Predicting Devices Failure

Dalila Benachenhou (d_b@femvestor.com)

05/03/2019

Introduction:

3D Technology has 1169 of devices transmitting daily aggregate telemetry attributes. At this moment, the company performs maintenance only when necessary, but would like to determine the condition of in-service equipment to predict when maintenance should be performed.

To help 3D Technology in its endeavor, we will first take the time to explore the dataset, and then build the appropriate models.

Exploration:

3D Technology provided little background about their devices. Do device that fail are fixed and put back in production? What do each attribute measure? Hence, One has to take the time to explore them before deciding on the appropriate Machine Learning model.

```
library(tidyverse)
library(quantmod)
library(GGally)
library(caret)
library(plyr)
library(cowplot)
library(reshape2)
device_failure <- read.csv("/Users/dalila/Documents/Job Application/Ama
zon AWS Project/AmazonApril19/device_failure.csv")
```

From here, one can deduce that except for date, and devices, all the other variables, even the failure signal, have been saved as numerical variables. one can also see that there are 1169 different devices. As for the response variable, failure, takes only 2 values 0—working—and 1--failure, one will have to change its type to a categorical variable to be able to build a classifier.

```
#Check variables type
str(device_failure)
```

```
## 'data.frame':    124494 obs. of  12 variables:
##  $ date      : Factor w/ 304 levels "2015-01-01","2015-01-02",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ device    : Factor w/ 1169 levels "S1F01085","S1F013BB",..: 1 3 4
5 6 7 8 9 10 11 ...
##  $ failure   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ attribute1: int  215630672 61370680 173295968 79694024 135970480
68837488 227721632 141503600 8217840 116440096 ...
##  $ attribute2: int  56 0 0 0 0 0 0 0 0 0 ...
##  $ attribute3: int  0 3 0 0 0 0 0 0 1 323 ...
##  $ attribute4: int  52 0 0 0 0 41 0 1 0 9 ...
##  $ attribute5: int  6 6 12 6 15 6 8 19 14 9 ...
##  $ attribute6: int  407438 403174 237394 410186 313173 413535 402525
494462 311869 407905 ...
##  $ attribute7: int  0 0 0 0 0 0 0 16 0 0 ...
##  $ attribute8: int  0 0 0 0 0 0 0 16 0 0 ...
##  $ attribute9: int  7 0 0 0 3 1 0 3 0 164 ...
```

```
#Change failure to factor and give the levels proper names
device_failure$failure <- factor(device_failure$failure)
device_failure$failure <- mapvalues(device_failure$failure, from = c("0
", "1"), to = c("no", "yes"))
device_failure$failure <- relevel(device_failure$failure, "no")
#Change date to date format
device_failure$date2 <- as.Date(device_failure$date, "%Y-%m-%d")
```

From the summary statistics, one can see the number of failure is extremely small. Does it mean when a device fails it is removed? Attributes 2,3,4,7, 8 and 9 have medians equal 0, in spite of having all their means larger than 0—implying high skeweness. Finally, attributes 7 and 8 are identical—implying one can just remove attribute8. Finally, devices are monitored continuously from January 1st, 2015 to November 2nd, 2015.

```
summary(device_failure)
```

```
##          date              device          failure        attribute1     
##  2015-01-01:  1163   S1F0E9EP:   304   no :124388   Min.   :        0  
##  2015-01-02:  1163   S1F0EGMT:   304   yes:   106   1st Qu.: 6128472  
##  2015-01-03:  1163   S1F0FGBQ:   304                Median :12279738  
##  2015-01-04:  1162   S1F0FP0C:   304                Mean   :12238813  
##  2015-01-05:  1161   S1F0GCED:   304                3rd Qu.:18330960  
##  2015-01-06:  1054   S1F0GGPP:   304                Max.   :24414040  
##  (Other)   :117628   (Other) :122670                                  
##    attribute2          attribute3          attribute4          attribut5
##  Min.   :     0.0   Min.   :     0.00   Min.   :     0.000   Min.   : 100
##  1st Qu.:     0.0   1st Qu.:     0.00   1st Qu.:     0.000   1st Qu.: 8.
```

```
00
##   Median :  0.0   Median :    0.00   Median :   0.000   Median :10.00
##   Mean   :159.5   Mean   :    9.94   Mean   :   1.741   Mean   :14.22
##   3rd Qu.:  0.0   3rd Qu.:    0.00   3rd Qu.:   0.000   3rd Qu.:12.00
##   Max.   :64968.0   Max.   :24929.00   Max.   :1666.000   Max.   :98
##
##    attribute6       attribute7         attribute8         attribute9
##   Min.   : 8   Min.   :  0.0000   Min.   :  0.0000   Min.   :    0.00
##   1st Qu.:221452   1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu  0.00
##   Median :249800   Median :  0.0000   Median :  0.0000   Median :0.00
##   Mean   :260173   Mean   :  0.2925   Mean   :  0.2925   Mean   :12.45
##   3rd Qu.:310266   3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:0.00
##   Max.   :689161   Max.   :832.0000   Max.   :832.0000   Max :18701.
##
##      date2
##   Min.   :2015-01-01
##   1st Qu.:2015-02-09
##   Median :2015-03-27
##   Mean   :2015-04-16
##   3rd Qu.:2015-06-17
##   Max.   :2015-11-02
##
```

One can see that attribute7 and attribute8 are equivalent. So remove attribute8

```
keepNames <- setdiff(names(device_failure),c("attribute8"))
device_failure <- device_failure[,keepNames]
```

Still, is there a difference between working and failed devices? To answer this question, one can perform summary statistics. Medians for attributes 2,3, 5, 7, and 9 for both working and failed devices are the same, but the means using same for working and failed devices are very different, especially for attributes 2, 4, 7, 9. In this case, the means are much larger for failed devices than working devices. For attribute 3, the mean is lower for failed devices than for working devices. Except for attribute 1,5, and 9, the variances are very different between working and failed devices. All this implies that Machine Learning model can be appropriate.

```
t(device_failure %>% group_by(failure) %>% summarise_if(is.numeric,c(Me
an = mean,Median = median,Var = var,Min = min,Max =max)))

##                    [,1]           [,2]
## failure            "no"           "yes"
## attribute1_Mean    "122384024"    "127175527"
## attribute2_Mean    " 156.1187"    "4109.4340"
## attribute3_Mean    "9.945598"     "3.905660"
## attribute4_Mean    " 1.696048"    "54.632075"
## attribute5_Mean    "14.22161"     "15.46226"
## attribute6_Mean    "260174.3"     "258303.5"
## attribute7_Mean    " 0.2666817"   "30.6226415"
## attribute9_Mean    "12.44246"     "23.08491"
```

```
## attribute1_Median "122786072"    "139117254"
## attribute2_Median "0"            "0"
## attribute3_Median "0"            "0"
## attribute4_Median "0.0"          "1.5"
## attribute5_Median "10"           "10"
## attribute6_Median "249794.0"     "267648.5"
## attribute7_Median "0"            "0"
## attribute9_Median "0"            "0"
## attribute1_Var    "4.964663e+15" "4.816591e+15"
## attribute2_Var    "  4603265"    "163935943"
## attribute3_Var    "34530.5974"   "  995.8577"
## attribute4_Var    "  491.2569"   "37439.3586"
## attribute5_Var    "254.1914"     "241.7176"
## attribute6_Var    " 9830294793"  "10681079587"
## attribute7_Var    "   43.0083"   "13696.8658"
## attribute9_Var    "36655.02"     "23546.90"
## attribute1_Min    "        0"    "4527376"
## attribute2_Min    "0"            "0"
## attribute3_Min    "0"            "0"
## attribute4_Min    "0"            "0"
## attribute5_Min    "1"            "3"
## attribute6_Min    " 8"           "24"
## attribute7_Min    "0"            "0"
## attribute9_Min    "0"            "0"
## attribute1_Max    "244140480"    "243261216"
## attribute2_Max    "64968"        "64784"
## attribute3_Max    "24929"        "  318"
## attribute4_Max    "1666"         "1666"
## attribute5_Max    "98"           "91"
## attribute6_Max    "689161"       "574599"
## attribute7_Max    "832"          "832"
## attribute9_Max    "18701"        " 1165"
```

```
Due to the high skewness, I have decided to investigate the number of z
eros in the following variables: attributes 2, 3, 4, and7.  In other pr
oblems I worked with, I just removed variables with few variability in
them, but in this case, I still need to investigate a little bit longer
before removing them.
```

```
l <- dim(device_failure)[1]
dim(subset(device_failure,attribute2 == 0))[1]/l
## [1] 0.9487204
dim(subset(device_failure,attribute3 == 0))[1]/l
## [1] 0.926623
dim(subset(device_failure,attribute4 == 0))[1]/l
## [1] 0.9249924
dim(subset(device_failure,attribute7 == 0))[1]/l
## [1] 0.9882886
```

Is a device removed when it fails?

