KEGMIL

Predictive Maintenance – Asset Component Failure

Abstract

This proof of concept project brings together the business best practices and analytical guidelines to successfully develop and deploy predictive solutions that can help businesses in several industries achieve high asset utilisation and savings in operational costs

Contents

Summary

Businesses require critical equipment to be running at peak efficiency and utilisation to realise their return on capital investments. Most businesses rely on corrective maintenance, where parts are replaced as and when they fail; at the expense of downtime and higher labour cost. The better practice is preventive maintenance, where they determine the useful lifespan for a part, and maintain or replace it before a failure; at the expense of under-utilisation of the component during its useful lifetime. Predictive maintenance is to optimise the balance between corrective and preventive maintenance, by enabling just-in-time replacement of components thus extending the components' lifespan.

Predictive maintenance is an application of predictive analytics that can help businesses in several industries achieve high asset utilisation and savings in operational costs. This proof of concept brings together the business best practices and analytical guidelines to successfully develop and deploy predictive solutions that aim to predict whether an asset may fail in the near future and identify the main causes of failure of an asset.

Technical Implementation

The business problem for this proof of concept is to answer the question "What is the probability that a machine will fail in the near future due to a certain component's failure?" and is formatted as a multiclass classification problem. A machine learning algorithm is used to create the predictive model that learns from historical and near real-time data from sensors installed on machines.

Workflow

1. The data collected from machines feeds into the Azure Event Hub service as data points.

2. Two Azure Stream Analytics jobs analyse the data to provide near real-time analytics on the input stream from the Event Hub. One of the Stream Analytics jobs archives all raw incoming events to the Azure Blob Storage service for later processing by the Azure Data Factory service, and the other publishes results onto Power BI dashboard.

3. The HDInsight service is used to run Hive scripts (orchestrated by Data Factory) to provide aggregations on the raw events that were archived by the aforementioned Stream Analytics job.

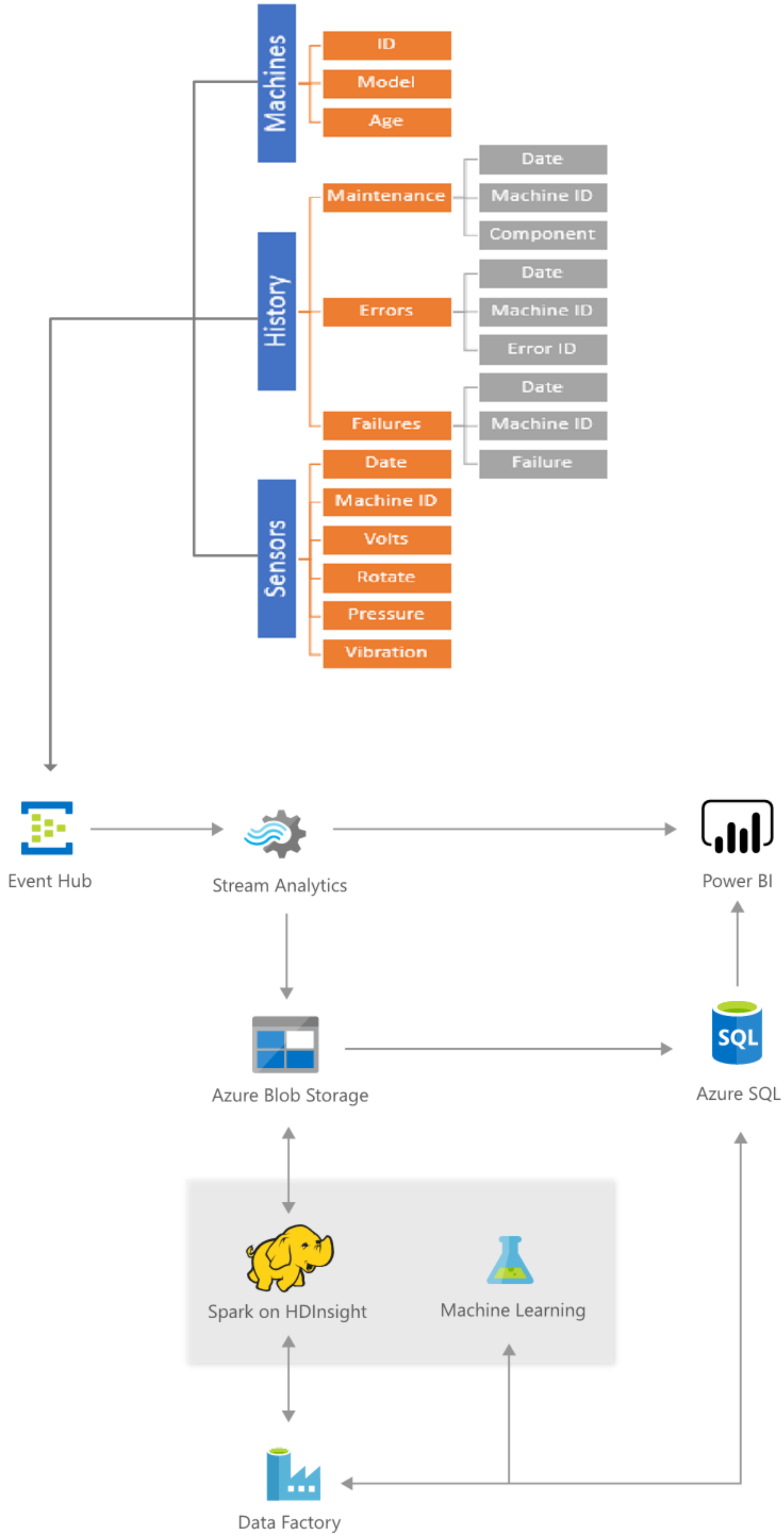
4. Azure Machine Learning is used (orchestrated by Data Factory) to make predictions on the machine's failure caused by a certain component given the inputs received.

5. Azure SQL Database is used (managed by Data Factory) to store the prediction results received from Machine Learning. These results are then consumed in the Power BI dashboard. A stored procedure is deployed in the SQL Database and later invoked in Data Factory pipeline to store the Machine Learning prediction results into the scoring result table.

6. Data Factory handles orchestration, scheduling, and monitoring of the batch processing pipeline.

7. Finally, Power BI is used for results visualisation, so that engineers can monitor the sensor data from the dashboard to schedule maintenance.

