FIT5149 S2 2021

Assessment 1: Electric Rotor Temperature Prediction

Student information

- Family Name: JAJORIA
- Given Name: PRASHANT
- Student ID: 31187366
- Student email: pjaj0001@student.monash.edu

Programming Language: R 3.5.1 in Jupyter Notebook

R Libraries used:

- psych
- ggplot2
- Hmisc
- GGally
- tidyverse
- leaps
- caret
- FNN

Table of Contents

1. Introduction

We are provided with 'pmsm_temperature_data_A1_2021.csv' file, which has observations from various sensors of a permanent magnet synchronous motor (PMSM). Due to limitations of being able to dirtectly measure the tempearture of the rotor because of safety issues and sensor outages, we study the dataset and make a Machine Learning model that can provide good prediction of the rotot temperature.

The first half of the notebook shows Exploratory Data Analysis (EDA), in which we understand the nature and explore the dataset. We take a closer look at each feature understand their distribution and how it affects out target variable. We make visual representations of different types to see the interactions between the variables and the target. Finally we summarize the key findings from the EDA and use these in the model development phase later.

The second half of the notebook discusses about making different Machine learning model and choosing the best model amongst them. We discuss the key points of model development, understand the role of features and how we arrive at our final Machine Learning model. We run our final ML model to get predictions on the Test datset and note key metrics about the model.

As we are given a single dataset, we partition it into two parts; training set which will be used for training the ML model. Testing dataset will be used to get key metrics to evaluate the performance of the ML model.

Loading the libraries needed for the notebook.

```
In [ ]: library(psych)
        library(ggplot2)
        library(reshape2)
        library(GGally)
        library(tidyverse)
        library(leaps)
        library(caret)
        library(FNN)
        library(gridExtra)
        library(car)
```

2. Exploratory Data Analysis

Load the dataset into the notebook

```
In [ ]: # Read the dataset
        dataset = read.csv('./pmsm_temperature_data_A1_2021.csv')
```

We remove the unnecessary columns as given in the requirement i.e. 'stator_yoke', 'stator_tooth', 'stator_winding'

```
In [ ]: # Remove the unnecessary columns
        dataset = dataset[,c(-10,-11,-12)]
```

Looking at the final features and the target variable 'pm' from the dataset.

```
In [ ]: colnames(dataset)
```

The dataset contains 15147 observations on the following 9 variables with **'profile_id'** used for sampling purpose. (The following detail is copied and pasted from here)

- ambinet Ambient temperature as measured by a thermal sensor located closely to the stator.
- coolant Coolant temperature. The motor is water cooled. Measurement is taken at outflow.
- u_d Voltage d-component
- u_q Voltage q-component
- motor_speed Motor speed
- torque Torque induced by current.
- i_d Current d-component
- i_q Current q-component
- pm Permanent Magnet surface temperature representing the rotor temperature. This was measured with an infrared
- profile_id Each measurement session has a unique ID. Make sure not to try to estimate from one session onto the other as they are

Looking at the datatypes of the dataset.

In [ ]:
```
str(dataset)
```

In [ ]:
```
# check the first few rows
head(dataset)
```

In [ ]:
```
# check the last few rows
tail(dataset)
```

We use the profile_id columns to split the dataset into training and testing sets. As stated in the specifications, we use profile_id = 72, 81 for Testing set and the rest of the data for training set. We also remove the profile_id after the data split process as it is no longer needed for the Model development process.

In [ ]:
```
# testing set
testing_set = dataset[dataset$profile_id %in% c(72,81), -10]
# training set
training_set = dataset[dataset$profile_id !=72 & dataset$profile_id !=8
1 , -10]
```

Computing basic statistics of the training set to know about the nature of each feature.

```
In [ ]: round(describe(training_set),3)
```

The following table gives the key points noted from the summary of the training set.

Attribute	Data Type	Comment
ambient	Numerical	Large skew value of -0.873
coolant	Numerical	Has low mean value of -0.009 with moderate positive skewness of data
u_d	Numerical	Small positive skewness
u_q	Numerical	Low skewness and small range from -1.815 to 1.770
motor_speed	Numerical	Lowest range amonght the features from -1.222 to 2.024
torque	Numerical	Highest range of value ranging from -3.316 to 3.014
i_d	Numerical	Moderately negative skewed
i_q	Numerical	Lowest skewness amongst the features with second highest range of values ranging from -3.309 to 2.914
pm	Numerical	Low negative skewness with relatively large range

Looking at the **mean** and the **standard deviation** of the features, the dataset looks **Standardization** as the mean of all the features is close to 0 and Standard devaition is close to 1.

2.1 Investigating each variable

Looking at Boxplots of each of the features

In [ ]:
```
# reshanping for plotting purpose
m1 <- melt(as.data.frame(training_set))

ggplot(m1,aes(x = variable,y = value)) +
    facet_wrap(~variable, scales="free") +
    geom_boxplot() +
    scale_y_continuous(labels=function (n) {format(n, scientific=FALSE
)})
```

We can notice outlier values in most of the columns. Removing these observations from the dataset will lead to loss of information, so we decide to keep them as of now for the analysis task.