Number of failures by device. One can see that a device, when it fails it is removed.

```
detach("package:plyr", unload=TRUE)

library(dplyr)
tmp <-device_failure %>% count(device,failure)

tmp %>% group_by(failure) %>% summarise(Mean = mean(n),Median = median(
n),Min = min(n),Max = max(n))

## # A tibble: 2 x 5
##   failure  Mean Median   Min   Max
##   <fct>   <dbl>  <dbl> <dbl> <dbl>
## 1 no       106.     84     1   304
## 2 yes        1       1     1     1
```

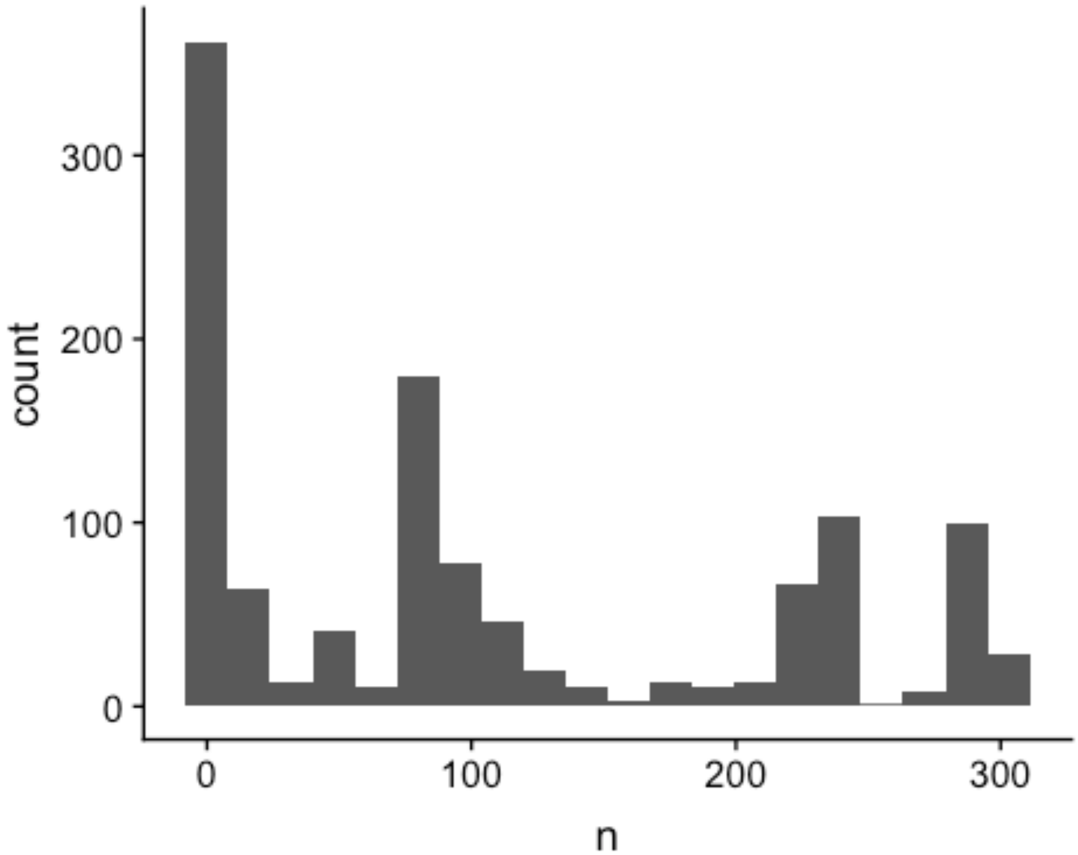
So, how long does a device on average work? At least 1 day, at most 304 days, but on average, a device will work for 106.5 days. Still a quarter of devices have been removed after 6 days.

```
library(dplyr)
deviceLife <- device_failure %>% group_by(device) %>% summarise(n = n()
)
summary(deviceLife$n)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0     6.0    84.0   106.5   224.0   304.0
```

This shows that while all devices will fail or are removed the distribution of device removed is not uniform. We have clusters of removals: high around 30, 100, 220, and over 250 numbers of days before failure. This confirm how 3 D technology perform maintenance.

```
p<- ggplot(deviceLife,aes(n))
p<- p+ geom_histogram(bins = 20)
p
```



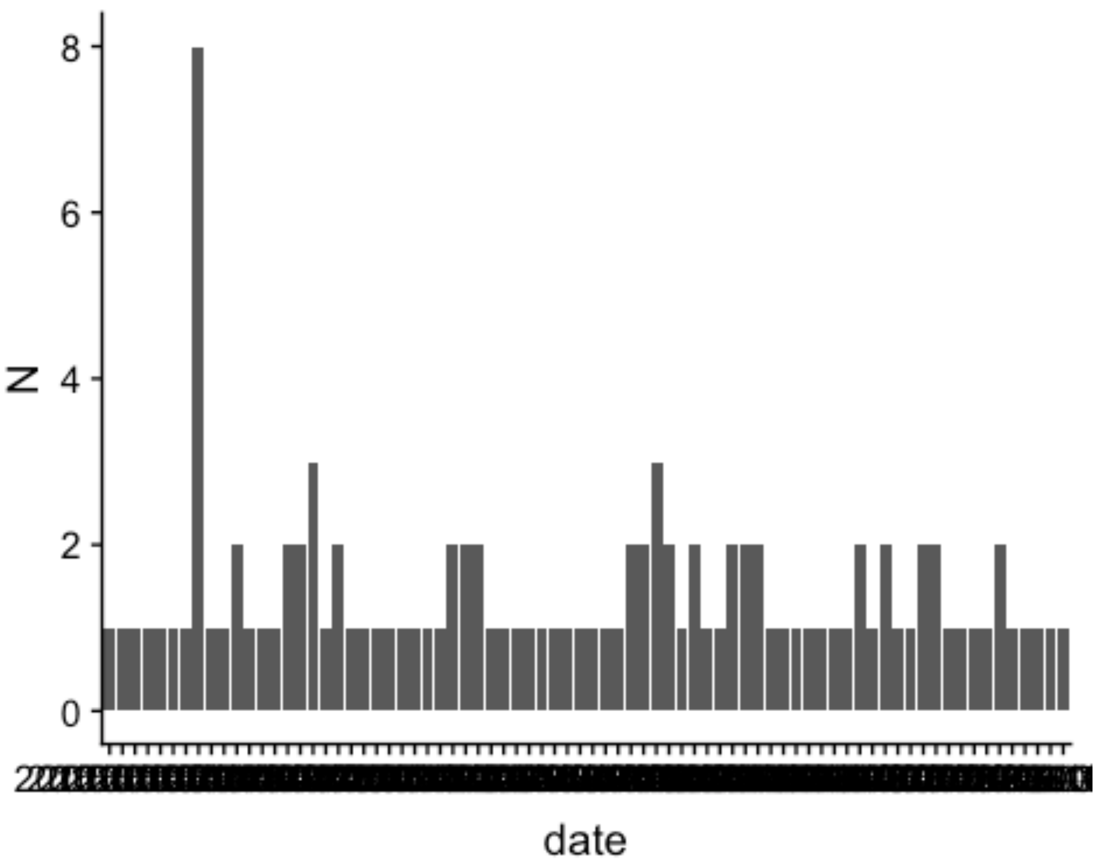
Within a day, how many devices are in:

```
nd <- device_failure %>% group_by(date2) %>% summarise(N = n())
summary(nd$N)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    31.0   261.0   350.5   409.5   672.0  1163.0
```

