Politecnico
di Torino

1859

Mathematics in Machine Learning

Professors: Francesco Vaccarino, Mauro Gasparini

Gabriele Rosi s291082

A.Y. 2021/2022

Summary

1 Introduction

Predictive maintenance is the one of the most used data-driven approach in companies in order to understand when maintenance should be performed. The aim of the current analysis is to develop a model to predict when a fail will occurs and the cause that generates it, given some data coming from an industrial machine.

The dataset used in the following work is called *AI4I 2020 Predictive Maintenance Dataset*[1] [1] and it contains synthetic data that reflects real predictive maintenance encountered in industry.

The code is available at the following link: `https://github.com/Gabrysse/MML-predictive-maintenance`.

2 Data exploration

2.1 Data overview and description

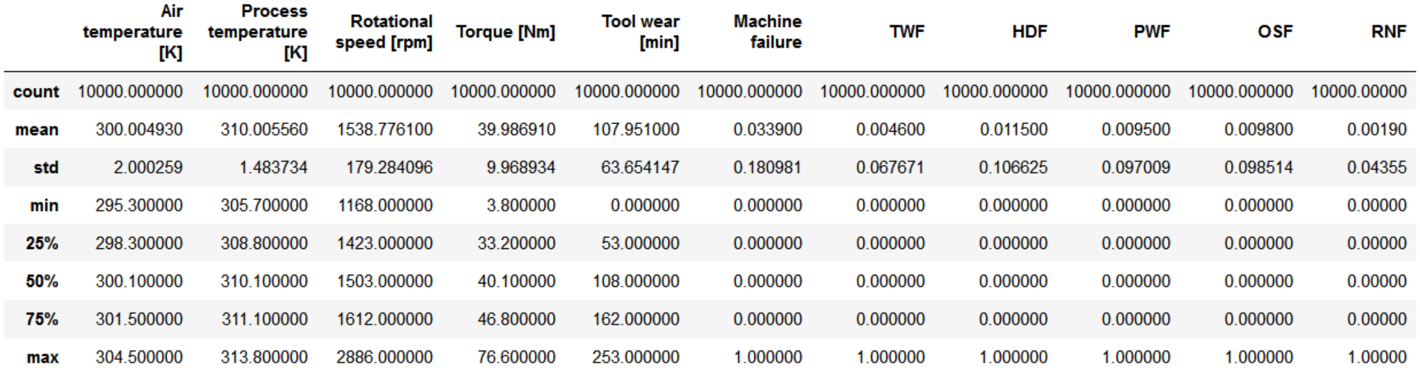
The dataset contains 10000 records and 14 columns that contains some machine parameter and the target variables. In particular:

- **UID** (*numerical*)
- **Product ID** (*string*)
- **Type** (*categorical*): machine type. It can assume: `L`, `M` or `H`.
- **Air temperature [K]** (*numerical*)
- **Process temperature [K]** (*numerical*)
- **Rotational speed [rpm]** (*numerical*)
- **Torque [Nm]** (*numerical*)
- **Tool wear [min]** (*numerical*)
- **Machine failure** (*binary*): target variable. 1 if the machine failed, 0 otherwise.

[1] https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset

- **TWF**, **HDF**, **PWF**, **OSF**, **RNF** (*binary*): target variables. If at least one of these failure modes is true, the process fails and the *Machine failure* label is set to 1.

In Figure 1 some statistics about features are summarized.



	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000
mean	300.004930	310.005560	1538.776100	39.986910	107.951000	0.033900	0.004600	0.011500	0.009500	0.009800	0.00190
std	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981	0.067671	0.106625	0.097009	0.098514	0.04355
min	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	298.300000	308.800000	1423.000000	33.200000	53.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
50%	300.100000	310.100000	1503.000000	40.100000	108.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
75%	301.500000	311.100000	1612.000000	46.800000	162.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
max	304.500000	313.800000	2886.000000	76.600000	253.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

Figure 1: Feature statistics

2.2 Target variables distribution

Since the machine failure is caused by one or more failure that in turn they depend on some event related to the other variables (e.g. torque that exceed a threshold value), we develop two classification model: one to predict if the machine will fail or not and one to predict the type of failure (if it presents). In Figure 2 and Figure 3 we can see the distributions of the two target variables. As the figures show, the data is skewed in favor of one class and this may cause problems in training. This can lead to incorrect results because machine learning algorithms tend to favor the class with the largest proportion of observations (majority class), but clearly we are interested in the correct classification also of the "rare" class (minority class).

2.3 Correlation

Another important analysis is the correlation among features. The correlations are calculated for each pair of features using the *Pearson's Correlation Coefficient*:

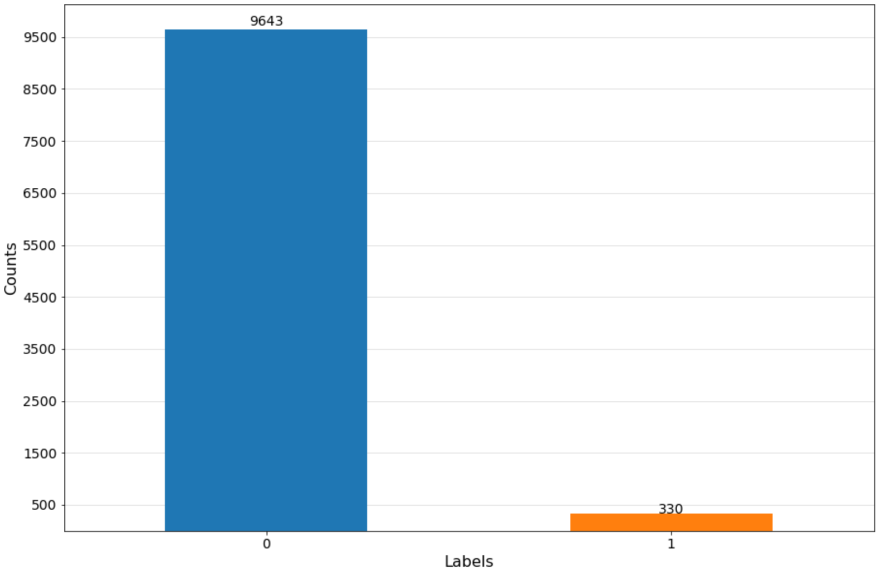


Figure 2: Machine failure distribution

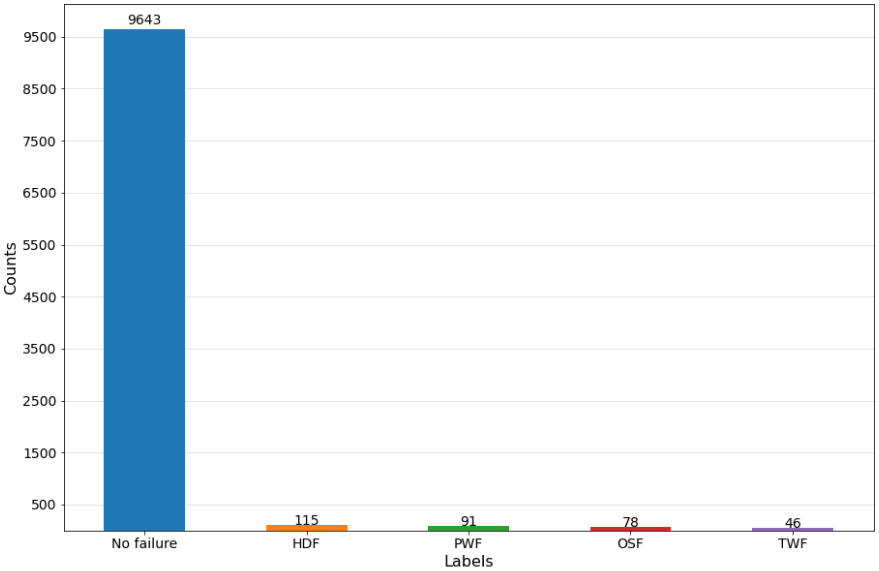


Figure 3: Failure type distribution

$$\rho_{(X,Y)} = \frac{\text{cov}(X,Y)}{\sigma_x \sigma_y}$$

This coefficient measures the strength of the relationship between two variables and their association with each other. Pearson's Correlation ranges between -1 and 1, where in both cases it means there is a strong linear correlation, negative and positive respectively, between the two features. In Figure 4 the correlation matrix is shown: high positive correlation is illustrated in dark red while high negative correlation is illustrated in dark blue. As said before, **Machine failure** is strongly correlated to **Failure Type** columns (TWF, HDF, PWF, OSF) since the failure types determine if a machine will fail or not. Instead, **Air temperature** and **Process temperature** are correlated because $\mathit{\Delta T} = AirTemperature - ProcessTemperature$ and **Torque** and **Rotational Speed** because $Power = Torque \cdot RotationalSpeed.$

In Figure 5 is it possible to see the pairwise relationships between variables. Also in this representation, the correlation **Air temperature - Process temperature** and **Torque - Rotational Speed** is pretty evident.

2.4 Outlier detection

Outliers detection is a fundamental step in order to avoid misclassification and errors. An easy way to visualize them are box plots. In Figure 6 and Figure 7 the two box plot relative to **Machine failure** and **Failure type** are reported. All the outliers are concentrated in **Rotational speed** and **Torque**. Since we do not have enough information about this data, we decided to keep them.

