Electrical power generation prediction

Created by Marcos Ikino

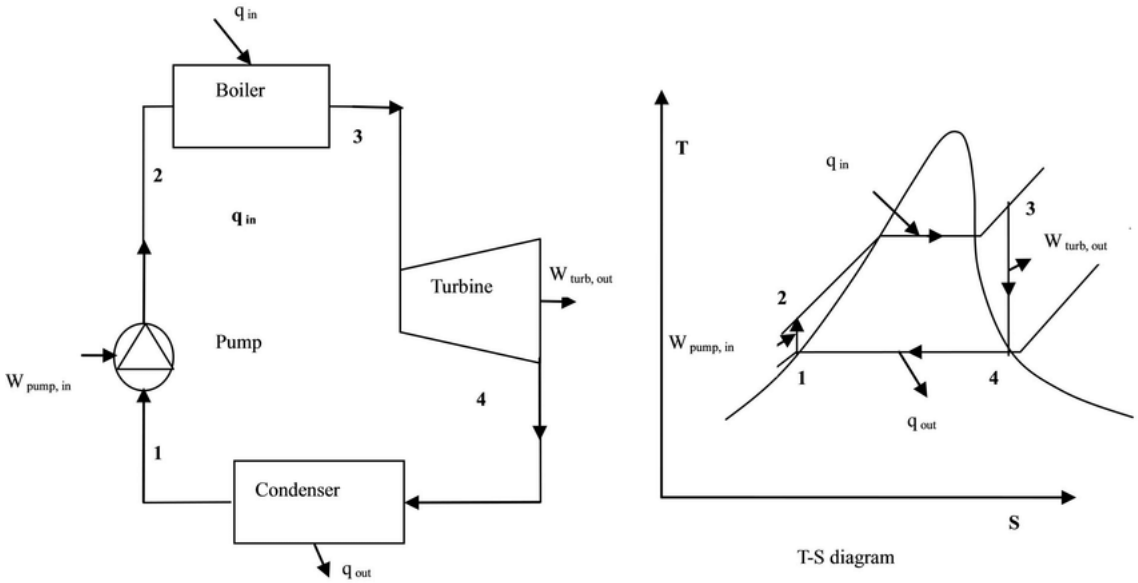
The dataset of this study was obtained from the UCI repository:
https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant

Attribute Information

The values consist of an hourly average of the ambient variables:

- Ambient Temperature(AT) range 1.81 to 37.11°C
- Ambient Pressure(AP) range 992.89 to 1033.30 milibar
- Relative Humidity(RH) range 25.56 to 100.16%
- Exhaust Vacuum(V) range 25.36 to 81.56 cm Hg
- Net hourly electrical energy output(PE) range 420.26 to 495.76 MW

The goal of this study is to predict the electrical power generation output of a combined gas & steam turbine represented by the PE variable on the dataset and with equivalency by the Wturbine on the diagram below. The predictive variables are composed by the ambient temperature (AT), ambient pressure (AP) and relative humidity (RH), which are environmental variables. The equipment represented by the exhaust vacuum (V) variable measures the pressure that is related on the turbine work process in the Rankine cycle, creating pressure difference on the last stages of the turbine expansion, avoiding the water formation on the turbine blades that could cause erosion on it. This process is characterized between points 3 and 4 on the diagram.



Rankine thermodynamic diagram

To predict the output values three machine learning algorithms will be developed, the Multivariate Linear Regression, the K-Nearest Neighbor and the Random Forest models, and the evaluation and comparison of the performances for each machine learning model, the R_squared and the Root Mean Square Error (RMSE) values were calculated, as well the mean relative error between the predicted and the actual values.

1. Exploratory data analysis

1.1 Importing and naming the dataset

```
library(readxl)

data <- read_excel('Cycle_power_plant_dataset.xlsx')
str(data)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    9568 obs. of  5 variables:
##  $ AT: num  14.96 25.18 5.11 20.86 10.82 ...
##  $ V : num  41.8 63 39.4 57.3 37.5 ...
##  $ AP: num  1024 1020 1012 1010 1009 ...
##  $ RH: num  73.2 59.1 92.1 76.6 96.6 ...
##  $ PE: num  463 444 489 446 474 ...
```

```
summary(data)
```

```
##        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  
##        PE        
##  Min.   :420.3  
##  1st Qu.:439.8  
##  Median :451.6  
##  Mean   :454.4  
##  3rd Qu.:468.4  
##  Max.   :495.8
```

