Università degli Studi di Catania

Statistical Learning

Predictive Maintenance

Orazio Pontorno
Giuseppe Pulino

July 29, 2022

Contents

1 Introduction

The purpose of the report is to build a classification model using the 'Predictive maintenance' dataset, which consists of 10 000 data points stored as rows with 8 features in columns. The classifier will have to be able to predict the target variable, which takes value '0' if the machine has no failure and therefore no maintenance is needed, '1' if on the contrary, if some kind of damage has been revealed in the machine and it needs maintenance.

The dataset is taken from kaggle: `https://www.kaggle.com/shivamb/machine-predictive-maintenance-classification`.

To build such a model, the original dataset was split into 3 parts: the 'training set' (60% of the data), the 'validation set' (20% of the data) and the 'test set' (20% of the data). Our model will be built by the use of training and validation. In particular, the former will be used for us to construct three classification models such as the 'logistic regression', the 'random forest' and a 'neural network'. The task of the second, on the other hand, will be to choose which of the three models performs best.

In the following table the predictor variables are presented:

Variable	Type	Description
UDI	chr	*Unique identifier ranging from 1 to 10000*
Product.ID	chr	*Unique identifier consisting in an alphanumeric string*
Type	Ord. Factor	*Consisting of a letter L, M, or H for low (50% of all products), medium (30%), and high (20%) as product quality variants and a variant-specific serial number*
Ait.temperature	num	*Generated using a random walk process later normalized to a standard deviation of 2 K around 300 K*
Process.temperature	num	*Generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K*
Rotational.speed	int	Calculated from power of 2860 W, overlaid with a normally distributed noise
Torque	num	*Torque values are normally distributed around 40 Nm with an If = 10 Nm and no negative values*
Tool.wear	int	*The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process*
Target	binary	*The Target variable displays if the machine is failed or not*

Table 1: dataset's variables

So the dataset consists of nine variables, of which 2 are categorical, one is a Ordered Factor and 5 are numeric/integer, while our response is binary.

1.1 Business question

Business questions are those questions that will guide our analysis and construction of the classifier. Indeed, it is through it that we will attempt to answer most of the questions.

- *Which approach allows the classification model to best predict the target variable?*
- *Which variables have the greatest weight in predicting the target variable? and which have the least?*
- *How accurately can we estimate a possible failure?*

Before moving on to model construction, it is necessary to analyze the dataset; in particular, it is important to analyze the distribution of individual predictor variables and the possible relationships present among them. To do this, an Explorative Data Analysis (EDA) will now be carried out.

2 Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is used to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. All the variables in the training set are then analysed, first individually and then all together.

2.1 Univariate Analysis

This is simplest form of data analysis, where the data being analyzed consists of just one variable. The main purpose of univariate analysis is to summarize the main features of each variable.

Quantitative variables

Let's start to analyze the numeric variables.

2.1.1 Air.temperature

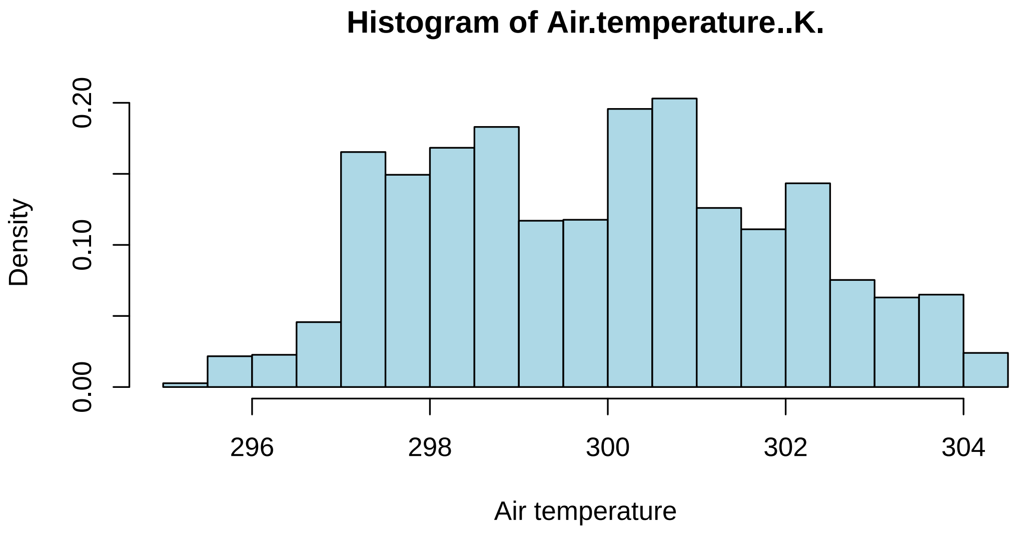


Figure 1: histogram of Air.temperature

Min	1st Qu	Median	Mean	3rd Qu	Max
295.4	298.3	300.1	300.0	301.5	304.5

Table 2: summary table of Air.temperature

skewness	kurtosis
0.1219684	2.156503

From the graph we can notice the presence of two peaks, yielding us thinking that there may be two subpopulation that influence the result of whole histogram. From the fact that the skewness value is very close to zero, we can deduce and find that the distribution of the data is almost symmetrical. In contrast, a kurtosis value of less than 3 indicates to us that the distribution is heavy-tailed, and the peak can be flatter, almost like punching the distribution or squishing it (Platykurtic).

2.1.2 Process.temperature

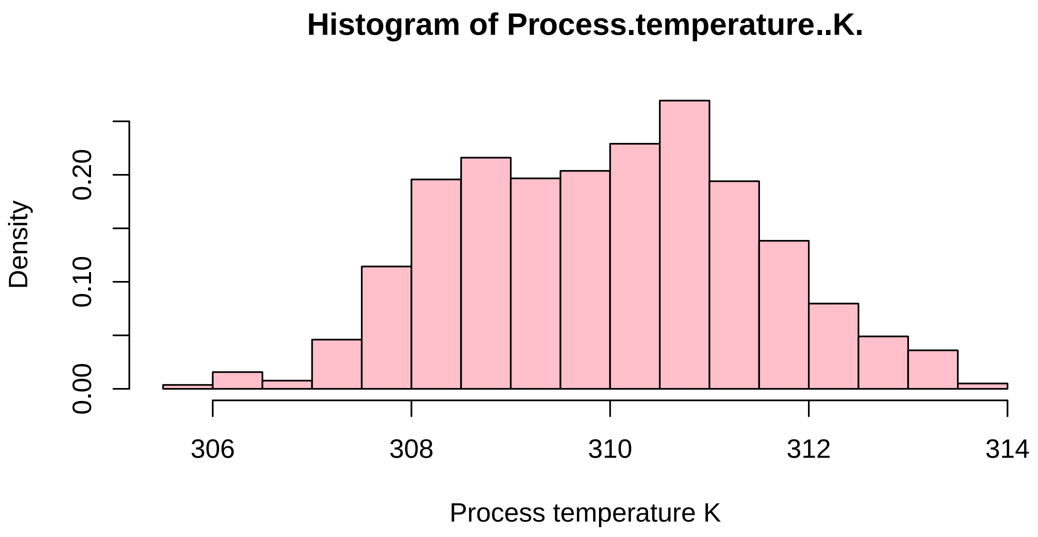


Figure 2: histogram of Process.temperature

Min	1st Qu	Median	Mean	3rd Qu	Max
205.8	308.8	310.1	310.0	311.1	313.8

Table 3: summary table of Air.temperature

skewness	kurtosis
0.02548641	-0.5098995

Again, it can be seen that the distribution of the data is almost symmetrical, which is also confirmed by the low skewness value. The distribution also has a negative kurtosis value, so it has a heavy tail and the peak may be flatter.

