

# **Predictive Maintenance: Leveraging Advanced Machine Learning Techniques**

**The University of Texas at Austin**

**Shubh Javia**

**Case Studies in Machine Learning**

**November 25th, 2024**

## **Abstract**

Predictive maintenance leverages machine learning to forecast equipment failures, minimize downtime, and optimize maintenance costs across various industries. This paper examines several machine learning techniques, using a supervised learning methodology to predict machine failures and classify the underlying cause. Key features such as air temperature, process temperature, tool wear, rotational speed, and power from operational data were analyzed. Binary classification models were used to predict machine failure, while multi-class classification models identified the causes of these failures. To deal with class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) was applied. Through comprehensive data preprocessing and feature engineering, with Recall as the primary metric to minimize false negatives, XGBoost achieved the highest predictive accuracy and recall while keeping the F-1 score balanced. These results demonstrate the potential of predictive maintenance to improve operational efficiency and safety.

## 1. Introduction

Machinery is used across multiple industries, and equipment failures can result in operational downtime, environmental risks, human danger, and significant financial losses. Predictive maintenance addresses these issues by utilizing advanced data analytics methods to predict machine failures before they occur, leveraging real-time data to enable timely interventions. Traditional machinery maintenance has typically followed a reactive or corrective approach, which is expensive and inefficient (Carvalho et al., 2019). In contrast, predictive maintenance empowered by machine learning, allows industries to monitor equipment performance, predict failures, and make informed decisions. This approach creates a safer environment, optimizes processes, and leads to financial savings—substantially improving traditional methods (Bousdekis et al., 2019).

Machine learning in predictive maintenance involves processing and analyzing large volumes of real-time sensory data, which includes features such as air temperature, process temperature, rotational speed, tool wear, torque, and in some cases features containing the reason for failure. Through classification tasks, we can predict whether a failure will occur and classify the underlying cause for the failure.

This paper presents a machine learning approach to predictive maintenance aimed at predicting two key scenarios: the first is predicting whether a failure will occur, also known as binary classification, and the second is classifying the cause of the failure across multiple categories, known as multi-class classification. Missing a true failure is critical and must be avoided at all costs. Therefore, a key focus of our approach will be optimizing recall to minimize false negatives, ensuring that all actual failures are considered. We will employ several classification models for model training and evaluation, including Logistic Regression, Decision Trees, Bagging, Boosting, and XGBoost. Extensive data processing and feature engineering will be conducted to prepare raw sensory data for these models (Spiegel et al., 2018). Our model evaluation will consider various classification metrics such as recall, accuracy, precision, and F1-score. For our best-performing model, we will also look at the most important features for prediction.

Our dataset consists of imbalanced classes, so we will apply the Synthetic Minority Over-sampling Technique (SMOTE) to address these issues. Additional features such as temperature difference and power will also be created to help the model understand patterns.

The remainder of this paper is organized as follows: Literature Review, Data Collection, Data Preprocessing and Feature Engineering, Model Evaluation, Evaluation Metrics, Conclusion, and Future Works.

## 2. Literature Review

Advances in machine learning techniques have significantly contributed to the evolution of Predictive Maintenance (PdM) over the last several years. As industries increasingly shift from traditional maintenance approaches to analytical, predictive methods, the fourth industrial revolution (Industry 4.0) has played a pivotal role in enabling this. Industry 4.0, as discussed in an article by McKinsey & Company, represents the next phase of digitization in the manufacturing and operations sector. Driven by the rise of data, connectivity, analytics, human-machine interaction, and advancements in robotics and technology, this revolution aims to create a highly interconnected and intelligent environment. This technological revolution lays the foundation for PdM, allowing computers to monitor real-time machine data and provide enhanced diagnostics (McKinsey & Company, 2022).

Various types of Machine Learning methodologies can be used for predictive maintenance. An article by IBM (IBM, 2022) explores different models for this case. Three kinds of models can be used primarily; Clustering, Time Series, and Classification.

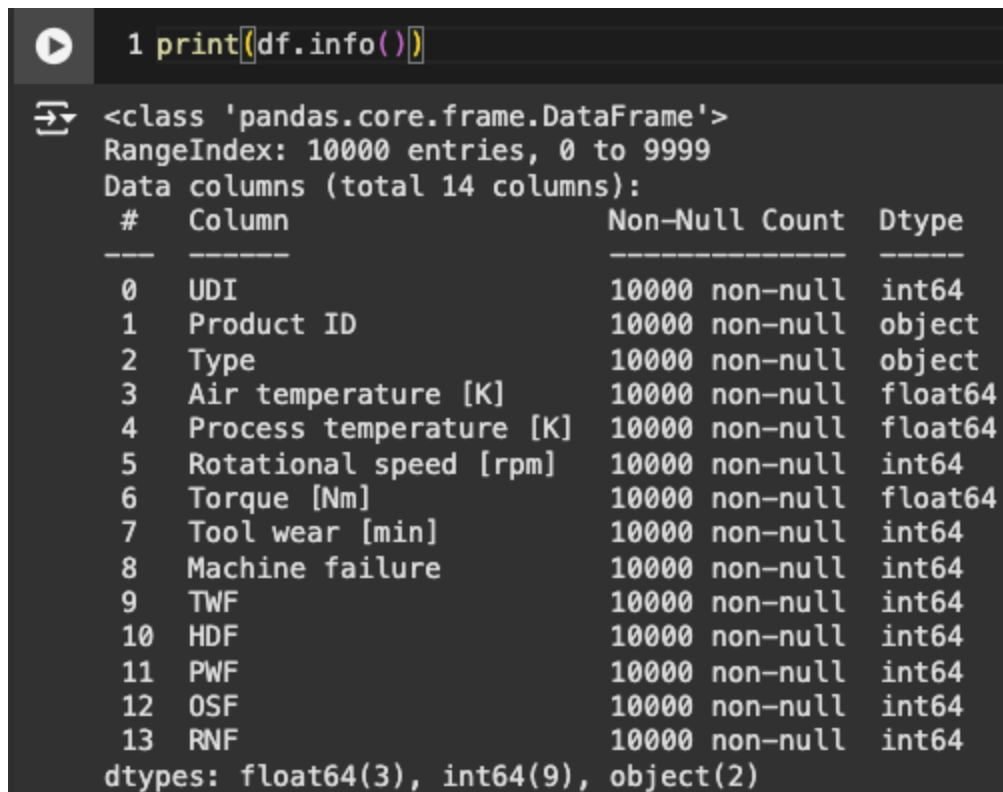
- **Clustering** is a part of the Unsupervised learning group where we do not know the target classes and data gets grouped according to similar characteristics which can reveal patterns related to machine failures. A research paper focused on grouping data from Laser Melting Machines (Uhlmann et al., 2018) explores clustering algorithms. Additional uses can be Anomaly Detection, Trend Analysis for the remaining useful life (RUL) of machines, and Condition-Based maintenance.

