448134_raport

Jakub Giezgała

11 maja 2024

Data exploration

```
# Load the data as number matrices
X_train <- as.matrix(read.delim("X_train.csv", sep = ',', dec = '.'))
X_test <- as.matrix(read.delim("X_test.csv", sep = ',', dec = '.'))
y_train <- read.delim("y_train.csv", sep = ',', dec = '.')
y <- y_train$CD36
```

```
# Check data type
is_integer_column <- function(column) {
  all(column == as.integer(column))
}
unique(unlist(lapply(X_train, class)))
integer_columns_X_train <- apply(X_train, 2, is_integer_column)
which(integer_columns_X_train)
unique(unlist(lapply(X_test, class)))
integer_columns_X_test <- apply(X_test, 2, is_integer_column)
which(integer_columns_X_test)
str(y_train)
```

```
# Check missing values in data
sum(is.na(X_train), is.null(X_train))
sum(is.na(X_test), is.null(X_test))
sum(is.na(y_train), is.null(y_train))
```

```
# Check data dimension
dim(X_train)
```

```
## [1] 6800 9000
```

```
dim(X_test)
```

```
## [1] 1200 9000
```

```
dim(y_train)
```

```
## [1] 6800    1
```

The training data contains 6800 observations and 9000 variables. So these are data from 6800 cells, containing the expression of 9000 genes. In comparison, the test data contain 1200 observations and exactly the same number of variables. This is a typical situation for RNA-Seq analysis with more variables than observations, where great care must be taken to avoid over-fitting. The data are complete (no NaN or NULL values) and all columns are numeric, so no conversion is needed. There is also no repeating variable with only natural number values in both data sets, so it can be assumed that each variable is quantitative.

The empirical distribution of the explained variable

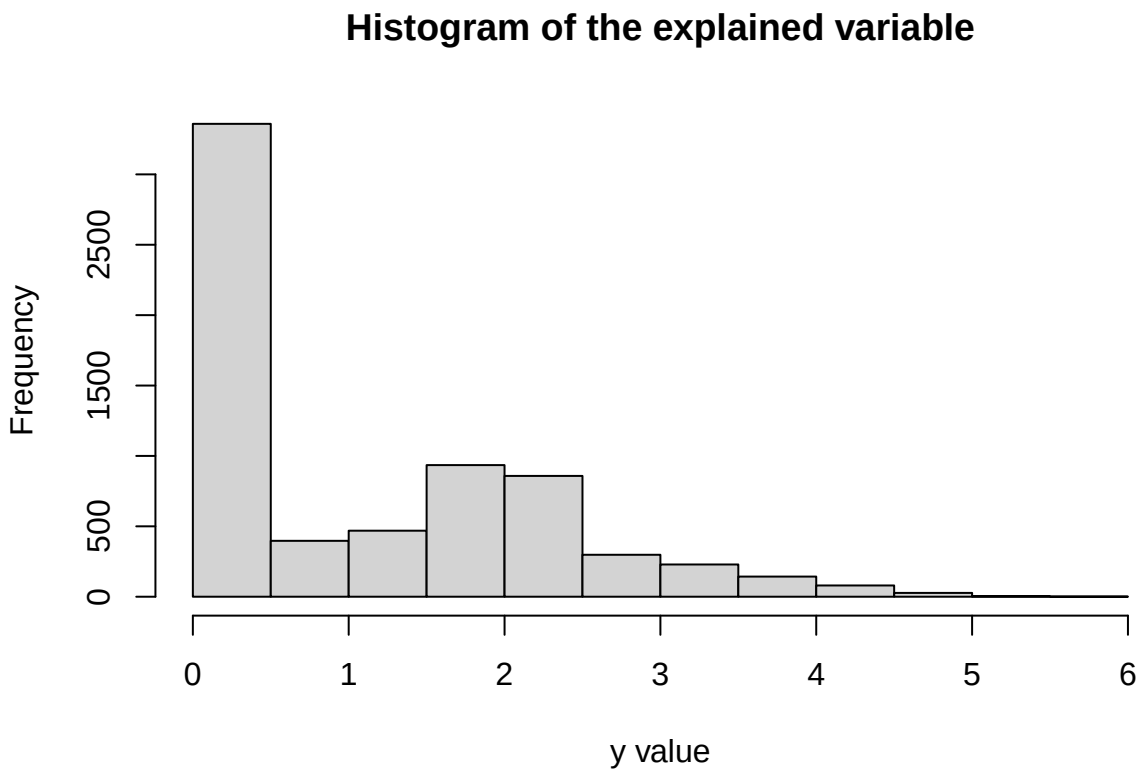
Basic statistic of the explained variable

```
summary(y_train)
```

```
##       CD36
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.526
##  Mean   :1.086
##  3rd Qu.:1.972
##  Max.   :5.768
```

