Predictive Maintenance: Capstone Project Report

HarvardX PH125.9x - Data Science: Capstone

Seyed M. Seyedtorabi

2024-Feb-15

I. Introduction

This document is the generated report for the "Chose Your Own Project" capstone of the "HarvardX PH125.9x: Data Science: Capstone" course provided by HarvardX. In the intoducion section of this document, we will go over the motivations and goals behind the project, the chosen data set, problem definition, and the adapted strategy to approach the problem. In later sections, we will go over the exploratory data analysis (EDA), different modelling strategies, model performance comparisons and final results. The document will end with concluding remarks and suggestions of possible improvements for a potential future work.

I.I. Motivation

As the previous "Movielens" capstone project was essentially a regression problem, it was desired to have a classification problem for the final capstone project. Working on a **predictive maintenance** project presented a relevant and intellectually stimulating prospect for this capstone. Predictive maintenance is a growing field of interest in data-driven engineering. The goal of predictive maintenance is to monitor the health state of an asset (e.g. a machine, a component thereof, or even a part of the infrastructure) through analysis of the data collected by sensors which monitor the asset of interest. For instance, if vibration levels of a component is under continuous monitoring by a sensor which sends the data to a database, and the machine learning models detect a change point or a trend in these vibration levels, the asset owner can send inspection personnel to investigate the alert, and possibly carry a preemptive maintenance action to avoid total failure of the asset. In this way, the resources of the inspection personnel is used more efficiently, because the periodic and/or reactive maintenance is replaced by the more targeted approach of **predictive maintenance**. In this project, we will work on a data set to classify the state of machines and predict which failure mode they have based on the provided data.

I.II. The Data set

The Data set chosen for this project is the **Machine Predictive Maintenance Classification** data set available on Kaggle website [1]. We can import the data set and have a first look after renaming the columns to convenient names:

```
# load data set
df_data <- read.csv("./data/predictive_maintenance.csv")

# rename columns to more convenient names
df_data <- df_data %>%
  rename("product_id" = Product.ID,
         "type" = Type,
         "air_t" = Air.temperature..K.,
         "process_t" = Process.temperature..K.,
         "rpm" = Rotational.speed..rpm.,
         "torque" = Torque..Nm.,
         "wear" = Tool.wear..min.,
         "failure_numeric" = Target,
         "failure_type" = Failure.Type)

# show the head
knitr::kable(head(df_data), row.names = FALSE)
```

UDI	product_id	type	air_t	process_t	rpm	torque	wear	failure_numeric	failure_type
1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
6	M14865	M	298.1	308.6	1425	41.9	11	0	No Failure

After a quick check we found out that the first two columns are the unique identifiers for every row. Therefore, they are not relevant for any modelling considerations.

According to the data set description [1], the **type** column describes the type of the machine and contains possible 3 values of L, M, and H, which stand for low, medium, and high quality, respectively. the **air_t** and **process_t** columns contain respectively the air temperatures and the machine processing temperatures in Kelvin. The **rpm** column stores the angular speed of the machine in revolutions per minute (rpm), while the **torque** columns contains the angular force in Nm. The **wear** column indicates the tool wear in minutes. The last 2 columns are the labels. The column **failure_numeric** contains binary values, 1 for machine failure and 0 for no failure, followed by the **failure_type** columns which indicates the mode of failure. Here, we can see the failure modes:

```
unique(df_data$failure_type)
```

```
## [1] "No Failure"                "Power Failure"
## [3] "Tool Wear Failure"         "Overstrain Failure"
## [5] "Random Failures"           "Heat Dissipation Failure"
```

We can also check if the data set is balanced or not by defining and looking at the distribution in the **failure** column:

```
# crate the failure label column form 0 and 1 ==> no failure and failure
df_data <- df_data %>%
  mutate(failure = ifelse(failure_numeric==1, "Failure", "No Failure"))

table(df_data$failure)
```

```
##
##    Failure No Failure
##        339       9661
```

The data set contains 9661 observations without failure, and only 339 failure observations. This shows that the data set is quite unbalanced and a classification algorithm may be affected by this.

I.III. The Objective

The objective in this project is to use different data analysis, data transformation, feature engineering, and machine learning methods to classify and predict the failure of the machines. Although there are 6 distinct categories in the **failure_type** column, one of them is essentially the negative case ("No Failure"), while the other categories are the positive cases. In other words, this classification problem is inherently a binary problem from this point of view. Additionally, it is possible to imagine a real-world scenario where the machine owner is more interested in the health status of the machine rather than the specific failure mode.

It is important to mention that in our modelling efforts, we may use the multi-class classification approaches, but if the model predicts e.g. **Heat Dissipation Failure** while the real case is **Overstrain Failure**, we will still count it as a true positive. Since the data set is very unbalanced, we will not use **accuracy** as the metric to evaluate the model performance. Instead, we will use **F1-score** for that purpose. As a reminder, the F1-score is calculated through the following formula:

$$F1 = 2 \times \frac{recall \times precision}{recall + precision}$$

Precision and recall are defined by the following formulas:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Additionally, we will use comparisons of receiver operating characteristic (ROC) curves to visualize the model performance. One more metric that will be calculated throughout some of the sections in this report is the true positive rate (TPR) at 10% false positive rate (FPR). This metric can helpful, because in a real-world scenario, the machine owner may require the data scientist to produce a model that does not have more than 10% false alarm rate, while maintaining a high true alarm rate. Please note that TPR is simply another name for recall, and FPR is defined as:

$$FPR = \frac{FP}{TN + FP}$$

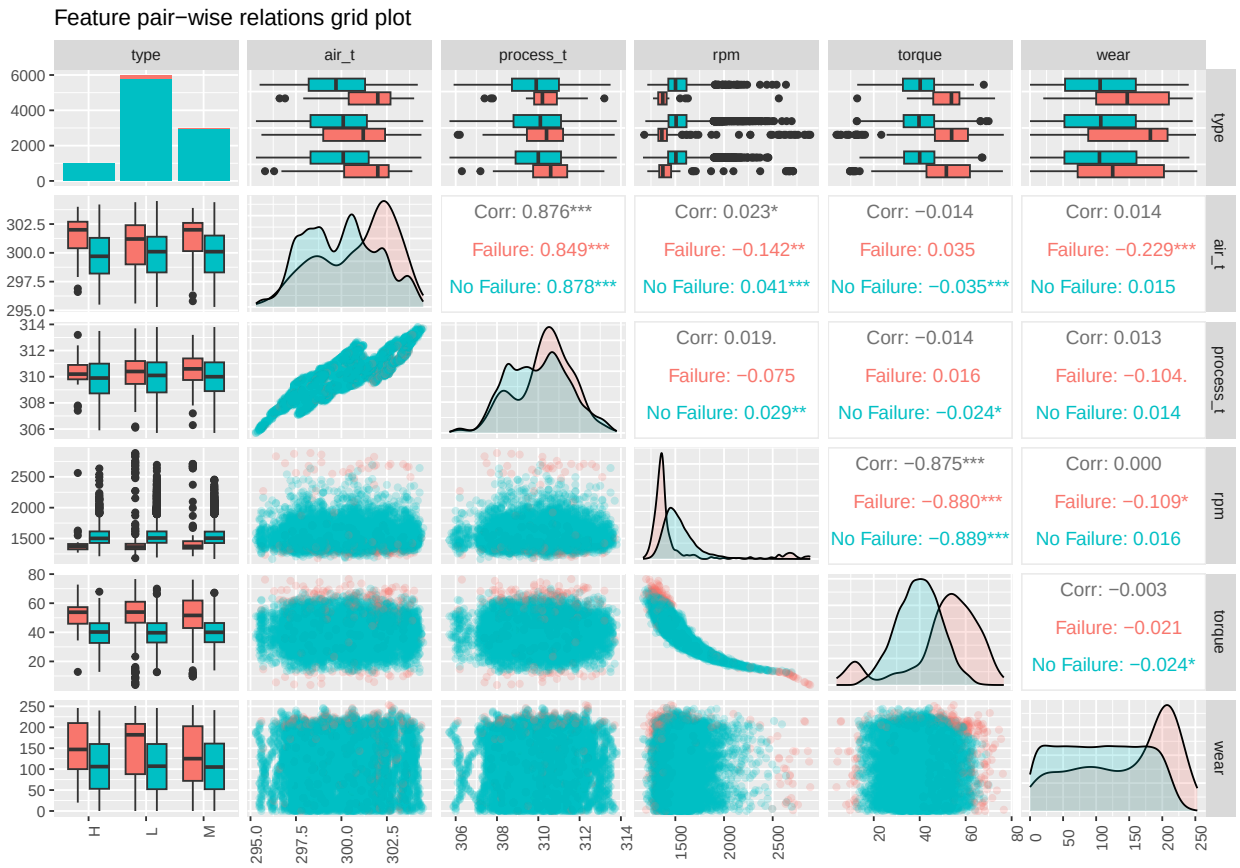
As mentioned before, the objective in this project make the binary failure/no failure classification. We will try to achieve an F1-score of **0.90**. We will also use the ROC curves and the TPR at 10% FPR metric as helpful means that guide our modelling efforts.