Architecture



Source: Microsoft Azure

Building the Machine Learning Model

Data Preparation

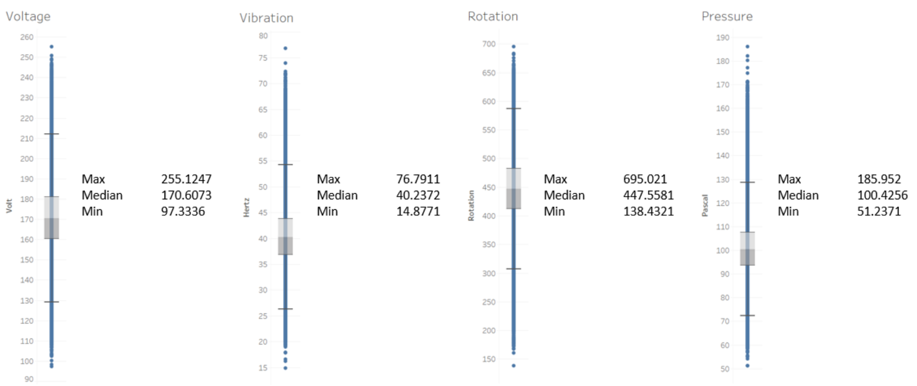
The first step in building the predictive model is to prepare the data. To predict failures, data must contain samples of both failures and no failures. A large number of samples will result in better predictive models and include the following data:

- Some information about the machines: model type and age (years in service)
- Scheduled and unscheduled maintenance records of 2015 which correspond to both regular inspection of components as well as failures
- The error logs in 2015 which are non-breaking errors thrown while the machine is still operational and do not constitute as failures
- Records of component replacements due to failures in 2015. Each record has a date and time, machine ID, and failed component type
- Telemetry data collected in 2015 which consists of voltage, rotation, pressure, and vibration measurements collected from 100 machines

Telemetry data from sensors installed on machines

```
# explore the telemetry data

str(telemetry)
summary(telemetry)

# format datetime field

telemetry$datetime <- as.POSIXct(telemetry$datetime,
                                 format = "%m/%d/%Y %I:%M:%S %p",
                                 tz = "UTC")
```

```
> telemetry <- read.csv('telemetry.csv')
> str(telemetry)
'data.frame':	876100 obs. of  6 variables:
 $ datetime : Factor w/ 8761 levels "1/1/2015 1:00:00 PM",..: 11 13 15 17 2 4 6 1 7 8 ...
 $ machineID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ volt     : num  176 163 171 162 158 ...
 $ rotate   : num  419 403 527 346 435 ...
 $ pressure : num  113.1 95.5 75.2 109.2 111.9 ...
 $ vibration: num  45.1 43.4 34.2 41.1 26 ...
> telemetry$datetime <- as.POSIXct(telemetry$datetime,
+                                  format = "%m/%d/%Y %I:%M:%S %p",
+                                  tz = "UTC")
> str(telemetry)
'data.frame':	876100 obs. of  6 variables:
 $ datetime : POSIXct, format: "2015-01-01 06:00:00" "2015-01-01 07:00:00" "2015-01-01 08:00:00" ...
 $ machineID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ volt     : num  176 163 171 162 158 ...
 $ rotate   : num  419 403 527 346 435 ...
 $ pressure : num  113.1 95.5 75.2 109.2 111.9 ...
 $ vibration: num  45.1 43.4 34.2 41.1 26 ...
```



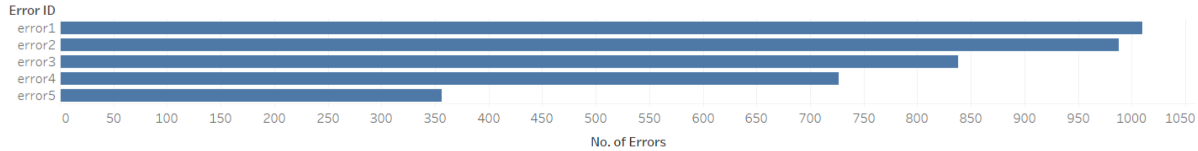
Insight: Checking for outliers as it may indicate irregularities in the machine's performance.

Error logs captured while the machine is still operational and do not constitute as failures

```
# explore the errors data

str(errors)
head(errors, 5)
tail(errors, 5)

# format datetime and errorID fields

errors$datetime <- as.POSIXct(errors$datetime,
                              format = "%m/%d/%Y %I:%M:%S %p",
                              tz = "UTC")
```

```
> errors <- read.csv('errors.csv')
> str(errors)
'data.frame':	3919 obs. of  3 variables:
 $ datetime : Factor w/ 2720 levels "1/1/2015 10:00:00 AM",..: 180 181 197 11 113 138 151 1305 1337 1231 ...
 $ machineID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ errorID  : Factor w/ 5 levels "error1","error2",..: 1 3 5 4 4 4 1 2 1 1 ...
> errors$datetime <- as.POSIXct(errors$datetime,
+                               format = "%m/%d/%Y %I:%M:%S %p",
+                               tz = "UTC")
> str(errors)
'data.frame':	3919 obs. of  3 variables:
 $ datetime : POSIXct, format: "2015-01-03 07:00:00" "2015-01-03 20:00:00" "2015-01-04 06:00:00" ...
 $ machineID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ errorID  : Factor w/ 5 levels "error1","error2",..: 1 3 5 4 4 4 1 2 1 1 ...
> head(errors, 5)
             datetime machineID errorID
1 2015-01-03 07:00:00         1  error1
2 2015-01-03 20:00:00         1  error3
3 2015-01-04 06:00:00         1  error5
4 2015-01-10 15:00:00         1  error4
5 2015-01-22 10:00:00         1  error4
> tail(errors, 5)
                datetime machineID errorID
3915 2015-11-21 08:00:00       100  error2
3916 2015-12-04 02:00:00       100  error1
3917 2015-12-08 06:00:00       100  error2
3918 2015-12-08 06:00:00       100  error3
3919 2015-12-22 03:00:00       100  error3
```

