Anomaly Detection

Paolo De Piante

2019-11-03

1. Introduction

This is a report on Anomaly Detection and Conditional Monitoring. The dataset can be downloaded from http://data-acoustics.com/measurements/bearing-faults/bearing-4/. This is a NASA.gov website. I used this open data to reproduce a very similar scenario with which I am figuring out in a Big Manufactoring Company. It is going to implement a project in the domain of Predictive Maintenance. The real data are not available (no disclosure agreement was released) for this analysis but the data set I found is very similar to real scenario. NASA.gov website requires to download a huge file named "IMS.7z". It contains several .txt files. Each one of them contains several vibration signal rows recorded at specific intervals. The sampling rate used was set at 20 kHz. The file name shows when the data was collected. So, I elaborated .txt files to merge them in a single .csv file operating a little resampling of data. The final data frame has 6324 rows and 5 columns (TimeStamp and four Bearings Vibration). The goal of this analysis is to demostrate how the Machine Learning algorithms can detect engine feilures in advance. This study can be applied to any kind of engine using special sensors named "Accelerometer" (in our scenario four of them) oriented toward the engine to catch its vibrations. The .csv file can be downloaded from https://raw.githubusercontent.com/PaoloDePiante/Anomaly_Detection/master/Dataset_for_Bearing.csv. I used GitHub to give access to the pre-processed NASA.dov data set. In this way, I used it as a data set repository too. The study has been inspired by HarvardX & edX "Data Science: Capstone" MOOC. Many Thanks to Professor Rafael Irizarry for his Data Science Certification Program provided by edX.

2. Analysis

2.1 Data Preparation

I used the following libraries:

```r
library(dplyr)
library(tidyverse)
library(caret)
library(ggpubr)
library(StatMatch)
library(scales)
```

After the data set is downloaded, basically two main data partitions have been defined: "Train Set" to train the model on the "Normal" running state of the Engine; "Test Set" to isolate the anomaly and test the model. The TimeStamp has been managed by "POSIXct" function. After that the Training Data have been plotted. Based on that I decided to normalize them using the min and max scalar method. The Formula I used is the following:

$$X' = a + \frac{(x - x_{min})(b - a)}{(x_{max} - x_{min})}$$

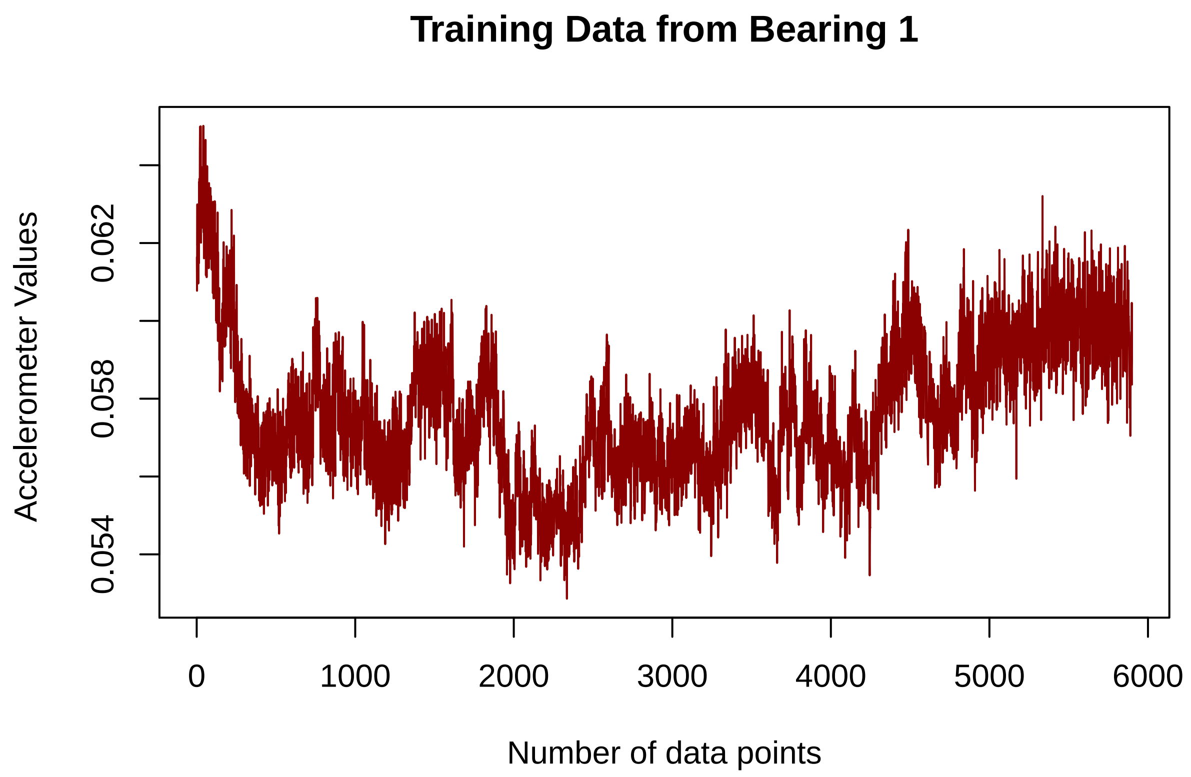
I am going to normalize data in the range [0.1] so a = 0 and b = 1. This is done both for the Train and the Test set.

```r
# Downlod Data set from GitHub

dat <- read.csv(url(
"https://raw.githubusercontent.com/PaoloDePiante/Anomaly_Detection/
master/Dataset_for_Bearing.csv"), sep=",", header = TRUE)
```