- **Time-series** models are used in Predictive maintenance to determine at what period a machine is likely to fail so the engineer can get ahead of it and perform maintenance. In a research paper (Lin et al., 2019) - they look at how time series models can be applied to PdM in industries like machine tools and solar-cell manufacturing by using historical data ordered by time to learn patterns and predict the future state of the machines.
- **Classification** is the methodology to classify data into either 2 or more classes. A paper by Chazhoor (Chazhoor et al., 2020) explores multiple different classification models like KNN classifier, random forest, decision tree, and multi-layer perceptron to predict failures in the Semiconductor manufacturing process dataset. Due to class imbalance, certain models did not perform well, however Logistic regression with PCA gave the best results. Upon applying SMOTE and feature engineering, the random forest seemed to perform better in accuracy and ROC curve.

A paper by Stephen (Matzka, 2020), investigates different ML models to classify machine failures, the same dataset used in this paper. The study examines models such as bagged tree ensemble classifiers and artificial neural networks. Through correlation analysis, the authors identified two features with little to no importance. In one of their models, they trained 15 decision trees with a maximum of 4 nodes. Interestingly, not all features were used in every tree—low-importance features appeared in only about 3 of the 15 trees. Marktan compares two approaches to explain the results of these evaluations: one method involves using multiple decision trees to explain which features were used in each tree, while the second approach normalizes feature deviation by scaling each feature to a mean of 0 and a standard deviation of 1. After stratifying 20 random data points to include all five failure types, they found that the model struggled to predict tool wear failure due to the low volume of samples. By using normalized feature deviation, the model achieved explanations for all classes. The study concluded that decision trees provided higher-quality results but lacked consistent explanations, whereas normalized features provided lower-quality but consistent explanations for inference.

### 3. Data Collection

The data used in this study is sourced from the Predictive Maintenance Dataset available through the University of California, Irvine's Machine Learning Repository (UCI Machine Learning Repository, n.d.), which provides an open platform for various datasets. Due to the challenges associated with obtaining real-world predictive maintenance datasets, UCI has created synthetic data that mimics real-world scenarios encountered in the industry. The dataset consists of 10,000 rows and 14 columns in total. As shown in Figure I below.



```
1 print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   UDI                                    10000 non-null  int64
1   Product ID                           10000 non-null  object
2   Type                                  10000 non-null  object
3   Air temperature [K]                  10000 non-null  float64
4   Process temperature [K]              10000 non-null  float64
5   Rotational speed [rpm]               10000 non-null  int64
6   Torque [Nm]                         10000 non-null  float64
7   Tool wear [min]                     10000 non-null  int64
8   Machine failure                     10000 non-null  int64
9   TWF                                  10000 non-null  int64
10  HDF                                  10000 non-null  int64
11  PWF                                  10000 non-null  int64
12  OSF                                  10000 non-null  int64
13  RNF                                  10000 non-null  int64
dtypes: float64(3), int64(9), object(2)
```

Figure I: Column information

The data primarily comprises integer and float-type values, with two categorical columns. Figure II provides a detailed schema of the dataset, explaining the meaning of each column and how it was derived.

UID: unique identifier ranging from 1 to 10000  
 product ID: 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  
 air temperature [K]: generated using a random walk process later normalized to a standard deviation of 2 K around 300 K  
 process temperature [K]: generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.  
 rotational speed [rpm]: calculated from a power of 2860 W, overlaid with a normally distributed noise  
 torque [Nm]: torque values are normally distributed around 40 Nm with a  $\sigma = 10$  Nm and no negative values.  
 tool wear [min]: The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process. and a 'machine failure' label that indicates, whether the machine has failed in this particular datapoint for any of the following failure modes are true.  
  
 The machine failure consists of five independent failure modes  
 tool wear failure (TWF): the tool will be replaced or fail at a randomly selected tool wear time between 200 – 240 mins (120 times in our dataset). At this point in time, the tool is replaced 69 times, and fails 51 times (randomly assigned).  
 heat dissipation failure (HDF): heat dissipation causes a process failure, if the difference between air- and process temperature is below 8.6 K and the tool's rotational speed is below 1380 rpm. This is the case for 115 data points.  
 power failure (PWF): the product of torque and rotational speed (in rad/s) equals the power required for the process. If this power is below 3500 W or above 9000 W, the process fails, which is the case 95 times in our dataset.  
 overstrain failure (OSF): if the product of tool wear and torque exceeds 11,000 minNm for the L product variant (12,000 M, 13,000 H), the process fails due to overstrain. This is true for 98 datapoints.  
 random failures (RNF): each process has a chance of 0,1 % to fail regardless of its process parameters. This is the case for only 5 datapoints, less than could be expected for 10,000 datapoints in our dataset.  
  
 If at least one of the above failure modes is true, the process fails and the 'machine failure' label is set to 1. It is therefore not transparent to the machine learning method, which of the failure modes has caused the process to fail

Figure II: schema definitions (UCI Machine Learning Repository, n.d.)

#### 4. Data Pre-Processing and Feature Engineering

The first part of the problem involves predicting whether a machine will fail. To begin, I examined the distribution of the target column to check for class imbalance. Upon plotting the value counts of the target class, it became evident that there was a significant imbalance. Class 0 (no failure) accounted for 9,661 cases (96.61% of the total), while Class 1 (failure) had only 339 cases (3.39%)—as shown in Figure III below. This imbalance could cause issues during model training, so addressing it became a priority. To tackle this, I used the Synthetic Minority Oversampling Technique (SMOTE).

