Prediction of the Electrical Power output of a Combined Cycle Power Plant

Capstone (CYO) Project of

Ezio Tarquilio

2022-03-09

Overview

I chose the *Combined Cycle Power Plant Data Set*[1] from the *UCI Machine Learning Repository*.

The dataset, provided as a .xlsx file in the *data* folder, contains 9568 observations collected from a combined-cycle Power Plant over six years (2006-2011) when the Power Plant worked with a full load.

Features consist of hourly average ambient variables Temperature (**AT**), Pressure (**AP**), Relative Humidity (**RH**), and Exhaust Vacuum (**V**) to predict the net hourly electrical power output (**EP**) of the plant. Predicting a Power Plant's full load energy output is essential to maximize profit.

In a combined-cycle power plant, gas and steam turbines combined in one cycle through the *heat recovery steam generators* generate electricity[2]. While the Vacuum variable affects the Steam Turbine, the other three ambient variables affect the Gas Turbine performance.

This project aims to find a good Machine Learning model among those studied in the course. I will compare results based on Root Mean-Squared Error (RMSE) and algorithm performance in terms of the execution time of the model training.

Key steps

1. Train and predict with linear regression
2. Train and predict with Knn regression
3. Preprocessing based on Data visualization
4. Train and predict with Random Forest, rf method
5. Train and predict with Random Forest, Rborist method
6. Comparing results

[1] https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant

[2] GE clear explanation - https://www.ge.com/gas-power/resources/education/combined-cycle-power-plants

Analysis

Before starting, open the spreadsheet *Folds5x2_pp.xlsx*. Note that my dataset comes cleaned, has no missing or strange values, and is ready for machine learning analysis. Let's write some code to:

- Load the R packages.
- Load the dataset.
- Define the RMSE function.
- Split the dataset into train set and test set.

```
if(!require(tidyverse)) install.packages("tidyverse")
library(readxl)
if(!require(dplyr)) install.packages("dplyr")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(gridExtra)) install.packages("gridExtra")
if(!require(caret)) install.packages("caret")
if(!require(randomForest)) install.packages("randomForest")
if(!require(Rborist)) install.packages("Rborist")

# Load dataset and correct name typo PE in EP (Electrical Power)
df <- read_xlsx("./data/Folds5x2_pp.xlsx",
                col_names=c("AT", "V", "AP", "RH", "EP"),
                skip=1)

RMSE <- function(actual_outcomes, predictions){
  sqrt(mean((actual_outcomes - predictions)^2))
}

# Split dataset into train and test sets
set.seed(1972)
test_index <- createDataPartition(y = df$EP, times = 1, p = 0.2, list = FALSE)
df_train <- df[-test_index,]
df_test <- df[test_index,]
```

Unlike the *MovieLens project*, the test set is the validation set because Caret's function `train()` performs cross-validation all and only on the train set. A quick look at the data summary reveals, as expected, all the variables are continuous. I am omitting units of measure because they are irrelevant to my prediction.

```
summary(df)
```