In [ ]:
```
outliers_ambient = boxplot.stats(training_set$ambient)$out
outliers_u_d = boxplot.stats(training_set$u_d)$out
outliers_torque = boxplot.stats(training_set$torque)$out
outliers_i_q = boxplot.stats(training_set$i_q)$out

outliers <- c()
for (name in colnames(training_set)){
    outliers <- c(outliers,
                  (which(training_set[,name] %in% boxplot(training_set[
,name],plot=F)$out)) )
}
total_outliers = length(unique(outliers))
```

In [ ]:
```
##outlier values
paste( "There are" , length(outliers_ambient), "outliers in feature Amb
ient" )
paste( "There are" , length(outliers_u_d), "outliers in feature u_d" )
paste( "There are" , length(outliers_torque), "outliers in feature torq
ue" )
paste( "There are" , length(outliers_i_q), "outliers in feature i_q" )

paste("In total there are", total_outliers,
      "outliers in the dataset, which forms",
      (total_outliers / dim(training_set)[1]) * 100,
        "% of the training dataset"
      )
```

Looking at distributions of each of the features

In [ ]:
```
library(Hmisc)
hist.data.frame(training_set)
```

Key points as noted from the graphs:

- Target variable 'pm' is normally distributed.
- 'i_d' is left skewed.
- 'torque' and 'i_q' is close to Normal distribution.
- 'coolant' and 'motor_speed' has right skewness.

2.2 Can we make the features normal ?

1. i_d - As feature is left skewed we try to make it normal by doing log, square or cube root transformation.

In [ ]:
```
g1 = ggplot(aes(x=i_d), data=training_set) +
      geom_histogram(bins=20)

g2 = ggplot(aes(x=log(i_d+max(i_d)+10,10)), data=training_set) +
      geom_histogram(bins=20)

g3 = ggplot(aes(x=(i_d)**2), data=training_set) +
      geom_histogram(bins=20)

g4 = ggplot(aes(x=(i_d +max(i_d)+10)**(1/3)), data=training_set) +
      geom_histogram(bins=20)
```

```
gridExtra::grid.arrange(g1,g2,g3,g4, nrow=2,ncol=2)
```

The i_d feature cannot be made normal as the transformations are also skewed and do not help the analysis.

1. coolant - As feature is right skewed we try to make it normal by doing log, square or cube root transformation.

In [ ]:
```
g1 = ggplot(aes(x=coolant), data=training_set) +
      geom_histogram(bins=20)

g2 = ggplot(aes(x=log(coolant+max(i_d)+10,10)), data=training_set) +
      geom_histogram(bins=20)

g3 = ggplot(aes(x=(coolant)**2), data=training_set) +
      geom_histogram(bins=20)

g4 = ggplot(aes(x=(coolant +max(coolant)+10)**(1/3)), data=training_set
) +
      geom_histogram(bins=20)



gridExtra::grid.arrange(g1,g2,g3,g4, nrow=2,ncol=2)
```

Transformations on 'coolant' does not improve the shape of the Distributions and do not help the analysis.

1. motor_speed - As feature is right skewed we try to make it normal by doing log, square or cube root transformation.

In [ ]:
```
g1 = ggplot(aes(x=motor_speed), data=training_set) +
      geom_histogram(bins=20)
```

```
g2 = ggplot(aes(x=log(motor_speed+max(i_d)+10,10)), data=training_set)
+
      geom_histogram(bins=20)

g3 = ggplot(aes(x=(motor_speed)**2), data=training_set) +
      geom_histogram(bins=20)

g4 = ggplot(aes(x=(motor_speed +max(motor_speed)+10)**(1/3)), data=trai
ning_set) +
      geom_histogram(bins=20)



gridExtra::grid.arrange(g1,g2,g3,g4, nrow=2,ncol=2)
```

Transformations on 'motor_speed' does not improve the shape of the Distributions and do not help the analysis.

2.3 Checking correlatin between variables

```
In [ ]: ggpairs(training_set)
```

Looking at the above pairs plot we observe the following things:

- a positive correlation between torque and i_q
- a negative correlation between u_d and i_q
- a positive correlation between u_q and motor_speed
- a negative correlation between motor_speed and i_d
- a negative correlation between u_d and torque

3. Task A : Prediction Task

3.1 Function to Calculate Model Accuracy Statistics

Name: Model.Accuracy

Input parameters:

- predicted - a vector of predictions made by Model
- target - a vector containing the target values for the predictions

Return Value:

A list containing:

- rss - Residaul Sum of squares -
- tss - Total sum of squares
- rsquared - the R-Squared value calculated from the predicted and target values
- rmse - Root Mean Square error

Description:

Calculate the TSS, RSS, R^2, RMSE as follows:

- TSS: $\sum_{i=1}^{n}(y_i - \bar{y})^2$
- RSS: $\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
- R-Squared value: $R^2 = 1 - \frac{RSS}{TSS}$
- RMSE: $RMSE = \sqrt{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 / N}$

In [ ]:
```
Model.Accuracy <- function(predicted, target) {
    rss <- 0
    tss <- 0
    se <- 0
    target.mean <- mean(target)
    mse <- 0

    for (i in 1:length(predicted)) {
        rss <- rss + (predicted[i]-target[i])^2
        tss <- tss + (target[i]-target.mean)^2

        se <- se + (predicted[i]-target[i])^2
    }
    rmse <- sqrt(se/length(predicted))
    rsquared <- 1 - rss/tss

    mse = mean((predicted - target)^2)

    return(list(rsquared=rsquared, rss=rss, tss=tss, rmse=rmse, mse=mse
))
}
```

Name: rmse

Input parameters:

- predicted - a vector of predictions made by Model
- target - a vector containing the target values for the predictions

Return Value:

Double value

- rmse - Root Mean Square error

In [ ]:
```
rmse <- function(predicted, target) {
    se <- 0
    for (i in 1:length(predicted)) {
        se <- se + (predicted[i]-target[i])^2
    }
    return (sqrt(se/length(predicted)))
}
```

3.2 Model 1 . Linear Regression Model

Using the EDA done above, we start by building a Linear Regression using all the features to predict the target variable 'pm'.

We us this Linear Regression model as the Base line model to compare other Linear Regression models built in this notebook.

3.2.1 Simple Linear regression model on Original Dataset

In [ ]:
```
lm.model.base <- lm(pm ~ ., data=training_set)
summary(lm.model.base)
```

An adjusted R-squared ($R2$) of 49.1% means that the model is able to explain 49.1% of the variations in the values of 'pm'.

The pr(>|t|) value for all the coefficeints shows that all the coefficients are significant at 0.05 level.

p-value of 1.17e+03 on 8 and 9643 DF, tells that the hypothesis that the model explains nothing is rejected.

3.2.2 Adding new features

The dataset contains voltage and current that are resolved into d and q components. Thus we can get the effective magnitude using basic equation as follows:

$$i_s = \sqrt{(i_q)^n + (i_d)^2}$$

$$u_s = \sqrt{(u_q)^n + (u_d)^2}$$

In [ ]:
```
i_s = c(sqrt(training_set['i_d']*training_set['i_d'] + training_set['i_
q']*training_set['i_q']))
i_s_test = c(sqrt(testing_set['i_d']*testing_set['i_d'] + testing_set[
'i_q']*testing_set['i_q']))
```

```
u_s = c(sqrt(training_set['u_d']*training_set['u_d'] + training_set['u_
q']*training_set['u_q']))
u_s_test = c(sqrt(testing_set['u_d']*testing_set['u_d'] + testing_set[
'u_q']*testing_set['u_q']))
```

In [ ]:
```
# adding the new features to training and test set
training_set_new = data.frame(cbind(training_set,i_s,u_s))
test_set_new = data.frame(cbind(testing_set,i_s_test,u_s_test))
```

In [ ]:
```
colnames(training_set_new) =  c('ambient' ,'coolant' ,'u_d', 'u_q' ,'mo
tor_speed' ,'torque', 'i_d' ,'i_q' ,'pm','i_s','u_s')
colnames(test_set_new) =  c('ambient' ,'coolant' ,'u_d', 'u_q' ,'motor_
speed' ,'torque', 'i_d' ,'i_q' ,'pm','i_s','u_s')
```

Computing Power the Active and Reactive power of voltage and current in dq reference frame using the below formulas:

active_power = $\frac{3}{2}((V_d * I_d) + (V_q * I_d))$

reactive_power = $\frac{3}{2}((V_q * I_d) - (V_d * I_d))$

In [ ]:
```
# computing active power
active_power = c(1.5* (training_set_new[,3]*training_set_new[,7] + trai
ning_set_new[,4]*training_set_new[,8]) )
training_set_new = data.frame(cbind(training_set_new, active_power))

active_power = c(1.5* (test_set_new[,3]*test_set_new[,7] + test_set_new
[,4]*test_set_new[,8]) )
test_set_new = data.frame(cbind(test_set_new, active_power))
```

In [ ]:
```
# computing reactive power
reactive_power = c(1.5 * (training_set_new[,4]*training_set_new[,8] - t
raining_set_new[,3]*training_set_new[,7])  )
training_set_new = data.frame(cbind(training_set_new, reactive_power))
```

```
reactive_power = c(1.5 * (test_set_new[,4]*test_set_new[,8] - test_set_
new[,3]*test_set_new[,7]) )
test_set_new = data.frame(cbind(test_set_new, reactive_power))
```

3.2.3 Simple Linear regression model on dataset with new features

In [ ]:
```
lm.model.no_interactions <- lm(pm ~ ., data=training_set_new)
summary(lm.model.no_interactions)
```

Adding the new features have increased the Adj-R squared value from 0.491 to 0.503 as compared the baseline model. One thing that we can notice is all the features are significant at 0.05 level. This is due to the fact that we have not yet introduced Interaction terms that we explored in the EDA.

