École Centrale Casablanca

Rapport

Rapport du Projet Intelligence Artificielle et applications en industrie 4.0

Élèves :
Ayyoub BENRGUIG
Walid HIROUCHE

Encadrants :
Sabeur Elkosantini

25 mars 2024

Table des matières

1 Introduction

1.1 La contextualisation du problème

Dans le paysage industriel contemporain, la maintenance préventive revêt une importance cruciale pour assurer une efficacité opérationnelle maximale et minimiser les temps d'arrêt coûteux. Dans ce contexte, notre étude se concentre sur l'application de techniques d'intelligence artificielle à la maintenance préventive, en mettant particulièrement l'accent sur la machine de fraisage. Notre objectif principal est de développer un modèle prédictif capable d'anticiper les défaillances de la machine de fraisage, ce qui permettra de réduire les temps d'arrêt et les coûts de maintenance. De plus, nous explorerons la prédiction de la probabilité de défaillance du processus sous diverses conditions de traitement et niveaux de qualité de l'équipement. En combinant ces approches, notre étude vise à démontrer l'efficacité de la maintenance préventive dans un environnement industriel moderne, en fournissant des outils et des insights précieux pour améliorer la fiabilité et la rentabilité des opérations.

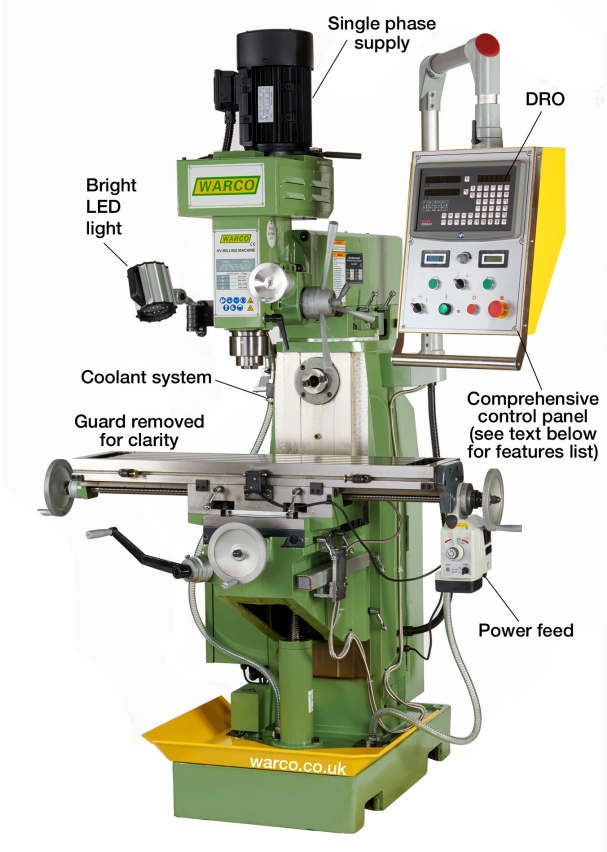

FIGURE 1 – Machine de fraisage

1.2 Datasets

Ce dataset est dédié à une machine de fraisage et comporte 10 000 points de données disposés en lignes, avec chacun présentant 14 caractéristiques distinctes réparties en colonnes.

- **UID** : Identifiant unique allant de 1 à 10 000.
- **ID du produit** : Composé d'une lettre L, M ou H pour les variantes de qualité faible (50% des produits), moyenne (30%) et élevée (20%), ainsi qu'un numéro de série spécifique à la variante.
- **Type** : Type de produit seulement (L, M ou H).
- **Température de l'air [K]** : Générée à l'aide d'un processus de marche aléatoire, ensuite normalisée à un écart type de 2 K autour de 300 K.
- **Température du processus [K]** : Générée à l'aide d'un processus de marche aléatoire, normalisée à un écart type de 1 K, ajoutée à la température de l'air plus 10 K.
- **Vitesse de rotation [rpm]** : Calculée à partir d'une puissance de 2860 W, avec un bruit normalement distribué superposé.
- **Couple [Nm]** : Les valeurs de couple sont distribuées normalement autour de 40 Nm avec un écart type de 10 Nm et aucune valeur négative.
- **Usure de l'outil [min]** : Les variantes de qualité H/M/L ajoutent respectivement 5/3/2 minutes d'usure à l'outil utilisé dans le processus.
- **Étiquette "défaillance de la machine"** : Indique si la machine a échoué pour ce point de données particulier en raison de l'un des cinq modes de défaillance indépendants suivants :
  - **Défaillance de l'usure de l'outil (TWF)** : Cette défaillance survient lorsque l'outil atteint un niveau d'usure spécifique, entraînant son remplacement ou son échec aléatoire après un certain temps d'utilisation.
  - **Défaillance de la dissipation de chaleur (HDF)** : Cette défaillance se produit lorsque la différence entre la température de l'air et la température du processus est inférieure à 8,6 K et que la vitesse de rotation de l'outil est inférieure à 1380 tr/min.
  - **Défaillance de l'alimentation (PWF)** : Cette défaillance survient lorsque la puissance requise pour le processus est en dehors d'une plage spécifique, déterminée par le produit du couple et de la vitesse de rotation.
  - **Défaillance de surcharge (OSF)** : Cette défaillance se produit lorsque le produit de l'usure de l'outil et du couple dépasse des seuils spécifiques pour chaque variante de qualité.
  - **Défaillances aléatoires (RNF)** : Ces défaillances sont aléatoires et se produisent indépendamment des paramètres du processus, avec une probabilité de 0,1 % pour chaque processus.

Ce dataset est extrait de l'article suivant : S. Matzka, **"Explainable Artificial Intelligence for Predictive Maintenance Applications,"** 2020 Third International Conference on Artificial Intelligence for Industries (AI4I), 2020, pp. 69-74

2 Analyse de données

2.1 Affichage de données

Voici nos données affichées à l'aide de la fonction head() de la bibliothèque pandas.

