Ministry of Communications and Information Technology

Digital Egypt Pioneers

Data Preprocessing Report

Names:

Mohamed Ahmed Rabea

Muhammad Usama Aowad

Useif Muhammed Saad Ebrahim

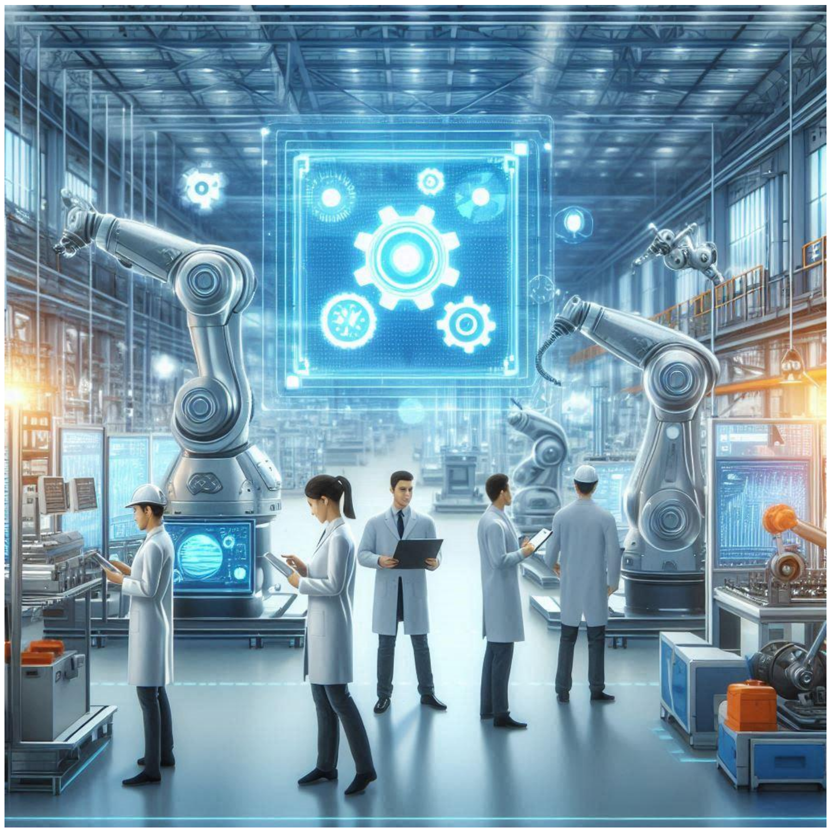
Ahmed Abdelsalam Abdelmaqsoud

Content :

Overview of the Predictive Maintenance Project

This project aims to create an AI-powered predictive maintenance system to anticipate machine failures before they occur. By leveraging advanced machine learning techniques, the system will predict potential failures in real-time, enabling proactive maintenance interventions. This approach helps to reduce machine downtime, optimize maintenance schedules, and lower operational costs.

The system will analyze various operational parameters, such as temperature, rotational speed, and torque, to determine when a machine is likely to fail. When a potential failure is detected, maintenance alerts will be generated, allowing for timely intervention. This will extend equipment life and ensure uninterrupted production processes, making the system a valuable tool for industries relying on complex machinery.



Data Set

The AI4I 2020 Predictive Maintenance Dataset, sourced from the UCI Data Repository, is used for this project. It contains synthetic data representing machine operating conditions and failure instances across various failure modes. The dataset includes features such as air temperature, process temperature, rotational speed, torque, and tool wear, which are crucial in predicting machine failures.

Machine failures are recorded in five distinct modes:

- **Tool Wear Failure (TWF):** Triggered by tool wear after a certain operational duration. In the dataset, the tool is replaced 69 times and fails 51 times.
- **Heat Dissipation Failure (HDF):** Occurs when the difference between air and process temperature is below a specific threshold, combined with low rotational speed, affecting 115 instances.
- **Power Failure (PWF):** Defined by power outside the acceptable range, calculated as the product of torque and rotational speed. This mode is observed in 95 instances.
- **Overstrain Failure (OSF):** Happens when the product of tool wear and torque exceeds defined thresholds for different product variants, affecting 98 data points.
- **Random Failure (RNF):** A random failure mode that occurs with a low probability, affecting 5 instances.