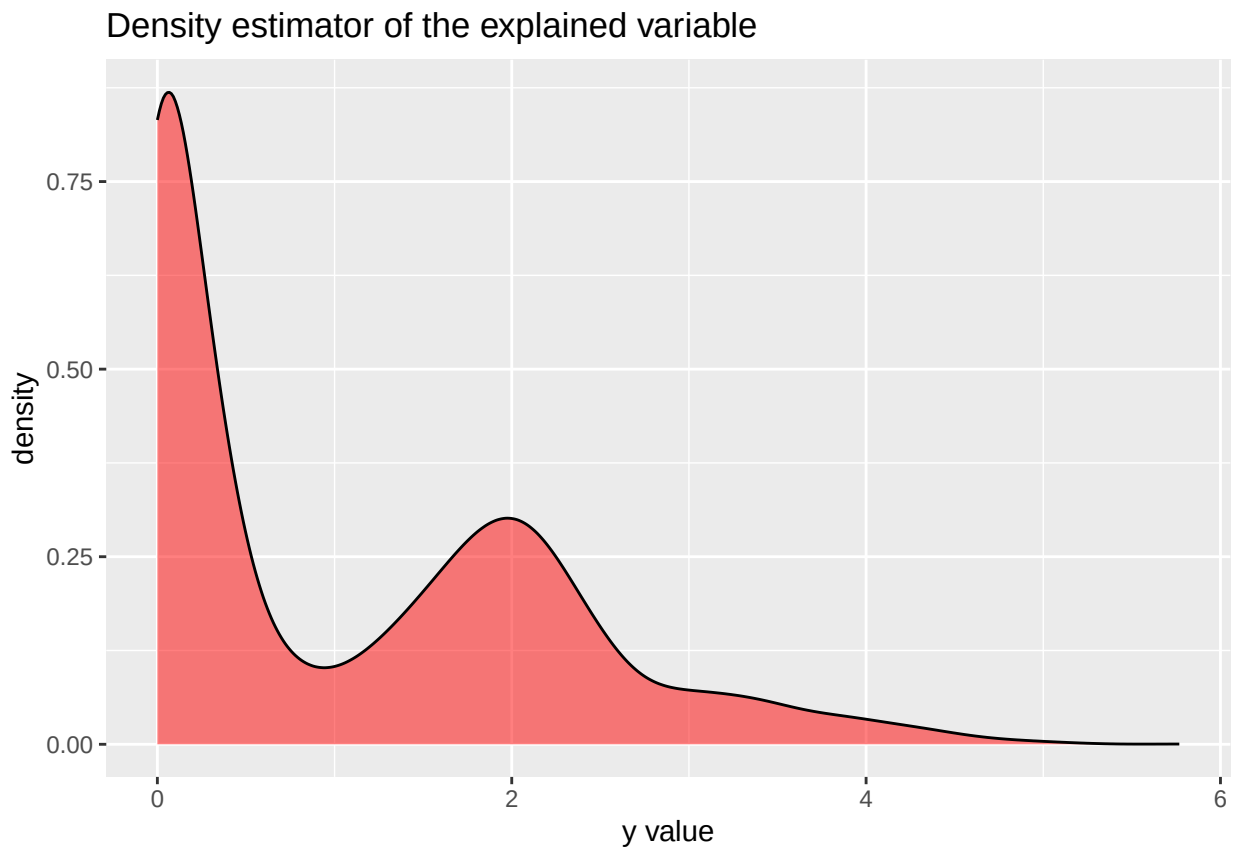
Histogram of the explained variable

```
hist(y_train$CD36, breaks = "Sturges", main = "Histogram of the explained variable",
     xlab = "y value")
```



Density estimator graph of the explained variable

```
ggplot(data = data.frame(y_train$CD36), aes(x = y_train$CD36)) +
  geom_density(fill = "red", alpha = 0.5) +
  labs(title = "Density estimator of the explained variable",
       x = "y value", y = "density")
```

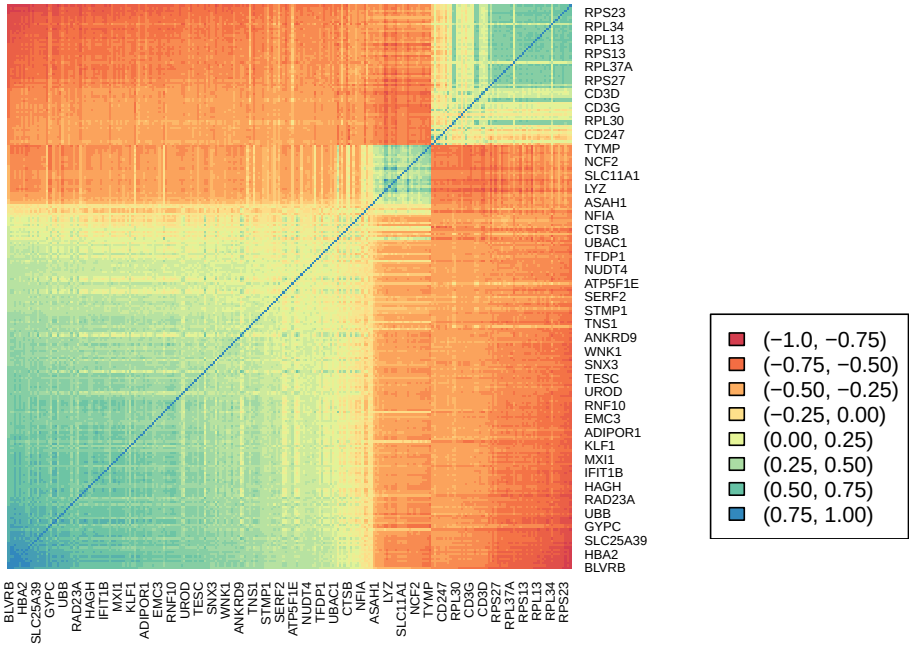


The distribution of the explanatory variable is bimodal, with most of the values concentrated around zero and a second major part concentrated around two.

