Reduce Maintenance Cost Through Predictive Techniques

Yassine Landa

13/02/2019

NOTE : This document as well as the original RMarkdown will be available momentarly here. For readability, an html notebook is also available here.

1 Introduction

When we are doing asset management we are interested in *keeping a high level of availability (utilization rate)* and *reducing the cost of maintenance (operational costs)*. Usually that's done by looking at historical data and deciding on a schedule for check-ups, repairs or replacements that will optimize the up time of the asset and cost of maintenance.

Nowadays with the availability of real-time data (connected assets) we can adopt a dynamic and adaptive approach that reduce the cost of maintenance even more by trying to predict exactly (or within a time frame) when a system will fail and act accordingly.

Furthermore, the advantages of this approach is that predictive analytics can provide information about the root causes of why such failures occurs on a granular level and in real-time, which helps improve future business decisions.

This notebook will explore how a predictive system can be implemented to detect such failures using the device_failure.csv data-set.

We will start by exploring the data and deriving a methodology for resolving the task based on initial insights. Then we will move to modelisation using two different approaches:

- Device data-set: a simpler problem to solve where we will focus on feature engineering, parameter tuning, feature importance and lastly evaluation.
- Logs/Device data-set: a harder problem to solve where we will take advantage of the time component discussing labeling strategies, avoiding leakage and the potential of such approach.

Last we will finish by listing potential improvement to take this analysis further.

2 Exploring The Data

2.1 Preliminary Exploration

By looking at the data it seems we have daily logs from different devices over a period of 9 months (between 2015-01-01 and 2015-11-02). For each row (log) we have a *failure* column and 9 attributes that contains numerical values. These can be direct output from sensors (e.g temperature, speed or pressure etc..) for the well distributed attributes like *attribute1* or can be error codes for the others.

We notice also that the *attribute7* and *attribute8* summary statistics looks suspiciously alike.

Looking at the different devices and failures more closely we notice that there is 1169 devices in total. 106 failures logged (0.085%). This analysis raises some questions like:

- What is the failure distribution over devices (can we have multiple failures for 1 device)?
- What is the failure distribution over time?
- What is the proportion of devices failing (is there any devices not failing in the data-sets)?
- Distribution of device logs per day (apparently we don't have 1169 log per day)?
- Do we stop receiving logs after device failure (is there logs from same device after failure recorded)?
- It seems we are dealing with an very imbalanced data-set, how can we tackle that?

2.2 Exploring Devices and Failures

To answer the first questions let's look at the distribution of logs and failures daily.

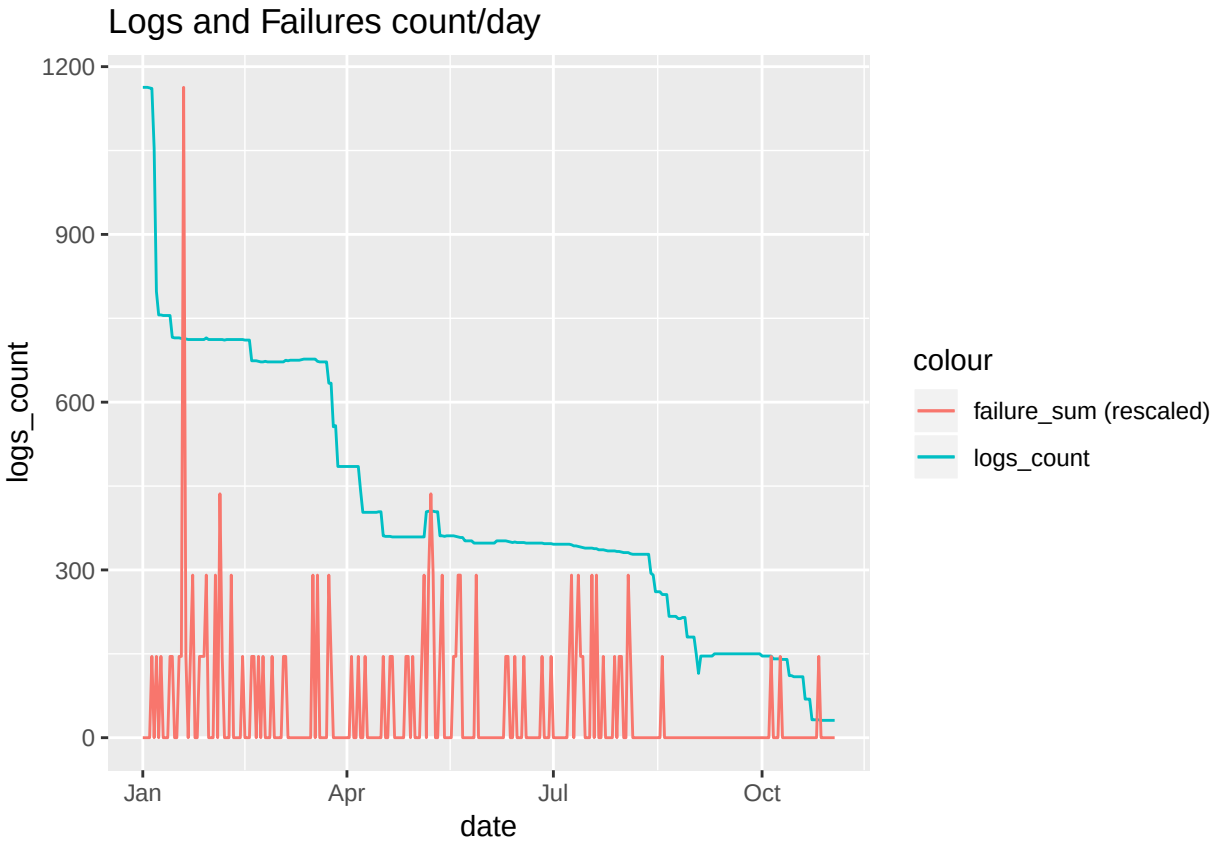
- In general we are receiving logs from an average of 409 devices each day, and we should expect a failure to happens once each 3 days (except the day when we had 8 failures).

```
foo <- dt %>%
  group_by(date) %>%
  summarise(failure_sum = sum(failure), logs_count = n())

summary(foo) #daily failures summary
```

```
##       date             failure_sum       logs_count
##  Min.   :2015-01-01   Min.   :0.0000   Min.   :  31.0
##  1st Qu.:2015-03-17   1st Qu.:0.0000   1st Qu.: 261.0
##  Median :2015-06-01   Median :0.0000   Median : 350.5
##  Mean   :2015-06-01   Mean   :0.3487   Mean   : 409.5
##  3rd Qu.:2015-08-16   3rd Qu.:0.2500   3rd Qu.: 672.0
##  Max.   :2015-11-02   Max.   :8.0000   Max.   :1163.0
```

- We also see that there is a decline of logs that is not necessary linked to having a device failing (big drop in January, March and August).

```
ggplot(foo, aes(date)) +
  geom_line(aes(y = logs_count, colour = "logs_count")) +
  geom_line(aes(y = failure_sum*(max(logs_count)/max(failure_sum)), colour = "failure_sum (rescaled)")) +
  ggtitle("Logs and Failures count/day")
```



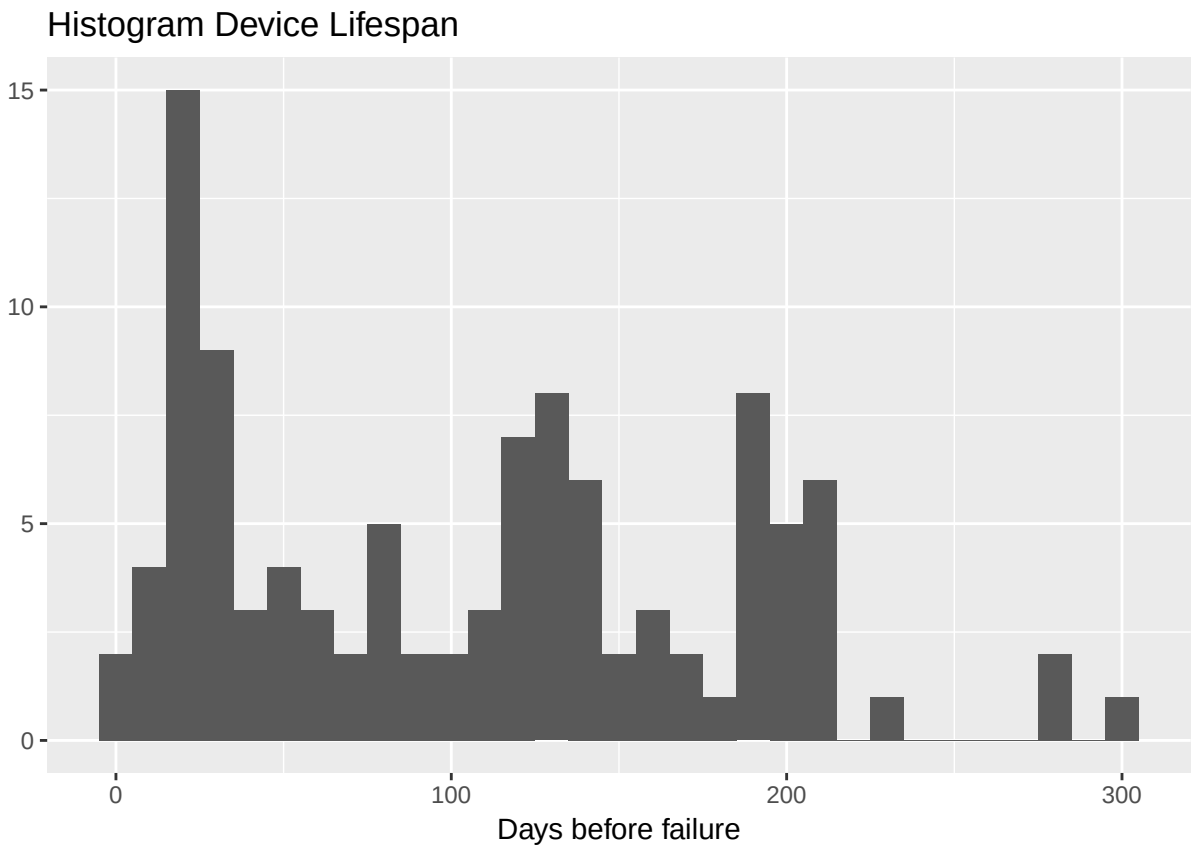
- We have also received some logs from devices after failure happened. But other than those, the log with `failure = 1` is usually the last for that device. For simplification we will remove all logs for failed devices after failure happened.

```
failed_device_names = dt[failure == 1]$device

for (dev in failed_device_names){
  device_logs = dt[ device == dev]
  total_logs = dim(device_logs)[1]
  index_failure = device_logs[failure == 1, which = TRUE]
  if(total_logs != index_failure){ #did failure was logged last?
    cat(total_logs, "== ",  index_failure, " ?\n")
    cat('There are', total_logs - index_failure, 'logs after failure for device =', dev,'!\n')
    #removing logs after failures from the dataset
    dt <- dt[!which(dt$device == dev)[(index_failure+1):total_logs]]
  }
}
```

```
## 126 ==  125  ?
## There are 1 logs after failure for device = S1F136J0 !
## 131 ==  129  ?
## There are 2 logs after failure for device = W1F0KCP2 !
## 131 ==  129  ?
## There are 2 logs after failure for device = W1F0M35B !
## 205 ==  193  ?
## There are 12 logs after failure for device = S1F0GPFZ !
## 229 ==  199  ?
## There are 30 logs after failure for device = W1F11ZG9 !
```

- We also see that the lifespan of devices (days before they fail) is not equally distributed or skewed towards the right as one might assume (older devices fail more). Most of devices failed after ~25 days and the rest failed around ~125 days and 200 days or sending data.

```
device_lifespan = data.table(failed_device_names,
   logs_before_failing = sapply(failed_device_names,
                          function(dev) dim(dt[ device == dev])[1]) - 1,
   days_before_failing = sapply(failed_device_names,
                          function(dev) as.numeric(sum(diff(dt[ device == dev ]$date))) - 1)
   )


qplot(device_lifespan$days_before_failing,
      geom="histogram",
      binwidth = 10,
      main = "Histogram Device Lifespan",
      xlab = "Days before failure")
```