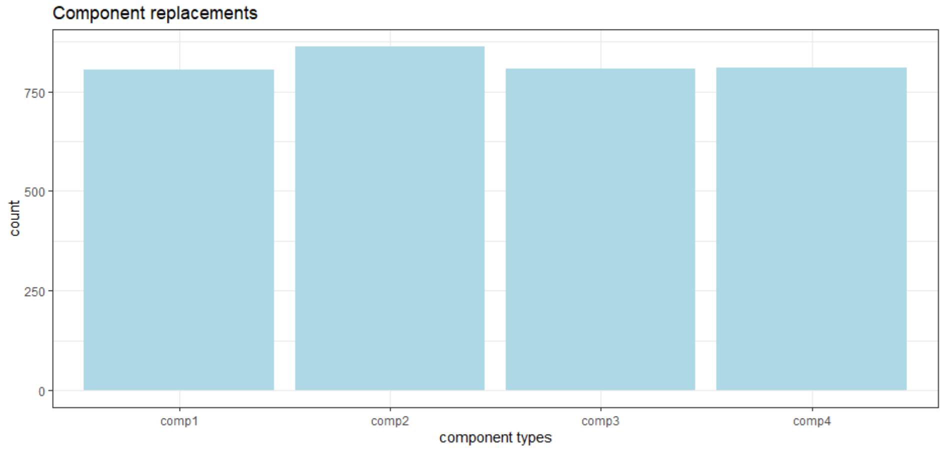
Error Logs



Scheduled and unscheduled maintenance records

```
# explore the maintenance data

str(maint)
head(maint, 10)

# format datetime and comp fields

maint$datetime <- as.POSIXct(maint$datetime,
                             format = "%m/%d/%Y %I:%M:%S",
                             tz = "UTC")

# plot replaced components by type

library("ggplot2")

options(repr.plot.width = 5, repr.plot.height = 3)
ggplot(maint, aes(x = comp)) +
       geom_histogram(stat = "count", fill = "light blue") +
       labs(title = "Component replacements", x = "component types")
```

```
> maint <- read.csv('maint.csv')
> str(maint)
'data.frame':	3286 obs. of  3 variables:
 $ datetime : Factor w/ 374 levels "1/1/2015 6:00:00 AM",..: 248 286 303 100 26 26 13 13 150 150 ...
 $ machineID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ comp     : Factor w/ 4 levels "comp1","comp2",..: 2 4 3 1 4 1 3 1 4 3 ...
> maint$datetime <- as.POSIXct(maint$datetime,
+                              format = "%m/%d/%Y %I:%M:%S",
+                              tz = "UTC")
> str(maint)
'data.frame':	3286 obs. of  3 variables:
 $ datetime : POSIXct, format: "2014-06-01 06:00:00" "2014-07-16 06:00:00" "2014-07-31 06:00:00" ...
 $ machineID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ comp     : Factor w/ 4 levels "comp1","comp2",..: 2 4 3 1 4 1 3 1 4 3 ...
> head(maint, 10)
              datetime machineID  comp
1  2014-06-01 06:00:00         1 comp2
2  2014-07-16 06:00:00         1 comp4
3  2014-07-31 06:00:00         1 comp3
4  2014-12-13 06:00:00         1 comp1
5  2015-01-05 06:00:00         1 comp4
6  2015-01-05 06:00:00         1 comp1
7  2015-01-20 06:00:00         1 comp3
8  2015-01-20 06:00:00         1 comp1
9  2015-02-04 06:00:00         1 comp4
10 2015-02-04 06:00:00         1 comp3
```

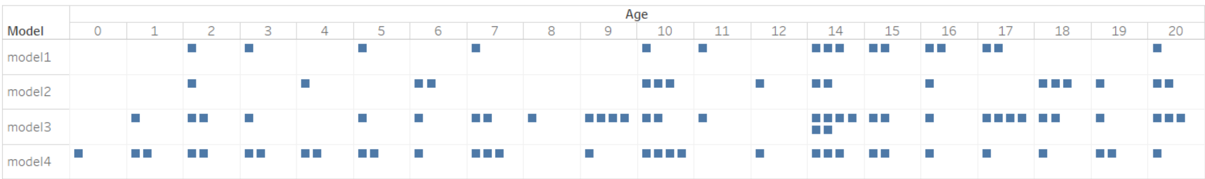


Insight: 2,879 maintenance recorded with component 2 being the most replaced, thus more spare component 2 could be prepared in advance compared to the other components.

Information about the machines

```
# explore machines data

str(machines)
head(machines, 10)
cat("Total number of machines:", nrow(machines))
> machines <- read.csv('machines.csv')
> str(machines)
'data.frame':	100 obs. of  3 variables:
 $ machineID: int  1 2 3 4 5 6 7 8 9 10 ...
 $ model    : Factor w/ 4 levels "model1","model2",..: 3 4 3 3 3 3 3 3 4 3 ...
 $ age      : int  18 7 8 7 2 7 20 16 7 10 ...
> head(machines, 10)
   machineID  model age
1          1 model3  18
2          2 model4   7
3          3 model3   8
4          4 model3   7
5          5 model3   2
6          6 model3   7
7          7 model3  20
8          8 model3  16
9          9 model4   7
10        10 model3  10
> cat("Total number of machines:", nrow(machines))
Total number of machines: 100
```

Age of Machines by Model

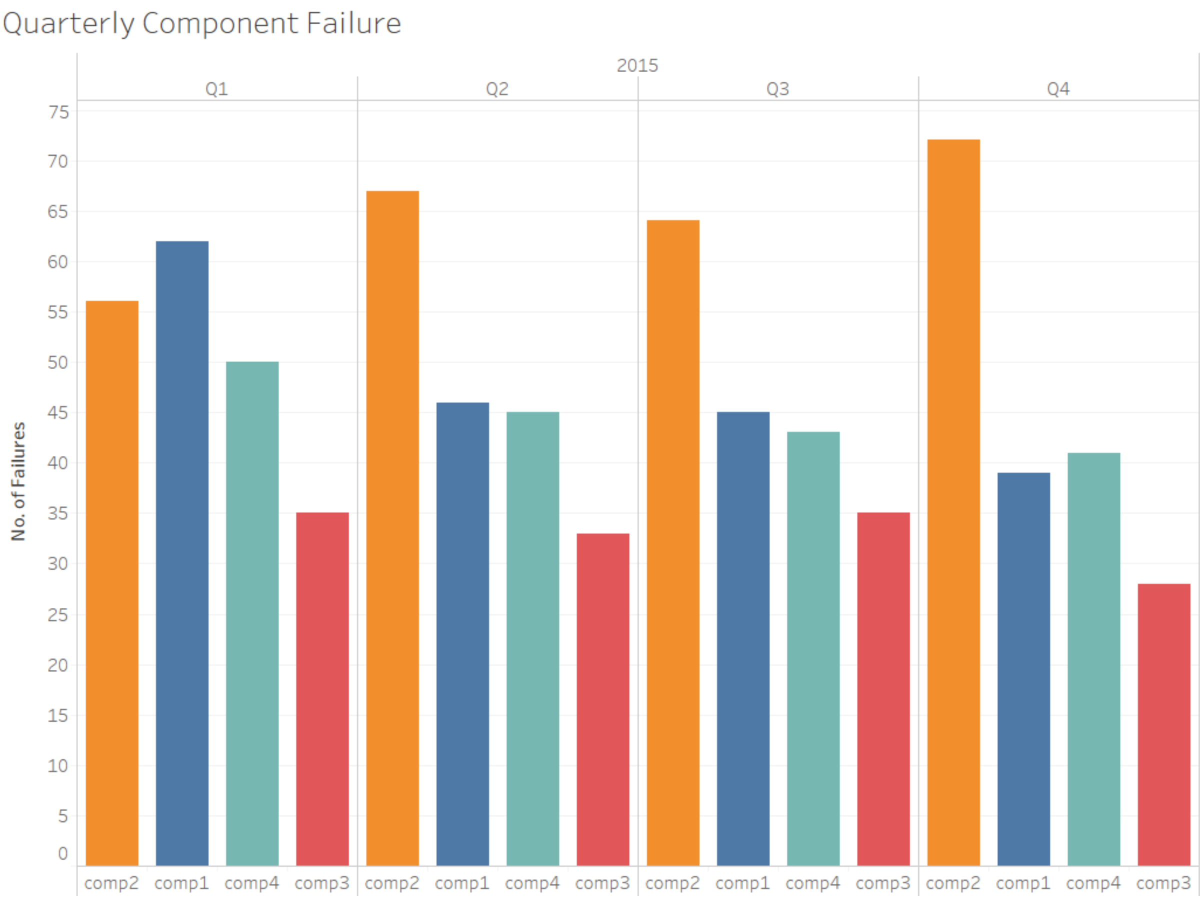


Insight: We should test the common belief that the age of the machine is a significant factor in causing the machine's failure i.e. the older the machine, the more likely it is going to fail