2.1.3 Rotational.speed

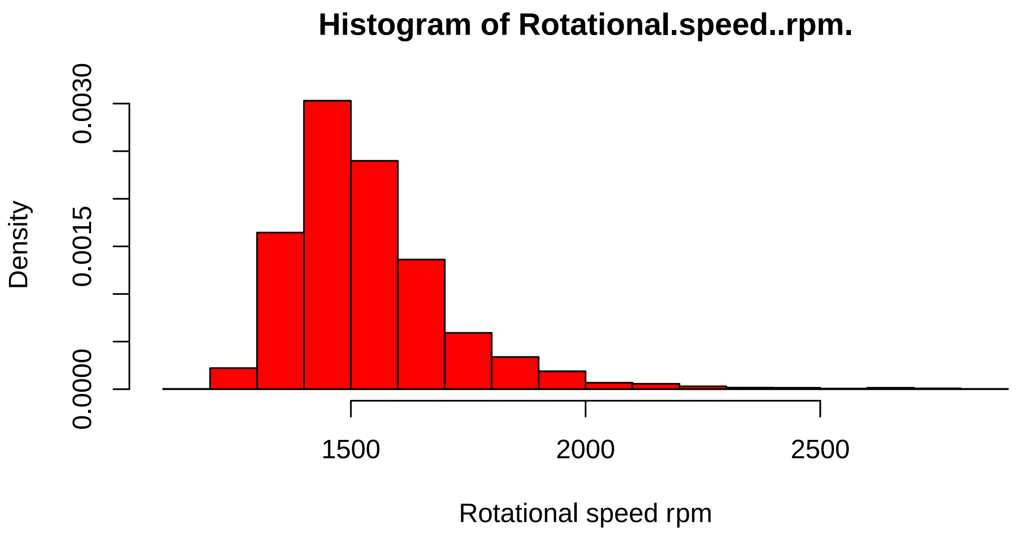


Figure 3: histogram of Rotational.speed

Min	1st Qu	Median	Mean	3rd Qu	Max
1181	1423	1504	1539	1611	2874

Table 4: summary table of Rotational.speed

skewness	kurtosis
1.993557	7.405396

Unlike the previous ones, the data distribution of the variable Rotational.speed has a single peak, with a higher skewness value, which can be seen from the fact that a large part of the data lies to the left. The kurtosis value is also very different from the previous cases, in fact it has a very high value and above all is greater than 3. This indicates that the distribution is light-tailed and the top curve steeper, like pulling up the distribution (Leptokurtic).

2.1.4 Torque

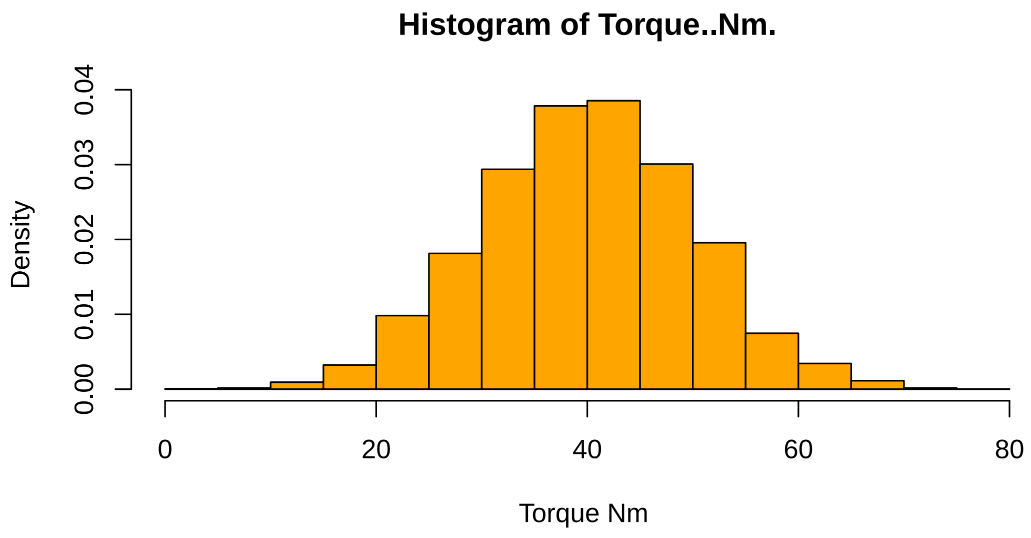


Figure 4: histogram of Torque

Min	1st Qu	Median	Mean	3rd Qu	Max
4.20	33.20	40.10	40.01	46.80	76.20

Table 5: summary table of Torque

skewness	kurtosis
-0.003551766	-0.03281282

The distribution of the data for the variable Torque shows a skewness value very close to zero; the symmetry of the data distribution can therefore be seen. Likewise, the kurtosis value is also very close to zero, but still less than 3. Therefore, this data distribution also has a heavy tail and the peak may be flatter.

2.1.5 Tool.wear

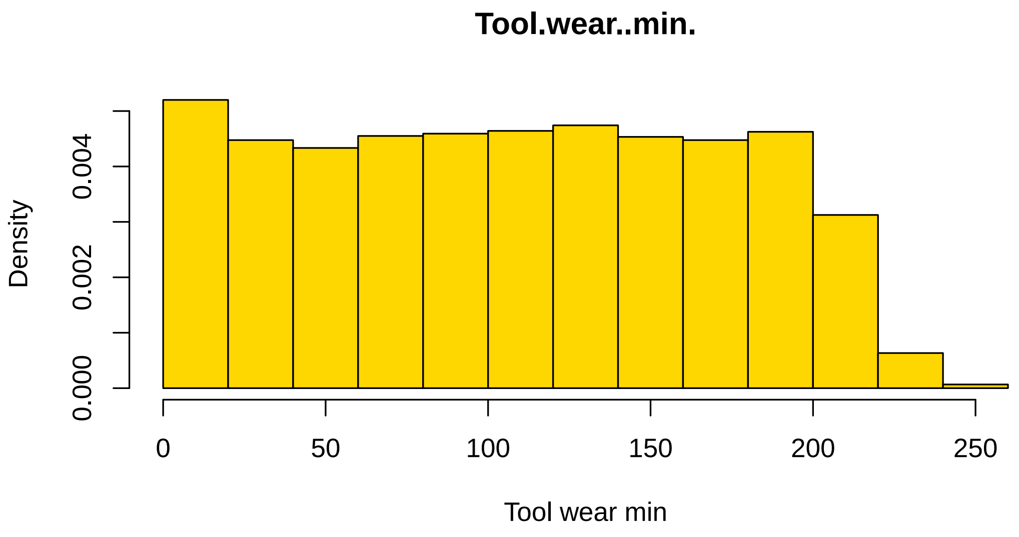


Figure 5: histogram of Tool.wear

Min	1st Qu	Median	Mean	3rd Qu	Max
0.0	53.0	109.0	108.2	162.0	253.0

Table 6: summary table of Tool.wear

skewness	kurtosis
0.01914566	-1.152117

From the graph, we see that the distribution is almost symmetrical, which is also confirmed by the very close to zero skewness value. In addition, a negative kurtosis value also appears; this is due to the presence of a flatter peak and very heavy tail.

Qualitative variables

The time has now come to analyse the qualitative variables in the training set. Let us begin by saying that the variables *UDI* and *Product.ID* have unique values for each observation. It therefore makes little sense to perform a univariate analysis of these.

2.1.6 Type

Below we can see the distribution of the qualitative variable *Type*. We note that most of the observations in the training set are of type 'L'. The remainder (about 40%) is three quarters from 'M' type observations and only one quarter from 'H' type observations.