Heatmap

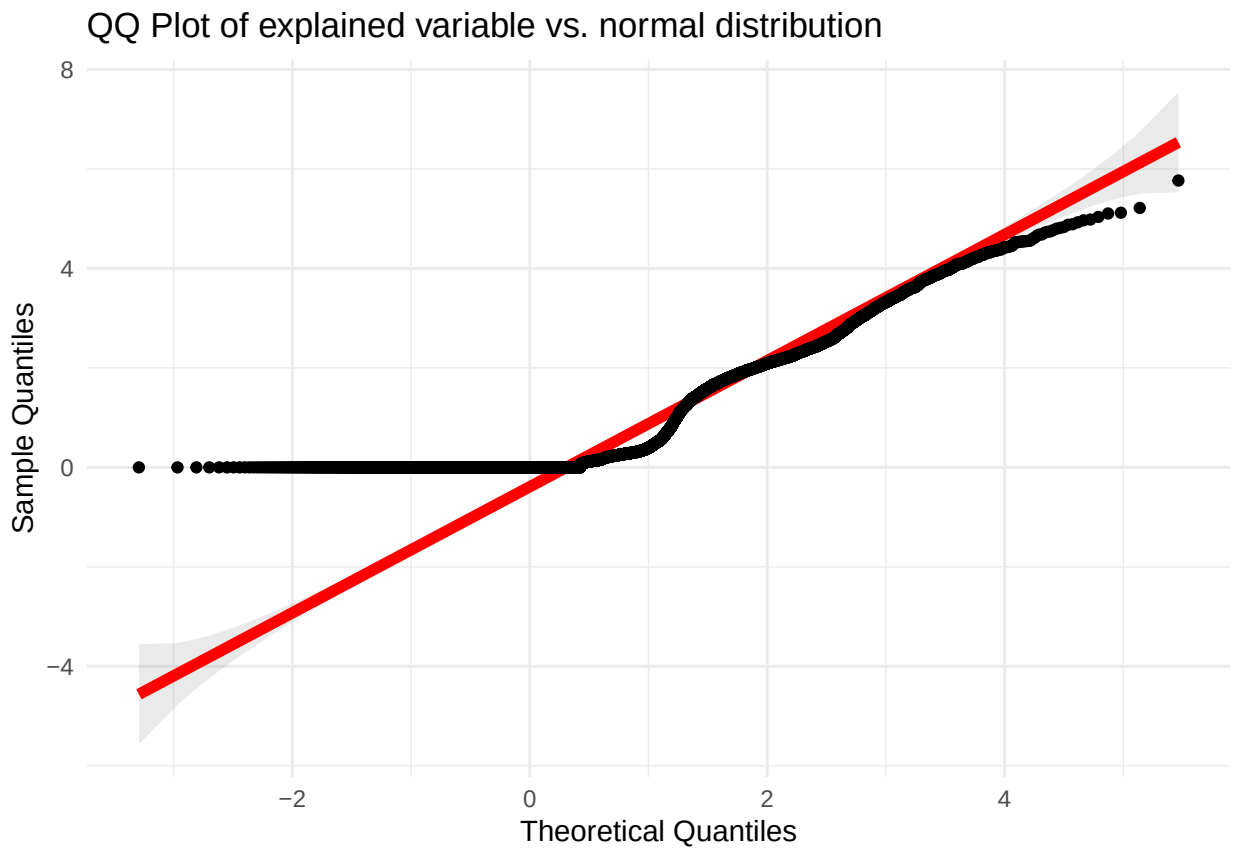
Correlation heatmap of the 250 explanatory variables most correlated with the explained variability

```
# Create a corerlation matrix for 250 features
corelation_matrix <- cor(X_train, y_train$CD36)
top_values <- order(abs(corelation_matrix[, 1]), decreasing = TRUE)[1:250]
selected_values <- X_train[, top_values]
selected_order <- cor(selected_values, selected_values[, 1])
selected_order <- selected_order[order(selected_order, decreasing = TRUE),]
selected_order <- names(selected_order)
corelation_selected <- cor(X_train[, selected_order])

# Create a heatmap
heatmap(corelation_selected, Rowv = NA, Colv = NA, scale = "none",
        col=colorRampPalette(brewer.pal(8, "Spectral"))(20))
legend(x = "bottomright", legend=c("(-1.0, -0.75)", "(-0.75, -0.50)",
                                   "(-0.50, -0.25)", "(-0.25, 0.00)",
                                   "(0.00, 0.25)", "(0.25, 0.50)",
                                   "(0.50, 0.75)", "(0.75, 1.00)"),
      fill=colorRampPalette(brewer.pal(8, "Spectral"))(8), cex = 0.8)
```