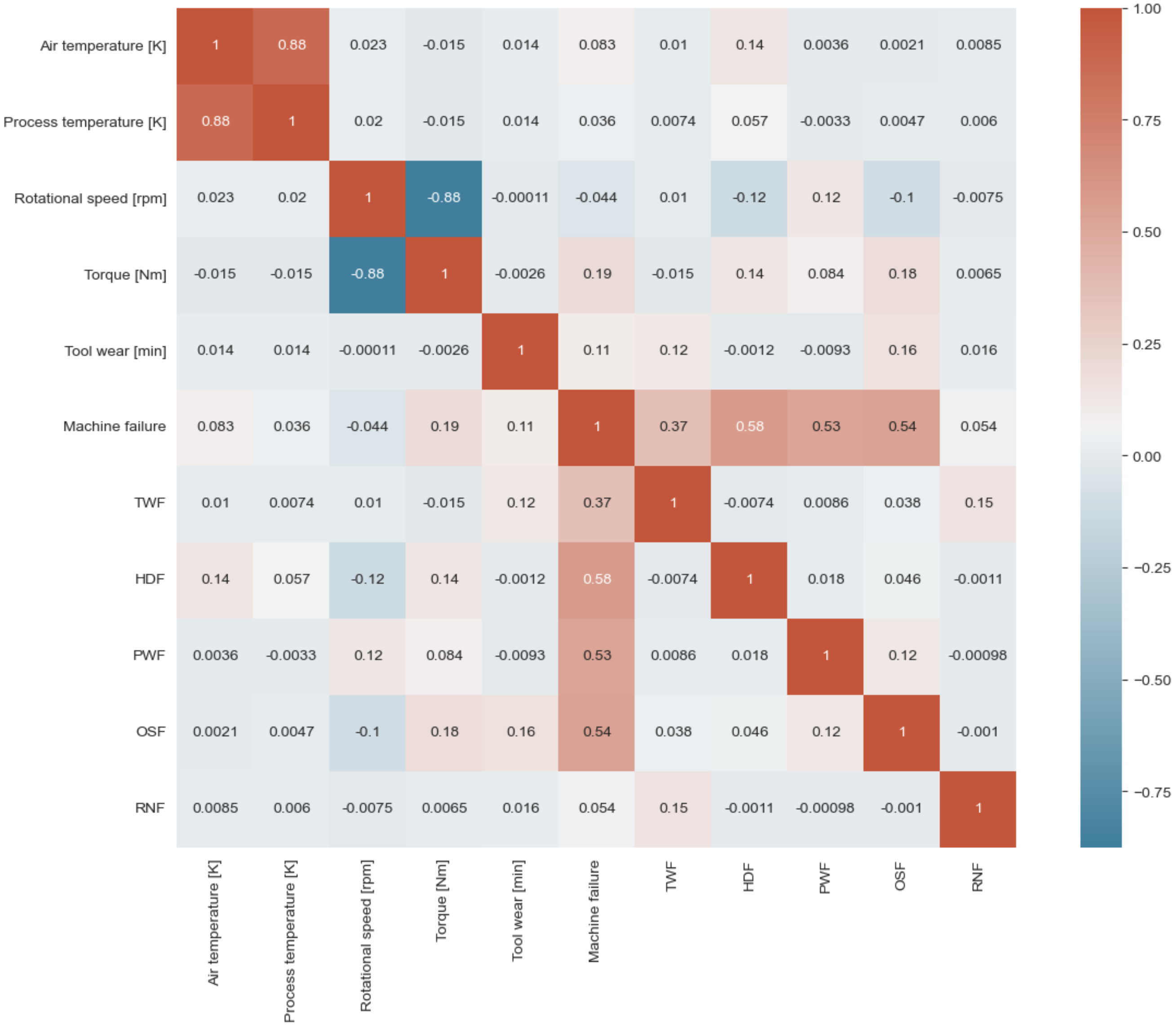


Figure 4: Correlation matrix

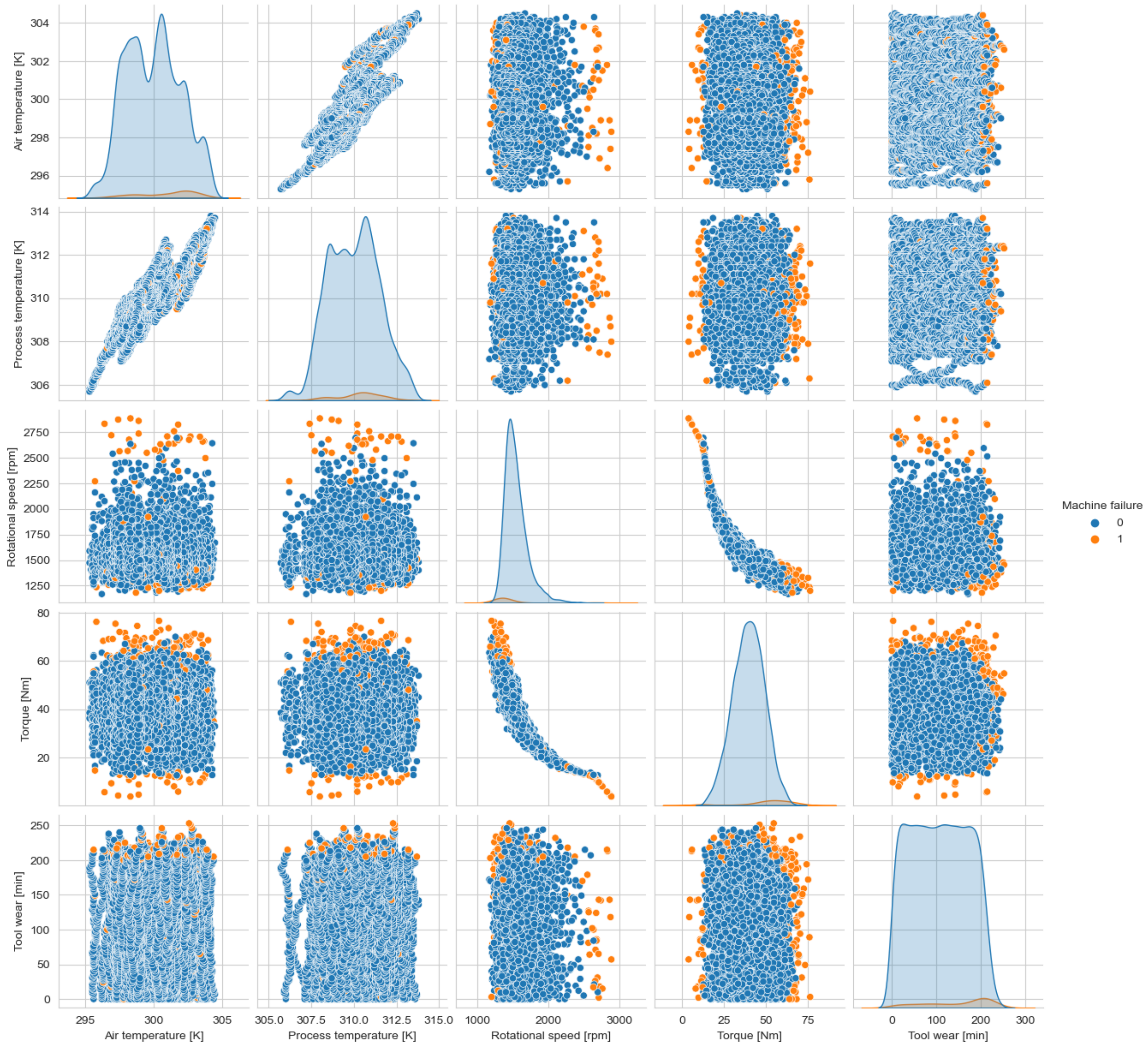


Figure 5: Pair plots

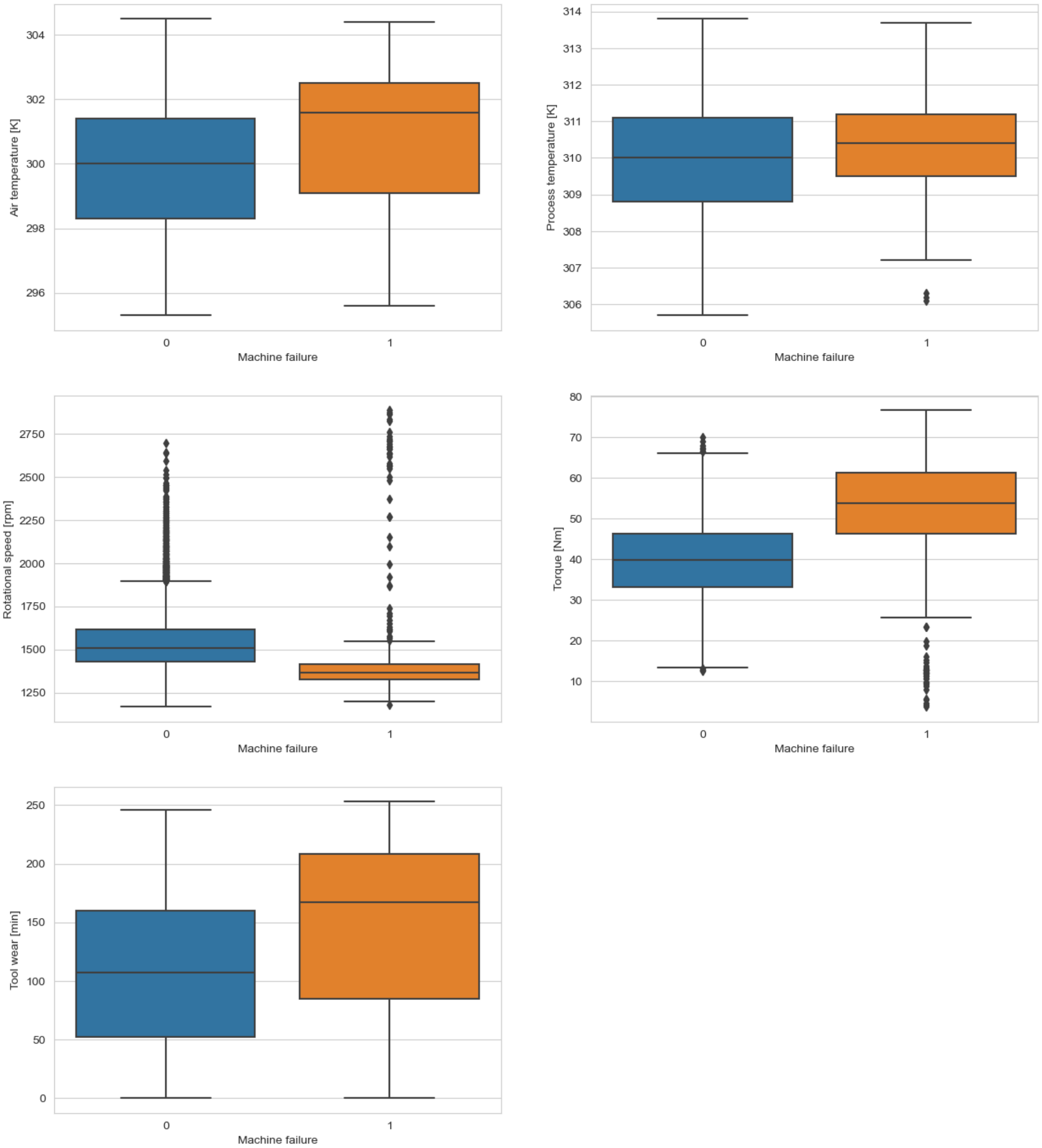


Figure 6: Machine failure box plots

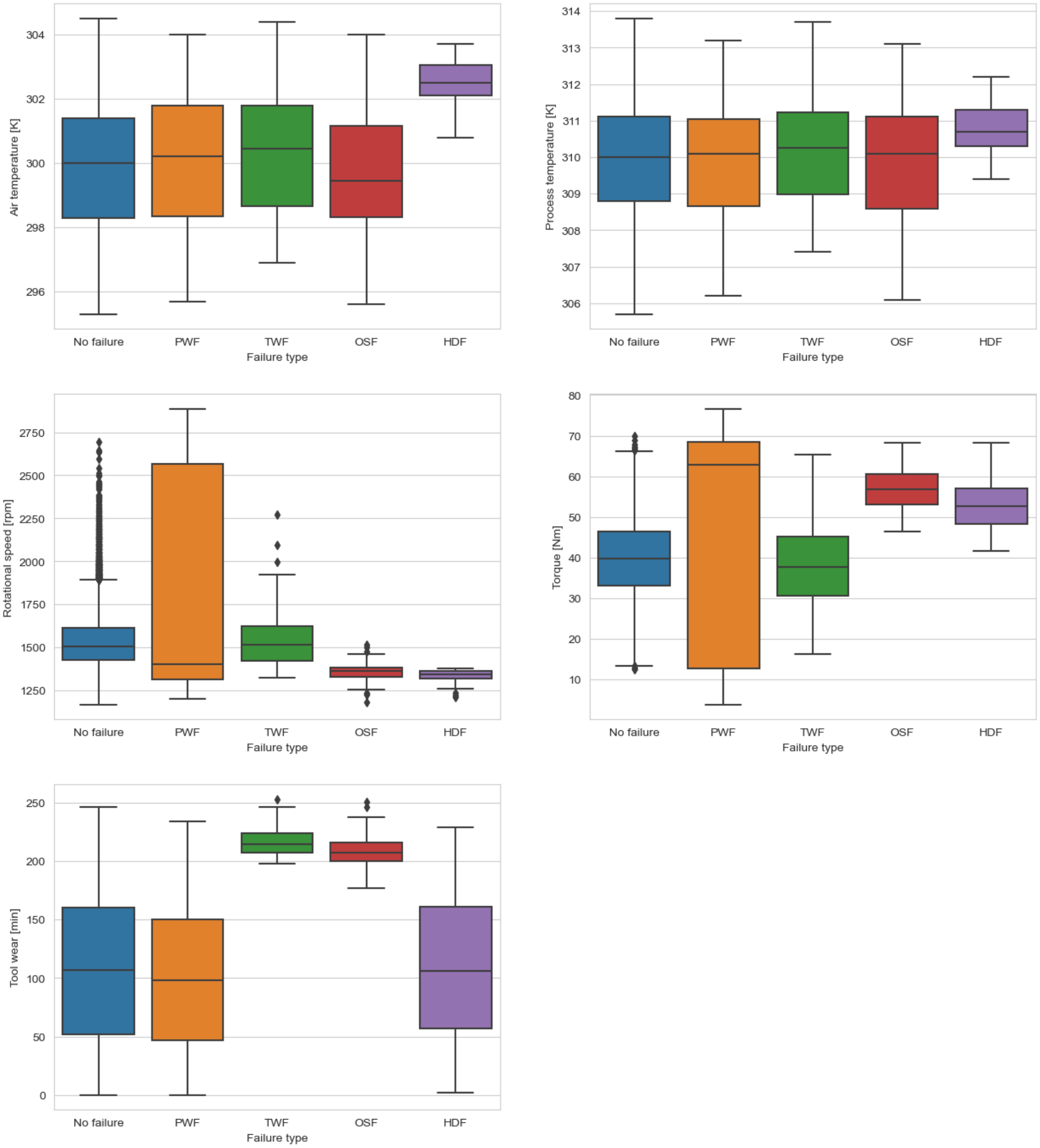


Figure 7: Failure type box plots

3 Data preprocessing

In this section, we are explaining the techniques used in order to prepare the dataset for the machine learning algorithms.

3.1 Feature selection

We remove **UID** and **Product ID** since they not provide any useful information.

3.2 Data cleaning

No missing values are present in this dataset. Nevertheless, we identify one situation that needs a particular attention in order to avoid errors. Specifically, the condition *"Machine failure = 1 if at least one of these failure modes is true"*, should be always verified. Instead, we have found:

- 9 records with $Machine\,failure = 1$ and all the $Failure\,type = 0$
- 18 records with $Machine\,failure = 0$ and at least one $Failure\,type = 1$

Since they are wrong records (according to the aforementioned condition), we simply remove them from our dataset.

3.3 Standardization

Many machine learning algorithms might behave badly when features are on different scales and not normally distributed. In Figure 1 is easy to see how different the scales and the distributions are between **Air Temperature**, **Process temperature**, **Rotational speed**, **Torque** and **Tool wear**. For this reason, a standardization step is required in order to obtain good results. One common standardization technique consist in removing the mean value (μ) of each feature, then scale it by dividing non-constant features by their standard deviation (σ):

$$Z = \frac{X - \mu}{\sigma}$$

In this way, now our data are normally distributed with $\mu = 0$ and $\sigma = 1$.

3.4 Categorical features encoding

Categorical features are a particular type of data in which we have text encoding an information. In particular, in our dataset we have the feature **Type** whose values are: *M*, *L* or *H*. In order to be understood by machine learning models, we have to transform such feature from categorical to continuous. Specifically, we chose to encode the feature **Type** using the one-hot encoding which transforms each categorical feature with *n_categories* possible values into *n_categories* binary features, with one of them 1, and all others 0. In our case, the result will be three columns: since we do not want to increase the correlation between features, we also decided to remove one of such columns.