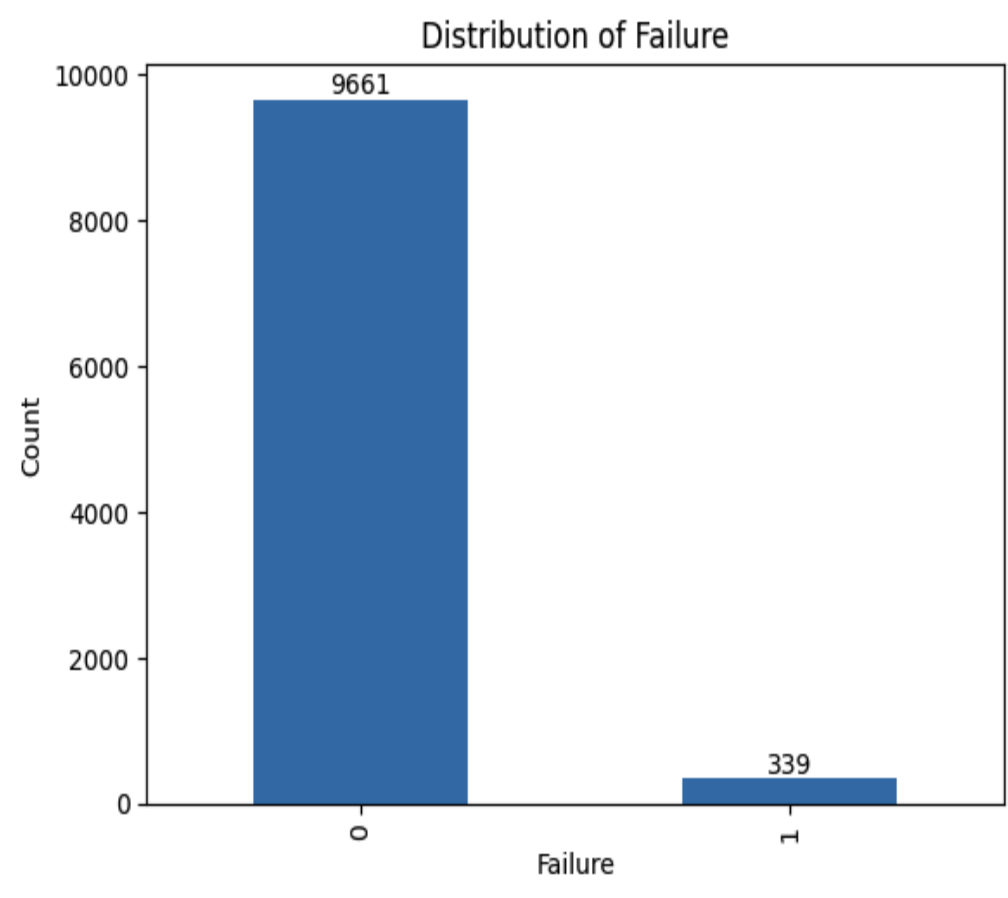


Figure III: Target column distribution

### **SMOTE (Synthetic Minority Oversampling Technique)**

Generally, oversampling is always preferred over under-sampling because removing samples could lead to loss of data. Under-sampling should only be performed if there is a specific reason. In our case that would mean getting rid of an entire class of target columns. We use SMOTE to over-sample and balance the cases for both classes by generating synthetic samples. To understand how SMOTE algorithm works I will take an excerpt from a paper where SMOTE was used to treat class imbalance on a big dataset (Blagus & Lusa, 2013).

- 1) It takes the difference between a sample and its nearest neighbor
- 2) Multiply the difference by a random number between 0 and 1

- 3) Add this difference to the sample to generate a new synthetic example in the feature space

A visual representation of the SMOTE algorithm is provided in Figure IV below.

**Algorithm** *SMOTE*( $T$ ,  $N$ ,  $k$ )

**Input:** Number of minority class samples  $T$ ; Amount of SMOTE  $N\%$ ; Number of nearest neighbors  $k$

**Output:**  $(N/100) * T$  synthetic minority class samples

1. (\* If  $N$  is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd. \*)
2. **if**  $N < 100$
3.     **then** Randomize the  $T$  minority class samples
4.      $T = (N/100) * T$
5.      $N = 100$
6. **endif**
7.  $N = (\text{int})(N/100)$  (\* The amount of SMOTE is assumed to be in integral multiples of 100. \*)
8.  $k$  = Number of nearest neighbors
9.  $\text{numattrs}$  = Number of attributes
10.  $\text{Sample}[\ ][\ ]$ : array for original minority class samples
11.  $\text{newindex}$ : keeps a count of number of synthetic samples generated, initialized to 0
12.  $\text{Synthetic}[\ ][\ ]$ : array for synthetic samples  
(\* Compute  $k$  nearest neighbors for each minority class sample only. \*)
13. **for**  $i \leftarrow 1$  **to**  $T$
14.     Compute  $k$  nearest neighbors for  $i$ , and save the indices in the  $\text{nnarray}$
15.      $\text{Populate}(N, i, \text{nnarray})$
16. **endfor**
17.  $\text{Populate}(N, i, \text{nnarray})$  (\* Function to generate the synthetic samples. \*)
18. **while**  $N \neq 0$
19.     Choose a random number between 1 and  $k$ , call it  $nn$ . This step chooses one of the  $k$  nearest neighbors of  $i$ .
20.     **for**  $\text{attr} \leftarrow 1$  **to**  $\text{numattrs}$
21.     Compute:  $\text{dif} = \text{Sample}[\text{nnarray}[nn]][\text{attr}] - \text{Sample}[i][\text{attr}]$
22.     Compute:  $\text{gap} = \text{random number between } 0 \text{ and } 1$
23.      $\text{Synthetic}[\text{newindex}][\text{attr}] = \text{Sample}[i][\text{attr}] + \text{gap} * \text{dif}$
24.     **endfor**
25.      $\text{newindex}++$
26.      $N = N - 1$
27. **endwhile**
28. **return** (\* End of  $\text{Populate}$ . \*)

End of Pseudo-Code.

Figure IV: SMOTE mathematical model. (Blagus & Lusa, 2013)



Additionally, I split the dataset into training and testing sets, with 75% allocated for training (7,500 rows) and 25% for testing (2,500 rows). The shape of the data for training is now 7,500 rows x 8 columns for features and 7,500 rows x 1 column for labels. After SMOTE, the total record count for both the features and labels increased to 14,466, with an equal distribution of Class 0 and Class 1.

Before SMOTE: (7500, 8) for features, (7500,) for labels

After SMOTE: (14466, 8) for features, (14466,) for labels

Label Variance After SMOTE:

Class 0: 7233

Class 1: 7233

## Feature Engineering

Feature engineering is a critical step in machine learning that plays a significant role in model performance. As summarized by the Milwaukee School of Engineering (The Importance of Feature Engineering in Machine Learning, 2024), "A feature is a measurable input that predictive models can use. Features are variables, and feature engineering is the process of selecting, transforming, and creating relevant input from raw data." Effective feature engineering can enhance model performance, reduce computational costs, and improve interpretability. The article also identifies the four main components of feature engineering: feature creation, transformations, feature extraction, and feature selection

Upon reviewing the data's characteristics and schema definitions, it became clear that two features needed to be created: one to calculate power and another to calculate the temperature difference.

Power = 'Torque [Nm] \* Rotational speed [rpm]

Temp\_Diff = \*abs(Air temperature [K] - Process temperature [K])

\*The absolute value was taken to avoid a negative difference.

Additionally, the dataset contains a categorical column called "Type," which denotes the machine type, it contains three distinct values: Medium(M), Light(L), and Heavy(H). Some of the models we intend to use only accept numerical inputs, so it will be necessary to convert these categorical values into numerical representations. To accomplish this, we performed Label Encoding, which assigns numeric values (0, 1, and 2) to the three machine types. Refer to the following article (Team, 2020) to learn more about Label encoding and how it is applied.

In the raw data, some features denote the cause of machine failure. Since we approach this problem in two parts, the first task is to predict if a machine will fail (binary classification: 0 vs. 1). We need to remove the failure reason features as well as IDs, we dropped the following features: 'Product ID', 'UDI', 'TWF', 'PWF', 'HDF', 'OSF', and 'RNF'. Including failure reason in part one, would create complications, as we are only concerned with predicting failure occurrence, and since only the failure class will contain failure reason the models can easily cheat.