Quantile diagram

```
ggplot(y_train, aes(sample = y)) +
  stat_qq_band(distribution = "norm", alpha = 0.2) +
  stat_qq_line(distribution = "norm", col = "red", size = 2) +
  stat_qq_point(distribution = "norm") +
  labs(title = "QQ Plot of explained variable vs. normal distribution",
       x="Theoretical Quantiles",
       y="Sample Quantiles") +
  theme_minimal()
```



The mean and variance of the experimental distribution cannot be read directly from the quantile plot above. The Q-Q graph compares the quantiles of two distributions, where on the X axis are the quantiles of one distribution and on the Y axis are the quantiles of the other distribution. Deviations from a straight line can indicate different types of deviation. In this case, we can conclude that the distribution of the explained variable is rather different from the normal distribution (fat tails).

Statistical test for the explained variable

H_0: y has a normal distribution;

H_1: y has a different distribution;

$\alpha = 0.05$

```
# shapiro.test(y)
# ks.test(y, "pnorm", mean(y), sd(y))
ad.test(y)
```

```
##
##  Anderson-Darling normality test
##
## data:  y
## A = 358.85, p-value < 2.2e-16
```

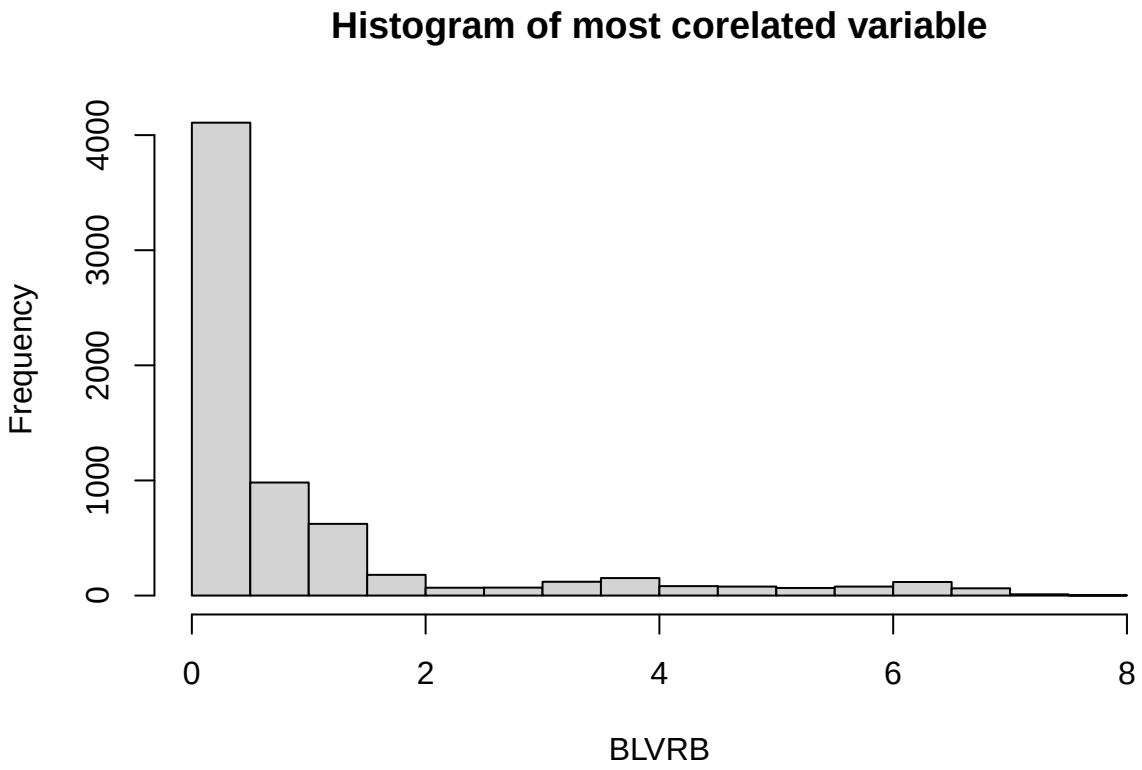
We cannot use the Shapiro-Wilk test due to the excessive amount of data. Meanwhile, we cannot use the Kolmogorov-Smirnov test due to repeated values. The Anderson-Darling test results in a p-value $< \alpha$, so we reject our null hypothesis. Thus, we recognise that the variable y does not have a normal distribution.

Test for variable most closely correlated with the explained variable

```
most_corelated_variable <-
  names(corelation_matrix[, 1])[which.max(abs(corelation_matrix[, 1]))]
most_corelated <- X_train[, most_corelated_variable]
```

BLVRB turns out to be the most correlated variable with CD36 expression. Let's start by painting a histogram.

```
data <- data.frame(CD36 = y, BLVRB = most_corelated)
hist(data$BLVRB, breaks="Sturges", main="Histogram of most corelated variable",
     xlab="BLVRB")
```



From the histogram, we can surmise that our most correlated variable may have an exponential distribution with a lambda parameter of 4000.