1.2 Checking the data distribution behavior

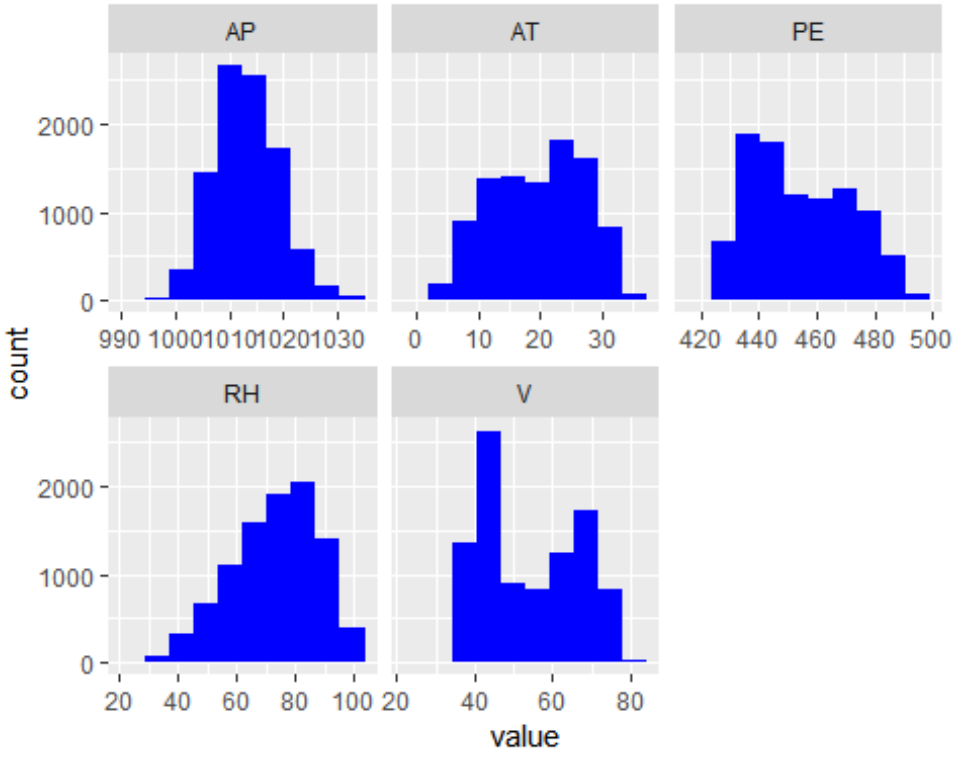
```
library(ggplot2)
library(tidyr)
library(e1071)

# Defining numerical variables
num_vars <- c("AT", "V", "AP", "RH", "PE")
```

```r
# Calculating the variable´s assimetry
skew_results <- do.call("rbind",lapply(data[, num_vars],function(x) skewness(x)))
colnames(skew_results) <- c('Skewness')
skew_results
```

```
##      Skewness
## AT -0.1363503
## V   0.1984588
## AP  0.2653615
## RH -0.4317034
## PE  0.3064133
```

```r
# Histogram graphs
ggplot(gather(data), aes(value)) +
    geom_histogram(bins = 10, fill = 'blue') +
    facet_wrap(~key, scales = 'free_x')
```



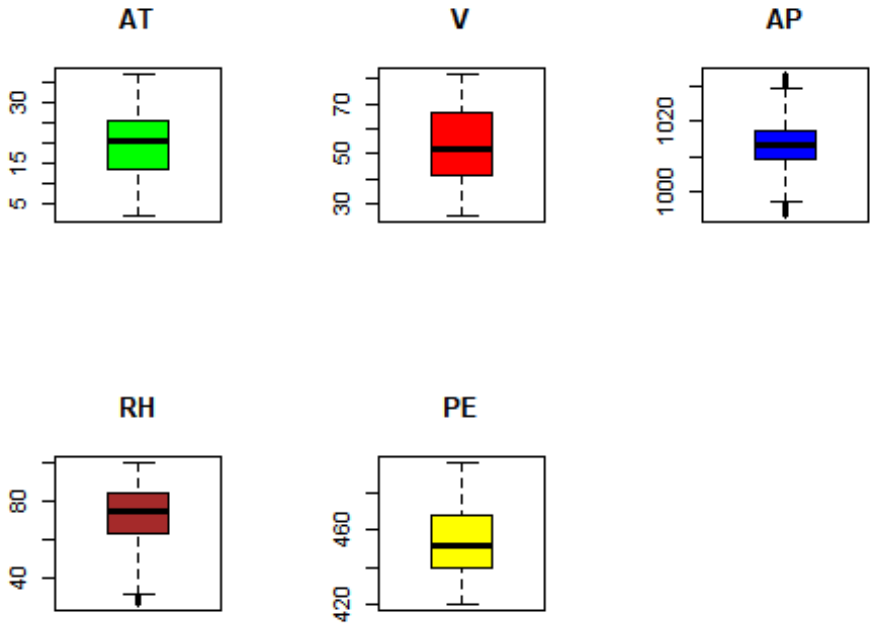
The histograms and the skewness values of all variables indicate that the data present a normal distribution.