The second part of the problem will focus on identifying the reason for failure. We retained the target columns removed in part one and filtered only the samples with machine failure class 1. However, the current format of the "target columns" (each representing a separate failure reason) is not ideal for model training. Therefore, we un-pivoted these columns into a single column called "failure\_reason", which serves as the target variable for multi-class classification. In cases where no failure reason is recorded, we assign the "random machine failure (RNF)" class

Figures V and VI below show snapshots of the dataset before and after applying this transformation.

Machine failure	TWF	HDF	PWF	OSF	RNF	Temp_Diff	Power
0	0	0	0	0	0	10.5	66382.8
0	0	0	0	0	0	10.5	65190.4
0	0	0	0	0	0	10.4	74001.2
0	0	0	0	0	0	10.4	56603.5
0	0	0	0	0	0	10.5	56320.0

Figure V

Machine failure	Temp_Diff	Power	Failure_Reason
1	10.2	13160.6	PWF
1	10.1	92637.0	PWF
1	10.1	60091.5	TWF
1	9.8	77817.4	OSF
1	9.8	73847.6	OSF

Figure VI

## 5. Model Training and Evaluation

To begin the model training process, we will start with a simple model by using Logistic Regression to predict the first part of our problem: whether or not a machine will fail.

In this section, we will evaluate the models using two primary metrics: Accuracy and Recall. The focus on Recall is crucial, as we aim to minimize false negatives—ensuring that, if a failure occurs, the model does not incorrectly classify it as "no failure." Although additional evaluation metrics, such as Precision and F1-Score, are also important, they will be discussed in detail in a later section of the paper.

### Logistic Regression

Logistic Regression is a widely used machine learning model for binary classification tasks. Despite its name, "regression," it is primarily used for classification. As explained in an

article by Spiceworks Publications (Kanade, 2022), "Logistic Regression analyzes the relationship between one or more independent variables and classifies data into discrete classes. The model estimates the mathematical probability of whether an instance belongs to a specific class. Logistic Regression is particularly effective when the data is linearly separable."

The article also discusses the mathematical foundation of Logistic Regression. To simplify, this model uses a function called sigmoid, which maps input values to a range between 0 and 1. The sigmoid curve, an S-shaped function, is used to estimate probabilities. Figures VII and VIII show the formulas for the sigmoid function and logistic regression.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure VII: sigmoid activation function (Kanade, 2022)

$$y = \frac{e^{(b_0 + b_1X)}}{1 + e^{(b_0 + b_1X)}}$$

Figure VIII: Logistic regression (Kanade, 2022)

In the logistic regression equation, X represents the input value, y is the predicted outcome,  $b_0$  is the bias (intercept), and  $b_1$  is the coefficient for the input X. The article provides further analysis of the formula and the model.

For model training, we use the sci-kit-learn library. The dataset is split into 75% training data and 25% testing data. For this model, we conducted three separate training sessions: first on the raw data (which contained class imbalances), then on pre-processed data with class balancing

applied, and finally, with a threshold value of 0.3. This threshold adjusts the prediction decision: a probability of more than 30% is classified as a "failure." The default threshold is 0.5, but depending on the business requirements, the threshold should be adjusted. In our case, I decided any prediction with more than a 30% chance of failure is considered significant. While adjusting the threshold improves recall, it also impacts the precision value, so careful tuning is necessary.

Here are the results from the three different configurations:

- Using Raw Data Metrics:
  - Accuracy: 97.6%
  - Recall: 31%
- Using Pre-processed Data Metrics:
  - Accuracy: 86.7%
  - Recall: 85.7%
- Using Threshold of 0.3 Metrics:
  - Accuracy: 85.7%
  - Recall: 93.7%

## **Decision Tree Classifier**

A Decision Tree Classifier is a supervised machine learning algorithm that uses a tree-like structure to make decisions. It begins with a root node, that represents the entire dataset and serves as the first decision point. Internal nodes represent decisions based on specific features, with branches connecting them to other nodes. Finally, the leaf nodes represent the final decision or classification.

Decision trees use a concept called impurity to measure the quality of a decision at each node. To understand impurity, let's refer to an article by Medium (MLMath.io, 2019). There are various measures of impurity, but for this paper, we focus on two: Gini Impurity and Entropy. Each node in the tree contains a subset of the data, and Entropy measures the uncertainty of the model when making a split at that node (MLMath.io, 2019). Lower entropy means higher

confidence in the model's decision. The mathematical formula for entropy is shown in Figure IX, where  $P_i$  represents the probability of class  $i$  summed over all possible classes.

$$Entropy = - \sum_{i=1}^n p_i * \log(p_i)$$

Figure IX (MLMath.io, 2019)

On the other hand, the Gini Index (or Gini Impurity) measures how similar the data points are within a node. The Gini index ranges from 0 to 1, with 0 indicating that the data points are identical (MLMath.io, 2019). Figure X shows the formula for calculating Gini.

$$Gini\ index = 1 - \sum_{i=1}^n p_i^2$$

Figure X (MLMath.io, 2019)

For part one of the problem (predicting whether a machine will fail or not), the Decision Tree algorithm in this study uses the Gini Index as the criterion, with a maximum depth of 8. The maximum depth defines how many splits the tree can make. When tuning hyperparameters in decision trees, be careful with overfitting. A deeper tree can capture more complex patterns but also increases the risk of overfitting. Therefore, finding a balance between model complexity and generalization is crucial.

Here are the evaluation metrics for part one:

- Accuracy: 93.2%
- Recall: 95.1%

For part two of the problem (predicting the reason for failure), we modify the model slightly since we now need to classify multiple classes (failure reasons). In this case, the decision tree is trained with a maximum depth of 5, minimum samples per split of 9, and minimum samples per leaf of 5. To learn more about these hyperparameters, please refer to the scikit-learn documentation (DecisionTreeClassifier, 2024).

Here are the evaluation metrics for part two:

- Accuracy: 90.6%
- Recall: 90.6%

### **Bagging Classifier**

Bootstrap Aggregating, or “Bagging”, was introduced by Breiman in 1996 (Breiman, 1996). Bagging is an ensemble learning technique that improves the performance of machine learning models by training multiple base models on different subsets of the data and combining their predictions. These base models are trained on bootstrapped samples, which are random samples drawn from the original dataset. Once trained, the model makes predictions by averaging the individual predictions of the base models. In a paper by Thomas G. Dietterich (Dietterich, 2000), the concept of ensemble methods is explained, explaining that these methods classify new data points by taking a weighted vote of their predictors. While the original approach was Bayesian averaging, Bagging is a more recent and popular technique. The paper discusses why ensemble methods are generally more powerful than using a single classifier by comparing them to previous methodologies.

We used a Decision Tree Classifier as our base model for part one, with a max\_depth of 7 and the Gini as the criterion. We set n\_estimators (the number of trees in the ensemble) to 10.

