PHS 597 – Homework 2 – Lasso and Cyclic Coordinate Descent – Fall 2021

Havell Markus, hmm5304@psu.edu

October 6, 2021

Assginment description:

- In this assignment we will implement the Cyclic Coordinate Descent algorithm to solve Lasso Regression with multiple predictors.
- I will compare the beta values with *glmnet* solution.
- I will then apply the Lasso regression implementation from *glmnet* to GEUVADIS data to predict gene expression using genotype information.

Part 1, implementation of Cyclic Coordinate Descent algorithm

- I will assume that each predictor is standardized to have mean 0 and standard deviation of 1.
- I will also assume that the outcome Y is standardized to have mean 0.
- I will first initialize all the $\hat{\beta}$ to 1
- I will the repeatedly cycle through each predictor in a sequential order from $j = 1, 2, ..., p$ where at j^{th} step, I will update coefficient $\hat{\beta}_j$ by minimizing the objective function while hold other $\hat{\beta}_k, k \neq j$ at their current values.
- I will calculate equation 1 using equation 2 and equation 3 for each $\hat{\beta}_j$
- I will loop through until the $\hat{\beta}$ converges, or do not change more than some tolerance value.

$$\hat{\beta}_j = S_\lambda\big(\frac{1}{N}\langle x_j, r^{(j)}\rangle\big) \tag{1}$$

$$r_i^{(j)} = y_i - \sum_{k \neq j} x_{ik}\hat{\beta}_k \tag{2}$$

$$S_\lambda\big(\frac{1}{N}\langle x_j, r^{(j)}\rangle\big) = \begin{cases} \frac{1}{N}\langle z, y\rangle - \lambda & if\ \frac{1}{N}\langle z, y\rangle > \lambda, \\ 0 & if\ \frac{1}{N}|\langle z, y\rangle| \leq \lambda, \\ \frac{1}{N}\langle z, y\rangle + \lambda & if\ \frac{1}{N}\langle z, y\rangle < -\lambda \end{cases} \tag{3}$$

```
## Equation 3
soft_threshold <- function(rho, lamda){
  if(rho < (-1*lamda)){
    return(rho + lamda)
  } else if(rho > lamda){
    return(rho - lamda)
  } else {
    return(0)
```

```
  }
}

coordinate_descent_lasso <- function(x_df, y_df, lambda, tolerance=0.0001){

  beta_vector <- rep(1, ncol(x_df))
  number_of_predictors <- ncol(x_df)
  number_of_samples <- nrow(x_df)
  change <-  rep(0, ncol(x_df))
  converge <- TRUE
  i <- 0

  while(converge){
    i <- i + 1
    for(j in 1:number_of_predictors){
      x_j <- x_df[,j]
      y_j <- y_df - (x_df[,-j] %*% beta_vector[-j])
      ## Equation 2
      rho_j <- as.numeric((x_j %*% y_j)/number_of_samples)

      old_weight <- beta_vector[j]
      ## Equation 1
      new_weight <- soft_threshold(rho_j, lambda)
      beta_vector[j] <- new_weight

      change[j] <- abs(new_weight - old_weight)
    }
    max_change <- max(change)
    if(max_change < tolerance){
      converge <- FALSE
    }
  }

  message(paste0("Total number of iterations: ", i))
  return(beta_vector)
}
```