How many devices fail per day? From the plot, it appears not that many, at most 8 but on average 1.

```
nd <- device_failure %>% group_by(failure,date) %>% summarise(N = n())
nd <- subset(nd,failure == "yes")
ggplot(nd,aes(date,N))+geom_col()
```



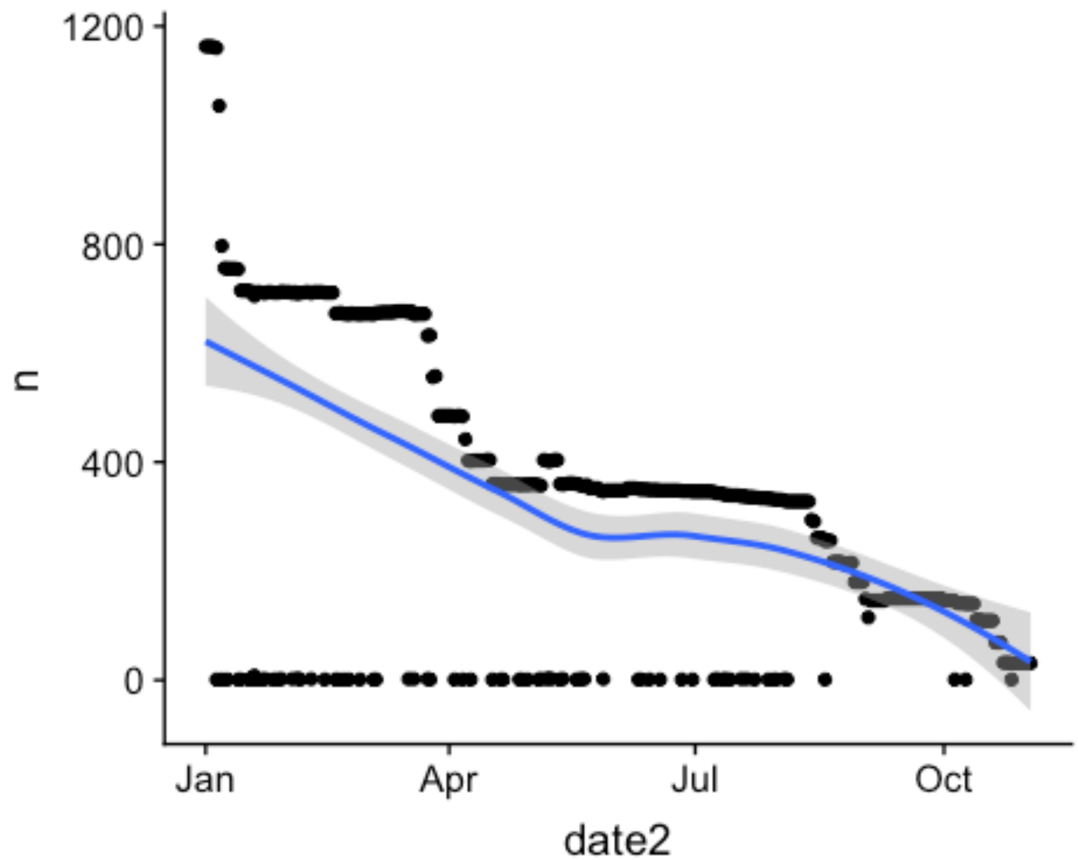

```
nd <- device_failure %>% as_tibble %>% group_by(date2,failure) %>% summ
arise(n= n())
summary(nd$n[nd$failure == "yes"])

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.395   2.000   8.000
```

This shows that we start with a large number of devices 1163 and finish with few (31 devices). As some days there are no failure, therefore, the plot has dots at 0.

```
nd <- nd[order(nd$date2),]
p<-ggplot(nd,aes(date2,n))+ geom_point()+geom_smooth()
p

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Where there devices that never failed? Yes, there were 31 devices that never failed or were removed.

```
life_devices <- device_failure %>% group_by(device)%>% summarize (N = n
())
longest_devices <- subset(life_devices,N > 300)[,1]
longest_devices <- as.data.frame(longest_devices)
longest_devices <- unique(longest_devices$device)
```

Find devices that never failed

```
df <-device_failure %>% group_by(device) %>% summarize(Count = length(u
nique(failure)),days_2_failure = n())
```

Devices that never failed. From all the devices 1169, only 106 devices failed, or 9% of devices failed. Why only these 106 failed when all the other kept working?

```
t <- table(df$Count)
t[2]/sum(t)
```

```
##          2
## 0.09067579
```

How many days to failure? From the summary of days to failure, one can see that minimum number of days to failure was 5 but longest working device to fail worked for 299. Still on average, failure happened at day 101.1 if one looks at the mean, or more precisely, 50% of failed devices worked for 92 days. What is interesting, quarter of all failed devices fail within 26.5 days, which is larger than when devices are removed. Furthermore, the median and the mean to failure are larger than the median and the mean of to removal, which are 84 and 106, respectively. These means 3D Technology are removing many devices to early.

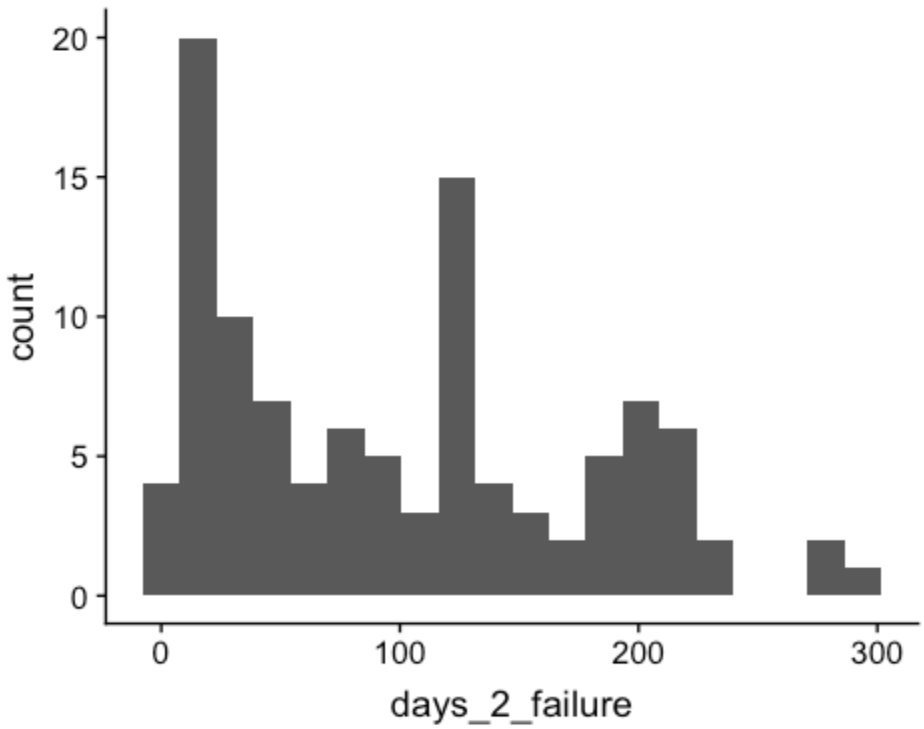
```
failed_devices <- subset(df,Count == 2)
summary(failed_devices)