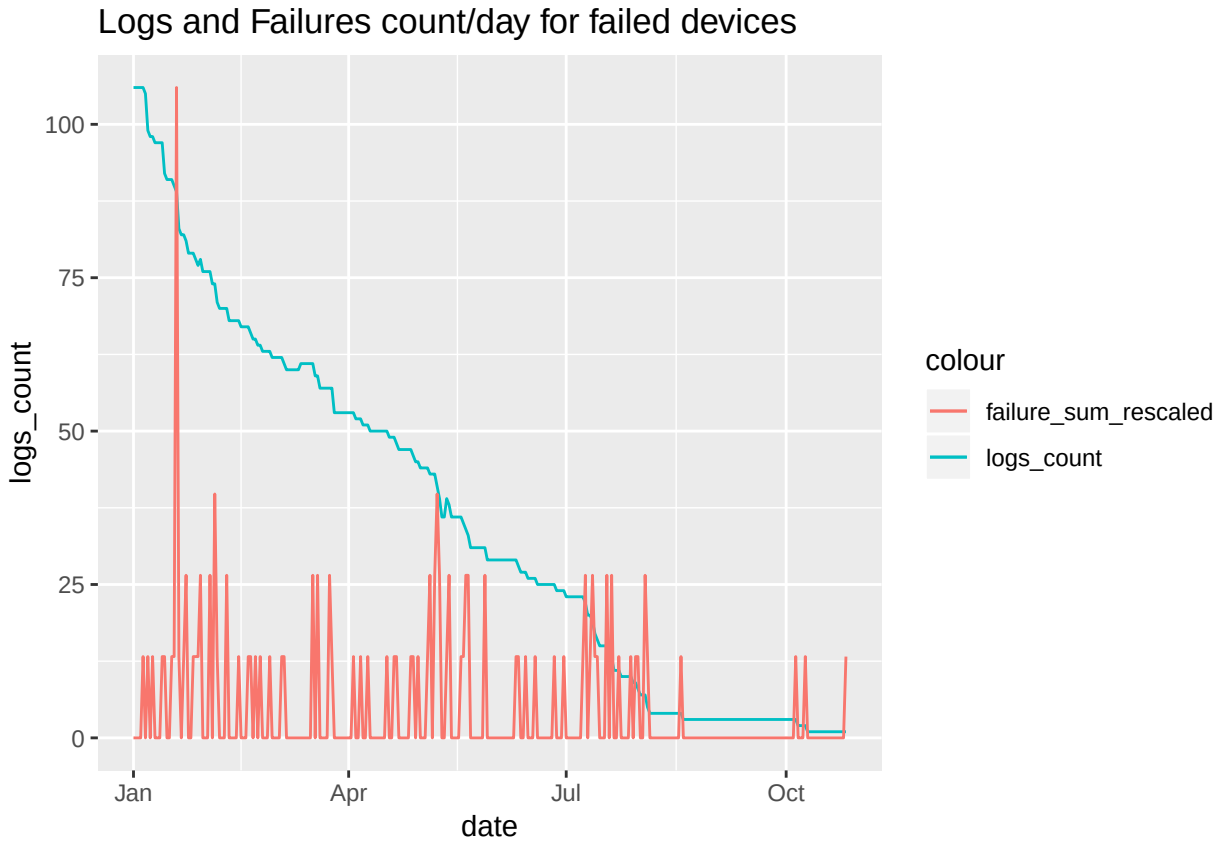
We also see that we have ~1% of logs related to failures within logs sent by failed devices. This suggests that we can take two approaches to try to predict failures:

- Construct a device data-set and try to classify devices that will fail from devices that didn't (positive class ~9%)
- Construct a log/device data-set just on failed devices and try to classify days when failure will happen and days when all is fine (positive class ~1%). This approach sounds more complicated but promising, can integrate the time components and will have more data to learn from, especially if adopt a different labeling approach (trying to predict failure X days in advance for example).

```
dt_failed = dt[device %in% failed_device_names]

cat("positive target class (failures) ratio:",
    formatC(100 * mean(dt_failed$failure),
            format = 'f', digits = 3),
    "%")
```

```
## positive target class (failures) ratio: 0.994 %
```

```
foo <- dt_failed %>%
  group_by(date) %>%
  summarise(failure_sum = sum(failure), logs_count = n())

ggplot(foo, aes(date)) +
  geom_line(aes(y = logs_count, colour = "logs_count")) +
  geom_line(aes(y = failure_sum*(106/8), colour = "failure_sum_rescaled")) +
  ggtitle("Logs and Failures count/day for failed devices")
```



2.3 Date Analysis

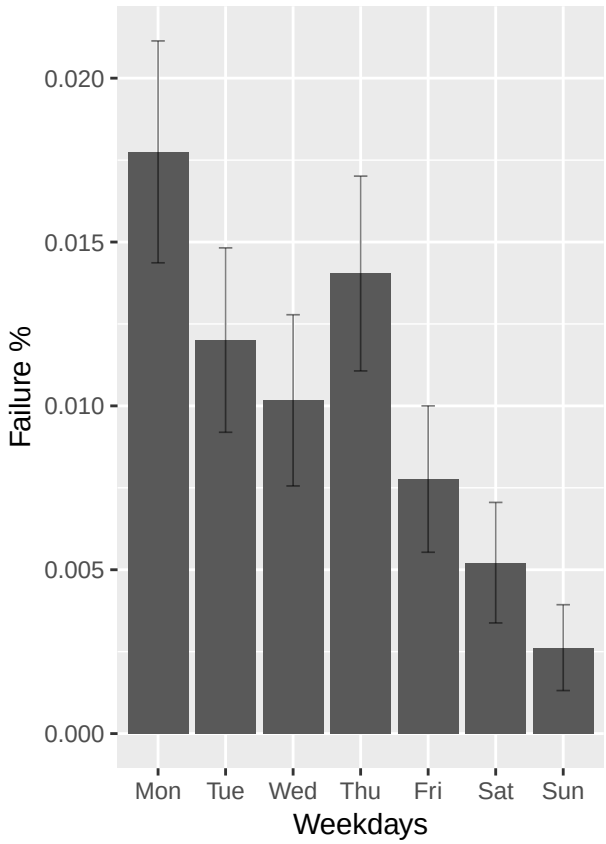
The time-stamp on each log can also contains interesting information. By looking at *when* the failures happen through the week or the month (ideally through the year if we ad a multi-year data-set) we can see that most of the failures happen on a Monday & Thursday as well as on the 3rd week of the month.

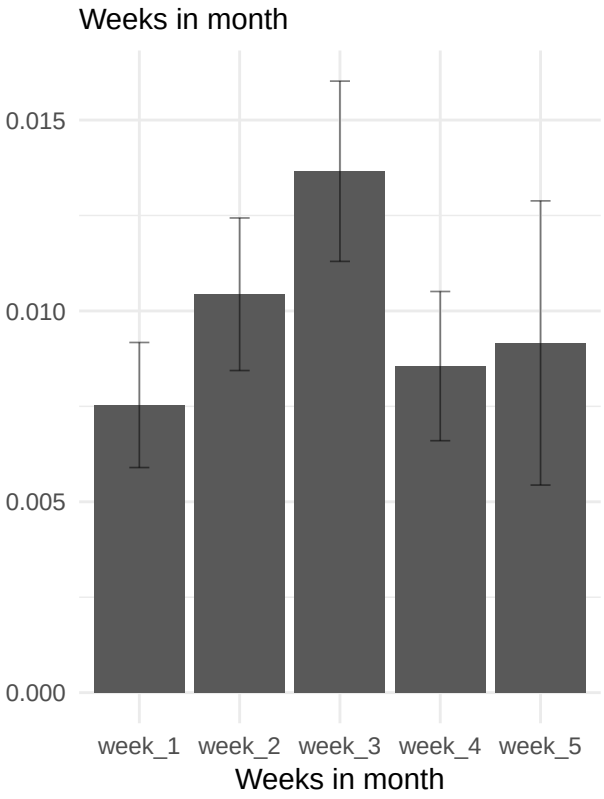
```
#analysis by weekday
foo <- dt_failed %>%
  group_by(weekdays = str_sub(weekdays(date),1,3)) %>%
```

```
  summarise(failure_mean = mean(failure),
            failure_sum = sum(failure),
            count = n(),
            error = sqrt((failure_mean * (1-failure_mean))/count))

foo<- data.table(foo)
foo$order = c(5,1,6,7,4,2,3)

plot1 <- ggplot(foo, aes(reorder(weekdays, order), failure_mean)) +
  geom_col(position = "dodge") +
  geom_errorbar(aes(ymin = failure_mean - error, ymax = failure_mean + error),
                position = position_dodge(0.9), size= .3, width= .2, alpha= .5) +
  xlab("Weekdays") +
  ylab("Failure %")
  ggtitle("Failures dist per weekday", subtitle = "Weekdays") +
  theme_minimal()
```

```
## NULL
```

```
#analysis by week in month
foo <- dt_failed %>%
  group_by(monthweeks = paste0('week_',ceiling(day(date) / 7)) ) %>%
  summarise(failure_mean = mean(failure),
            failure_sum = sum(failure),
            count = n(),
            error = sqrt((failure_mean * (1-failure_mean))/count))

plot2 <- ggplot(data.table(foo), aes(monthweeks, failure_mean)) +
  geom_col(position = "dodge") +
  geom_errorbar(aes(ymin = failure_mean - error, ymax = failure_mean + error),
                position = position_dodge(0.9), size= .3, width= .2, alpha= .5) +
  xlab("Weeks in month") +
  ylab(NULL) +
  ggtitle(" ", subtitle = "Weeks in month") +
  theme_minimal()

grid.arrange(plot1, plot2, ncol=2)
```





2.4 Attributes Analysis

To start exploring how can we use the attributes let's calculate some summary statistics (box-plot) and look at the unique values within each attribute.

- Almost all attributes have very few unique values (except *attribute1* and *attribute6*) which is consistent with the hypothesis that they might treated as categorical values (they might be representing error codes, messages, etc. . .)
- *attribute1* looks like a time series with a very high frequency but nothing points to any difference between to the classes. . .
- *attribute6* value increases almost linearly (on a long term view $>$ 1.5 month) as long as the device is still working. But again nothing seems to suggest any difference between the two classes.
- Looking at the distribution of attributes across both classes, we notice that attribute1 can be normalized (if needed)
- *attribute7* and *attribute8* are actually duplicates.

```
dt$attribute8 <- NULL
```

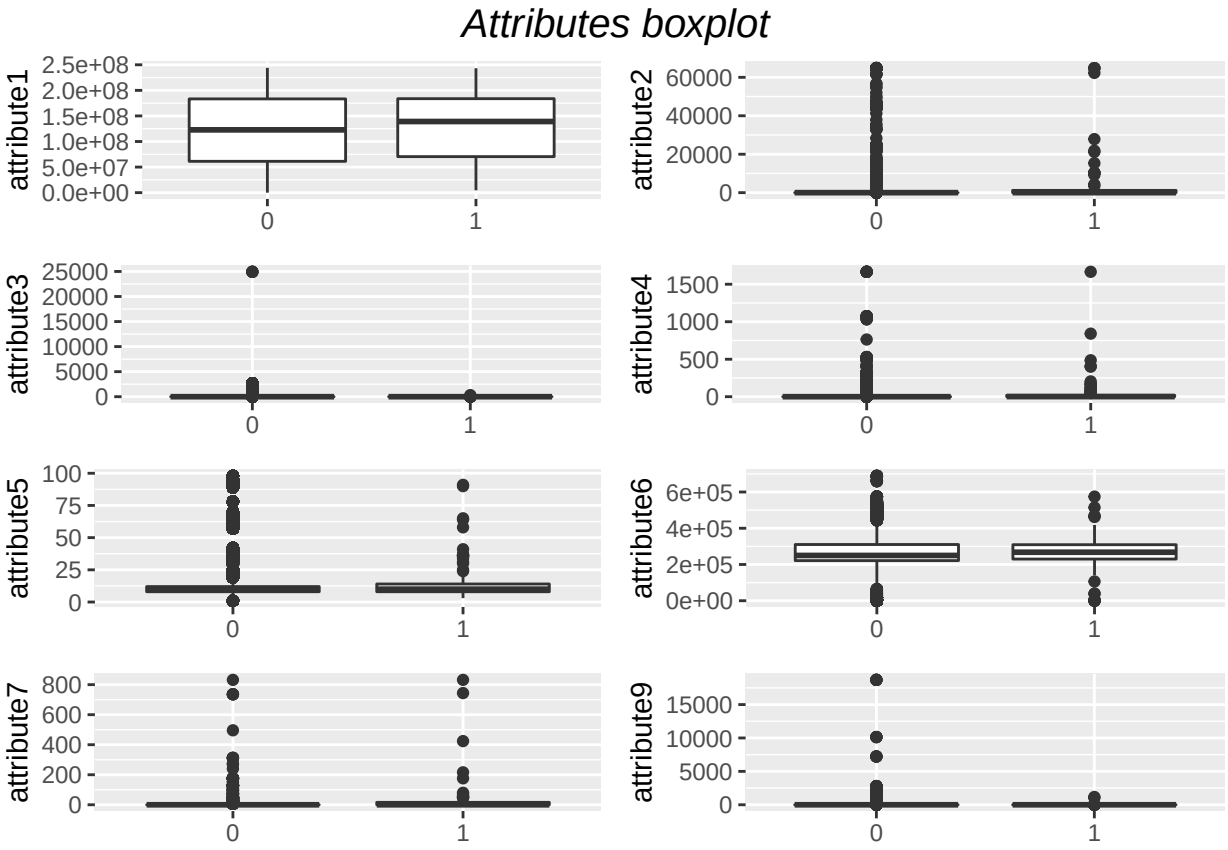
While the distribution of attributes doesn't give us a lot of information about the difference between the 2 classes (especially considering we just have a few observations of the positive class), looking at the change of attributes before failure is more insightful (Figure *Change in attributes before failure by device (all devices)* failed devices are colored, working devices in light gray):