Records of component replacements due to failures

```
# explore failures data

str(failures)
head(failures, 10)

# format datetime field

failures$datetime <- as.POSIXct(failures$datetime,
                               format = "%m/%d/%Y %I:%M:%S",
                               tz = "UTC")
```

```
> failures <- read.csv('failures.csv')
> str(failures)
'data.frame':   761 obs. of  3 variables:
 $ datetime : Factor w/ 302 levels "1/10/2015 6:00:00 AM",..: 23 151 165 213 289 34 84 136 136 162 ...
 $ machineID: int  1 1 1 1 1 1 1 2 2 2 ...
 $ failure  : Factor w/ 4 levels "comp1","comp2",..: 4 1 2 4 4 2 4 1 2 2 ...
> failures$datetime <- as.POSIXct(failures$datetime,
+                                 format="%m/%d/%Y %I:%M:%S",
+                                 tz="UTC")
> str(failures)
'data.frame':   761 obs. of  3 variables:
 $ datetime : POSIXct, format: "2015-01-05 06:00:00" "2015-03-06 06:00:00" "2015-04-20 06:00:00" ...
 $ machineID: int  1 1 1 1 1 1 1 2 2 2 ...
 $ failure  : Factor w/ 4 levels "comp1","comp2",..: 4 1 2 4 4 2 4 1 2 2 ...
> head(failures, 10)
              datetime machineID failure
1  2015-01-05 06:00:00         1   comp4
2  2015-03-06 06:00:00         1   comp1
3  2015-04-20 06:00:00         1   comp2
4  2015-06-19 06:00:00         1   comp4
5  2015-09-02 06:00:00         1   comp4
6  2015-10-17 06:00:00         1   comp2
7  2015-12-16 06:00:00         1   comp4
8  2015-03-19 06:00:00         2   comp1
9  2015-03-19 06:00:00         2   comp2
10 2015-04-18 06:00:00         2   comp2
```



Insight: 761 component failures occurred mainly caused by component 2, thus investigation should be done to explain the high number of component 2 failures.

Feature Engineering

Create features that best describe a machine's condition at a given point in time to improve the model's predictions.

Lag windows from Telemetry

Rolling mean and standard deviation over the last 3-hour lag window are calculated for every 3 hours

```
voltmean_24hrs = rollapply(volt, width = 3, FUN = mean, align = "right", fill = NA, by = 3)
```

Using 'rollapply' (a generic function for applying a function to rolling margins of an array),
'FUN' is equal to 'mean' to calculate the mean or equal to 'sd' to calculate the standard deviation,
'width' is set to '3' for the 3-hour lag window, to calculate the mean for every 3 hours 'by' is set to '3',
and do the same for 'rotate', 'pressure' and 'vibration' to create the feature 'telemetrymean'

```
head(telemetrymean)
```

	datetime	machineID	voltmean	rotatemean	pressuremean	vibrationmean
	<dttm>	<int>	<dbl>	<dbl>	<dbl>	<dbl>
1	2015-01-01 08:00:00	1	170.	450.	94.6	40.9
2	2015-01-01 11:00:00	1	164.	404.	106.	34.3
3	2015-01-01 14:00:00	1	168.	436.	108.	41.2
4	2015-01-01 17:00:00	1	166.	430.	102.	40.4
5	2015-01-01 20:00:00	1	169.	437.	90.9	41.7
6	2015-01-01 23:00:00	1	169.	486.	90.4	41.8

To capture a longer term effect, rolling mean and standard deviation over the last 24-hour lag window are also calculated for every 3 hours

```
voltmean_24hrs = rollapply(volt, width = 24, FUN = mean, align = "right", fill = NA, by = 3)
```

Using 'rollapply', 'FUN' is equal to 'mean' to calculate the mean or equal to 'sd' to calculate the standard deviation,
'width' is set to '24' for the 24-hour lag window, to calculate the mean for every 3 hours 'by' is set to '3',
and do the same for 'rotate', 'pressure' and 'vibration' to create the feature 'telemetrymean_24hrs'

```
head(telemetrymean_24hrs)
```

	datetime	machineID	voltmean_24hrs	rotatemean_24hrs	pressuremean_24hrs	vibrationmean_24hrs
	<dttm>	<int>	<dbl>	<dbl>	<dbl>	<dbl>
1	2015-01-02 05:00:00	1	170.	445.	96.8	40.4
2	2015-01-02 08:00:00	1	171.	444.	97.7	39.8
3	2015-01-02 11:00:00	1	170.	446.	96.9	40.0
4	2015-01-02 14:00:00	1	170.	447.	96.2	39.9
5	2015-01-02 17:00:00	1	170.	452.	96.4	40.0
6	2015-01-02 20:00:00	1	169.	453.	98.0	39.5

Lag windows from Errors

Error IDs are categorical values, so instead of rolling mean and standard deviation, count the number of errors of each type in a lag window

```
errorcount <- errorcount %>%
  group_by(machineID,datetime) %>%
  summarise(error1sum = sum(error1),
            error2sum = sum(error2),
            error3sum = sum(error3),
            error4sum = sum(error4),
            error5sum = sum(error5)) %>%
  ungroup()
```

Using the 'group_by' function to group the error counts by 'machineID' and 'datetime',
sum the duplicate errors within an hour, and using 'ungroup' to return to a non-grouped form

```
head(errorcount)
```

	machineID	datetime	error1sum	error2sum	error3sum	error4sum	error5sum
	<int>	<dttm>	<int>	<int>	<int>	<int>	<int>
1	1	2015-01-03 07:00:00	1	0	0	0	0
2	1	2015-01-03 20:00:00	0	0	1	0	0
3	1	2015-01-04 06:00:00	0	0	0	0	1
4	1	2015-01-10 15:00:00	0	0	0	1	0
5	1	2015-01-22 10:00:00	0	0	0	1	0
6	1	2015-01-25 15:00:00	0	0	0	1	0

```
errorfeat <- telemetry %>%
  select(datetime, machineID) %>%
  left_join(errorcount, by = c("datetime", "machineID"))
```

Align errors with telemetry's datetime field using the 'left_join' function by 'datetime' and 'machineID'

```
errorfeat[is.na(errorfeat)] <- 0
```