##       device        Count   days_2_failure
##  S1F023H2:  1   Min.   :2   Min.   :  5.0
##  S1F03YZM:  1   1st Qu.:2   1st Qu.: 26.5
##  S1F09DZQ:  1   Median :2   Median : 92.0
##  S1F0CTDN:  1   Mean   :2   Mean   :101.1
##  S1F0DSTY:  1   3rd Qu.:2   3rd Qu.:148.0
##  S1F0F4EB:  1   Max.   :2   Max.   :299.0
##  (Other) :100

only_failed_devices <- subset(device_failure, device %in% failed_device
s$device)
```

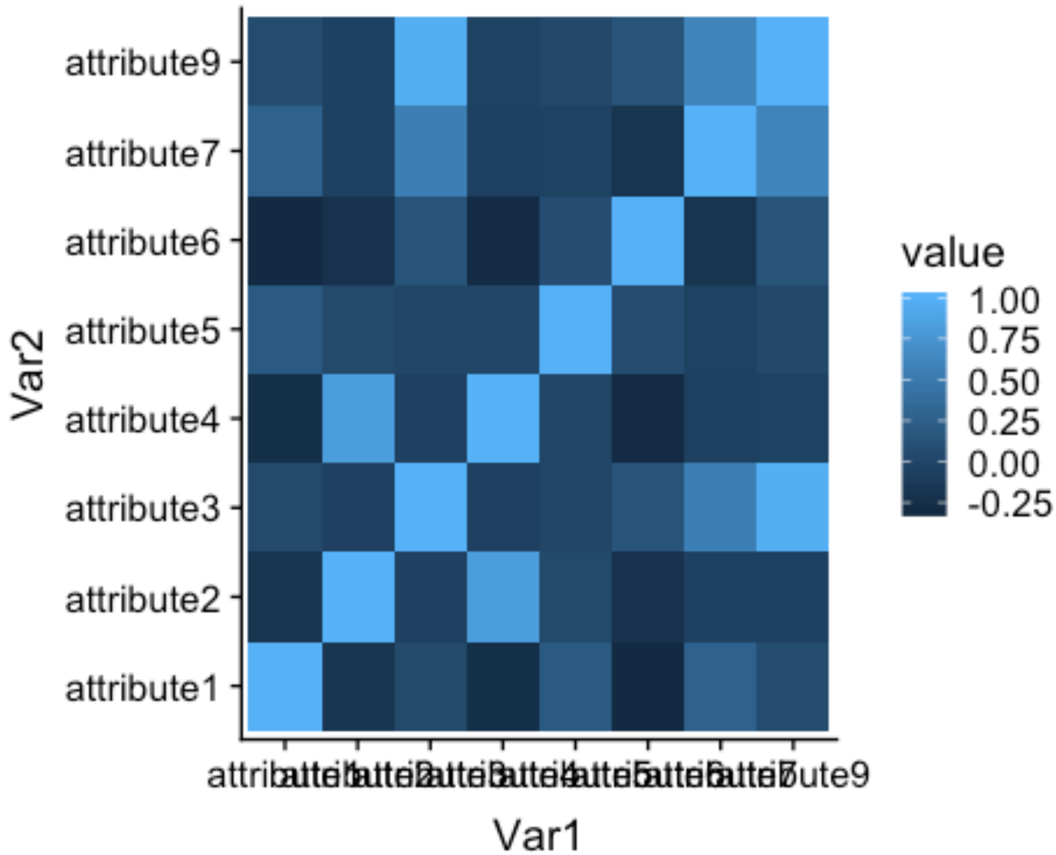
A simple histogram by number of days to failure gives a great view showing that a three decreasing picks at around 20 day, 120, and 200.

```
p<- ggplot(failed_devices,aes(days_2_failure))+geom_histogram(bins= 20)
p
```



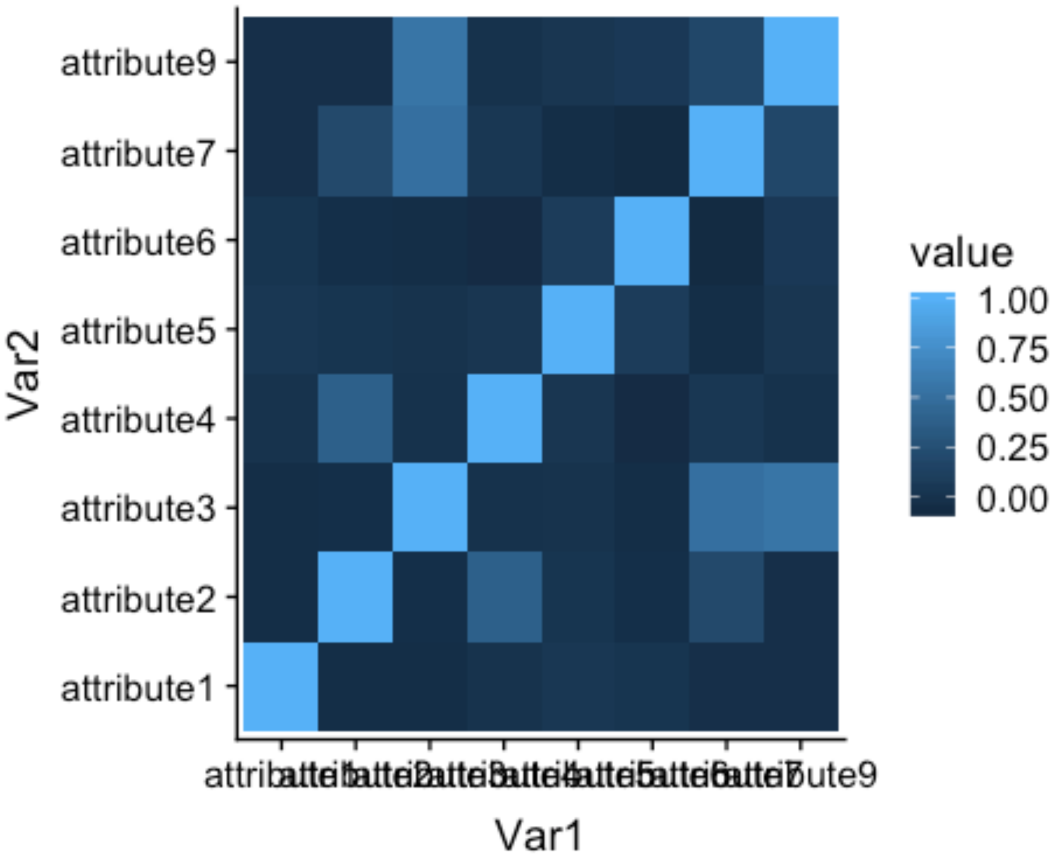
As one noticed, some attributes had mostly 0, and I was reluctant of removing them. While not shown here, I did calculate a general correlation between all the attributes, but came up with no or little correlation between the attributes. However, what if one calculates the correlation between attribute when a device fail? The heat-correlation matrix plot shows a correlation between attributes 9 and 3; attributes 9 and 8; 4 and 2; and 9 and 7. This means one cannot just dismiss these attributes as they work together to provide a signal of failure. But how can one capture such interaction?

```
s <- subset(device_failure, attribute7 > 0 & failure == "yes")
cormat <- round(cor(s[,4:11]),2)
melted_cormat <- melt(cormat)
ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()
```



When the correlation between attributes is calculated for working devices, no discerning relationship appears. This may tell one that the attributes are used to decide if a device fails.