H_0: BLVRB has an exponential distribution with $\lambda = 4000$;

H_1: BLVRB has a different distribution;

$\alpha = 0.05$

```
lambda <- 4000
ks.test(data$BLVRB, "pexp", rate = lambda)
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  data$BLVRB
## D = 0.57971, p-value < 2.2e-16
```

```
## alternative hypothesis: two-sided
```

The Kolmogorov-Smirnov test, due to repeated values, may have returned an incorrect result. Let us try to fit the BLVRB to any exponential distribution using the Monte-Carlo method.

```
shapiro.exp.test<-function(x, nrepl=2000) {
  DNAME <- deparse(substitute(x))
  l<-0
  n<-length(x)
  x<-sort(x)
  y<-mean(x)
  w<-n*(y-x[1])^2/((n-1)*sum((x-y)^2))
  for(i in 1:nrepl)
  {
    s<-rexp(n)
    s<-sort(s)
    y<-mean(s)
    W<-n*(y-s[1])^2/((n-1)*sum((s-y)^2))
    if (W<w) l=l+1
  }
  p.value<-l/nrepl
  RVAL<-list(statistic=c(W=w), p.value=p.value,
             method="Shapiro-Wilk test for exponentiality",
             data.name = DNAME)
  class(RVAL)<-"htest"
  return(RVAL)
}
shapiro.exp.test(data$BLVRB)
```

```
##
##  Shapiro-Wilk test for exponentiality
##
## data:  data$BLVRB
## W = 4.513e-05, p-value < 2.2e-16
```

In both cases p-value $< \alpha$, so we can consider that the explanatory variable in question does not have an exponential distribution. Now let us check whether BLVRB has a similar distribution in the test and training set. First we will check if it has a normal distribution, if so we will use the Student's t-test, otherwise we will use the Wilcoxon test.

```
# Similarity of the most correlated variable
most_corelated_train <- X_train[, "BLVRB"]
most_corelated_test <- X_test[, "BLVRB"]
ad.test(most_corelated_train)
```

```
##
##  Anderson-Darling normality test
##
## data:  most_corelated_train
## A = 1020.3, p-value < 2.2e-16
```

```
ad.test(most_corelated_test)
```

```
##
##  Anderson-Darling normality test
##
## data:  most_corelated_test
```

```
## A = 192.05, p-value < 2.2e-16
```

According to the above results, we have to use the Wilcoxon test because the assumptions of the t-student test are not met.

H_0: BLVRB has a similar distribution in the training and test sets;

H_1: BLVRB has a different distribution;

$\alpha = 0.05$;

```
wilcox.test(most_corelated_train, most_corelated_test)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  most_corelated_train and most_corelated_test
## W = 4149242, p-value = 0.2945
## alternative hypothesis: true location shift is not equal to 0
```

We receive p-value $> \alpha$, thus accepting the null hypothesis. BLVRB has a similar distribution in the training and test sets.

ElasticNet

Method description

The ElasticNet model is a combination of ridge regression and lasso regression. It is used to solve regression problems, especially in cases where the data set contains many explanatory variables that may be correlated with each other. ElasticNet estimates the regression coefficients for each explanatory variable by minimising the cost function of the following form

$$L(\beta) = \text{RSS}(\beta) + \lambda \sum_{j=1}^{p} (\alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2)$$

The cost function is a combination of two components:

1) Linear L1 norm - as in Lasso regression - which promotes sparsity in the resulting coefficients, meaning that many of them can be exactly zero, leading to variable selection;
2) Quadratic L2 norm - as in Ridge regression - which reduces the values of the coefficients but does not cause them to be exactly zero.

For α value equal to 0, we are dealing with pure Ridge Regression, and for 1 â€“ with pure Lasso Regression.

Hyperparameters of the method:

1) α - the mixture coefficient between lasso regression and ridge regression. α controls the balance between variable selection (lasso regression) and coefficient reduction (ridge regression). A value of 0 means pure Ridge Regression, a value of 1 means pure Lasso Regression.

2) λ - the regularisation parameter, which controls how strongly the parameters are fixed at 0. The larger the value of λ, the stronger the regularisation and the smaller the value of the regression coefficients.

Grid of hyperparameters

```
alphas <- seq(0, 1, by=0.1)
lambdas <- seq(1, 0, by=-0.1)
```

Let's perform k-fold cross-validation to select the parameter combination that gives the smallest validation error. Let's take $k = 10$, one of the two most commonly used values, which, despite more calculations, will provide better accuracy with so much data than $k = 5$.