Replace missing values with '0'

```
error1count = rollapply(error1sum, width = 24, FUN = sum, align = "right", fill = NA, by = 3
```

Using 'rollapply', count the number of errors in the last 24 hours (width=24), for every 3 hours (by=3), by setting 'FUN' equal to sum, and do the same for the other types of errors

```
head(errorfeat)
```

	datetime	machineID	error1count	error2count	error3count	error4count	error5count
	<dttm>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2015-01-02 05:00:00	1	0	0	0	0	0
2	2015-01-02 08:00:00	1	0	0	0	0	0
3	2015-01-02 11:00:00	1	0	0	0	0	0
4	2015-01-02 14:00:00	1	0	0	0	0	0
5	2015-01-02 17:00:00	1	0	0	0	0	0
6	2015-01-02 20:00:00	1	0	0	0	0	0

Time since last component replacement from Maintenance

Number of days since component was replaced is calculated for each component type and should explain component failures better since the longer a component is used, the more wear-out should be expected.

```
comprep <- maint %>%
  select(datetime, machineID, comp) %>%
  mutate(comp1 = as.integer(comp == "comp1"),
         comp2 = as.integer(comp == "comp2"),
         comp3 = as.integer(comp == "comp3"),
         comp4 = as.integer(comp == "comp4")) %>%
  select(-comp)
```

Create a binary column for each component, '1' if replacement occurred and '0' if it did not occur

```
head(comprep)
```

	datetime	machineID	comp1	comp2	comp3	comp4
1:	2014-06-01 06:00:00	1	0	1	0	0
2:	2014-07-16 06:00:00	1	0	0	0	1
3:	2014-07-31 06:00:00	1	0	0	1	0
4:	2014-12-13 06:00:00	1	1	0	0	0
5:	2015-01-05 06:00:00	1	0	0	0	1
6:	2015-01-05 06:00:00	1	1	0	0	0

```
comp1rep <- comprep[comp1 == 1, .(machineID, datetime, lastrepcomp1 = datetime)]
```

Separate the 4 different component type replacements into different tables

```
compdate <- as.data.table(telemetryfeat[,c(1:2)])
setkey(compdate, machineID, datetime)
```

Use telemetry's feature table 'datetime' and 'machineID' to be matched with replacements using the setkey (sorts a 'data.table' and marks it as sorted) function

```
comp1feat <- comp1rep[compdate[,.(machineID, datetime)],roll = TRUE]
comp1feat$sincelastcomp1 <- as.numeric(difftime(comp1feat$datetime, comp1feat$lastrepcomp1, units = "days"))
```

'data.table' rolling match will attach the latest record from the component replacement tables to the telemetry's 'datetime' and 'machineID' by setting 'roll = TRUE' and the 'difftime' function is used to calculate the number of days since last component replacement

```
compfeat <- data.frame(compdate, comp1feat[,.(sincelastcomp1)], comp2feat[,.(sincelastcomp2)],
            comp3feat[,.(sincelastcomp3)],comp4feat[,.(sincelastcomp4)])
```

Merging all the tables into a data frame to create the 'compfeat' feature

```
head(compfeat,10)
```

Print the first 10 rows of 'compfeat'

	datetime	machineID	sincelastcomp1	sincelastcomp2	sincelastcomp3	sincelastcomp4
1	2015-01-02 05:00:00	1	19.95833	214.9583	154.9583	169.9583
2	2015-01-02 08:00:00	1	20.08333	215.0833	155.0833	170.0833
3	2015-01-02 11:00:00	1	20.20833	215.2083	155.2083	170.2083
4	2015-01-02 14:00:00	1	20.33333	215.3333	155.3333	170.3333
5	2015-01-02 17:00:00	1	20.45833	215.4583	155.4583	170.4583
6	2015-01-02 20:00:00	1	20.58333	215.5833	155.5833	170.5833
7	2015-01-02 23:00:00	1	20.70833	215.7083	155.7083	170.7083
8	2015-01-03 02:00:00	1	20.83333	215.8333	155.8333	170.8333
9	2015-01-03 05:00:00	1	20.95833	215.9583	155.9583	170.9583
10	2015-01-03 08:00:00	1	21.08333	216.0833	156.0833	171.0833

Merging the component and machine features for the final features

```
finalfeat <- finalfeat %>%
            left_join(compfeat, by = c("datetime","machineID")) %>%
            left_join(machines, by = c("machineID"))
```

The final set of features (27 variables) are: datetime, machineID, voltmean, rotatemean, pressuremean, vibrationmean, voltsd, rotatesd, pressuresd, vibrationsd, voltmean_24hrs, rotatemean_24hrs, pressuremean_24hrs, vibrationmean_24hrs, voltsd_24hrs, rotatesd_24hrs, pressuresd_24hrs, vibrationsd_24hrs, error1count, error2count, error3count, error4count, error5count, sincelastcomp1, sincelastcomp2, sincelastcomp3, sincelastcomp4, model and age

Label Construction

Labelling is done by taking a time window prior to the failure and the time window should be picked according to the business case. In this case, the goal is to compute the probability that a machine will fail in the next 24 hours due to either component 1, 2, 3 or 4 failure.

```
labeled <- left_join(finalfeat, failures, by = c("machineID")) %>%
          mutate(datediff = difftime(datetime.y, datetime.x, units = "hours")) %>%
          filter(datediff <= 24, datediff >= 0)
```

Left join final features with failures by 'machineID' then mutate a column for 'datetime' difference with the 'difftime' function, and using the 'filter' function to filter the date difference for the prediction horizon which is 24 hours

```
labeledfeatures <- left_join(finalfeat,
                    labeled %>% select(datetime.x, machineID, failure),
                    by = c("datetime" = "datetime.x", "machineID")) %>%
 arrange(machineID,datetime)

levels(labeledfeatures$failure) <- c(levels(labeledfeatures$failure), "none")

labeledfeatures$failure[is.na(labeledfeatures$failure)] <- "none"
```