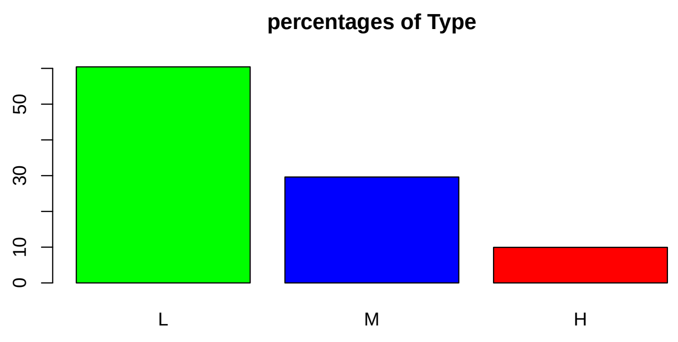
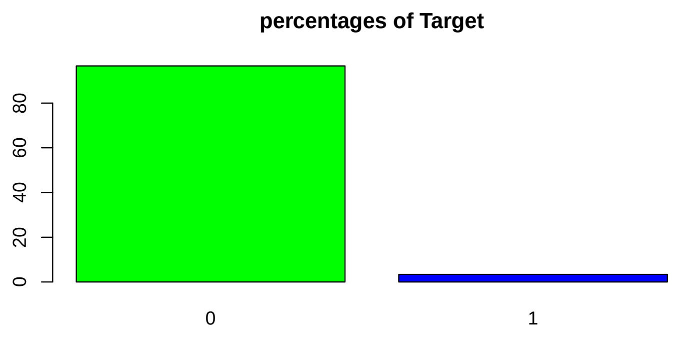


Figure 6: barplot of Type's distribution

2.1.7 Target

Let us now see how our response variable is distributed in the training set.



From the bar chat, we see that, perceptually, observations with a Target value of 0 are far greater than those with a Target value of 1 (approximately by a factor of 1:20). This means that our dataset is highly unbalanced.

2.2 Multivariate Analysis

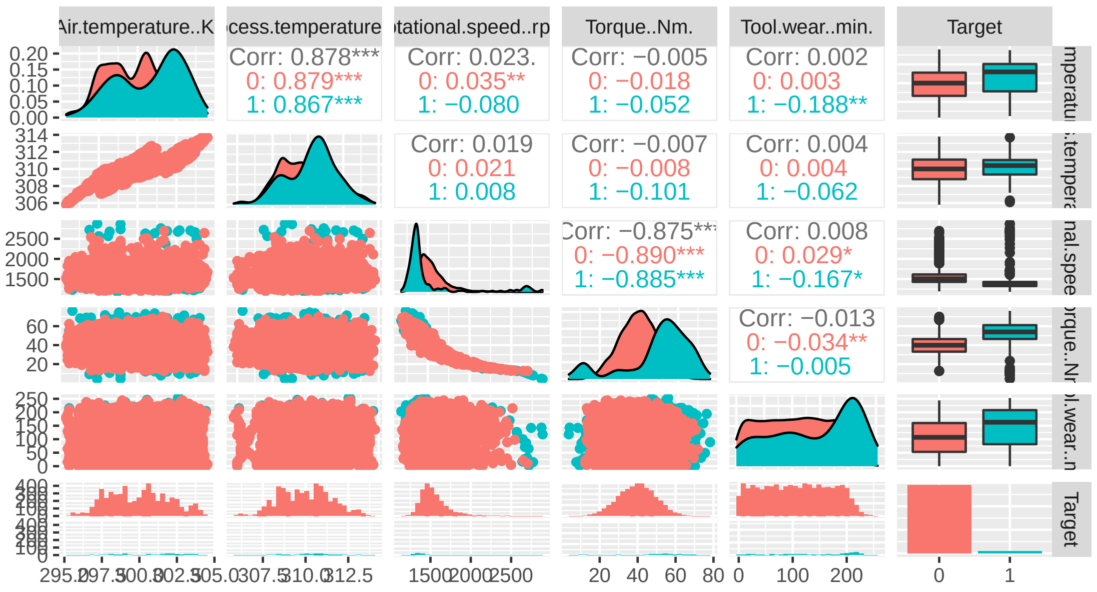
Multivariate EDA techniques generally show the relationship between two or more variables. Let us first take a look at the correlation matrix between the numerical variables.

	Air.temperature	Process.temperature	Rotational.speed	Torque	Tool.wear
Air. temperature	1.00	0.88	0.02	-0.01	0.00
Process. temperature	0.88	1.00	0.02	-0.01	0.00
Rotational. speed	0.02	0.02	1.00	-0.88	0.01
Torque	-0.01	-0.01	-0.88	1.00	-0.01
Tool. wear	0.00	0.00	0.01	-0.01	1.00

Table 7: correlation matrix of quantitative variables

Looking at the correlation matrix, we can see how the variables are highly correlated in pairs: we can see that the variables **Air.temperature** and **Process.temperature** have a high correlation coefficient with each other but not with the other variables, which is null; similarly, the variables **Rotational.speed** and **Torque** also have a high correlation coefficient with each other and almost null with the other variables. As for the variable **Tool.wear**, we note that it has practically zero correlation with all variables.

In the scatterplot matrix below, it is described the conditional distribution between the quantitative variables and the response Target



By looking at the diagonal, we can see the probability density functions of the predictors with respect to our target variable. For instance, we can notice how the more the Air temperature increases, the more machine failure (similar situation also considering the Process temperature). Considering the Rotational speed variable, much more Rotational makes more failure.

Looking at the boxplot, we can notice that Rotational speed presents lots of outliers (especially in the case for which target is equal to 1, meaning that failure occur among machines which generally have less Rotational speed, there are some exceptions). There are also outliers for Process temperature, Torque Nm , when target is equal to 1 and Process temperature and Torque Nm when target is equal to 0.

3 Modelling phase

It is now time to build our classification model. To do this we will try three approaches: the *logistic regression*, the *random forest* and the *neural network*.

Although the ultimate purpose of the three is the same, the three models are profoundly different. Logistic regression is an extension of linear regression to the case where the response variable is a class. Random forest, on the other hand, is a ensemble technique in which a classification tree is constructed for each sample, and then, based on the predictor values, the assigned class will be the one "highest rated" by the individual trees. Finally, neural network is a technique that attempts to emulate the synaptic connections between neurons in the human brain. In fact, just as occurs in the vital organ, information (weights) pass from one layer of neurons to the next. These three approaches will be seen and explained in more detail in this section.

The purpose of this section is to build the model from the training set, making all due considerations in the choice of parameters, and to evaluate its accuracy and effectiveness using the validation set. The model found to have the highest accuracy will be chosen to rank the test set data.

3.1 Logistic regression

Logistic regression models the probabilities for classification problems with two possible outcomes. It's an extension of the linear regression model for classification problems. In order to obtain a probability, the model uses a sigmoidal function that provides precisely an output between 0 and 1 that represents the probability of an event occurring taking as input the predictors.

First of all, we will perform logistic regression using all predictors, obtaining the following results:

	Estimates	$Pr(>\|z\|)$
(Intercept)	-3.052e+01	0.1092
Type.L	-6.575e-01	0.0125 *
Type.Q	-1.279e-01	0.5264
Air.temperature..K.	7.587e-01	2.32e-15 ***
Process.temperature..K.	-7.549e-01	3.61e-09 ***
Rotational.speed..rpm.	1.209e-02	< 2e-16 ***
Torque..Nm.	2.906e-01	< 2e-16 ***
Tool.wear..min.	1.357e-02	< 2e-16 ***

Table 8: coefficient's estimates of logistic regression w/ all predictors

The **AIC** obtained from this model with all variables as predictors is equal 1143.6, all predictors are statistically significant, even the ordered categorical variable("Type") turns out to be so since it possesses the linear term (*Type.L*) with a p-value less than 0.05.

To have a comparison we will perform stepwise logistic regression "*stepAIC*" in order to obtain the variables that provide the lowest value of AIC (provides a means for model selection). From stepwise logistic regression ,we obtained the same model as in the previous case so with the same AIC index and with the same significant variables then we are inclined to say that all variables should be retained.

In addition, we can see that there is a substantial difference between the null and residual deviance respectively 1773.8 and 1127.6, which it means that the model with the variables used fits the data better than the model with only the intercept.

At this point we can point out that the variables "Air.temperature..K," the quadratic type term and "Process.temperature..K." have a negative estimate this indicates that an increase in these variables is associated with a decrease in the probability of failure. The other variables, on the other hand, have a positive estimate and so it means the opposite I.e. an increase in those variables is associated with an increase in the probability of failures.