Part 2, implementation to gene expression data

- The expression dataset has expression values from 358 samples and for 545 genes
- The genotype dataset has 133,965 SNPs data from 358 samples
- We remove SNPs that have the same state in every samples

```
#############################################################################
## 545 genes and 358 samples
expression_df <- fread("/storage/home/hmm5304/scratch/gradclass/gene_expression_sample/GEUVADIS_normali
expression_df[1:4,1:6]
```

```
##            gene_id chromosome    start      end  HG00096  HG00097
## 1: ENSG00000000419         20 49551404 49575092 28.29987 23.46323
## 2: ENSG00000020256         20 50668202 50820847 10.40689 10.55414
## 3: ENSG00000022277         20 55043647 55093943 48.91700 48.59284
## 4: ENSG00000025293         20 34359896 34538303 13.68761 17.04665
```

```r
dim(expression_df)
```

```
## [1] 545 362
```

```r
## 133965 SNPs and 358 samples
genotype_df <- fread("/storage/home/hmm5304/scratch/gradclass/gene_expression_sample/GEUVADIS_chr20_pro
genotype_df[1:4,1:6]
```

```
##    CHR               SNP (C)M   POS COUNTED ALT
## 1:  20  20_61098_C_T_b37    0 61098       C   T
## 2:  20 20_61138_C_CT_b37    0 61138       C  CT
## 3:  20  20_61795_G_T_b37    0 61795       G   T
## 4:  20  20_62731_C_A_b37    0 62731       C   A
```

```r
dim(genotype_df)
```

```
## [1] 133965    364
```

```r
rows_w_same_genotype <- which(apply(genotype_df[,-c(1:6)], 1, function(x) length(unique(x))) == 1)
length(rows_w_same_genotype)
```

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

```r
genotype_df <- genotype_df[-rows_w_same_genotype, ]
```

- Split the data into training and testing dataset
  - I will use 80% of the sample for traning, and 20% for testing.
- I will also define a function for calculating mean squared error.
  - The function is called **mse**

```r
##############################################################################
## set training and test samples, 75:25 split
set.seed(123)
in_train <- sample(1:358, round(358*0.8))
in_test <- c(1:358)[-in_train]

# Defines the Mean Square Error function
mse = function(x,y) { mean((x-y)^2)}
```

- I will select SNPs within $\pm 500,000$ bp of TSS of the gene
- I will the split the data into training and testing set
- For training data, I will standardize each SNP to have mean 0 and standard deviation of 1
- For test data, I will also standardize the gene expression value Y to have mean 0

```
#############################################################################
## Running for 1 example using glmnet
## one gene expression vs. SNPs with in 500kb window

i <- 1

snps_in_gene_ranges <- genotype_df %>% filter(CHR == expression_df$chromosome[i] &
                                                POS >= (expression_df$start[i] - 500000) &
                                                POS <= (expression_df$end[i] + 500000))

genotype_mat <- snps_in_gene_ranges[,-c(1:6)] %>% data.matrix() %>% t()
expression_mat <- expression_df[i,-c(1:4)] %>% data.matrix() %>% t()

genotype_mat_train <- genotype_mat[in_train,]
genotype_mat_test <- genotype_mat[in_test,]

expression_mat_train <- expression_mat[in_train,] %>% data.matrix()
expression_mat_test <- expression_mat[in_test,] %>% data.matrix()

## Centering Y to have mean 0
expression_mat_train_scaled <- scale(expression_mat_train[,1], center = TRUE, scale = FALSE)
## Centering each column (feature) to 0 and having SD = 1
genotype_mat_train_scaled <- scale(genotype_mat_train, center = TRUE, scale = TRUE)
```

- I will next apply Cyclic Coordinate Descent algorithm on the training data for one gene
- I will try different values of $\lambda = 1, 0.5, 0.25, 0.1, 0.01$

```
bata_3 <- coordinate_descent_lasso(x_df = genotype_mat_train_scaled, y_df = expression_mat_train_scaled
bata_2 <- coordinate_descent_lasso(x_df = genotype_mat_train_scaled, y_df = expression_mat_train_scaled
bata_1 <- coordinate_descent_lasso(x_df = genotype_mat_train_scaled, y_df = expression_mat_train_scaled
bata_0 <- coordinate_descent_lasso(x_df = genotype_mat_train_scaled, y_df = expression_mat_train_scaled
```