If any one of these failure modes occurs, the process is labeled as a failure, marked with a 'machine failure' indicator. This label is used as the target variable for predictive models.

Data Exploration

The initial exploration of the dataset involved examining summary statistics and checking for data quality issues such as missing values and outliers.

Summary Statistics

We generated descriptive statistics for the dataset using the **.describe()** method. This provided insights into the range, mean, and standard deviation of each feature. Specifically, we focused on analyzing the distribution of **rotational speed** across different thresholds. Upon further exploration, we found that data points with rotational speeds above 2000 rpm did not exhibit significant outliers, confirming that the dataset is well-organized and free from major anomalies.

Additionally, we confirmed that there are **no missing values** in the dataset, making it ready for further analysis and modeling without the need for imputation or data cleaning.

Class Imbalance

Class imbalance is a common challenge in predictive maintenance, as machine failures are typically less frequent than successful operations. To better understand the distribution of the target variable, **machine failure**, we performed the following steps:

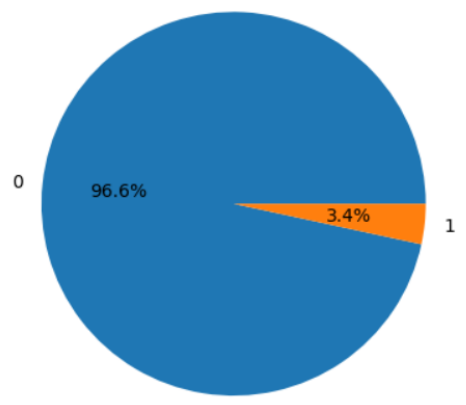
- **Class Distribution:** We first analyzed how often machine failures occur by checking the counts for **both failure (machine_failure = 1) and success (machine_failure = 0) classes**. A **pie chart** was plotted to visually illustrate the distribution. The dataset shows a significant class imbalance, with far fewer failures compared to successful operations.
- **Implications of Class Imbalance:** Class imbalance poses a challenge for machine learning models, as they may struggle to learn meaningful patterns from the minority class (machine failures). This can lead to biased predictions toward the majority class, which, in this case, is machine success.

Handling Class Imbalance

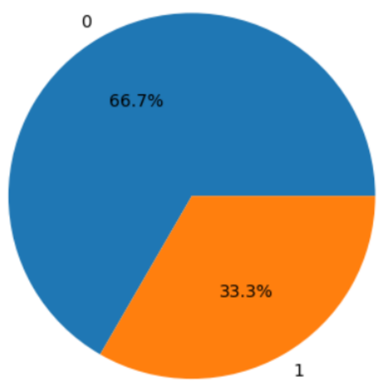
To address the imbalance, we opted for **up-sampling** the minority class (machine failure). This technique involves replicating samples from the minority class to increase its representation in the dataset. Specifically, we performed the following steps:

- **Up-sampling:** We up-sampled the machine failure class (machine_failure = 1) to have half the number of samples compared to the majority class. This ensures a more balanced dataset while still reflecting the real-world imbalance to some extent.
- **Dataset Merge:** After up-sampling, we merged the newly up-sampled failure data with the success class data to create a more balanced dataset for training machine learning models. The final shape of the up-sampled dataset was then verified.