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	0	0	0	0	0
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	0	0	0	0	0

FIGURE 2 – affichage de données

2.1.1 Quelques informations sur notre data

Listing 1 – Affichage des informations sur les données

```
data.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)
memory usage: 1.1+ MB
```

Listing 2 – Affichage des informations sur les données

```
missing_values = data.isnull().sum()
missing_values
```

```
UDI                        0
Product ID                 0
Type                       0
Air temperature [K]        0
Process temperature [K]    0
Rotational speed [rpm]     0
Torque [Nm]                0
Tool wear [min]            0
Machine failure            0
TWF                        0
HDF                        0
PWF                        0
OSF                        0
RNF                        0
dtype: int64
```

2.1.2 Résumé statistique des données numériques

TABLE 1 – Statistiques descriptives des données numériques

	count	mean	std	min	25%	50%	75%	max
UDI	10000.0	5000.50000	2886.895680	1.0	2500.75	5000.5	7500.25	10000.0
Air temperature [K]	10000.0	300.00493	2.000259	295.3	298.30	300.1	301.50	304.5
Process temperature [K]	10000.0	310.00556	1.483734	305.7	308.80	310.1	311.10	313.8
Rotational speed [rpm]	10000.0	1538.77610	179.284096	1168.0	1423.00	1503.0	1612.00	2886.0
Torque [Nm]	10000.0	39.98691	9.968934	3.8	33.20	40.1	46.80	76.6
Tool wear [min]	10000.0	107.95100	63.654147	0.0	53.00	108.0	162.00	253.0
Machine failure	10000.0	0.03390	0.180981	0.0	0.00	0.0	0.00	1.0
TWF	10000.0	0.00460	0.067671	0.0	0.00	0.0	0.00	1.0
HDF	10000.0	0.01150	0.106625	0.0	0.00	0.0	0.00	1.0
PWF	10000.0	0.00950	0.097009	0.0	0.00	0.0	0.00	1.0
OSF	10000.0	0.00980	0.098514	0.0	0.00	0.0	0.00	1.0
RNF	10000.0	0.00190	0.043550	0.0	0.00	0.0	0.00	1.0

Le tableau fournit une vue d'ensemble des statistiques descriptives pour chaque variable numérique dans le jeu de données. Voici quelques observations :

1. **Comptage (count)** : Toutes les variables ont un comptage de 10 000, **ce qui indique qu'il n'y a pas de valeurs manquantes dans ces colonnes.**
2. **Moyenne** : La moyenne donne une idée de la tendance centrale des données. Par exemple, la moyenne de la température de l'air est d'environ 300.00 Kelvin.
3. **Écart type (std)** : L'écart type mesure la dispersion des données par rapport à la moyenne. Plus l'écart type est grand, plus les données sont dispersées autour de la moyenne. Par exemple, l'écart type pour la vitesse de rotation est d'environ 179.28 rpm, indiquant une certaine variabilité dans cette mesure.
4. **Minimum et maximum (min, max)** : Ces valeurs indiquent les limites des données. Par exemple, la température de l'air varie de 295.3 Kelvin à 304.5 Kelvin.donc il presque constante.

2.1.3 Calcul du nombre d'occurrences de chaque classe

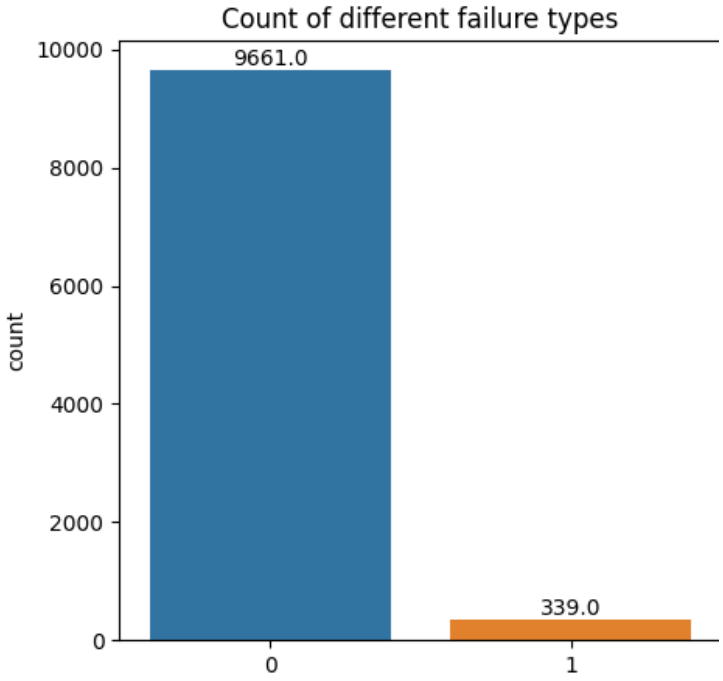

FIGURE 3 – Occurence de chaque classe

TABLE 2 – Pourcentage de 0 et de 1

	Pourcentage de 0	**Pourcentage de 1**
TWF	99.54 %	0.46 %
HDF	98.85 %	1.15 %
PWF	99.05 %	0.95 %
OSF	99.02 %	0.98 %
RNF	99.81 %	0.19 %
Machine failure	96.61%	3.39%

Les deux figures montrent que les données utilisées ne sont pas équilibrées, avec une grande majorité de données dans la classe 0, où la machine n'est pas défectueuse (il n'y a pas de panne), par rapport à la classe 1. Par conséquent, il est nécessaire de rééquilibrer les données avant d'entraîner le modèle.

2.1.4 Visualisation de la distribution des qualités de la machine

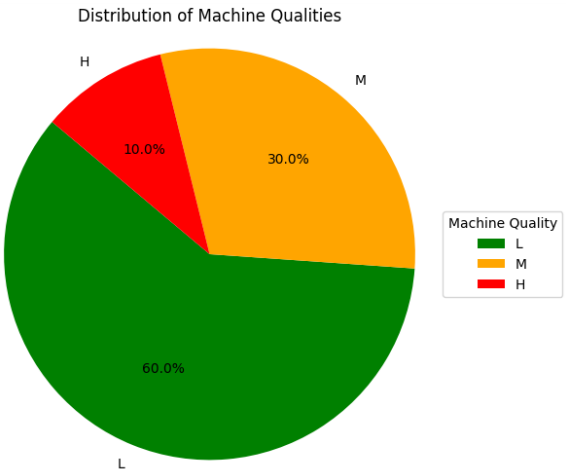

FIGURE 4 – Occurence de chaque classe

Ce diagramme circulaire ci-dessous montre la répartition des défaillances selon les différentes qualités de machines. Cette analyse est essentielle pour déterminer si certaines qualités de machines présentent un risque plus élevé de défaillance.