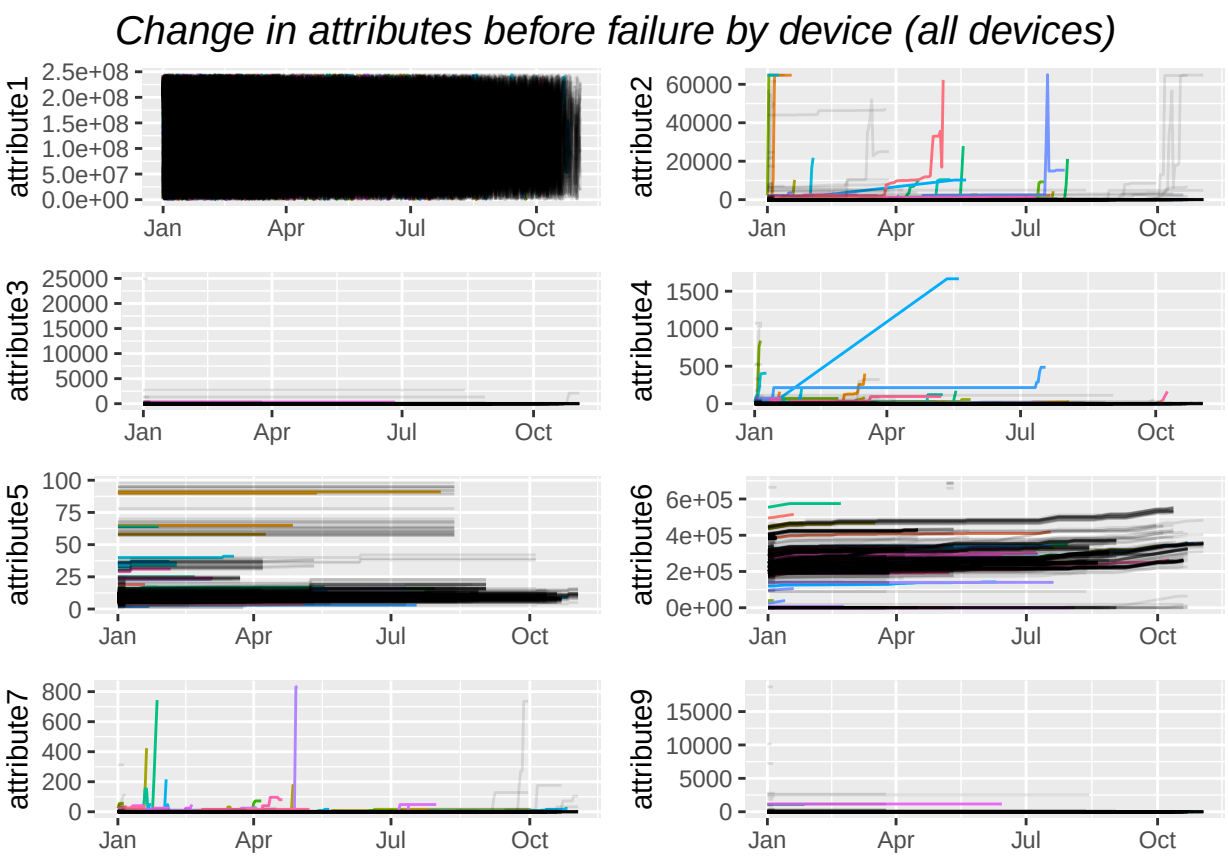
- In fact, *attribute2*, *attribute4* and *attribute7* change dramatically just before the failure point.
- small increases in *attribute2* looks like what characterize values before failure

- *attribute4* is the only variable where there is a clear surge in its values right before the devices failed (*attribute2* and *attribute7* have these type of increases in values too but not necessary before a failure ~ data from October). So *attribute4* might the best predictor we have in the data-set.

```
dt = transform(dt, failure = as.character(failure))

dist_plots <- list()
time_plots <- list()
time_plots_all <- list()

for (i in 4:length(colnames(dt))){
  cat('>>', colnames(dt)[i], ':\n')

  dist_plots[[(i-3)]] <- ggplot(data = dt,
                                aes_string('failure', colnames(dt)[i])) +
                               geom_boxplot() +  theme(legend.position="none") +
                               ylab(colnames(dt)[i]) + xlab(NULL)

  time_plots[[(i-3)]]   <- ggplot(data = dt[device %in% failed_device_names], aes_string('date')) +
                               geom_line(aes_string(y = colnames(dt)[i], color='device')) +
                               theme(legend.position="none") +
                               ylab(colnames(dt)[i]) + xlab(NULL)

  time_plots_all[[(i-3)]]   <- ggplot(data = dt[device %in% failed_device_names],
                                      aes_string('date')) +
                               geom_line(data = dt[device %in% failed_device_names],
                                      aes_string(y = colnames(dt)[i], color='device')) +
                               geom_line(data = dt[!device %in% failed_device_names],
                                      aes_string(y = colnames(dt)[i], group='device'),
                                      alpha=0.1) +
                               theme(legend.position="none") +
                               ylab(colnames(dt)[i]) + xlab(NULL)

  u_attr = unique(unlist(dt[, colnames(dt)[i] , with = FALSE]))
  n_attr = length(u_attr)
  cat('number of unique attributes:',n_attr,'\n')
  # if(n_attr < 100) cat('The unique attributes are:\n',u_attr,'\n')
}
```

```
## >> attribute1 :
## number of unique attributes: 123829
## >> attribute2 :
## number of unique attributes: 557
## >> attribute3 :
## number of unique attributes: 47
## >> attribute4 :
## number of unique attributes: 113
## >> attribute5 :
## number of unique attributes: 60
## >> attribute6 :
## number of unique attributes: 44814
## >> attribute7 :
## number of unique attributes: 28
## >> attribute9 :
## number of unique attributes: 65
```


Attributes boxplot
attribute1
attribute2
attribute3
attribute4
attribute5
attribute6
attribute7
attribute9
0
1


Change in attributes before failure by device (all devices)
attribute1
attribute2
attribute3
attribute4
attribute5
attribute6
attribute7
attribute9
Jan
Apr
Jul
Oct

3 Device Dataset

3.1 Dataset construction & Feature engineering

From the analysis above it seems we can classify the faulty devices by capturing the change in attributes right before the failure happens.

For this approach to be successful we need also to capture the general difference to the overall attribute distribution across devices, the relative difference to the values for the device itself.

This approach also "squashes" the time components so some information might be lost, this can effect can be reduced by choosing the right time window to calculate these features. We are going to start with a time window = 3 days and this value can be considered as a parameter of the model to be tuned.

3.1.1 Features for device failure classification:

1- mean, sd, max, min calculated over all attributes values. Example features for *attribute1*: **attr1mean**, **attr1sd**...

2- last 3 values*. Example features for *attribute1*: **attr1last1**, **attr1last2**, **attr1last3**

3- mean, sd, max, min calculated over the last 3 days values*. Example features for *attribute1*: **attr1mean_3d**, **attr1sd_3d**...

4- difference between 1st value (1st log from device) and last 3 values before failure*. Example features for *attribute1*: **attr1diffto_last1**, **attr1diffto_last2**, **attr1diffto_last3**

5- count of logs (lifespan) (**logs_n**)

6- count of days (lifespan) (**n_days**)

*: failure day value included

```
device_dataset <- dt %>%
  arrange(device, date) %>%
  dplyr::group_by(device) %>%
  dplyr::summarise(

          #mean over device lifespan
          attr1mean = mean(attribute1),
          attr2mean = mean(attribute2),
          attr3mean = mean(attribute3),
          attr4mean = mean(attribute4),
          attr5mean = mean(attribute5),
          attr6mean = mean(attribute6),
          attr7mean = mean(attribute7),
          attr9mean = mean(attribute9),

          #sd over device lifespan
          attr1sd = sd(attribute1),
          attr2sd = sd(attribute2),
          attr3sd = sd(attribute3),
          attr4sd = sd(attribute4),
          attr5sd = sd(attribute5),
          attr6sd = sd(attribute6),
          attr7sd = sd(attribute7),
          attr9sd = sd(attribute9),
```

```
        #max over device lifespan
        attr1max = max(attribute1),
        attr2max = max(attribute2),
        attr3max = max(attribute3),
        attr4max = max(attribute4),
        attr5max = max(attribute5),
        attr6max = max(attribute6),
        attr7max = max(attribute7),
        attr9max = max(attribute9),

        #min over device lifespan
        attr1min = min(attribute1),
        attr2min = min(attribute2),
        attr3min = min(attribute3),
        attr4min = min(attribute4),
        attr5min = min(attribute5),
        attr6min = min(attribute6),
        attr7min = min(attribute7),
        attr9min = min(attribute9),

        #last 3 values
        attr1last1 = attribute1[max(length(attribute1),1)],
        attr1last2 = attribute1[max(length(attribute1)-1,1)],
        attr1last3 = attribute1[max(length(attribute1)-2,1)],

        attr2last1 = attribute1[max(length(attribute2),1)],
        attr2last2 = attribute1[max(length(attribute2)-1,1)],
        attr2last3 = attribute1[max(length(attribute2)-2,1)],

        attr3last1 = attribute1[max(length(attribute3),1)],
        attr3last2 = attribute1[max(length(attribute3)-1,1)],
        attr3last3 = attribute1[max(length(attribute3)-2,1)],

        attr4last1 = attribute1[max(length(attribute4),1)],
        attr4last2 = attribute1[max(length(attribute4)-1,1)],
        attr4last3 = attribute1[max(length(attribute4)-2,1)],

        attr5last1 = attribute1[max(length(attribute5),1)],
        attr5last2 = attribute1[max(length(attribute5)-1,1)],
        attr5last3 = attribute1[max(length(attribute5)-2,1)],

        attr6last1 = attribute1[max(length(attribute6),1)],
        attr6last2 = attribute1[max(length(attribute6)-1,1)],
        attr6last3 = attribute1[max(length(attribute6)-2,1)],

        attr7last1 = attribute1[max(length(attribute7),1)],
        attr7last2 = attribute1[max(length(attribute7)-1,1)],
        attr7last3 = attribute1[max(length(attribute7)-2,1)],

        attr9last1 = attribute1[max(length(attribute9),1)],
        attr9last2 = attribute1[max(length(attribute9)-1,1)],
        attr9last3 = attribute1[max(length(attribute9)-2,1)],
```

```
        #mean over last 3 days
        attr1mean_3d = mean(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2mean_3d = mean(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3mean_3d = mean(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4mean_3d = mean(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5mean_3d = mean(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6mean_3d = mean(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7mean_3d = mean(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9mean_3d = mean(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        #sd over last 3 days
        attr1sd_3d = sd(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2sd_3d = sd(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3sd_3d = sd(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4sd_3d = sd(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5sd_3d = sd(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6sd_3d = sd(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7sd_3d = sd(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9sd_3d = sd(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        # max over last 3 days
        attr1max_3d = max(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2max_3d = max(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3max_3d = max(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4max_3d = max(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5max_3d = max(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6max_3d = max(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7max_3d = max(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9max_3d = max(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        # min over last 3 days
        attr1min_3d = min(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2min_3d = min(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3min_3d = min(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4min_3d = min(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5min_3d = min(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6min_3d = min(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7min_3d = min(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9min_3d = min(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        #difference between 1st value and last 3
        attr1diffto_last1 = attribute1[1] - attribute1[max(length(attribute1),1)],
        attr1diffto_last2 = attribute1[1] - attribute1[max(length(attribute1)-1,1)],
        attr1diffto_last3 = attribute1[1] - attribute1[max(length(attribute1)-2,1)],

        attr2diffto_last1 = attribute2[1] - attribute2[max(length(attribute1),1)],
        attr2diffto_last2 = attribute2[1] - attribute2[max(length(attribute1)-1,1)],
        attr2diffto_last3 = attribute2[1] - attribute2[max(length(attribute1)-2,1)],

        attr3diffto_last1 = attribute3[1] - attribute3[max(length(attribute1),1)],
        attr3diffto_last2 = attribute3[1] - attribute3[max(length(attribute1)-1,1)],
        attr3diffto_last3 = attribute3[1] - attribute3[max(length(attribute1)-2,1)],
```

```
        attr4diffto_last1 = attribute4[1] - attribute4[max(length(attribute1),1)],
        attr4diffto_last2 = attribute4[1] - attribute4[max(length(attribute1)-1,1)],
        attr4diffto_last3 = attribute4[1] - attribute4[max(length(attribute1)-2,1)],

        attr5diffto_last1 = attribute5[1] - attribute5[max(length(attribute1),1)],
        attr5diffto_last2 = attribute5[1] - attribute5[max(length(attribute1)-1,1)],
        attr5diffto_last3 = attribute5[1] - attribute5[max(length(attribute1)-2,1)],

        attr6diffto_last1 = attribute6[1] - attribute6[max(length(attribute1),1)],
        attr6diffto_last2 = attribute6[1] - attribute6[max(length(attribute1)-1,1)],
        attr6diffto_last3 = attribute6[1] - attribute6[max(length(attribute1)-2,1)],

        attr7diffto_last1 = attribute7[1] - attribute7[max(length(attribute1),1)],
        attr7diffto_last2 = attribute7[1] - attribute7[max(length(attribute1)-1,1)],
        attr7diffto_last3 = attribute7[1] - attribute7[max(length(attribute1)-2,1)],

        attr9diffto_last1 = attribute9[1] - attribute9[max(length(attribute1),1)],
        attr9diffto_last2 = attribute9[1] - attribute9[max(length(attribute1)-1,1)],
        attr9diffto_last3 = attribute9[1] - attribute9[max(length(attribute1)-2,1)],

        #count of logs (lifespan)
        logs_n = n(),

        #count of days (lifespan)
        n_days = as.integer(sum(diff(date))),

        failure = max(failure),
        last_date = max(date)

        )
```

```
ncols <- length(names(device_dataset))
trainformula <- as.formula(paste('failure', paste(names(device_dataset)[c(2:(ncols-2))],collapse=' + ')))
trainformula
```

```
## failure ~ attr1mean + attr2mean + attr3mean + attr4mean + attr5mean +
##     attr6mean + attr7mean + attr9mean + attr1sd + attr2sd + attr3sd +
##     attr4sd + attr5sd + attr6sd + attr7sd + attr9sd + attr1max +
##     attr2max + attr3max + attr4max + attr5max + attr6max + attr7max +
##     attr9max + attr1min + attr2min + attr3min + attr4min + attr5min +
##     attr6min + attr7min + attr9min + attr1last1 + attr1last2 +
##     attr1last3 + attr2last1 + attr2last2 + attr2last3 + attr3last1 +
##     attr3last2 + attr3last3 + attr4last1 + attr4last2 + attr4last3 +
##     attr5last1 + attr5last2 + attr5last3 + attr6last1 + attr6last2 +
##     attr6last3 + attr7last1 + attr7last2 + attr7last3 + attr9last1 +
##     attr9last2 + attr9last3 + attr1mean_3d + attr2mean_3d + attr3mean_3d +
##     attr4mean_3d + attr5mean_3d + attr6mean_3d + attr7mean_3d +
##     attr9mean_3d + attr1sd_3d + attr2sd_3d + attr3sd_3d + attr4sd_3d +
##     attr5sd_3d + attr6sd_3d + attr7sd_3d + attr9sd_3d + attr1max_3d +
##     attr2max_3d + attr3max_3d + attr4max_3d + attr5max_3d + attr6max_3d +
##     attr7max_3d + attr9max_3d + attr1min_3d + attr2min_3d + attr3min_3d +
##     attr4min_3d + attr5min_3d + attr6min_3d + attr7min_3d + attr9min_3d +
##     attr1diffto_last1 + attr1diffto_last2 + attr1diffto_last3 +
##     attr2diffto_last1 + attr2diffto_last2 + attr2diffto_last3 +
```

```
##     attr3diffto_last1 + attr3diffto_last2 + attr3diffto_last3 +
##     attr4diffto_last1 + attr4diffto_last2 + attr4diffto_last3 +
##     attr5diffto_last1 + attr5diffto_last2 + attr5diffto_last3 +
##     attr6diffto_last1 + attr6diffto_last2 + attr6diffto_last3 +
##     attr7diffto_last1 + attr7diffto_last2 + attr7diffto_last3 +
##     attr9diffto_last1 + attr9diffto_last2 + attr9diffto_last3 +
##     logs_n + n_days
```