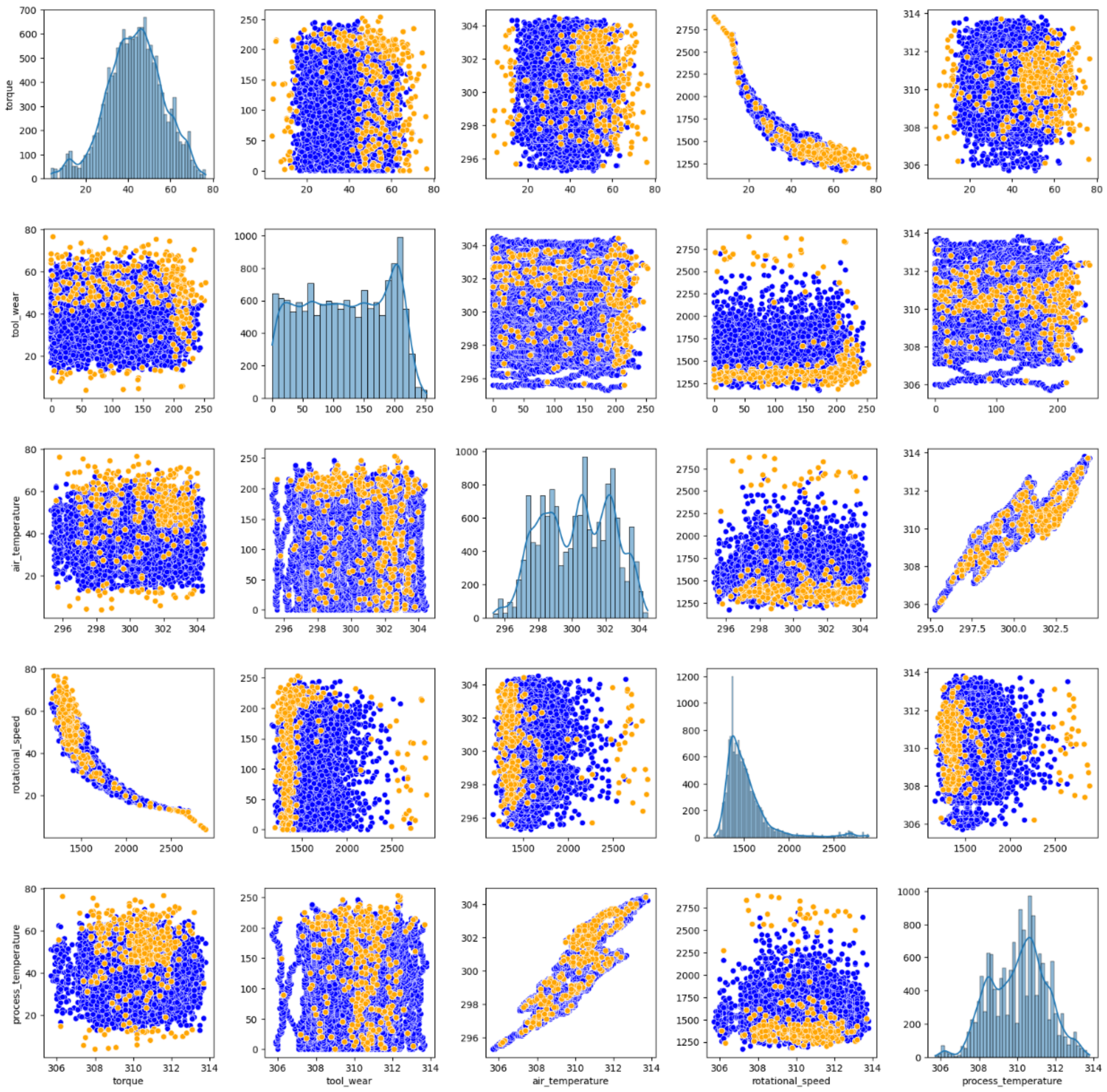
Before



After



Feature Analysis



Feature Pairwise Relationships & Failure Impact Analysis

Visualizing Feature Interactions:

The plot above represents pairwise comparisons of key operational features including air temperature, process temperature, rotational speed, torque, and tool wear, color-coded by the presence or absence of machine failure. Each subplot reveals interactions between two variables while highlighting **machine failures (in orange) and non-failures (in blue).**

Key Observations from the Pairwise Plots:

- **Torque vs. Rotational Speed:** A clear nonlinear relationship is visible, with torque values dropping significantly as rotational speed increases. Failures (orange) tend to cluster at lower torque and higher rotational speed, hinting at overstrain failures.
- **Tool Wear:** There is no obvious linear relationship between tool wear and other features like air temperature or torque. However, failures are scattered across various levels of tool wear, suggesting that this feature alone may not be a direct indicator but might contribute to failures when combined with other factors.
- **Air Temperature vs. Process Temperature:** These two features show a strong linear correlation, as expected in machine environments. The failures are relatively evenly distributed across different temperatures, indicating that failure modes linked to temperature (like Heat Dissipation Failures) are rare or not strongly correlated.
- **Rotational Speed:** High rotational speeds (above 2000 rpm) are generally associated with non-failures, but a cluster of machine failures is observed at specific ranges of torque and rotational speed.