II. Analysis

II.I. EDA

Prior to any modelling step, it is crucial to perform some level of EDA and visualization in order to be aware of the relationships and correlations that exist in our data. This will be the starting point in our journey to create a classification model. As the first step, let us create a detailed figure including multiple plots, so that we will see the big picture more clearly:

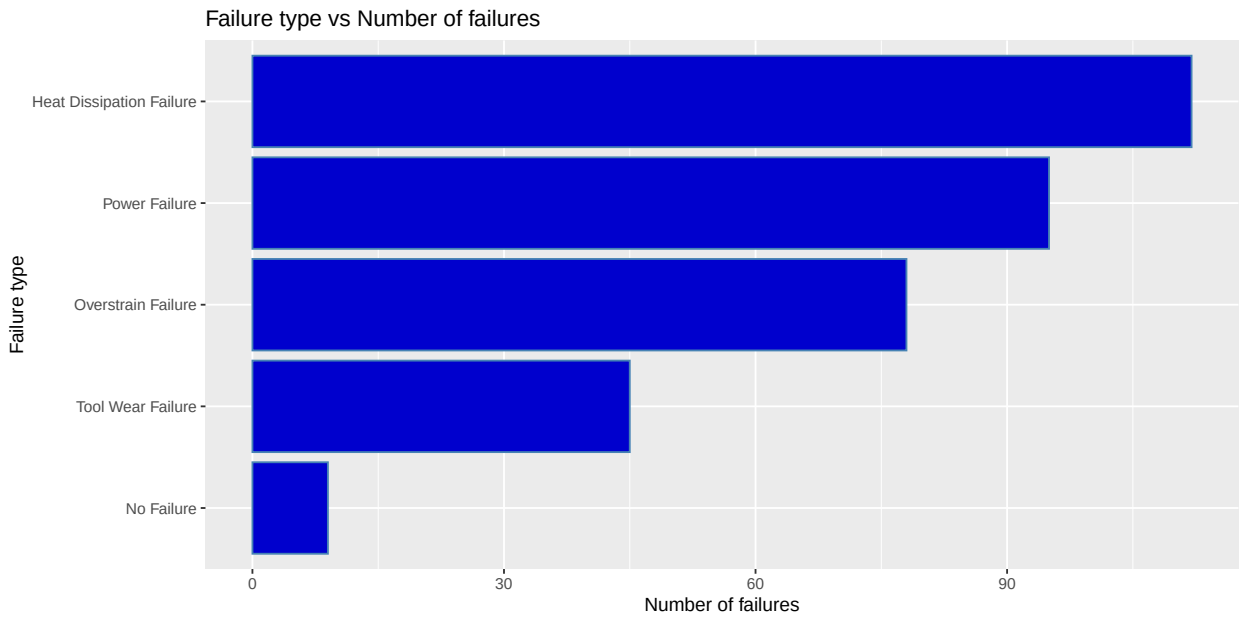
```
# select columns for pair plot
df_pair <- df_data %>%
  dplyr::select(-any_of(c("UDI", "failure", "failure_numeric",
                          "product_id", "failure_type")))

# create and show the pair plot
pair_plot <- ggpairs(df_pair,
                     mapping=aes(color=df_data$failure),
                     lower = list(continuous = wrap("points", alpha = 0.2),
                                  combo = "box_no_facet"),
                     diag = list(continuous = wrap("densityDiag", alpha = 0.2))) +
  labs(title ="Feature pair-wise relations grid plot") +
  theme(axis.text.x = element_text(angle = 90,))

pair_plot
```



This is a very dense figure, but one can already draw very useful insights from it. The first thing to notice is that there are many more blue points compared to red points in the subplots. This means the data set is extremely unbalanced and has a lot more "No Failure" cases than "Failure" ones. From the bar plot at the top left, one can see that L type machines have the most failures, followed by M and H type machines. The box plots on the left and top of the figure show us some rpm, torque, and wear have substantially different distributions for failed and not failed machines, regardless of the machine type. This is also visible in the density plots. The air and process temperatures also seem to be higher in case of failed machines. The figure also shows the correlation coefficients of each feature pair in both failure and no failure cases. In some pairs such as **air_t** vs **wear**, the difference in correlations for the 2 classes seem to be very high. However, one needs to take into consideration that the correlation values for failure cases are based on much smaller number of samples, and different failure types might actually have different correlation values. It is also clear from the figure that the relationship between torque and rpm is inverse. This is expected, because in machines higher torque always comes with a lower rotational speed assuming a constant power output.

Next, let us have a look at the distribution of the failure modes:

```
# see number of different types of failure
df_data %>%
  filter(failure_numeric==1) %>%
  group_by(failure_type) %>%
  summarize(n_failure = n()) %>%
  mutate(failure_type=reorder(failure_type,n_failure)) %>%
  ggplot(aes(x=failure_type, y=n_failure)) +
  geom_col(color = "steelblue", fill="blue3") +
  labs(x = "Failure type",
       y = "Number of failures",
       title ="Failure type vs Number of failures") +
  coord_flip()
```



There are 2 issues one can observe in this plot:
1) "No Failure" appears in the plot despite we filtered them out.
2) "Random Failures" are not in the plot despite the exist in the data set as we discussed in the introduction section.

Therefore, we need to perform some data cleaning. We change the **failure_numeric** column to 0 and

failure column to "No Failure" for all instances with "No Failure" in **failure_type** columns. We also change the failure to to "No Failure" for all instances with 0 in the **failure_numeric** column:

```
# reassign correct failure and failure_numeric to the problematic rows
df_data <- df_data %>%
  mutate(failure_numeric = ifelse(failure_type=="No Failure", 0, 1),
         failure = ifelse(failure_type=="No Failure", "No Failure", failure))

# reassign correct failure_type to the random failure rows
df_data <- df_data %>%
  mutate(failure_type = ifelse(failure=="No Failure",
                               "No Failure",
                               failure_type))
```

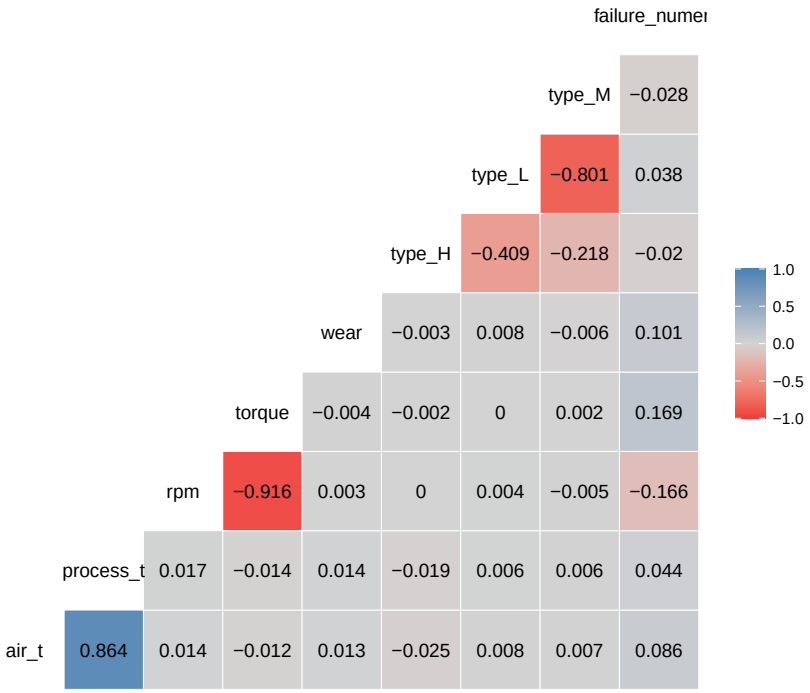
Next, let us visualize the correlations in a better way. This time we also want to include the machine type. However, the machine type in out data set is categorical and we cannot calculate correlation for non-numeric variables. To overcome this, we do **one-hot encoding** of the type data, i.e. we encode the type data in a numeric way. After this encoding, we will have 3 new columns in our data set. Here, we see how these columns are related to the **type** column:

```
# one-hot encoding the categorical type column using the self-defined function
df_data <- one_hot_encode(df_data = df_data, categorical_col = "type")

# show encoding
tail(df_data %>% dplyr::select(type, type_L, type_M, type_H))
```

```
##       type type_L type_M type_H
## 9995     L      1      0      0
## 9996     M      0      1      0
## 9997     H      0      0      1
## 9998     M      0      1      0
## 9999     H      0      0      1
## 10000    M      0      1      0
```

Now that we have numerically encoded the machine types, we can make create a comprehensive correlation plot. Please note that the correlation coefficients are calculated using the Spearman method.

```
# select columns for correlations plot
df_corr <- df_data %>%
  dplyr::select(-any_of(c("UDI", "type", "failure",
                    "product_id", "failure_type")))

# plot the correlations using spearman method
ggcorr(df_corr, label = TRUE, label_round=3,
       method=c("pairwise", "spearman"),
       low = "brown2",
       mid = "lightgrey",
       high = "steelblue")
```



In this plot, we can see that higher torque and wear are correlated with failure. Also, rpm is inversely correlated. There is some variation in correlation coefficients between failure and different machine types, however, the coefficients are too small to conclude anything.

II.II. Splitting the Data

Now that we have an initial understanding of our data, it is time to prepare the data set for training models. The first step in that direction would be to split our data into a test set and a training set:

```
# Split the data set into training and test sets
test_indices <- createDataPartition(df_data$failure, times=1, p=0.2, list=FALSE)
test_set     <- df_data[test_indices, ]
train_set    <- df_data[-test_indices,]
```

We use a 80/20 split for our project. This is to make sure we have enough data to train our models, and have enough data to test them. The 80/20 split is preferred over the more common approach of 90/10, this is because our data set is very unbalanced and we need to have all different modes of failure in the test data.

II.III. Machine Learning Model Algorithms

In this project we will use 7 different machine learning algorithms. The list below includes the Caret name of these algorithms, their actual names, and a short description of them:

1) raprt (Decision tree): A machine learning model that is based on iterative partitioning of feature space and assigning labels to each region.
2) rf (Random forest): An ensemble of multiple decision trees. The output of the classification is the output chosen the most by the trees in the ensemble.
3) lda2 (Linear discriminant analysis): A machine learning algorithms that tries to linearly divide the

data after improving the separability of classes by projecting the data into the lower dimensional linear discriminant (LD) space by maximizing the cross-class distances, while minimizing the in-class variation.
4) pda2 (Penalized discriminant analysis): an extension/variation of the LDA method, which incorporates penalty terms in the loss function for improving the generalizability of the model and prevention of overfitting.
5) nb (Naive Bayes): A machine learning algorithm based on Bayes' theorem. This method creates the a priori probabilities based on the original distribution of the classes in the training set, and updates those probabilities based on observation of other features.
6) xgbTree (Extreme gradient boosting): A greatly optimized version of gradient boosting algorithm, which is effectively an ensemble of weak learners (e.g. trees) trained in a sequential manner, where each learner improves on the error made by the ensemble thus far.
7) LogitBoost (Boosted logistic regression): A method that operates in a comparable way to gradient boosting, and uses an ensemble of logistic regression models which are trained sequentially and learn from mistakes of the ensemble.