2.1.5 Visualisation du pourcentage de défaillance par qualité de machine

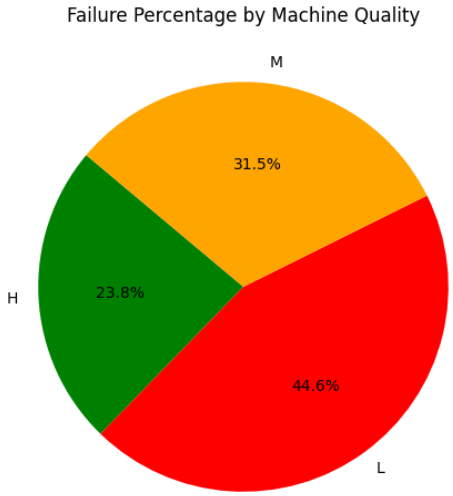

FIGURE 5 – Distribution de défaillance par qualité de machine

Il est clair que les machines classées comme étant de « Qualité Basse » présentent un taux de défaillance plus élevé que celles classées comme étant de « Qualité Élevée ». Cela suggère qu'il est nécessaire d'effectuer des maintenances plus fréquentes ou des évaluations de qualité pour les machines de « Qualité Basse » afin de réduire les temps d'arrêt

3 Prétraitement des données

Pour cette partie de séparation des données, nous avons entrepris de résoudre les problèmes détectés lors de la visualisation des données précédentes. Nous avons d'abord posé deux conditions nécessaires pour résoudre le problème d'incohérence entre les défaillances de la machine et les autres modes de défaillance. Ensuite, nous avons supprimé toutes les caractéristiques inutiles et transformé la caractéristique de qualité de la machine d'un objet en un nombre flottant. En outre, nous avons ajouté une nouvelle caractéristique appelée 'Power' en multipliant deux vecteurs, la vitesse de rotation et le couple. Nous avons ensuite tenté d'analyser la corrélation entre ces caractéristiques et avons procédé à l'augmentation des données pour les équilibrer.

3.1 Ajoute la caractéristique Power[W]

Nous avons ajouté la caractéristique 'Power' en multipliant le 'Couple' par la 'Vitesse de rotation'. Cette décision est basée sur le concept physique de la puissance, qui représente la quantité de travail effectuée par unité de temps. En multipliant le couple, qui mesure la force appliquée à une certaine distance du point de rotation, par la vitesse de rotation, qui mesure le taux de changement de l'angle de rotation par unité de temps, nous obtenons une mesure de la vitesse à laquelle le travail est effectué. Ainsi, la 'Puissance' nous donne une indication de la quantité de travail effectuée par la machine dans un laps de temps donné, ce qui peut être un indicateur important pour comprendre les performances et le comportement de la machine.

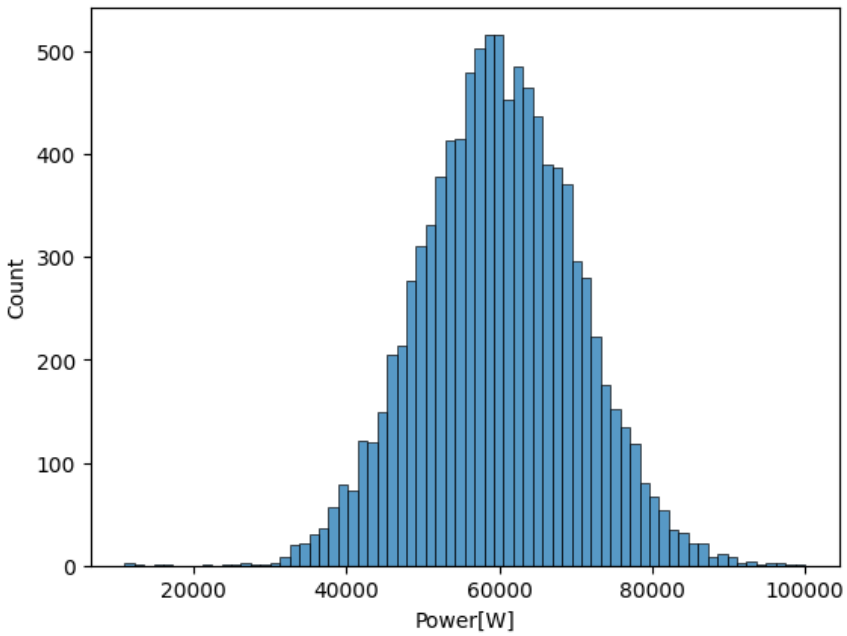

FIGURE 6 – Histogramme de l'attribut Power

4 Modeling

4.1 Préparation de données

4.1.1 Importation des bibliothèques

Dans cette première partie, j'ai importé toutes les bibliothèques nécessaires pour entraîner mes modèles d'apprentissage automatique et les évaluer.

Listing 3 – Import the bibliothèque

```
import time

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import SVMSMOTE
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import RandomOverSampler
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import learning_curve
from sklearn.metrics import f1_score, precision_score, recall_score, accura
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

4.1.2 Création d'une dataframe

Nous allons créer un DataFrame nommé 'modele_performance' pour enregistrer toutes les métriques des modèles utilisés dans ce projet, afin de faciliter la comparaison de ces modèles.

Listing 4 – Création de $\text{model}_performance$

