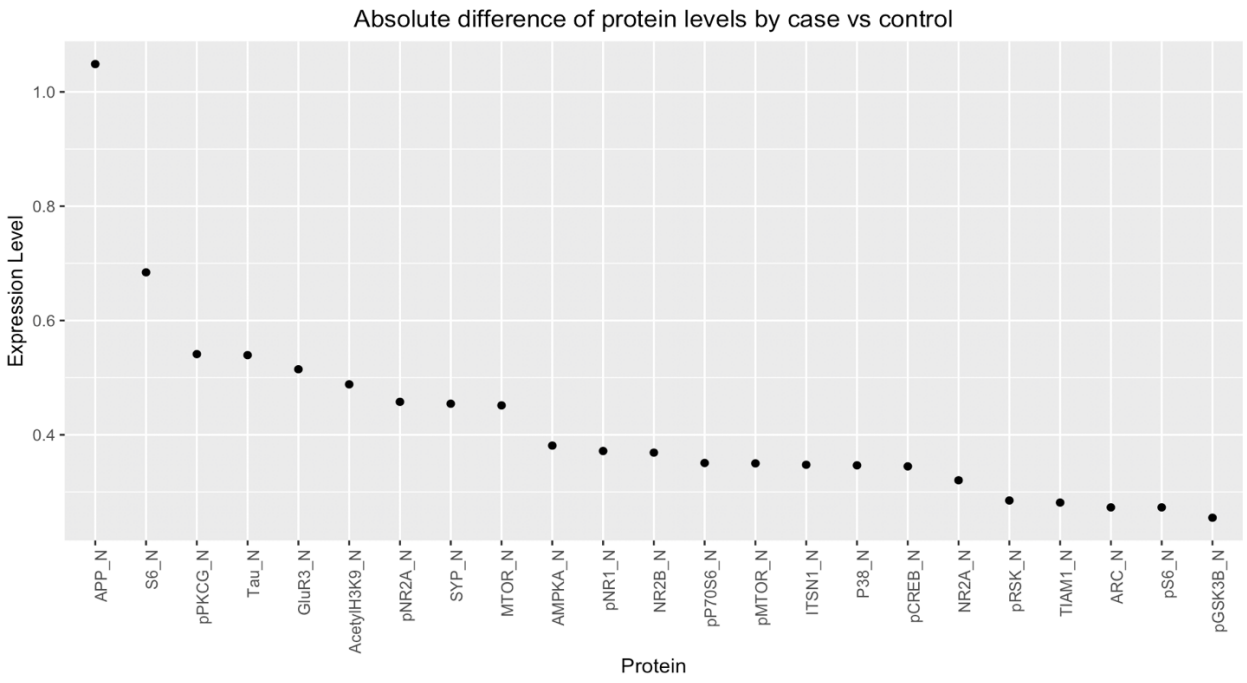


Figure 2: Proteins with at least a difference of 0.25 in their mean protein expression levels by case and control

Principal Component Analysis

Following the findings from the correlation plot, we decided to see if principal component analysis would allow us to reduce the dimensionality of our dataset for supervised learning.

Figure 4 illustrates that a high degree of variance can be explained by just a few principal components. The blue, green and red lines represent the threshold for 80%, 90% and 95% of the variance, respectively. 9 components explain at least 80% of the variance, 18 components explain 90% of the variance, and 27 components explain 95%. For our report, we want to emphasize greater predictive ability, so we'll use the first 27 principal components. We hoped to see that the PCA would cause greater separation between the cases and controls, but we did not see this in plotting the principal components against each other, as seen in Figure 5:

Despite this, we still felt that there was value in using the reduced dataset in prediction. As such, we'll also compare the predictive ability of our models using both the full dataset and the reduced PCA version. Our final model will be chosen based off of how it performs in cross-validation.

Models

The nature of our dataset and the results of our exploratory data analysis led us to pick a set of 5 models to attempt to predict Down Syndrome: a logistic-LASSO, K-Nearest neighbors (KNN), boosting, bagging and a support vector machine (SVM). With the high dimensionality of the dataset, we thought that the logistic-LASSO would provide some useful variable selection to narrow down the full set of proteins to just those that had true associations, though it assumes a linear relationship between the proteins and the probability of having Down Syndrome.

We saw that many of the differences in mean protein expression level were extremely slight between cases and controls, so this led us to boosting and bagging; we thought that perhaps the aggregation of trees that

Table 1: Table 1: CV and test error

Model	Full: CV Error	PCA: CV Error	Full: Test Error	PCA: Test Error
Logistic-LASSO	0.03590	0.04873	0.03721	0.05581
KNN	0.00233	0.00233	0.00000	0.00000
Boosting	0.01512	0.01744	0.00930	0.03256
Bagging	0.05271	0.04623	0.04651	0.08372
SVM	0.00116	0.00346	0.00465	0.00465

incorporate all the truly necessary proteins into the decision would produce better results and take advantage of all these small differences. Though we sacrifice model interpretability, we hoped to gain in predictive power.