It is worth mention mentioning that we use 5 fold cross-validation with default parameters for tuning of all models during the training:

```
# 5 fold cross validation
ctrl <- trainControl(method = "cv", number = 5)
```

II.IV. Other Utilized Methods

To improve the performance of the models through different iterations of modelling, some additional techniques are used int this project. Here, we list these methods and provide a short description for each one:

1) Synthetic Minority Oversampling Technique (SMOTE): A technique to generate synthetic data for the minority class in unbalanced data sets by interpolating between the existing instances of the minority class.
2) Density Based Synthetic Minority Oversampling Technique (DBSMOTE): An extension of SMOTE where synthetic instances are created based on the density of the minority class in each region [2]. DBSMOTE tries to overcome some undesired artifact caused by SMOTE (e.g. building bridges of data points between instances of the minority class that are far away from the main cluster of minority class).
3) Edited Nearest Neighbor (ENN): An undersampling method aimed at reducing the noise in the data by removing the instances which are surrounded by the "neighbors" from other classes [3].
4) Projection to LD space: Projecting the feature space into a lower dimensional linear discriminant (LD) space to improve the separability of the classes and minimize the in-class variations.

II.V. Use of Functions in R

In order to have a cleaner script and to avoid copy-pasting big chunks of code, we have defined many useful functions in the R script of this project. Through out this report, you may encounter some of these functions. The interested reader is encouraged to inspect these functions in the R script provided alongside with this report.

III. Results

In this section, we are going to go through the details and results of different approaches and models during different iterations of model development. Firstly, we are going to start with a simple binary classification approach and compare the performance of different algorithms in the simple case. Then we extend our efforts to multi-class classification, and we also employ some techniques to mitigate the effect of the unbalance and the noise in the data. Later, we try an approach to improve the separability of classes by means of transformations and projections, as well as engineering of physically more meaningful features. Finally, we create an ensemble of our the best performing models. The metrics described in the introduction section are used to evaluate model performance and guide our efforts through different iterations of model development.

III.I. Binray Classification

In the first iteration, we treat the problem as an entirely binary classification problem, so we will use the **failure** columns as our target. Additionally, we will use all the relevant features present in the data set:

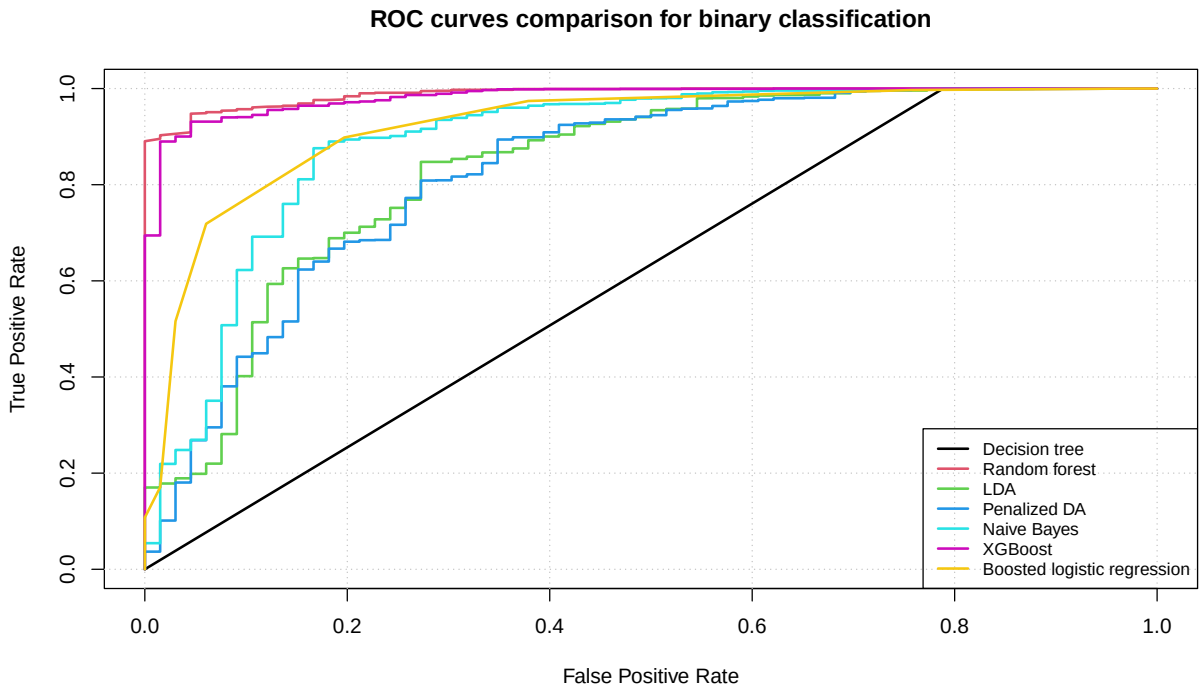
```
# features to use for training and target to predict
my_features <- c("air_t", "process_t", "rpm", "torque",
                 "wear", "type_H", "type_L", "type_M")

my_target   <- "failure"
```

The training will include all the 7 algorithms described in the analysis section, and will take place using "train_model" function defined in our R script:

```
# methods and respective names to train models
methods      <- c("rpart", "rf", "lda2","pda2", "nb", "xgbTree",  "LogitBoost")
mehtod_names <- c("Decision tree", "Random forest", "LDA", "Penalized DA",
                  "Naive Bayes", "XGBoost", "Boosted logistic regression")

# train the models in lapply
invisible(capture.output(
  trained_models_binary <- lapply(methods, function(method){

    model <- train_model(df_training=train_set,
              target=my_target,
              features=my_features,
              method=method,
              control=ctrl)

  model
  })
))
```

After the training, we can plot the ROC curves for the models to get an idea of the performance of each algorithm. This takes place using the "plot_ROC_curves" function defined in the R script, which uses test set to get the probabilities for each prediction, and extract the TPR and FPR before plotting the curves:

```
plot_ROC_curves(trained_models=trained_models_binary,
                model_names=mehtod_names,
                df_test=test_set,
                target=my_target,
                title= "ROC curves comparison for binary classification")
```



It can be seen that random forest and XGBoost are performing better than other models. The table below shows the TPR at 10% FPR for each of the models:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.432	0.6204	0.9418	0.7706

And here are the F1-scores:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.75	0.3883	0.3918	0.381	0.729	0.3636

The F1-scores are not really in the desired range yet. Therefore, we need to take another approach.

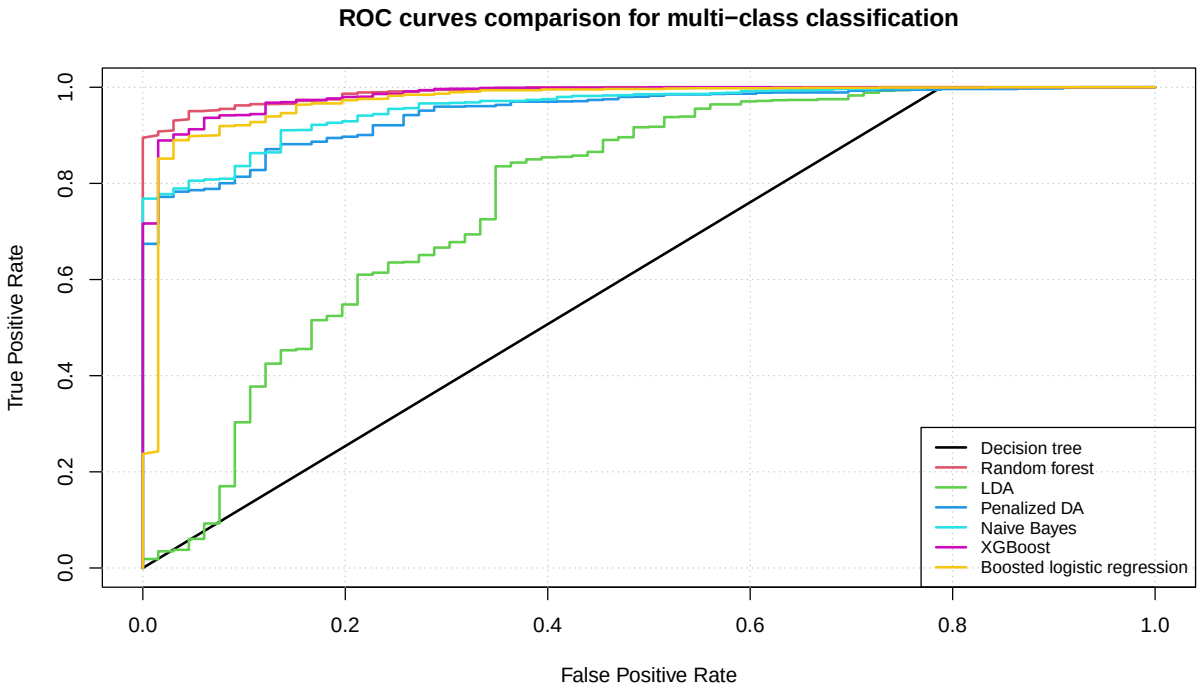
III.II. Multi-Class Classification

In this iteration, we will train the our models using the same ML algorithms, with the exception that this time we are going to train them to classify the **failure_type** columns instead of the **failure** column:

```
# multi-class classification
my_multiclass_target   <- "failure_type"

# train the models in lapply
invisible(capture.output(
  trained_models_multiclass <- lapply(methods, function(method){
    print(method)
    model <- train_model(df_training=train_set,
                         target=my_multiclass_target,
```

```
                        features=my_features,
                        method=method,
                        control=ctrl)
    model
  })
))
```

For evaluating the model performance, we will not distinguish between different positive classes (i.e. if the model detected the failure but the mode of the failure is wrong, we still count it as a true positive). Consequently, the ROC curves look like this:



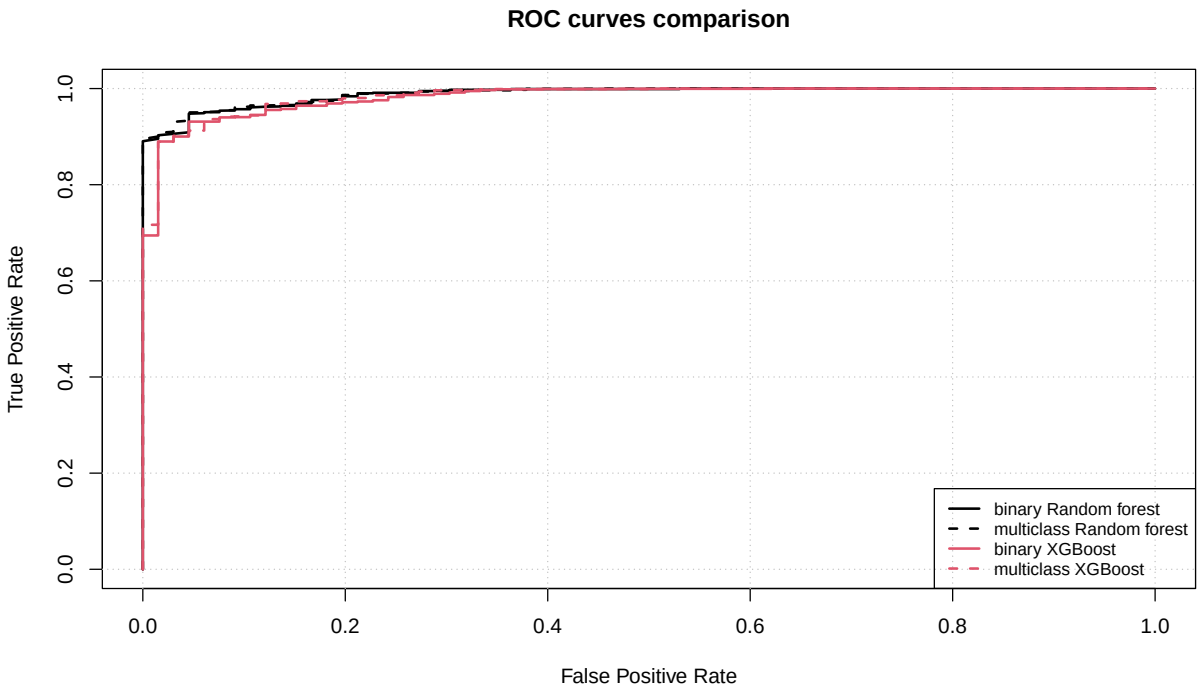
The TPR metric at 10% FPR shows substantial improvement for some of the models:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.4320	0.6204	0.9418	0.7706
multiclass classification	0.1268	0.9615	0.2988	0.8154	0.8389	0.9426	0.9226

The F1-scores changed for each model in different ways. This shows that we need more improvements to get to the desired levels of model performance:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.7500	0.3883	0.3918	0.3810	0.7290	0.3636
multiclass classification	0.3415	0.7184	0.3182	0.3023	0.4091	0.7679	0.4928

Here is the comparison of the ROC curves compared to the binary classification case:



To find out the weak point of our models, we can look at the confusion matrices for the best performing models. The following tables are the confusion matrices for the XGBoost and the random forest models respectively:

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	20	0	1	0	0
No Failure	2	1930	1	6	13
Overstrain Failure	0	1	12	0	0
Power Failure	0	3	0	11	0
Tool Wear Failure	0	0	0	0	0

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	19	0	0	0	0
No Failure	2	1931	9	3	12
Overstrain Failure	1	2	5	1	1
Power Failure	0	1	0	13	0
Tool Wear Failure	0	0	0	0	0

It seems to be the case that the “Tool Wear Failure” is the most difficult type of failure to detect. This may be related to the fact that “Tool Wear Failure” is the least common type of failure in our data set, as shown in the analysis section. To overcome this problem, we will employ SMOTE as described previously in this report.

III.III. Multi-Class Classification with SMOTE

For applying SMOTE on our data set, we will use the "SMOTE" function of the "smotefamily" package in R. However, since this function is designed for binary classifications, we define another function that applies SMOTE to each minority class separately. Although we will oversample the minorities, it is not recommended to oversample them to the point of having a completely balanced data set. This is because some of the models use the a priori probabilities in their predictions, and since we are oversampling only in training set to avoid data leakage, once these models are employed to predict on the test set, they may have too many false positives. After applying SMOTE to our training set, we can see how the distribution of the minority classes has changed:

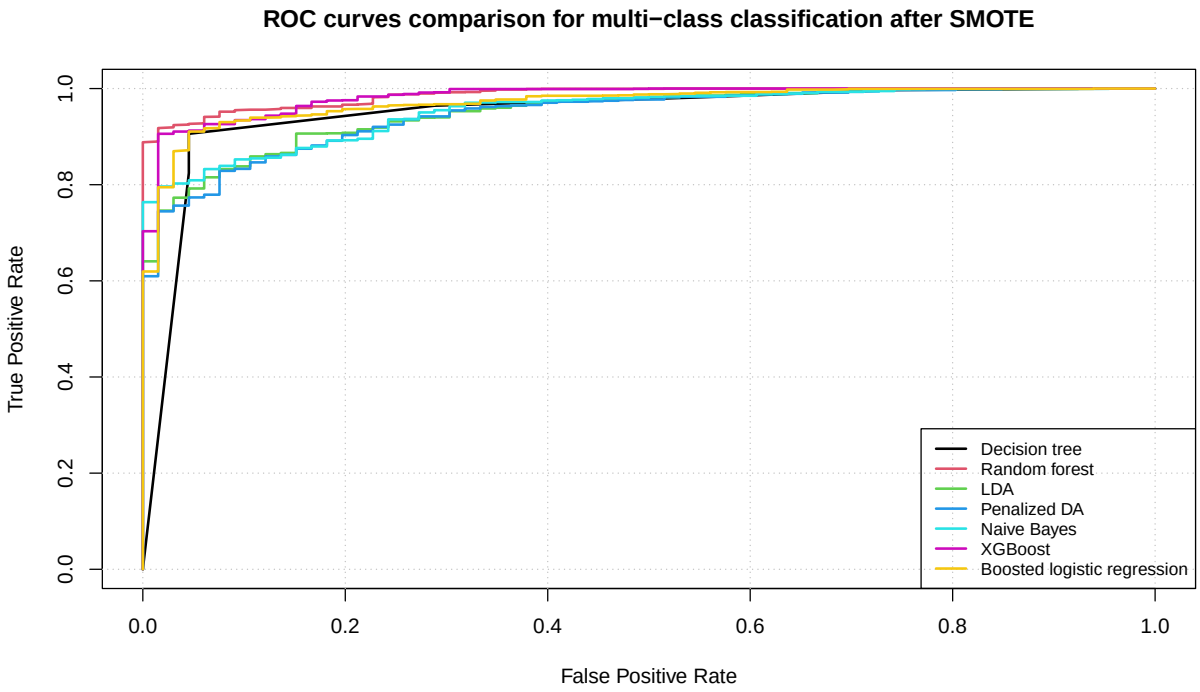
```
train_set_subclass_smote <- smote_for_failure_type(
  train_data=train_set,
  target_col=my_binary_target,
  positive_class=positive_class,
  subclass_type_col=my_multiclass_target,
  features_cols=my_features,
  apprx_subclass_pop=n_subclass_synth)

# show the resulting distribution of target column after SMOTE
knitr::kable(table(train_set$failure_type), col.names = c("Failure Type","Count"))
```

Failure Type	Count
Heat Dissipation Failure	90
No Failure	7736
Overstrain Failure	64
Power Failure	78
Tool Wear Failure	32

```
knitr::kable(table(train_set_subclass_smote$failure_type), col.names = c("Failure Type","Count"))
```

Failure Type	Count
Heat Dissipation Failure	540
No Failure	7736
Overstrain Failure	512
Power Failure	546
Tool Wear Failure	512

As it can be seen, the distribution has changed and now we have around 500 positive sample for each minority class. After we train our models on this modified data set, we get the following ROC curves:



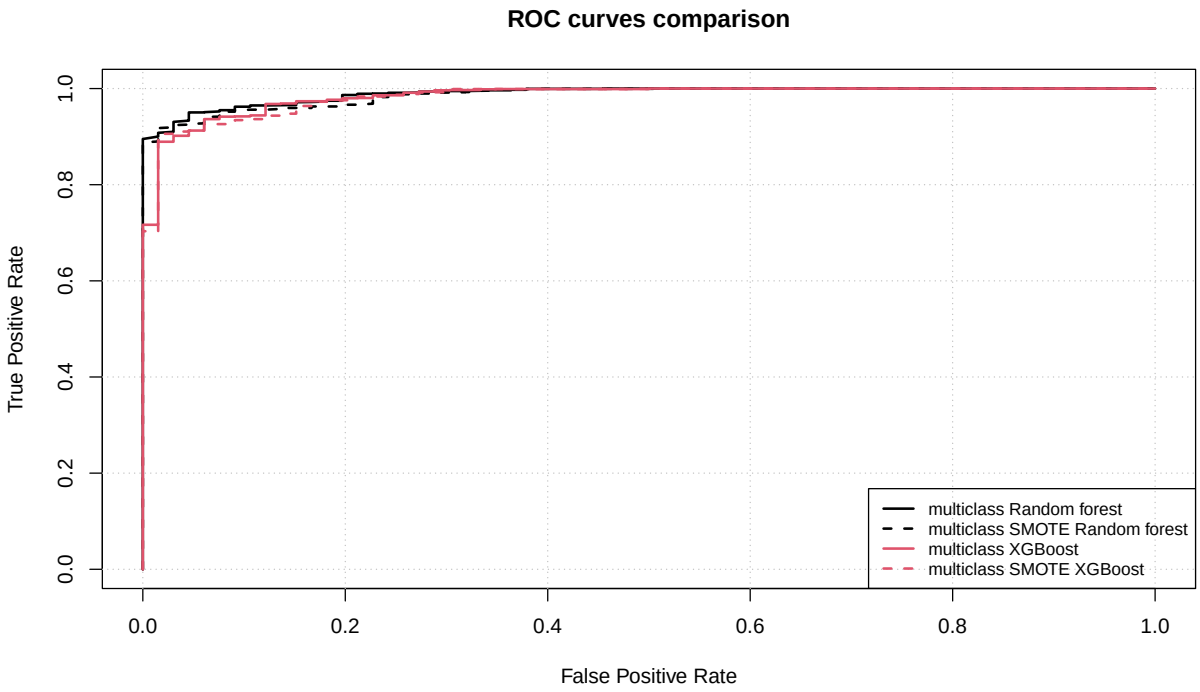
The TPR metric at 10% FPR after SMOTE is applied shows us that some models like decision tree and LDA show significant improvement:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.4320	0.6204	0.9418	0.7706
multiclass classification	0.1268	0.9615	0.2988	0.8154	0.8389	0.9426	0.9226
multiclass + SMOTE	0.8874	0.9551	0.8432	0.8362	0.8506	0.9333	0.9346

Although the F1-scores after SMOTE seems to have improved for most models, but it did not improved a lot for random forest, and it even got worse for XGBoost:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.7500	0.3883	0.3918	0.3810	0.7290	0.3636
multiclass classification	0.3415	0.7184	0.3182	0.3023	0.4091	0.7679	0.4928
multiclass + SMOTE	0.3599	0.7317	0.4615	0.4459	0.5000	0.7377	0.6446

The following figure shows the ROC curves before and after applying SMOTE:



We can have a look at the confusion matrices to see if SMOTE has helped in detecting the "Tool Wear Failure". The following tables are the confusion matrices for the XGBoost and the random forest models respectively:

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	20	0	1	0	0
No Failure	2	1922	1	4	13
Overstrain Failure	0	2	12	0	0
Power Failure	0	3	0	13	0
Tool Wear Failure	0	7	0	0	0

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	19	4	0	0	0
No Failure	2	1920	1	5	11
Overstrain Failure	1	3	13	1	0
Power Failure	0	4	0	11	0
Tool Wear Failure	0	3	0	0	2

As it can be seen, we still fail in detecting the "Tool Wear Failure" instances. The application of SMOTE did not drastically changed the performance of our best models. Therefore, Further improvements are necessary.