Now to evaluate the model we will create the confusion matrix with the value of the train dataset using 0.5 as the decision threshold.

	true values		
predicted	0	1	sum
0	5776	158	5934
1	20	45	65
sum	5796	203	5999

Table 9: confusion matrix logistic w/ threshold 0.5 on training

After that we have created the confusion matrix we can compute a series of metrics to evaluate model like:

- Accuracy : 97.04
- Error rate: 2.96
- Sensitivity: 22.16
- Specificity: 99,65

Accuracy can be defined as the percentage of correct predictions made by our classification model. Accuracy is a good metric to use when the classes are balanced, i.e. proportion of instances of all classes are somewhat similar. However, it is to be noted that accuracy is not a reliable metric for datasets having class imbalance, i.e. the total number of instances of a class of data is far less than the total number of instances for another class of data. For this reason, although the model has good accuracy, we have a decidedly low sensitivity.

To increase the sensitivity and consequently reduce the specificity we can compute the ROC curve that is used to assess the accuracy of a continuous measurement for predicting a binary outcome. Each point of the chart has been realized varying the prediction threshold from 0 to 1.

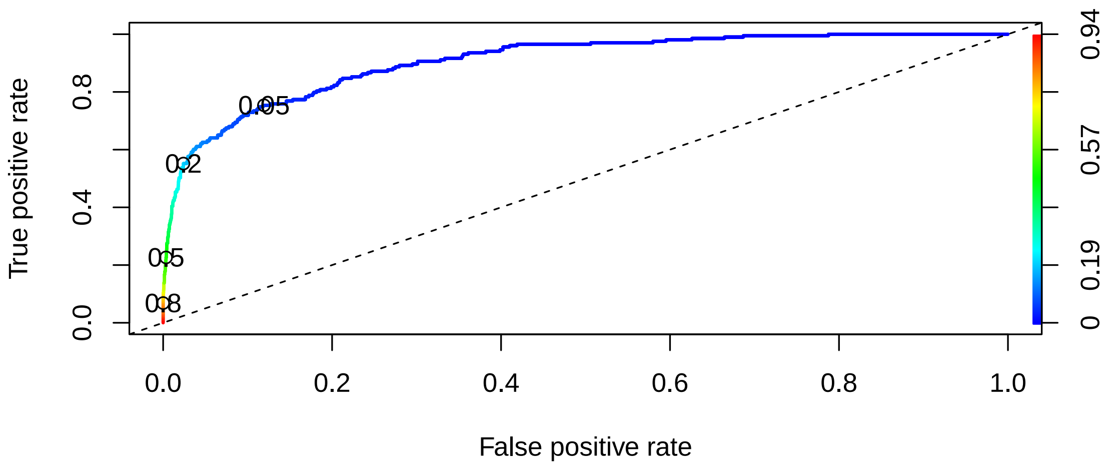


Figure 7: ROC curve

Thanks to the ROC curve we can observe that by decreasing the threshold for example from 0.5 to 0.2 we will get a sensitivity of about 60%, losing only about 5% specificity, moreover to evaluate the best threshold we used a function from the pRoc library that returns a value of 0.05, testing that threshold in the train dataset we obtained the following values:

- Accuracy : 87.64%
- Error rate: 12.36%
- Sensitivity: 75.36%
- Specificity: 88%

As we can see the metrics using 0.05 as decision threshold improved in terms of sensitivity but worsened all others, so after several attempts the best compromise between accuracy and sensitivity was obtained using 0.1 as decision threshold. The result of this configuration is the following confusion matrix:

	true values		
predicted	0	1	sum
0	5405	73	5523
1	346	130	476
sum	5796	203	5999

Table 10: confusion matrix logistic w/ threshold 0.1 on training

From this, the following metrics can be calculated:

- Accuracy : 93%
- Error rate: 7%
- Sensitivity: 64%
- Specificity: 94%

Moreover the overall performance of a classifier, summarized over all possible thresholds, is given by the Area Under the ROC Curve (AUC). An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier. In our case the AUC is about 0.9 that at first glance it would seem a good classifier but anyway we should compare it with other models.

Validation of the model

In this phase we will test the model built mediate the logistic regression approach using 0.1 as decision threshold on validation set.

The resulting confusion matrix is presented below.

	true values		
predicted	0	1	sum
0	1835	28	1863
1	98	41	139
sum	1933	69	2002

Table 11: confusion matrix logistic w/ threshold 0.1 on validation

In this case it is possible to calculate the following metrics:

- Accuracy : 93,70%
- Error rate: 6.30%
- Sensitivity: 59.40%
- Specificity: 94,90%

We note how, in the validation set, it results in a slightly improved accuracy value (and thus also error rate) compared to that of the training set. All in all, the two values deviated by a very small amount, so that we can say that the logistic regression gave the same result in both datasets.

3.2 Random forest

The second approach we will use to construct the classifier is the "*Random Forest.*" This is an ensemble technique, which consists of applying a basic technique several times to small samples generated from the initial dataset and then averaging the results; this procedure is intended to reduce the variance.

The random forest then consists of applying many classification trees to the various de-correlated samples generated from the dataset. The special feature of the random forest is that, unlike other ensemble techniques such as *Bagging* (or *Bootstrap Aggregation*), it uses a smaller number of predictors for each constructed tree. In the case of classification that number is equal to $\lfloor\sqrt{p}\rfloor$ which therefore in our case it is equal to 3.

The random forest in constructing each tree uses two-thirds of the total observations and then evaluates it in the remaining third. It practically builds a validation set automatically for each individual tree. For this reason, it was decided to combine the observations of the training set and the validation set so as not to deprive the model of useful observations for a more accurate construction.

Once we constructed the random forest with the above elements, it returned the following counfusion matrix:

	predicted		
true values	0	1	class.error
0	7706	23	0.002975805
1	82	190	0.301470588

Table 12: confusion matrix random forest.

With an **OOB estimate of error rate** equals to 1.31% and an **accuracy** of 98.69%. Although the prediction has a very high accuracy, it is predicted to have a classification error for positives of 30%, i.e. the model can only correctly reveal 70% of failed machines. This result is obviously not acceptable, as the prediction of failures is more important than the prediction of non-failures. This error is due to the unbalanced nature of the dataset, in fact the model trains a lot to recognise non-faulty machinery but at the same time very little to recognise faulted machinery.
As already mentioned in the logistic regression approach, accuracy alone is not a reliable metric for assessing the goodness of the model in an unbalanced dataset. It is therefore necessary to combine accuracy with sensitivity (i.e. the complement of the classification error of positives).

To solve this problem, it was decided to 'balance' the dataset. In particular, the '*stratified sampling*' was set as the sampling method for constructing each tree by taking from the new trainiing set (given by the union of the original training and validation) a number of observations with a target value of '0' ten times greater than the observations with a Target value of '1', reducing the proportion from 1:20 to 1:10.

Below is the confusion matrix with the new configuration just described.

	predicted		
true values	0	1	class.error
0	677	52	0.006727908
1	63	209	0.231617647

Table 13: confusion matrix random forest w/ stratified sampling.

As we can now see, a good compromise between accuracy and sensitivity has been achieved. In fact, compared with the previous set-up, the accuracy value went from 98.69% to 98.56%, while the specificity increased from $(100 - 30)\% = 70\%$ to $(100 - 23)\% = 77\%$. In other words, we have 'sacrificed' a little over a tenth of accuracy to obtain 7 percentage units of specificity.

Other tests were also carried out by varying the ratio of 1 : 10 of observations with Target '0' and with Target '1', but most of the time we obtained a very large reduction in accuracy, thus causing an equally large reduction in the accuracy value (i.e. too many non-failing machines are predicted as failures). We therefore came to the conclusion that the 1 : 10 ratio is the one that gives the best compromise between the two metrics.

The following graph shows the importance that each variable had in the construction of the random forest, which we recall was intended to predict the categorical variable 'Target'.