3.2.4 Adding Interaction terms to our model

The interaction terms that are added are the features that we saw have high correlation between each other. Introducing them can help explain the missing variance of the previous model and supports the EDA.

In [ ]:
```
lm.model.interactions <- lm(pm ~ . + torque*i_q + u_d*i_q + u_q*motor_s
peed + motor_speed*i_d + u_d*torque , data=training_set_new)
summary(lm.model.interactions)
```

We can see from the summary of the new model that introducing Interaction terms have increased the Adj R-squared value by a significant value from 0.491 to 0.573 comparing to the baseline model. This is a 16.7% increase in the Adj R-Squared value comparing to the baseline model.

4. Feature Selection

The Linear Regression that we have built up till now have been using all the features and different interactions between them. This can make the model overfit the training set, increase the computation time and difficult to interpret the model due to increases feature space.

In this section we will try to reduce the feature space to make the models light weight, as this is one of our requiremnts for the prediction task. We perform step wise selection with exhaustive, forward and backward approach to compare the features obtained at each stage and select the final feature for the Linear Regression model.

4.1. Exhaustive method

In [ ]:
```
lm.model.exhaustive <- regsubsets(pm ~ . + torque*i_q + u_d*i_q + u_q*m
otor_speed + motor_speed*i_d + u_d*torque , data = training_set_new)
reg.summary <- summary(lm.model.exhaustive)
reg.summary
```

Looking at the C_p, BIC, Adj R-squared and RSS for Exhaustive stepwise selection and how they change as the number of features increases.

In [ ]:
```
par(mfrow = c(2, 2))
plot(reg.summary$cp, xlab = "Number of variables", ylab = "C_p", type =
"l")
points(which.min(reg.summary$cp), reg.summary$cp[which.min(reg.summary$
cp)], col = "red", cex = 2, pch = 20)
plot(reg.summary$bic, xlab = "Number of variables", ylab = "BIC", type
= "l")
points(which.min(reg.summary$bic), reg.summary$bic[which.min(reg.summar
y$bic)], col = "red", cex = 2, pch = 20)
plot(reg.summary$adjr2, xlab = "Number of variables", ylab = "Adjusted
 R^2", type = "l")
points(which.max(reg.summary$adjr2), reg.summary$adjr2[which.max(reg.su
mmary$adjr2)], col = "red", cex = 2, pch = 20)
plot(reg.summary$rss, xlab = "Number of variables", ylab = "RSS", type
= "l")
mtext("Plots of C_p, BIC, adjusted R^2 and RSS for forward stepwise sel
ection", side = 3, line = -2, outer = TRUE)
```

We can observe from the above plots that the model with 8 features is overall the best model.

4.2. Forward search

In [ ]:
```
lm.model.forward <- regsubsets(pm ~ . + torque*i_q + u_d*i_q + u_q*moto
r_speed + motor_speed*i_d + u_d*torque , data = training_set_new)
reg.summary <- summary(lm.model.forward)
reg.summary
```

Lets see the optimum number of features.

In [ ]:
```
par(mfrow = c(2, 2))
plot(reg.summary$cp, xlab = "Number of variables", ylab = "C_p", type =
"l")
points(which.min(reg.summary$cp), reg.summary$cp[which.min(reg.summary$
cp)], col = "red", cex = 2, pch = 20)
plot(reg.summary$bic, xlab = "Number of variables", ylab = "BIC", type
= "l")
points(which.min(reg.summary$bic), reg.summary$bic[which.min(reg.summar
y$bic)], col = "red", cex = 2, pch = 20)
plot(reg.summary$adjr2, xlab = "Number of variables", ylab = "Adjusted
 R^2", type = "l")
points(which.max(reg.summary$adjr2), reg.summary$adjr2[which.max(reg.su
mmary$adjr2)], col = "red", cex = 2, pch = 20)
plot(reg.summary$rss, xlab = "Number of variables", ylab = "RSS", type
= "l")
mtext("Plots of C_p, BIC, adjusted R^2 and RSS for forward stepwise sel
ection", side = 3, line = -2, outer = TRUE)
```

Forward method also gives the same set of features as Exhaustive method with 8 features selected.

4.3 Backward Search

We finally perform Backward seaerch to see if we can find new features that can be include in the Linear Regression model.