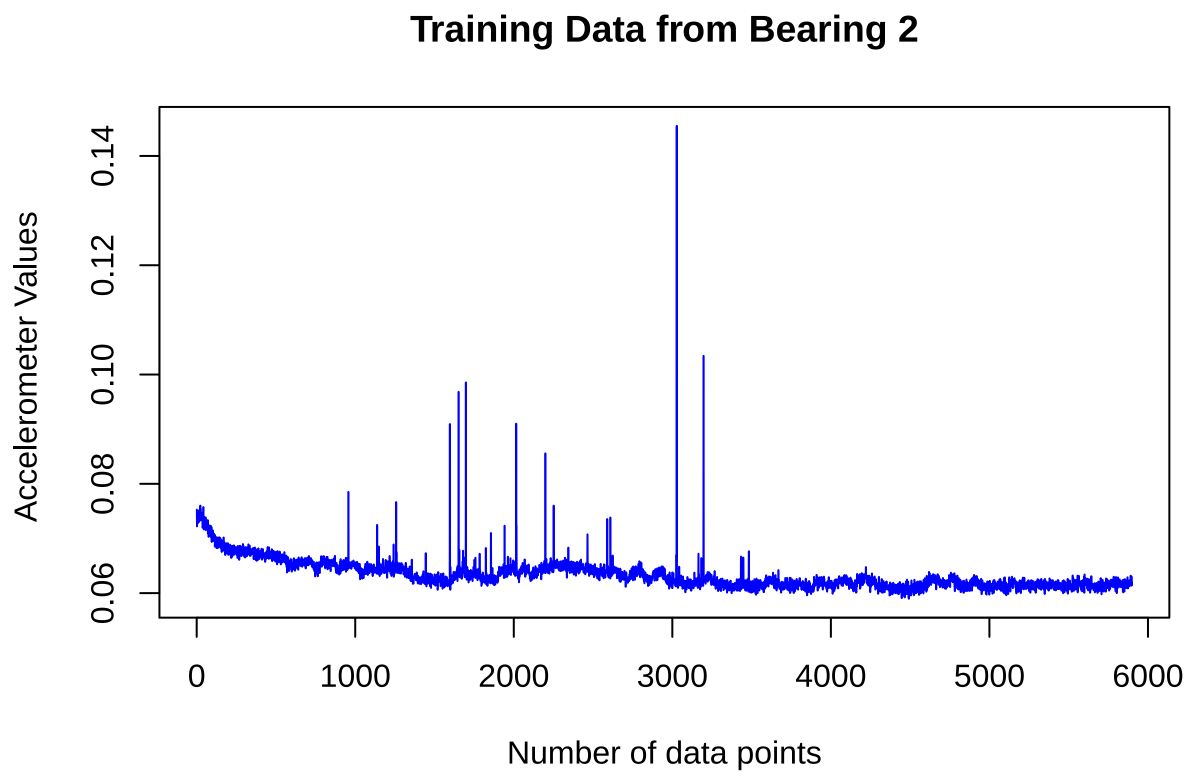
```r
# Split data in "train set" for normal behaviour and "test set" for a failure
head(dat)
```

```
##                    TS  Bearing.1  Bearing.2  Bearing.3  Bearing.4
## 1 2004-03-04 09:27:46 0.06163001 0.07527833 0.05199087 0.04347458
## 2 2004-03-04 09:32:46 0.06077416 0.07220577 0.05223868 0.04203936
## 3 2004-03-04 09:42:46 0.06219483 0.07288554 0.05480937 0.04135988
## 4 2004-03-04 09:52:46 0.06299438 0.07343327 0.05426110 0.04312637
## 5 2004-03-04 10:02:46 0.06194531 0.07300610 0.05323653 0.04261702
## 6 2004-03-04 10:12:46 0.06223536 0.07331569 0.05501787 0.04203609
```