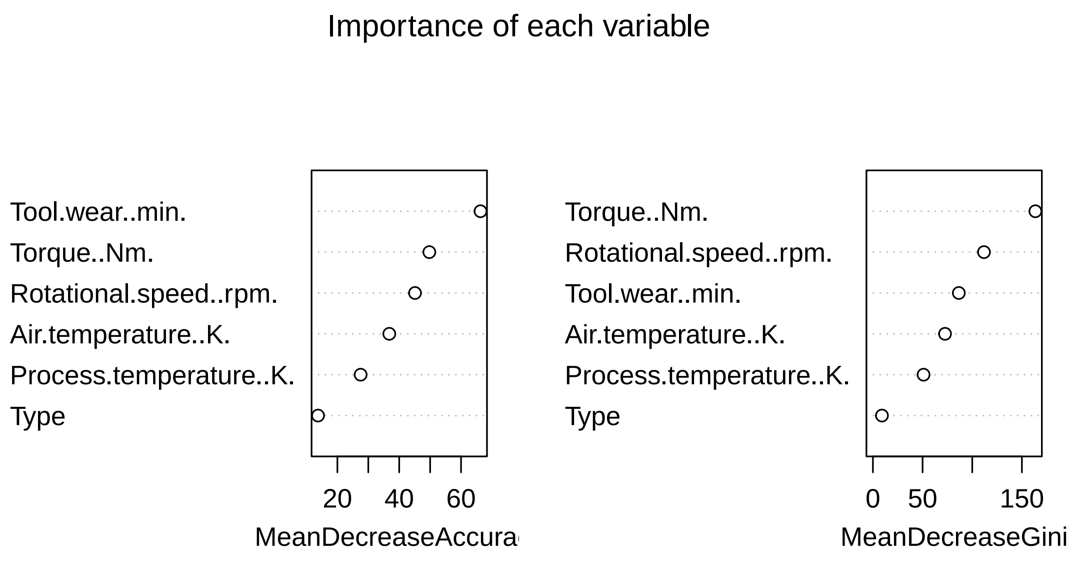


Figure 8: Variable importance in stratified RF

The *Mean Decrease Accuracy* refers to the mean decrease of accuracy predictions on the OOB (out of bag) samples, when a given variable is excluded by the model; the *Mean Decrease in Gini coefficient* is a measure of how each variable contributes to the homogeneity of the nodes and leaves in the resulting random forest.

Looking at the left graph, it is possible to see that the variables 'Tool.wear'is the most influential in the model accuracy value; in fact, its value of *MeanDecreaseAccuracy* differs from that of the other variables by a not insignificant value. Looking instead at the graph on the right, it can be seen that this time the variable 'Torque' is the one that most influences the homogeneity. In fact, its MeanDecreaseGini value deviates from that of the other variables by a not insignificant amount.

3.3 Feed-foward Neural Network

The last approach that will be used to build our classification model is the *feed-foward Neural Network*.
Neural networks are computational systems with interconnected nodes that function much like neurons in the human brain. Using algorithms, they can recognize hidden patterns and correlations in unstructured data, group and classify them, and, over time, continuously learn and improve. In feed-forward neural networks each perceptron in one layer is connected to each perceptron in the next layer, and information advances in a single direction, without feedback loops.

To build our feed-foward neural network we made use of the *keras* library. Several attempts were made, varying the number of hidden layers, the number of neurons in each layer and the learning rate value, eventually a network was constructed with two hidden layers of 10 and 7 neurons respectively with a learning rate value of 0.001. This parameter composition was the one that resulted in the best accuracy and error rate values.

It was decided to fit the newly constructed model to both the training test and the validation set by setting the learning process a number of *epochs* equal to 50 and a value of *batch size* equal to 32. Both are integer values and both are hyper-parameters for the learning algorithm, i.e. parameters for the learning process, not internal model parameters found by the learning process. The number of epochs defines the number of times the learning algorithm will work on the entire training data set. The batch size defines the number of samples to be analysed before updating the internal model parameters.

The following graph shows the trend of the accuracy and the value of the loss function as the epochs increase.

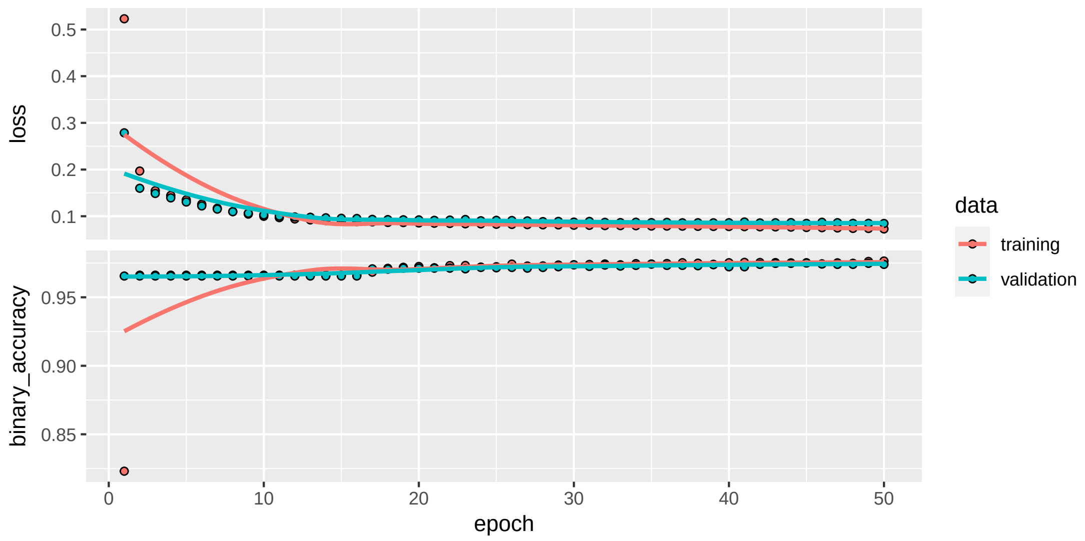


Figure 9: trend of accuracy and loss curves over epochs

The values of accuracy and loss results after applying the model to the two datasets are shown in the following table.

	loss	accuracy
training	0.07166519	0.9756626
validation	0.0845117	0.9725275

Our model was thus able to correctly predict the 97.25% of the target variables in the validation set. As mentioned in the previous approach, only accuracy is not enough to evaluate the goodness of the model in an unbalanced dataset. For this reason, the confusion matrix calculated using the observations of the validation set is presented below, from which we could subsequently derive the sensitivity values.

	true values		
predicted	0	1	sum
0	1929	51	1980
1	4	18	22
sum	1933	69	2002

Table 14: confusion matrix neural network on validation set

Looking at the confusion matrix, we can calculate that the sensitivity value is equal to 26.08%. A very low value when compared to that obtained by the random forest.

4 Results

To choose the best model we have to compare the metrics obtained from the 3 models applied to the validation set, recalling the 'Table 11', 'Table 13' and 'Table 14', we have following table:

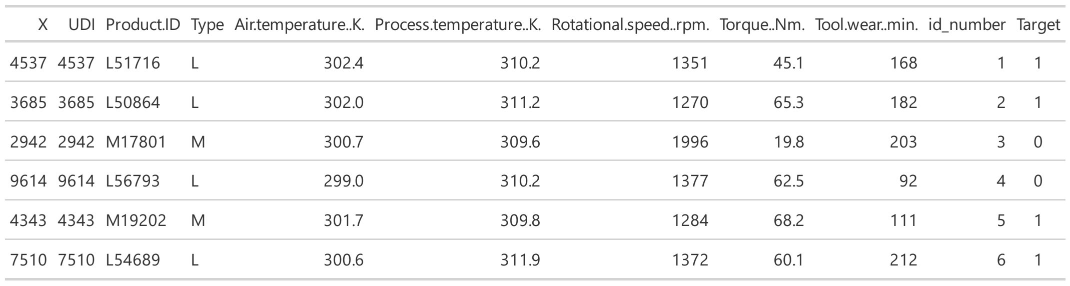
	accuracy	sensitivity	specificity
Logistic regression	93.70%	59.40%	94.90%
Random forest	98.56%	76.83%	99.32%
Neural network	97.25%	26.08%	99.80%

Table 15: summary approches

Since we have an unbalanced dataset, we have to look for the best trade-off between accuracy and sensitivity. That's why predicting failures has a higher importance than predicting non-failures without giving up the overall accuracy of the model. Therefore it is possible to observe that the random forest is the best model that is able to predict failure for the machine.

Applying the model to the test set

Here it is shown a preview of the results on the test set.