Biological intuition led us to choose KNN and SVM with a radial kernel; since we found that many of the proteins were correlated with each other, we thought that the cases and controls might be clustered to each other on a higher dimesion than we could see with just simple visualization. KNN and SVM would take advantage of this. Like bagging and boosting, we knew we would exchange interpretability for prediction. KNN suffers in high dimensional problems, whereas SVM does not.

With each of these models, we used `caret` to select a set of tuning parameters that produced the best cross-validation classification error. We also used this metric to pick our final model. In deciding a proper range for tuning parameters, we experimented and investigated which range of values properly captured a maximum value before doing the comparisons.

Table 1 below lays out the cross-validation and test error for each of the models and for both the full and PCA-reduced dataset.

The table demonstrates that the CV error approximates the test error well. A curiousity that arose during analysis was that the KNN model had perfect test performance. The model that performed the best in cross-validation was the SVM model, so we choose it as our final model instead of KNN. Our final choice was influenced by the fact that our dataset was high dimensional, so the SVM was ultimately better suited to this problem. It performed the best in the full dataset, and even second-best for the reduced dataset. The best tune for our radial-kernel was `cost = 1` and $\sigma = 0.04798$.

One of the main limitations of the support vector machine is its diminished interpretability, and this is compounded by the high dimensionality of our dataset. The black-box nature of the SVM makes it difficult to ascertain which variables play a role in predicting Down Syndrome status. In our exploratory data analysis, we found the proteins that had the greatest mean difference between cases and controls. We used this to our advantage and visualized how well the SVM classified the observations along the top 2 proteins with the largest differences.

We can see in Figure 6 that the cases (red) have a reasonable amount of separation from the controls (black), and that the SVM boundary captures this weak separation well. This plot represents the best case scenario, but it at least helps convince us that the SVM is doing a satisfactory job of separation in a difficult context.

Conclusions

We conclude that the radial SVM is an excellent model for this problem, but note that while it maximizes predictive value, it poses a potential problem for researchers looking to understand the relationships between the proteins and the condition. We expected SVM to excel in our high dimensional problem, and it proved to be useful with both the full and the PCA-reduced datasets. We thought that having PCA dataset might help in better dividing the cases and controls, but we did not see this in the cross-validation and test errors. Despite the lack of interpretability from the SVM, the success of the model indicates that even small differences in many, many proteins can help discern between a control and a case. With a condition as life-altering as Down