Left join labels to final features by 'datetime',
and fill NA's with "none" indicating no failure using the 'is.na' function

```
head(labeledfeatures)
```

```
             datetime machineID voltmean rotatemean pressuremean vibrationmean   voltsd rotatesd pressuresd
1 2015-01-02 05:00:00         1 180.1338   440.6083     94.13797      41.55154 21.32273 48.77051   2.135684
2 2015-01-02 08:00:00         1 176.3643   439.3497    101.55321      36.10558 18.95221 51.32964  13.789279
3 2015-01-02 11:00:00         1 160.3846   424.3853     99.59872      36.09464 13.04708 13.70250   9.988609
4 2015-01-02 14:00:00         1 170.4725   442.9340    102.38059      40.48300 16.64235 56.29045   3.305739
5 2015-01-02 17:00:00         1 163.2638   468.9376    102.72665      40.92180 17.42469 38.68038   9.105775
6 2015-01-02 20:00:00         1 163.2785   446.4932    104.38758      38.06812 21.58049 41.38096  20.725597
  vibrationsd voltmean_24hrs rotatemean_24hrs pressuremean_24hrs vibrationmean_24hrs voltsd_24hrs
1   10.037208       169.7338         445.1799           96.79711            40.38516     11.23312
2    6.737739       170.5257         443.9068           97.66725            39.78667     12.59195
3    1.639962       170.0497         446.4613           96.90616            40.01651     13.27734
4    8.854145       170.3420         447.3553           96.22952            39.92196     13.81716
5    3.060781       170.0606         452.1634           96.35744            39.99047     14.79287
6    6.932127       169.3693         453.3362           98.04201            39.53167     15.67479
  rotatesd_24hrs pressuresd_24hrs vibrationsd_24hrs error1count error2count error3count error4count
1       48.71739        10.079880          5.853209           0           0           0           0
2       46.93028         9.406795          6.098173           0           0           0           0
3       42.83678         9.071472          5.481724           0           0           0           0
4       42.80863         8.256794          5.862312           0           0           0           0
5       42.52529         8.669605          5.907157           0           0           0           0
6       41.68962        10.607947          6.205887           0           0           0           0
  error5count sincelastcomp1 sincelastcomp2 sincelastcomp3 sincelastcomp4  model age failure
1           0       19.95833       214.9583       154.9583       169.9583 model3  18    none
2           0       20.08333       215.0833       155.0833       170.0833 model3  18    none
3           0       20.20833       215.2083       155.2083       170.2083 model3  18    none
4           0       20.33333       215.3333       155.3333       170.3333 model3  18    none
5           0       20.45833       215.4583       155.4583       170.4583 model3  18    none
6           0       20.58333       215.5833       155.5833       170.5833 model3  18    none
```

Data Splitting

Data is split by choosing a point in time so that records before the point in time are used to train the model while records after the point in time are used to test the model.

```
trainingdata1 <- labeledfeatures[labeledfeatures$datetime < "2015-07-31 01:00:00",]
testingdata1 <- labeledfeatures[labeledfeatures$datetime > "2015-08-01 01:00:00",]
```

The split is set at '2015-08-01 01:00:00' and records within 24 hours prior to split point are left out
to prevent overlapping since the labelling window is set at 24 hours,
and the predictive model is to be trained on the first 8 months and tested on the last 4 months

After the split, trainingdata1 has 167,776 number of samples while testingdata1 has 122,952 number of samples

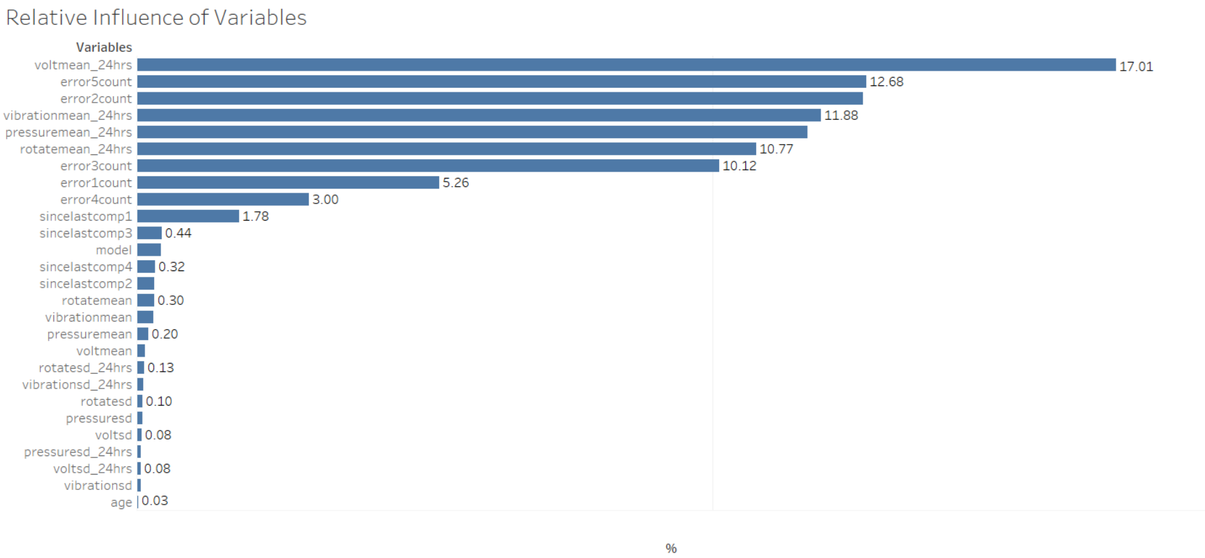
Build and Train the Model

As the machines do not fail most of the time, the class for no failure is expected to be the majority class and cause a severe class imbalance problem. The possible technical approaches include training a Neural Network model with N binary neurons leading to multiclass classification but the standard neural network algorithm does not support imbalanced classification.

Gradient Boosting Machine (GBM) is used to predict the probabilities of the different possible outcomes i.e. the failure of any 1 of the 4 components or none. GBM trains many models sequentially and is a numerical optimisation algorithm where each model minimises the loss function using the Gradient Descent Method. Through this method, a GBM model deals with class imbalance better than other types of models.

Building the model with RStudio:

```
# loading the package 'gbm'
library(gbm)

# create the training formula using all of the variables except datetime and machineID
trainformula <- as.formula(paste('failure',
                                 paste(names(labeledfeatures)[c(3:29)], collapse = ' + '),
                                 sep = ' ~ '))

# train first model
set.seed(1234)

gbm_model1 <- gbm(formula = trainformula, data = trainingdata1,
                  distribution = "multinomial", n.trees = 100,
                  interaction.depth = 6, shrinkage = 0.1)
# 'distribution' is set to 'multinomial' for multiclass classification
# 'n.trees' is at the default '100' number of gradient boosting iteration
# 'interaction.depth' (maximum nodes per tree) is at the default '6'
# 'shrinkage' (the learning rate) is set to '0.1' to prevent overfitting

# print relative influence of variables for first model
summary(gbm_model1)
```



Variables that have been identified from the first model as significant in predicting failure:

- Rolling mean of Telemetry over the last 24-hour lag window calculated for every 3 hours
- Number of errors of each type from Errors over the last 24-hour lag window calculated for every 3 hours

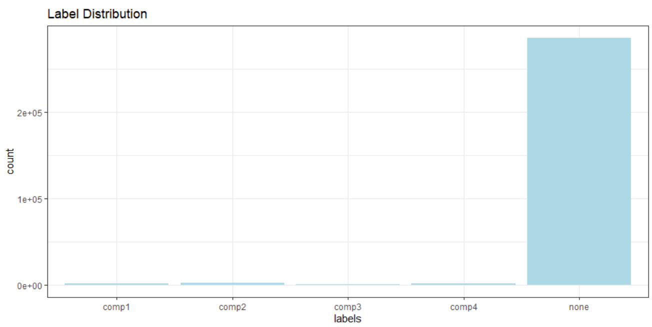
Variables identified as insignificant in predicting failure:

- Rolling standard deviation of Telemetry over the last 24-hour lag window calculated for every 3 hours
- Rolling mean and standard deviation over the last 3-hour lag window calculated for every 3 hours
- Time since last component replacement from Maintenance
- Age of the machine
- The machine's model type

Insight: Commonly-held beliefs that age of the machine and time since component was replaced are good predictors of failure turn out to be insignificant when compared to variables that best describe a machine's condition at a given point in time.