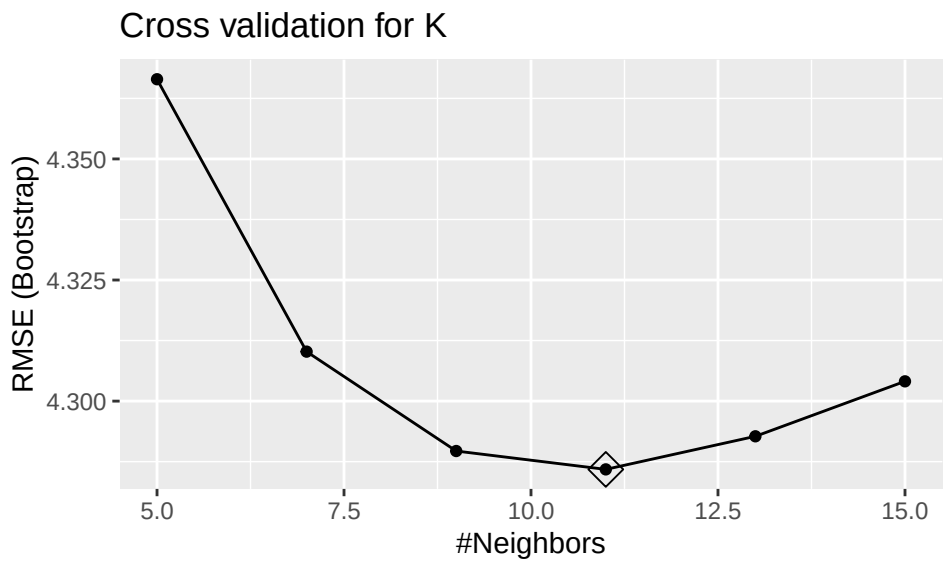
```
##        AT               V               AP               RH        
##  Min.   : 1.81   Min.   :25.36   Min.   : 992.9   Min.   : 25.56  
##  1st Qu.:13.51   1st Qu.:41.74   1st Qu.:1009.1   1st Qu.: 63.33  
##  Median :20.34   Median :52.08   Median :1012.9   Median : 74.97  
##  Mean   :19.65   Mean   :54.31   Mean   :1013.3   Mean   : 73.31  
##  3rd Qu.:25.72   3rd Qu.:66.54   3rd Qu.:1017.3   3rd Qu.: 84.83  
##  Max.   :37.11   Max.   :81.56   Max.   :1033.3   Max.   :100.16  
##        EP        
##  Min.   :420.3  
##  1st Qu.:439.8  
##  Median :451.6  
##  Mean   :454.4  
##  3rd Qu.:468.4  
##  Max.   :495.8
```

The Knn model

First, I will set the baseline with a **linear regression** model, and then I will train the **Knn** model tuning the **K** parameter through cross-validation.

```
# linear regression
fit_lm <- train(EP ~ ., data = df_train, method = 'lm')
predictions  <- predict(fit_lm, df_test)
rmse_lm <- RMSE(df_test$EP,  predictions)

# knn
fit_knn <- train(EP ~ ., data = df_train, method = 'knn',
             tuneGrid = data.frame(k = seq(5, 15, 2)))
predictions  <- predict(fit_knn, df_test)
rmse_knn <- RMSE(df_test$EP,  predictions)
ggplot(fit_knn, highlight = TRUE) +
  labs(title = "Cross validation for K")
```