3.5 Failure type decoding-encoding

In Section 2.1 we saw that the **Failure type** is divided in five different variables. In order to be a suitable target variable, we need to encode such information in one single variable. For this reason, first we reverse the one-hot encoding and then we encode each class with an ordinal number from 1 to 5. For the sake of simplicity, we assume that one variable at a time can be = 1 and therefore we assign the corresponding class based on the first variable = 1 found.

3.6 Dataset balancing

As we have already seen in Section 2.2, our dataset is very unbalanced. With an unbalanced dataset, machine learning models are more sensitive to detecting the majority class and less sensitive to the minority class. This leads to a biased classification output in many cases, resulting in always predicting the majority class.

During the years, different methods to deal with this situation have been proposed. It is possible to distinguish two types of methods: undersampling techniques and oversampling techniques. The former sample the majority class in order to have the same number as the minority class, while the latter create copies of the minority class in order to obtain the same number as the majority class.

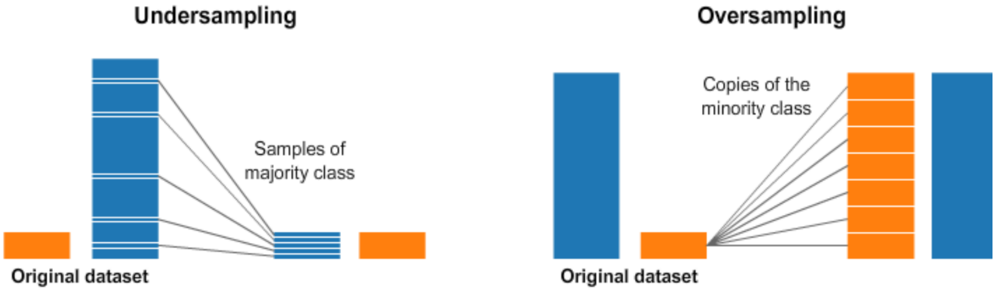


Figure 8: Undersampling vs oversampling

In the current situation, undersampling will lead to a huge loss of data (in the machine failure case, the minority class counts only 330 record while the majority, 9643). Therefore, among many oversampling techniques, one of the most common is **SMOTE** (*Synthetic Minority Over-sampling Technique*) [2].

The idea behind SMOTE is the following: for each point in the minority class, k nearest neighbours are found. Then by selecting one random neighbour, we compute the difference between the two points and then we multiply it by a random number $\in [0, 1]$. This gives us a synthetic example along the line between the two points.

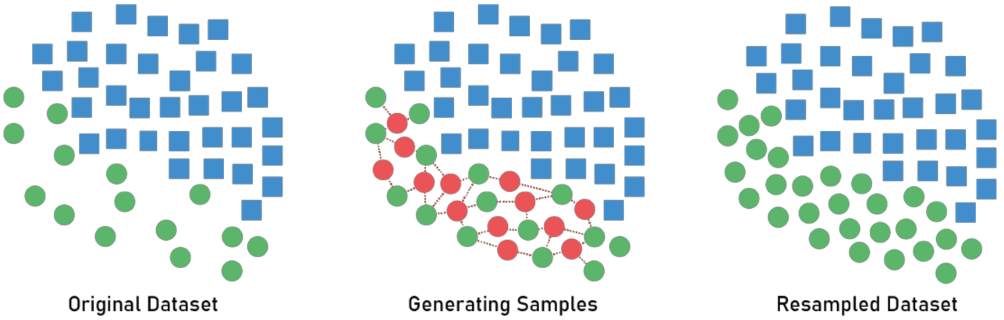


Figure 9: SMOTE technique

3.7 Dimensionality reduction

Curse of dimensionality is a common problem while optimizing a model with many features. With dimensionality reduction techniques, we try to map data from high dimensional space to a lower one. One of the most used method for dimensionality reduction is **PCA** (Principal Component Analysis).