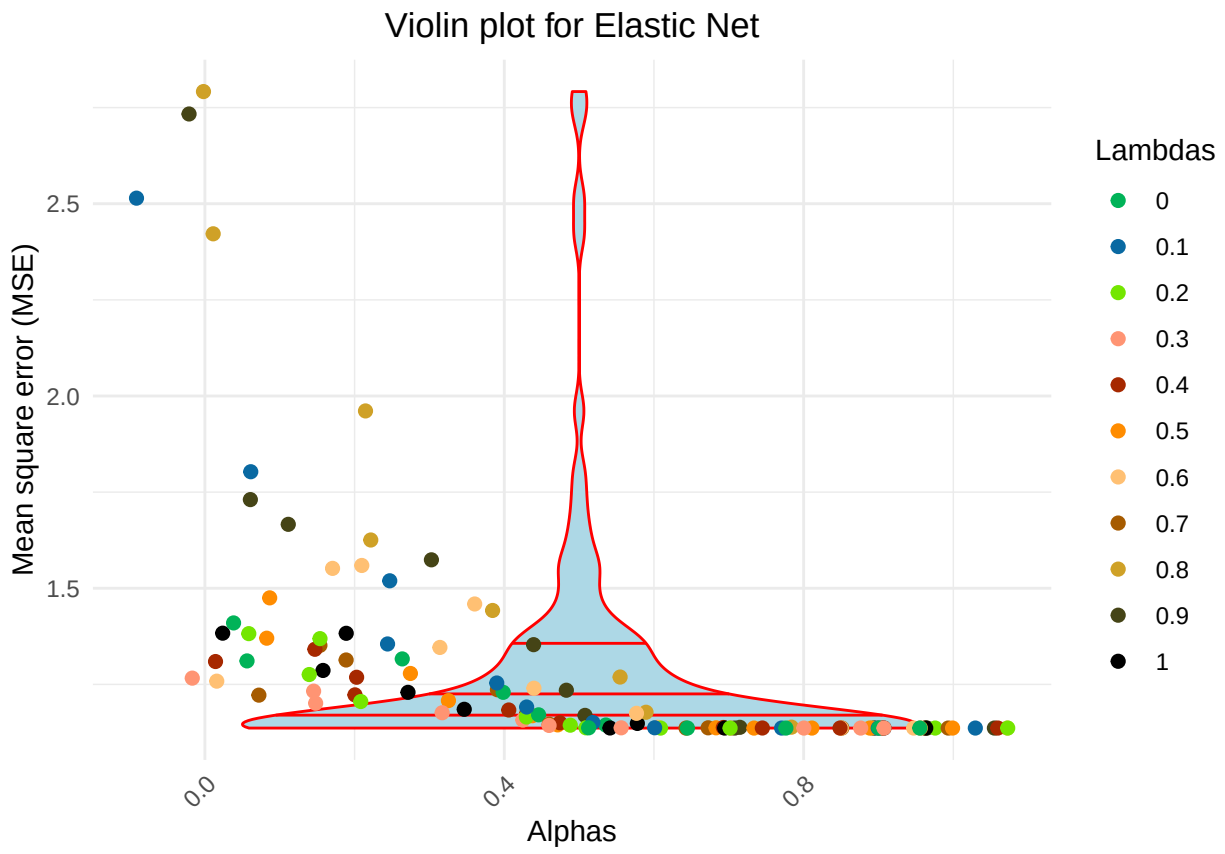
```
k <- 10
folds <- createFolds(y, k)
```

```
# Start parallel
parallelStartSocket(cpus=detectCores())

# Matrices of training and validation errors
en_train_errors <- matrix(NA, nrow = length(alphas), ncol = length(lambdas))
en_validation_errors <- matrix(NA, nrow = length(alphas), ncol = length(lambdas))

# 10-fold cross-validation for each parameter combination
for (i in 1:length(alphas)) {
  train_errors <- c(1:length(lambdas))
  validation_errors <- c(1:length(lambdas))

  for (fold in 1:k) {
    train_index <- unlist(folds[-fold])
    validation_index <- folds[[fold]]

    x_train_fold <- X_train[train_index, ]
    x_validation_fold <- X_train[validation_index, ]
    y_train_fold <- y[train_index]
    y_validation_fold <- y[validation_index]

    en_model <- glmnet(x_train_fold, y_train_fold, alpha = alphas[i], lambda = lambdas)
    pred <- predict(en_model, X_train, s = lambdas)

    # predictions
    train_pred <- pred[train_index, ]
    validation_pred <- pred[validation_index, ]

    # RMSE
    for (j in 1:length(lambdas)) {
      train_errors[j] <- sqrt(mean((train_pred[j] - y_train_fold)^2))
      validation_errors[j] <- sqrt(mean((validation_pred[j] - y_validation_fold)^2))
    }
  }
  en_train_errors[i, ] <- train_errors
  en_validation_errors[i, ] <- validation_errors
}

# Naming columns of errors
rownames(en_train_errors) <- alphas
colnames(en_train_errors) <- lambdas
rownames(en_validation_errors) <- alphas
colnames(en_validation_errors) <- lambdas

# End for parallel
parallelStop()
```

Violin plot

```r
# Create data frame for violin plot
en_validation_errors_violin <- as.data.frame(en_validation_errors)
rownames(en_validation_errors_violin) <- alphas
colnames(en_validation_errors_violin) <- lambdas

en_validation_errors_violin <- en_validation_errors_violin %>%
  rownames_to_column(var = "alphas") %>%
  pivot_longer(-alphas, names_to = "lambdas", values_to = "mse") %>%
  mutate(alphas = as.numeric(alphas), lambdas = as.factor(lambdas))
```

```r
ggplot(en_validation_errors_violin, aes(x = alphas, y = mse, color = lambdas)) +
  geom_violin(fill = "lightblue", color = "red", draw_quantiles = c(0.25, 0.5, 0.75)) +
  geom_jitter(size = 2, width = 0.1) +  # MSE
  scale_color_manual(values=c("#00B358", "#0969A2", "#74E600", "#FF9473", "#A62800",
                              "#FF8C00", "#FFC073", "#A65B00", "#CFA127", "#454416",
                              "#000000")) +
labs(
  title = "Violin plot for Elastic Net",
  x = "Alphas",
  y = "Mean square error (MSE)",
  color = "Lambdas"
) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
```