3.2 Gradient Boosted Trees and experience setup

We will try to fit a tree boosted model as it is shown they perform very well on unbalanced data. We will be using `caret` package to setup our experience and do the training. Our metric that we will be using to optimize our training will ROC as it better suited unbalanced data-sets.

First we randomize the rows (devices) in our data-set, and use 80% as a training set and 20% as a holdout set to get an unbiased evaluation of the model.

3.3 Parameter tuning

We will do a grid search to find the optimal parameters using 3-folds 5-repeated CV. After that we train the model using the best parameters on all our training data-set (whole 80%) and do our evaluation on the holdout set.

For a gradient boosting machine (GBM) model, there are three main tuning parameters:

1- number of iterations, i.e. trees, (called n.trees in the gbm function) *gridsearch: 1,2,3,4,5*

2- complexity of the tree, called interaction.depth *gridsearch: 50,100,150,200,250,300,350,400*

3- learning rate: how quickly the algorithm adapts, called shrinkage *gridsearch: .001,.01,.1*

4- the minimum number of training set samples in a node to commence splitting (n.minobsinnode)

The optimal parameters within this space (n_trees=300, depth=2, lr=0.1) give us a 0.9691 AUC (Repeated CV).

```
device_dataset <- data.table(device_dataset)
set.seed(42)
device_dataset <- device_dataset %>% sample_frac()
device_dataset[ , failure := as.factor(failure)]
levels(device_dataset$failure) <- c("NF", "F")
# summary(device_dataset)

inTraining <- createDataPartition(device_dataset$failure, p = .8, list = FALSE)
training <- device_dataset[ inTraining,]
testing  <- device_dataset[-inTraining,]

cat("Training- number of failures:",sum(training$failure == 'F'))
```

```
## Training- number of failures: 85
```

```
cat("Training- failures percentage:",formatC(100 * mean(training$failure == 'F'),
                                             format = 'f', digits = 3), "%")
```

```
## Training- failures percentage: 9.091 %
```

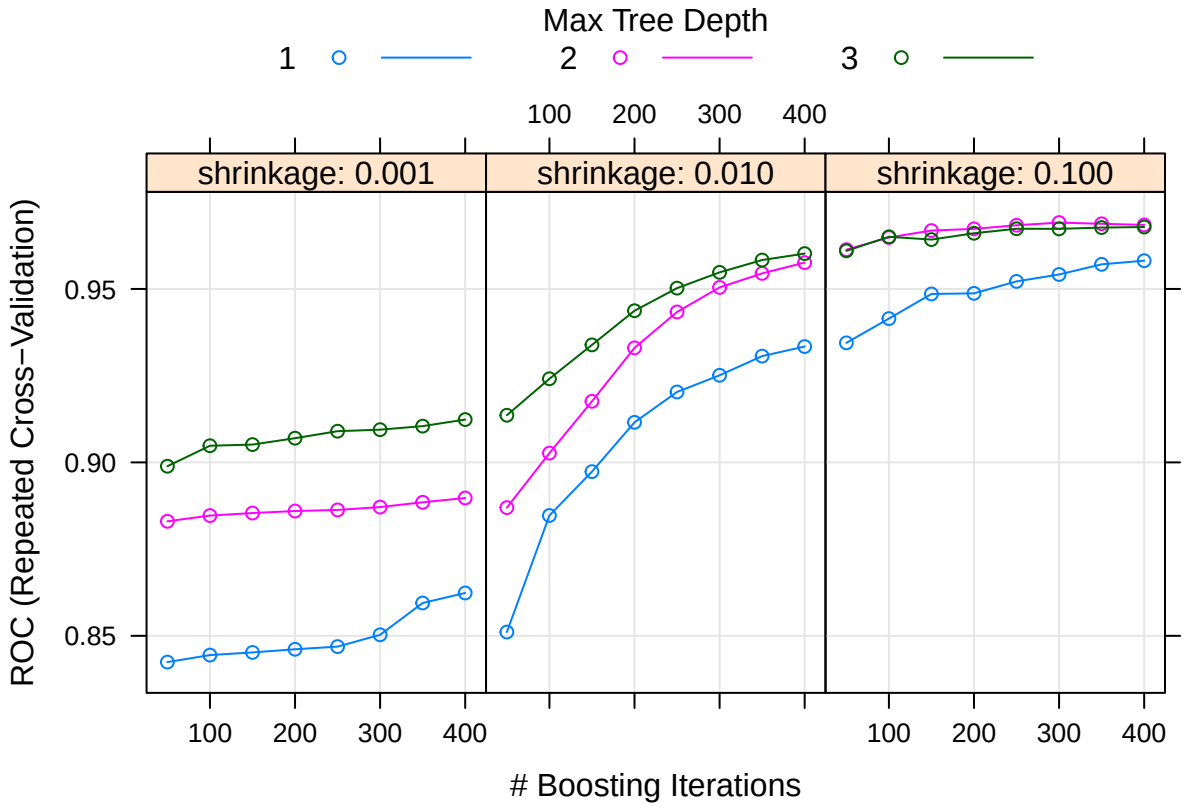
```
cat("Testing- number of failures:",sum(testing$failure == 'F'))
```

```
## Testing- number of failures: 21
```

```
cat("Testing- failure percentage:",formatC(100 * mean(testing$failure == 'F'),
                                           format = 'f', digits = 3), "%")
```

```
## Testing- failure percentage: 9.013 %
```

```
if (file.exists('gbmFit1')){ #to not run gridsearch each time we Knit
  gbmFit1 = readRDS('gbmFit1')
  # gbmFit1
  print(plot(gbmFit1))
  gbmFit1$bestTune
}else{
  fitControl <- trainControl(## 3-fold 5-repeated CV
    method = "repeatedcv",
    number = 3,
    repeats = 5,
    classProbs = TRUE,
    summaryFunction = twoClassSummary
  )

  gbmGrid <-  expand.grid(interaction.depth = (1:3),
                          n.trees = (1:8)*50,
                          shrinkage = c(0.1,.01,.001),
                          n.minobsinnode = 20)
  set.seed(4242)
  gbmFit1 <- train(trainformula,
                   data = training,
                   method = "gbm",
                   trControl = fitControl,
                   metric = "ROC",
                   tuneGrid = gbmGrid,
                   verbose = FALSE)
  # summary(gbmFit1$results$ROC)
  plot(gbmFit1)
  gbmFit1$bestTune
  saveRDS(gbmFit1, "gbmFit1")
}
```




```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 62     300                 2       0.1             20
```

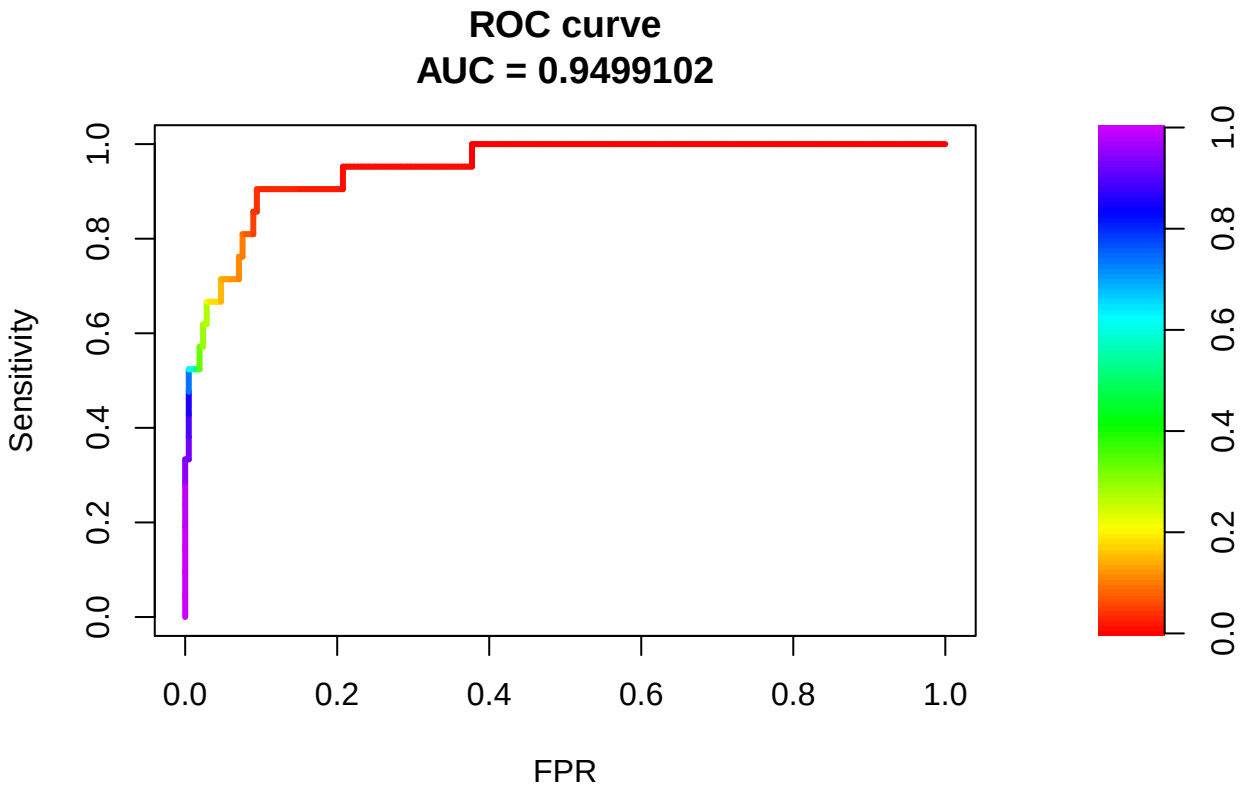
3.4 Holdout dataset

To get an idea on how well the model will perform on data it never seen, we train on all the training data available use that model to generate predictions on the holdout data-set and calculate the ROC again.