PCA is an unsupervised algorithm whose goal is to project data to a lower dimensional space by maximizing the variance of each dimension. Formally, let $x_1, ..., x_n$ be the original vector from $\mathbb{R}^d$ and we want to find a matrix $W \in \mathbb{R}^{n,d}$ with $n < d$ such that $x \mapsto Wx$ with $Wx \in \mathbb{R}^n$. Furthermore, we need a matrix $U \in \mathbb{R}^{d,n}$ that is able to recover the original vector x from its compressed version $y = Wx$. In fact, the exact recovery is not possible. For this reason, PCA tries to find a linear transformation that minimize the difference between the original vector x and recovered vector $\tilde{x} = Ux$:

$$\underset{W \in \mathbb{R}^{n,d}, U \in \mathbb{R}^{d,n}}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - UXx_i\|_2^2 \tag{1}$$

It can be proven that the previous PCA formulation can be rewritten as:

$$\underset{U \in \mathbb{R}^{d,n}, U^T U = \mathbb{I}_n}{\operatorname{argmax}} \; trace\left(U^T \sum_{i=1}^{m} x_i x_i^T U\right)$$

If we define the matrix $A = \sum_{i=1}^{m} x_i x_i^T$ and its spectral decomposition (since it is symmetric) as $A = VDV^T$, we can easily see that the elements of D are the eigenvalues of A while the columns of V are the corresponding eigenvectors. From this, we can claim that the solution of (1) is the matrix U whose columns $u_1, ..., u_n$ are the n eigenvectors of A corresponding to the largest n eigenvalues (A is semidefinite positive), while $W = U^T$.

Regarding the dataset considered in this work, we decide to not apply a dimensionality reduction technique since the features are not so many and the number of instances is way bigger than the number of features.

4 Validation and metrics

4.1 Data splitting

The dataset is split into a training and a test set with a ratio of 80/20. In order to approximately preserve the same distribution of the target variable, a stratified split is performed.

4.2 Cross validation

While performing hyperparameters tuning, we have a high risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. To resolve such issue, we should leave out another part of the dataset in order to have a ”validation set”, but in this way we are drastically reducing the number of training samples. Cross validation could be really helpful in this scenario: you still need to keep the test set for the final evaluation, but the validation set is no longer needed.

During the years, many cross validation techniques have been proposed. Here we have chosen to use the so called **Stratified K-Folds cross validation**. In this technique, the training set is divided in k smaller set (called *”folds”*) which contains the same percentage of sample of each target class as the original set. Then for each fold, the model is trained on $k-1$ folds and validated on the remaining one. The cross-validation process is then repeated k times, with each of the k folds used exactly once as the validation data. The performance measure reported by k-fold cross-validation is then the average of the values computed.

4.3 Metrics

Metrics are very important functions because they are telling how is the performance of the chosen machine learning algorithm. The metrics taken into consideration are:

- **Accuracy**: measures the number of correct predictions over the total number of predictions. This metrics has a problem called *Accuracy paradox*: in the case of unbalanced data, for example 99% vs 1%, predicting completely

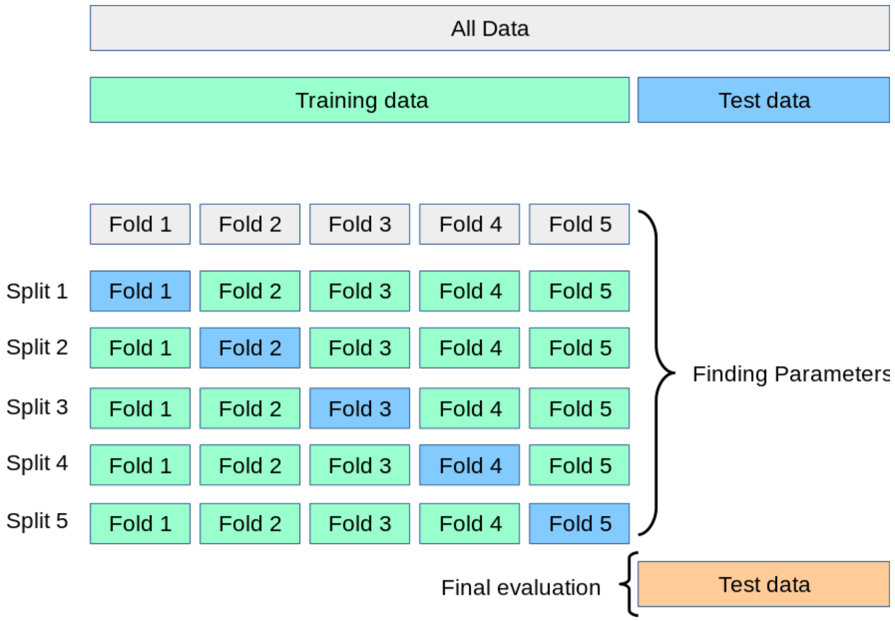


Figure 10: Hyperparameters tuning with cross validation

wrong the entire minority class would still lead to an impressive 99% accuracy score.

- **Precision**: measures the number of true predicted positives over the total number of positives. It is the ability of the classifier not to label as positive a sample that is negative.
- **Recall**: measure the number of correctly positive predicted values over the total actual positives. It is the ability of the classifier to find all the positive samples.
- **F1-score**: harmonic mean between precision and recall.
- **Confusion matrix**: matrix in which each row is an instance of the true class, and each column are the instances of the predicted classes.
- **ROC curve**: it is a plot illustrating the classification performance. Basically on the x-axis we have the false positive rate and on the y-axis the true positive rate. The ”ideal” point of this curve is in the top left corner: here we have a false positive rate of zero, and a true positive rate of one. Not very realistic but, a larger area under the curve (AUC) is usually better. In

the multi-label classification (e.g. the failure type classification) two averaging strategies are currently supported: the one-vs-one algorithm computes the average of the pairwise ROC AUC scores, and the one-vs-rest algorithm computes the average of the ROC AUC scores for each class against all other classes. In the current work, the latter is used.

- **Precision-Recall curve**: like the ROC curve, the Precision-Recall curve is a plot illustrating the classification performance. Furthermore, is a useful measure of success of prediction when the classes are very imbalanced. A high area under the curve represents both high recall (low false negative rate) and high precision (low false positive rate).

5 Classification models

In section 2.2 we have discussed that in this dataset we have two targets variable: **Machine Failure** (indicates if the machine will fail) and **Failure type** (indicates the type of failure if the machine failed). For this reason, we built two model for each of the chosen classification techniques.

5.1 SVM

Support Vector Machine (SVM) is a supervised method used both for classification and regression, whose goal is to find a hyperplane that maximize the distance between two classes in a feature space.

A training set S is linearly separable if:

$$\forall i \quad \exists w \in \mathrm{R}^{\mathrm{d}}, b \in \mathbb{R} : y_{\mathrm{i}} \left(\langle w, x_i \rangle + b \right) > 0 \tag{2}$$

For any separable training set, there are many possible hyperplanes: we take the one that maximize the margin, i.e. the minimal distance between a point of the training set and the hyperplane itself. From this assumption, we can obtain the formulation for **Hard Margin**:

$$\underset{(w,b):\|w\|=1}{\operatorname{argmax}} \ \min_{i\in[m]} |\langle w, x_i \rangle + b| \qquad \text{s.t.} \quad y_i \left(\langle w, x_i \rangle + b \right) > 0 \quad \forall i \tag{3}$$

The solution of the Hard-SVM is therefore a solution to the above formula:

$$\widehat{\mathrm{w}} = \frac{\mathrm{w}_0}{||w_0\|} \quad \hat{b} = \frac{\mathrm{b}_0}{||w_0||} \tag{4}$$

Linearly separable data is a quite hard assumption: for this reason a relaxation of the algorithm, called **Soft Margin**, has been proposed. We can allow the violation of the constraint, introducing *slack variables* $\xi_1, \dots, \xi_m$.

$$\min_{\mathrm{w},\mathrm{b},\xi} \left(\lambda ||\mathrm{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \qquad \text{s.t.} \quad y_i \left(\langle w, x_i \rangle + b \right) \geq 1 - \xi_{\mathrm{i}} \tag{5}$$

ξ_i measure how many violations we allow. We introduce a tuning parameter C called *Regularization parameter* in order to control the number of violation we allow. In fact: if C is small, the margin is smaller and it is rare to make mistake;

instead, if C in big, the margin is bigger and it is more likely to make mistakes. Formally, it holds that $\sum_{i=1}^{m} \xi_i \leq C$.

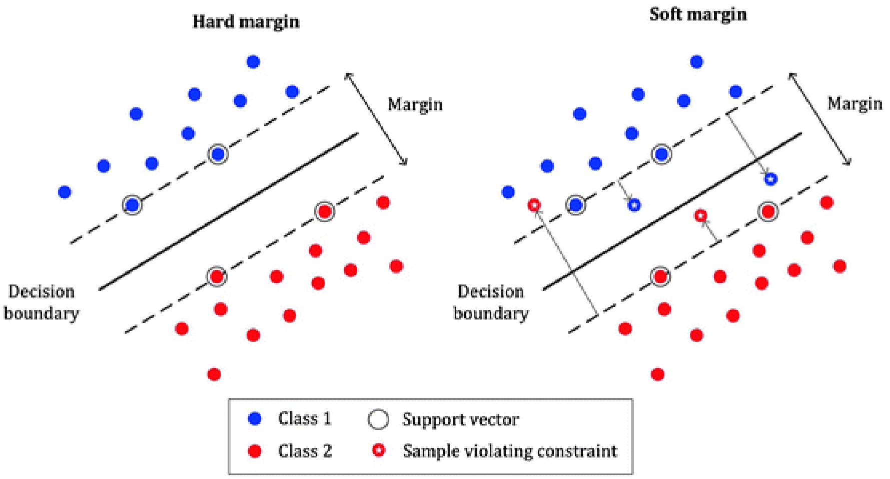


Figure 11: Hard margin vs soft margin

Very often data are not linearly separable. To resolve this issue, we map the data from the original space into a higher dimensional feature space. The goal is that after the transformation to the higher dimensional space, the classes are now linearly separable in this higher dimensional feature space. However, computing linear separators in very high dimensional space can be very computation expensive. The **kernel "trick"** provides a solution to this problem. The kernel function is a function that takes as its inputs vectors in the original space and return the dot product of the vector in the features space. Formally, if we have data $x, x' \in X$ and a map $\psi : X \to \mathbb{R}^n$, then a kernel function is defined as:

$$K(\mathrm{x}, \mathrm{x}') = \langle \psi(\mathrm{x}), \psi(\mathrm{x}') \rangle \tag{6}$$