III.IV. Multi-Class Classification with DBSMOTE

DBSMOTE is an extension of the regular SMOTE, but it usually performs better due to avoiding oversampling in noisy areas. We will employ DBSOMTE in a similar way as we did for SMOTE, and inspect the distribution of failure types before and after:

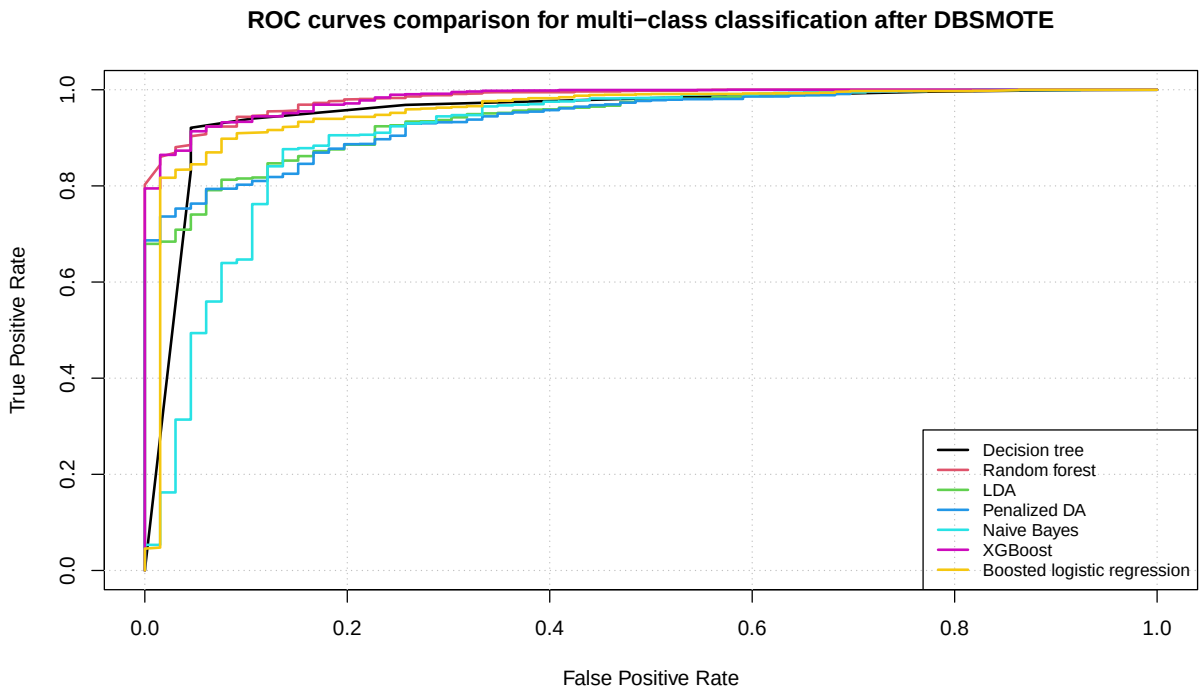
```
# show the resulting distribution of target column after SMOTE
knitr::kable(table(train_set$failure_type), col.names = c("Failure Type","Count"))
```

Failure Type	Count
Heat Dissipation Failure	90
No Failure	7736
Overstrain Failure	64
Power Failure	78
Tool Wear Failure	32

```
knitr::kable(table(train_set_subclass_DBSMOTE$failure_type), col.names = c("Failure Type","Count"))
```

Failure Type	Count
Heat Dissipation Failure	495
No Failure	7736
Overstrain Failure	456
Power Failure	480
Tool Wear Failure	452

Using this new data set, we train our models and look at the new ROC curves:



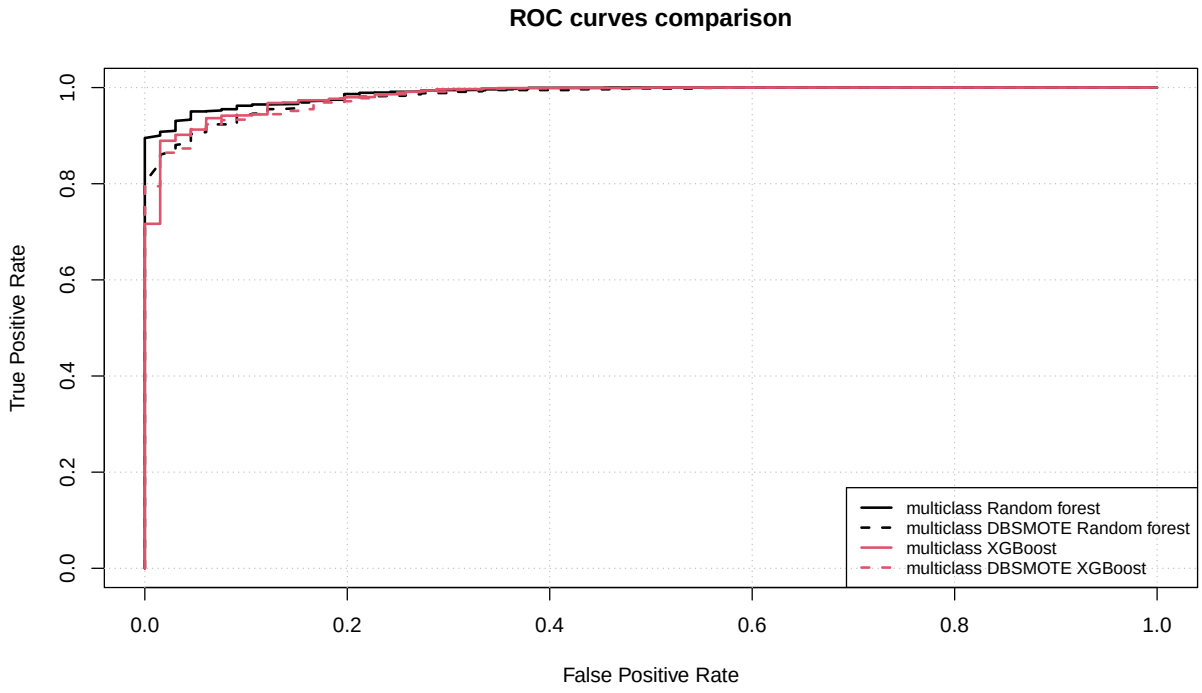
The curves do not look very different from the last time. We can confirm this by checking out the TPR at 10% FPR for each model:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.4320	0.6204	0.9418	0.7706
multiclass classification	0.1268	0.9615	0.2988	0.8154	0.8389	0.9426	0.9226
multiclass + SMOTE	0.8874	0.9551	0.8432	0.8362	0.8506	0.9333	0.9346
multiclass + DBSMOTE	0.9336	0.9401	0.8155	0.8032	0.6800	0.9365	0.9064

Our F1-scores seem to have gotten worse in all cases except for the decision tree classifier:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.7500	0.3883	0.3918	0.3810	0.7290	0.3636
multiclass classification	0.3415	0.7184	0.3182	0.3023	0.4091	0.7679	0.4928
multiclass + SMOTE	0.3599	0.7317	0.4615	0.4459	0.5000	0.7377	0.6446
multiclass + DBSMOTE	0.4332	0.7009	0.4000	0.4255	0.4286	0.7368	0.5962

We can also compare the ROC curves before and after applying DBSMOTE:



Confusion matrices for XGBoost and random forest after applying DBSMOTE are shown below. Yet again, we failed to adequately detect the "Tool Wear Failure" instances. Therefore, we will take more measures to improve the model performance.

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	19	0	1	1	0
No Failure	2	1924	2	4	12
Overstrain Failure	1	1	11	1	0
Power Failure	0	3	0	11	0
Tool Wear Failure	0	6	0	0	1

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	17	4	0	0	0
No Failure	4	1922	3	4	12
Overstrain Failure	1	1	11	1	0
Power Failure	0	3	0	12	0
Tool Wear Failure	0	4	0	0	1

III.V. Multi-Class Classification with DBSMOTE/ENN

ENN, as described previously in this report, is used to de-noise the data by undersampling the instances in noisy regions in our feature space. Combination ENN with one of the variations of SMOTE is a pretty common practice in data science. Here we combine DBSMOTE and ENN to see if we can improve our model performance:

```
# apply ENN
train_set_subclass_DBSMOTE_ENN <- apply_ENN(df=train_set_subclass_DBSMOTE,
                                            target=my_multiclass_target,
                                            features=my_features)
```

The distribution of failure type after applying ENN to the data set produced by DBSOMTE, we can see that some instances have been removed from the data set as expected.