X	UDI	Product.ID	Type	Air.temperature..K.	Process.temperature..K.	Rotational.speed..rpm.	Torque..Nm.	Tool.wear..min.	id_number	Target
4537	4537	L51716	L	302.4	310.2	1351	45.1	168	1	1
3685	3685	L50864	L	302.0	311.2	1270	65.3	182	2	1
2942	2942	M17801	M	300.7	309.6	1996	19.8	203	3	0
9614	9614	L56793	L	299.0	310.2	1377	62.5	92	4	0
4343	4343	M19202	M	301.7	309.8	1284	68.2	111	5	1
7510	7510	L54689	L	300.6	311.9	1372	60.1	212	6	1

5 Conclusion

At the beginning of the report, we asked ourselves questions that guided us in the construction of the model. In this section, we will answer these questions.

To answer the first question, we have to take a look at Table 14 and, as already mentioned in section 4 of the report, the best model, which can best predict the Target variable, is the one with the best trade-off between accuracy and sensitivity. Such a model is the one built using the random forest approach with the sampling method setting as 'stratified sampling'.

With regard to the second question, to see which variables have a greater importance in the construction of the model, we have to look at Figure 8, where we find that the variables that most influence the prediction of the Target variable are *Torque* and *Tool.wear*.

The last question asks us how accurately failures can be predicted. Contrary to what one might think, the answer is not to be found in the accuracy of the random forest, but in the sensitivity, i.e. that index which shows us the percentage of correctly predicted failures. The answer to the question is therefore: 'We are able to predict a failure 76.83% of the time'.