```
modele_performance = pd.DataFrame(columns=['Accuracy', 'Recall', 'Precision
```

4.2 Decision Tree Model

Le modèle d'arbre de décision est une choix judicieux pour résoudre un problème de classification en raison de sa capacité à fournir une interprétation claire et intuitive des décisions prises par le modèle

4.2.1 Choisir la bonne valeur de la profondeur de l'arbre de décision.

Pour éviter le surajustement, on va tracer les courbes de la fonction de perte pour l'ensemble d'entraînement et de test en fonction de la valeur de la profondeur de l'arbre de décision.

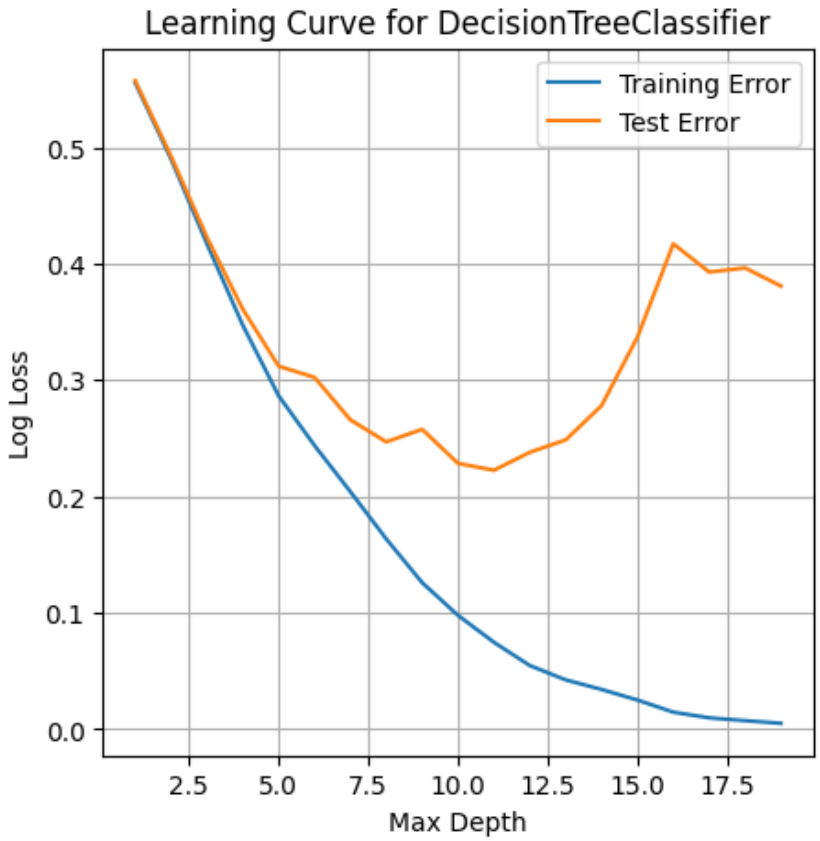

FIGURE 7 – Fonction de perte en fonction de la profondeur maximale de l'arbre (max_depth)

Cette figure montre qu'il y a un surajustement à partir maxd_Depth = 7. Par conséquent, la meilleure valeur pour notre modèle serait celle qui offre de bonnes performances en termes de métriques sans surajustement.

4.2.2 Evaluation modèle avec max_deep = 7

Métrique	Valeur
Précision	92,16%
Rappel	92,16%
Précision	92,16%
F1-Score	92,16%
MCC	84,33%
Score ROC AUC	96,39%
Temps d'entraînement	0,38 s
Temps de prédiction	0,01 s
Total	0,39 s

FIGURE 8 – Tableau de métriques

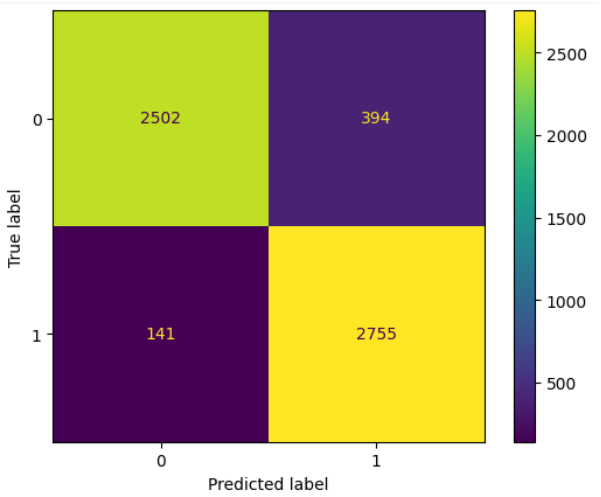

FIGURE 9 – Matrice confusion

Ces valeurs montrent des performances globalement solides du modèle, avec des scores élevés de précision, rappel et F1-Score, tous évalués à 92,16%. Le score MCC de 84,33% indique également une bonne concordance entre les prédictions du modèle et les observations réelles. De plus, un score ROC AUC élevé de 96,39% suggère une bonne capacité du modèle à discriminer entre les classes positives et négatives. Les temps d'entraînement et de prédiction sont également très courts, ce qui indique une efficacité opérationnelle du modèle lors de son déploiement en production. En résumé, ces résultats témoignent d'un modèle performant et bien équilibré.

4.2.3 Evolution du score

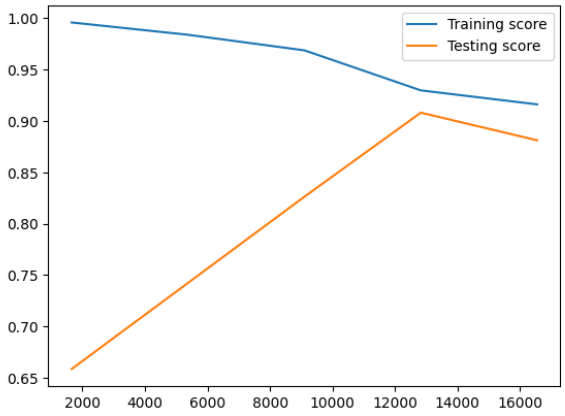

FIGURE 10 – Evolution du score

4.2.4 la courbe ROC et l'AUC

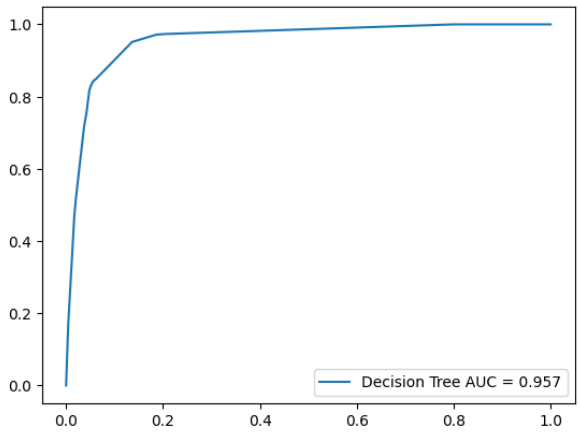

FIGURE 11 – la courbe ROC et l'AUC

4.3 Modèle du Regression logistique

La régression logistique est un choix judicieux pour les problèmes de classification binaires en raison de sa simplicité et de son efficacité. Ce modèle offre une interprétabilité des coefficients, ce qui permet de comprendre comment chaque variable explicative influence la probabilité de chaque classe. Avec un nombre limité de paramètres à estimer, elle réduit le risque de surajustement, ce qui en fait un outil efficace même avec des ensembles de données limités.

4.3.1 Evaluation du modèle

Métrique	Valeur
Accuracy	84.15%
Recall	84.15%
Precision	84.15%
F1-Score	84.15%
MCC	68.30%
ROC AUC score	91.56%
Temps d'entraînement	0.05 s
Temps de prédiction	0.00 s
Total	0.05 s

FIGURE 12 – Tableau de métriques

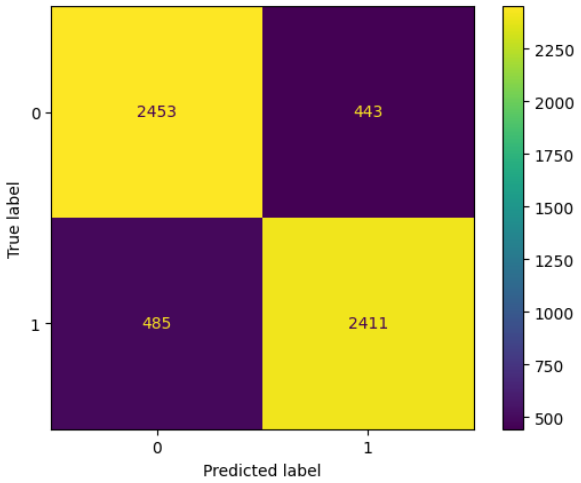

FIGURE 13 – Matrice de confusion

4.3.2 La courbe Roc et l'AUC

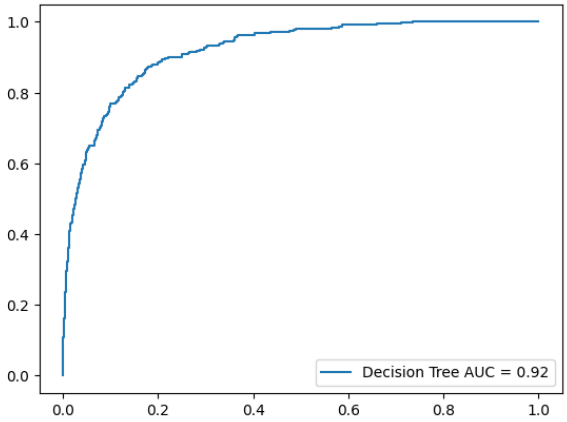

FIGURE 14 – la courbe ROC et l'AUC

4.4 Modèle du Random Forest