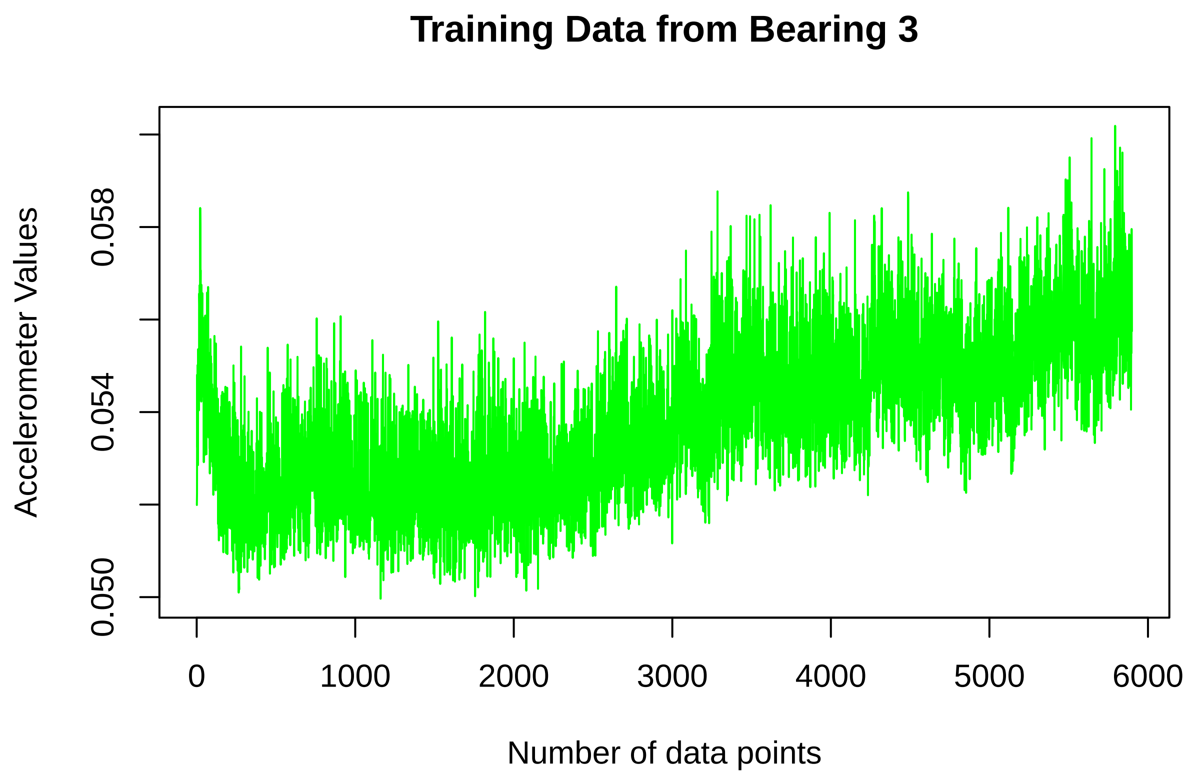
```r
set.seed(2)

train_set <- dat[which((as.POSIXct(dat$TS, origin="1970-01-01", tz="UTC") >=
                as.POSIXct('2004-03-04 09:27:46', origin="1970-01-01", tz="UTC"))
                & (as.POSIXct(dat$TS, origin="1970-01-01", tz="UTC") <=
                as.POSIXct('2004-04-14 20:51:57', origin="1970-01-01", tz="UTC"))),]

test_set <- dat[which(as.POSIXct(dat$TS, origin="1970-01-01", tz="UTC") >=
                as.POSIXct('2004-04-14 20:51:01', origin="1970-01-01", tz="UTC")),]

# Looking to Training Data for all Bearings
# par(mfrow=c(4,1))
plot(train_set$Bearing.1, type="l", col="dark red", main="Training Data from Bearing 1",
     ylab="Accelerometer Values ", xlab = "Number of data points")
```