6 Appendix

The following pages will present all the R-codes used in Exploratory Analysis and in the creation of the classification model.

```
### SETTING THE DATASETS ###

setwd("/home/orazio/Desktop/Link to Final Report/predictive maintenance")
train <- read.csv("predictive_maintenance_train.csv", header = TRUE)
valid <- read.csv("predictive_maintenance.csv", header = TRUE)

str(train)
##
## 'data.frame': 5999 obs. of  10 variables:
## $ X                      : int  3530 4502 1145 2016 443 4643 4410 4122
4571 4557 ...
## $ UDI                    : int  3530 4502 1145 2016 443 4643 4410 4122
4571 4557 ...
## $ Product.ID             : chr  "L50709" "L51681" "L48324" "L49195" ...
## $ Type                   : chr  "L" "L" "L" "L" ...
## $ Air.temperature..K.    : num  302 302 297 298 297 ...
## $ Process.temperature..K.: num  311 310 308 308 308 ...
## $ Rotational.speed..rpm. : int  1567 1307 1296 1301 1399 1238 1358 1364
1312 1349 ...
## $ Torque..Nm.            : num  39 54 69.1 66.3 61.5 54.6 54.6 47.8 52.2
51.2 ...
## $ Tool.wear..min.        : int  214 86 153 42 61 226 61 213 40 6 ...
## $ Target                 : int  1 1 1 1 1 1 1 1 1 1 ...

# As we can see the variable "Type" appear as character variable, but we
need it as Factor variable.
# Therefore, with the following command we will transform it in ordered
factor. In addition, the variable 'Target' appears as
# a integer, so a transformation in factor it necessary.

train$Type <- factor(train$Type, ordered = TRUE, levels = c("L","M","H"))
train$Target <- as.factor(train$Target)

str(train)
## 'data.frame': 5999 obs. of  10 variables:
## $ X                      : int  3530 4502 1145 2016 443 4643 4410 4122
4571 4557 ...
## $ UDI                    : int  3530 4502 1145 2016 443 4643 4410 4122
4571 4557 ...
## $ Product.ID             : chr  "L50709" "L51681" "L48324" "L49195" ...
## $ Type                   : Ord.factor w/ 3 levels "L"<"M"<"H": 1 1 1 1 1
1 2 2 1 2 ...
## $ Air.temperature..K.    : num  302 302 297 298 297 ...
## $ Process.temperature..K.: num  311 310 308 308 308 ...
## $ Rotational.speed..rpm. : int  1567 1307 1296 1301 1399 1238 1358 1364
1312 1349 ...
## $ Torque..Nm.            : num  39 54 69.1 66.3 61.5 54.6 54.6 47.8 52.2
51.2 ...
## $ Tool.wear..min.        : int  214 86 153 42 61 226 61 213 40 6 ...
## $ Target                 : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2
2 ...

# Now, in order to perform at best the various classification approaches, we
will remove the identifier variables: X,
# UDI, Product.ID.
```

```
train$X <- NULL
train$UDI <- NULL
train$Product.ID <- NULL

str(train)
## 'data.frame': 5999 obs. of  7 variables:
## $ Type                   : Ord.factor w/ 3 levels "L"<"M"<"H": 1 1 1 1 1
1 2 2 1 2 ...
## $ Air.temperature..K.    : num  302 302 297 298 297 ...
## $ Process.temperature..K.: num  311 310 308 308 308 ...
## $ Rotational.speed..rpm. : int  1567 1307 1296 1301 1399 1238 1358 1364
1312 1349 ...
## $ Torque..Nm.            : num  39 54 69.1 66.3 61.5 54.6 54.6 47.8 52.2
51.2 ...
## $ Tool.wear..min.        : int  214 86 153 42 61 226 61 213 40 6 ...
## $ Target                 : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2
2 ...

# In order to perform our analysis approaches, we will generate two
different datasets from the train one: one excluding
# the Factor variable 'Type' and another one including it. Then we will
perform the analysis approaches using both them
# in order to evaluate hoe the accuracy is influenced by this variable.

# We apply the same changes to validation set and test set.

valid$Type <- factor(valid$Type, ordered = TRUE, levels = c("L","M","H"))
valid$Target <- as.factor(valid$Target)
valid$X <- NULL
valid$UDI <- NULL
valid$Product.ID <- NULL

str(valid)
## 'data.frame': 2002 obs. of  7 variables:
## $ Type                   : Ord.factor w/ 3 levels "L"<"M"<"H": 1 1 1 3 1
1 1 2 2 1 ...
## $ Air.temperature..K.    : num  298 297 296 297 299 ...
## $ Process.temperature..K.: num  308 308 306 308 310 ...
## $ Rotational.speed..rpm. : int  1348 1289 2270 1549 1371 1365 2886 1867
1542 1329 ...
## $ Torque..Nm.            : num  58.8 62 14.6 35.8 53.8 52.9 3.8 23.4 37.5
53.6 ...
## $ Tool.wear..min.        : int  202 199 149 206 228 218 57 225 203 207
...
## $ Target                 : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2
2 ...



### UNIVARIATE ANALYSIS ###

library("e1071")
library(GGally)
## Registered S3 method overwritten by 'GGally':
## method from
## +.gg   ggplot2
```

```
# Check if there are some missing values
sum(is.na(train))
## [1] 0

# Air.temperature..K.

hist(train$Air.temperature..K., freq=FALSE, xlab= "Air temperature", col =
"lightblue", main = "Histogram of Air.temperature..K.")
summary(train$Air.temperature..K.)
## Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 295.4   298.3   300.1   300.0   301.5   304.5

skewness(train$Air.temperature..K.)
## [1] 0.1219379

kurtosis(train$Air.temperature..K.)
## [1] -0.8442163


# Process.temperature..K.

hist(train$Process.temperature..K., freq=FALSE, xlab= "Process temperature
K", col = "#FFC0CB", main = "Histogram of Process.temperature..K.")

summary(train$Process.temperature..K.)
##  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 305.8   308.8   310.1   310.0   311.1   313.8

skewness(train$Process.temperature..K.)
## [1] 0.02548641

kurtosis(train$Process.temperature..K.)
## [1] -0.5098995


# Rotational.speed..rpm.

hist(train$Rotational.speed..rpm., freq=FALSE, xlab= "Rotational speed rpm",
col = "red", main = "Histogram of Rotational.speed..rpm.")

summary(train$Rotational.speed..rpm.)
## Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 1181    1423    1504    1539    1611    2874

skewness(train$Rotational.speed..rpm.)
## [1] 1.993557

kurtosis(train$Rotational.speed..rpm.)
## [1] 7.405396


# Torque..Nm.
```

```
hist(train$Torque..Nm., freq=FALSE, xlab= "Torque Nm ", col = "orange", main
= "Histogram of Torque..Nm.")

summary(train$Torque..Nm.)
##  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 4.20   33.20   40.10   40.01   46.80   76.20

skewness(train$Torque..Nm.)
## [1] -0.003551766

kurtosis(train$Torque..Nm.)
## [1] -0.03281282


# Tool.wear..min.

hist(train$Tool.wear..min., freq=FALSE, xlab= "Tool wear min", col =
"#FFD700", main = "Tool.wear..min.")

summary(train$Tool.wear..min.)
## Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.0    53.0   109.0   108.2   162.0   253.0

skewness(train$Tool.wear..min.)
## [1] 0.01914566

kurtosis(train$Tool.wear..min.)
## [1] -1.152117


# Type

table(train$Type)
##    L    M    H
## 3625 1777  597

round(table(train$Type)/length(train$Type)*100, digits=2)
##     L     M     H
## 60.43 29.62  9.95

barplot(table(train$Type)/length(train$Type)*100, col = c("green","blue",
"red"), main = "percentages of Type")


# Target

table(train$Target)
##    0    1
## 5796  203

round(table(train$Target)/length(train$Target)*100, digits=2)
##     0     1
## 96.62  3.38
```

```
barplot(table(train$Target)/length(train$Target)*100, col =
c("green","blue"), main = "percentages of Target")


### MULTIVARIATE ANALYSIS ###

train_quant_var <- train[, - 1]
Target<- as.factor(train$Target)
ggpairs(train_quant_var, aes(colour=Target))



### LOGISTIC REGRESSION ###

library(tidyverse)
library(MASS)
library(ROCR)
library(pROC)

# Logistic regression number 0 ALL Predictors

modelLr = glm(Target~., data=train, family="binomial")
summary(modelLr)
##
## Call:
## glm(formula = Target ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6598  -0.1850  -0.1007  -0.0543   3.6329
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -3.052e+01  1.905e+01  -1.602   0.1092
## Type.L                   -6.575e-01  2.634e-01  -2.496   0.0125 *
## Type.Q                   -1.279e-01  2.019e-01  -0.634   0.5264
## Air.temperature..K.       7.587e-01  9.576e-02   7.923 2.32e-15 ***
## Process.temperature..K. -7.549e-01  1.279e-01  -5.901 3.61e-09 ***
## Rotational.speed..rpm.   1.209e-02  6.831e-04  17.695  < 2e-16 ***
## Torque..Nm.              2.906e-01  1.497e-02  19.409  < 2e-16 ***
## Tool.wear..min.          1.357e-02  1.479e-03   9.176  < 2e-16 ***
##   ---
## Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1773.8  on 5998  degrees of freedom
## Residual deviance: 1127.6  on 5991  degrees of freedom
## AIC: 1143.6
##
## Number of Fisher Scoring iterations: 8


# Logistic regression number 1 STEPAIC
```

```
#STEP AIC
modelLr1 <- glm(Target ~., data = train, family = binomial) %>%
  stepAIC(trace = TRUE)
## Start:  AIC=1143.65
## Target ~ Type + Air.temperature..K. + Process.temperature..K. +
##    Rotational.speed..rpm. + Torque..Nm. + Tool.wear..min.
##
##                           Df Deviance    AIC
## <none>                         1127.7 1143.7
## - Type                     2   1136.5 1148.5
## - Process.temperature..K.  1   1165.5 1179.5
## - Air.temperature..K.      1   1198.4 1212.4
## - Tool.wear..min.          1   1225.4 1239.4
## - Rotational.speed..rpm.   1   1406.9 1420.9
## - Torque..Nm.              1   1628.3 1642.3


# Summarize the final selected model
summary(modelLr1)
##
## Call:
## glm(formula = Target ~ Type + Air.temperature..K. +
Process.temperature..K. +
## Rotational.speed..rpm. + Torque..Nm. + Tool.wear..min., family =
binomial,
## data = train)
##
## Deviance Residuals:
##   Min       1Q   Median       3Q      Max
## -1.6598  -0.1850  -0.1007  -0.0543   3.6329
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -3.052e+01  1.905e+01  -1.602   0.1092
## Type.L                  -6.575e-01  2.634e-01  -2.496   0.0125 *
## Type.Q                  -1.279e-01  2.019e-01  -0.634   0.5264
## Air.temperature..K.      7.587e-01  9.576e-02   7.923 2.32e-15 ***
## Process.temperature..K. -7.549e-01  1.279e-01  -5.901 3.61e-09 ***
## Rotational.speed..rpm.   1.209e-02  6.831e-04  17.695  < 2e-16 ***
## Torque..Nm.              2.906e-01  1.497e-02  19.409  < 2e-16 ***
## Tool.wear..min.          1.357e-02  1.479e-03   9.176  < 2e-16 ***
## ---
## Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1773.8  on 5998  degrees of freedom