In [ ]:
```
lm.model.backward <- regsubsets(pm ~ . + torque*i_q + u_d*i_q + u_q*mot
or_speed + motor_speed*i_d + u_d*torque , data = training_set_new)
reg.summary <- summary(lm.model.backward)
reg.summary
```

In [ ]:
```
par(mfrow = c(2, 2))
plot(reg.summary$cp, xlab = "Number of variables", ylab = "C_p", type =
"l")
points(which.min(reg.summary$cp), reg.summary$cp[which.min(reg.summary$
cp)], col = "red", cex = 2, pch = 20)
plot(reg.summary$bic, xlab = "Number of variables", ylab = "BIC", type
= "l")
points(which.min(reg.summary$bic), reg.summary$bic[which.min(reg.summar
y$bic)], col = "red", cex = 2, pch = 20)
plot(reg.summary$adjr2, xlab = "Number of variables", ylab = "Adjusted
 R^2", type = "l")
points(which.max(reg.summary$adjr2), reg.summary$adjr2[which.max(reg.su
mmary$adjr2)], col = "red", cex = 2, pch = 20)
plot(reg.summary$rss, xlab = "Number of variables", ylab = "RSS", type
= "l")
mtext("Plots of C_p, BIC, adjusted R^2 and RSS for forward stepwise sel
ection", side = 3, line = -2, outer = TRUE)
```

Backward search also gives the same set of features i.e. 8.

4.4 Final reduced features

We can thus say the reduced feature set contains following terms:

1. ambient
2. coolant
3. u_d
4. u_q
5. motor_speed
6. i_d
7. u_q:motor_speed
8. motor_speed:i_d

4.5 Reduced Linear regression model

We us the reduced set of features from the result of feature selection process done to make the model light weight.

In [ ]:
```
lm.model.reduced <- lm(pm ~ . + u_q*motor_speed + motor_speed*i_d, data
=training_set[,-c(6,8,10,11,12)])
summary(lm.model.reduced)
```

In [ ]:
```
par(mfrow=c(2,2))
plot(lm.model.reduced)
```

We can see that the Linear model is greatly affected by some outliers. Lets check the influencial outliers

In [ ]:
```
outlierTest(lm.model.reduced, cutoff=0.05, digits = 1)
```

Plotting the Influence plot to see is these are influencing our model.

In [ ]:
```
influencePlot(lm.model.reduced, scale=5, id.method="noteworthy", main=
"Influence Plot", sub="Circle size is proportial to Cook's Distance" )
```

The influence plot shows five outliers but have reported 2 of them:

- 4162, 4172 have a large studentized residual and is also an outlier. Large value of studentized residuals means the modle has performed badly for this datapoint.

The datapoints 4192, 4201, 4221, 1471 and 1909 also have large Student Residuals. We make a new Linear model and evaluate the model for performance.

4.6 Removing large Outliers

In [ ]:
```
lm.model.reduced.no_outliers <- lm(pm ~ . + u_q*motor_speed + motor_spe
ed*i_d, data=training_set[ -c(4162,4172,4192,4201,4221,1471) ,-c(6,8,10
,11,12)])
summary(lm.model.reduced.no_outliers)
```

We can observe from the Adj R-squared value that the model is now able to explain more variance of the dataset. The Adj R-Squared value has increased from 0.546 to 0.607 as compared to the previous model with outliers. And we have an overall increase of 23.62% compared to the Baseline model.

In [ ]:
```
par(mfrow=c(2,2))
plot(lm.model.reduced.no_outliers)
```

The model plots show:

- Residual vs Fitted - shows the residuals are distributed in a better way as compared to the previous model. The residuals are more evenly distributed along the regression line, meeting the assumption of Linear regression model.
- Normal Q-Q - the residauls have now been evenly distributed making them normally distributed.
- Scale-Location - Variance of residuals looks consistent around 1.
- Residuals vs Leverage - The chart still shows some outliers though they have a small Cook distance.

4.7 Comparing the Linear Regression models

A. Training error

As we noticed in the model development phase that there are few outliers that deviate to a large extent for the model. We make predictions for the training set without those outliers and will predict the values for outliers seperately to investigate further.

Training error without the outliers

In [ ]:
```
# predictions for different models
y_pred_train_1 = predict(lm.model.reduced.no_outliers, training_set_new
[-c(4162,4172,4192,4201,4221,1471),])
y_pred_train_2 = predict(lm.model.reduced, training_set_new[-c(4162,417
2,4192,4201,4221,1471),])
y_pred_train_3 = predict(lm.model.base, training_set[-c(4162,4172,4192,
4201,4221,1471),])
y_pred_train_4 = predict(lm.model.no_interactions, training_set_new[-c(
4162,4172,4192,4201,4221,1471),])
y_pred_train_5 = predict(lm.model.interactions, training_set_new[-c(416
2,4172,4192,4201,4221,1471),])

# metrics
paste("Chosen model")
Model.Accuracy(y_pred_train_1,training_set_new[-c(4162,4172,4192,4201,4
221,1471),]$pm)
paste("Model with outliers")
Model.Accuracy(y_pred_train_2,training_set_new[-c(4162,4172,4192,4201,4
221,1471),]$pm)
paste("BaseLine model")
Model.Accuracy(y_pred_train_3,training_set[-c(4162,4172,4192,4201,4221,
1471),]$pm)
paste("Model without Intearction terms")
```

```
Model.Accuracy(y_pred_train_4,training_set_new[-c(4162,4172,4192,4201,4
221,1471),]$pm)
paste("Model with interaction terms")
Model.Accuracy(y_pred_train_5,training_set_new[-c(4162,4172,4192,4201,4
221,1471),]$pm)
```