We get .9499 AUC which is lower than the training value but still very close. This indicates that the model was able to generalize very well on unseen data.

```
#training on all data and variable explanation
fitControl <- trainControl(method = "none", classProbs = TRUE)
set.seed(925)
gbmFitAll_caret <- train(trainformula,
                  data = training,
                  method = "gbm",
                  trControl = fitControl,
                  verbose = FALSE,
                  tuneGrid = data.frame(interaction.depth = gbmFit1$bestTune$interaction.depth,
                                        n.trees = gbmFit1$bestTune$n.trees,
                                        shrinkage = gbmFit1$bestTune$shrinkage,
                                        n.minobsinnode = gbmFit1$bestTune$n.minobsinnode),
                  metric = "ROC")

#Score on Holdout Dataset
holdout_pred_gbm1 <- predict(gbmFitAll_caret,newdata = testing, type="prob")$F
```

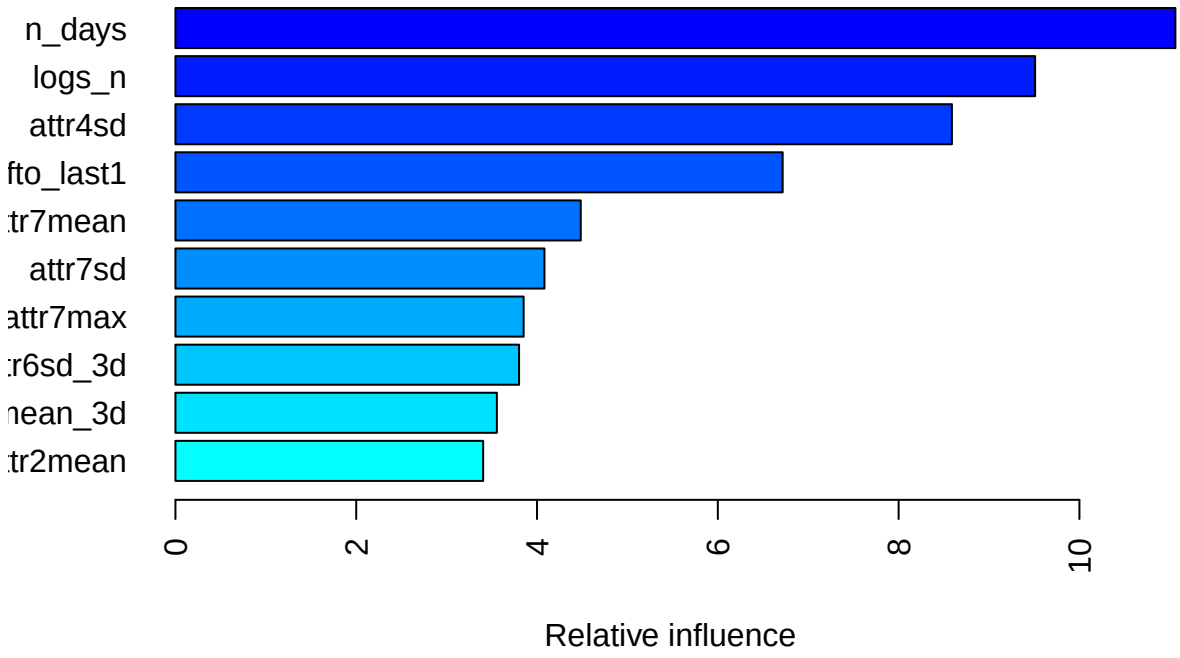
```
fg <- holdout_pred_gbm1[testing$failure == 'F']
bg <- holdout_pred_gbm1[testing$failure == 'NF']

# ROC Curve
roc <- roc.curve(scores.class0 = fg, scores.class1 = bg, curve = T)
plot(roc)
```



3.5 Feature Importance

3.5.1 Variable importance

After re-running our final model we likely want to understand the variables that have the largest influence on failure detection. And as expected from the attribute analysis before **logs_n** (proxy for the lifespan of device) and **attribute4** and **attribute7** are the most important.



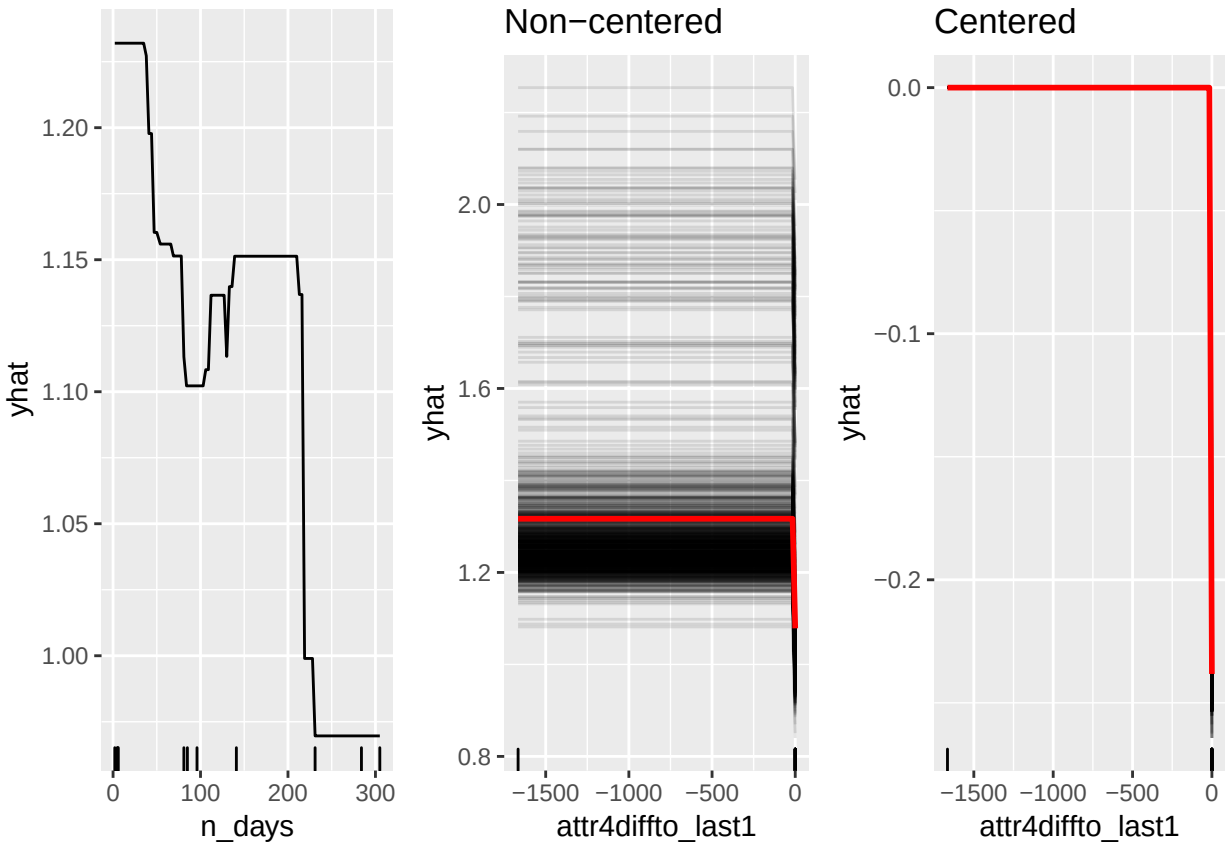
3.5.2 Partial dependence plots

To understand more about this important predictors we can look at how the predicted response (failure) as we vary them. One way of doing this is plotting PDP (Partial dependence plots) and ICE curves (Individual Conditional Expectation plots). The first give the general trend over the data and the later a more granular view (by observation)

We do so for **n_days** (PDP) and we notice how failure prediction actually change as function of the device lifespan. The ICE curves on **att4diffto_last1** confirms again the attribute analysis where we see that all observation follow a common trend as **attribute4** value differs from value at first day.

```
pdp_n_days = gbmFitAll %>%
  partial(pred.var = "n_days", n.trees = gbmFitAll$n.trees, grid.resolution = 100) %>%
  autoplot(rug = TRUE, train = training)

ice1 <- gbmFitAll %>%
  partial(
    pred.var = "attr4diffto_last1",
    n.trees = gbmFitAll$n.trees,
    grid.resolution = 100,
    ice = TRUE
  ) %>%
  autoplot(rug = TRUE, train = training, alpha = .1) +
  ggtitle("Non-centered")

ice2 <- gbmFitAll %>%
  partial(
    pred.var = "attr4diffto_last1",
    n.trees = gbmFitAll$n.trees,
    grid.resolution = 100,
    ice = TRUE
  ) %>%
  autoplot(rug = TRUE, train = training, alpha = .1, center = TRUE) +
```

```
  ggtitle("Centered")

gridExtra::grid.arrange(pdp_n_days, ice1, ice2, nrow = 1)
```

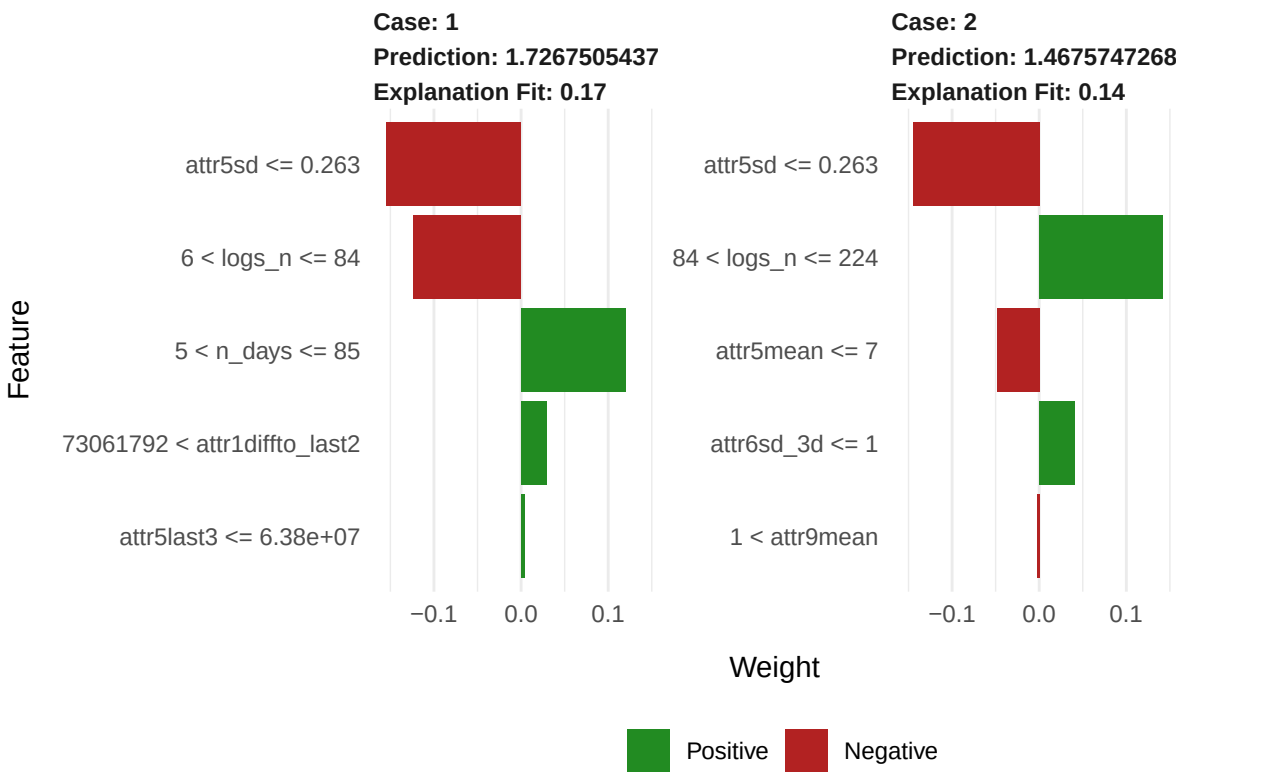


3.5.3 LIME

In order to understand why a prediction resulted in a failure or non-failure for a single observation we can use the LIME procedure (Local Interpretable Model-Agnostic Explanations).