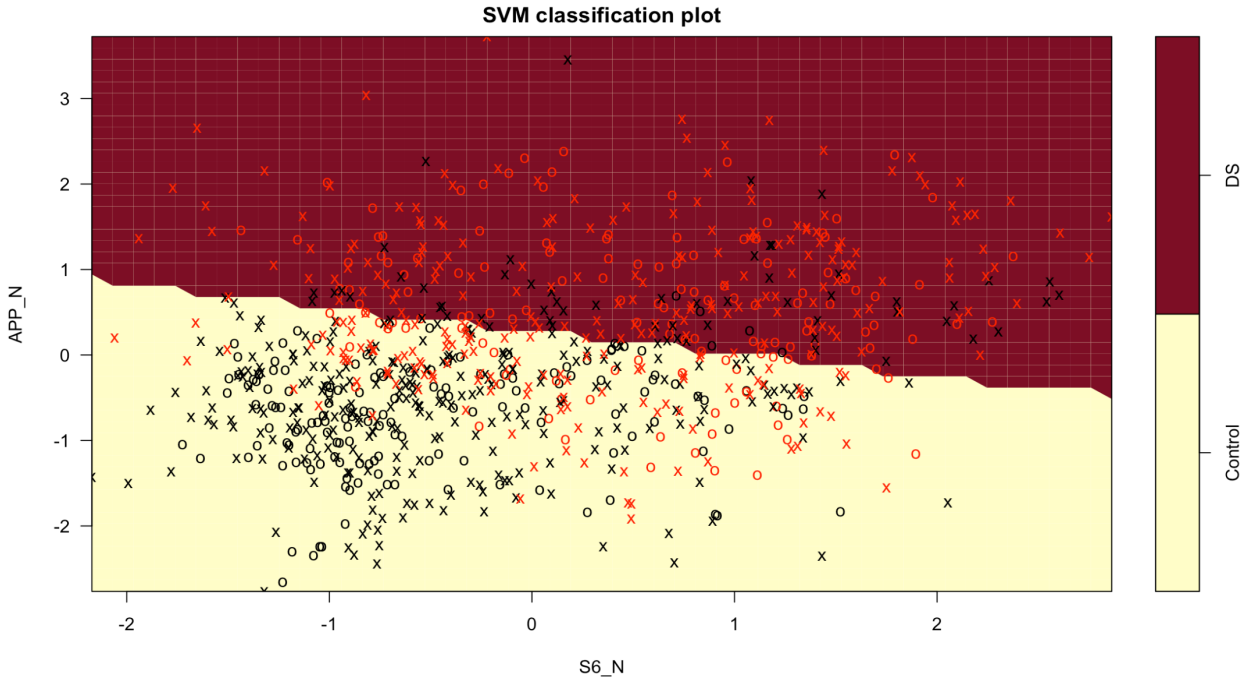


Figure 6: 2D view of the hyperplane along the proteins with the greatest mean differences

Syndrome, we may want to accept the lack of interpretability if it helps us detect it with greater precision than current diagnostic tools allow.

References

1. Yao Y, Liao Y, Han M, Li SL, Luo J, Zhang B. Two kinds of common prenatal screening tests for Down's Syndrome: a systematic review and meta-analysis. Sci Rep. 2016;6:18866. Published 2016 Jan 6. doi:10.1038/srep18866