Le modèle de forêt aléatoire (Random Forest) a été choisi pour résoudre le problème de classification des machines en panne et des machines non en panne pour plusieurs raisons. Tout d'abord, en raison de sa robustesse aux données bruitées et aux valeurs aberrantes, le Random Forest est capable de gérer efficacement les variations et les anomalies dans les données, assurant ainsi une meilleure généralisation du modèle. De plus, sa capacité à gérer les relations non linéaires entre les variables d'entrée et la variable cible, ainsi que les interactions complexes entre les caractéristiques, en fait un choix idéal pour des problèmes où les relations peuvent être complexes et difficiles à modéliser.

4.4.1 Evaluation du modèle

Metric	Value (%)
Accuracy	99.31
Recall	99.31
Precision	99.32
F1-Score	99.31
MCC	98.63
ROC AUC score	92.04
Time to train	0.09 s
Time to predict	0.02 s
Total time	0.11 s

FIGURE 15 – Tableau de métriques

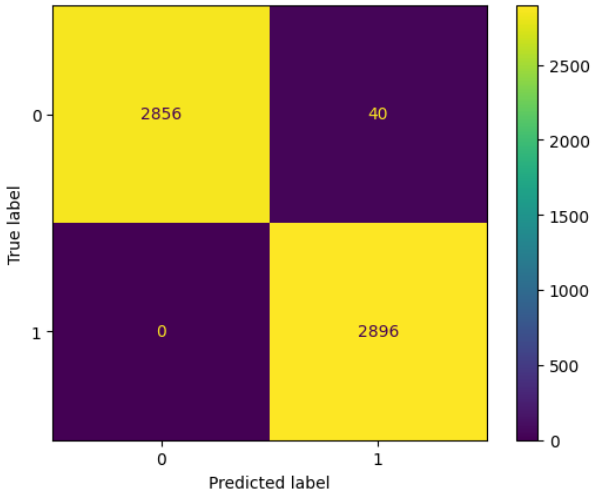

FIGURE 16 – Matrice confusion

4.4.2 Evolution de la fonction de perte

La figure suivante montre que les courbes de perte d'entraînement et de test sont décroissantes et proches l'une de l'autre. Cela indique qu'il n'y a pas de grande différence entre les deux courbes, ce qui suggère qu'il n'y a pas de surapprentissage du modèle. En d'autres termes, le modèle généralise bien les données et ne surajuste pas aux données d'entraînement. Cette observation est essentielle car un surapprentissage peut entraîner une performance médiocre du modèle sur de nouvelles données.

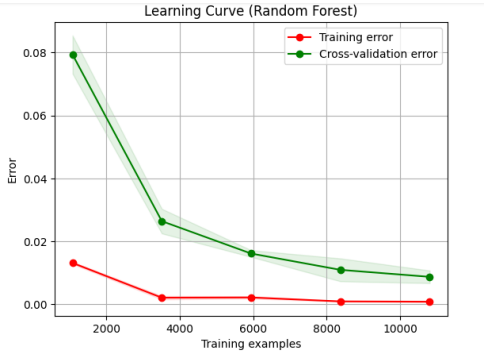

FIGURE 17 – la courbe ROC et l'AUC

4.5 Support Vector Machine

Le choix du modèle SVM (Support Vector Machine) pour ce problème de classification est justifié par sa capacité à trouver un hyperplan optimal qui sépare les deux classes de manière à maximiser la marge entre les points de données les plus proches de chaque classe, tout en minimisant l'erreur de classification. Cela signifie que le SVM est capable de trouver une frontière de décision complexe, même dans des espaces de grande dimension, ce qui est souvent le cas dans les problèmes de classification réels. De plus, le SVM est efficace dans la gestion des données à grande échelle et est moins sensible au surajustement par rapport à certains autres algorithmes de classification.

4.5.1 Evaluation du modèle avec kernel='linear'

TABLE 3 – Tableau des métriques

Métrique	Valeur
Accuracy	84.74%
Recall	84.74%
Precision	84.74%
F1-Score	84.74%
MCC	69.48%
ROC AUC score	92.10%
Temps d'entraînement	22.09 s
Temps de prédiction	0.79 s
Temps total	22.87 s

4.5.2 Evaluation du modèle avec kernel='rbf'

TABLE 4 – Tableau des métriques

Métrique	**Valeur**
Accuracy	91.68%
Recall	91.68%
Precision	91.73%
F1-Score	91.68%
MCC	83.41%
ROC AUC score	97.20%
Temps d'entraînement	20.32 s
Temps de prédiction	1.20 s
Temps total	21.52 s

4.5.3 Evaluation du modèle avec kernel='sigmoid'

TABLE 5 – Résultats de classification

Métrique	**Valeur**
Accuracy	61.15%
Recall	61.15%
Precision	61.16%
F1-Score	61.15%
MCC	22.31%
ROC AUC score	69.26%
Temps d'entraînement	51.09 s
Temps de prédiction	2.71 s
Temps total	53.80 s

4.5.4 Evaluation du modèle avec kernel='poly'

TABLE 6 – Résultats de classification

Métrique	Valeur
Accuracy	87.64%
Recall	87.64%
Precision	87.64%
F1-Score	87.64%
MCC	75.28%
ROC AUC score	94.67%
Temps d'entraînement	19.85 s
Temps de prédiction	0.66 s
Temps total	20.51 s

4.6 Conclusion et discussion