Here are the evaluation metrics for part one:

- Accuracy: 96.8%
- Recall: 97.1%

For part two, we used the same Decision Tree base model but increased the `n_estimators` to 50.

Here are the evaluation metrics for part two:

- Accuracy: 92.9%
- Recall: 93%

### **Boosting Classifier**

Boosting is another ensemble learning technique, but unlike Bagging, the models are trained sequentially. Each model in the ensemble is trained to correct the mistakes of its predecessor. The key idea behind Boosting is to focus on improving the performance of the models on data points that were incorrectly classified by previous trees. This process continues iteratively until we minimize the number of misclassified data points based on the selected hyperparameters (Freund & Schapire, 1997).

For part one, we used the same Decision Tree Classifier as in Bagging, but this time we set `n_estimators` to 50. Increasing the number of trees in the ensemble, allows the model to focus on improving accuracy by sequentially correcting misclassifications.

Here are the evaluation metrics for part one:

- Accuracy: 98.5%
- Recall: 98.8%

For part two, we again used the same Decision Tree as the base model, but this time with `n_estimators` set to 75. This further increased the number of trees, allowing the model to refine its predictions even more as we go from binary to multiple classes.

Here are the evaluation metrics for part two:

- Accuracy: 90.6%
- Recall: 90%



Both Bagging and Boosting are powerful ensemble methods, but they each have their advantages and disadvantages. Bagging works by training independent base models and aggregating their results, making them more robust to overfitting. Boosting, on the other hand, works by training models sequentially, allowing it to focus more on misclassified data points. Boosting is slightly more powerful than Bagging, especially when dealing with higher-dimensional data and missing values (Junfeng Jiao, 2024). Figure XI below shows the pros and cons to Boosting

Strength	Weakness
Easy to interpret	sensitive to outliers
implicit feature selection	hard to scale up (sequential)
resilient to overfitting (in a degree)	slower to train (compared to bagging)
Strong prediction power	

Figure XI (Junfeng Jiao, 2024)

## XGBoost

Extreme Gradient Boosting (XGBoost) is an advanced extension of the Gradient Boosting technique, which is itself a powerful ensemble learning method. To fully understand XGBoost, we must first delve into Gradient Boosting. Gradient Boosting combines multiple weak models to create a stronger predictive model by assigning weights to data and learning from these weighted examples (GeeksForGeeks, 2020). The core idea is that each new model aims to improve the loss of the previous one. This is achieved by minimizing a loss function. For classification common loss functions include softmax and binary cross-entropy, while for regression, common loss functions include root mean squared error (RMSE) and Huber loss.

A simple way to think of Gradient Boosting is that it continuously adjusts the models to get closer to the "ideal" prediction, essentially trying to find the point closest to the x-axis, or local minima (as shown in Figure XII below)

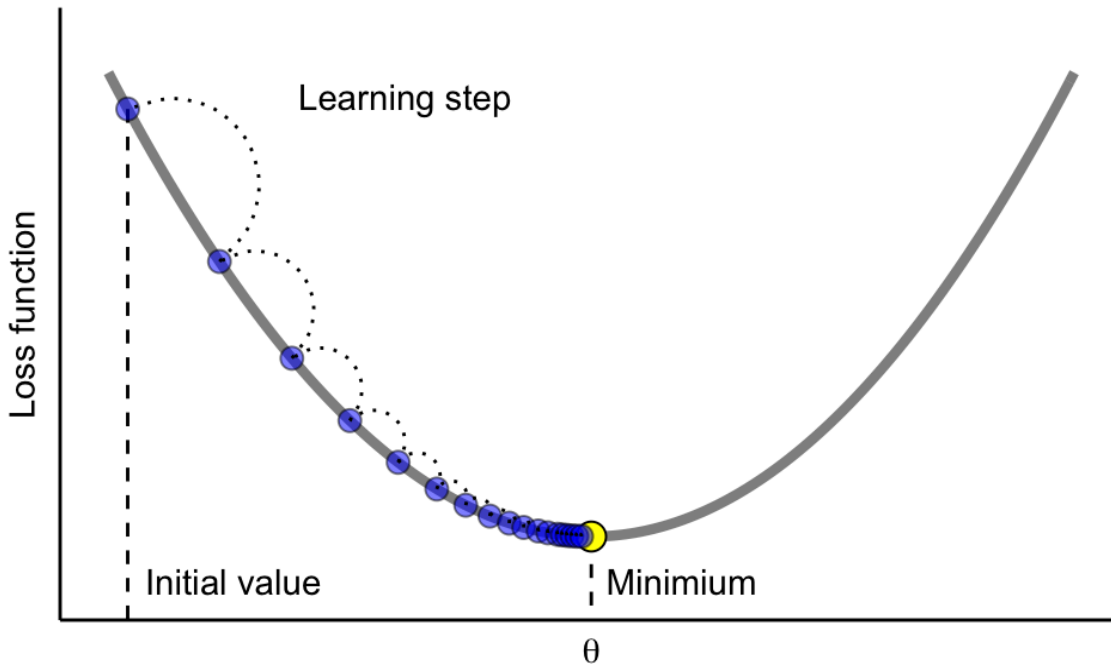


Figure XII (Boehmke, 2019)

XGBoost leverages Gradient Boosting as its foundation but improves on it by making it more efficient and scalable. XGBoost has a built-in feature to handle missing values, which makes it particularly robust for real-world data. For part one, we used the XGBoost package in Python to train a binary classification model. We set the following hyperparameters: “binary: logistic” as our objective, `n_estimators` of 75, a learning rate of 0.5 (controls the step size and learning frequency), and `max_depth` of 4. To learn more about these parameters please refer to the official documentation by XGBoost (XGBoost Python Package — Xgboost 1.0.0-SNAPSHOT Documentation, 2019).

Here are the evaluation metrics for part one:

- Accuracy: 98.3%
- Recall: 98.6%

For part two. We used “multi: softmax” as our objective to classify multiple classes, `n_estimators` of 100, learning rate of 0.5, and `max_depth` of 7.

Here are the evaluation metrics for part two:

- Accuracy: 95.3%
- Recall: 95.0%

In this section, we explored several machine-learning models and evaluated them using accuracy and recall. Among all the models, XGBoost performed the best, demonstrating its strength in both binary and multi-class classification tasks. This model excels because of its ability to handle large datasets, missing values, and improve iteratively. In the next section, we will explore additional classification metrics and conduct a more detailed comparison of this model.

## 6. Evaluation Metrics

In our model evaluations, we primarily focused on accuracy and recall, but there are two other important metrics to consider: precision and the F-1 score. To fully understand these metrics, we need to first examine the confusion matrix. The confusion matrix compares the predicted values against the actual values, helping us assess the performance of the model. Figure XIII below visualizes the matrix:

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure XIII (Narkhede, 2018)

TP = True Positive: model correctly predicts the positive class

TN: True Negative: model correctly predicts the negative class

FP = False Positive: model incorrectly predicts the positive class

FN = False Negative: model incorrectly predicts the negative class

A paper by (Luque et al., 2019) explores the full mechanism and definition of the confusion matrix along with the metrics that can be derived from it. See Figure XIV below.