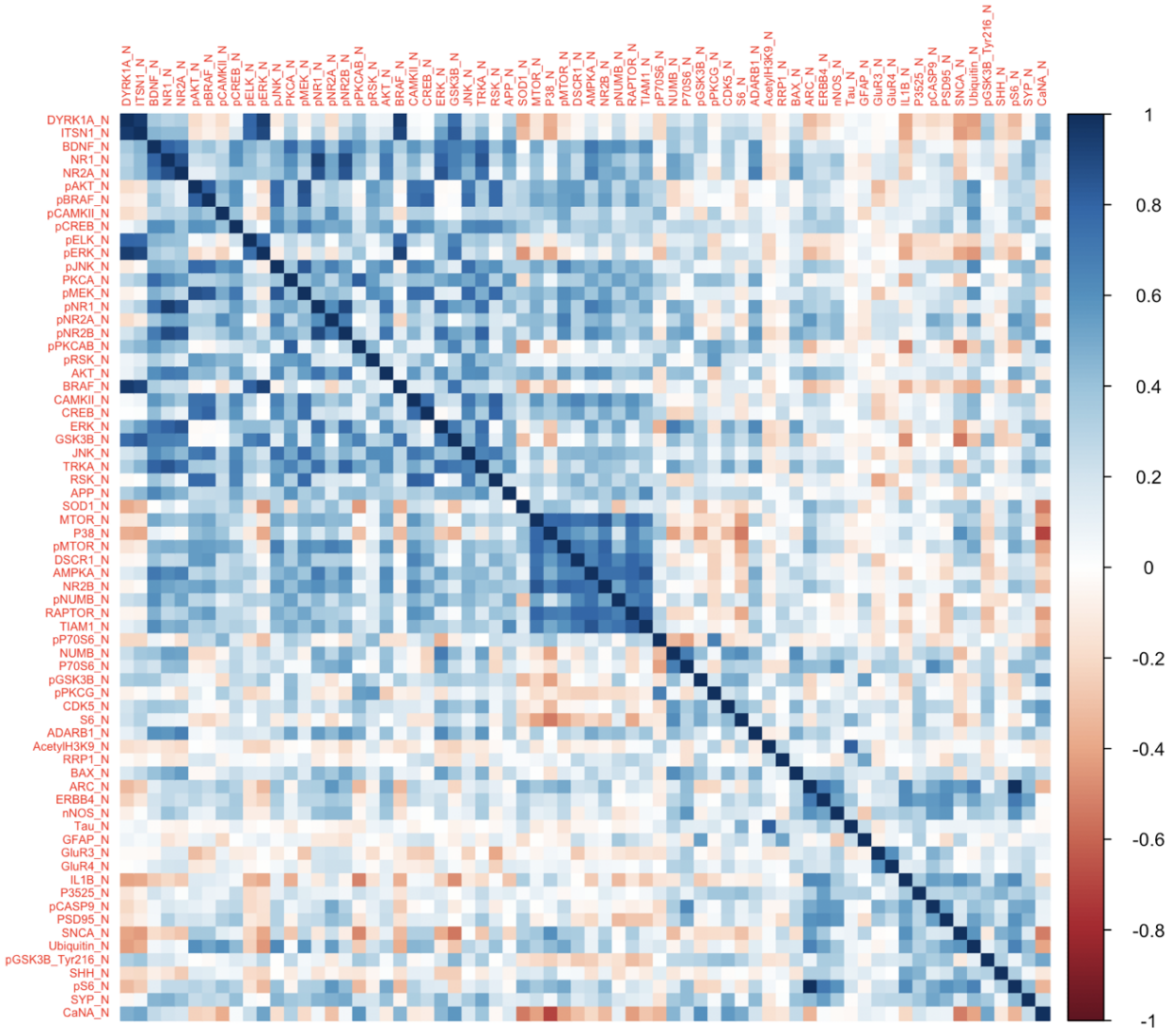


Figure 3: Correlation matrix for all 68 proteins in dataset

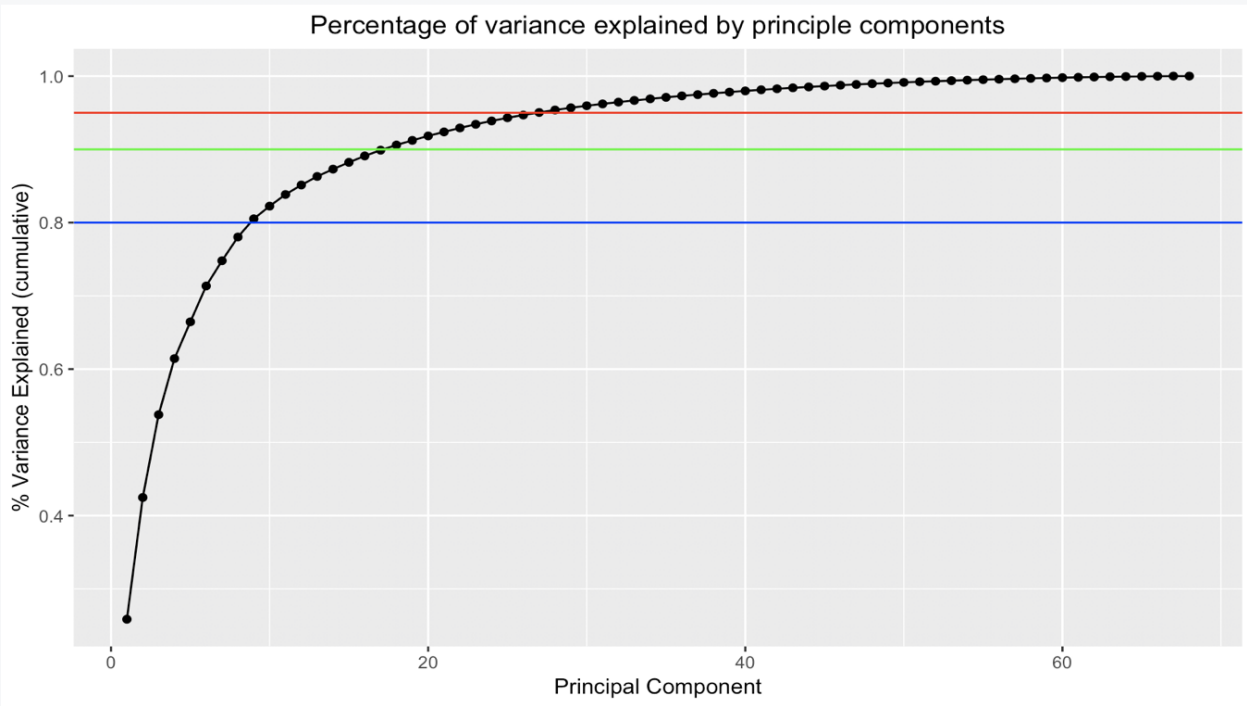


Figure 4: Percentage of variance explained by principal components

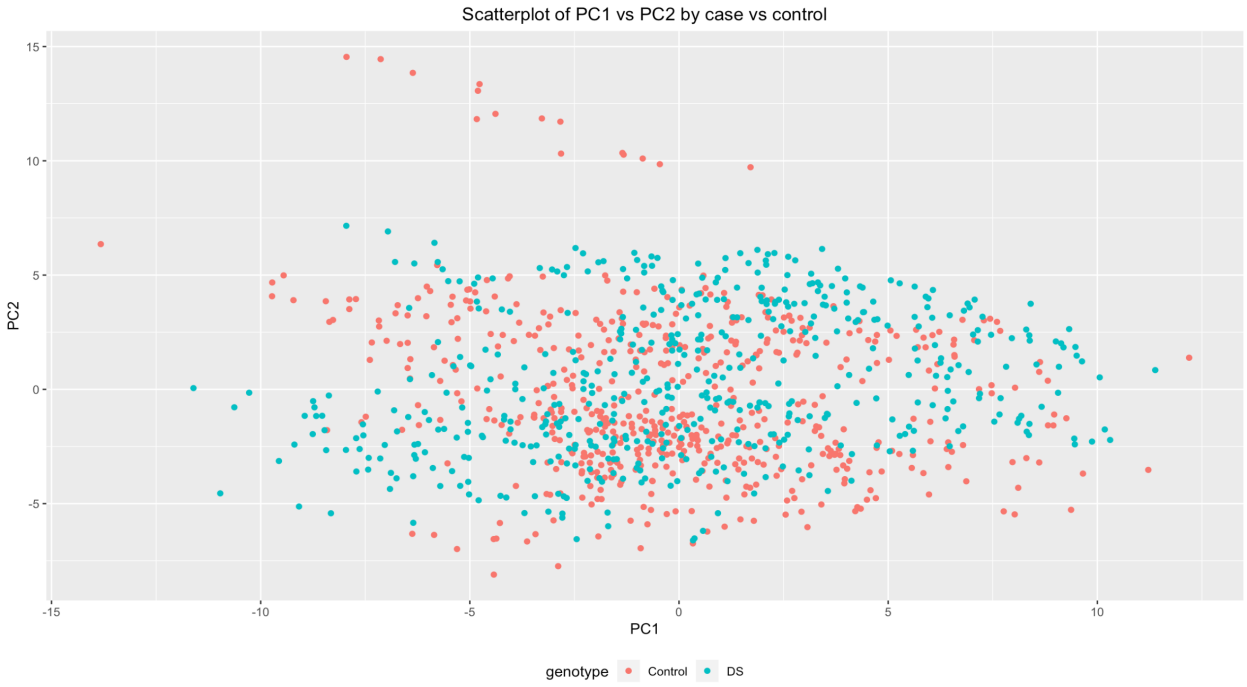


Figure 5: PCA did not seem to result in good separation between cases and controls

```{r, eval = FALSE }
library(tidyverse)
library(glmnet)
library(parallel)
library(doParallel)
library(foreach)
library(iterators)
source("LogLasso.R")

nCores = 6 # change to whatever fits your specific system
registerDoParallel(nCores)

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

ds = read.csv("Down.csv") %>%
  mutate(
    class = ifelse(Class == "Control", 0, 1)
    ) %>%
  select(-MouseID, -Class) %>%
  na.omit() # getting rid of nas since glmnet cannot handle missing data

ds.X = ds %>% select(-class) %>% map_df(.x = ., standardize)
ds.X = ds.X %>% mutate(intercept = 1) %>% select(intercept, everything())
ds.y = ds$class
```

Setup

```{r setup, eval = FALSE }
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(glmnet)
library(doParallel)
source("LogLasso.R")
cl<-makeCluster(2) #change the 2 to your number of CPU cores
registerDoParallel(cl)
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

Reading and cleaning the data
```{r, eval = FALSE }
down_dat = read_csv("Down.csv") %>% select(-MouseID) %>% mutate(Class = as.factor(Class)) %>%
  na.omit() #getting rid of nas since glmnet cannot handle missing data
y_full = as.matrix(ifelse(down_dat$Class == "Control",0,1))
x_full = as.matrix(down_dat %>% select(-Class))
set.seed(1001)
#Geting the trained dataset
train_index = sample(1:nrow(x_full), ceiling(nrow(x_full) * 0.9))
y_train = y_full[train_index]
x_train = x_full[train_index,]
```

```
#getting the test dataset
y_test = y_full[-train_index]
x_test = x_full[-train_index,]
```

Fitting and predicting using our model and built in lasso regression function