D'après les résultats précédents, il est observé que le modèle SVM performe mieux lorsqu'il utilise le noyau 'rbf'. Le noyau RBF est souvent privilégié en raison de sa capacité à saisir des relations complexes entre les données, même dans des espaces de grande dimension, grâce à sa flexibilité. De plus, il convient mieux aux données non linéaires, qui sont courantes dans les problèmes de classification réels.

En revanche, le noyau Sigmoid est moins flexible et peut rencontrer des difficultés à modéliser de manière efficace des relations complexes entre les données. Il est généralement plus adapté à des tâches de classification simples et linéaires.

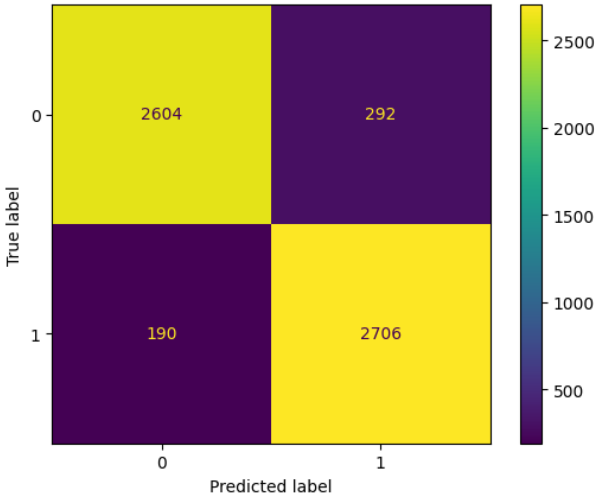

FIGURE 18 – matrice confusion

5 Réseaux de neurones artificiels

5.1 Modèle MLPClassifier

Le choix du modèle MLPClassifier pour résoudre le problème de classification des machines en panne et non en panne est justifié par sa capacité à capturer des relations complexes entre les données, notamment grâce à sa structure multicouche. En tant que réseau de neurones multicouches, le MLPClassifier est capable d'apprendre des représentations hiérarchiques des données, ce qui est souvent nécessaire pour des problèmes de classification complexes comme celui-ci. De plus, les réseaux de neurones multicouches sont connus pour leur flexibilité et leur capacité à modéliser des relations non linéaires entre les caractéristiques, ce qui est probablement nécessaire compte tenu de la nature des données de défaillance des machines. En utilisant le MLPClassifier, nous pouvons donc exploiter la puissance des réseaux de neurones pour obtenir des performances élevées dans la classification des machines en panne et non en panne.

5.1.1 Evaluation du modèle

Metric	Value (%)
Accuracy	98.45%
Recall	98.45%
Precision	98.49%
F1-Score	98.45%
MCC	96.94%
ROC AUC score	99.64%
Temps d'entraînement	128.95 s
Temps de prédiction	0.04 s
Temps total	128.99 s

FIGURE 19 – Tableau de métriques

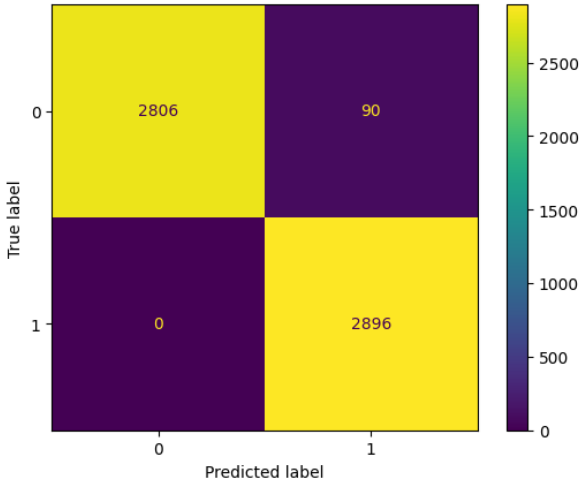

FIGURE 20 – Matrice confusion

5.2 Réseau de neuronne avec Tensorflow

Le choix d'utiliser un réseau de neurones avec TensorFlow pour ce problème de classification entre les machines en panne et les machines non en panne est motivé par plusieurs raisons. Tout d'abord, les réseaux de neurones sont capables de capturer des motifs complexes et non linéaires dans les données, ce qui est souvent nécessaire pour la classification de données complexes telles que celles provenant de machines industrielles. De plus, TensorFlow est une bibliothèque populaire et puissante pour l'apprentissage automatique et l'apprentissage profond, offrant une grande flexibilité pour la construction et la personnalisation des modèles de réseaux de neurones. En utilisant TensorFlow, nous pouvons facilement construire, entraîner et évaluer des modèles de réseaux de neurones, tout en bénéficiant de fonctionnalités avancées telles que le calcul sur GPU pour accélérer l'entraînement des modèles. Enfin, les réseaux de neurones sont robustes face à des volumes de

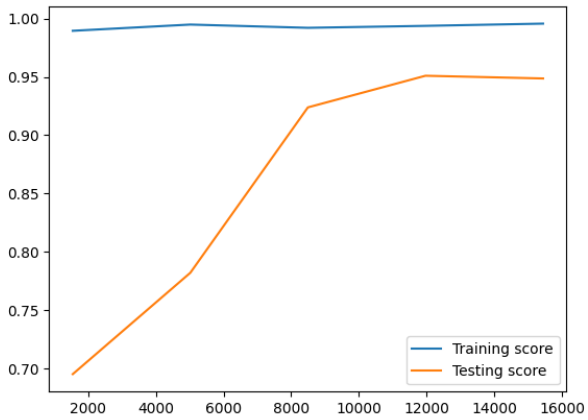

FIGURE 21 – Evolution de la fonction perte

données importants et peuvent être adaptés à différents types de données et de problèmes de classification, ce qui en fait un choix prometteur pour la classification des machines en panne et non en panne.

5.2.1 Archétecture du modèle