Applying it to two observations (failure and non-failure). The results show the predicted value (Case 1: NF, Case 2: NF), local model fit (both are relatively poor), and the most influential variables driving the predicted value for each observation.

```
model_type.gbm <- function(x, ...) {
  return("regression")
}

predict_model.gbm <- function(x, newdata, ...) {
  pred <- predict(x, newdata, n.trees = x$n.trees)
  return(as.data.frame(pred))
}
# apply LIME
explainer <- lime(training, gbmFitAll)
# get a few observations to perform local interpretation on
local_obs <- testing[8:9, ]
explanation <- explain(local_obs, explainer, n_features = 5)
plot_features(explanation)
```



3.6 Evaluation

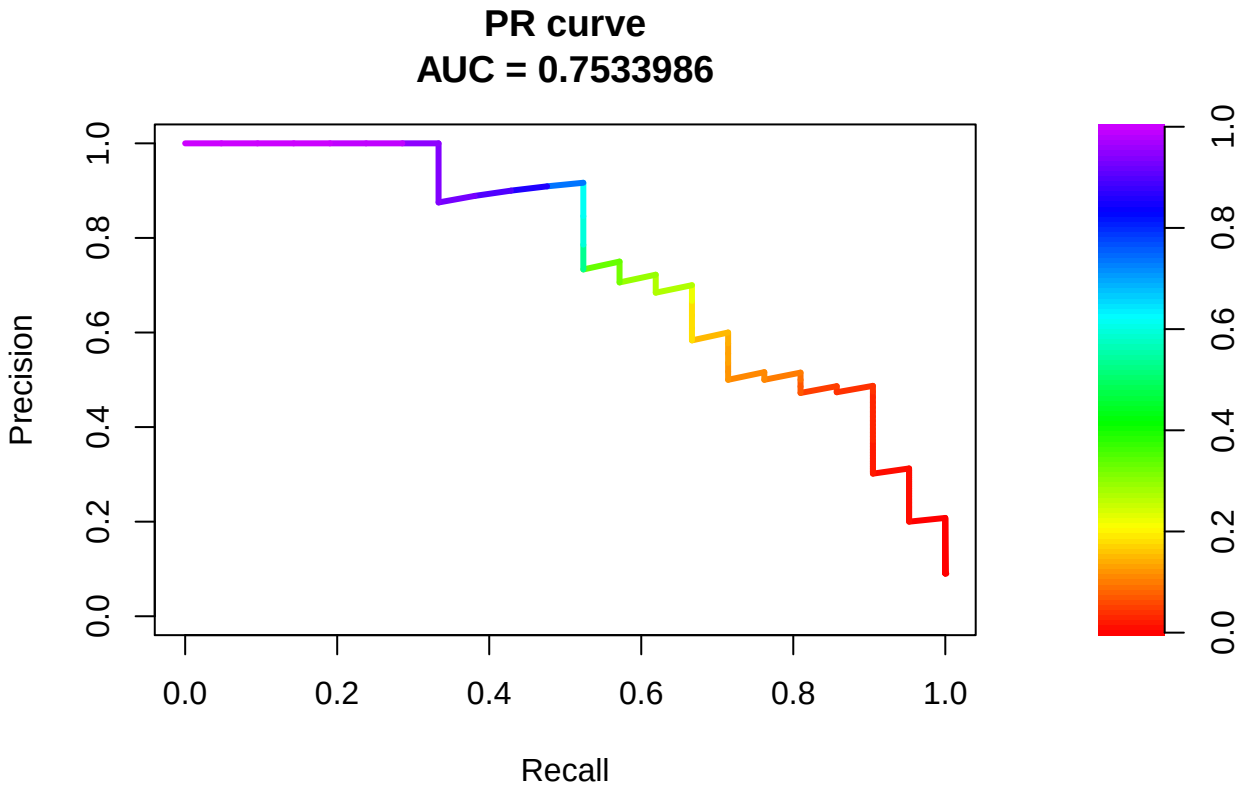
3.6.1 PR curve and cutoff selection:

To evaluate the model let's plot the Precision/Recall curve and decide on a sensible cutoff

Our main goals are:

- **Goal 1** : minimize False Negatives (actual failures we didn't classify correctly), which amounts to increase the Recall.
- **Goal 2** : minimize False Positives (actual working device that we did classify as failing), which amount to increase the Precision.

Choosing a *cutoff=0.2* we manage to classify 66% of failing devices with a 66% precision.



	NF	F
0	205	7
1	7	14

cutoff	precision	recall	accuracy	F1	auc
0.2	66.67%	66.67%	93.99%	66.67%	94.99%

3.6.2 'Maintenance cost' avoided and 'Loss suffered'

Achieving our **Goal 1** means that we want to minimize the number of failures we didn't catch, i.e reducing the *loss suffered* when a device fail (might be calculated by number of days to fix device, revenue lost, repair costs...).

Achieving our **Goal 2** means that we want to minimize the number of non-failures we classified as failures, i.e reducing the *maintenance cost* by not doing maintenance on on devices that are still functional.

If we had more information about the *loss suffered* and the *maintenance cost* we can determine a decision boundary that will optimize our overall cost.

For example if the cost of repairing a device is **twice** the cost of doing maintenance. the overall cost will be:

$$Cost = FalsePositive * losssuffered + FalseNegative * maintenancecost$$

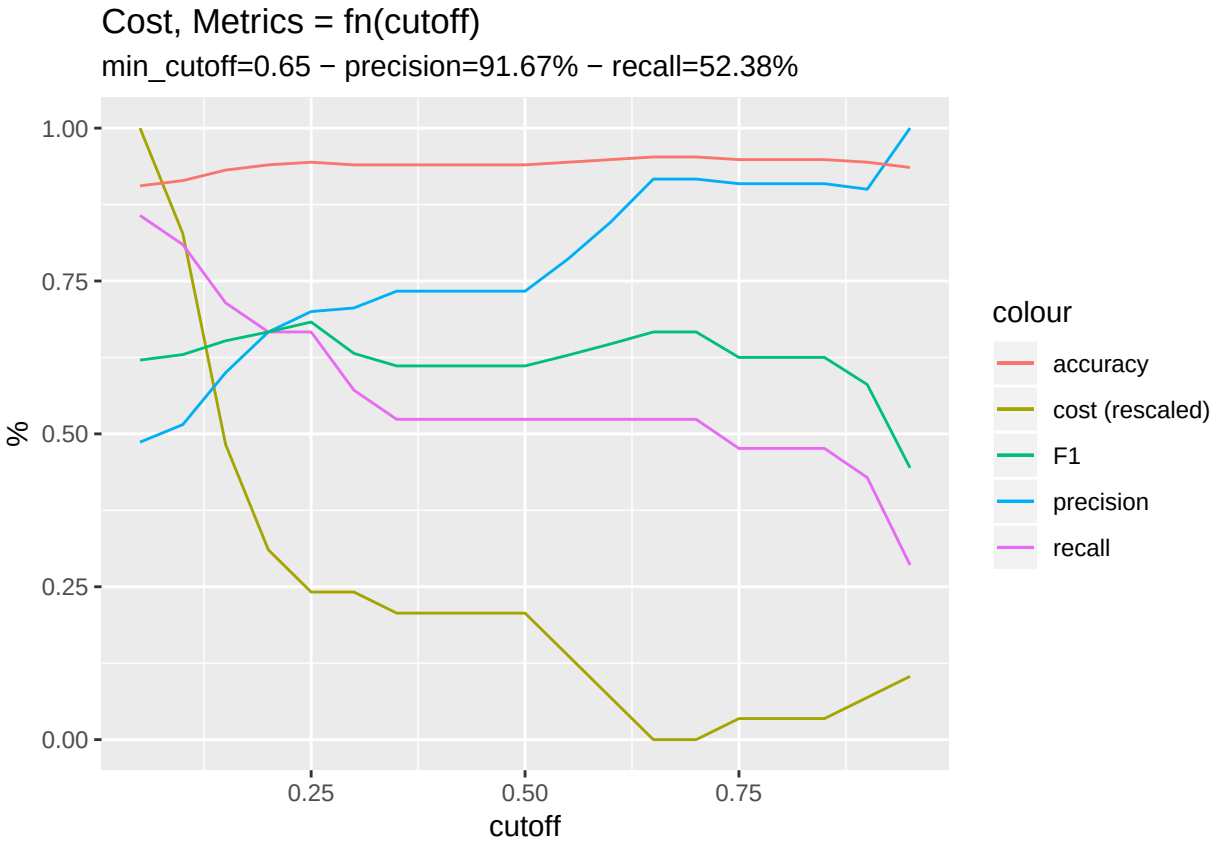
$$Cost = (2 * FalsePositive + FalseNegative) * maintenancecost$$

Since *maintenance cost* is fixed, drawing $2 * FalsePositive + FalseNegative$ function of the cutoff will help us find the optimal cutoff value.

In this case:

- The cutoff that minimize the overall cost is **0.65**.
- In this case we sacrifice some Recall, catching only ~52% of the failures as doing so is more expensive.
- Bu we gain in Precision, accurately maintaing ~92% of failing devices as we can afford to (maintenance is cheaper).

```
range01 <- function(x){(x-min(x))/(max(x)-min(x))}
cost <- data.table()
for (cut in seq_len(20) * .05) {
  pred_cut <- factor(ifelse(holdout_pred_gbm1 >= cut, 1, 0))
  cm <- table(pred_cut, testing$failure)
  if (nrow(cm) == 2) {
    cm_precision <- cm[2,2] / sum(cm[2,])
    cm_recall <- cm[2,2] / sum(cm[ ,2])
    cm_accuracy <- (cm[1,1] + cm[2,2]) / sum(cm)
    cm_f1 <- 2 * cm_precision * cm_recall / (cm_precision + cm_recall)
    cost <- rbind(cost, data.table(cutoff = cut,
                                   cost_at_cutoff = 2 * cm[2,1] + cm[1,2],
                                   precision = percent(cm_precision),
                                   recall = percent(cm_recall),
                                   accuracy = percent(cm_accuracy),
                                   F1 = percent(cm_f1)))
  }
}
ggplot(data = cost , aes(cutoff)) +
        geom_line(aes(y=range01(cost_at_cutoff), color='cost (rescaled)')) +
        geom_line(aes(y=precision, color='precision')) +
        geom_line(aes(y=recall, color='recall')) +
        geom_line(aes(y=accuracy, color='accuracy')) +
        geom_line(aes(y=F1, color='F1')) +
        labs(title = 'Cost, Metrics = fn(cutoff) ',
             subtitle = paste0('min_cutoff=', cost$cutoff[which.min(cost$cost_at_cutoff)], ' - ',
                       'precision=', cost$precision[which.min(cost$cost_at_cutoff)], ' - ',
                       'recall=', cost$recall[which.min(cost$cost_at_cutoff)]),
             x = "cutoff", y = "%")
```



4 Logs/Device Dataset

4.1 Lag & Date Features

In this approach we will make use of the information contained in the time dimension by computing rolling aggregate measures (mean and sd) over the last 3 days lag window for every 3 days.