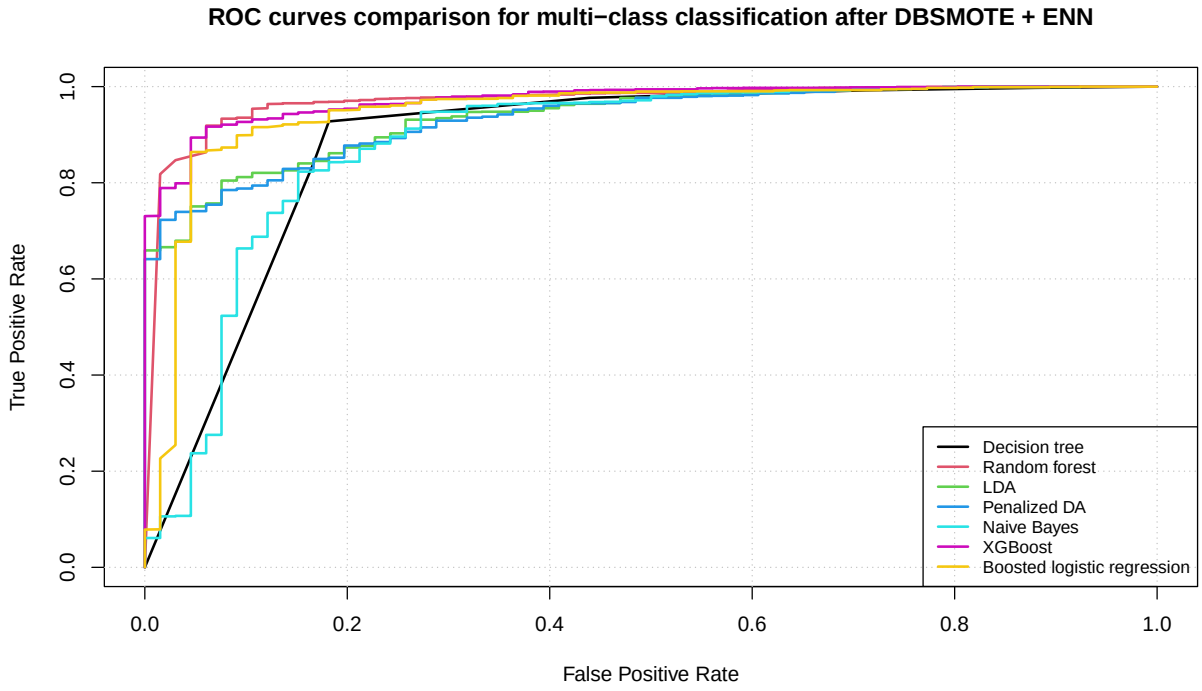
```
# show the resulting distribution of target column after SMOTE
knitr::kable(table(train_set_subclass_DBSMOTE$failure_type), col.names = c("Failure Type","Count"))
```

Failure Type	Count
Heat Dissipation Failure	495
No Failure	7736
Overstrain Failure	456
Power Failure	480
Tool Wear Failure	452

```
knitr::kable(table(train_set_subclass_DBSMOTE_ENN$failure_type), col.names = c("Failure Type","Count"))
```

Failure Type	Count
Heat Dissipation Failure	403
No Failure	7345
Overstrain Failure	444
Power Failure	421
Tool Wear Failure	397

After we using this modified data set to train our models, we get the following ROC curves from our trained models:



We can look at our metrics to see if the model performance has improved. The TPR values at 10% FPR show us that ENN did not improve model performances;

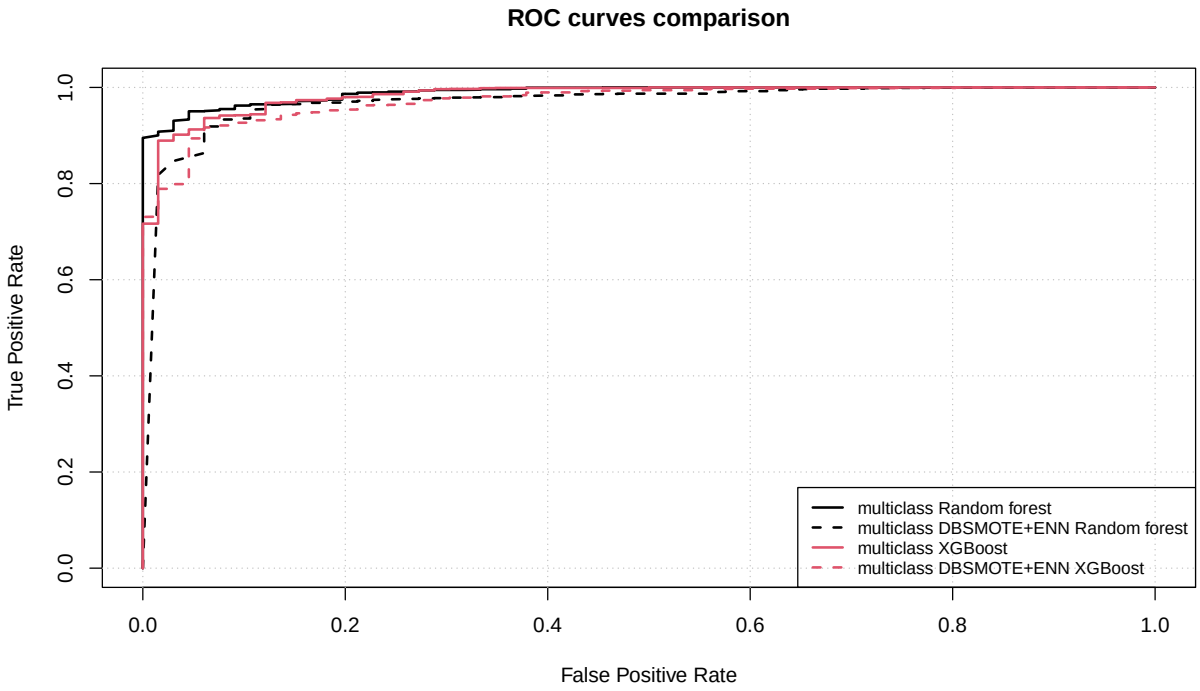
	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.4320	0.6204	0.9418	0.7706
multiclass classification	0.1268	0.9615	0.2988	0.8154	0.8389	0.9426	0.9226
multiclass + SMOTE	0.8874	0.9551	0.8432	0.8362	0.8506	0.9333	0.9346
multiclass + DBSMOTE	0.9336	0.9401	0.8155	0.8032	0.6800	0.9365	0.9064
multiclass + DBSMOTE/ENN	0.5044	0.9403	0.8130	0.7892	0.6427	0.9270	0.8993

The following table compares contains the updated F1-scores after DBSOMTE/ENN. We can clearly see the decrease in performance for our best 2 models. Therfore, we will stop using this strategy (DBSMOTE+ENN).

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.7500	0.3883	0.3918	0.3810	0.7290	0.3636
multiclass classification	0.3415	0.7184	0.3182	0.3023	0.4091	0.7679	0.4928

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
multiclass + SMOTE	0.3599	0.7317	0.4615	0.4459	0.5000	0.7377	0.6446
multiclass + DBSMOTE	0.4332	0.7009	0.4000	0.4255	0.4286	0.7368	0.5962
multiclass + DBSMOTE/ENN	0.3613	0.5455	0.3975	0.3919	0.4235	0.6290	0.5818

The following plot compares the ROC curves before and after DBSMOTE/ENN for our 2 best models. We can see that the models perform worse after applying DBSMOTE/ENN to the data set.



Having a look at our confusion matrices for XGBoost and random forest, we can see that we did not succeed in detecting "Tool Wear Failure", and we have significantly more false positives:

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	14	0	0	0	0
No Failure	7	1913	3	4	11
Overstrain Failure	1	7	11	1	0
Power Failure	0	1	0	12	0
Tool Wear Failure	0	13	0	0	2

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	12	6	0	0	0
No Failure	9	1902	3	5	11
Overstrain Failure	1	8	11	1	0

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Power Failure	0	5	0	11	0
Tool Wear Failure	0	13	0	0	2

III.VI. Multi-Class Classification in LD Space

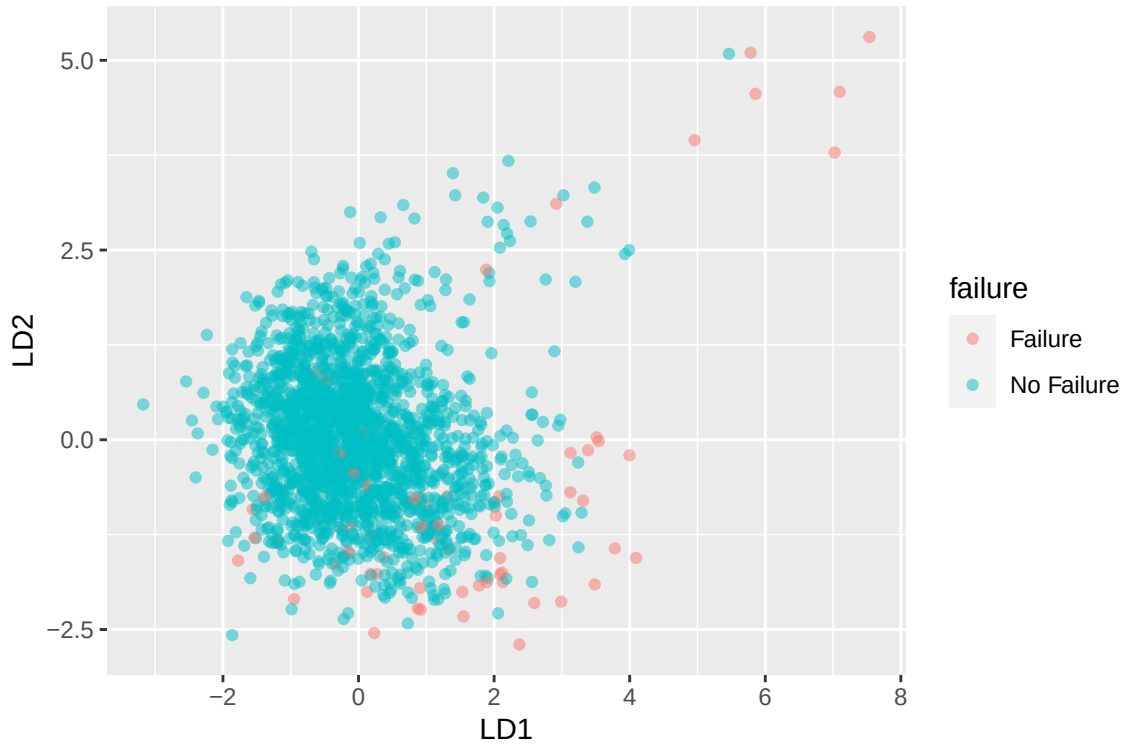
As we have failed to improve the model performance by means of SMOTE, DBSMOTE, and ENN, it makes sense to try a different approach. As described in the analysis section of this report, we can project our features into a lower dimensional space of linear discriminants (LDs) to improve separability of our classes and minimize the in-class variations. The LD space is in fact the space where the LDA model classifies the instances. Here we try to classify instances in the LD space but using models other than LDA. First we project the training set onto the LD space:

```
# extract features and target
train_set_pre_LDA <- train_set[c(my_features,my_multiclass_target)]

# apply LDA
lda_result <- MASS::lda(failure_type ~ ., data = train_set_pre_LDA)

# put labels back in the dataframe
train_set_LDA <- data.frame(predict(lda_result, train_set_pre_LDA)$x)
train_set_LDA[[my_binary_target]] <- train_set[[my_binary_target]]
train_set_LDA[[my_multiclass_target]] <- train_set[[my_multiclass_target]]
```