```
s <- subset(device_failure, attribute7 > 0 & failure == "no")
cormat <- round(cor(s[,4:11]),2)
melted_cormat <- melt(cormat)
ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile()
```

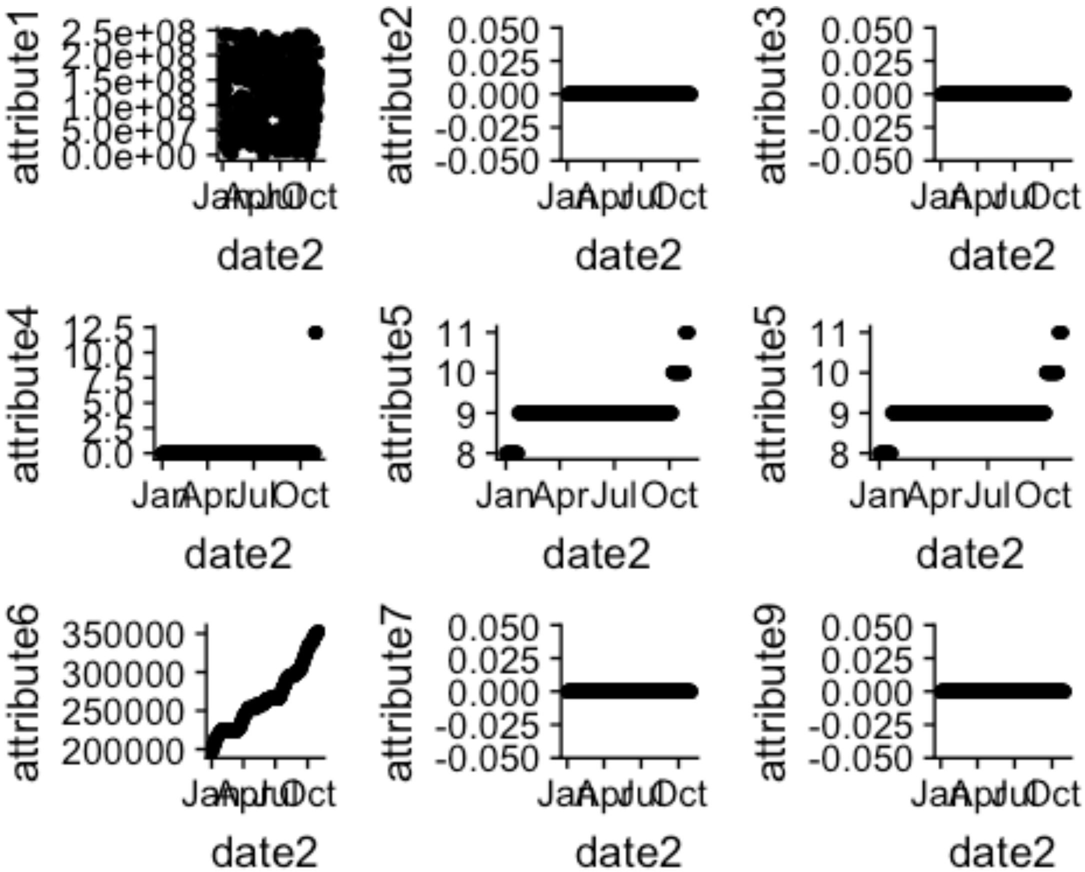


What are the characteristics of such attributes for a working and a failed device? In here, I'm presenting only one, but I have done many plots to for different devices to understand the predictors.

For a working device, attributes 5, 6 appears to be continuous time-series; attribute 1 appears random, while all the other attributes appear discrete. For a failed device, attribute 1 appears to have a pattern; attribute 5 has only 2 states, and attribute 6 appears to have reached a plateau. Furthermore, attribute 2 jumps from 0 to 1000 to 2000

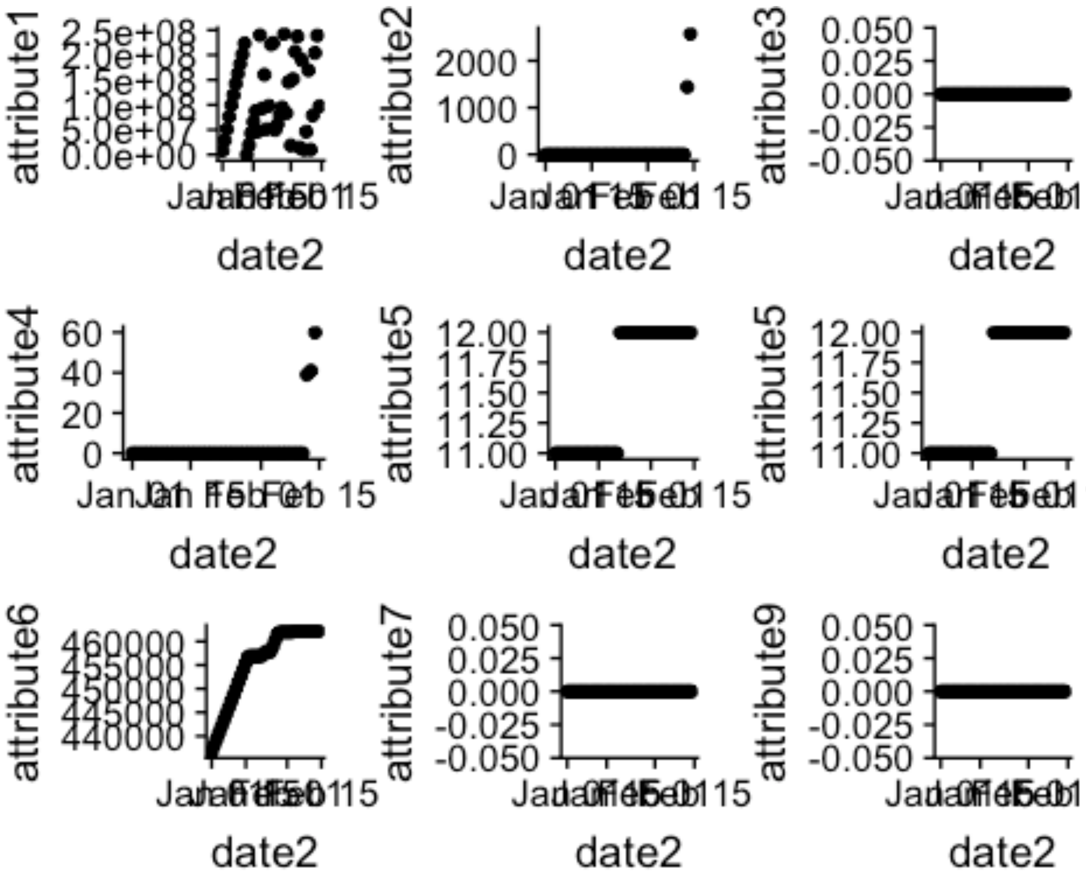
```
p1 <- subset(device_failure, device == "S1F0E9EP")
pw1 <- ggplot(p1,aes(x=date2,y=attribute1))+geom_point()
pw2 <- ggplot(p1,aes(x=date2,y=attribute2))+geom_point()
pw3 <- ggplot(p1,aes(x=date2,y=attribute3))+geom_point()
pw4 <- ggplot(p1,aes(x=date2,y=attribute4))+geom_point()
pw5 <- ggplot(p1,aes(x=date2,y=attribute5))+geom_point()
pw6 <- ggplot(p1,aes(x=date2,y=attribute6))+geom_point()
pw7 <- ggplot(p1,aes(x=date2,y=attribute7))+geom_point()
pw9 <- ggplot(p1,aes(x=date2,y=attribute9))+geom_point()

plot_grid(pw1,pw2,pw3,pw4,pw5,pw5,pw6,pw7,pw9)
```



Let's see some variables behavior for failed devices

```
p1 <- subset(device_failure, device == "S1F0DSTY")
pw1 <- ggplot(p1,aes(x=date2,y=attribute1))+geom_point()
pw2 <- ggplot(p1,aes(x=date2,y=attribute2))+geom_point()
pw3 <- ggplot(p1,aes(x=date2,y=attribute3))+geom_point()
pw4 <- ggplot(p1,aes(x=date2,y=attribute4))+geom_point()
pw5 <- ggplot(p1,aes(x=date2,y=attribute5))+geom_point()
pw6 <- ggplot(p1,aes(x=date2,y=attribute6))+geom_point()
pw7 <- ggplot(p1,aes(x=date2,y=attribute7))+geom_point()
pw9 <- ggplot(p1,aes(x=date2,y=attribute9))+geom_point()
plot_grid(pw1,pw2,pw3,pw4,pw5,pw5,pw6,pw7,pw9)
```



Prepare Dataset For Model

From exploring the attibutes, one can come to the conclusion that attibutes 1,5, and 6 are continuous while the others are discrete with mostly 0. Also, one noticed that all the other variables have mostly zeros, but when combined the number of zeros in the new variable when a device fails is small. Hence, one can create percentage change for each attributes 1, 5, and 6, and create intermediary variables that are zero if their associated variable has a value 0 or 1 if the associated variable has 1. Finally, add up these new variables into a new variable which will have either 0, 1, 2, 3, 4.

As one cannot calculate percentage change is a device is active only one day. Devices with life length 2 or less are removed. Note: there were no devices with a length life of 2.

```
df <- device_failure %>% group_by(device) %>% summarise(N = n())
df <- subset(df, N > 2)

x <- subset(device_failure, device %in% df$device)
x <- x %>% as_tibble %>%
  arrange(device,date2) %>%
  group_by(device) %>%mutate(
```

```
                           ret.at1 = Ifelse(attribute1 == 0 |
lag(attribute1) == 0,0,Delt(attribute1,type="log")),
                           ret.at5 =Delt(attribute5,type="log"),
                           ret.at6 =Delt(attribute6,type="log")
               )

x <- na.omit(x)
```

We noticed that a correlation exists between attributes with mostly zeroes when the devices fail. Hence, one can try to see if creating a variable that count the number of non-zero may help build a better model. I still think that an RNN will be more appropriate, but I still have to start with the a simpler explainable model.