Error for the outliers

In [ ]:

```
# predictions for different models
y_pred_train_1 = predict(lm.model.reduced.no_outliers, training_set_new
[c(4162,4172,4192,4201,4221,1471),])
y_pred_train_2 = predict(lm.model.reduced, training_set_new[c(4162,4172
,4192,4201,4221,1471),])
y_pred_train_3 = predict(lm.model.base, training_set[c(4162,4172,4192,4
201,4221,1471),])
y_pred_train_4 = predict(lm.model.no_interactions, training_set_new[c(4
162,4172,4192,4201,4221,1471),])
y_pred_train_5 = predict(lm.model.interactions, training_set_new[c(4162
,4172,4192,4201,4221,1471),])

# metrics
paste("Chosen model")
Model.Accuracy(y_pred_train_1,training_set_new[c(4162,4172,4192,4201,42
21,1471),]$pm)
paste("Model with outliers")
Model.Accuracy(y_pred_train_2,training_set_new[c(4162,4172,4192,4201,42
21,1471),]$pm)
paste("BaseLine model")
Model.Accuracy(y_pred_train_3,training_set[c(4162,4172,4192,4201,4221,1
471),]$pm)
paste("Model without Intearction terms")
Model.Accuracy(y_pred_train_4,training_set_new[c(4162,4172,4192,4201,42
21,1471),]$pm)
paste("Model with interaction terms")
Model.Accuracy(y_pred_train_5,training_set_new[c(4162,4172,4192,4201,42
21,1471),]$pm)
```

We can observe from the predictions for outliers that the Linear model cannot make good predictions for outliers. This means that the outliers need to be further studied seperately. Following are the outliers that need further investigation, which is out of the scope of this notebook. One possible approach is to make a seperate Machine learning model for such data points so that we can better handle the outliers.

In [ ]:
```
training_set_new[c(4162,4172,4192,4201,4221,1471),]
```

B. Testing error

In [ ]:
```
y_pred_1 = predict(lm.model.reduced.no_outliers, test_set_new)
y_pred_2 = predict(lm.model.reduced, test_set_new)
y_pred_3 = predict(lm.model.base, testing_set)
y_pred_4 = predict(lm.model.no_interactions, test_set_new)
y_pred_5 = predict(lm.model.interactions, test_set_new)
```

In [ ]:
```
paste("Chosen model")
Model.Accuracy(y_pred_1,test_set_new$pm)
paste("Model with outliers")
Model.Accuracy(y_pred_2,test_set_new$pm)
paste("BaseLine model")
Model.Accuracy(y_pred_3,testing_set$pm)
paste("Model without Intearction terms")
Model.Accuracy(y_pred_4,test_set_new$pm)
paste("Model with interaction terms")
Model.Accuracy(y_pred_5,test_set_new$pm)
```

5. Model 2 - KNN

5.1 Why KNN ?

- KNN is easy to implement and also easy to intrepret the results.

- Only 1 hyper paramaters i.e.k
- KNN can be used for regression task as well.
- It has no assumptions about the underlying distribution of data.
- Can we easily adapted to new datapopints.

5.2 Scaling the features

As we are using KNN algorithm, we perform feature scaling to bring all the values in same scale. KNN being a distance based algorithm will be highly fluctuating if the features are not scaled

In [ ]:
```
min_max_processor1 <- preProcess(training_set_new, method=c("range"))
min_max_training_set <- predict(min_max_processor1,training_set_new)

min_max_processor2 <- preProcess(test_set_new, method=c("range"))
min_max_testing_set <- predict(min_max_processor2,test_set_new)
```

5.3 Cross Validation

In [ ]:
```
set.seed(123)
grid = expand.grid(k = seq(1,20)) # dataframe of different k values

train.control.knn <- trainControl(method = "cv",
                                  number = 10,
                                  search="grid",)

# Train the model
knn_cv_grid <- train(pm ~. ,
                data = min_max_training_set,
                method = "knn",
                trControl = train.control.knn,
                tuneGrid = grid)

knn_cv_grid
```

```
In [ ]: plot(knn_cv_grid)
```

We can see from the cross-validation that k=4 is a good choice for KNN to predict the 'pm' feature. So we now build the KNN model with k=4 on the whole training dataset.