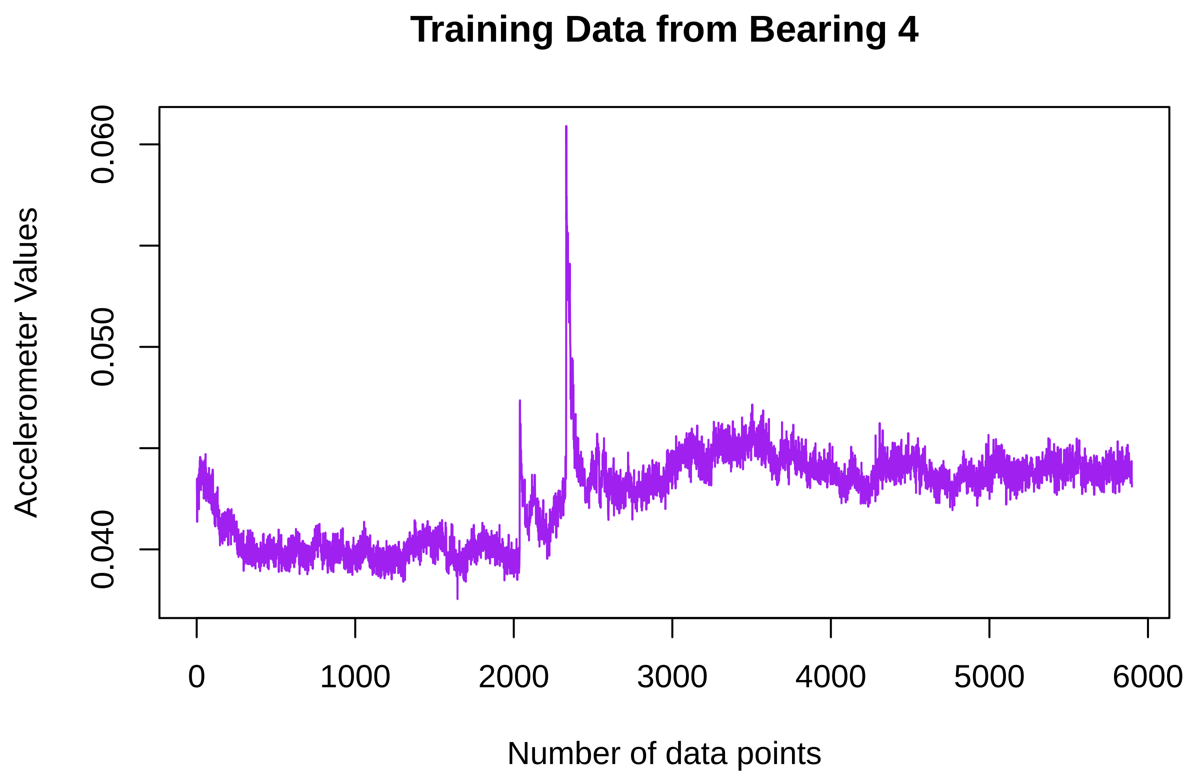
```r
plot(train_set$Bearing.2, type="l", col="blue", main="Training Data from Bearing 2",
     ylab="Accelerometer Values ", xlab = "Number of data points")
```



```
plot(train_set$Bearing.3, type="l", col="green", main="Training Data from Bearing 3",
     ylab="Accelerometer Values", xlab = "Number of data points")
```



```r
plot(train_set$Bearing.4, type="l", col="purple", main="Training Data from Bearing 4",
     ylab="Accelerometer Values", xlab = "Number of data points")
```



```
# Normalize Data using Min Max Scaler to be in the range of [0,1]
min_max_train <- c(min(train_set$Bearing.1),max(train_set$Bearing.1),
                   min(train_set$Bearing.2),max(train_set$Bearing.2),
                   min(train_set$Bearing.3),max(train_set$Bearing.3),
                   min(train_set$Bearing.4),max(train_set$Bearing.4))

Z_train_set <- train_set %>% mutate(Bearing.1 =
((train_set$Bearing.1-min_max_train[1])/(min_max_train[2]-min_max_train[1])),
                                    Bearing.2 =
  ((train_set$Bearing.2-min_max_train[3])/(min_max_train[4]-min_max_train[3])),
                                    Bearing.3 =
  ((train_set$Bearing.3-min_max_train[5])/(min_max_train[6]-min_max_train[5])),
                                    Bearing.4 =
  ((train_set$Bearing.4-min_max_train[7])/(min_max_train[8]-min_max_train[7])))

Z_test_set <- test_set %>% mutate(Bearing.1 =
((test_set$Bearing.1-min_max_train[1])/(min_max_train[2]-min_max_train[1])),
                                  Bearing.2 =
  ((test_set$Bearing.2-min_max_train[3])/(min_max_train[4]-min_max_train[3])),
                                  Bearing.3 =
  ((test_set$Bearing.3-min_max_train[5])/(min_max_train[6]-min_max_train[5])),
                                  Bearing.4 =
  ((test_set$Bearing.4-min_max_train[7])/(min_max_train[8]-min_max_train[7])))
```