Failure Patterns:

The scatterplots help identify regions where machine failures are more frequent. Specifically:

- Failures appear concentrated where **torque is low**, especially in combination with **high rotational speed**. This might suggest that machines are more prone to overstrain or power-related failures under these conditions.
- **Process temperature and air temperature** tend to vary slightly with failures, but their effect seems minimal in isolation. However, they may have indirect effects in combination with other features.

Feature Selection

Feature selection is an important step in ensuring that only the most relevant features are included in the predictive model. By focusing on the features that have a strong correlation with the target label, we can improve the model's performance and interpretability.

Feature Correlations

To identify the most relevant features for predicting machine failures, we explored the correlations between the various features and the target label, **machine failure**. We used a **correlation heatmap** to visualize the relationships between the features. From the heatmap, we observed a certain degree of correlation between the failure modes (TWF, HDF, PWF, OSF) and system readings such as temperature, torque, and rotational speed. This is expected since the failure modes are derived from these underlying system measurements.

Identifying Strongly Correlated Features

Next, we generated an ordered list of features that show the strongest correlation with the failure label. By sorting the correlation values and focusing on those with the highest absolute correlation, we identified the following features as having the strongest relationship with machine failure:

- **TWF (Tool Wear Failure)**
- **HDF (Heat Dissipation Failure)**
- **PWF (Power Failure)**
- **OSF (Overstrain Failure)**
- **RNF (Random Failure)**

However, since these failure mode features are derived from system readings and would not be available in real-time during inference, we decided to **remove these features**. This ensures that the model learns to predict failures based only on the core system measurements that are available during the operation of the machines.

Removing Derived Features

We removed the failure mode features (TWF, HDF, PWF, OSF, and RNF) from the dataset, leaving only the foundational system features:

- **Torque**
- **Tool wear**
- **Air temperature**
- **Rotational speed**
- **Process temperature**
- **Type (encoded)**

These remaining features will be used as the input for training the predictive model. The dataset was also cleaned by replacing any missing values (NA) with 0 and ensuring all features are in numerical format.

Final Dataset

After the removal of the failure mode features, we rearranged the columns so that the target label, **machine failure**, is the last column in the dataset. This prepares the data for input into machine learning models.

The final shape of the processed dataset is:

- **Shape:** (Number of rows, Number of columns)

The processed dataset is now ready for the next phase of the project: model training.

Data Split & Export

To prepare the dataset for model training, we performed a **train-test split** to separate the data into training and testing sets. This ensures that the model can be evaluated on unseen data, giving a reliable estimate of its performance.

Splitting the Data

We split the processed dataset into training and test sets using an 80-20 split, meaning that 80% of the data is used for training, and 20% is reserved for testing. The split is performed randomly using a fixed random_state to ensure reproducibility of the results.

The final split resulted in the following:

- **Training Data Shape:** (Number of rows, Number of columns)
- **Test Data Shape:** (Number of rows, Number of columns)

Exporting the Data

After splitting the data, both the training and test sets were exported as CSV files to local directories for use in model training and evaluation. The files were saved as follows:

- **Training Data:** Saved as training_data/train.csv
- **Test Data:** Saved as test_data/test.csv
- **Full Dataset:** The entire processed dataset was also exported as fulldataset.csv.

These files will be used in the subsequent phases of model development and testing.

Model Training & Evaluation

First, we got training and test data into variables

We applied standard data preprocessing techniques, including:

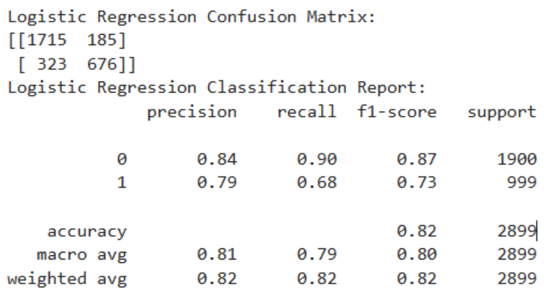
- **Scaling**: We used **MinMaxScaler** to normalize the feature space, ensuring that all features contributed equally to the model.