```{r, eval = FALSE}
#Our model
start = Sys.time()
log_lasso = optimize(as.data.frame(x_train),y_train, lambda_seq, rep(0.01, 77))
our_model = log_lasso$finalModel
y_predict_our_model = predict(our_model,x_test)
y_response_our_model = as.numeric(y_predict_our_model > 0.5)
end = Sys.time()
#reuslts
accuracy_our_model = mean(y_response_our_model == y_test)
mse_our_model = mean((y_predict_our_model - y_test)^2)
time_our_model = end - start
lambda_our_model = log_lasso$lambda.min
#Compare with the built in lasso
start = Sys.time()
lasso_glmnet_cv = cv.glmnet(x_train, y_train, alpha = 1, family = "binomial", lambda =
lambda_seq)
y_response_glmnet = predict.cv.glmnet(lasso_glmnet_cv, newx = x_test, type = "response", s =
"lambda.min")
end = Sys.time()
#results
accuracy_glmnet = mean((y_response_glmnet>0.5) == y_test)
mse_glmnet = mean((y_response_glmnet - y_test)^2)
time_glmnet = end - start
lambda_glmnet = lasso_glmnet_cv$lambda.min
```

Building the bootstrap-smoothing function
```{r, eval = FALSE }
boot_smooth <- function(x_train,y_train,x_test,y_test,lambda_seq,iterations=100){
  predictions <- foreach(m = 1:iterations,.combine = cbind, .packages = "glmnet") %dopar% {
    boot_positions <- sample(nrow(x_train), size = nrow(x_train), replace = T)
    boot_pos <- 1:nrow(x_train) %in% boot_positions
    y = y_train[boot_pos]
    x = x_train[boot_pos,]
    cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial", lambda = lambda_seq)
    # Fit the final model on the training data
    model <- glmnet(x, y, alpha = 1, family = "binomial",
                lambda = cv.lasso$lambda.min)
    predict(model,newx = x_test, type = "response")
  }
  apply(predictions, 1, mean) #Getting the mean
}

```

Prediction for bootstrap-smoothing approach
```{r, eval = FALSE }
start = Sys.time()
```

```
y_response_boot = boot_smooth(x_train,y_train,x_test,y_test, lambda_seq = lambda_seq,
iterations = 100)
end = Sys.time()
#results
accuracy_boot = mean((y_response_boot > 0.5) == y_test)
mse_boot = mean((y_response_boot - y_test)^2)
time_boot = end - start
lambda_boot = NA
```

Results comparing the three methods
```{r, eval = FALSE }
#Build a result table
result_frame = data.frame(model = c("Our lasso", "Built in cv lasso", "Boot smooth mehtod"),
accuracy = c(accuracy_our_model, accuracy_glmnet, accuracy_boot),
           MSE = c(mse_our_model, mse_glmnet, mse_boot), time =
c(time_our_model,time_glmnet,time_boot),
           lambda_min = c(lambda_our_model, lambda_glmnet, lambda_boot))
knitr::kable(result_frame)
```

```{r, eval = FALSE }
# Set the index for 5 fold cross-validation
set.seed(1000)
n_folds <- 4
folds_i <- sample(rep(1:n_folds, length.out = nrow(x_full)))
result_pool = data.frame()
for (boot_n in c(50,100,200,300)) {#Number of bootstrap samples for smooth bootstraping
  cv_tmp <- matrix(NA, nrow = n_folds, ncol = 4)
  for (k in 1:n_folds) {
    #getting the train and test dataset for this fold
    test_i <- which(folds_i == k)
    x_test = x_full[test_i,]
    y_test = y_full[test_i]
    x_train = x_full[-test_i,]
    y_train = y_full[-test_i]

    #Laso
    lasso_glmnet_cv = cv.glmnet(x_train, y_train, alpha = 1, family = "binomial", lambda =
lambda_seq)
    y_response_glmnet = predict.cv.glmnet(lasso_glmnet_cv, newx = x_test, type = "response", s
= "lambda.min")
    #results
    accuracy_glmnet = mean((y_response_glmnet>0.5) == y_test)
    mse_glmnet = mean((y_response_glmnet - y_test)^2)

    #bootstrap smooth
    y_response_boot = boot_smooth(x_train,y_train,x_test,y_test, lambda_seq = lambda_seq,
iterations = boot_n)
    #results
    accuracy_boot = mean((y_response_boot > 0.5) == y_test)
    mse_boot = mean((y_response_boot - y_test)^2)

    #Building the result dataframe
    cv_tmp[k, 1] = accuracy_glmnet
    cv_tmp[k, 2] = accuracy_boot
    cv_tmp[k, 3] = mse_glmnet
```

```
    cv_tmp[k, 4] = mse_boot
  }
  result_cv = as.data.frame(cv_tmp)
  names(result_cv) = c("accuracy_glmnet", "accuracy_boot", "mse_glmnet", "mse_boot")
  result_cv = colMeans(result_cv)
  result_df = data.frame(Model = c("Logistic-Lasso", "Bootstrap smoothing"),
                         accuracy = c(result_cv[["accuracy_glmnet"]],
result_cv[["accuracy_boot"]]),
                         MSE = c(result_cv[["mse_glmnet"]], result_cv[["mse_boot"]]),
                         Boot_sample = c(1,boot_n))
  result_pool = rbind(result_pool, result_df)
  print(c("finished ", boot_n))
}
result_pool = unique(result_pool) #Logistic-lasso results are repeated
knitr::kable(result_pool)
```

```{r, eval = FALSE }
result_pool %>% filter(Boot_sample > 2) %>% ggplot(aes(x = Boot_sample, y = MSE)) +
geom_line() +
  labs(title = "The bootstraped sample size for each prediction and prediction error")
```