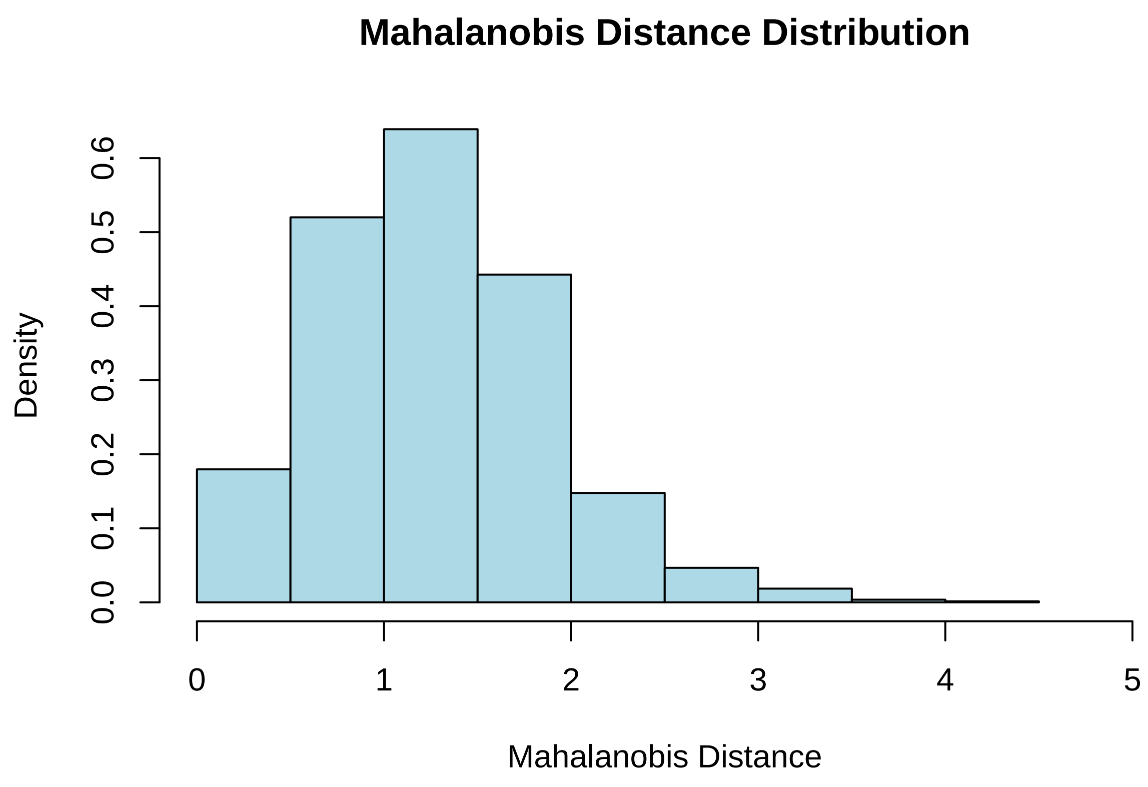
2.2 Building the Model and Data Processing