1.3 Outlier identification

Applying boxplot visualization on the variables

```r
par(mfrow = c(2,3))
boxplot(data$AT, main = 'AT', col = 'green')
boxplot(data$V, main = 'V', col = 'red')
```

```
boxplot(data$AP, main = 'AP', col = 'blue')
boxplot(data$RH, main = 'RH', col = 'brown')
boxplot(data$PE, main = 'PE', col = 'yellow')
```



Defining the upper and lower limits to each variable

```
box_results <- do.call("cbind",lapply(data[, num_vars],function(x)
boxplot.stats(x)$stats))
row.names(box_results) <- c("lower whisker", "lower hinge", "median", "upper
hinge", "upper whisker")
box_results
```

```
##                    AT     V      AP      RH     PE
## lower whisker  1.810 25.36  996.87  31.150 420.26
## lower hinge   13.510 41.74 1009.10  63.325 439.75
## median        20.345 52.08 1012.94  74.975 451.55
## upper hinge   25.720 66.54 1017.26  84.830 468.43
## upper whisker 37.110 81.56 1029.41 100.160 495.76
```

Removing the outlier values

```
library(dplyr)

data <- data %>%
  filter(
        (AT >= 1.810 & AT <= 37.110) &
        (V >= 25.36 & V <= 81.56) &
```

```
        (AP >= 996.87 & AP <= 1029.41) &
        (RH >= 31.150 & RH <= 100.160) &
        (PE >= 420.26 & PE <= 495.76)
        )
str(data)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    9468 obs. of  5 variables:
##  $ AT: num  14.96 25.18 5.11 20.86 10.82 ...
##  $ V : num  41.8 63 39.4 57.3 37.5 ...
##  $ AP: num  1024 1020 1012 1010 1009 ...
##  $ RH: num  73.2 59.1 92.1 76.6 96.6 ...
##  $ PE: num  463 444 489 446 474 ...
```

2. Constructing the predictive models

```
library(caret)


trainIndex <- createDataPartition(data$PE,
                                  p = 0.7,
                                  list = FALSE,
                                  times = 1)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]
```

2.1 Linear regression application

```
control <- trainControl(method = 'cv', number = 10, savePredictions = TRUE)
model_lm <- train(PE ~.,
                  data = train_data,
                  method = 'lm',
                  trControl = control,
                  preProcess = c('center', 'scale'))
summary(model_lm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.377  -3.183  -0.125   3.150  17.757
##
```

```
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 454.23071    0.05674 8005.084  < 2e-16 ***
## AT          -14.59660    0.13852 -105.378  < 2e-16 ***
## V            -3.02831    0.11247  -26.925  < 2e-16 ***
## AP            0.46651    0.06812    6.848 8.15e-12 ***
## RH           -2.30378    0.07432  -31.000  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.62 on 6624 degrees of freedom
## Multiple R-squared:  0.9265, Adjusted R-squared:  0.9265
## F-statistic: 2.089e+04 on 4 and 6624 DF,  p-value: < 2.2e-16
```

2.2 KNN application

```
model_knn <- train(PE ~.,
                   data = train_data,
                   method = 'knn',
                   trControl = control,
                   preProcess = c('center', 'scale'),
                   tuneLength = 10
                   )
summary(model_knn)

model_knn$results
```

```
##     k     RMSE  Rsquared      MAE    RMSESD  RsquaredSD      MAESD
## 1   5 3.980675 0.9453913 2.859791 0.1655555 0.004635466 0.09167156
## 2   7 4.013171 0.9444566 2.915472 0.1807337 0.005026177 0.09614633
## 3   9 4.039141 0.9437667 2.949121 0.1610938 0.004575637 0.08026156
## 4  11 4.062290 0.9431219 2.988673 0.1654917 0.004741835 0.08521858
## 5  13 4.104555 0.9419335 3.028156 0.1712553 0.004879109 0.08858665
## 6  15 4.122742 0.9414472 3.056707 0.1628439 0.004712196 0.07934991
## 7  17 4.144242 0.9408515 3.081037 0.1547387 0.004589849 0.08416475
## 8  19 4.164592 0.9402919 3.101406 0.1569292 0.004676931 0.08675278
## 9  21 4.181094 0.9398498 3.124215 0.1482053 0.004442558 0.08581133
## 10 23 4.207193 0.9391092 3.147926 0.1474524 0.004504538 0.08260038
```