Model Selection and Training

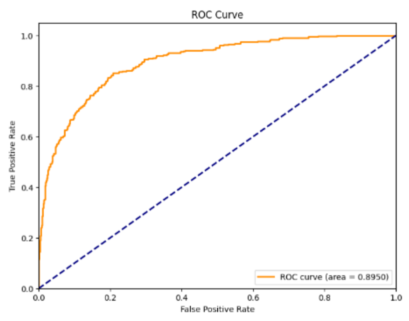
We implemented an ensemble learning method at the end, a **Voting Classifier**, which combined multiple models to improve predictive performance. The following base models were used in the voting classifier:

- Logistic Regression
- Decision Tree
- Random Forest Classifier
- Gradient Boosting Classifier
- XGBoosting

For the Logistic Regression we got an accuracy of **82.4%**

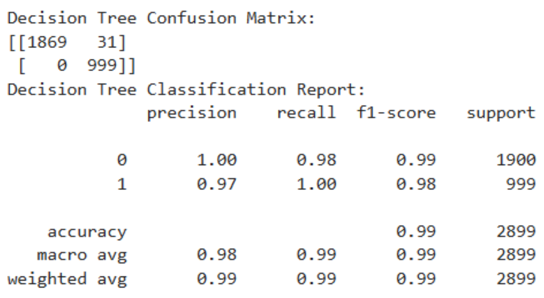


```
Logistic Regression Confusion Matrix:
[[1715  185]
 [ 323  676]]
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1900
           1       0.79      0.68      0.73       999

    accuracy                           0.82      2899
   macro avg       0.81      0.79      0.80      2899
weighted avg       0.82      0.82      0.82      2899
```

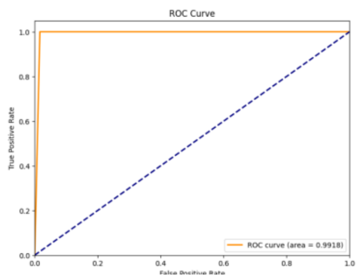


For the Decision Tree we got an accuracy of **98.9%**

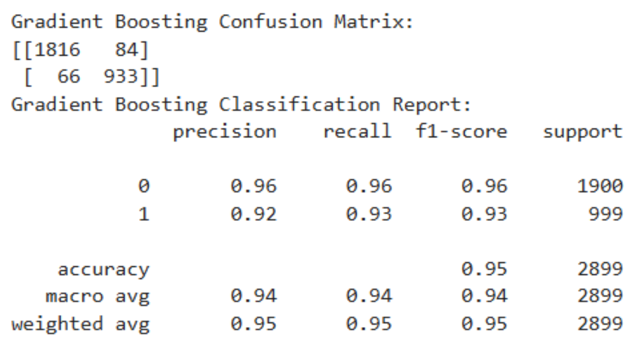
Most likely this model suffers from **overfitting** because of how Decision Tree deals with such data



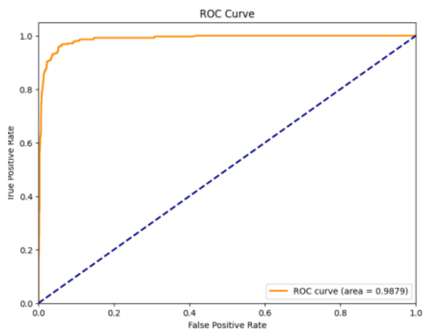
```
Decision Tree Confusion Matrix:
[[1869   31]
 [   0  999]]
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1900
           1       0.97      1.00      0.98       999

    accuracy                           0.99      2899
   macro avg       0.98      0.99      0.99      2899
weighted avg       0.99      0.99      0.99      2899
```