## Residual deviance: 1127.6  on 5991  degrees of freedom
## AIC: 1143.6
##
## Number of Fisher Scoring iterations: 8
##

# Testing prediction train set with threshold 0.5
predTrainLr = predict(modelLr1, type="response")
```

```
addmargins(table(predTrainLr>0.5,train$Target))
##          0    1  Sum
## FALSE 5776  158 5934
## TRUE    20   45   65
## Sum   5796  203 5999

predictedtrain <-as.numeric(predTrainLr > 0.5)
error_Lr_0.5 <- mean(predictedtrain != train$Target)
error_Lr_0.5
## [1] 0.02967161

accuracy_Lr_0.5 <- 1 - error_Lr_0.5
accuracy_Lr_0.5
## [1] 0.9703284

# ROC curve
ROCPred <- ROCR::prediction(predTrainLr, train$Target)
ROCPerf <- performance(ROCPred, "tpr", "fpr")
plot(ROCPerf,colorize=TRUE,lwd=2)
plot(ROCPerf,colorize=TRUE,lwd=2, print.cutoffs.at=c(0.05,0.2,0.5,0.8))
abline(a=0,b=1, lty=2)

# Best threshold value
my_roc <- roc(train$Target, predTrainLr)
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases

coords(my_roc, "best", ret = "threshold")
##   threshold
## 1  0.050318

# AUC value
ROCauc <-performance(ROCPred, measure ="auc")
ROCauc@y.values[[1]]
## [1] 0.9059322

# testing prediction train set with threshold 0.1
addmargins(table(predTrainLr>0.1,train$Target))
##          0    1  Sum
## FALSE 5450   73 5523
## TRUE   346  130  476
## Sum   5796  203 5999

predictedtrain1 <-as.numeric(predTrainLr > 0.1)
error_Lr_0.1 <-  mean(predictedtrain1 != train$Target)
error_Lr_0.1
## [1] 0.06984497

accuracy_Lr_0.1 <- 1 - mean(predictedtrain1 != train$Target)
accuracy_Lr_0.1
## [1] 0.930155

# Testing prediction w/ thr 0.05 validation set
```

```
predTestLr = predict(modelLr1, newdata=valid, type="response")
addmargins(table(predTestLr>0.05,valid$Target))
##
##          0    1  Sum
## FALSE 1689   21 1710
## TRUE   244   48  292
## Sum   1933   69 2002

predictedtest <-as.numeric(predTestLr > 0.05)
error_Lr.v0.05 <- mean(predictedtest != valid$Target)
error_Lr.v0.05
## [1] 0.1323676
accuracy_Lr.v0.05 <- 1 - error_Lr.v
accuracy_Lr.v0.05
## [1] 0.8676324

# Testing prediction w/ thr 0.1 validation set
predTestLr = predict(modelLr1, newdata=valid, type="response")
addmargins(table(predTestLr>0.1,valid$Target))
##
## 0    1  Sum
## FALSE 1835   28 1863
## TRUE    98   41  139
## Sum   1933   69 2002

predictedtest0.1 <-as.numeric(predTestLr > 0.1)
error_Lr.v0.1 <- mean(predictedtest0.1 != valid$Target)
error_Lr.v0.1
## [1] 0.06293706
accuracy_Lr.v0.1 <- 1 - error_Lr.v0.1
accuracy_Lr.v0.1
## [1] 0.9370629



### RANDOM FOREST ###

library(randomForest)
##randomForest 4.7-1.1
##Type rfNews() to see new features/changes/bug fixes.

# The random forest is a ensemble method that use $sqrt(p)$ predictors,
therefore in our case we will set mtry = 3.

set.seed(13)
df <- rbind(train, valid)

rf <- randomForest(Target ~ ., data = df,
                   mtry = 3, importance = TRUE)
rf
##
## Call:
## randomForest(formula = Target ~ ., data = df, mtry = 3, importance =
TRUE)
## Type of random forest: classification
## Number of trees: 500
## No. of variables tried at each split: 3
##
##        OOB estimate of  error rate: 1.31%
## Confusion matrix:
##      0   1 class.error
## 0 7706  23 0.002975805
## 1   82 190 0.301470588

accuracy.rf <- (rf$confusion[1,1] + rf$confusion[2,2]) / (rf$confusion[1,1]
+ rf$confusion[1,2]
                                                           + rf$confusion[2,
1] + rf$confusion[2,2])
accuracy.rf
# [1] 0.9868766

# Now we can visualize the importance of each variable in the built random
forest.

importance(rf)
##                                0         1 MeanDecreaseAccuracy
MeanDecreaseGini
## Type                    8.717722 18.078235             17.12808
13.42436
## Air.temperature..K.     48.112550 66.950043             60.45186
80.76690
## Process.temperature..K. 41.597453  7.096397             44.67121
73.10198
## Rotational.speed..rpm.  45.103999 50.025417             57.03827
100.58031
## Torque..Nm.             53.084299 88.550556             69.59543
170.90059
## Tool.wear..min.         46.422980 67.924776             70.05821
85.80741

varImpPlot(rf, main = "Importance of each variable")


### WITH STRATIFIED SAMPLING ###
k <- length(df$Target[df$Target == 1])
stratified.rf = randomForest(Target ~ ., data = df, strata = df$Target,
sampsize = c(10*k,k) ,
                             mtry = 3, importance = TRUE)
stratified.rf
##
## Call:
##   randomForest(formula = Target ~ ., data = df, strata = df$Target,
sampsize = c(10 * k, k), mtry = 3, importance = TRUE)
## Type of random forest: classification
## Number of trees: 500
## No. of variables tried at each split: 3
##
## OOB estimate of  error rate: 1.44%
## Confusion matrix:
##      0   1 class.error
## 0 7677  52 0.006727908
## 1   63 209 0.231617647

accuracy.stratified.rf <- (stratified.rf$confusion[1,1] +
stratified.rf$confusion[2,2]) /
  (stratified.rf$confusion[1,1] + stratified.rf$confusion[1,2] +
stratified.rf$confusion[2,1]
   + stratified.rf$confusion[2,2])
accuracy.stratified.rf
## [1] 0.9856268

importance(stratified.rf)
##                                 0         1 MeanDecreaseAccuracy
MeanDecreaseGini
## Type                      7.755253 16.905939             13.74844
9.095684
## Air.temperature..K.     32.110556 66.266869             36.79287
72.592087
## Process.temperature..K. 26.301736  8.838071             27.52204
50.980035
## Rotational.speed..rpm.  38.417083 49.496883             45.11411
111.817435
## Torque..Nm.             40.594234 77.632196             49.74724
163.552538
## Tool.wear..min.         46.462691 72.283030             66.29542
86.490675

varImpPlot(stratified.rf, main = "Importance of each variable")




### NEURAL NETWORK ###

library(keras)
tensorflow::set_random_seed(13)
#Loaded Tensorflow version 2.9.1


scaled.train <- scale(model.matrix(Target ~. - 1, data = train))
true.train.targets <- to_categorical(train$Target)
scaled.valid <- scale(model.matrix(Target ~. - 1, data = valid))
true.valid.targets <- to_categorical(valid$Target)

modelnn <-  keras_model_sequential()
modelnn %>%
  layer_dense(units = 10, activation = 'relu', input_shape =
ncol(scaled.train)) %>%
  layer_dense(units = 7, activation = 'relu')  %>%
  layer_dense(units = 2, activation = "sigmoid")

summary(modelnn)
##Model: "sequential"
##____________________________________________________________________________________
##  Layer (type)                                       Output Shape
Param #
##====================================================================================
##  dense_2 (Dense)                                     (None, 10)
90
##  dense_1 (Dense)                                     (None, 7)
77
##  dense (Dense)                                       (None, 2)
16
##==========================================================================================
## Total params: 183
## Trainable params: 183
## Non-trainable params: 0
##__________________________________________________________________________________________

modelnn %>% compile(
  loss = loss_binary_crossentropy,
  optimizer = optimizer_rmsprop(learning_rate = 0.001),
  metrics = metric_binary_accuracy
)

system.time(
  history <- modelnn %>% fit(
    scaled.train,
    true.train.targets,
    epochs = 50,
    batch_size = 32,
    validation_data = list(scaled.valid, true.valid.targets)
  )
)


plot(history)

modelnn %>% evaluate(scaled.train, true.train.targets)
#188/188 [==============================] - 0s 702us/step - loss: 0.0717 -
binary_accuracy: 0.9762
#           loss binary_accuracy
#     0.07166519      0.97616267

modelnn %>% evaluate(scaled.valid, true.valid.targets)
#63/63 [==============================] - 0s 2ms/step - loss: 0.0845 -
binary_accuracy: 0.9740
#          loss binary_accuracy
#    0.0845117       0.9740260


# Confusion matrix on validation
pred<-as.matrix(modelnn %>% predict(scaled.valid) %>% `>`(0.5) %>%
k_cast("int32"))
## 63/63 [==============================] - 0s 643us/step

table(pred[,2], true.valid.targets[,2])
##
##      0    1
## 0 1929   51
## 1    4   18
```

```
### APPLAYING THE MODEL TO THE TEST DATASET ###

test <- read.csv("predictive_maintenance_test.csv", header = TRUE)

trained.test <- test
trained.test$X <- NULL
trained.test$UDI <- NULL
trained.test$Product.ID <- NULL
trained.test$id_number <- NULL
trained.test$Type <- factor(test$Type, ordered = TRUE, levels =
c("L","M","H"))

str(trained.test)
## 'data.frame': 1999 obs. of  6 variables:
## $ Type                    : Ord.factor w/ 3 levels "L"<"M"<"H": 1 1 2 1 2
1 2 1 1 1 ...
## $ Air.temperature..K.     : num  302 302 301 299 302 ...
## $ Process.temperature..K.: num  310 311 310 310 310 ...
## $ Rotational.speed..rpm. : int  1351 1270 1996 1377 1284 1372 1671 1326
1312 1316 ...
## $ Torque..Nm.             : num  45.1 65.3 19.8 62.5 68.2 60.1 30.5 58.5
65.3 61.2 ...
## $ Tool.wear..min.         : int  168 182 203 92 111 212 234 55 192 200 ...

pred.target = predict(stratified.rf, newdata=trained.test, type="response")
test$Target <- pred.target
head(test)
##
## X  UDI Product.ID Type Air.temperature..K. Process.temperature..K.
## 1 4537 4537     L51716    L               302.4                   310.2
## 2 3685 3685     L50864    L               302.0                   311.2
## 3 2942 2942     M17801    M               300.7                   309.6
## 4 9614 9614     L56793    L               299.0                   310.2
## 5 4343 4343     M19202    M               301.7                   309.8
## 6 7510 7510     L54689    L               300.6                   311.9
## Rotational.speed..rpm. Torque..Nm. Tool.wear..min. id_number Target
## 1                   1351        45.1             168         1      1
## 2                   1270        65.3             182         2      1
## 3                   1996        19.8             203         3      0
## 4                   1377        62.5              92         4      0
## 5                   1284        68.2             111         5      1
## 6                   1372        60.1             212         6      1

write.csv(test, file = "final data frame.csv")
```