5.4 Training Error

```
In [ ]: prediction.knn.train = knn.reg(
                                    train = min_max_training_set[,-9],
                                    y = min_max_training_set[,9],
                                    test = min_max_training_set[,-9],
                                    k = 4
                                )
```

```
In [ ]: df.prediction.knn.train = data.frame(observed = min_max_training_set$pm
        ,
                                       predicted = prediction.knn.train$pred )
```

```
In [ ]: df.prediction.knn.train %>%
          ggplot(aes(observed, predicted)) +
            geom_hline(yintercept = 0) +
            geom_point() +
            stat_smooth(method = "loess") +
            theme_minimal()+
            ggtitle("Predictions on Training Set")
```

5.5 Test Error

```
In [ ]: prediction.knn.test = knn.reg(
                                    train = min_max_training_set[,-9],
                                    y = min_max_training_set[,9],
                                    test = min_max_testing_set[,-9],
                          k = 4
                      )
```

In [ ]:
```
df.prediction.knn.test = data.frame(observed = min_max_testing_set$pm,
                       predicted = prediction.knn.test$pred )
```

In [ ]:
```
df.prediction.knn.test %>%
  ggplot(aes(observed, predicted)) +
    geom_hline(yintercept = 0) +
    geom_point() +
    stat_smooth(method = "loess") +
    theme_minimal()+
    ggtitle("Predictions on Testing Set")
```

We can see from the above plots that the value of k = 4 obtained from cross-validation gives good predictions when the target variable 'pm' is less than 0.5. But for latger values it underestimates the target.

5.6 Can we find a better K value ?

In [ ]:
```
df.error.knn = data.frame(k = double(),
                          train.rmse = double(),
                          test.rmse = double()
                          )
```

In [ ]:
```
for (k in 1:200){
    prediction.knn.train = knn.reg(
                            train = min_max_training_set[,-9],
                            y = min_max_training_set[,9],
                            test = min_max_training_set[,-9],
                            k = k
                        )

    prediction.knn.test = knn.reg(
                            train = min_max_training_set[,-9],
                             y = min_max_training_set[,9],
                             test = min_max_testing_set[,-9],
                             k = k
                         )

    rmse.train = rmse(prediction.knn.train$pred, min_max_training_set$p
m)
    rmse.test = rmse(prediction.knn.test$pred, min_max_testing_set$pm)

    df.error.knn = rbind(df.error.knn, data.frame(k = k, train.rmse = r
mse.train, test.rmse = rmse.test) )
}
```

In [ ]:
```
prediction_errors_long <- melt(df.error.knn, id="k")
```

In [ ]:
```
ggplot(data=prediction_errors_long,
       aes(x=k, y=value, colour=variable)) +
       geom_line()
```

We can observe from the graph above that the test and train error do not converge for k in range [0,200]. Also when we increase the K value we are underfitting the training set and overfitting to the test test. Finding a trade off between the two means finding large value of k. This will be computationally expensive for the given requirement and thus KNN is thus not a good candidate.

6. Comparing the metrics of both Linear model and KNN

6.1 Training error

In [ ]:
```
y_pred_train_1 = predict(lm.model.reduced.no_outliers, training_set_new
[-c(4162,4172,4192,4201,4221,1471),])
paste("Linear regression model")
Model.Accuracy(y_pred_train_1,training_set_new[-c(4162,4172,4192,4201,4
221,1471),]$pm)
```

In [ ]:
```
paste("Metrics of KNN k=4")
Model.Accuracy(prediction.knn.train$pred, min_max_training_set$pm)
```

6.2 Testing error

In [ ]:
```
y_pred_1 = predict(lm.model.reduced.no_outliers, test_set_new)
paste("Linear regression model")
Model.Accuracy(y_pred_1,test_set_new$pm)
```

In [ ]:
```
paste("Metrics of KNN k=4")
Model.Accuracy(prediction.knn.test$pred, min_max_testing_set$pm)
```

7. Results and discussion

7.1 Why we reject KNN model ?

KNN although is easy to intrepret, does not fits well for the current requirement. As we have seen from the above plots that Cross validation suggests K= 4 as a good value for KNN. But this K=4 KNN model fails to give good predictions on the test set for 'pm' > 0.5.

Then we go take ahead our analysis to find a better K value that fits well to both the training and the test set. In doing so we can see from the graph of K values from [1,200] that k values do not converge for training and test set. And there is not trade-off between the two. Thus to keep finfing a good k value becomes another problem, which is expensive in computation and requires different approaches like division of datapoints into intervals and evaluating the performance, which is beyond the scope of our current study and the EDA done for the task does not support as of now.

7.2 Why we choose Linear Regression model ?

The Linear regression model performs well on both the training and testing set given to us for this task. As seen from the EDA there exists interactions amongst features, which is accounted by Linear Regression model. It has a good Adj R-Squared value than KNN model and is thus able to explain more variance of the target feature. Other reason to choose Linear Regression is that we can perform features selection. This greatly reduces the feature space and makes the model light weight whicj is one of the requirement.

The chosen model has the highest Adj-R Squared of 0.607 amongst all the Linear models created. Which means it can explain 60.1% of the variance of the target variable. The F-statistics is less than 2e-16 which states that the model is better than a null model.

The final chosen model has reduced feature space, having only 6 features of the total 12 features (including the additional 4 features). This model also takes into account the interactions between features whcih was lacking in KNN.