ElasticNet model

```
# Select alpha and lambda with the smallest validation error
index <- which(en_validation_errors == min(en_validation_errors), arr.ind = TRUE)

alphas[index[1]]
```

```
## [1] 0.7
```

```
lambdas[index[2]]
```

```
## [1] 0.3
```

```
# Mean validation error for selected parameters
en_train_error <- en_train_errors[index]
en_train_error <- en_train_error[1]
en_validation_error <- en_validation_errors[index]
en_validation_error <- en_validation_error[1]

en_train_error
```

```
## [1] 1.155254
```

```
en_validation_error
```

```
## [1] 1.13638
```

Random forest

Hyperparameters grid

1) `min.node.size` – the minimum number of observations in a node.
2) `max.depth` – the maximum depth of the tree
3) `mtry` – the number of predictors randomly sampled to construct a tree (for regression usually $m = \frac{p}{3}$)

```
rf_hyperparams_grid <- expand.grid(
  mtry = floor(ncol(X_train) * c(.01, .1, .2, .333)),
  min.node.size = c(5, 10, 15, 20),
  max.depth = c(10, 25, 50, NULL),
  train_rmse = NA,
  validation_rmse = NA
)
```

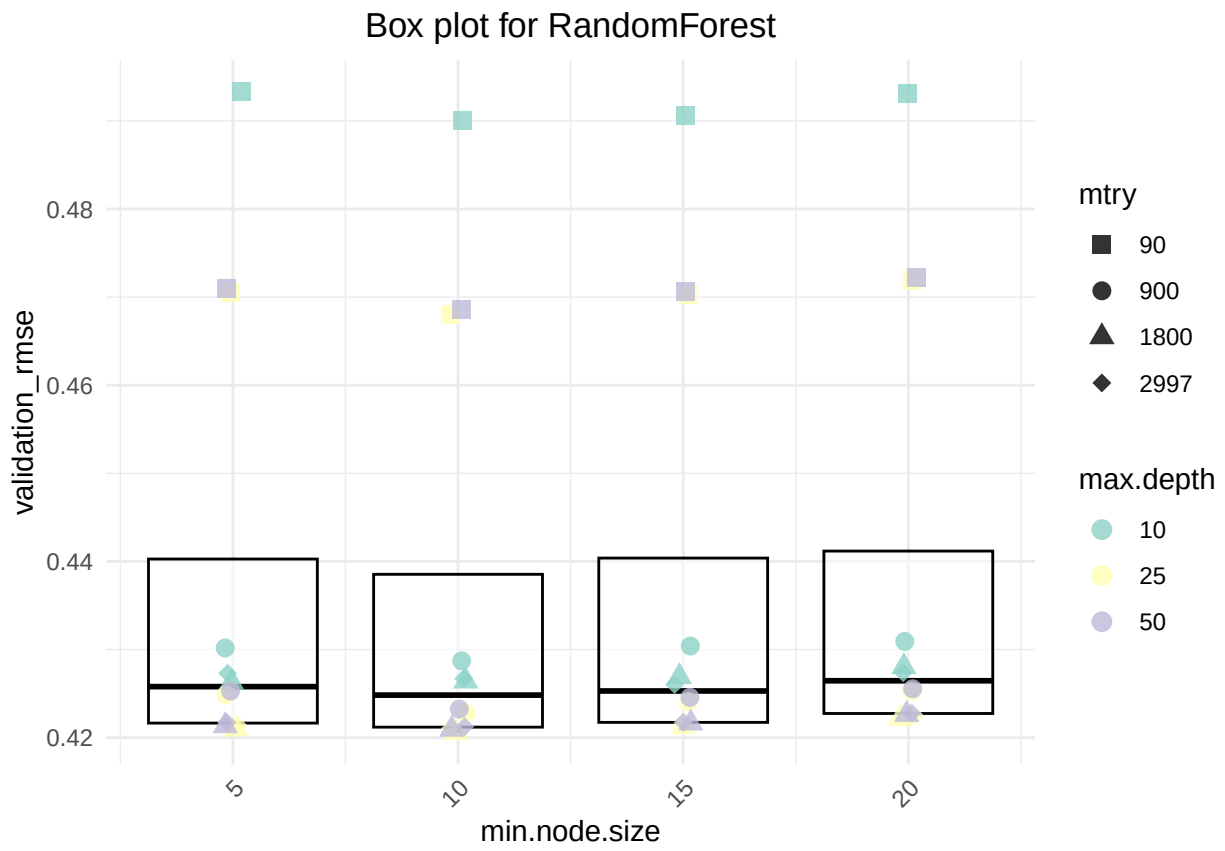
```
# Start parallel
parallelStartSocket(cpus=detectCores())

# For each combinations of parameters
for(i in seq_len(nrow(rf_hyperparams_grid))) {
  train_errors <- c(1:k)
  validation_errors <- c(1:k)

  for (fold in 1:k) {
    train_index <- unlist(folds[-fold])
    validation_index <- folds[[fold]]

    x_train_fold <- X_train[train_index, ]
    y_train_fold <- y[train_index]
    x_validation_fold <- X_train[validation_index, ]
    y_validation_fold <- y[validation_index]

    model <- ranger(
      x = x_train_fold,
      y = y_train_fold,
      seed = 20,
      verbose = FALSE,
      num.trees = 50,
      mtry = rf_hyperparams_grid$mtry[i],
      min.node.size = rf_hyperparams_grid$min.node.size[i],
      max.depth = rf_hyperparams_grid$max.depth[i]
    )
    # Predictions for the validation set
    validation_pred <- predict(model, x_validation_fold)$predictions

    # RMSE
    train_errors[fold] <- sqrt(model$prediction.error)
    validation_errors[fold] <- sqrt(mean((validation_pred - y_validation_fold)^2))
  }

  # Mean RMSE values
  rf_hyperparams_grid$train_rmse[i] <- mean(train_errors)
  rf_hyperparams_grid$validation_rmse[i] <- mean(validation_errors)
}
```

```r
# End for parallel
parallelStop()
```