```
x$at2.zero <- ifelse(x$attribute2 == 0,0,1)
x$at3.zero <- ifelse(x$attribute3 == 0,0,1)
x$at4.zero <- ifelse(x$attribute4 == 0,0,1)
x$at7.zero <- ifelse(x$attribute7 == 0,0,1)
x$at9.zero <- ifelse(x$attribute9 == 0,0,1)
x <- x %>% mutate(num_zero =
at2.zero+at3.zero+at4.zero+at7.zero+at9.zero)
x$num_zero <- ifelse(x$num_zero >= 4,3,x$num_zero)
```

Next step is to scale and center attributes 1, 5,and 6

```
preProc <-
preProcess(x[,c("attribute1","attribute5","attribute6")],method=
c("scale","center"))

x <-   predict(preProc,x)
```

Model Building

For training, as we have only 106 devices that failed for our training set, we will choose 85 failed devices, approximately, 80%, and 500 observations from the devices that never failed. Not knowing why devices were removed, sampling from removed devices may introduce noise.

```
df <- subset(x,device %in% longest_devices)
l <- dim(df)[1]
trainNo <- sample(1:l,500)
df2 <- subset(x,failure == "yes")
l <- dim(df2)[1]
trainYes <- sample(1:l,round(l*0.8))
train <- rbind(df[trainNo,],df2[trainYes,])
test <- rbind(df[-trainNo,],df2[-trainYes,])
```

I decided to start with the simplest model, a Random Forest model with only 50 number of trees and mytr equal to 3.

Simple Random Forest Model

The simple model is a randomForest with only the following dependent variables.

failure~ attribute1+attribute5+attribute6+ret.at1+ret.at5+ret.at6+num_zero

It has has only 50 trees, and mytr is equal to 3. As the classes are unbalanced, the data are down sampled during training.

```
fiveStats <- function(...)c(twoClassSummary(...),defaultSummary(...))
fourStats <- function(data,lev=levels(data$obs),model=NULL){
  accKapp<- postResample(data[,"pred"],data[,"obs"])
  out <- c(accKapp,sensitivity(data[,"pred"],data[,"obs"],lev[1]),
           specificity(data[,"pred"],data[,"obs"],lev[2]))
  names(out)[3:4]<- c("Sens","Spec")
  out
}
```

For training, I decided to sample down, and to just perform adaptive cross validation. While I instructed train function to keep all the measures during the training, only the performance of the best model is presented in the confusion matrix.

```
fitControl <- trainControl(method = "adaptive_cv",
                           number = 10,
                           repeats = 5,
                           ## Estimate class probabilities
                           classProbs = TRUE,
                           sampling = "down",
                           ## Evaluate performance using
                           ## the following function
                           summaryFunction = twoClassSummary,
                           ## Adaptive resampling information:
                           adaptive = list(min = 10,
                                           alpha = 0.05,
                                           method = "gls",
                                           complete = TRUE))
RF_fit <-
train(failure~attribute1+attribute5+attribute6+ret.at1+ret.at5+ret.at6+
num_zero,data= train,
              method = "rf",
              trControl = fitControl
              )
```

On the test data, the accuracy of the model was 77.46%. The recall for yes was 80.95% but for no was only 77.45%. Still the precision for the no was 99.93% but only 0.97% for the yes—still 4.40 times higher than its detection rate of 0.22%
.

Table 1.Confusion Matrix

		Reference	
		No	yes
Precision	no	5949	4
	yes	1732	17

Table 2. Performance Measures

Accuracy	**77.46%**
Recall Yes	80.95%
Recall No	77.45%
Precision Yes	0.97%
Precision No	99.93%

Naïve Bayes Model

I also decided to build a Naïve Bayes model.

```
NB_fit <-
train(failure~attribute1+attribute5+attribute6+ret.at1+ret.at5+ret.at6+
num_zero,data= train,
              method = "nb",
              trControl = fitControl
              )
```

On the test data, the accuracy of the model decreased to 55.95 %. While the recall for yes was 95.23% for no it was only 55.84%. Still the precision for the no was 99.97% but only 0.5% for the yes—only twice higher than its detection rate of 0.22%
.

Table 3.Confusion Matrix

		Reference	
		No	yes
Precision	no	4289	1
	yes	3392	20

Table 4. Performance Measures

Accuracy	**55.95%**
Recall Yes	95.23%
Recall No	55.84%
Precision Yes	0.5%
Precision No	99.97%

Gradient Boosting Model

In the next model, I decided to use gradient boosting tree model, but use H20 implementation with number of trees to 50, but added a max_depth to 5, and kept the cross validation fold to 5. I also set up balance_classes to True, which is similar to down sampling. In here, I decided to us h2o library, which implementations of machine models learning appear to be more stable, and allow more flexibility than other packages. Furthermore, the package allows one to build simple auto-encoders.

```
library(h2o)
h2o.no_progress()
h2o.init(max_mem_size = "5g")
train.h2o <- as.h2o(train)

test.h2o <- as.h2o(test,destination_frame = "test.hex")

var4Training <-
c("attribute1","attribute5","attribute6","ret.at1","ret.at5","ret.at6","num_zero")
gbm_fit <- h2o.gbm(var4Training, "failure", ntree =50, seed = 100, max_depth = 5,
nfolds = 5, balance_classes = TRUE,train.h2o)
```

On the test data, the accuracy of the model decreased to 96.59 %. The recall for yes was 66.67% for no it was 96.67%. Still the precision for the no was 99.90% but only 5.18 % for the yes—20X higher than its detection rate of 0.22%. Hence, this is a better model.

Table 5. Confusion Matrix for GBM

		Reference	
		No	yes
Precision	no	7425	7
	yes	256	14

Table 6. Performance Measures

Accuracy	96.57%
Recall Yes	66.67%
Recall No	96.67%
Precision Yes	5.18%
Precision No	99.90%

In the next model, I still kept GBM but I set the number of trees to 50, but added a max_depth to 5, and kept the cross validation fold to 5 and added a learning rate of 0.01. I also set up balance_classes to True, which is similar to down sampling.

```
library(h2o)
h2o.no_progress()
h2o.init(max_mem_size = "5g")
train.h2o <- as.h2o(train)
test.h2o <- as.h2o(test,destination_frame = "test.hex")


var4Training <-
c("attribute1","attribute5","attribute6","ret.at1","ret.at5","ret.at6","num_zero")
gbm_fit <- h2o.gbm(var4Training, "failure", ntree =50, seed = 100, max_depth = 5,
nfolds = 5, learn_rate = 0.01,balance_classes = TRUE,train.h2o)
```

On the test data, the accuracy of the model decreased to 93.25 %. The recall for yes was 71.43% for no it was 93.30%. Still the precision for the no was 99.91% but only 2.83% for the yes—10X higher than its detection rate of 0.22%. Hence, this is a better model. Still can I do better?

Table 7. Extended GBM Confusion Matrix

		Reference	
		No	yes
Precision	no	7167	6
	yes	514	15

Table 8. Performance Measures

Accuracy	93.25%
Recall Yes	71.43%
Recall No	93.3%
Precision Yes	2.83%
Precision No	99.1%

What is the performance on devices that were removed but didn't fail?

In the case of devices that never failed, I decided to use the best model