The Class Imbalance Problem

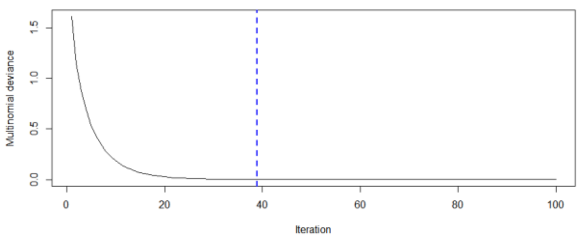
```
# label distribution after features are labeled
ggplot(labeledfeatures, aes(x = failure)) +
          geom_bar(fill = "light blue") +
          labs(title = "Label Distribution", x = "labels")
```



Accuracy is not the metric to use when working with an imbalanced dataset as it is misleading. The following metrics give more insight into the accuracy of the model:

- Precision: A measure of a classifiers exactness
- Recall: A measure of a classifiers completeness
- F-score: A weighted average of precision and recall
- Kappa: Classification accuracy normalised by the imbalance of the classes in the data

```
# define the evaluation function
Evaluate <- function(actual = NULL, predicted = NULL, cm = NULL){
            if(is.null(cm)) {
              actual = actual[!is.na(actual)]
              predicted = predicted[!is.na(predicted)]
              f = factor(union(unique(actual), unique(predicted)))
              actual = factor(actual, levels = levels(f))
              predicted = factor(predicted, levels = levels(f))
              cm = as.matrix(table(Actual = actual, Predicted = predicted))
            }

  # number of instances
  n = sum(cm)

  # number of correctly classified instances per class
  diag = diag(cm)

  # number of instances per class
  rowsums = apply(cm, 1, sum)

  # number of predictions per class
  colsums = apply(cm, 2, sum)

  # distribution of instances over the classes
  p = rowsums / n

  # distribution of instances over the predicted classes
  q = colsums / n

  # accuracy
  accuracy = sum(diag) / n

  # per class
  recall = diag / rowsums
  precision = diag / colsums
  f1 = 2 * precision * recall / (precision + recall)

  # random accuracy
  expAccuracy = sum(p * q)

  # kappa
  kappa = (accuracy - expAccuracy) / (1 - expAccuracy)

  return(list(ConfusionMatrix = cm,
              Metrics = data.frame(
              Accuracy = accuracy,
              Precision = precision,
              Recall = recall,
              F1 = f1,
              Kappa = kappa)))
        }

# find the number of optimum iterations based on the 'OOB' method

ntree_opt_oob <- gbm.perf(gbm_model1, method = "OOB")
ntree_opt_oob
```



```
# predicting the probabilties of outcomes with 'testingdata1'

pred_gbm1 <- as.data.frame(predict(gbm_model1, testingdata1,
                                   n.trees = 50, type = "response"))

# 'n.trees' is set to '50' as the 'OOB' method generally underestimates the optimal number of iterations,
# 'type' is set to 'response' to give the predicted probabilities of outcomes

# renaming the column names and adding a 'failure' column indicating the type of failure

names(pred_gbm1) <- gsub(".50", "", names(pred_gbm1))
pred_gbm1$failure <- as.factor(colnames(pred_gbm1)[max.col(pred_gbm1)])

# evaluation metrics for first model

eval1 <- Evaluate(actual = testingdata1$failure, predicted = pred_gbm1$failure)
eval1$ConfusionMatrix
t(eval1$Metrics)
```

		Predicted Class				
		comp1	comp2	comp3	comp4	none
Actual Class	comp1	93.0%	3.0%	1.5%	1.1%	1.4%
	comp2	0.9%	94.1%	0.7%	4.1%	0.2%
	comp3	3.8%	2.4%	92.5%	1.0%	0.3%
	comp4	3.1%	3.3%	0.7%	92.7%	0.2%
	none	0.1%	0.0%	0.0%	0.1%	99.8%

```
              comp1     comp2     comp3     comp4      none
Accuracy  0.9985035 0.9985035 0.9985035 0.9985035 0.9985035
Precision 0.9020333 0.9420935 0.9740260 0.9069767 0.9998921
Recall    0.9242424 0.9484305 0.9014423 0.9349315 0.9998424
F1        0.9130028 0.9452514 0.9363296 0.9207420 0.9998673
Kappa     0.9615485 0.9615485 0.9615485 0.9615485 0.9615485
```

Building the second model with significant variables only:

```
# create the training formula with the 9 significant variables
trainformula_9 <- failure ~ voltmean_24hrs + error5count + error2count + vibrationmean_24hrs +
  pressuremean_24hrs + error3count + rotatemean_24hrs + error1count + error4count

# train second model with the 9 significant variables

gbm_model2 <- gbm(formula = trainformula_9, data = trainingdata1,
                  distribution = "multinomial", n.trees = 100,
                  interaction.depth = 6, shrinkage = 0.1)
# training the second model with the 'trainformula_9' while keeping other parameters unchanged

# using the second model to predict the probabilities of outcomes

pred_gbm2 <- as.data.frame(predict(gbm_model2, testingdata1,
                                   n.trees = 50, type = "response"))

names(pred_gbm2) <- gsub(".50", "", names(pred_gbm2))
pred_gbm2$failure <- as.factor(colnames(pred_gbm2)[max.col(pred_gbm2)])

# evaluation metrics for second model

eval2 <- Evaluate(actual = testingdata1$failure, predicted = pred_gbm2$failure)
eval2$ConfusionMatrix
t(eval2$Metrics)
```

		Predicted Class				
		comp1	comp2	comp3	comp4	none
Actual Class	comp1	85.8%	3.2%	2.3%	2.1%	6.6%
	comp2	0.8%	95.8%	0.3%	2.9%	0.2%
	comp3	2.9%	3.4%	92.0%	1.0%	0.7%
	comp4	2.2%	5.1%	0.7%	91.8%	0.2%
	none	0.1%	0.0%	0.1%	0.1%	99.7%

```
              comp1     comp2     comp3     comp4      none
Accuracy  0.9977552 0.9977552 0.9977552 0.9977552 0.9977552
Precision 0.8296703 0.9333333 0.9162679 0.9209622 0.9996597
Recall    0.8579545 0.9573991 0.9206731 0.9178082 0.9993197
F1        0.8435754 0.9452131 0.9184652 0.9193825 0.9994897
Kappa     0.9427315 0.9427315 0.9427315 0.9427315 0.9427315
```