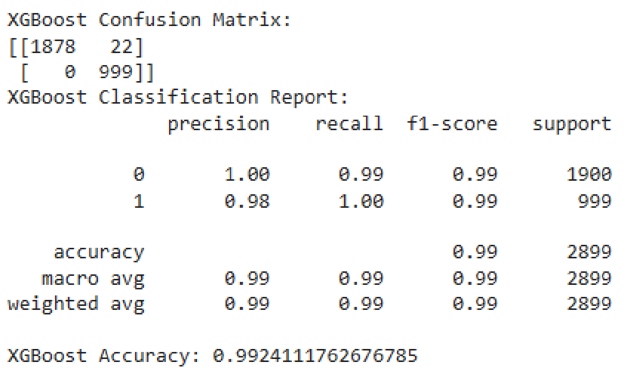
For the Gradient Boosting we got an accuracy of **94.8%**



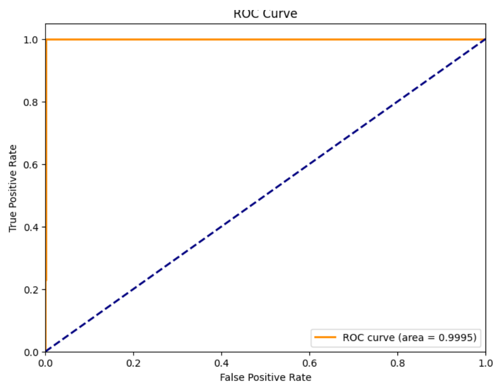
```
Gradient Boosting Confusion Matrix:
[[1816   84]
 [  66  933]]
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      1900
           1       0.92      0.93      0.93       999

    accuracy                           0.95      2899
   macro avg       0.94      0.94      0.94      2899
weighted avg       0.95      0.95      0.95      2899
```



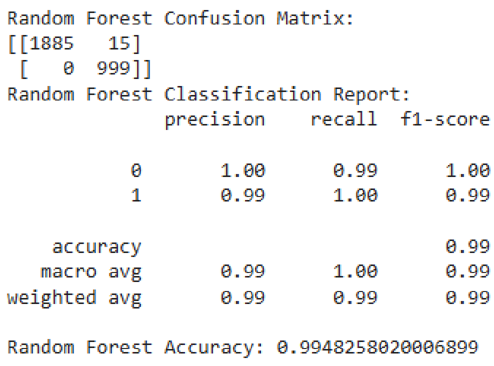
For the XG Boosting we got an accuracy of **99.2%**



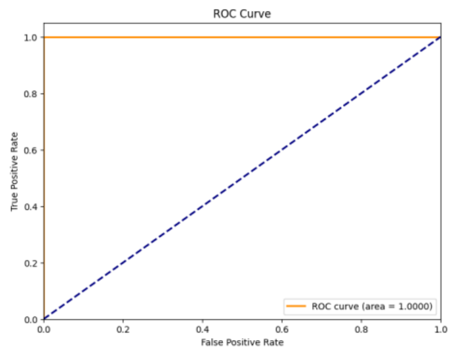
```
XGBoost Confusion Matrix:
[[1878   22]
 [   0  999]]
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1900
           1       0.98      1.00      0.99       999

    accuracy                           0.99      2899
   macro avg       0.99      0.99      0.99      2899
weighted avg       0.99      0.99      0.99      2899

XGBoost Accuracy: 0.9924111762676785
```



For the Random Forest we got an accuracy of **99.4%**

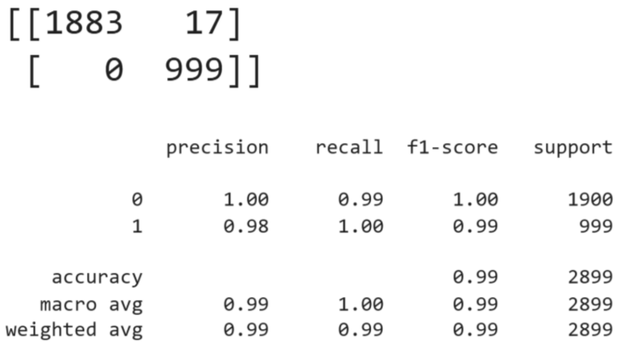


```
Random Forest Confusion Matrix:
[[1885   15]
 [   0  999]]
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1900
           1       0.99      1.00      0.99       999

    accuracy                           0.99      2899
   macro avg       0.99      1.00      0.99      2899
weighted avg       0.99      0.99      0.99      2899

Random Forest Accuracy: 0.9948258020006899
```