Part 3, comparing the results with *glmnet*

- I will next apply *glmnet* to the training data for one gene
- I will get the $\hat{\beta}$ value for $\lambda = 1, 0.5, 0.25, 0.1, 0.01$
- Since the data was normalized, the intercept value should be 0, which it is.

```
set.seed(480)
lasso_fit <- glmnet(x = genotype_mat_train_scaled, y = expression_mat_train_scaled , alpha = 1)
lasso_coef3 <- coef(lasso_fit, s = 0.1) %>% as.matrix() %>% as.data.frame()
lasso_coef2 <- coef(lasso_fit, s = 0.25) %>% as.matrix() %>% as.data.frame()
lasso_coef1 <- coef(lasso_fit, s = 0.5) %>% as.matrix() %>% as.data.frame()
lasso_coef0 <- coef(lasso_fit, s = 1) %>% as.matrix() %>% as.data.frame()

c(lasso_coef3$s1[1], lasso_coef2$s1[1], lasso_coef1$s1[1], lasso_coef0$s1[1])
```

```
## [1] -1.059792e-17 -1.126660e-16 -1.009121e-16 -1.009121e-16
```

- I will next compare the $\hat{\beta}$ values from the two models
- The MSE is small, and most of the $\hat{\beta}$ values are similar for the two methods for $\lambda = 1, 0.5, 0.25, 0.1, 0.01$

- For large $\lambda = 0.5$ and $\lambda = 1$, the $\hat{\beta}$ values go to 0, which is expected as it minimizes the objective function.

```
compare_beta <- function(b1, b2, lamda_value = ""){

  message(paste0("For ", lamda_value,"  , the MSE between two beta estimates is: ", mse(b1, b2)))
  plot(b1, b2, main = lamda_value, xlab = "cyclic_cordinate_descent_beta", ylab = "glmnet_beta")
}

par(mfrow=c(2,2))
compare_beta(bata_3, lasso_coef3$s1[-1], lamda_value = "Lambda = 0.1")
```

```
## For Lambda = 0.1  , the MSE between two beta estimates is: 2.45432413039776e-05
```

```
compare_beta(bata_2, lasso_coef2$s1[-1], lamda_value = "Lambda = 0.25")
```