<b>Metric</b>	<b>Defined as</b>
Sensitivity	$\frac{TP}{TP+FN}$
Specificity	$\frac{TN}{TN+FP}$
Precision	$\frac{TP}{TP+FP}$
Negative Predictive Value	$\frac{TN}{TN+FN}$
Accuracy	$\frac{TP+TN}{TP+FN+TN+FP}$
$F_1$ score	$2 \frac{PRC \cdot SNS}{PRC + SNS}$
Geometric Mean	$\sqrt{SNS \cdot SPC}$
Matthews Correlation Coefficient	$\frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
Bookmaker Informedness	$SNS + SPC - 1$
Markedness	$PPV + NPV - 1$

Figure XIV (Luque et al., 2019)

Accuracy is a general measure of the model's correctness. It is calculated by dividing the sum of True Positives (TP) and True Negatives (TN) by the total number of predictions made (TP + TN + FP + FN). This gives an overall percentage of correct predictions.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision is the proportion of correctly predicted positive instances out of all predicted positive instances. In other words, it tells us how many of the instances classified as positive by

the model are positive. Precision is particularly important when the cost of false positives is high, such as in medical diagnosis.

$$\text{Precision} = \text{TP} / \text{TP} + \text{FP}$$

Recall, also known as sensitivity, measures the proportion of actual positive instances that were correctly identified by the model. Recall is crucial when we want to minimize false negatives, ensuring that no true positive cases are missed, especially in scenarios like failure prediction, where missing a failure could lead to costly consequences.

$$\text{Recall} = \text{TP} / \text{TP} + \text{FN}$$

The f-1 score is the harmonic mean combining precision and recall. It is a balance between the two. Ideally the higher the F-1 score the better because that means your model is not skewed and there is a good balance between precision and recall.

$$\text{F-1 score} = 2 * ( [\text{precision} \times \text{recall}] / [\text{precision} + \text{recall}] )$$

Table I below shows the performance metrics for the models used to predict machine failure in part one. After analyzing the results, it is evident that the AdaBoost classifier and XGBoost classifier are the top performers, achieving the highest recall values among all models with strong accuracy. This allows them to correctly predict true positives and minimize false negatives, which is critical for our case.

An important observation in Table I is the significant difference in recall between Logistic Regression with raw data and Logistic Regression with SMOTE and threshold. We can see that applying proper data augmentation (SMOTE) and adjusting the threshold (0.3) leads to a substantial improvement in performance. This highlights the importance of data preparation: how you augment and preprocess your data can have a profound impact on the model's performance.

Model Name	Accuracy	Precision	Recall	F-1 Score
Logistic Regression raw	86.70%	87.60%	85.70%	86.70%
Logistic regression with SMOTE and threshold	85.40%	80.10%	94.60%	86.80%
Decision Tree Classifier	93.70%	91.60%	95.10%	93.30%
Bagging Classifier	96.80%	96.60%	97.10%	96.80%
AdaBoostClassifier	98.50%	98.30%	98.70%	98.50%
XGBoost	98.20%	98.00%	98.60%	98.30%

Table I. Part One Results

Table II shows the results for all models used in part two. XGBoost emerges as the clear leader, performing exceptionally well across all metrics, and achieving the highest values for all metrics. XGBoost's strength lies in its ability to model complex relationships and learn patterns within the data.

Model Name	Accuracy	Precision	Recall	F-1 Score
XGBoost	95.20%	95.20%	95.20%	95.20%
Decision Tree Classifier	90.50%	91.50%	90.50%	90.60%
AdaBoostClassifier	90.50%	90.80%	90.50%	90.50%
Bagging Classifier	92.90%	92.80%	91.40%	92.50%

Table II. Part two results

Since XGBoost performed well for both our problem statements let's explore a little deeper into this model. Focusing on the second part of our problem, we will look at the macro average, cross-validation results, and Feature importance analysis to see which features were the highest contributing to inference.

The macro average approach confers all the classes as basic elements where each class has the same weight in the average so there is no difference between highly and poorly populated classes (Grandini et al., 2020). This is an important metric when you have high cardinality

features with some class imbalance. Below figure XV consists of the formulas for macro precision, macro recall, and macro F-1 score.

$$MacroAveragePrecision = \frac{\sum_{k=1}^K Precision_k}{K}$$

$$MacroAverageRecall = \frac{\sum_{k=1}^K Recall_k}{K}$$

Eventually, Macro F1-Score is the harmonic mean of Macro-Precision and Macro-Recall:

$$Macro\ F1-Score = 2 * \left( \frac{MacroAveragePrecision * MacroAverageRecall}{MacroAveragePrecision^{-1} + MacroAverageRecall^{-1}} \right)$$

Figure XV (Grandini et al., 2020)

Table III displays the macro average values for the XGBoost model's performance, which reflects the model's ability to treat all classes equally, regardless of their frequency. The fact that SMOTE was applied to balance the classes likely contributes to the model's balanced performance across all classes. Here are the values from:

Metric	Precision	Recall	F-1 score
Macro_avg with SMOTE	0.96	0.96	0.96
Macro_avg on raw data	0.83	0.92	0.89

Table III

Up to this point, we simply performed training and testing on one single split of the data set. Stratified K-fold cross-validation addresses this challenge by partitioning the dataset into K-folds while ensuring that each fold(split) maintains the same class distribution as the original dataset (T r et al., 2023). This helps in ensuring there is no imbalance and bias in each fold of the dataset. For our XGBoost model, we performed a stratified 5-fold cross-validation to evaluate the accuracy of each fold. The results are as follows:

Individual fold scores: [0.97, 1.0, 0.90, 0.94, 0.94]

Additionally, we also looked at the mean and standard deviation for these results. We achieved a mean accuracy of 95.5% and a standard deviation of 0.06 telling us that there was very little variance across the accuracy for all folds and that our model will perform very well on new unseen data.

Feature importance analysis provides valuable insights into which features in the dataset have the most significant impact on the predictions. We performed a feature importance analysis on the XGBoost model. To learn more about how this works please refer to the official documentation (Python API Reference — Xgboost 2.1.2 Documentation, 2024). Figure XVI shows the feature importance for our model. Torque, temp\_diff, tools wear, and power have significant importance, telling us that any change in values of these features highly influenced the predictions. It is important to note that temp\_diff and power were features created during feature engineering. Properly engineered features can significantly boost the performance of the models. In this case, creating features like temp\_diff and power helped the model identify key patterns that might not have been available from the raw data alone.