There are different kernels available. In this work, we tested the following:

- Linear: $\langle x, x' \rangle$
- Polynomial: $(\gamma \langle x, x' \rangle + r)^d$
- RBF: $\exp^{-\gamma||x-x'||^2}$

5.1.1 Machine failure classification

Best parameters: `"C": 100, "gamma": 'scale', "kernel": 'rbf'`

Precision	Recall	F1-score	Accuracy
0.680	0.913	0.743	0.945

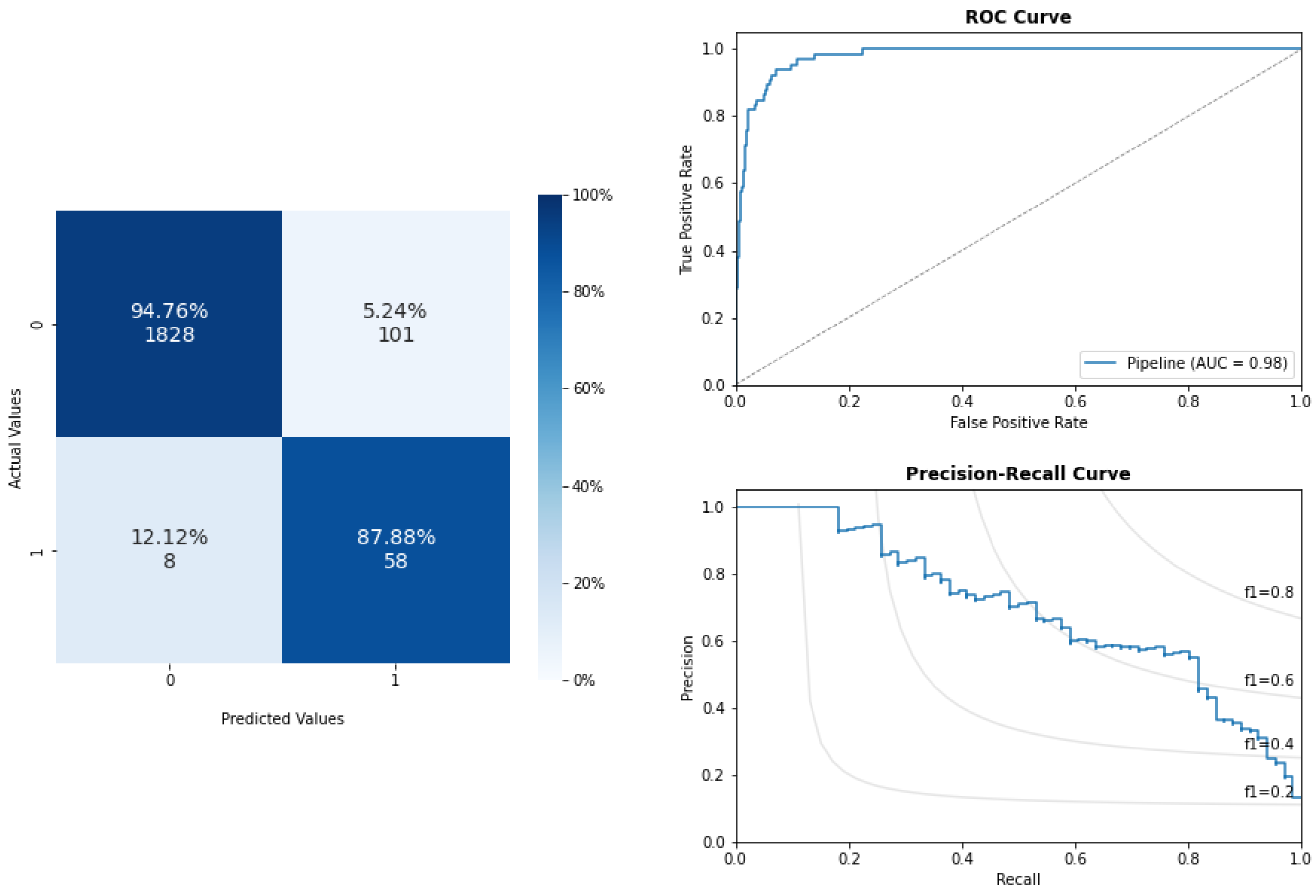


Figure 12: Machine failure SVM classifier results

5.1.2 Failure type classification

Best parameters: `"C": 100, "gamma": 'scale', "kernel": 'rbf'`

Precision	Recall	F1-score	Accuracy
0.609	0.780	0.671	0.958

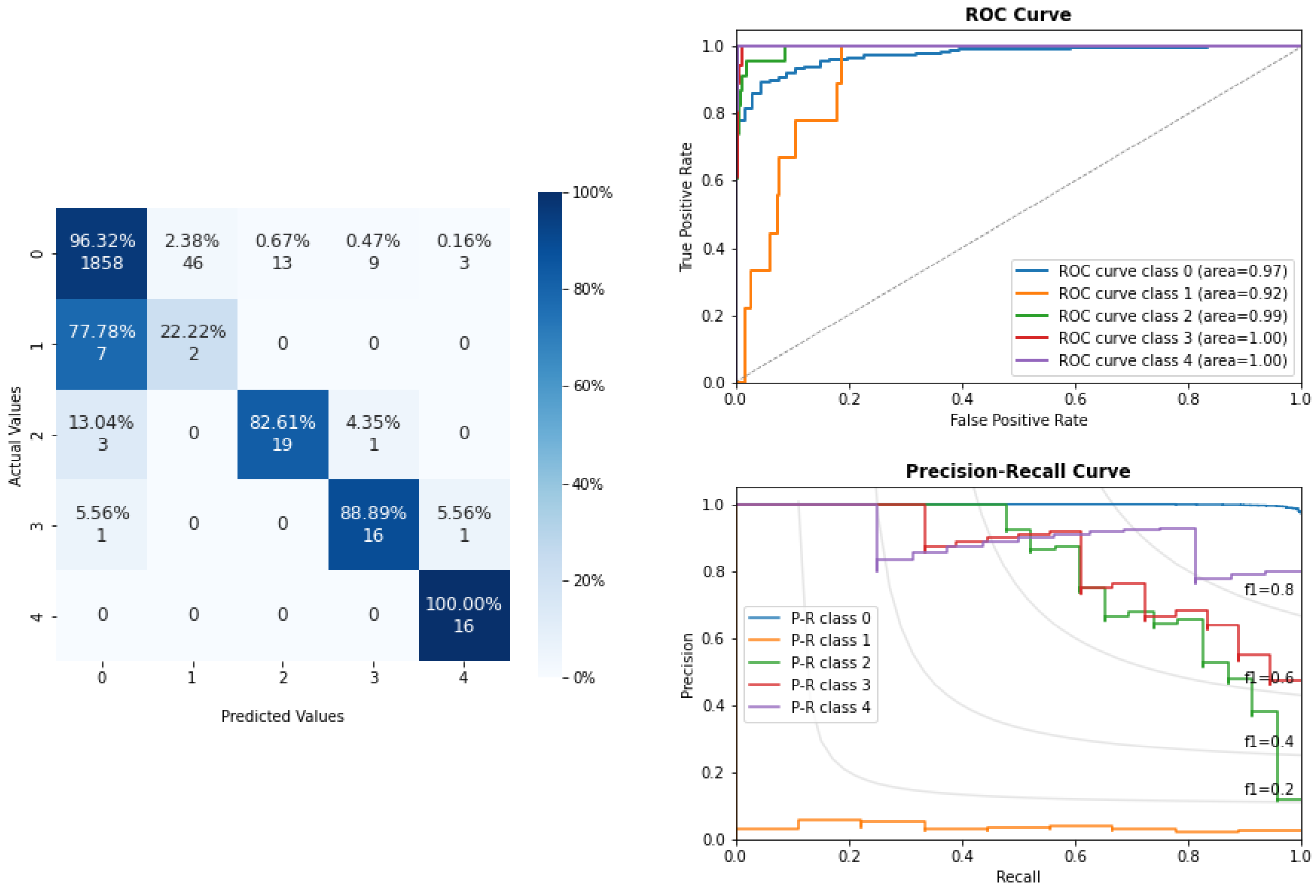


Figure 13: Failure type SVM classifier results

5.2 Decision trees

Decision Tree is a supervised learning algorithm whose goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The problem of finding every possible partition of the feature space is unfeasible at training time. For this reason, a top-down greedy approach is adopted. We start at the top of the tree and at each step, the local best split is performed, rather than the one that would lead to the best global result. The most common criteria used to obtain the best split are:

- **Gini Index**: $H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$
- **Cross entropy**: $H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$

where $H(Q_m)$ are the data at node m and p_{mk} is the proportion of observations from class k in node m.

The main disadvantage of decision trees is that they tend to overfit the training set, especially if we build very complex trees. A possible solution is to use a smaller tree by avoiding split that would lower the entropy (or Gini) no more than a threshold. In this case, a worthless split could be followed by a good split, which will never be performed.