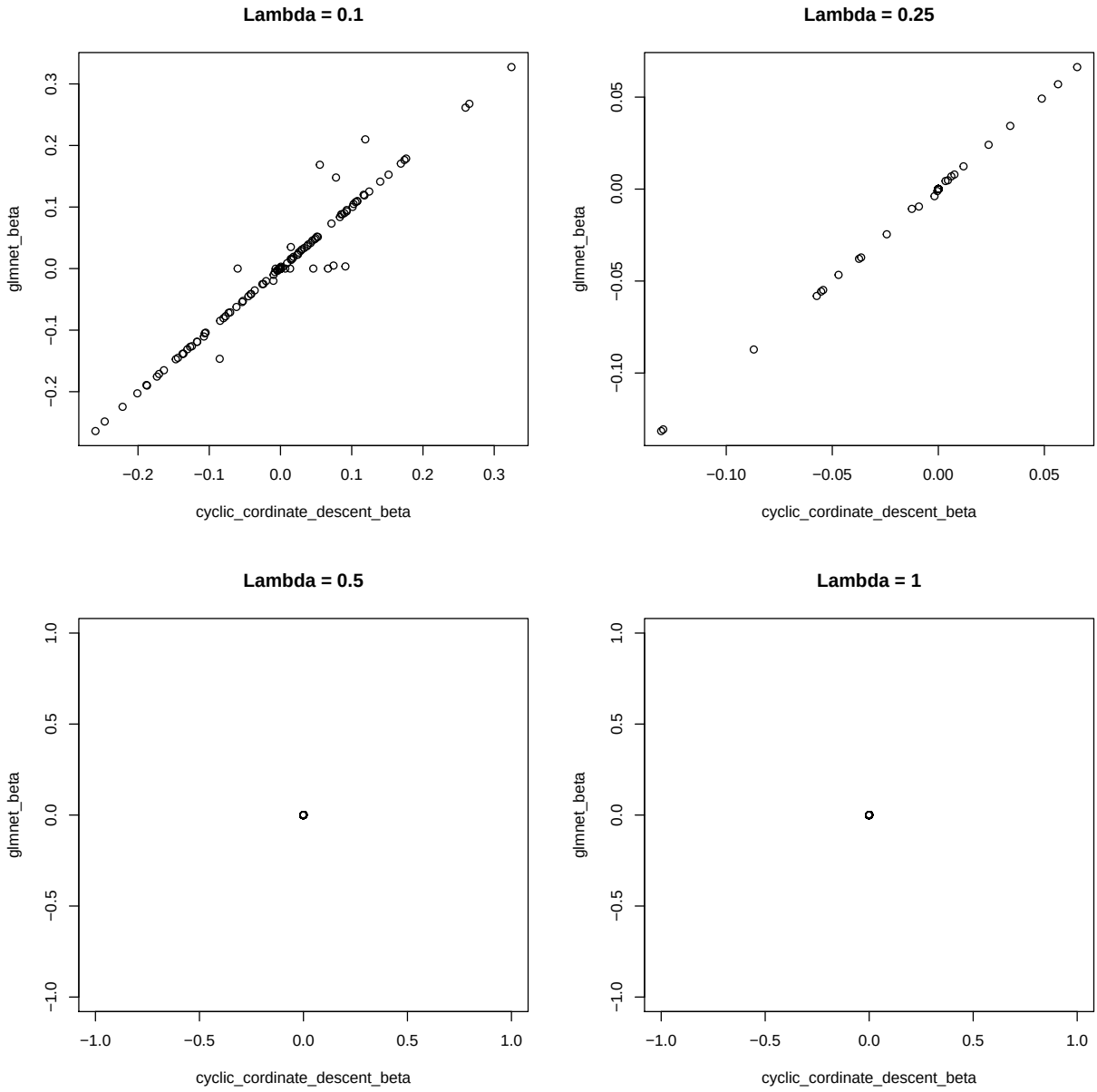
```
## For Lambda = 0.25  , the MSE between two beta estimates is: 6.99144718027069e-09
```

```
compare_beta(bata_1, lasso_coef1$s1[-1], lamda_value = "Lambda = 0.5")
```

```
## For Lambda = 0.5  , the MSE between two beta estimates is: 0
```

```
compare_beta(bata_0, lasso_coef0$s1[-1], lamda_value = "Lambda = 1")
```

```
## For Lambda = 1  , the MSE between two beta estimates is: 0
```




```
par(mfrow=c(1,1))
```

- *Thus, the implementation of cyclic coordinate descent algorithm for Lasso regression was successful*
- Since, these λ values are arbitrarily picked, and are not necessarily the optimal value, I will not apply the test set to it.
- Instead I will use *glmnet* implementaiton of Lasso regression, and use cross-validation to find the optimal value of λ and show its performance on test data set.