```{r, eval = FALSE }
boot_smooth_select_feature <- function(x,y,lambda_seq,iterations=100){
  coef_boot <- foreach(m = 1:iterations,.combine = cbind, .packages = c("glmnet","tidyverse"))
%dopar% {
    boot_positions <- sample(nrow(x), size = nrow(x), replace = T)
    boot_pos <- 1:nrow(x) %in% boot_positions
    y_boot = y[boot_pos]
    x_boot = x[boot_pos,]
    cv.lasso <- cv.glmnet(x_boot, y_boot, alpha = 1, family = "binomial", lambda = lambda_seq)
    # Fit the final model on the training data
    model <- glmnet(x_boot, y_boot, alpha = 1, family = "binomial",
                lambda = cv.lasso$lambda.min)
    coef(model)
  }
  approach1 = !(apply(coef_boot, 1, quantile, probs = c(0.025, 0.975)) %>%
apply(.,2,is.element,0) %>% apply(.,2,any))
  approach2 = !(apply(coef_boot, 1, is.element, 0) %>% apply(.,2, any))
  approach3 = !(apply(coef_boot, 1, is.element, 0) %>% apply(.,2, all))
  approach4 = !(apply(coef_boot, 1, Mode)  %>% is.element(.,0))
  result = cbind(approach1,approach2,approach3, approach4)
  return(result)
 }
```

Using this algorithm on the full dataset
```{r, eval = FALSE }
result = boot_smooth_select_feature(x_full,y_full, lambda_seq, iterations = 10)
knitr::kable(result)
```

```{r, eval = FALSE }
boot.count = function(idx, data) {
  counts = NULL
```

```
  for (i in 1:nrow(data)) {
    counts[i] = length(which(idx == i))
  }
  return(counts)
}
```

```{r, eval = FALSE }
set.seed(8160)
# Custom function to combine multiple outputs in parallel
comb = function(x, ...) {
  mapply(cbind, x, ..., SIMPLIFY=FALSE)
}

# Parallelize the bootstrapping
B = 1000
bs = foreach(i = 1:B, .combine = "comb") %dopar% {
  # Create the bootstrap sample
  boot.idx = sample(nrow(ds), size = nrow(ds), replace = T)
  boot.X = ds.X[boot.idx,] %>% as.matrix(.)
  boot.y = ds.y[boot.idx] %>% as.matrix(.)

  # Calculate the count matrix
  Yij = boot.count(boot.idx, boot.X)

  # Cross-validate for lambda and then fit this optimal model
  cv.lasso = cv.glmnet(boot.X, boot.y, alpha = 1, family = "binomial")
  best.lasso = glmnet(boot.X, boot.y, alpha = 1, lambda = cv.lasso$lambda.min)

  # Extract the coefficients and store in a matrix
  coeffs = matrix(coef(best.lasso))

  list(coeff.mat = coeffs, Yij.mat = Yij)
  }

# Finishing off the calculations
Y.j = rowSums(bs$Yij.mat)/B
t.j = rowSums(bs$coeff.mat)/B
t.diff = bs$coeff.mat - t.j
Yij.diff = bs$Yij.mat - Y.j
```

```{r, eval = FALSE }
# Perform this action for each coefficient
cov.mat = NULL
for (b in 1:78) {
  covs = NULL
  for (j in 1:nrow(ds)) {
    # Each individual point will have a bootstrap
    # covariance with the bootstrap statistic
    cov.j = 0
    for (i in 1:B) {
      cov.j = cov.j + ((bs$Yij.mat[j,i] - Y.j[j]) * (bs$coeff.mat[b,i] - t.j[b]))/B
    }
    covs = c(covs, cov.j)
  }
  cov.mat = rbind(cov.mat, covs)
```

```
}
```

```{r, eval = FALSE }
# With the calculated covariances for each point,
# calculate the smooth bootstrap standard deviation
sdbs = rep(0, 78)
for (i in 1:nrow(cov.mat)) {
  sdbs[i] = (sum(cov.mat[i,] * cov.mat[i,]))^(1/2)
}

# Calculate the standard bootstrap confidence intervals
standard.std = NULL
for (i in 1:nrow(cov.mat)) {
  std = sum((1/B) * (bs$coeff.mat[i,] - mean(bs$coeff.mat[i,]))^2)^(1/2)
  standard.std = c(standard.std, std)
}