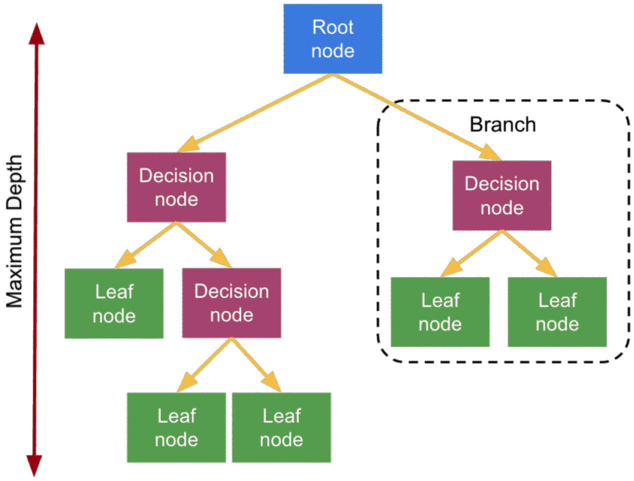


Figure 14: Decision tree structure

5.2.1 Machine failure classification

Best parameters: `"criterion": 'entropy', "max_depth": 20, "max_features": None, "splitter": 'best'`

Precision	Recall	F1-score	Accuracy
0.699	0.924	0.764	0.952

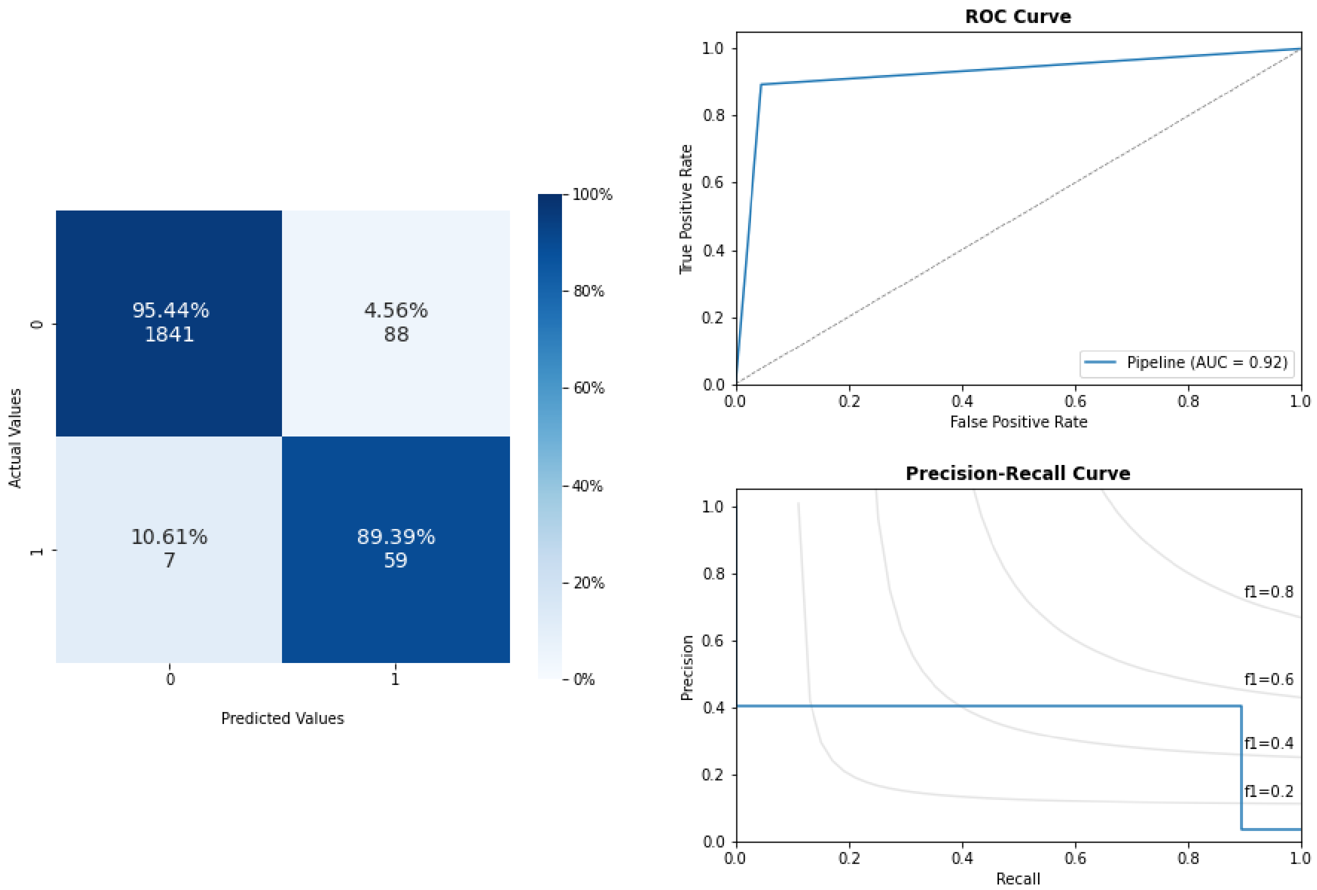


Figure 15: Machine failure decision tree classifier results

5.2.2 Failure type classification

Best parameters: `"criterion": 'entropy', "max_depth": None, "max_features": None, "splitter": 'best'`

Precision	Recall	F1-score	Accuracy
0.595	0.710	0.640	0.966

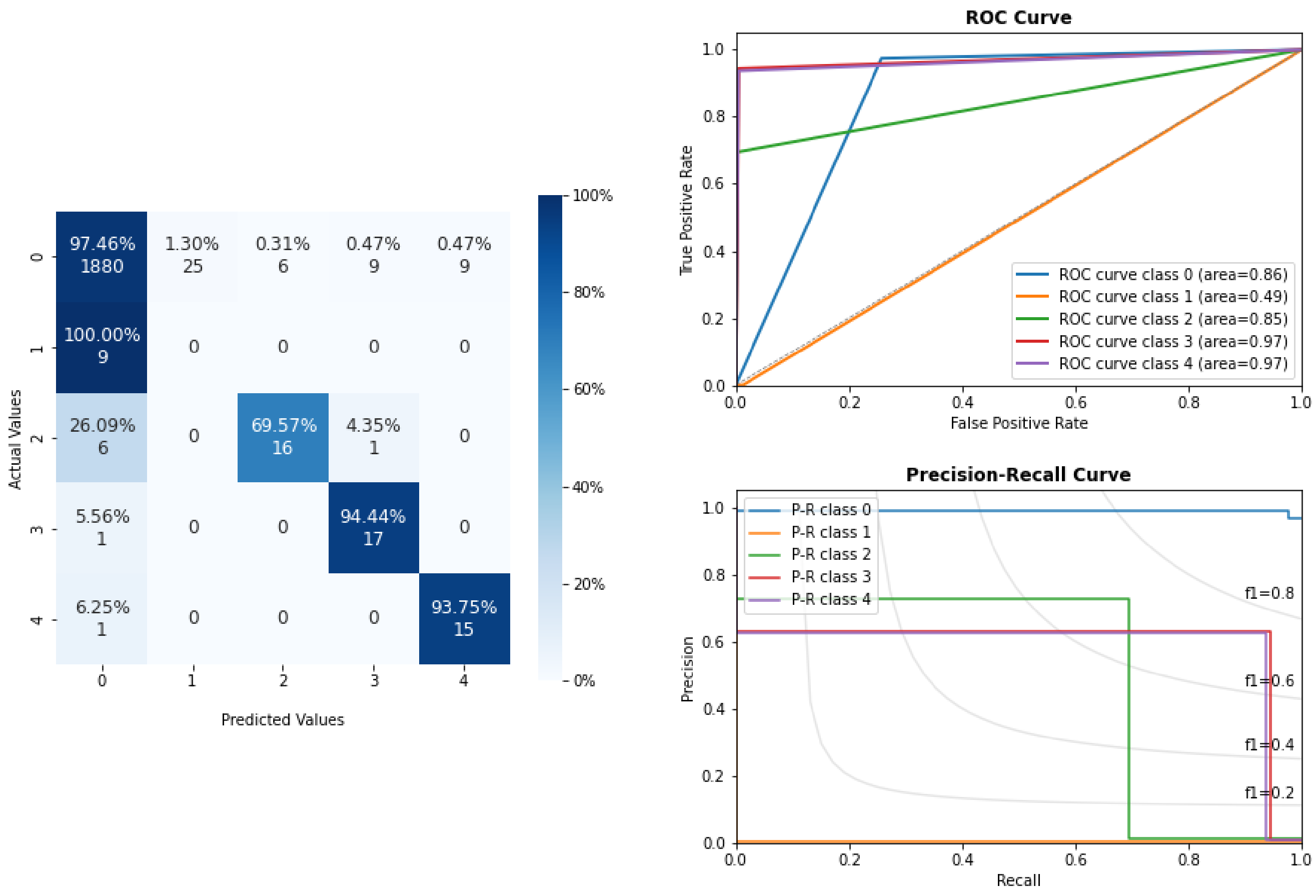


Figure 16: Failure type decision tree classifier results

5.3 Random forest

Random forest method is a supervised learning technique that uses decision trees by **bagging**. This technique in fact has been created in order to resolve the overfitting issue of decision trees. The main idea behind bagging is to combine multiple prediction functions learned from different dataset, reducing the variance. In fact, given a set of n independent observations $Z_1, ..., Z_n$ each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by σ^2/n. Therefore, B different bootstrapped training sets with the same size as the original dataset are generated by sampling uniformly at random with replacement from the training data. On those different sets, we can now build B separate models and, in a classification scenario, the class is chosen via majority voting among the different predictions.

5.3.1 Machine failure classification

Best parameters: `"criterion": 'gini', "max_depth": 20, "max_features": 'sqrt', "n_estimators": 300`