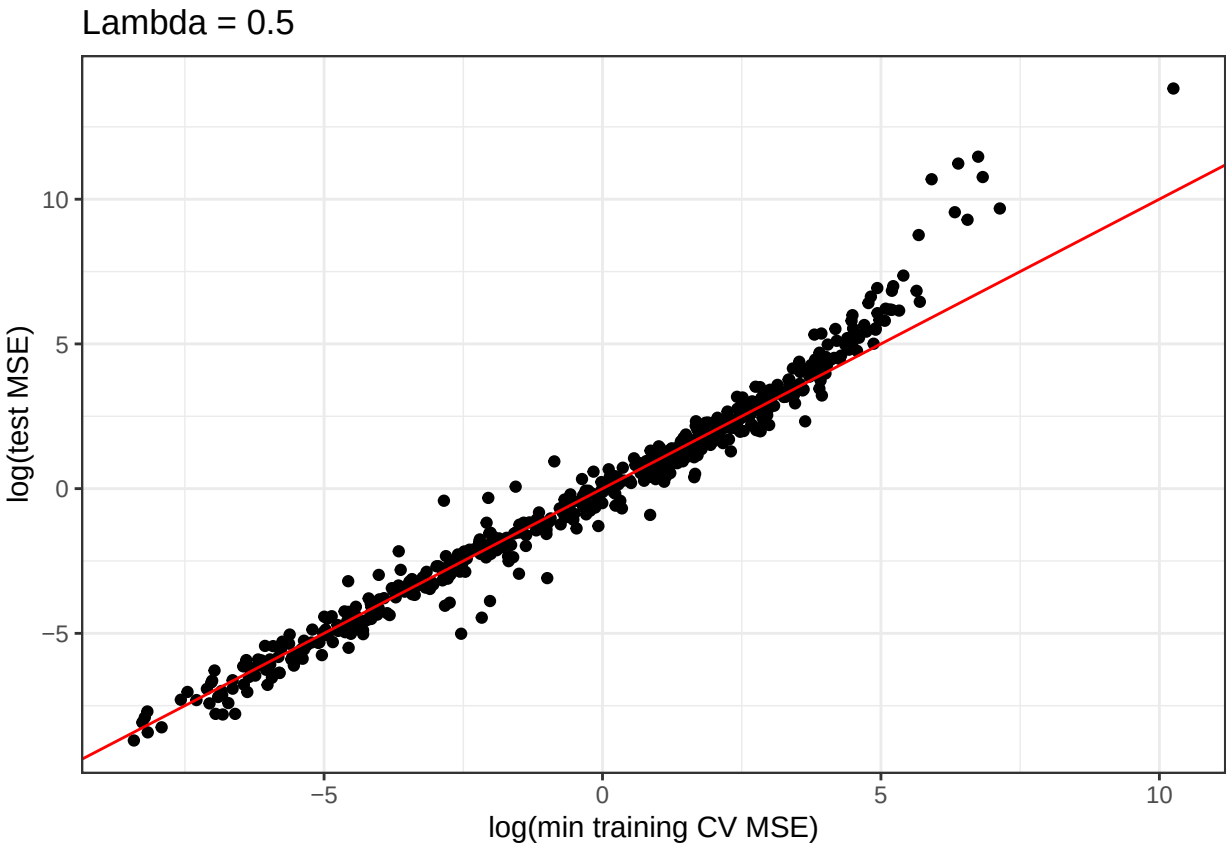
Part 4, using *glmnet* to train and test models for every gene