`RMSE` = 4.5666273 for **linear regression**

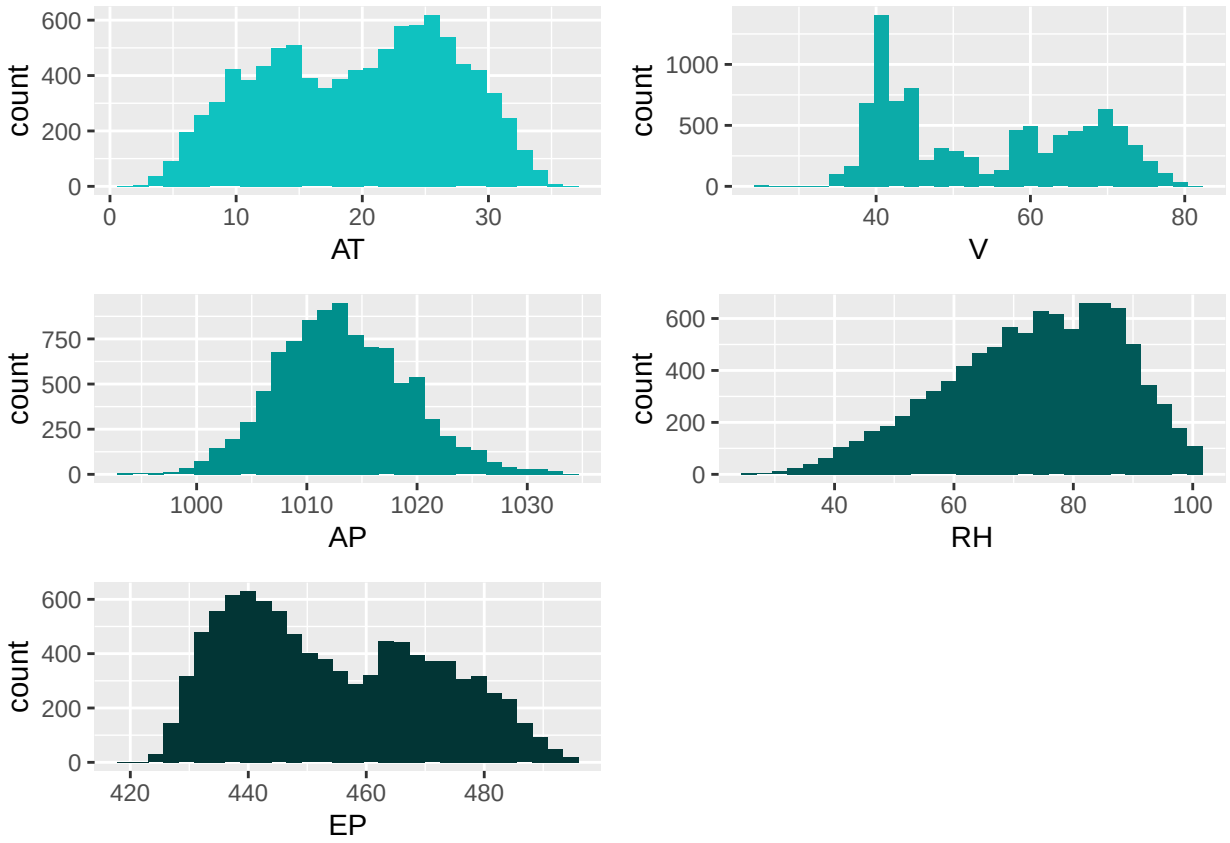
`RMSE` = 4.0400053 for **Knn** | optimized num. of neighbors `k` = 11 (see plot above)

I get a lower error for **Knn**. Before comparing **Knn** with **Random Forest**, which is challenging training the model, some preprocessing reveals that it is possible to remove one feature.