```python
model_1 = Sequential()

# Couche d'entrée
model_1.add(Dense(units=64, activation="relu", input_shape=(7,)))

# Couches cachées
model_1.add(Dense(units=128, activation="relu"))
model_1.add(Dropout(0.5))
model_1.add(Dense(units=64, activation="relu"))
model_1.add(Dense(units=64, activation="relu"))
model_1.add(Dropout(0.5))

# Couche de sortie
model_1.add(Dense(units=1, activation="sigmoid"))

# Compilation du modèle avec l'optimiseur "adam", la perte "binary_crossentropy" et la métrique "accuracy"
model_1.compile(optimizer="adam", loss="mean_squared_error", metrics=['accuracy'])

# optimizer: SGD ou adam
# loss : mean_squared_error ou categorical_crossentropy
```

FIGURE 22 – Architecture du notre modèle

5.2.2 Evaluation du modèle

Les résultats montrent que le modèle de réseau de neurones, entraîné avec TensorFlow, offre une performance solide dans la classification des machines en panne et non en panne. Avec une précision de 97.20% et un rappel de 98.3%, le modèle démontre une capacité élevée à classifier correctement les échantillons positifs et négatifs. De plus, un F1-score de 97.23% et un coefficient de corrélation de Matthews de 94.43% soulignent l'équilibre entre précision et rappel, ainsi que la corrélation entre les prédictions du modèle et les

observations réelles. Le score de l'aire sous la courbe ROC de 97.20% indique une bonne capacité du modèle à distinguer les classes positives et négatives. Enfin, les temps de formation et de prédiction relativement courts (62.35 s et 0.51 s respectivement) témoignent de l'efficacité du modèle dans l'utilisation des ressources informatiques. En résumé, ces résultats démontrent une performance satisfaisante du modèle de réseau de neurones dans la classification des machines en panne et non en panne.

Metric	Value (%)
Accuracy	98.31%
Recall	98.58%
Precision	98.04%
F1-Score	98.31%
MCC	96.62%
ROC AUC score	98.31%
Time to train	83.29 s
Time to predict	0.60 s
Total	83.89 s

FIGURE 23 – Tableau de métriques

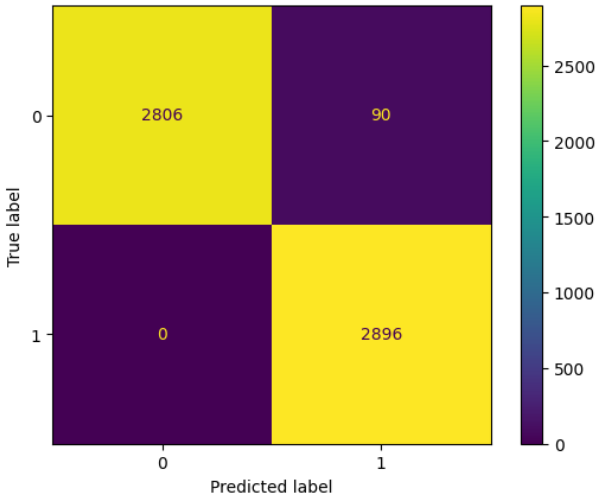

FIGURE 24 – Matrice confusion

5.2.3 Traitement la performance de ce modèle

La figure suivante montre que les courbes de perte d'entraînement et de test sont décroissantes et proches l'une de l'autre. Aussi pour les courbes d'accuracy sont croissantes et proches l'une de l'autre Cela indique qu'il n'y a pas de grande différence entre les deux courbes, ce qui suggère qu'il n'y a pas de surapprentissage du modèle.

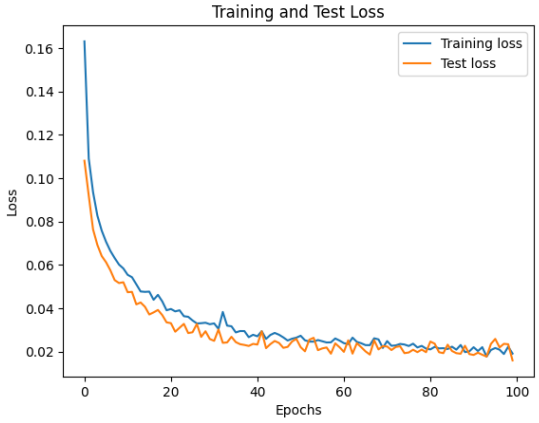

FIGURE 25 – Matrice de confusion 1

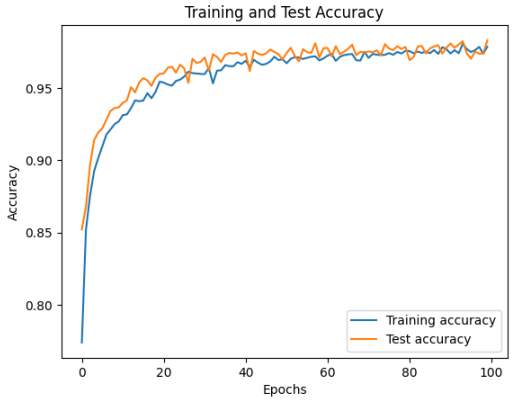

FIGURE 26 – Matrice de confusion 2

6 Conclusion

6.1 Affichage des métriques

	Accuracy	Recall	Precision	F1-Score	MCC	ROC AUC	Training Time	Prediction Time	Total Time
Decision Tree	0.928349	0.928349	0.929919	0.928284	0.858267	0.962325	0.067718	0.000900	0.068619
Regression Logistic	0.835808	0.835808	0.835863	0.835801	0.671671	0.919198	0.062105	0.000536	0.062641
Random forest	0.992403	0.992403	0.992517	0.992403	0.984920	0.999482	0.102923	0.013201	0.116124
SVM_Linear	0.844613	0.844613	0.844667	0.844607	0.689280	0.919059	19.378748	0.814462	20.193210
SVM_RBF	0.912638	0.912638	0.913073	0.912615	0.825711	0.972488	18.570148	1.251885	19.822033
SVM_Poly	0.614986	0.614986	0.615029	0.614950	0.230015	0.682699	52.693883	2.712376	55.406259
SVM_POLY	0.873446	0.873446	0.873501	0.873442	0.746947	0.945796	17.124339	0.709343	17.833682
RNN_MLP	0.986706	0.986706	0.987050	0.986703	0.973756	0.996494	115.354668	0.014833	115.369502
RNN_Tensorflow	0.972030	0.983425	0.961512	0.972346	0.944306	0.972030	62.354981	0.513282	62.868263

FIGURE 27 – Métrique de toutes les modèles

6.2 Comparaison des métriques

	count	mean	std	min	25%	50%	75%	max
Accuracy	9.0	0.884553	0.116826	0.614986	0.844613	0.912638	0.972030	0.992403
Recall	9.0	0.885820	0.117949	0.614986	0.844613	0.912638	0.983425	0.992403
Precision	9.0	0.883681	0.116015	0.615029	0.844667	0.913073	0.961512	0.992517
F1-Score	9.0	0.884572	0.116863	0.614950	0.844607	0.912615	0.972346	0.992403
MCC	9.0	0.769430	0.233795	0.230015	0.689280	0.825711	0.944306	0.984920
ROC AUC	9.0	0.929952	0.097153	0.682699	0.919198	0.962325	0.972488	0.999482
Training Time	9.0	31.745501	38.592656	0.062105	0.102923	18.570148	52.693883	115.354668
Prediction Time	9.0	0.670091	0.888353	0.000536	0.013201	0.513282	0.814462	2.712376
Total Time	9.0	32.415592	38.720471	0.062641	0.116124	19.822033	55.406259	115.369502

FIGURE 28 – Statistiques

6.3 Discussion

Dans le tableau précédent, nous avons examiné les performances des différents modèles utilisés, en tenant compte à la fois de leurs valeurs métriques (telles que l'accuracy, le recall, le F-score, etc.) et de leur temps d'exécution total. En analysant ces résultats, il apparaît clairement que le modèle de réseau de neurones avec TensorFlow se distingue comme le plus solide et performant dans notre situation. Non seulement il affiche de bonnes valeurs pour les métriques d'évaluation telles que l'accuracy, le recall et le F-score, mais il parvient également à maintenir un équilibre entre précision et rappel, ce qui est essentiel pour une classification précise. De plus, le modèle de réseau de neurones avec TensorFlow démontre une capacité notable à généraliser les données sans surajustement, comme en témoigne l'absence de grandes différences entre les performances sur les ensembles d'entraînement et de test. En ce qui concerne le temps total d'exécution, le modèle de réseau de neurones avec TensorFlow reste compétitif avec des temps relativement courts, ce qui renforce sa position en tant que choix privilégié pour notre problème de classification. En résumé, les performances globales, la capacité à éviter le surajustement et l'efficacité du temps d'exécution font du modèle de réseau de neurones avec TensorFlow le choix le plus favorable et le plus performant pour notre tâche de classification.

7 Déploiement de notre modèle

Pour que notre modèle puisse être utilisé, nous avons entrepris de le déployer et de le rendre accessible via un site web. Cette étape permettra aux utilisateurs d'interagir avec le modèle de manière conviviale et intuitive. En déployant le modèle sur un site web `http://172.22.5.120:5000`, nous offrons aux utilisateurs la possibilité d'accéder à ses fonctionnalités et de soumettre leurs propres données pour des prédictions en temps réel. **Pour plus de détails sur le processus de déploiement et l'utilisation du modèle,**

nous avons prévu d'inclure une vidéo explicative dans le dossier de notre projet. Cette vidéo fournira des instructions détaillées sur la manière d'accéder et d'utiliser le modèle déployé, offrant ainsi aux utilisateurs une ressource précieuse pour tirer le meilleur parti de notre solution.

7.1 Le module déploié

Dans ce code, nous avons défini trois fonctions clés :

— **normalise()** : Cette fonction est responsable de la normalisation des données sélectionnées par l'utilisateur sur notre site web. Étant donné que le modèle a été entraîné sur des données normalisées, il est crucial de normaliser également les données d'entrée avant de les utiliser pour les prédictions. Pour cela, nous utilisons les statistiques calculées à partir de notre jeu de données d'origine, telles que la moyenne (mean) et l'écart-type (std), afin de standardiser les valeurs des caractéristiques dans les données d'entrée.

— **load_model()** : Cette fonction charge le modèle pré-entraîné et enregistre les poids des paramètres du modèle. Elle assure que le modèle est prêt à être utilisé pour les prédictions une fois chargé, en lui fournissant les poids corrects pour chaque couche.

— **predict()** : La fonction predict est responsable de faire des prédictions sur de nouvelles données d'entrée. En fonction de la sortie de la prédiction (0 ou 1), cette fonction renvoie "Functional" si la prédiction est égale à 1, ce qui signifie que la machine est fonctionnelle, sinon elle renvoie "Failure", indiquant que la machine est défectueuse.

Listing 5 – Création de $\text{model}_performance$