Comparison of models:

MODEL WITH 27 VARIABLES

		Predicted Class				
		comp1	comp2	comp3	comp4	none
Actual Class	comp1	93.0%	3.0%	1.5%	1.1%	1.4%
	comp2	0.9%	94.1%	0.7%	4.1%	0.2%
	comp3	3.8%	2.4%	92.5%	1.0%	0.3%
	comp4	3.1%	3.3%	0.7%	92.7%	0.2%
	none	0.1%	0.0%	0.0%	0.1%	99.8%

```
          comp1     comp2     comp3     comp4     none
Accuracy  0.9985035 0.9985035 0.9985035 0.9985035 0.9985035
Precision 0.9020333 0.9420935 0.9740260 0.9069767 0.9998921
Recall    0.9242424 0.9484305 0.9014423 0.9349315 0.9998424
F1        0.9130028 0.9452514 0.9363296 0.9207420 0.9998673
Kappa     0.9615485 0.9615485 0.9615485 0.9615485 0.9615485
```

MODEL WITH 9 SIGNIFICANT VARIABLES

		Predicted Class				
		comp1	comp2	comp3	comp4	none
Actual Class	comp1	85.8%	3.2%	2.3%	2.1%	6.6%
	comp2	0.8%	95.8%	0.3%	2.9%	0.2%
	comp3	2.9%	3.4%	92.0%	1.0%	0.7%
	comp4	2.2%	5.1%	0.7%	91.8%	0.2%
	none	0.1%	0.0%	0.1%	0.1%	99.7%

```
          comp1     comp2     comp3     comp4     none
Accuracy  0.9977552 0.9977552 0.9977552 0.9977552 0.9977552
Precision 0.8296703 0.9333333 0.9162679 0.9209622 0.9996597
Recall    0.8579545 0.9573991 0.9206731 0.9178082 0.9993197
F1        0.8435754 0.9452131 0.9184652 0.9193825 0.9994897
Kappa     0.9427315 0.9427315 0.9427315 0.9427315 0.9427315
```

Removing the insignificant variables did not improve the model, including them in the model improved the metrics (precision, recall, f-score, kappa) and confusion matrix, thus the 27 variables are included in building the model

Building the third model with H2O's Gradient Boosting Machine to address the class imbalance problem by undersampling the majority class to balance the class distribution:

```
library(h2o)
h2o.init()

# import training and testing dataset to the h2o cluster

trainingdata1_h2o <- trainingdata1[ , -c(1, 2)]
trainingdata1_h2o <- as.h2o(trainingdata1_h2o)

testingdata1_h2o <- testingdata1[ , -c(1, 2)]
testingdata1_h2o <- as.h2o(testingdata1_h2o)

# build the model

sample_factors <- c(1., 1., 1., 1., 0.01)

# class 'none' is undersampled by a ratio of '0.01'
# while not changing the sampling rate of the other classes

h2o_gbm_model1 <- h2o.gbm(y = "failure", training_frame = trainingdata1_h2o,
                          ntrees = 100, max_depth = 5, learn_rate = 0.1, distribution = "multinomial",
                          validation_frame = testingdata1_h2o, balance_classes = TRUE,
                          class_sampling_factors = sample_factors, seed = 1234)

# 'ntrees' is set to '100' and 'learn_rate' is set to '0.1' similar to the initial model,
# 'max_depth' (maximum tree depth) is at the default '5',
# 'balance_classes' is enabled  to undersample the majority class

# print the performance of the model

h2o.performance(h2o_gbm_model1)
```

```
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
       comp1 comp2 comp3 comp4 none  Error          Rate
comp1    879    31     3     9    0 0.0466 =    43 / 922
comp2     17  1082    14    39    0 0.0608 =   70 / 1,152
comp3      5     2   584    10    0 0.0283 =    17 / 601
comp4     15    17    14   773    0 0.0562 =    46 / 819
none       0     0     0     0 1653 0.0000 =   0 / 1,653
Totals   916  1132   615   831 1653 0.0342 = 176 / 5,147
```

Class 'none' has been undersampled to similar proportion as the other classes.

Comparison of models:

GBM (27 VARIABLES)

		Predicted Class				
		comp1	comp2	comp3	comp4	none
Actual Class	comp1	93.0%	3.0%	1.5%	1.1%	1.4%
	comp2	0.9%	94.1%	0.7%	4.1%	0.2%
	comp3	3.8%	2.4%	92.5%	1.0%	0.3%
	comp4	3.1%	3.3%	0.7%	92.7%	0.2%
	none	0.1%	0.0%	0.0%	0.1%	99.8%

	comp1	comp2	comp3	comp4	none
Accuracy	0.9985035	0.9985035	0.9985035	0.9985035	0.9985035
Precision	0.9020333	0.9420935	0.9740260	0.9069767	0.9998921
Recall	0.9242424	0.9484305	0.9014423	0.9349315	0.9998424
F1	0.9130028	0.9452514	0.9363296	0.9207420	0.9998673
Kappa	0.9615485	0.9615485	0.9615485	0.9615485	0.9615485

H2O GBM (27 VARIABLES)

		Predicted Class				
		comp1	comp2	comp3	comp4	none
Actual Class	comp1	95.3%	3.4%	0.3%	1.0%	0.0%
	comp2	1.5%	93.9%	1.2%	3.4%	0.0%
	comp3	0.8%	0.3%	97.2%	1.7%	0.0%
	comp4	1.8%	2.1%	1.7%	94.4%	0.0%
	none	0.0%	0.0%	0.0%	0.0%	100.0%

	comp1	comp2	comp3	comp4	none
Accuracy	0.9658	0.9658	0.9658	0.9658	0.9658
Precision	0.9596	0.9558	0.9496	0.9302	1.0000
Recall	0.9534	0.9392	0.9717	0.9438	1.0000
F-Score	0.9565	0.9474	0.9605	0.9370	1.0000
Kappa	0.9559	0.9559	0.9559	0.9559	0.9559

Looking at the confusion matrix and the metrics (Precision, Recall and F-Score), the H2O GBM model performed better than the GBM at predicting the component failures.

Conclusion

Some of the features engineered to describe a machine's condition at a given time proved to be significant predictors of failure. Nevertheless, all 27 variables will be included in the predictive model as it improved the model slightly. The H2O GBM addressed the class imbalance problem better and improved the prediction of failures compared to the standard GBM, thus it will be chosen as the predictive model. This proof of concept allows any of the components that will likely fail in the next 24 hours to be identified. An automated alert could be set up to notify the field technician on standby to prepare for a job to replace the identified component within the next 24 hours. This form of predictive maintenance will reduce the downtime of any of the machines which is costly to the business.