Box plot

```r
# Preparation date to box plot
rf_hyperparams_grid_boxplot <- rf_hyperparams_grid
rf_hyperparams_grid_boxplot$max.depth <- as.factor(rf_hyperparams_grid_boxplot$max.depth)
rf_hyperparams_grid_boxplot$mtry <- as.factor(rf_hyperparams_grid_boxplot$mtry)
```

```r
ggplot(rf_hyperparams_grid_boxplot, aes(x = min.node.size, y = validation_rmse)) +
  geom_boxplot(aes(group = min.node.size), alpha = 0.5, outlier.shape = NA,
               linetype = "solid", varwidth = TRUE, color = "black") +
  geom_jitter(aes(color = factor(max.depth), shape = factor(mtry)),
              position = position_jitter(width = 0.2), size = 3, alpha = 0.8) +
  theme_minimal() +
  labs(
    title = "Box plot for RandomForest",
    x = "min.node.size",
    y = "validation_rmse"
  ) +
  scale_color_brewer(palette = "Set3", name = "max.depth") +
  scale_shape_manual(values = c(15, 16, 17, 18), name = "mtry") +
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
```



Random forest model

```
# Mean validation error for selected parameters
index <- which.min(rf_hyperparams_grid$validation_rmse)

rf_hyperparams_grid[index, ]
```

```
##    mtry min.node.size max.depth train_rmse validation_rmse
## 23 1800            10        25  0.4318332       0.4205768
```

```
rf_best_params <- rf_hyperparams_grid[index, 1:3]

rf_train_error <- rf_hyperparams_grid$train_rmse[index]
rf_train_error <- rf_train_error[1]
rf_validation_error <- rf_hyperparams_grid$validation_rmse[index]
rf_validation_error <- rf_validation_error[1]

rf_train_error
```

```
## [1] 0.4318332
```

```
rf_validation_error
```

```
## [1] 0.4205768
```

Summary

```
train_errors <- c(1:k)
validation_errors <- c(1:k)
```

Reference model

```
reference_y <- mean(y)

for (fold in 1:k) {
  train_index <- unlist(folds[-fold])
  validation_index <- folds[[fold]]

  x_train_fold <- X_train[train_index, ]
  y_train_fold <- y[train_index]
  x_validation_fold <- X_train[validation_index, ]
  y_validation_fold <- y[validation_index]

  train_errors[fold] <- sqrt(mean((reference_y - y_train_fold)^2))
  validation_errors[fold] <- sqrt(mean((reference_y - y_validation_fold)^2))
}

index <- which.min(validation_errors)
ref_train_error <- train_errors[index]
ref_train_error <- ref_train_error[1]
ref_validation_error <- validation_errors[index]
ref_validation_error <- ref_validation_error[1]
```

Models comparison

```
comparison <- matrix(c(ref_train_error, ref_validation_error, en_train_error,
                       en_validation_error, rf_train_error, rf_validation_error),
                     nrow=2)
colnames(comparison) <- c("Reference", "ElasticNet", "RandomForest")
rownames(comparison) <- c("training", "validation")
comparison
```

```
##            Reference ElasticNet RandomForest
## training    1.157347   1.155254    0.4318332
## validation  1.116747   1.136380    0.4205768
```

Comparing the superior results of all three different models, it is easy to see that the model based on random forest is the best. It was able to achieve the best results without much training, which, as mentioned earlier, was the major problem with this particular data. Such results are not unexpected because random forests, as the name suggests, use a random subset of predictors and observations to reduce the variance of the model. This makes them less prone to overfitting. They are also able to model non-linear relationships between explanatory variables and the explanatory variable itself. Consequently, of the models presented above, random forests are much more appropriate for prediction.