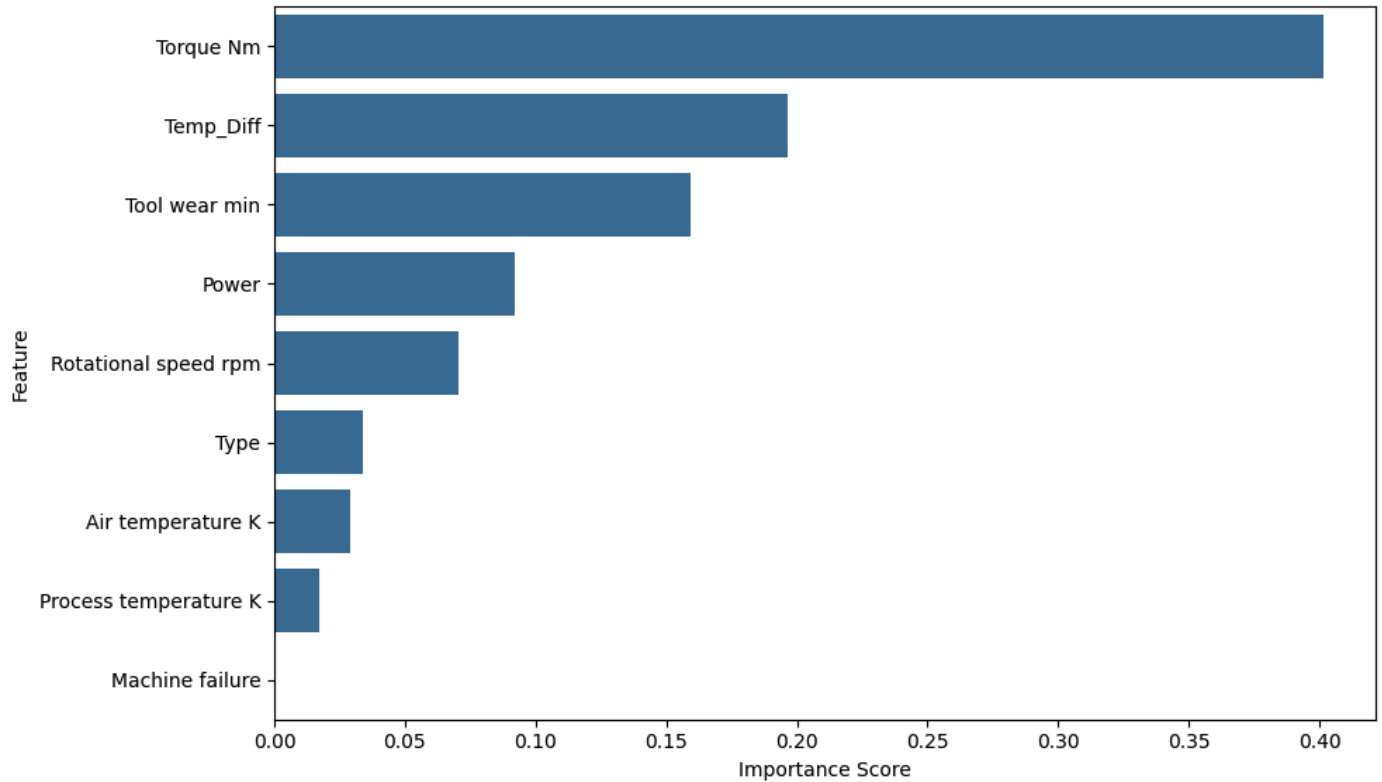


Figure XVI

## 7. Conclusion

In this paper, we have explored and applied various classification models to predict machine failures and determine the reasons. The goal was to identify the most effective machine learning approach for this problem, which has significant real-world implications for predictive maintenance and operational efficiency.

Throughout the analysis, XGBoost consistently outperformed other models, demonstrating its ability to capture complex relationships within the data. Its use of gradient boosting and decision trees allows the model to learn from iterative improvements, making it highly efficient.

Additionally, this paper emphasized the importance of feature engineering in improving model performance. We observed that the right features—such as torque, temperature difference, power, and tool wear—significantly influenced the model's predictions. These features provided insights into the physical conditions of the machine. Torque, for instance, measures the force applied on a component, while temp\_diff highlights heat deviations that can lead to failure. Class imbalance in the dataset was addressed using techniques like SMOTE (Synthetic Minority Over-sampling Technique). By balancing the classes, we improved the recall and overall performance of our models, ensuring that failure instances—the minority class—were captured.

In summary, XGBoost proved to be the most effective model for this problem, with feature engineering and class balancing techniques playing key roles in improving its performance. Understanding important features like torque, temp\_diff, and tool wear can directly influence the predictions, offering valuable insights for machine maintenance strategies. As machine learning continues to play a pivotal role in predictive maintenance, the findings from this study contribute valuable knowledge for improving both model performance and operational practices.

## **8. Future Works**

While this study has provided valuable insights into the application of classification models for machine failure prediction, there are still additional techniques that could improve current performance.

One of the limitations we saw during this study was the small number of samples. In real-world cases, machine failure is rare and occasional. Gathering additional data can significantly improve the model's ability to generalize unseen information.

In terms of feature engineering, as discussed earlier we use label encoding to convert categorical features into numerical ones. However, label encoding introduces an ordinal relationship (i.e.,  $2 > 1 > 0$ ), which could potentially mislead the model. An alternative method that we might explore is One-Hot Encoding, where each category is converted into a separate

binary feature (0 or 1). This method avoids implying an ordinal relationship between categories. Additional information can be found in their documentation (Scikit-learn, 2019).

We can also use the Adaptive Synthetic Minority Oversampling Technique (ADASYN) instead of SMOTE to over-sample our minority class. This method is similar to SMOTE but it generates samples by taking into account the local distribution of data points (ADASYN — Version 0.9.1, n.d.).

Another method to explore is Principal Component Analysis (PCA). “PCA is a multivariate statistical technique that analyzes data where observations are described by inter-correlated quantitative dependent variables. PCA aims to extract the most important information and represent them as a set of new orthogonal variables” (Abdi & Williams, 2010). We can use PCA to determine the most important values and only retain them for our analysis. This technique is useful in high-dimensional data to remove unnecessary noise and focus on content that matters.