Preprocessing

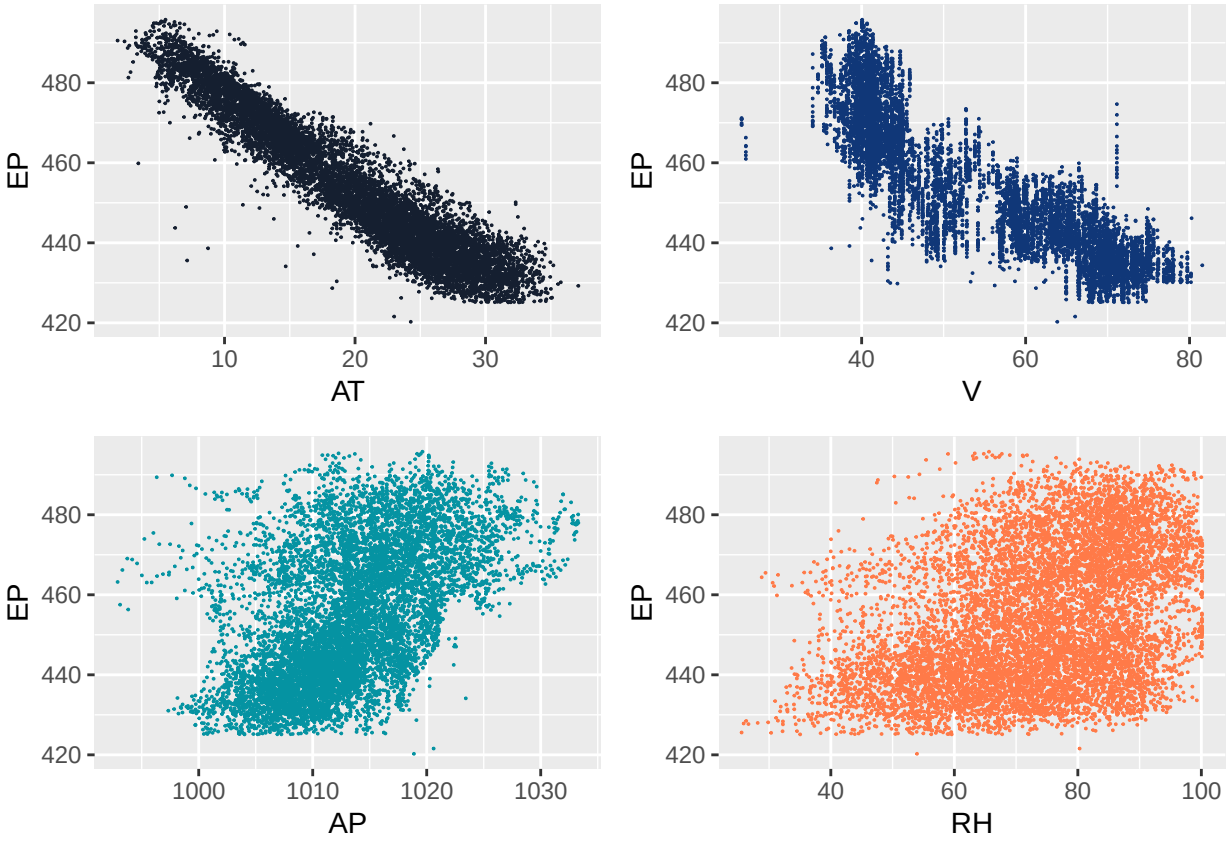
Plotting the distributions of all the single variables gives the following graphs.

```
# single variable distributions
p1 <- df %>% ggplot(aes(AT)) + geom_histogram(bins = 30, fill="#0FC2C0")
p2 <- df %>% ggplot(aes(V))  + geom_histogram(bins = 30, fill="#0CABA8")
p3 <- df %>% ggplot(aes(AP)) + geom_histogram(bins = 30, fill="#008F8C")
p4 <- df %>% ggplot(aes(RH)) + geom_histogram(bins = 30, fill="#015958")
p5 <- df %>% ggplot(aes(EP)) + geom_histogram(bins = 30, fill="#023535")
grid.arrange(p1, p2, p3, p4, p5, ncol = 2)
```



EP, **AT**, and **V** have all three two peaks, suggesting a strong correlation among them. In contrast, the ambient pressure **AP** and the relative humidity **RH** seem to have a weaker correlation. Further looking into scatterplots between the outcome and the features gives the following.

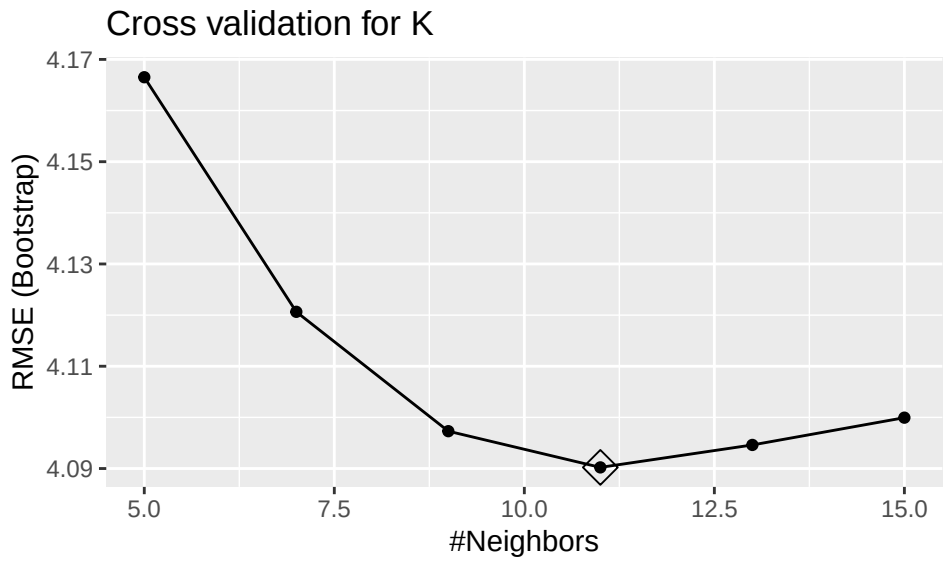
```
p1 <- df %>% ggplot(aes(x= AT, y= EP)) + geom_point(color = "#151F30", size = 0)
p2 <- df %>% ggplot(aes(x= V, y= EP))  + geom_point(color = "#103778", size = 0)
p3 <- df %>% ggplot(aes(x= AP, y= EP)) + geom_point(color = "#0593A2", size = 0)
p4 <- df %>% ggplot(aes(x= RH, y= EP)) + geom_point(color = "#FF7A48", size = 0)
grid.arrange(p1, p2, p3, p4, ncol = 2)
```