Precision	Recall	F1-score	Accuracy
0.708	0.890	0.767	0.957

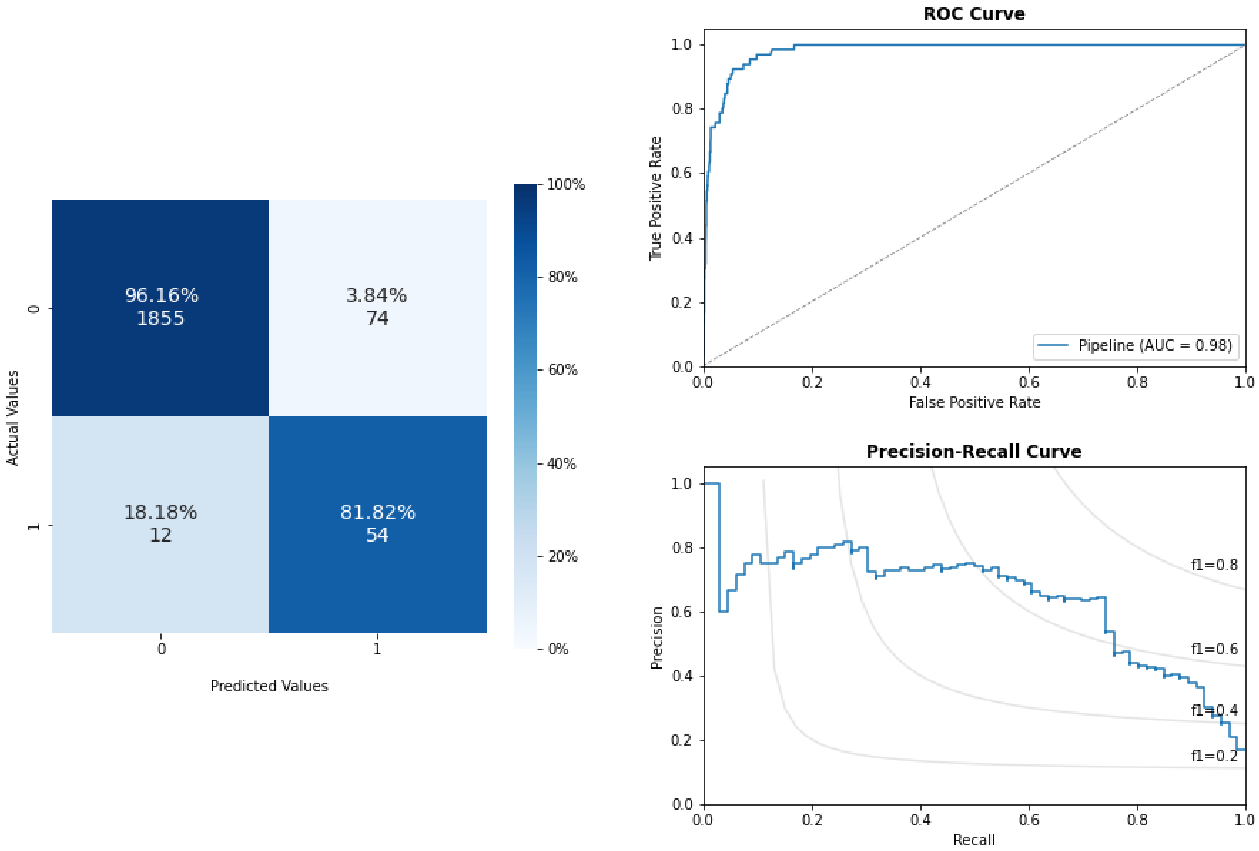


Figure 17: Machine failure random forest classifier results

5.3.2 Failure type classification

Best parameters: `"criterion": 'gini', "max_depth": None, "max_features": 'sqrt', "n_estimators": 500`

Precision	Recall	F1-score	Accuracy
0.594	0.672	0.615	0.967

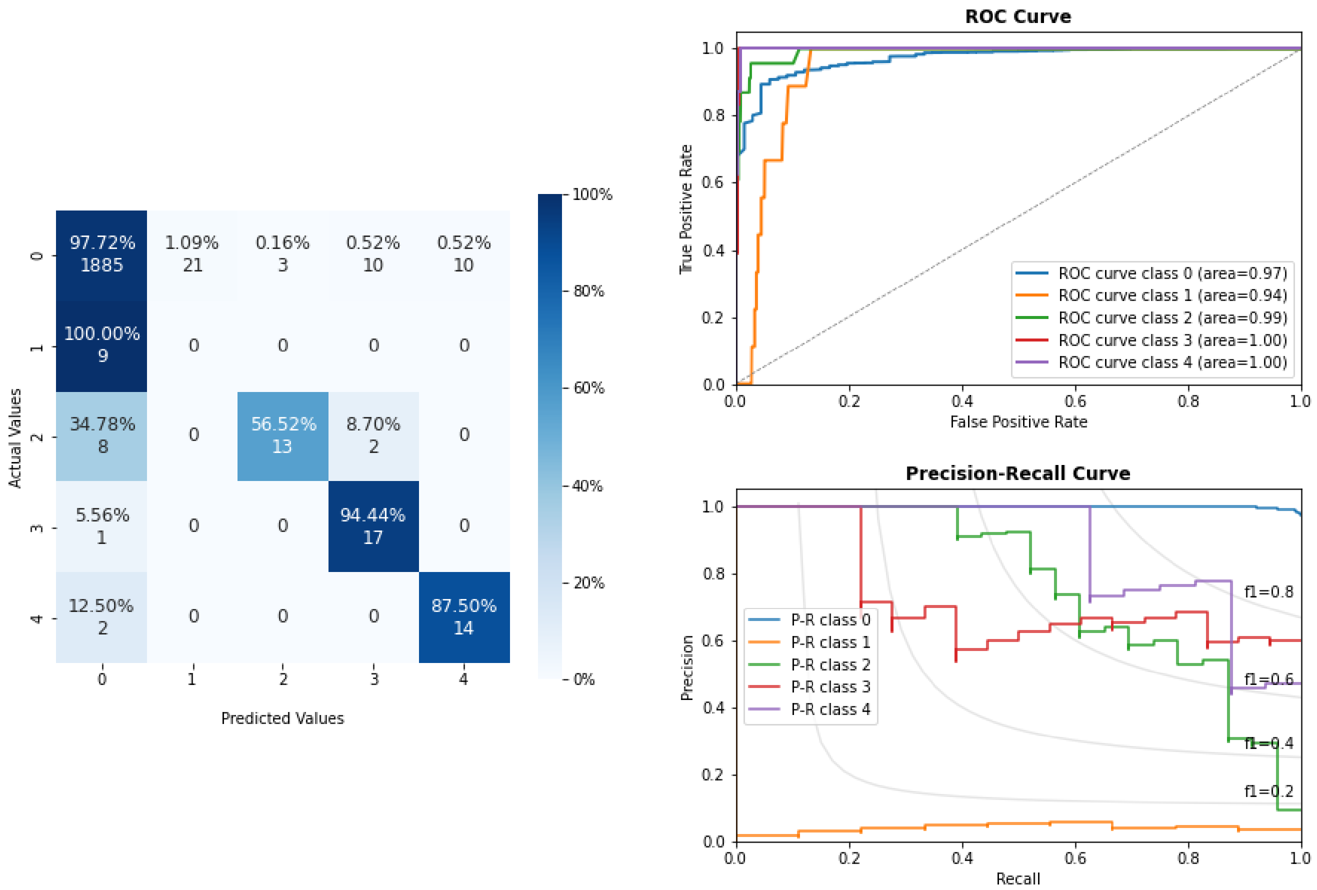


Figure 18: Failure type random forest classifier results

5.4 Logistic regression

Logistic regression, despite its name, is a linear model for classification. Instead of just predicting the class, we are also interested in the probability. To do so, we need a function (called *logistic function*) that returns values between 0 and 1.

In the binary case (e.g. $y \in [0, 1]$), we use the sigmoid function:

$$\mathbb{P}(y_i = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{7}$$

In order to obtain the class instead of the probability, we need to define a decision boundary. By default, if the probability is above 0.5, then the class 1 will be assigned and the class 0 otherwise.

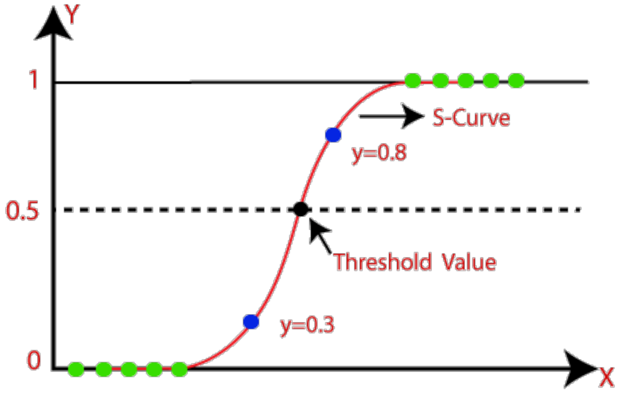


Figure 19: Logistic regression threshold

In case of multi-label classification (e.g. $y \in [0, ..., K]$), we use the softmax function:

$$\mathbb{P}(y_i = k) = \frac{e^{\beta_{0k} + \beta_{ik} X_i}}{\sum_{l=0}^{K-1} e^{\beta_{0l} + \beta_{il} X_i}} \tag{8}$$

5.4.1 Machine failure classification

Best parameters: `"C": 0.1, "max_iter": 100, "penalty": 'l2', "solver": 'newton-cg'`