2.3 Random Forest application

```
model_rf <- train(PE ~.,
                  data = train_data,
                  method = 'rf',
                  trControl = control,
                  preProcess = c('center', 'scale')
```

```
                  )


model_rf$results
```

```
##   mtry     RMSE  Rsquared      MAE    RMSESD  RsquaredSD      MAESD
## 1    2 3.374455 0.9605430 2.382064 0.3333814 0.008681472 0.09561054
## 2    3 3.436246 0.9590769 2.425465 0.3286936 0.008725933 0.09329486
## 3    4 3.481080 0.9580243 2.457362 0.3156212 0.008491123 0.09147197
```

3. Predicting the results

For each algorithm model applied previously, it will be created its respective variable with the predictive results.

```
test_data$PE_pred_lm <- predict(model_lm, test_data)
test_data$PE_pred_knn <- predict(model_knn, test_data)
test_data$PE_pred_rf <- predict(model_rf, test_data)

head(test_data)
```

```
## # A tibble: 6 x 8
##      AT     V    AP    RH    PE PE_pred_lm PE_pred_knn PE_pred_rf
##   <dbl> <dbl> <dbl> <dbl> <dbl>      <dbl>       <dbl>      <dbl>
## 1 25.2   63.0 1020.  59.1  444.       444.        444.       444.
## 2 11.7   43.6 1015.  70.7  478.       473.        477.       473.
## 3 26.2   69.3 1009.  87.6  434.       435.        434.       436.
## 4 11.0   41.7 1023.  77.5  477.       474.        478.       476.
## 5 14.4   52.8 1024.  63.6  460.       467.        461.       461.
## 6  5.41  40.1 1019.  64.8  495.       488.        491.       493.
```

3.1 Verifying overfitting

The equivalency of the R squared values between the train and test models demonstrates that there is no overfitting on the training process, as showed on table below.

```
# Linear Regression
R_squared_train_lm <- cor(model_lm$pred$pred, model_lm$pred$obs)^2
R_squared_test_lm <- cor(test_data$PE, test_data$PE_pred_lm)^2
r1 <- cbind(R_squared_train_lm, R_squared_test_lm)

# KNN
R_squared_train_knn <- cor(model_knn$pred$pred, model_knn$pred$obs)^2
R_squared_test_knn <- cor(test_data$PE, test_data$PE_pred_knn)^2
r2 <- cbind(R_squared_train_knn, R_squared_train_knn)

# Random Forest
R_squared_train_rf <- cor(model_rf$pred$pred, model_rf$pred$obs)^2
R_squared_test_rf <- cor(test_data$PE, test_data$PE_pred_rf)^2
```

```r
r3 <- cbind(R_squared_train_rf, R_squared_train_rf)

results <- round(rbind(r1, r2, r3),3)
colnames(results) <- c('R_squared_train', 'R_squared_test')
rownames(results) <- c('Linear Regression', 'KNN', 'Random Forest')
results
```

```
##                   R_squared_train R_squared_test
## Linear Regression           0.926          0.932
## KNN                         0.942          0.942
## Random Forest               0.959          0.959
```

4. Mean relative error (%)

Calculating the mean relative error (%) between the predicted and the actual values to each model.

```r
library(dplyr)

# Linear Regression
lm_error <- test_data %>%
  select(PE, PE_pred_lm) %>%
  summarise(error = mean((abs(PE - PE_pred_lm)/PE)*100))
lm_error <- as.numeric(lm_error)

# KNN
knn_error <- test_data %>%
  select(PE, PE_pred_knn) %>%
  summarise(error = mean((abs(PE - PE_pred_knn)/PE)*100))
knn_error <- as.numeric(knn_error)

# Random Forest
rf_error <- test_data %>%
  select(PE, PE_pred_rf) %>%
  summarise(error = mean((abs(PE - PE_pred_rf)/PE)*100))
rf_error <- as.numeric(rf_error)

results <- round(rbind(lm_error, knn_error, rf_error),2)
colnames(results) <- c('Error(%)')
row.names(results) <- c('Linear Regression', 'KNN', 'Random Forest')
results
```

```
##                   Error(%)
## Linear Regression     0.80
## KNN                   0.62
## Random Forest         0.52
```

Conclusion

For the proposal of this study to predict the electrical output power generation, all machine learning models, the Linear Regression, the KNN and the Random Forest presented efficient results with the mean relative error results being below 1%. A reason that can explain this low error measures is that this power plant generator is in a very controlled environment with high precision components that get the values of the parameters with low rate error, and besides all of them are known and monitored.

Despite the slightly better result presented by the Random Forest model, the performance and the time to process and to fit the model were much higher compared to the Linear Regression and KNN models, even working in a not big dataset and with few variables involved.