These scatterplots confirm the strong influence that **AT** and **V** have on the **EP**, while the less influential feature seems to be **RH**. Therefore, I hypothesize that the relative humidity **RH** feature is not helpful in the predictions, and I can remove it from the **Knn** model.

```
# Removing RH from features
df_train_no_RH <- df_train %>% select(AT, V, AP, EP)

fit_knn_no_RH <- train(EP ~ ., data = df_train_no_RH, method = 'knn',
                       tuneGrid = data.frame(k = seq(5, 15, 2)))
predictions  <- predict(fit_knn_no_RH, df_test)
rmse_knn_no_RH <- RMSE(df_test$EP,  predictions)
ggplot(fit_knn_no_RH, highlight = TRUE) +
  labs(title = "Cross validation for K")
```



`RMSE` = 3.8391301 for **Knn** on 3 features | optimized num. of neighbors `k` = 11 (see plot above)

The RMSE is even better, and I go on without the **RH** feature.

The Random Forest models

Random Forest is challenging when training the model compared to the **Knn**. However, there are many algorithms I can choose from the Caret's function `train()`. I will use the classical method **rf** and the more performing method **Rborist**. Both use cross-validation inside the `train()` function with the following tuning parameters.

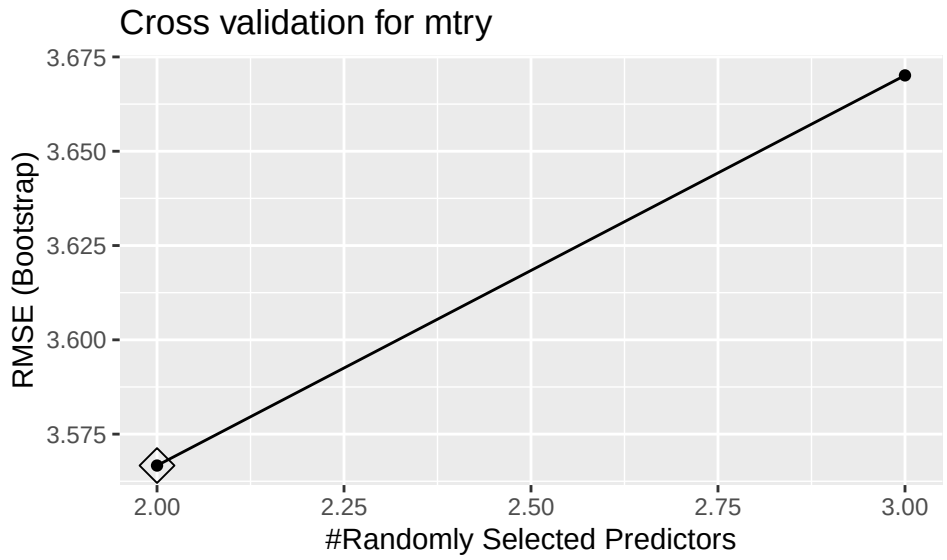
```
modelLookup() %>%
  select('model', 'parameter', 'label') %>%
  filter(model %in% c('rf', 'Rborist'))
```

```
##     model parameter                         label
## 1 Rborist predFixed #Randomly Selected Predictors
## 2 Rborist   minNode             Minimal Node Size
## 3      rf      mtry #Randomly Selected Predictors
```

Despite a different name, parameters `predFixed` and `mtry` are the same and represent the - *number of variables randomly sampled as candidates at each split*. In contrast, parameter `minNode` is tuneable only for the **Rborist** method and represents the - *minimum number of data points in the nodes of the trees*.