# Calculate the percentile confidence intervals
p025 = NULL
p975 = NULL
for (i in 1:nrow(cov.mat)) {
  p025 = c(p025, quantile(t.diff[i,], 0.025))
  p975 = c(p975, quantile(t.diff[i,], 0.975))
}

conf.mat = as_tibble(cbind(t.j, sdbs, standard.std, p025, p975))
conf.mat.extra = conf.mat %>%
  mutate(
    efron.lb = t.j - qnorm(0.975) * sdbs,
    efron.ub = t.j + qnorm(0.975) * sdbs,
    is.significant = ifelse((efron.lb <= 0 & 0 <= efron.ub), FALSE, TRUE),
    idx = 1:length(sdbs)
  )
```

```{r, eval = FALSE }
# Version highlighting only the significant proteins
ggplot(data = conf.mat.extra) +
  geom_point(aes(x = idx, y = t.j), data = subset(conf.mat.extra, is.significant == TRUE),
             color = "blue") +
  geom_errorbar(aes(x = idx, ymin = t.j - qnorm(0.975) * sdbs, ymax = t.j + qnorm(0.975) *
sdbs),
                data = subset(conf.mat.extra, is.significant == TRUE), color = "blue") +
  geom_point(aes(x = idx, y = t.j), data = subset(conf.mat.extra, is.significant == FALSE),
             color = "blue", alpha = 0.1) +
  geom_errorbar(aes(x = idx, ymin = t.j - qnorm(0.975) * sdbs, ymax = t.j + qnorm(0.975) *
sdbs),
                color = "blue", alpha = 0.1, data = conf.mat.extra) +
  geom_errorbar(aes(x = idx, ymin = p025, ymax = p975),
                color = "red", alpha = 0.2, data = conf.mat.extra) +
  theme_bw() +
  labs(
    title = "All significant proteins, according to Efron's smoothed interval",
    x = "Protein index",
    y = "Estimated coefficient"
  )
```

```
# Version highlighting the percentile intervals
ggplot(data = conf.mat.extra) +
  geom_point(aes(x = idx, y = t.j), data = subset(conf.mat.extra, is.significant == TRUE),
             color = "blue", alpha = 0.1) +
  geom_errorbar(aes(x = idx, ymin = t.j - qnorm(0.975) * sdbs, ymax = t.j + qnorm(0.975) *
sdbs),
                data = subset(conf.mat.extra, is.significant == TRUE), color = "blue", alpha =
0.1) +
  geom_point(aes(x = idx, y = t.j), data = subset(conf.mat.extra, is.significant == FALSE),
             color = "blue", alpha = 0.1) +
  geom_errorbar(aes(x = idx, ymin = t.j - qnorm(0.975) * sdbs, ymax = t.j + qnorm(0.975) *
sdbs),
                color = "blue", alpha = 0.1, data = conf.mat.extra) +
  geom_errorbar(aes(x = idx, ymin = p025, ymax = p975),
                color = "red", data = conf.mat.extra) +
  theme_bw() +
  labs(
    title = "Bootstrap percentile interval",
    x = "Protein index",
    y = "Estimated coefficient"
  )

# Version highlighting how close the standard and bootstrap intervals are
ggplot(data = conf.mat.extra) +
  geom_point(aes(x = idx, y = t.j), data = subset(conf.mat.extra, is.significant == TRUE),
             color = "blue") +
  geom_errorbar(aes(x = idx, ymin = t.j - qnorm(0.975) * sdbs, ymax = t.j + qnorm(0.975) *
sdbs),
                data = subset(conf.mat.extra, is.significant == TRUE), color = "blue") +
  geom_point(aes(x = idx + 1, y = t.j),
             color = "green", data = subset(conf.mat.extra, is.significant == TRUE)) +
  geom_errorbar(aes(x = idx + 1, ymin = t.j - qnorm(0.975) * standard.std,
                    ymax = t.j + qnorm(0.975) * standard.std),
                color = "green", data = subset(conf.mat.extra, is.significant == TRUE)) +
  geom_point(aes(x = idx, y = t.j), data = subset(conf.mat.extra, is.significant == FALSE),
             color = "blue", alpha = 0) +
  geom_errorbar(aes(x = idx, ymin = t.j - qnorm(0.975) * sdbs, ymax = t.j + qnorm(0.975) *
sdbs),
                color = "blue", alpha = 0, data = conf.mat.extra) +
  theme_bw() +
  labs(
    title = "Comparison of standard bootstrap confidence interval against Efron's",
    x = "Protein index",
    y = "Estimated coefficient"
  )
```

```{r, eval = FALSE }
# Need to extract names of significant proteins
sigs = conf.mat.extra %>%
  mutate(protein = c("Intercept", colnames(ds.X))) %>%
  filter(is.significant == TRUE) %>%
  select(protein, t.j, efron.lb, efron.ub)

knitr::kable(sigs)
```