Precision	Recall	F1-score	Accuracy
0.572	0.830	0.583	0.841

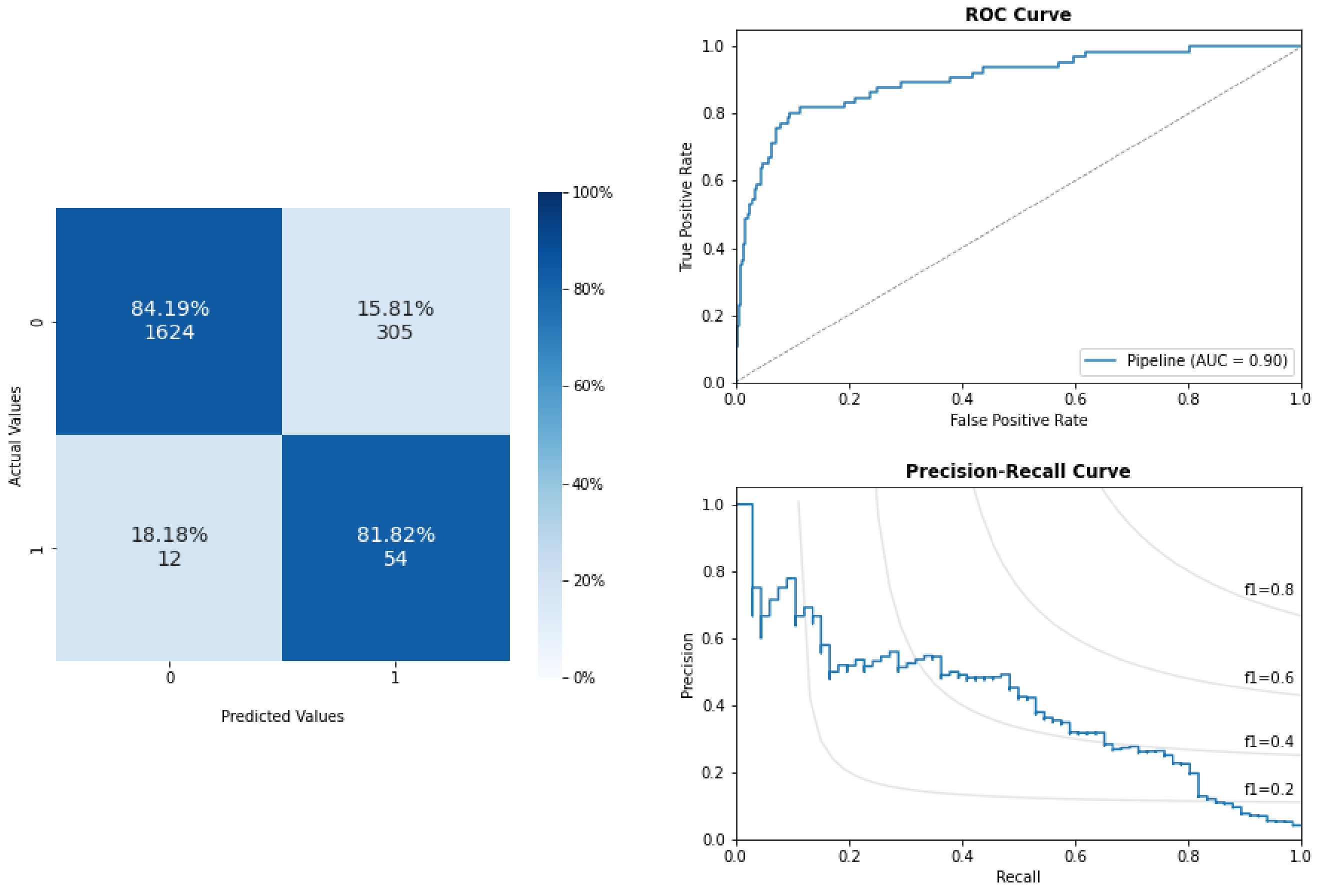


Figure 20: Machine failure logistic regression classifier results

5.4.2 Failure type classification

Best parameters: `"C": 10, "max_iter": 100, "penalty": 'l2', "solver": 'newton-cg'`

Precision	Recall	F1-score	Accuracy
0.421	0.892	0.514	0.886

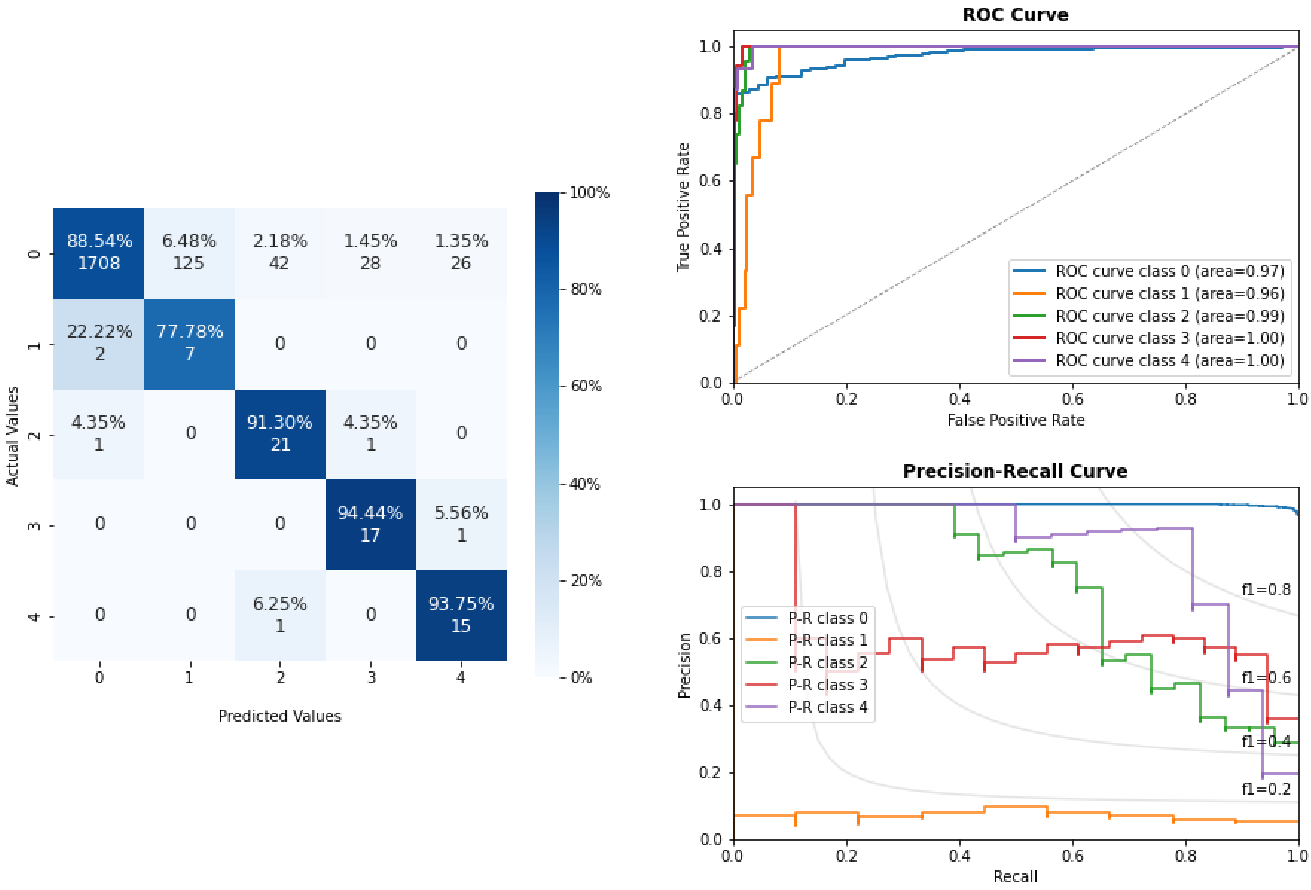


Figure 21: Failure type logistic regression classifier results

6 Conclusion

Comparing the results obtained in the previous section, we can easily see the impact of data imbalance. In both the classification task, we obtain a pretty high accuracy with all the machine learning model tested. But F1-score and precision scores get a negative influence from data imbalance. Regarding Precision, Recall and F1-score, the machine failure classification got higher results probably because the problem that we are addressing is more simple and the machine learning models are able to ”understand” it better.

Which model to choose depends on the final result that we need. For example, in the machine failure classification, we can choose a model with high recall because we allow to predict a machine failure also if it will not and we want to be sure that almost all the real failure will be detected. In the contrast, if we do not want any false positive failure alarm and we can allow that some machine failure will be classified as non failure, we may prefer a model with higher precision.

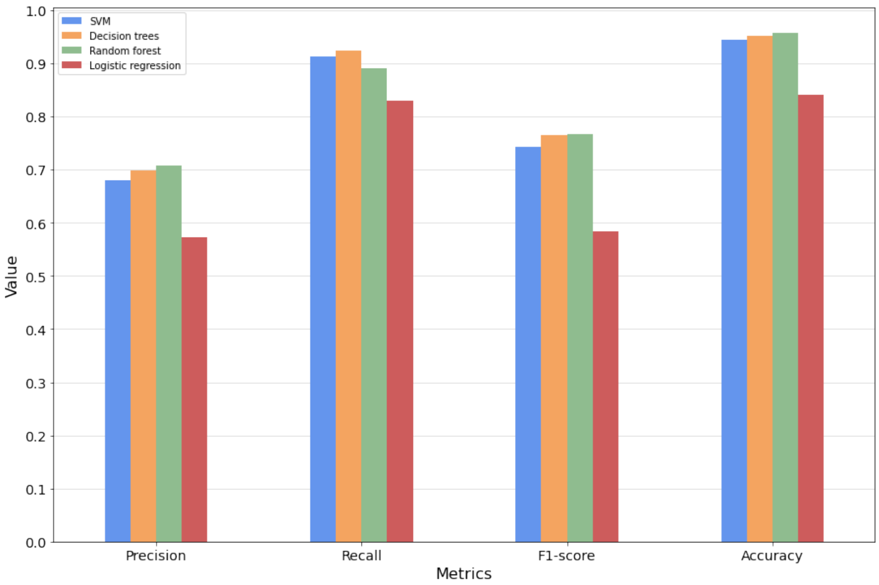


Figure 22: Results comparison for machine failure classification

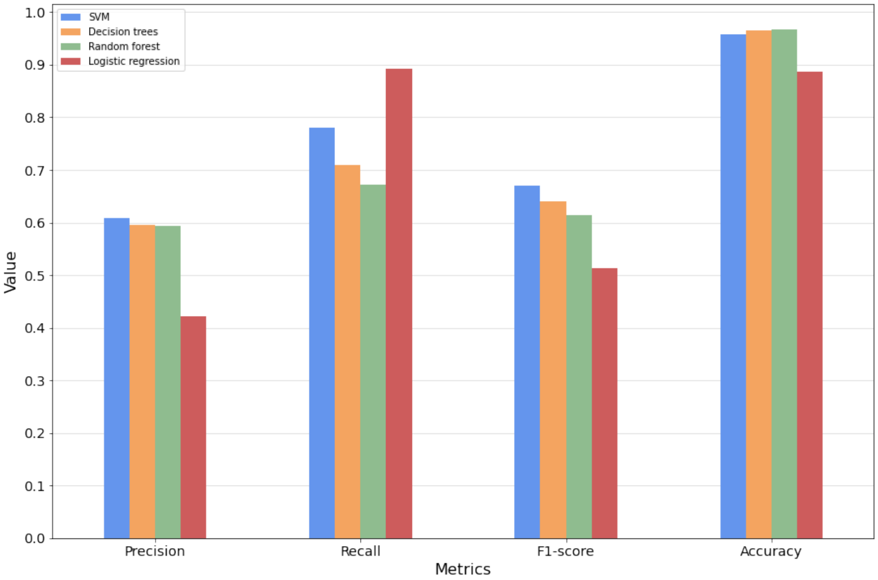


Figure 23: Results comparison for failure type classification

References

[1] S. Matzka, "Explainable artificial intelligence for predictive maintenance applications," in *2020 Third International Conference on Artificial Intelligence for Industries (AI4I)*, 2020, pp. 69–74.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, jun 2002.