- I will now apply *glmnet* to every gene
- I will show validation results for fixed $\lambda = 0.5$ and the optimal λ chosen by smallest MSE on 10-fold cross validation

```r
#############################################################################
## using glmnet to get predictors for each gene

epxresion_regression <- foreach(i = 1:nrow(expression_df), .combine = "c") %do% {

  snps_in_gene_ranges <- genotype_df %>% filter(CHR == expression_df$chromosome[i] &
                                                  POS >= (expression_df$start[i] - 500000) &
                                                  POS <= (expression_df$end[i] + 500000))

  genotype_mat <- snps_in_gene_ranges[,-c(1:6)] %>% data.matrix() %>% t()
  expression_mat <- expression_df[i,-c(1:4)] %>% data.matrix() %>% t()

  ## Centering Y to have mean 0
  expression_mat_scaled <- scale(expression_mat[,1], center = TRUE, scale = FALSE)
  ## Centering each column (feature) to 0 and having SD = 1
  genotype_mat_scaled <- scale(genotype_mat, center = TRUE, scale = TRUE)

  genotype_mat_train <- genotype_mat_scaled[in_train,]
  genotype_mat_test <- genotype_mat_scaled[in_test,]

  expression_mat_train <- expression_mat_scaled[in_train,]
  expression_mat_test <- expression_mat_scaled[in_test,]

  set.seed(480)
  lasso_fit <- cv.glmnet(x = genotype_mat_train, y = expression_mat_train, alpha = 1,
                         type.measure = "mse", nfolds = 10, parallel = TRUE)

  ## Minimum train MSE
  mse.min.train <- lasso_fit$cvm[lasso_fit$lambda == lasso_fit$lambda.min]

  test_pred_exprs <- cbind(predict(lasso_fit, genotype_mat_test, s = "lambda.min"), expression_mat_test)
  colnames(test_pred_exprs) <- c("y_hat", "y")
  test_pred_exprs <- as.data.frame(test_pred_exprs)
  mse.min.test <- mse(test_pred_exprs$y, test_pred_exprs$y_hat)

  out_opt <- list(mse.min.train = mse.min.train, mse.test = mse.min.test, test_pred_exprs = test_pred_ex

  ## Second smallest MSE
  mse.min.train <- mse(predict(lasso_fit, genotype_mat_train, s = 0.5), expression_mat_train)
  test_pred_exprs <- cbind(predict(lasso_fit, genotype_mat_test, s = 0.5), expression_mat_test)
  colnames(test_pred_exprs) <- c("y_hat", "y")
  test_pred_exprs <- as.data.frame(test_pred_exprs)
  mse.min.test <- mse(test_pred_exprs$y, test_pred_exprs$y_hat)

  out_manual <- list(mse.min.train = mse.min.train, mse.test = mse.min.test, test_pred_exprs = test_pred

  return(list(out_opt = out_opt, out_manual = out_manual))
}
```

- x-axis show MSE for training dataset and y-axis shows MSE for test dataset for $\lambda = 0.5$
- More values lie above the x=y line, which means the model overfits the training set as it has smaller MSE