For these two algorithms, I will measure the resulting error RMSE and the performance as the execution time of the model training. For this purpose, I will embed the fitting process into the function `system.time({})` which returns that execution time. Let's start with the **rf** method.

```
rf_exec_time <- system.time({
  fit_rf_no_RH <- train(EP ~ ., data = df_train_no_RH, method = 'rf')
})
predictions  <- predict(fit_rf_no_RH, df_test)
rmse_rf_no_RH <- RMSE(df_test$EP,  predictions)
ggplot(fit_rf_no_RH, highlight = TRUE) +
  labs(title = "Cross validation for mtry")
```

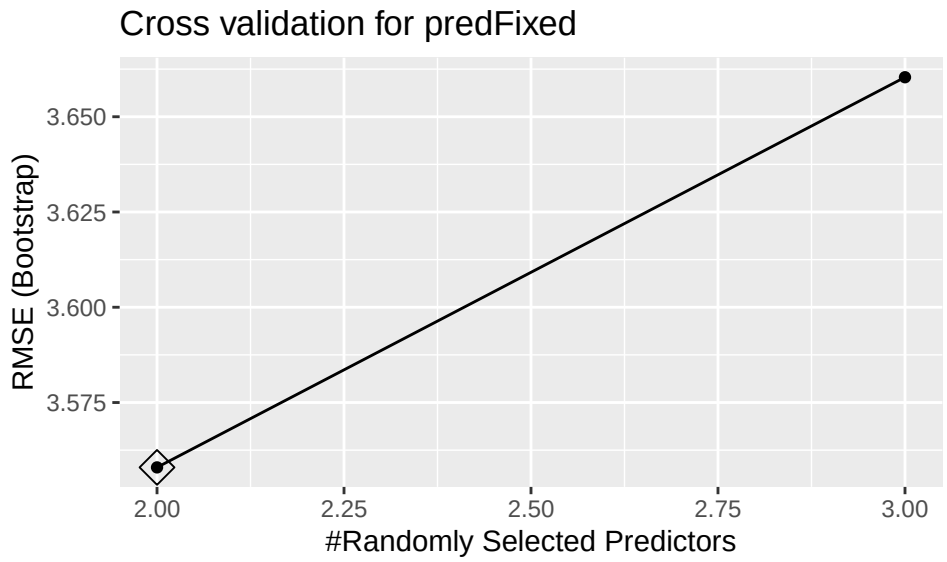


`RMSE` = 3.3674401 for **rf** method on 3 features | optimized `mtry` = 2 (see plot above)

I get a lower RMSE and an `execution time` of 33 mins on my laptop[3]. Can the **Rborist** method do better?

```
rborist_exec_time <- system.time({
  fit_rborist_no_RH <- train(EP ~ ., data = df_train_no_RH, method = 'Rborist')
})
predictions  <- predict(fit_rborist_no_RH, df_test)
rmse_rborist_no_RH <- RMSE(df_test$EP,  predictions)
ggplot(fit_rborist_no_RH, highlight = TRUE) +
  labs(title = "Cross validation for predFixed")
```

[3] Processor Intel(R) Pentium(R) CPU 2020M @ 2.40GHz - Installed RAM 16.0 GB



`RMSE` = 3.3650959 for **Rborist** method on 3 features | optimized `predFixed` = 2 (see plot above)