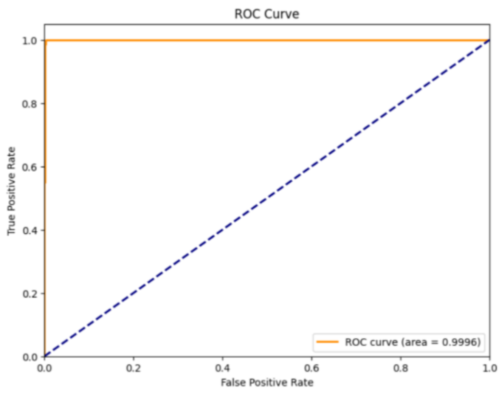


After getting the evaluation of these models we decided to make an ensample model to combine 5 of them so that we can benefit from the voting technique it uses and also avoid overfitting as much as possible and maintain the high accuracy

After making the voting classifier we got an accuracy of **99.4%**



```
[[1883   17]
 [   0  999]]

              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1900
           1       0.98      1.00      0.99       999

    accuracy                           0.99      2899
   macro avg       0.99      1.00      0.99      2899
weighted avg       0.99      0.99      0.99      2899
```



Advantages of using ensample technique:

1- **Improved Robustness**:

- A Voting Classifier combines the strengths of multiple models. Even if one model performs very well on most of the data. The other models can compensate for these weaknesses. This creates a more robust and stable model overall, less prone to overfitting or underfitting specific patterns in the data.

2- **Error Reduction**:

- By averaging out the predictions of multiple models (in case of soft voting), the ensemble can reduce errors caused by individual model inconsistencies or incorrect predictions. This can lead to a lower overall error rate compared to relying solely on a single model.

3- **Generalization to New Data**:

- A single model with high accuracy may excel on a particular dataset but fail to generalize to unseen data. The combination of different models in a Voting Classifier tends to generalize better across different types of data, as each model brings a unique perspective to the prediction task.

4- **Bias-Variance Trade-off**:

- Different models have different biases and variances. A Voting Classifier can balance this trade-off by leveraging the diversity among the models, resulting in a lower overall error than any single model alone.

Exporting the model

By using joblib library we used it to extract the scaler model (function) so that we can pass any new values to it and normalize it just as we did for the train and test data

And then we extract the voting model itself, so that we can use it in the deployment phase.

Model Deployment

After developing and evaluating the machine learning model, the next step was to deploy it in a web application, allowing users to interact with the model in real-time. This section outlines the deployment process.

Flask Web Application

We used **Flask**, a lightweight Python web framework, to create the web interface and manage requests to the machine learning model. The application consists of the following components:

- **Frontend (HTML)**: The user interface is built using basic HTML and CSS, providing input fields where users can submit feature values such as air temperature, rotational speed, torque, tool wear, process temperature, and type (encoded as 0 for low, 1 for medium, and 2 for high).
- **Backend (Flask and Model Integration)**:
    - When a user submits the input data, the Flask application captures the data via a POST request and processes it in the backend.
    - The input data is first scaled using the pre-trained MinMaxScaler to ensure consistency with the data used during training.
    - The pre-trained **Voting Classifier** model is then loaded and used to make a prediction on whether the machine will fail or not.
    - Based on the prediction result, the application returns either "The machine will FAIL" or "The machine will NOT FAIL" to the user.

Key Flask Code Components

- **Prediction Endpoint**: The /predict route processes the input data, applies the trained model, and returns the prediction to the user.
- **Model and Scaler Loading**: Both the trained voting classifier and the data scaler were loaded using **joblib**, allowing for easy integration of the pre-trained model with the web application.