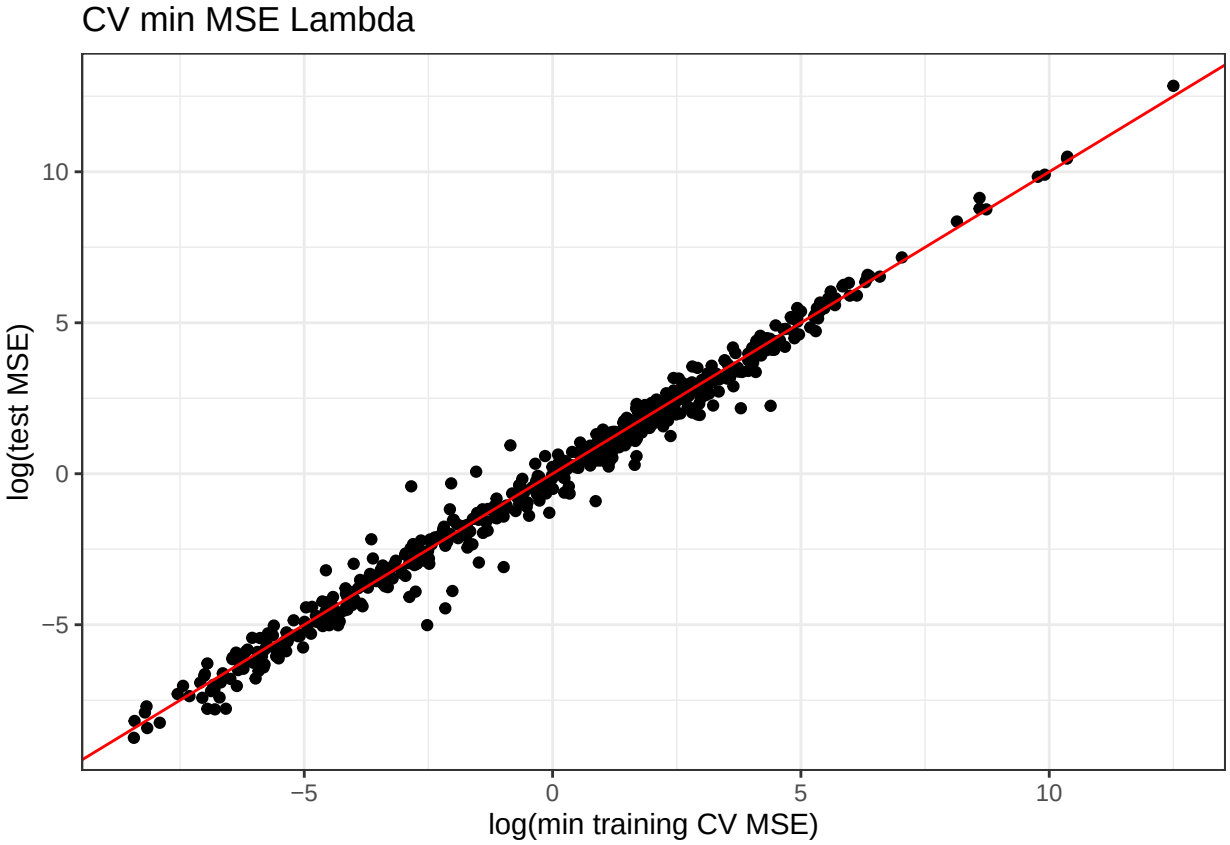
```
index <- 1:length(epxresion_regression) %% 2
odd_index <- seq(1:length(epxresion_regression))[index==1]
even_index <- seq(1:length(epxresion_regression))[index==0]

cv.train.test.mse <- foreach(i = even_index, .combine = "rbind") %do% {
  data.frame(mse.min.train = epxresion_regression[[i]]$mse.min.train, mse.test = epxresion_regression[[
}

cv.train.test.mse$gene_id <- expression_df$gene_id

ggplot(cv.train.test.mse) + aes(x = log(mse.min.train), y = log(mse.test)) + geom_point() + theme_bw() +
  xlab("log(min training CV MSE)") + ylab("log(test MSE)") + geom_abline(slope=1, intercept=0, col = "r
  ggtitle("Lambda = 0.5")
```



- x-axis show minimum MSE from 10-fold CV and y-axis shows test MSE
- Most of the values lie on x=y line