```
td <- subset(x , device %in% device_left)
td.h2o <- as.h2o(td,destination_frame = "td.hex")
result_gbm2 <-  h2o.predict(gbm_fit, td.h2o)
result <- as.data.frame(result_gbm2$predict)
```

From 104,537 observations, the model was able to detect with an accuracy of 72.55% or 75,844 as true negatives.

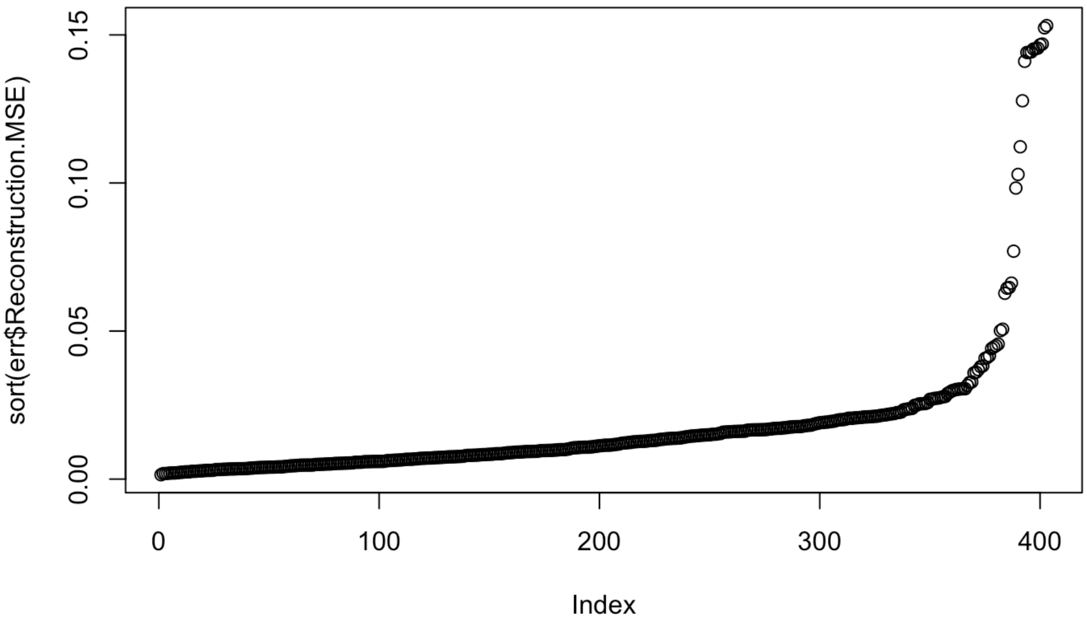
Dimension Reduction/ Anomaly Detection With Random Forest

Can we improve the models performance without tuning parameters? One approach is to perform a dimension reduction using auto-encoding, find where the auto-encoder failed (find the anomalies) and rebuild a RandomForest model with the cleaner data.

I've decided to use auto-encoding not only for dimension reduction but also to get a better test sample.

```
var4Training <-
c("attribute1","attribute2","attribute3","attribute4","attribute5",

"attribute6","attribute7","attribute9","at2.zero","at3.zero","at7.zero",
                  "at9.zero","ret.at1","ret.at5","ret.at6")
train.dl = h2o.deeplearning(x=var4Training ,training_frame = train.h2o,
                            autoencoder = TRUE,
                            reproducible = T,
                            seed = 1234,
                            categorical_encoding = "Binary",
                            hidden = c(15,10,5, 10,15),epoch = 100)
train.anon <- h2o.anomaly(train.dl,train.h2o,per_feature = FALSE)
head(train.anon)
err <- as.data.frame(train.anon)
plot(sort(err$Reconstruction.MSE))
```




```
Any value larger than this implies large error, so the observation will be
removed from the test set .

sort(err$Reconstruction.MSE)[350]
0.0269581
```

Build a new model with the cut off 0.0269581. This allows one to build a model with less noise.

```
var4Model <- c(var4Training,"failure")
train_df_auto <- train[err$Reconstruction.MSE < 0.0269581,var4Model]
train.h2o <- as.h2o(train_df_auto,destination_frame = "testDevice.hex")
Build a Random Forest model
h2oRF <- h2o.randomForest(var4Training,y="failure",
                                    training_frame = train.h2o,
                                     seed = 1234,nfolds = 5
)
```

```
resultInc <- predict(h2oRF,newdata=test.h2o,type="response")

t <- as.data.frame(test.h2o$failure)
r <- as.data.frame(resultInc$predict)
confusionMatrix(r$predict,t$failure,positive="yes")
```

On the test data, the accuracy of the model decreased to 99.83 %. The recall for yes was 66.67% for no it was 99.47%. Still the precision for the no was 99.91% but only 25.45% for the yes—50X higher than its detection rate of 0.22%. If the goal is to reduce the false positives than this is the best model, even if our recall for yes decreased.

Table 9 Dimension Reduction with Anomaly Detection Confusion Matrix

		Reference	
		No	yes
Precision	no	7640	7
	yes	41	14

Table 10. Performance Measure

Accuracy	99.83%
Recall Yes	66.67%
Recall No	97%
Precision Yes	25.45%
Precision No	99.1%

For the data that never failed, our recall is 72.66%, or 75,963 out of 10,4537 which is better than any of the other models.

Can RNN produce better results?

The previous models built were 1 layer, "flat", Machine learning models. One can improve them by creating new variables (feature engineering). However, there are alternative models in Machine Learning that can take in consideration temporal changes (sequence) which may be more appropriate for such problem where some attributes are time-series. These models are Recurrent Neural Networks.

From the exploration of the dataset, one noticed that attributes 1,5, and 6 were time-series. Hence, makes sense to use RNN to differentiate between the behaviors of working device from a failed device. To build RNN, I will use MxNet, which is the deep learning architecture used by Amazon, in Mathematica. I could have built the models in Python using Keras, but the code is almost exactly the same as Mathematica code, except that Mathematica uses MxNet, which is the deep learning architecture used by Amazon.

Prepare the datasets:

Take only the last 5 days until failure of a device

```
partition4FailedDevices = {};
Table[tp = Select[onlyYes,#[[1]] == i&];
      tp= SortBy[tp,#[[2]]&];
      l = Dimensions[tp][[1]];
      partition4FailedDevices  = Append[partition4FailedDevices
,Rule[tp[[l-4;;,3;;15]],"yes"]],{i,nameOfFailedDevices }]
```

```
Slice each device that never failed into a matrix of size 5 by the
number of variables
onlyNo = {};
Table[tp = Select[data,#[[1]] == i&];
      tp= SortBy[tp,#[[2]]&];
      X= Partition[tp,5];
            onlyNo =
      Append[onlyNo,Map[Rule[#[[All,3;;15]],"no"]&,X]],{i,devicesThatNe
      verFailed }

onlyNo = Flatten[onlyNo]

Take 80% of the the devices with failed signal, and 500 5-days sequence
from the devices that never failed

trainYesNdx = RandomChoice[Range[106],85]
trainNoNdx = RandomChoice[Range[1620],500]
testNoNdx = Complement[Range[1620],trainNoNdx]
testYesNdx = Complement[Range[106],trainYesNdx]

train =
Join[partition4FailedDevices[[trainYesNdx,All]],onlyNo[[trainNoNdx,All]
]]

 test =
Join[partition4FailedDevices[[testYesNdx,All]],onlyNo[[testNoNdx,All]]]
 train = DeleteCases[train,{}->"yes"]

In the testing set there are 44 sequences labeled “yes” and 1190
labeled “no”.  Hence, the prevalence is 3.70%

Build a Simple LSTM Model with 3 hidden layers.

net=NetChain[{LongShortTermMemoryLayer[64],SequenceLastLayer[],LinearLa
yer[2],SoftmaxLayer[]},"Input"->{5,13},"Output"-
>NetDecoder[{"Class",{"no","yes"}}]]
net=NetInitialize[net];

As the batch size is small use learning rate equal to 0.001, and to
insure that the model doesn’t get stuck in a local minimum use ADAM
with beta1 = 0.99

trained2=NetTrain[net,train,BatchSize->50,
MaxTrainingRounds->100,Method-> {"ADAM","LearningRate"-> 0.001}]

r = Map[trainedNetSimple[#] &, test[[All, 1, All]]]; val =
Counts[MapThread[List, {r, test[[All, 2]]}]]
```

On the test data, the accuracy of the model decreased to 95 %. The recall for yes was 70% for no it was 96%. Still the precision for the no was 99% and for the yes was 40%—10X higher than its detection rate of 3.70%.   This model had a precision of 74% when I ran it only on devices removed.

Table 11. Confusion Matrix

		Reference	
		No	yes
Precision	no	1144	12
	yes	46	32

Table 12. Performance Measures

Accuracy	**95%**
Recall Yes	70%
Recall No	96%
Precision Yes	40%
Precision No	99%

I also created a more sophisticated model with 3 Gated RNN (reason for Gated RNN), with dropout layers to reduce overfitting.