- Example features for *attribute1*: **attr1mean_rolling3d**, **attr1sd_rolling3d**
- We will also add **weekday** and **monthweeks** as features as they might capture some seasonality in the data.

```
dt[ , failure := as.integer(failure)]
logdevice_dataset <- dt[device %in% failed_device_names] %>%
  arrange(device, date) %>%
  dplyr::group_by(device) %>%
  dplyr::mutate(

          #mean over device lifespan
          attr1mean = mean(attribute1),
          attr2mean = mean(attribute2),
          attr3mean = mean(attribute3),
          attr4mean = mean(attribute4),
          attr5mean = mean(attribute5),
          attr6mean = mean(attribute6),
          attr7mean = mean(attribute7),
```

```
        attr9mean = mean(attribute9),

        #sd over device lifespan
        attr1sd = sd(attribute1),
        attr2sd = sd(attribute2),
        attr3sd = sd(attribute3),
        attr4sd = sd(attribute4),
        attr5sd = sd(attribute5),
        attr6sd = sd(attribute6),
        attr7sd = sd(attribute7),
        attr9sd = sd(attribute9),

        #max over device lifespan
        attr1max = max(attribute1),
        attr2max = max(attribute2),
        attr3max = max(attribute3),
        attr4max = max(attribute4),
        attr5max = max(attribute5),
        attr6max = max(attribute6),
        attr7max = max(attribute7),
        attr9max = max(attribute9),

        #min over device lifespan
        attr1min = min(attribute1),
        attr2min = min(attribute2),
        attr3min = min(attribute3),
        attr4min = min(attribute4),
        attr5min = min(attribute5),
        attr6min = min(attribute6),
        attr7min = min(attribute7),
        attr9min = min(attribute9),

        #last 3 values
        attr1last1 = attribute1[max(length(attribute1),1)],
        attr1last2 = attribute1[max(length(attribute1)-1,1)],
        attr1last3 = attribute1[max(length(attribute1)-2,1)],

        attr2last1 = attribute1[max(length(attribute2),1)],
        attr2last2 = attribute1[max(length(attribute2)-1,1)],
        attr2last3 = attribute1[max(length(attribute2)-2,1)],

        attr3last1 = attribute1[max(length(attribute3),1)],
        attr3last2 = attribute1[max(length(attribute3)-1,1)],
        attr3last3 = attribute1[max(length(attribute3)-2,1)],

        attr4last1 = attribute1[max(length(attribute4),1)],
        attr4last2 = attribute1[max(length(attribute4)-1,1)],
        attr4last3 = attribute1[max(length(attribute4)-2,1)],

        attr5last1 = attribute1[max(length(attribute5),1)],
        attr5last2 = attribute1[max(length(attribute5)-1,1)],
        attr5last3 = attribute1[max(length(attribute5)-2,1)],
```

```
        attr6last1 = attribute1[max(length(attribute6),1)],
        attr6last2 = attribute1[max(length(attribute6)-1,1)],
        attr6last3 = attribute1[max(length(attribute6)-2,1)],

        attr7last1 = attribute1[max(length(attribute7),1)],
        attr7last2 = attribute1[max(length(attribute7)-1,1)],
        attr7last3 = attribute1[max(length(attribute7)-2,1)],

        attr9last1 = attribute1[max(length(attribute9),1)],
        attr9last2 = attribute1[max(length(attribute9)-1,1)],
        attr9last3 = attribute1[max(length(attribute9)-2,1)],

        #mean over last 3 days
        attr1mean_3d = mean(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2mean_3d = mean(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3mean_3d = mean(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4mean_3d = mean(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5mean_3d = mean(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6mean_3d = mean(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7mean_3d = mean(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9mean_3d = mean(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        #sd over last 3 days
        attr1sd_3d = sd(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2sd_3d = sd(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3sd_3d = sd(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4sd_3d = sd(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5sd_3d = sd(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6sd_3d = sd(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7sd_3d = sd(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9sd_3d = sd(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        # max over last 3 days
        attr1max_3d = max(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2max_3d = max(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3max_3d = max(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4max_3d = max(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5max_3d = max(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6max_3d = max(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7max_3d = max(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9max_3d = max(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        # min over last 3 days
        attr1min_3d = min(attribute1[max(length(attribute1)-2,1):max(length(attribute1),1)]),
        attr2min_3d = min(attribute2[max(length(attribute2)-2,1):max(length(attribute2),1)]),
        attr3min_3d = min(attribute3[max(length(attribute3)-2,1):max(length(attribute3),1)]),
        attr4min_3d = min(attribute4[max(length(attribute4)-2,1):max(length(attribute4),1)]),
        attr5min_3d = min(attribute5[max(length(attribute5)-2,1):max(length(attribute5),1)]),
        attr6min_3d = min(attribute6[max(length(attribute6)-2,1):max(length(attribute6),1)]),
        attr7min_3d = min(attribute7[max(length(attribute7)-2,1):max(length(attribute7),1)]),
        attr9min_3d = min(attribute9[max(length(attribute9)-2,1):max(length(attribute9),1)]),

        #difference between 1st value and last 3
```

```
      attr1diffto_last1 = attribute1[1] - attribute1[max(length(attribute1),1)],
      attr1diffto_last2 = attribute1[1] - attribute1[max(length(attribute1)-1,1)],
      attr1diffto_last3 = attribute1[1] - attribute1[max(length(attribute1)-2,1)],

      attr2diffto_last1 = attribute2[1] - attribute2[max(length(attribute1),1)],
      attr2diffto_last2 = attribute2[1] - attribute2[max(length(attribute1)-1,1)],
      attr2diffto_last3 = attribute2[1] - attribute2[max(length(attribute1)-2,1)],

      attr3diffto_last1 = attribute3[1] - attribute3[max(length(attribute1),1)],
      attr3diffto_last2 = attribute3[1] - attribute3[max(length(attribute1)-1,1)],
      attr3diffto_last3 = attribute3[1] - attribute3[max(length(attribute1)-2,1)],

      attr4diffto_last1 = attribute4[1] - attribute4[max(length(attribute1),1)],
      attr4diffto_last2 = attribute4[1] - attribute4[max(length(attribute1)-1,1)],
      attr4diffto_last3 = attribute4[1] - attribute4[max(length(attribute1)-2,1)],

      attr5diffto_last1 = attribute5[1] - attribute5[max(length(attribute1),1)],
      attr5diffto_last2 = attribute5[1] - attribute5[max(length(attribute1)-1,1)],
      attr5diffto_last3 = attribute5[1] - attribute5[max(length(attribute1)-2,1)],

      attr6diffto_last1 = attribute6[1] - attribute6[max(length(attribute1),1)],
      attr6diffto_last2 = attribute6[1] - attribute6[max(length(attribute1)-1,1)],
      attr6diffto_last3 = attribute6[1] - attribute6[max(length(attribute1)-2,1)],

      attr7diffto_last1 = attribute7[1] - attribute7[max(length(attribute1),1)],
      attr7diffto_last2 = attribute7[1] - attribute7[max(length(attribute1)-1,1)],
      attr7diffto_last3 = attribute7[1] - attribute7[max(length(attribute1)-2,1)],

      attr9diffto_last1 = attribute9[1] - attribute9[max(length(attribute1),1)],
      attr9diffto_last2 = attribute9[1] - attribute9[max(length(attribute1)-1,1)],
      attr9diffto_last3 = attribute9[1] - attribute9[max(length(attribute1)-2,1)],

    #mean over last 3 days
      attr1mean_rolling3d = rollapply(attribute1, width = 3, FUN = mean, align = "right", fill = NA)
      attr2mean_rolling3d = rollapply(attribute2, width = 3, FUN = mean, align = "right", fill = NA)
      attr3mean_rolling3d = rollapply(attribute3, width = 3, FUN = mean, align = "right", fill = NA)
      attr4mean_rolling3d = rollapply(attribute4, width = 3, FUN = mean, align = "right", fill = NA)
      attr5mean_rolling3d = rollapply(attribute5, width = 3, FUN = mean, align = "right", fill = NA)
      attr6mean_rolling3d = rollapply(attribute6, width = 3, FUN = mean, align = "right", fill = NA)
      attr7mean_rolling3d = rollapply(attribute7, width = 3, FUN = mean, align = "right", fill = NA)
      attr9mean_rolling3d = rollapply(attribute9, width = 3, FUN = mean, align = "right", fill = NA)

      #sd over last 3 days
      attr1sd_rolling3d = rollapply(attribute1, width = 3, FUN = sd, align = "right", fill = NA),
      attr2sd_rolling3d = rollapply(attribute2, width = 3, FUN = sd, align = "right", fill = NA),
      attr3sd_rolling3d = rollapply(attribute3, width = 3, FUN = sd, align = "right", fill = NA),
      attr4sd_rolling3d = rollapply(attribute4, width = 3, FUN = sd, align = "right", fill = NA),
      attr5sd_rolling3d = rollapply(attribute5, width = 3, FUN = sd, align = "right", fill = NA),
      attr6sd_rolling3d = rollapply(attribute6, width = 3, FUN = sd, align = "right", fill = NA),
      attr7sd_rolling3d = rollapply(attribute7, width = 3, FUN = sd, align = "right", fill = NA),
      attr9sd_rolling3d = rollapply(attribute9, width = 3, FUN = sd, align = "right", fill = NA),

      # #date variables
```

```
        monthweeks = paste0('week_',ceiling(day(date) / 7)),
        weekdays = weekdays(date),

        #count of logs (lifespan)
        logs_n = n(),

        #count of days (lifespan)
        n_days = as.integer(sum(diff(date)))

        ) %>%
    filter(!is.na(attr1mean_rolling3d)) %>% ungroup()
```

4.2 Labeling

4.2.1 As a classification problem

labeling is done by taking a time window prior to the failure of an asset and labeling the feature records that fall into that window as "about to fail ('F')" while labeling all other records as "normal ('NF')". This time window should be picked according to the business case where in some situations it may be enough to predict failures hours in advance while in others days or weeks maybe needed to allow for the arrival of parts to be replaced as an example.

For this case we will consider a time window of 7 days and limit ourselves again to devices without any failures as a way of addressing the imbalance in the data-set (even if not necessary for classification). Since the ratio of failures in this case is 1%, the new data-set will have *.1 * time_window = 7%* failure ratio.

4.2.2 As a regression problem

In this approach we will fit a Regression model to compute the remaining useful life (RUL) of a device. This number denotes the period of time remaining before the failure. The model should calculate the RUL of each new example as a continuous number.

4.2.2.1 Label construction for regression

The question here is: "What is the remaining useful life (RUL) of the device?" For each log prior to the failure, calculate the label to be the number of days remaining before the next failure.

For regression, labeling is done with reference to a failure point. Its calculation is not possible without knowing how long the device has survived before a failure. So in contrast to binary classification, assets without any failures in the data cannot be used for modeling.

```
logdevice_dataset = logdevice_dataset %>%
  dplyr::group_by(device) %>%
  dplyr::mutate(failure_6d = ifelse(row_number() >= n() - 6, 1, failure),
                failure_reg = sort(seq_len(n()), decreasing = T)) %>%
  ungroup()
logdevice_dataset = data.table(logdevice_dataset)
```

```
logdevice_dataset[ , failure_6d := as.factor(failure_6d)]
levels(logdevice_dataset$failure_6d) <- c("NF", "F")
cat("#failures:",sum(logdevice_dataset$failure_6d == 'F'),
    "failures ratio:",formatC(100 * mean(logdevice_dataset$failure_6d == 'F'),
                                          format = 'f', digits = 3), "%")
```

```
## #failures: 732 failures ratio: 7.002 %
```

```
## failure_6d ~ attr1mean + attr2mean + attr3mean + attr4mean +
##     attr5mean + attr6mean + attr7mean + attr9mean + attr1sd +
##     attr2sd + attr3sd + attr4sd + attr5sd + attr6sd + attr7sd +
##     attr9sd + attr1max + attr2max + attr3max + attr4max + attr5max +
##     attr6max + attr7max + attr9max + attr1min + attr2min + attr3min +
##     attr4min + attr5min + attr6min + attr7min + attr9min + attr1last1 +
##     attr1last2 + attr1last3 + attr2last1 + attr2last2 + attr2last3 +
##     attr3last1 + attr3last2 + attr3last3 + attr4last1 + attr4last2 +
##     attr4last3 + attr5last1 + attr5last2 + attr5last3 + attr6last1 +
##     attr6last2 + attr6last3 + attr7last1 + attr7last2 + attr7last3 +
##     attr9last1 + attr9last2 + attr9last3 + attr1mean_3d + attr2mean_3d +
##     attr3mean_3d + attr4mean_3d + attr5mean_3d + attr6mean_3d +
##     attr7mean_3d + attr9mean_3d + attr1sd_3d + attr2sd_3d + attr3sd_3d +
##     attr4sd_3d + attr5sd_3d + attr6sd_3d + attr7sd_3d + attr9sd_3d +
##     attr1max_3d + attr2max_3d + attr3max_3d + attr4max_3d + attr5max_3d +
##     attr6max_3d + attr7max_3d + attr9max_3d + attr1min_3d + attr2min_3d +
##     attr3min_3d + attr4min_3d + attr5min_3d + attr6min_3d + attr7min_3d +
##     attr9min_3d + attr1diffto_last1 + attr1diffto_last2 + attr1diffto_last3 +
##     attr2diffto_last1 + attr2diffto_last2 + attr2diffto_last3 +
##     attr3diffto_last1 + attr3diffto_last2 + attr3diffto_last3 +
##     attr4diffto_last1 + attr4diffto_last2 + attr4diffto_last3 +
##     attr5diffto_last1 + attr5diffto_last2 + attr5diffto_last3 +
##     attr6diffto_last1 + attr6diffto_last2 + attr6diffto_last3 +
##     attr7diffto_last1 + attr7diffto_last2 + attr7diffto_last3 +
##     attr9diffto_last1 + attr9diffto_last2 + attr9diffto_last3 +
##     attr1mean_rolling3d + attr2mean_rolling3d + attr3mean_rolling3d +
##     attr4mean_rolling3d + attr5mean_rolling3d + attr6mean_rolling3d +
##     attr7mean_rolling3d + attr9mean_rolling3d + attr1sd_rolling3d +
##     attr2sd_rolling3d + attr3sd_rolling3d + attr4sd_rolling3d +
##     attr5sd_rolling3d + attr6sd_rolling3d + attr7sd_rolling3d +
##     attr9sd_rolling3d + monthweeks + weekdays + logs_n + n_days
```

4.3 Time-dependant splitting

To estimate performance we adopt a time-dependent splitting strategy(testing on examples that are later in time than the training examples) to avoid leaking from train to test.

For a time-dependent split, a point in time is picked and model is trained on examples up to that point in time, and validated on the examples after that point assuming that the future data after the splitting point is not known.

Validation can be performed by picking different split points and examining the performance of the models trained on different time splits. In the following, we use 3 different splitting points to train the model and look at the performances for different splits in the evaluation section.

- **Split point 1** : `2015-07-03`, to train on the first 6 months and test on last 4 months
- **Split point 2** : `2015-08-03`, to train on the first 7 months and test on last 3 months
- **Split point 3** : `2015-09-03`, to train on the first 8 months and test on last 2 months

However, this effects the labeling of features falling into the labeling window right before the split as it is assumed that failure information is not known beyond the splitting cut-off. Due to that, those feature records can not be labeled and will not be used. This also prevents the leaking problem at the splitting point.

In our case the labeling window is 6 days so records within 6 days prior to split point are left out.

```
logdevice_dataset = logdevice_dataset %>% arrange(date)

train6 <- logdevice_dataset[logdevice_dataset$date < "2015-06-28",]
test4 <- logdevice_dataset[logdevice_dataset$date > "2015-07-03",]

train7 <- logdevice_dataset[logdevice_dataset$date <= "2015-07-29",]
test3 <- logdevice_dataset[logdevice_dataset$date > "2015-08-03",]

train8 <- logdevice_dataset[logdevice_dataset$date <= "2015-08-29",]
test2 <- logdevice_dataset[logdevice_dataset$date > "2015-09-03",]
```

```
## train6 - #failures: 568 train6 - failure rate: 5.893 %
## test4 - #failures: 157 failure rate: 23.259 %
## train7 - #failures: 685 - failure rate: 6.725 %
## test3 - #failures: 29 - failure rate: 12.609 %
## train8 - #failures: 711 - failure rate: 6.891 %
## test2 - #failures: 21 - failure rate: 17.355 %
```

4.4 Training and Evalution

4.4.0.1 "Will this device fail in the next 6 days?"