After this projection, we project the test set using the same coefficients we obtained during the projection of the training set. This is done to avoid data leakage:

```
# transform the test set with the same coefficient
test_set_LDA <- data.frame(predict(lda_result, test_set)$x)
test_set_LDA[[my_binary_target]] <- test_set[[my_binary_target]]
test_set_LDA[[my_multiclass_target]] <- test_set[[my_multiclass_target]]
```

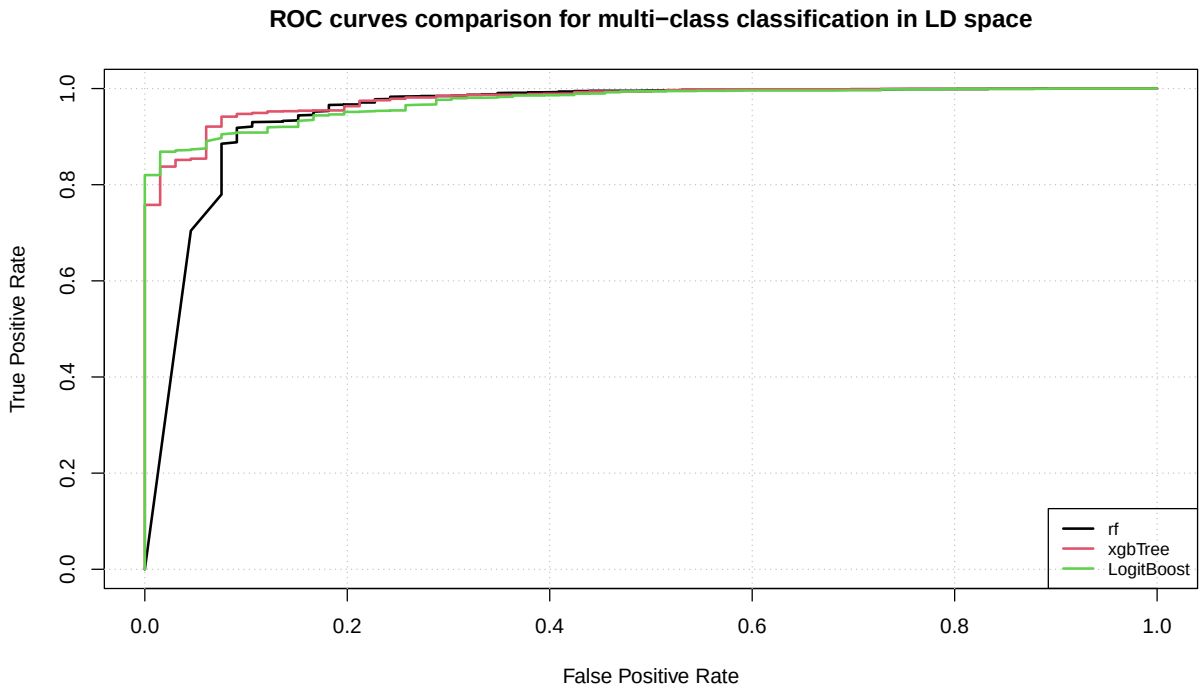
Here is a plot of our training set data projected onto the plane defined by the first 2 linear discriminants:



It seems like the data is more separable in LD space when we compare this plot to the ones provided in the analysis section. Here are the models we are going to train on the data projected into the 4 dimensional LD space:

```
# redefine methods
new_methods      <- c("rf", "xgbTree",  "LogitBoost")
new_mehtod_names <- c("Random forest","XGBoost", "Boosted logistic regression")
```

And after training our models, we obtain the following ROC curves:



It does not seem like an improvement. We can confirm that by looking into our metrics. The table below shows the TPR at 10% FPR:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.4320	0.6204	0.9418	0.7706
multiclass classification	0.1268	0.9615	0.2988	0.8154	0.8389	0.9426	0.9226
multiclass + SMOTE	0.8874	0.9551	0.8432	0.8362	0.8506	0.9333	0.9346
multiclass + DBSMOTE	0.9336	0.9401	0.8155	0.8032	0.6800	0.9365	0.9064
multiclass + DBSMOTE/ENN	0.5044	0.9403	0.8130	0.7892	0.6427	0.9270	0.8993
multiclass in LD space	NA	0.9178	NA	NA	NA	0.9467	0.9096

The F1-score values obtained from this approach show significant worsening of the performance of our models:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.7500	0.3883	0.3918	0.3810	0.7290	0.3636
multiclass classification	0.3415	0.7184	0.3182	0.3023	0.4091	0.7679	0.4928
multiclass + SMOTE	0.3599	0.7317	0.4615	0.4459	0.5000	0.7377	0.6446
multiclass + DBSMOTE	0.4332	0.7009	0.4000	0.4255	0.4286	0.7368	0.5962
multiclass + DBSMOTE/ENN	0.3613	0.5455	0.3975	0.3919	0.4235	0.6290	0.5818
multiclass in LD space	NA	0.3613	NA	NA	NA	0.5455	0.3975

III.VII. Multi-Class Classification + Feature Engineering

So far we have used different ML algorithms, oversampling techniques, de-noising the data, and projecting features onto lower transformed spaces. However, in every case we used the same set of features. Maybe

the key to improve the model performance is to come up with better features. Coming from an engineering background, it was not difficult to define the following new features:
1) power: Multiplication of rpm and toque.
2) strain: The cumulative stain on the machine, calculated by multiplying the torque and the wear (in minutes).
3) delta_T: The difference between process temperature and the air temperature.

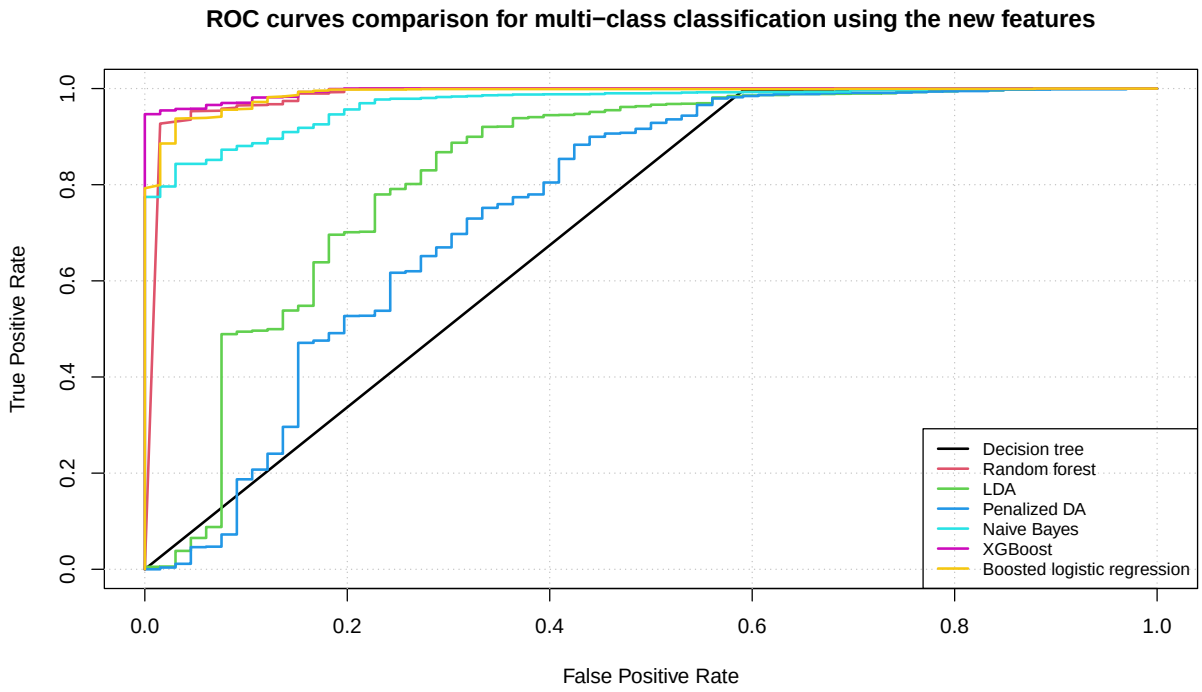
Let us create the necessary columns in the training set and the test set:

```
# define new features which make physical sense
train_set <- train_set %>%
  mutate(strain=wear*torque,
         delta_T= process_t-air_t,
         power=rpm*torque)

# define the same features for the test set
test_set <- test_set %>%
  mutate(strain=wear*torque,
         delta_T= process_t-air_t,
         power=rpm*torque)
```

We will use the same 7 ML algorithms, but we will change the features we will use for the training:

```
# methods and respective names to train models
methods      <- c("rpart", "rf", "lda2","pda2", "nb", "xgbTree",  "LogitBoost")
mehtod_names <- c("Decision tree", "Random forest", "LDA", "Penalized DA",
                  "Naive Bayes", "XGBoost", "Boosted logistic regression")

# features to use for training and target to predict
my_features <- c("delta_T", "power", "strain", "rpm", "torque",
                 "wear", "type_H", "type_L", "type_M")

my_multiclass_target   <- "failure_type"
```

After training our models using the new features, we obtain the following ROC curves:



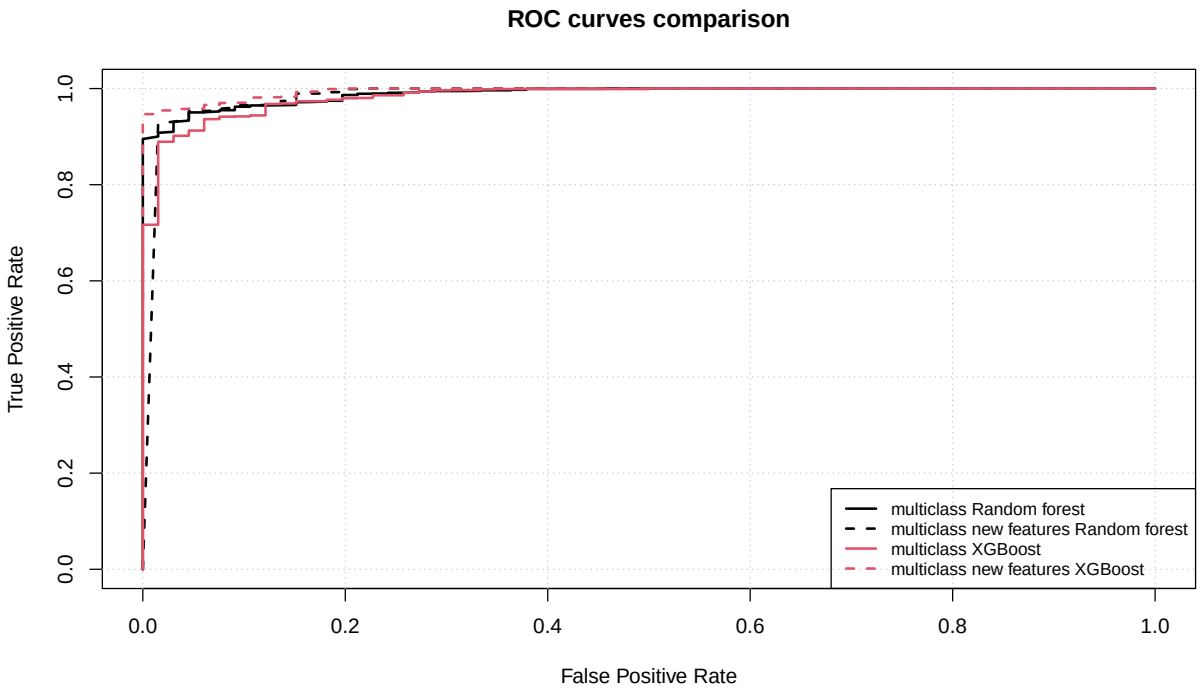
We can already see a significant improvement in our ROC curves. We can confirm the improvement in our models by calculating the TPR at 10% FPR for each model

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.1268	0.9579	0.4113	0.4320	0.6204	0.9418	0.7706
multiclass classification	0.1268	0.9615	0.2988	0.8154	0.8389	0.9426	0.9226
multiclass + SMOTE	0.8874	0.9551	0.8432	0.8362	0.8506	0.9333	0.9346
multiclass + DBSMOTE	0.9336	0.9401	0.8155	0.8032	0.6800	0.9365	0.9064
multiclass + DBSMOTE/ENN	0.5044	0.9403	0.8130	0.7892	0.6427	0.9270	0.8993
multiclass in LD space	NA	0.9178	NA	NA	NA	0.9467	0.9096
multiclass + new features	0.1685	0.9638	0.4939	0.1703	0.8807	0.9737	0.9623

We see significant improvement for most of the models. We can see similar results in our F1-scores:

	rpart	rf	lda2	pda2	nb	xgbTree	LogitBoost
binary classification	0.3415	0.7500	0.3883	0.3918	0.3810	0.7290	0.3636
multiclass classification	0.3415	0.7184	0.3182	0.3023	0.4091	0.7679	0.4928
multiclass + SMOTE	0.3599	0.7317	0.4615	0.4459	0.5000	0.7377	0.6446
multiclass + DBSMOTE	0.4332	0.7009	0.4000	0.4255	0.4286	0.7368	0.5962
multiclass + DBSMOTE/ENN	0.3613	0.5455	0.3975	0.3919	0.4235	0.6290	0.5818
multiclass in LD space	NA	0.3613	NA	NA	NA	0.5455	0.3975
multiclass + new features	0.4946	0.8793	0.4000	NaN	0.5714	0.8833	0.8545

The F1-scores are now very close to the desired level mentioned in the introduction section. We can also visually compare the model performances before and after adding the new features by looking at the ROC curves:



And here are the confusion matrices for XGBoost and random forest after adding the new features to our data set:

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	21	0	0	0	0
No Failure	1	1933	0	0	12
Overstrain Failure	0	0	14	0	0
Power Failure	0	0	0	17	0
Tool Wear Failure	0	1	0	0	1

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	20	0	0	0	0
No Failure	1	1934	0	0	13
Overstrain Failure	1	0	14	0	0
Power Failure	0	0	0	17	0
Tool Wear Failure	0	0	0	0	0

Although we still cannot address the “Tool Wear Failure” class as adequately as we may desire, but the rest of the cases seem pretty good in general. At this point, we can stop trying new techniques, and make an ensemble of our best models.

III.VIII. The Ensemble

We will use our 3 best models from the last section to create the ensemble, and we will use hard voting to carry out the classification task. The multi-class confusion matrix of the ensemble model is provided below:

```
# models to be used in ensemble
ensemble_methods <- c("rf","xgbTree", "LogitBoost")

# predict based on voting of models in ensemble
ensemble_predictions <- predict_ensemble(
  models=trained_models_subclass_physical_features,
  ensemble_methods=ensemble_methods,
  new_data=test_set)

# assemble confusion matrix
ensemble_prediction_table <- table(ensemble_predictions, test_set$failure_type)
knitr::kable(ensemble_prediction_table)
```

	Heat Dissipation Failure	No Failure	Overstrain Failure	Power Failure	Tool Wear Failure
Heat Dissipation Failure	22	0	0	0	0
No Failure	0	1934	0	0	13
Overstrain Failure	0	0	14	0	0
Power Failure	0	0	0	17	0

We can see that in total we have 13 false negatives, all belonging to the "Tool Wear Failure" class. And we have no False positives. Here is the binary confusion matrix of the ensemble model:

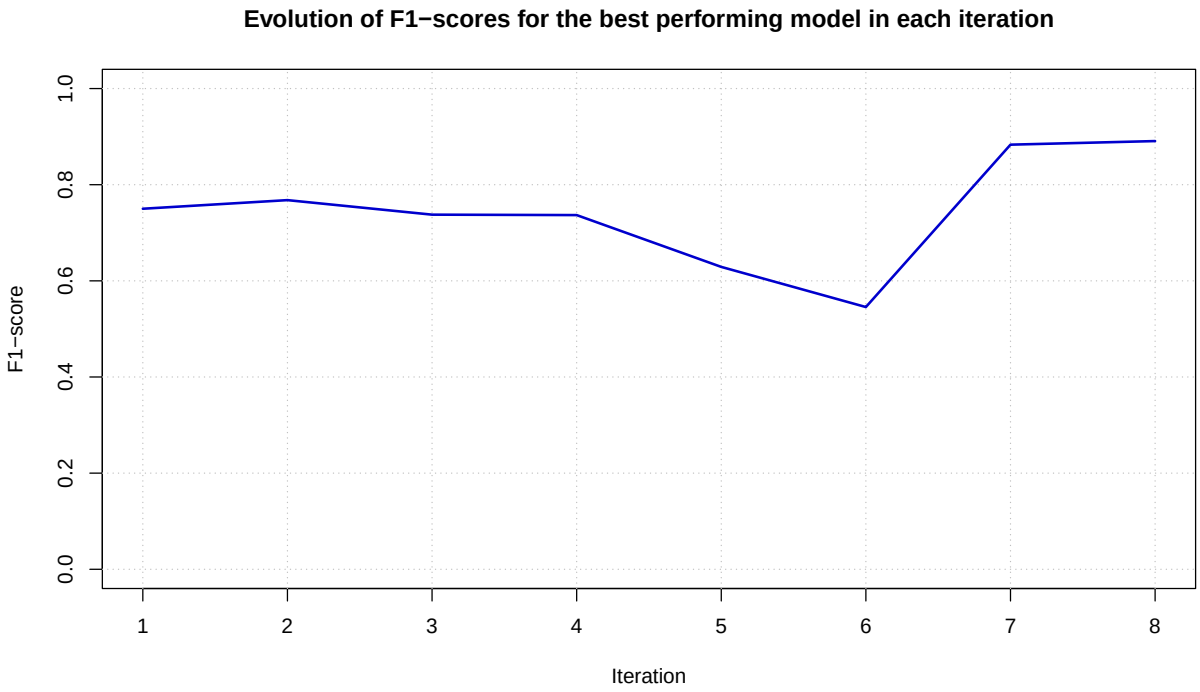
	Failure	No Failure
Predicted Failure	53	0
Predicted No Failure	13	1934

We can easily calculate our final F1-score:

```
# calculate f1-score
get_f1_score(binary_confusion_matrix)
```

```
## [1] 0.8907563
```

This is very close to our stated goal of 0.9 F1-score in the introduction section. Therefore, further iterations of the model is not needed. The plot below shows the maximum F1-scores in each iteration of model development, with the last point being the ensemble model:



Since the final model is an ensemble of 3 models, it would be a cumbersome operation to derive an ROC curve for it. For the same reason, we do not state a TPR at 10% FPR for the final model.

IV. Conclusions

In this project, we developed the basis for a predictive maintenance model for industrial machines give in our data set. Firstly, we investigated the data set, observed the unbalance within the target columns, and did some data cleaning. Then, we carried out an exploratory data analysis (EDA) and made the necessary visualizations on the data set to find out about correlations within our data set. Then we started training models using 7 different ML algorithms. We applied different techniques such as SMOTE, DBSMOTE, ENN, projection onto LD space, and feature engineering in order to enhance our model performance and the F1-score. Once we brought the F1-score to the desired level, we created and ensemble of our best models and presented the results for the final model. The final model has perfect prediction capabilities with the exception of 1 failure class. To improve this, one can try to train better models by defining the proper grid search parameters and tuning the hyper parameters of the model. Another possible way to improve the model would be to create a digital twin of the monitored system and to generate synthetic data for the failure case which is difficult to detect. This approach will probably result in better synthetic samples than the ones generated by SMOTE or DBSMOTE.

V. References

[1] https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification (accessed on 15.02.2024, at 20:37 CET)

[2] Bunkhumpornpat, C., Sinapiromsaran, K. & Lursinsap, C. DBSMOTE: Density-Based Synthetic Minority Over-sampling TEchnique. Appl Intell 36, 664–684 (2012). https://doi.org/10.1007/s10489-011-0287-y

[3] Wilson, Dennis L.. "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data." IEEE Trans. Syst. Man Cybern. 2 (1972): 408-421.