```
netSimpleRNN=NetChain[{GatedRecurrentLayer[32,"Dropout"->
{"VariationalInput"-> 0.2,"VariationalState"-> 0.3}],

GatedRecurrentLayer[16,"Dropout"->  {"VariationalInput"->
0.2,"VariationalState"-> 0.3}],

GatedRecurrentLayer[8,"Dropout"->  {"VariationalInput"->
0.3,"VariationalState"-> 0.3}],
GatedRecurrentLayer[8,"Dropout"->  {"VariationalInput"->
0.3,"VariationalState"-> 0.3}],


SequenceLastLayer[],LinearLayer[2],SoftmaxLayer[]},
"Input"->{5,13},"Output"->NetDecoder[{"Class",{"no","yes"}}]]
netSimpleRNN=NetInitialize[netSimpleRNN];
```

For this model I increase the epoch, added a validation set to reduce overfitting during training, and reduce my learning rate to 0.001

```
trainedNetSimple=NetTrain[netSimpleRNN,train,
BatchSize->64,MaxTrainingRounds->300,ValidationSet->Scaled[0.2],
Method-> {"ADAM","LearningRate"-> 0.001}]
```

On this model, the accuracy, precision of failure, and the precision of the working devices of the model increased to 96 %, 43%, and 97%, respectively, but the recall of failure decreased. However, the precision of working devices stayed the same.

		Reference	
		No	yes
Precision	no	1150	14
	yes	40	30

Accuracy	**96%**
Recall Yes	68%
Recall No	97%
Precision Yes	43%
Precision No	99%

What is interesting is this same model recall of "no" 78.17% or 16,085 out of 20,577 (5 days sequences) or models that never failed
To improve the model, I reduced the batch size to 32, the epoch to 100, and set learning rate to 0.001 (see results below). I also added the number of neuron in layers (64, 128, 64) instead of (16,8,8):

```
netSimpleRNN2 =   NetChain[{
GatedRecurrentLayer[64,      "Dropout" -> {"VariationalInput" -> 0.3,
"VariationalState" -> 0.2}],
GatedRecurrentLayer[128,      "Dropout" ->  {"VariationalInput" -> 0.3,
"VariationalState" -> 0.2}],
GatedRecurrentLayer[64,      "Dropout" ->  {"VariationalInput" -> 0.3,
"VariationalState" -> 0.2}], SequenceLastLayer[],     LinearLayer[2],
SoftmaxLayer[]}, "Input" -> {5, 13},    "Output" ->
NetDecoder[{"Class", {"no", "yes"}}]] netSimpleRNN2 =
NetInitialize[netSimpleRNN2];
```

When I trained the model with validation set using 20% of the training set, the model was, as expected, optimized for the larger class to the detriment for the minority class. I though by adding L2 Regularization will improve the performance, but the opposite happened. I believe this is due to the low colinearity.

```
trainedNetSimple =   NetTrain[netSimpleRNN2, train, BatchSize -> 32,
MaxTrainingRounds -> 100,    Method -> {"ADAM", "LearningRate" ->
0.001}]
```

This model was an improvement of the previous model in it ability to recall more yes (72% Vs 68%) and higher precision (49.23% vs 43%) while all other measure were comparable . Most importantly, it was able to recall 79.46% of non-failed devices, which is an improvement from the previous model—78..17%.

Table 13. Performance Measure

Accuracy	**96.35%**
Recall Yes	72.73%
Recall No	97.23%
Precision Yes	49.23%
Precision No	98.97%

Conclusion

I was given the task of predicting devices failure knowing they were monitored by 9 attributes.   Before, building the model, I took the time to explore the dataset (size). The dataset had only 106 devices that failed, and 27 that never failed. The rest, or over 1000 devices, they were removed but the reasons of their removal were not stated. Hence, it made sense to build and test the models using only failed and never failed devices.

From the exploration of the dataset, I discovered that the attribute 7 and 8 are identical. Hence, I removed attribute 8. Furthermore, attribute 1, 5, 6 are continuous while the other attributes tend to be mostly equal to 0. Hence, I decided to calculate the daily percentage change for each of these variables and add them to the dataset. For attributes 2,3,4,7, and 9 I could have just removed them, but further exploration showed that when 1 or more of these attributes have a value larger than 0 these might imply a failure. Still, these variables were scaled. Therefore, I decided to create a variable that just count the number of these attributes with at least one value different from 0.

From exploring the dataset, I discovered few other interesting facts: about 25% of device failed within 26.5 days, and 50% failed within 92 days. On the other hand, 25% of all devices are removed within 6 days, but 50% are removed within 84 days, or 8 days before failed devices. This means, that working devices are removed too soon. If 3D Technology wants to decide when to perform a maintenance than 27 days interval should be good enough to catch failed devices. From the table, it looks the maintenance is performed either too late or early.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
All Devices	1.0	6.0	84.0	106.5	224.0	304.0
Failed Devices	5.0	26.5	92.0	101.1	148.0	299.0

Before building the models, I have to decide on the objective function. What is the objective function that I want to pursue? Do I want to recall most of the no, improve the yes precision, or recall most of the yes? For my experience with extremely unbalanced classes, it is very hard to have everything, and from the model created, one can see that improving one measure can result in a deterioration of another measure.   Still, I build basic “flat” machine learning models: RandomForest, Naïve Bayes, Gradient Boosted Model (GBM), Enhanced GBM, and a more complicated model that used auto-encoding anomaly detection to improve sample used for training and improve model performance. In all these cases, I made sure to perform a down sampling during the training.

Given the nature of some attributes, I also decided to build Recurrent Neural Networks with a 5 day sequence window: Long-Short Term Memory (LSTM) and

Gated Recurrent Neural Network (GRN). I used 5 days because the 1st device to fail failed within 5 days of putting it in production.

The following table provides the performance of all the models I built. In accuracy, the model that used dimension reduction and anomaly detection to build a randomForest (RF) model had the highest accuracy, 99.83 and almost perfect recall of yes, 99.47, and came in second to GRN in yes precision, 25.45 versus 43. This is great when one knows that the prevalence of yes is around 0.2% in the original dataset.

While Naïve Bayes had the highest recall of failure (yes), 95.23%, this came with a lowest accuracy, 55.95%, and lowest yes precision, 0.5, just over twice the prevalence of yes in the original dataset. This model is inappropriate.

I think, in device failure, 3D Technology is more interested in increasing its detection rate while still having low false positive. In this case, the GRN MDF will be the most appropriate model. It has a detection rate of 49.23% while having a no precision of 98.23%. This model, even when provided removed devises, had an accuracy of 79.46%, versus 74% from LSTM, still better than all the "flat" models. These results are better than any "flat" machine learning models.

	RF	**Naïve Bayes**	**GBM**	**GBM Enhanced**	**DimRed**	**LSTM**	**GRN**	**GRN MDF**
Accuracy	67.48	55.95	96.59	93.25	99.83	95	96	96
Recall Yes	80.95	95.23	66.67	71.43	66.67	70	68	72.73
Recall No	77.45	55.84	96.67	93.3	99.47	96	97	97.23
Precision Yes	0.97	0.5	5.18	2.83	25.45	40	43	49.23
Precision No	97	99.97	99.9	99.91	99.91	99	99	98.97
F1 Score Yes	1.92	0.99	9.61	5.44	36.84	51	52.68	58.72
F1 Score No	86.13	71.67	98.26	96.49	99.69	97.5	97.99	98.09

Of course, these are just initial models and there is more room for improvement. For instance, the machine learning models could have been improved by thinking about new features and the RNN could have had a larger window than 5 days.

F1 = 39.34 Reference

Prediction no yes

no 7653 9

yes 28 12

Accuracy : 0.9952

95% CI : (0.9934, 0.9966)

No Information Rate : 0.9973

P-Value [Acc > NIR] : 0.999474

Kappa : 0.3913

Mcnemar's Test P-Value : 0.003085

Sensitivity : 0.571429

Specificity : 0.996355

Pos Pred Value : 0.300000

Neg Pred Value : 0.998825

Prevalence : 0.002727

Detection Rate : 0.001558

Detection Prevalence : 0.005193

Balanced Accuracy : 0.783892

'Positive' Class : yes