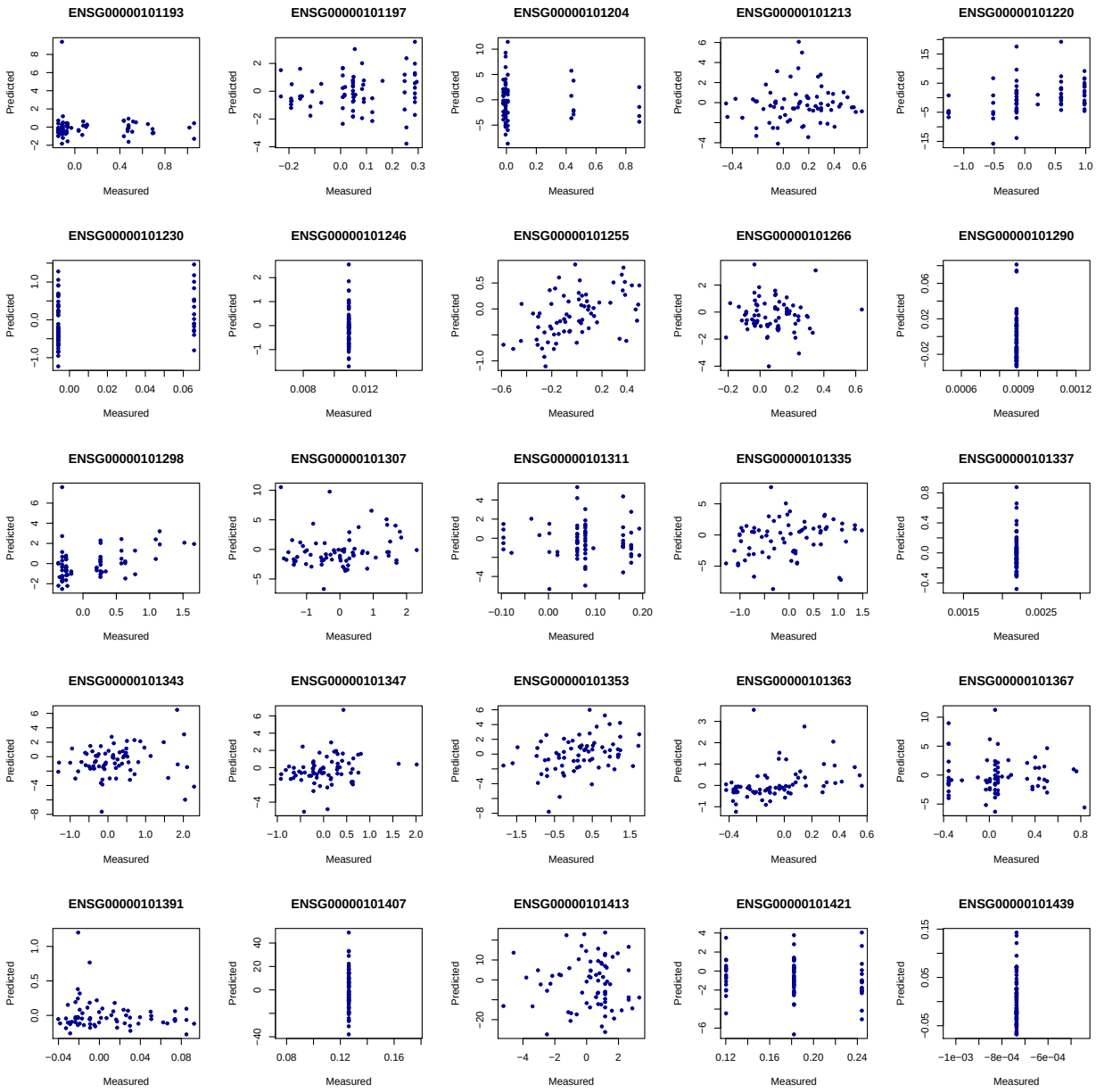
```
cv.train.test.mse <- foreach(i = odd_index, .combine = "rbind") %do% {
  data.frame(mse.min.train = epxresion_regression[[i]]$mse.min.train, mse.test = epxresion_regression[[
}

cv.train.test.mse$gene_id <- expression_df$gene_id

ggplot(cv.train.test.mse) + aes(x = log(mse.min.train), y = log(mse.test)) + geom_point() + theme_bw() +
```

```
  xlab("log(min training CV MSE)") + ylab("log(test MSE)") + geom_abline(slope=1, intercept=0, col = "r
  ggtitle("CV min MSE Lambda")
```



- I will next show predicted vs. measured expression value for random 25 genes
- Some models the predictions are constant, which means the β values converged to 0

```
par(mfrow=c(5,5))
for(i in odd_index[51:75]){
  plot(epxresion_regression[[i]]$test_pred_exprs, main = expression_df$gene_id[i],
       xlab = "Measured", ylab = "Predicted", col = "darkblue", pch = 20)
}
```




```
par(mfrow=c(1,1))
```

- About 46.8 (n=545) of gene prediction models had constant predictions, which means the β values converged to 0

```
number.null.models <- foreach(i = odd_index, .combine = "c") %do% {
  length(unique(epxresion_regression[[i]]$test_pred_exprs$y_hat)) == 1
}

sum(number.null.models)/length(number.null.models)
```

```
## [1] 0.4678899
```