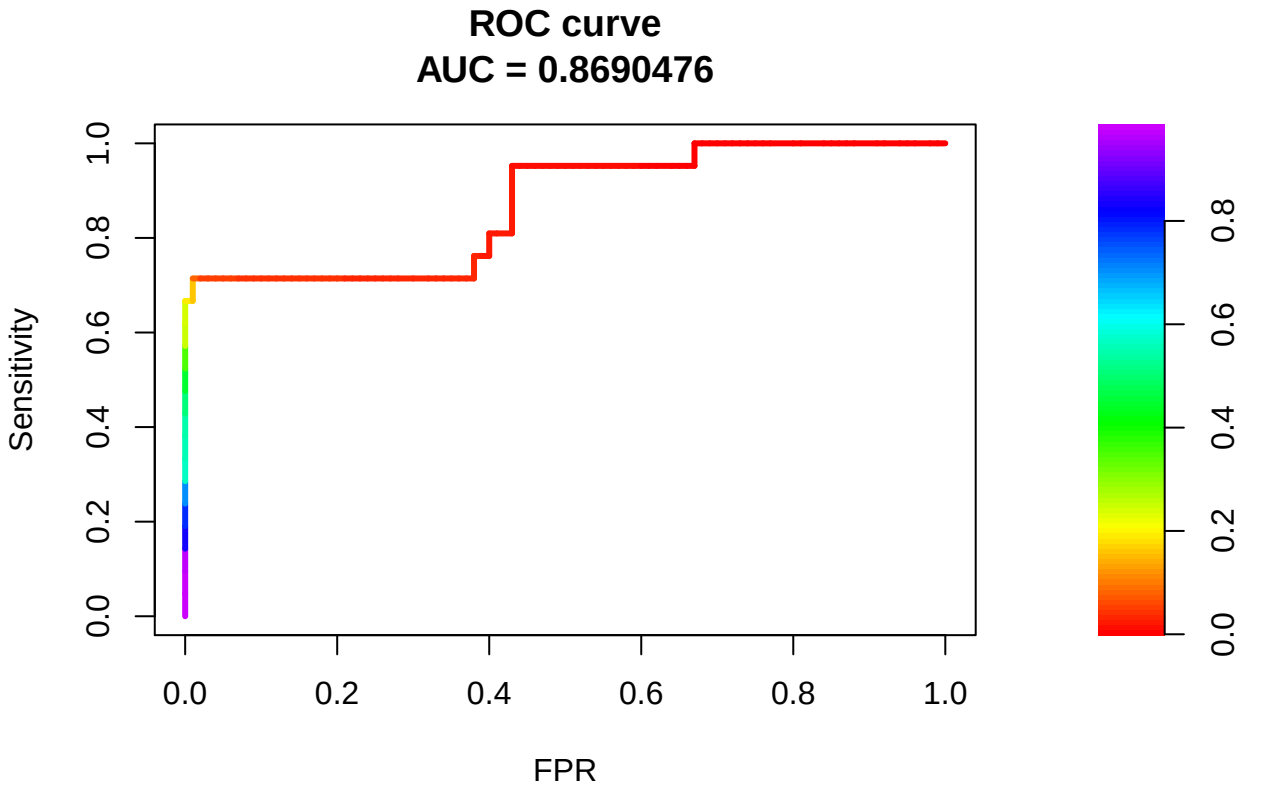
Using this approach on training on more than 8 months we can classify more 65% of our cases (about to fail in the next 6 days or less) with a 90% precision.

This approach can lead to better results if we have more data as the evaluation on the different splits suggests.

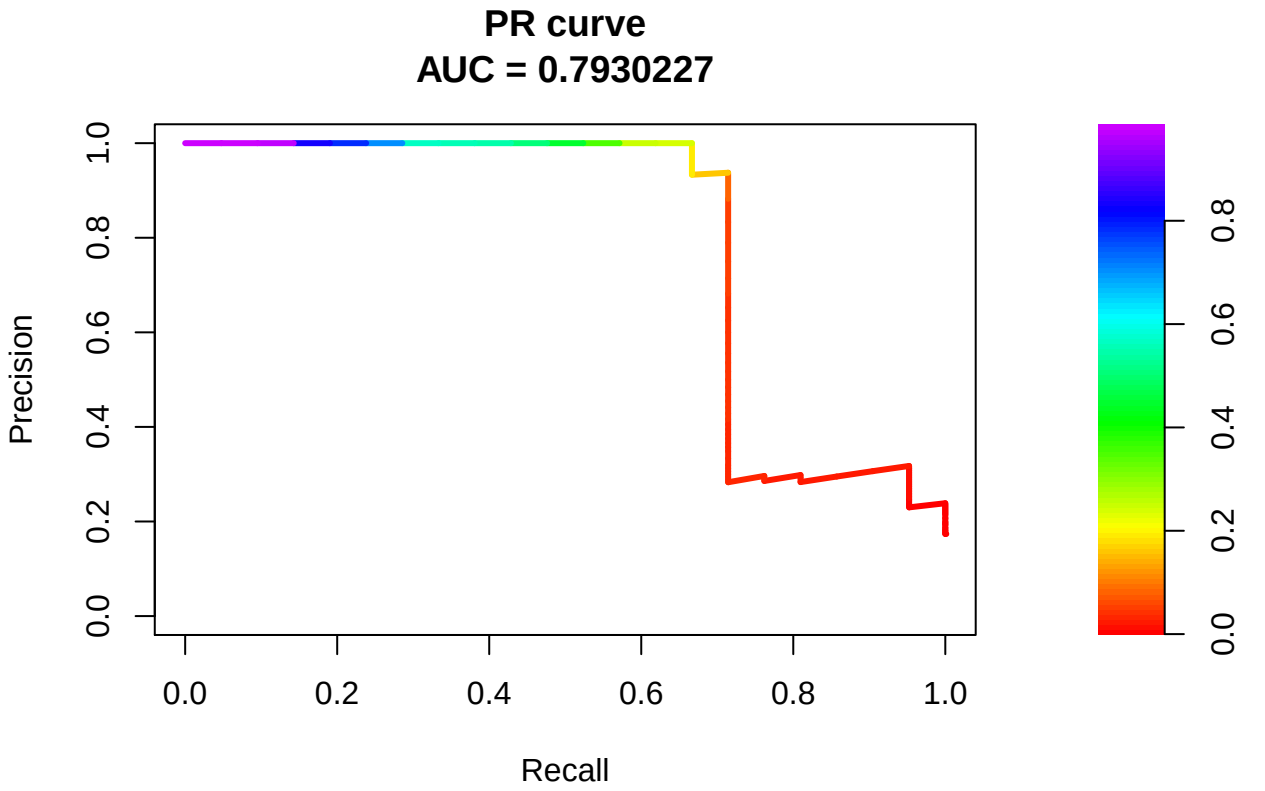
```
if (file.exists('gbmFitSplit1')){ #to not run training each time we Knit
  gbmFitSplit1 = readRDS('gbmFitSplit1')
  gbmFitSplit2 = readRDS('gbmFitSplit2')
  gbmFitSplit3 = readRDS('gbmFitSplit3')
}else{
  fitControl <- trainControl(method = "none", classProbs = TRUE)

  set.seed(925)
  gbmFitSplit1 <- train(trainformula_logdevice_6d,
                   data = train6,
                   method = "gbm",
                   trControl = fitControl,
                   verbose = FALSE,
                   tuneGrid = data.frame(interaction.depth = 5,
                                         n.trees = 300,
                                         shrinkage = .1,
                                         n.minobsinnode = 20),
                   metric = "ROC")

  saveRDS(gbmFitSplit1, "gbmFitSplit1")

  set.seed(925)
  gbmFitSplit2 <- train(trainformula_logdevice_6d,
                   data = train7,
                   method = "gbm",
```

```
                    trControl = fitControl,
                    verbose = FALSE,
                    tuneGrid = data.frame(interaction.depth = 5,
                                          n.trees = 300,
                                          shrinkage = .1,
                                          n.minobsinnode = 20),
                    metric = "ROC")

  saveRDS(gbmFitSplit2, "gbmFitSplit2")

  set.seed(925)
  gbmFitSplit3 <- train(trainformula_logdevice_6d,
                    data = train8,
                    method = "gbm",
                    trControl = fitControl,
                    verbose = FALSE,
                    tuneGrid = data.frame(interaction.depth = 5,
                                          n.trees = 300,
                                          shrinkage = .1,
                                          n.minobsinnode = 20),
                    metric = "ROC")

  saveRDS(gbmFitSplit3, "gbmFitSplit3")
}
```

```
holdout_pred_split1 <- predict(gbmFitSplit1, newdata = test4, type="prob")$F
holdout_pred_split2 <- predict(gbmFitSplit2, newdata = test3, type="prob")$F
holdout_pred_split3 <- predict(gbmFitSplit3, newdata = test2, type="prob")$F

# fg <- holdout_pred_split1[test4$failure_6d == 'F']
# bg <- holdout_pred_split1[test4$failure_6d == 'NF']
# plot(roc.curve(scores.class0 = fg, scores.class1 = bg, curve = T))
# plot(pr.curve(scores.class0 = fg, scores.class1 = bg, curve = T))
# fg <- holdout_pred_split2[test3$failure_6d == 'F']
# bg <- holdout_pred_split2[test3$failure_6d == 'NF']
# plot(roc.curve(scores.class0 = fg, scores.class1 = bg, curve = T))
# plot(pr.curve(scores.class0 = fg, scores.class1 = bg, curve = T))
fg <- holdout_pred_split3[test2$failure_6d == 'F']
bg <- holdout_pred_split3[test2$failure_6d == 'NF']
plot(roc.curve(scores.class0 = fg, scores.class1 = bg, curve = T))
```




```
plot(pr.curve(scores.class0 = fg, scores.class1 = bg, curve = T))
```



4.4.0.2 "How many days left before a device will fail?"

To answer this question we will try to fit an xgboost model to our data. this will let us to fine tune its parameters faster (parallel implementation).

After finding initial sensible values manually and looking at the test sets we achieve a *MAE=10 days* and *RMSE=12 days* that will probably go down if we train on all months.

Even if this approach is interesting from a business point of view, clearly more data is needed to have actionable results.

```
if (file.exists('gbmFitSplitReg1')){ #to not run training each time we Knit
  gbmFitSplitReg1 = readRDS('gbmFitSplitReg1')
  gbmFitSplitReg2 = readRDS('gbmFitSplitReg2')
  gbmFitSplitReg3 = readRDS('gbmFitSplitReg3')
}else{
  fitControl <- trainControl(method = "none", classProbs = TRUE, allowParallel = TRUE)

  grid_default <- expand.grid(
    nrounds = 300,
    max_depth = 5,
    eta = 0.1,
    gamma = 0.3,
    colsample_bytree = 1,
    min_child_weight = 1,
    subsample = 1
  )

  set.seed(925)
  gbmFitSplitReg1 <- train(trainformula_logdevice_reg,
                   data = train6,
                   method = "xgbTree",
                   trControl = fitControl,
                   verbose = FALSE,
                   tuneGrid = grid_default,
                   metric = "RMSE")

  saveRDS(gbmFitSplitReg1, "gbmFitSplitReg1")

  set.seed(925)
  gbmFitSplitReg2 <- train(trainformula_logdevice_reg,
                   data = train7,
                   method = "xgbTree",
                   trControl = fitControl,
                   verbose = FALSE,
                   tuneGrid = grid_default,
                   metric = "RMSE")

  saveRDS(gbmFitSplitReg2, "gbmFitSplitReg2")

  set.seed(925)
  gbmFitSplitReg3 <- train(trainformula_logdevice_reg,
                   data = train8,
                   method = "xgbTree",
                   trControl = fitControl,
                   verbose = FALSE,
                   tuneGrid = grid_default,
```

```
                    metric = "RMSE")

  saveRDS(gbmFitSplitReg3, "gbmFitSplitReg3")
}
```

```
holdout_pred_splitreg1 <- predict(gbmFitSplitReg1, newdata = test4)
holdout_pred_splitreg2 <- predict(gbmFitSplitReg2, newdata = test3)
holdout_pred_splitreg3 <- predict(gbmFitSplitReg3, newdata = test2)

print(paste('Split 1',
            'RMSE:', RMSE(pred = holdout_pred_splitreg1, obs = test4$failure_reg),
            '- MAE:', MAE(pred = holdout_pred_splitreg1, obs = test4$failure_reg)))
```

```
## [1] "Split 1 RMSE: 38.0026649460347 - MAE: 31.2040316621462"
```

```
print(paste('Split 2',
            'RMSE:', RMSE(pred = holdout_pred_splitreg2, obs = test3$failure_reg),
            '- MAE:', MAE(pred = holdout_pred_splitreg2, obs = test3$failure_reg)))
```

```
## [1] "Split 2 RMSE: 23.3980533279762 - MAE: 17.9516016126975"
```

```
print(paste('Split 3',
            'RMSE:', RMSE(pred = holdout_pred_splitreg3, obs = test2$failure_reg),
            '- MAE:', MAE(pred = holdout_pred_splitreg3, obs = test2$failure_reg)))
```

```
## [1] "Split 3 RMSE: 12.360236360743 - MAE: 10.2006062239655"
```

5 What can we improve

To improve our performance on this task, here is what we can consider doing:

- Gathering more data (multi-year logs)
- Use stationary data about device (year, make, ...)
- Tackling the imbalance in the data-set (over-sampling, under-sampling, generation of synthetic data)
- Doing better Hyperparameter search (faster training on Xgboost instead of GBM)
- Using a custom objective function for GBM if we have more information about the cost of False Positives & False Negatives
- Trying different models (not ensemble based like Survival Analysis, LSTMs...)
- Trying a different approach like semi-supervised learning (hopefully classifying failures about to happen in the last month of the data-set)