Based on the nature of the data set we could deal with high dimensional sensors data so it is always suggested to try to reduce the complexity to only two principal components starting from training data normalized. Once I obtained the two main components, I calculated the prediction using normalized test data. After that, I calculated three main values for the two Train Principal Components: the covariance matrix (I decided to use it because they have the same unit of measure); the inverse of covariance matrix; the columns average. Using these values I calculated the Mahalanobis distance both for the Test and the Train PCA data frame. The Mahalanobis distance can also be defined as a dissimilarity measure between an observation $\vec{x} = (x_1, x_2, x_3, \ldots, x_N)^T$ and a set of observations, with mean $\vec{\mu} = (\mu_1, \mu_2, \mu_3, \ldots, \mu_N)^T$ and covariance matrix S. The formula is:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \mu)^T S^{-1} (\vec{x} - \mu)}$$

The final step of this section is related to the Threshold calculation. This Threshold permits us to identify outliers to identify anomalies. From the graph it appears quite clear that if we multiply by 3 σ the central value we could likely identify an anomaly behaviour. The Threshold is equal to 3.826834.

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

# Reduce to two principal components the sensor readings for training data normalized
Train.pca <- prcomp(Z_train_set[2:5])
# Predict principal components based on Training ones using test set normalized
Test.pca <- predict(Train.pca, newdata = Z_test_set[2:5])

# Convert Matrix in Data Frame just for the first two principal components
Test.pca <- as.data.frame(Test.pca[,1:2])
Train.pca <- (Train.pca$x[,1:2])
Train.pca <- as.data.frame(Train.pca[,1:2])

# Since the unit of measure is the same the matrix of covariance can be used.
covariance_matrix <- cov(Train.pca)
inv_covariance_matrix <- cov(Train.pca)^-1
mean_distr <- colMeans(Train.pca)

# Mahalanobis distance calculation both for Train and Test Set
dist_train <- sqrt(mahalanobis(Train.pca, mean_distr, covariance_matrix, inverted=FALSE))
dist_test <- sqrt(mahalanobis(Test.pca, mean_distr, covariance_matrix, inverted = FALSE))

# Threshold calculation to identify outliers (anomalies) based on training data
MD_threshold <- function(dist, extreme){
  if (extreme) {k=3}
  else {k=2}
  threshold = mean(dist) * k
  return(threshold)
}