## Works Cited

1. Carvalho, T. P., Soares, F. A. A. M. N., Vita, R., Francisco, R. da P., Basto, J. P., & Alcalá, S. G. S. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137(1), 106024. <https://doi.org/10.1016/j.cie.2019.106024>
2. Bousdekis, A., Lepenioti, K., Apostolou, D., & Mentzas, G. (2019). Decision Making in Predictive Maintenance: Literature Review and Research Agenda for Industry 4.0. *IFAC-PapersOnLine*, 52(13), 607–612. <https://doi.org/10.1016/j.ifacol.2019.11.226>
3. Spiegel, S., Mueller, F., Wiesmann, D., & Bird, J. (2018, September 28). Cost-Sensitive Learning for Predictive Maintenance. ResearchGate; unknown. [https://www.researchgate.net/publication/327982563\\_Cost-Sensitive\\_Learning\\_for\\_Predictive\\_Maintenance](https://www.researchgate.net/publication/327982563_Cost-Sensitive_Learning_for_Predictive_Maintenance)
4. McKinsey & Company. (2022, August 17). What Is Industry 4.0 and the Fourth Industrial Revolution? McKinsey & Company. <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-are-industry-4-0-the-fourth-industrial-revolution-and-4ir>
5. IBM. (2022, August 8). Predictive Analytics. Ibm.com. [https://ibm.com/topics/predictive-analytics?utm\\_content=SRCWW&p1=Search&p4=43700075153304567&p5=p&p9=58700008227853819&gad\\_source=1&gclid=Cj0KCQiAoa\\_e5BhCNARIsADVLzZfLzbyUe2GmDiGBCYE123vsHowIdRLJObLqHCSbpYf0QKcfuHr\\_YcoaAlfAEALw\\_wcB&gclid=aw.ds](https://ibm.com/topics/predictive-analytics?utm_content=SRCWW&p1=Search&p4=43700075153304567&p5=p&p9=58700008227853819&gad_source=1&gclid=Cj0KCQiAoa_e5BhCNARIsADVLzZfLzbyUe2GmDiGBCYE123vsHowIdRLJObLqHCSbpYf0QKcfuHr_YcoaAlfAEALw_wcB&gclid=aw.ds)
6. Lin, C.-Y., Hsieh, Y.-M., Cheng, F.-T., Huang, H.-C., & Adnan, M. (2019). Time Series Prediction Algorithm for Intelligent Predictive Maintenance. *IEEE Robotics and Automation Letters*, 4(3), 2807–2814. <https://doi.org/10.1109/lra.2019.2918684>
7. Chazhoor, A., Mounika, Y., Vergin Raja Sarobin, M., Sanjana, M. V., & Yasashvini, R. (2020). Predictive Maintenance using Machine Learning Based Classification Models. *IOP Conference Series: Materials Science and Engineering*, 954, 012001. <https://doi.org/10.1088/1757-899x/954/1/012001>

8. Matzka, S. (2020). Explainable Artificial Intelligence for Predictive Maintenance Applications. 2020 Third International Conference on Artificial Intelligence for Industries (AI4I). <https://doi.org/10.1109/ai4i49448.2020.00023>
9. UCI Machine Learning Repository. (n.d.). Archive.ics.uci.edu.  
<https://archive.ics.uci.edu/dataset/601/ai4i+2020+predictive+maintenance+dataset>
10. Blagus, R., & Lusa, L. (2013). SMOTE for high-dimensional class-imbalanced data. BMC Bioinformatics, 14(1). <https://doi.org/10.1186/1471-2105-14-106>
11. The Importance of Feature Engineering in Machine Learning. (2024, April 3). Online Degree Programs | Milwaukee School of Engineering; Milwaukee School of Engineering.  
<https://online.msoe.edu/engineering/blog/importance-of-feature-engineering-in-machine-learning>
12. Kanade, V. (2022, April 18). What Is Logistic Regression? Equation, Assumptions, Types, and Best Practices. Spiceworks.  
<https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/>
13. MLMath.io. (2019, February 21). Math behind Decision Tree Algorithm. Medium.  
<https://ankitnitjsr13.medium.com/math-behind-decision-tree-algorithm-2aa398561d6d>
14. DecisionTreeClassifier. (2024). Scikit-Learn.  
<https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
15. Breiman, L. (1996). Bagging Predictors. Machine Learning, 24(2), 123–140.  
<https://doi.org/10.1023/a:1018054314350>
16. Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. Multiple Classifier Systems, 1857, 1–15. [https://doi.org/10.1007/3-540-45014-9\\_1](https://doi.org/10.1007/3-540-45014-9_1)
17. Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences, 55(1), 119–139. <https://doi.org/10.1006/jcss.1997.1504>
18. Jiao, J. (2024). Machine Learning and its Applications: Supervised Learning: Classification & KNN [PowerPoint slides]. University of Texas at Austin. Slide 50.  
[https://utexas.instructure.com/courses/1404604/pages/module-2-slides-and-lab-sources?module\\_item\\_id=14044647](https://utexas.instructure.com/courses/1404604/pages/module-2-slides-and-lab-sources?module_item_id=14044647)

19. GeeksForGeeks. (2020, August 25). ML - Gradient Boosting. GeeksforGeeks.  
<https://www.geeksforgeeks.org/ml-gradient-boosting/>
20. Boehmke, B. (2019, October 19). Chapter 12 Gradient Boosting | Hands-On Machine Learning with R. Github.io. <https://bradleyboehmke.github.io/HOML/gbm.html>
21. XGBoost Python Package — xgboost 1.0.0-SNAPSHOT documentation. (2019). Readthedocs.io. <https://xgboost.readthedocs.io/en/latest/python/index.html>
22. Narkhede, S. (2018, May 9). Understanding Confusion Matrix. Medium; Towards Data Science. <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>
23. Luque, A., Carrasco, A., Martín, A., & de las Heras, A. (2019). The impact of class imbalance in classification performance metrics based on the binary confusion matrix. Pattern Recognition, 91, 216–231. <https://doi.org/10.1016/j.patcog.2019.02.023>
24. Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for Multi-Class Classification: an Overview. ArXiv.org. <https://arxiv.org/abs/2008.05756v1>
25. Tr, M., V, V. K., V, D. K., Geman, O., Margala, M., & Guduri, M. (2023). The stratified K-folds cross-validation and class-balancing methods with high-performance ensemble classifiers for breast cancer classification. Healthcare Analytics, 4, 100247.  
<https://doi.org/10.1016/j.health.2023.100247>
26. Python API Reference — xgboost 2.1.2 documentation. (2024). Readthedocs.io.  
[https://xgboost.readthedocs.io/en/latest/python/python\\_api.html#xgboost.XGBClassifier.feature\\_importances](https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.XGBClassifier.feature_importances)
27. Abdi, H., & Williams, L. J. (2010). Principal component analysis. Wiley Interdisciplinary Reviews: Computational Statistics, 2(4), 433–459. <https://doi.org/10.1002/wics.101>
28. Scikit-learn. (2019). sklearn.preprocessing.OneHotEncoder — scikit-learn 0.22 documentation. Scikit-Learn.org.  
<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html>
29. Uhlmann, E., Pontes, R. P., Geisert, C., & Hohwieler, E. (2018). Cluster identification of sensor data for predictive maintenance in a Selective Laser Melting machine tool. Procedia Manufacturing, 24, 60–65. <https://doi.org/10.1016/j.promfg.2018.06.009>

30. Team, G. L. (2020, August 22). What is Label Encoding in Python | Great Learning.  
GreatLearning Blog: Free Resources What Matters to Shape Your Career!  
<https://www.mygreatlearning.com/blog/label-encoding-in-python/>
31. ADASYN — Version 0.9.1. (n.d.). Imbalanced-Learn.org.  
[https://imbalanced-learn.org/stable/references/generated/imblearn.over\\_sampling.ADASYN.html](https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html)