The coefficients of the Linear model are comparable. We have been able to remove the high correlation which was introduced due to increasing features we introduced.

Final Linear Regression formula that we obtain is as follows:

pm = 0.30892*ambient + 0.19566*coolant - 0.40500*u_d - 5.91057*u_q + 7.95350*motor_speed + 3.89440*i_d - 0.23224*u_q* motor_speed + 3.53836*motor_speed* i_d + 2.72639

8. Task B : Inference task

Intrepreting the final chosen model

In [ ]:
```
summary(lm.model.reduced.no_outliers)
```

- Factors that have strong affect on the rotor temperature:
  1. 'ambient'
  2. 'coolant'

3. 'u_d'
4. 'u_q'
5. 'motor_speed'
6. 'i_d'

- Interactions affecting the rotor temperature:
  1. u_q:motor_speed
  2. motor_speed:i_d

- Interpretting the coefficients of the Linear Regression model:
  - Positive coefficients indicate that as independent variable increases the dependent variable 'pm' aslo increases.
  - Negative coefficients indicate that as independent variable increases the dependent variable 'pm' aslo decreases.
  - The value of the coefficient indicate the importance of that feature on the depenednt variable.
  - Looking at the magnitude of the coefficients, motor_speed is the most important feature to predict the dependent feature 'pm'.
  - And 'coolant' which has the smallest coefficient is the least important feature amongst the features.
  - 1 unit increase in motor_speed increases the 'pm' value by 7.95350 provided all other variables are kept constant.Same principle applies to all the variables.
  - The effect of 'motor_speed' on 'pm' is now 7.95350 + 3.53536*i_d. When u_q=1, the effect of 'motor_speed' is 7.95350+3.53536 = 11.4888; meaning an increase of 11.4888 in 'pm' when 'u_q' = 1 for an increase in 1 unit of 'motor_speed'.

In [ ]:
```
cor(training_set[-c(4162, 4172, 4192, 4201, 4221, 1471),
        -c(6, 8, 10, 11, 12)])
```

- Why motor_speed contribues the most to 'pm'
  - The magnitude of coeffiencient of motor_speed is large which contributes more to the 'pm'. This large value of coefficient is because of high positive correlation between motor_speed and u_q; also high negative correlation between motor_speed and i_q.

This makes motor_speed more important because of its intearction with other 2 features.

- Why the coefficients of coolant, ambient and u_d is small ?
    - The small coefficient of these features is due to the fact that they do not have a high correlation amonst other features and also a low correlation with the target variable 'pm'.
- Why the coefficients of motor_speed, u_q, i_d is larger than other ?
    - This large coefficinets of the 3 features is due to the correlation between them. Other features are not highly correlated and thus have a small value.

9. Conclusion

We have successfully compared two different Machine Learning model i.e Linear Regression model and KNN. We have been able to deliver a light weight model which was the main aim of the assessment with good accuracy as compared to KNN. Both the models and their development process was supported with logical steps and approaches follwed with garphs and statistics.

By developing a regression model to predict the 'pm' rotor temperature using real world dataset, we are able to critically analyze different Machine learning algorithms that are appropriate to solve the given problem whilst keeping in mind the pros and cons of the approaches, and evaluating results supporting with statistics and plots when and whereever possible. To understand the requiremnts and deliver the results with limitations, have developed transparency in taking any study at hand. Also, the skills and fundamantal knowledge needed to learn new algorithms and apporaches have been successfully imbibed with the implementation of the given task.

10. References

- Liu, C. (2021). Data Transformation: Standardization vs Normalization - KDnuggets. Retrieved 23 April 2021, from https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html

- Team, T., & Team, T. (2021). How, When, and Why Should You Normalize / Standardize / Rescale Your Data?. Retrieved 23 April 2021, from https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff
- In-phase and quadrature components - Wikipedia. (2021). Retrieved 23 April 2021, from https://en.wikipedia.org/wiki/In-phase_and_quadrature_components
- Compute three-phase instantaneous active and reactivepowers - Simulink- MathWorks Australia. (2021). Retrieved 23 April 2021, from https://au.mathworks.com/help/physmod/sps/powersys/ref/powerdq0instantaneous.html
- What is a Synchronous Motor? - Definition, Construction, Working & its Features - Circuit Globe. (2021). Retrieved 23 April 2021, from https://circuitglobe.com/synchronous-motor.html
- RSS Vs TSS Vs R-square - Dataunbox. (2021). Retrieved 23 April 2021, from https://dataunbox.com/rss-vs-tss-vs-r-square/
- AC - Active, Reactive and Apparent Power. (2021). Retrieved 23 April 2021, from https://www.engineeringtoolbox.com/kva-reactive-d_886.html
- Pros and Cons of K-Nearest Neighbors - From The GENESIS. (2021). Retrieved 23 April 2021, from https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/
- Linear regression - Wikipedia. (2021). Retrieved 23 April 2021, from https://en.wikipedia.org/wiki/Linear_regression