# The Hist to set the threshold to say when could be identified an anomaly
d <- dist_train
hist(d, include.lowest = TRUE, right = TRUE, freq = FALSE,
     angle = 45, col = "LightBlue",
     axes = TRUE, plot = TRUE, labels = FALSE,
     xlim = range(0,5), xlab ="Mahalanobis Distance",
     main = "Mahalanobis Distance Distribution")
```



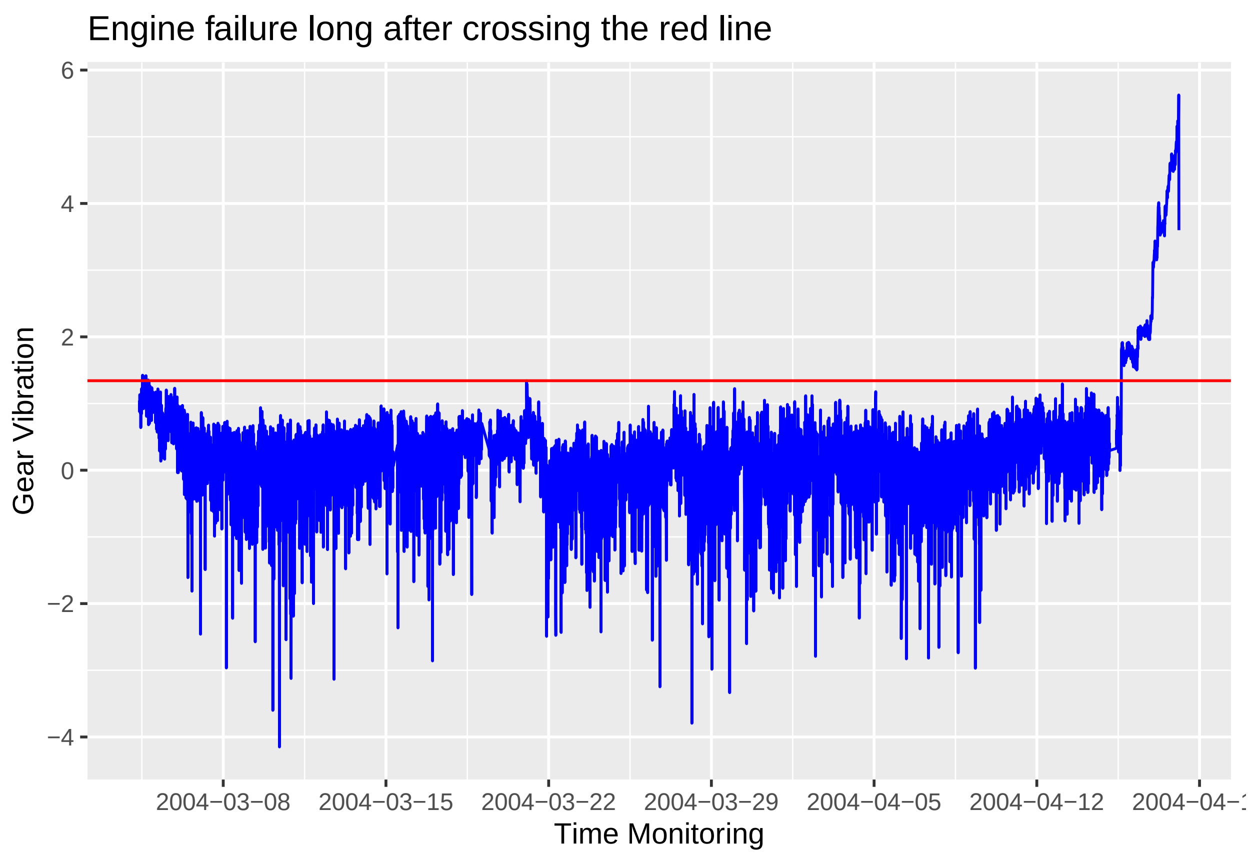
```r
# 3 sigma from mean distance has been used for upper threshold
threshold = MD_threshold(dist_train, extreme = TRUE)