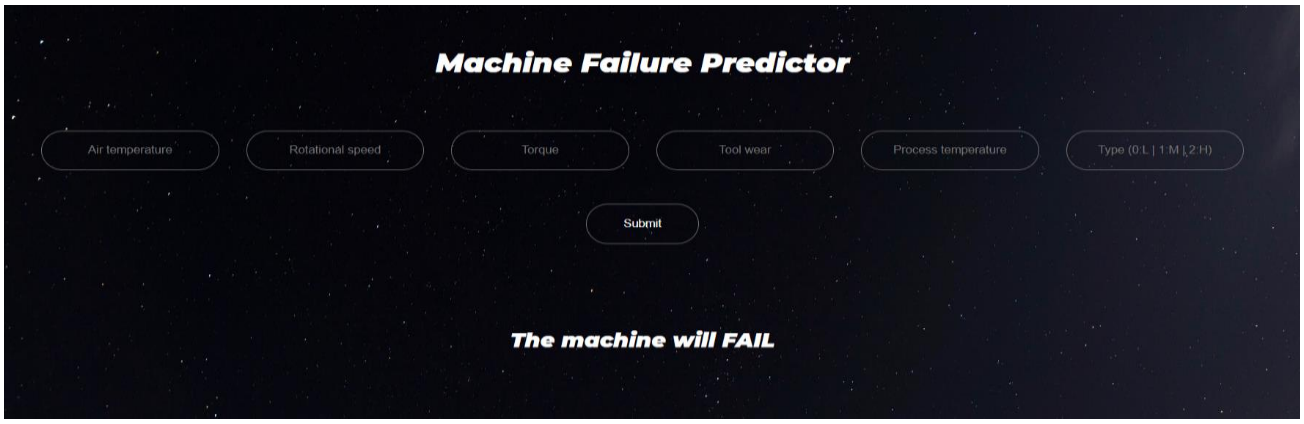
User Interface

The user interface allows easy interaction with the model. As shown in the image below, the web application contains input fields for the necessary machine parameters and provides immediate feedback on whether the machine is predicted to fail or not after the user submits the data.

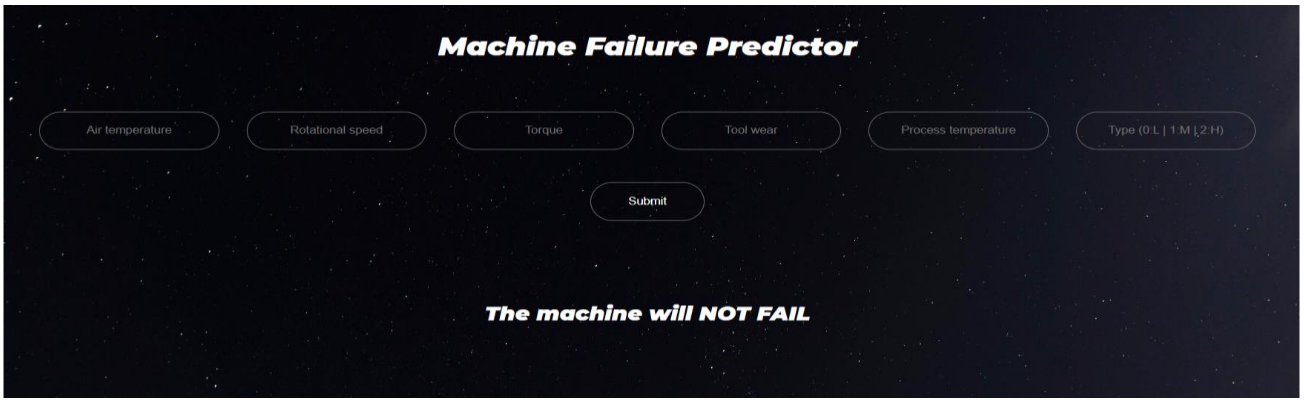
Running the Web Application

To run the web application, the following steps were taken:

- The application was developed and tested locally using the Flask framework.
- Users can interact with the application by entering the required inputs and clicking the **Submit** button. The prediction result is displayed on the screen after processing the input data.



The machine will FAIL



The machine will NOT FAIL