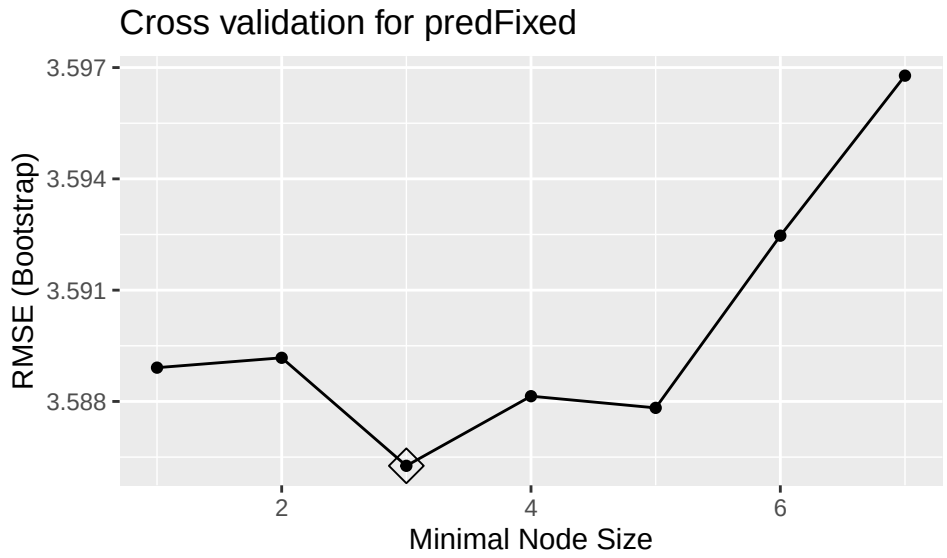
I get an `execution time` of 7 mins. **Rborist** compared to **rf** significantly improves the execution time while RMSE error is about the same. Note that the **Rborist** training just done held `minNode` constant at a value of 3 (see the print below).

```
print(fit_rborist_no_RH)
```

```
## Random Forest
##
## 7653 samples
##    3 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 7653, 7653, 7653, 7653, 7653, 7653, ...
## Resampling results across tuning parameters:
##
##   predFixed  RMSE      Rsquared   MAE
##   2          3.557973  0.9562768  2.526722
##   3          3.660367  0.9537320  2.591683
##
## Tuning parameter 'minNode' was held constant at a value of 3
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were predFixed = 2 and minNode = 3.
```

It may be worth trying to tune that parameter. To do this, I will use the `tuneGrid` option as follows

```
rborist_exec_time_2 <- system.time({
  fit_rborist_no_RH_2 <- train(EP ~ ., data = df_train_no_RH, method = 'Rborist',
                               tuneGrid = data.frame(predFixed = 2,
                                                     minNode = seq(1, 7)))
})
predictions  <- predict(fit_rborist_no_RH_2, df_test)
rmse_rborist_no_RH_2 <- RMSE(df_test$EP,  predictions)
ggplot(fit_rborist_no_RH_2, highlight = TRUE) +
  labs(title = "Cross validation for predFixed")
```



`RMSE` = 3.358945 for **Rborisr** method on 3 features | optimized `minNode` = 3 (see plot above)

I get a similar RMSE at the cost of about a triple `execution time` of 24 mins. Therefore, the previous training is preferable, which holds the `minNode` constant at a value of 3.

Results

Let's recap results and performance in a table.

Model	Method	RMSE	Exec. Time	Note
Linear Regression	`lm`	4.5666273	< 1 min	4 features
Knn	`knn`	4.0400053	< 1 min	4 features
Knn	`knn`	3.8391301	< 1 min	3 features
Random Forest	`rf`	3.3674401	33 min	3 features
Random Forest	`Rborist`	3.3650959	7 min	3 features

The two models analyzed do better than linear regression. **Knn** improves the RMSE error by 15.9% and **Random Forest** by 26.3%. The latter gives better results in exchange for longer execution times, reducible by choosing a good algorithm.

Conclusion

I developed a predictive model for a Power Plant Energy Output based on the dataset provided. After exploring the dataset, I removed one of the four features and tested four ML regression algorithms. I discovered a trade-off between RMSE and the execution time of the training.

Based on the Power Plant needs, one can choose the simpler **Knn** algorithm or the more complex **Rborist** for a smaller RMSE.

It should be exciting to test the effect of all the features' combinations on the result in future work.