# Threshold value
threshold
```

```
## [1] 3.826834
```

Now, the scope is to visualize data for the entire period combining Train and Test data sets once again. The Graph shows the data sensors over the entire time period. It is possible to note that the blue line remains under the red line (the threshold) until an anomaly arose. Already in some points the blue line touched the red one but just for a while. Differently, when the engine had a failure the blue line went decisively over the threshold.

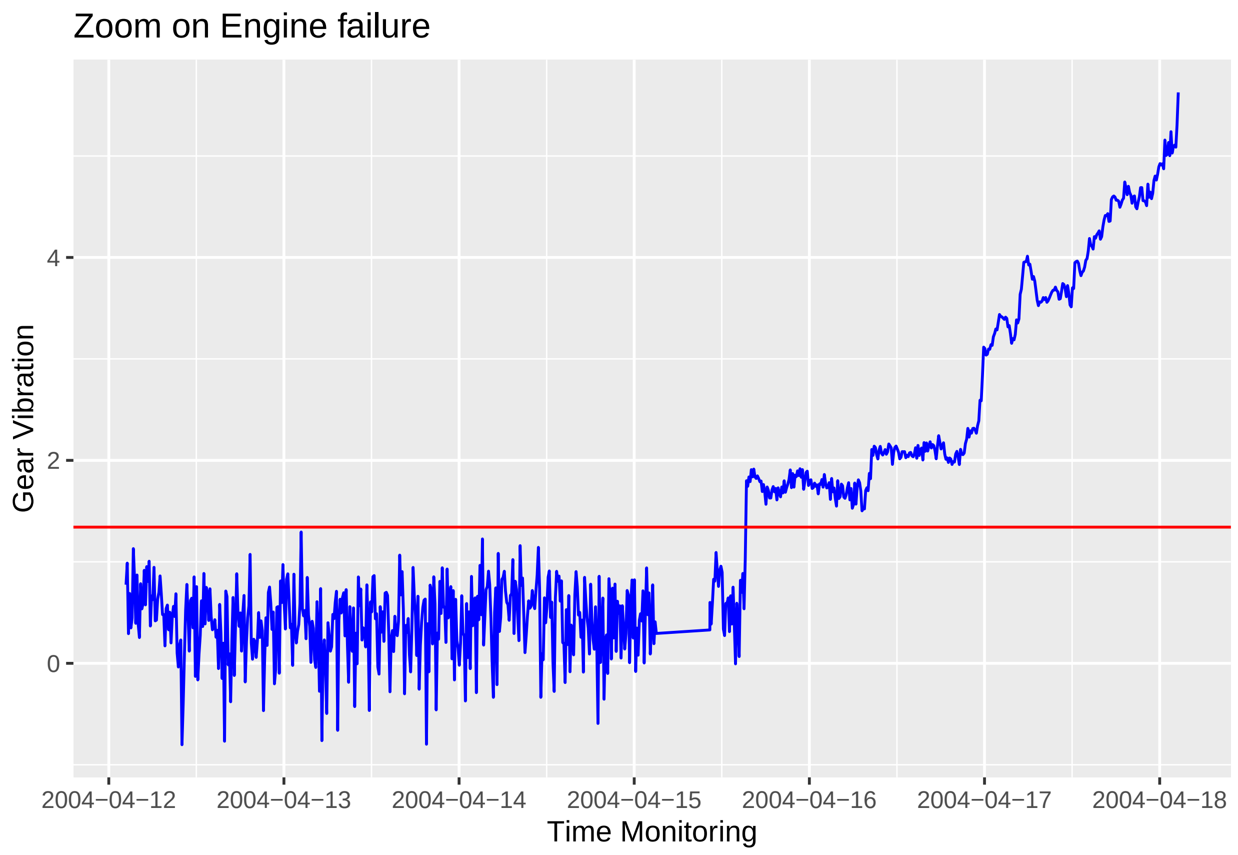
```r
# Show the entire period observed joining Trainind and Testing Data
v_dist_train <-as.data.frame(dist_train)
v_dist_test <-as.data.frame(dist_test)
colnames(v_dist_train) <- "MD"
colnames(v_dist_test) <- "MD"
Train_Test <- rbind(v_dist_train, v_dist_test)
```

```r
v_z_train_set <-as.data.frame(Z_train_set[1])
v_z_test_set <-as.data.frame(Z_test_set[1])
colnames(v_z_train_set) <- "Time"
colnames(v_z_test_set) <- "Time"
Ind_Train_Test <- rbind(v_z_train_set, v_z_test_set)

Train_Test <- log(Train_Test)
Anomaly <- as.data.frame(cbind(Ind_Train_Test,Train_Test))

df <- Anomaly %>% mutate(Time = as.POSIXct(Anomaly$Time, origin="1970-01-01", tz="UTC"))
p <- ggplot(df,aes(x=df$Time, y=df$MD))
p$labels$y <- "Gear Vibration"
p + scale_x_datetime(breaks = date_breaks("1 week"), name = "Time Monitoring") +
  geom_line(col = "Blue") +
  geom_hline(yintercept=log(threshold), col = "Red") +
  ggtitle("Engine failure long after crossing the red line")
```



3. Results

If we zoom insde the Graph in the last period just before the fault, we notice that we have been able to predict the engine failure two days and half in advance. Obviously, the accuracy of the predictions depends on the engine but this approach demostrates that the model built can alert the supervisor in advance, warning him to perform a preventive maintenance long before the engine stops, to lessen the likelihood of failure and breakdown of the production machine.



4. Conclusions

Abundent documentation on this topic is available on the net. Anomaly Detection and Condition Monitoring can be implemented by a Deep Learning approach too. Most of the times it seems that good results comes without the need of implementing sofisticated algorithms. This is an example. Using tools like a PCA and a Mahalanobis distance it seems that the model is good enough to save money and time. Based on my real data I obtained similar results, which were very appreciated by the Customer.

Special thanks to Professor Rafael Irizarry and to the edX staff.

Sincerely,

Paolo De Piante