```
def normalise(normalisation_params, X):

    with open(normalisation_params, 'rb') as f:
        parametres = pickle.load(f)
    mu = parametres['mu']
    sigma = parametres['sigma']

    scaler = StandardScaler()
    scaler.mean_ = mu
    scaler.scale_ = sigma

    X_scaled = scaler.transform(X)
    return X_scaled

def load_model(model_weights):

    model = Sequential()
    model.add(Dense(units=64, activation="relu", input_shape=(7,)))
    model.add(Dense(units=128, activation="relu"))
    model.add(Dropout(0.5))
```

```
    model.add(Dense(units=64, activation="relu"))
    model.add(Dense(units=64, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(units=1, activation="sigmoid"))

    model.load_weights(model_weights)

    return model

model = load_model('model/model.weights.h5')

def predict(features):

    normalise('model_data/normalisation_params.pkl', features)

    features = np.array(features)

    prediction_proba = model.predict(features)
    prediction = int(prediction_proba > 0.5)
    return "Functional" if prediction == 1 else "Failure"

if __name__ == "__main__":
```

7.2 Docker

```
// put FROM python:3.9-slim
WORKDIR ./
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]your code here
```

FIGURE 29 – Configuration Docker pour une application Python avec Flask

7.3 Serveur

app.py > main

```python
from flask import Flask, render_template, request, redirect, url_for, flash
from model import *
app = Flask(__name__)
app.secret_key = '123456789'
@app.route('/')
def index():
    return render_template('index.html')
# Route principale
@app.route('/main', methods=['GET', 'POST'])
def main():
    if request.method == 'POST':
        # Récupérer les données du formulaire
        user_input = [[
            int(request.form['Type']),
            float(request.form['Air_temperature_K']),
            float(request.form['Process_temperature_K']),
            float(request.form['Rotational_speed_rpm']),
            float(request.form['Torque_Nm']),
            float(request.form['Tool_wear_min']),
            float(request.form['Power'])
        ]]
        prediction = predict(user_input)

        # Renvoyer les résultats à la page HTML
        return render_template('index.html', prediction=prediction)
    else:
        return render_template('index.html')
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

FIGURE 30 – Application Flask pour la prédiction de défaillances de machines"

7.4 Page site wep

Voici page wep de notre site :

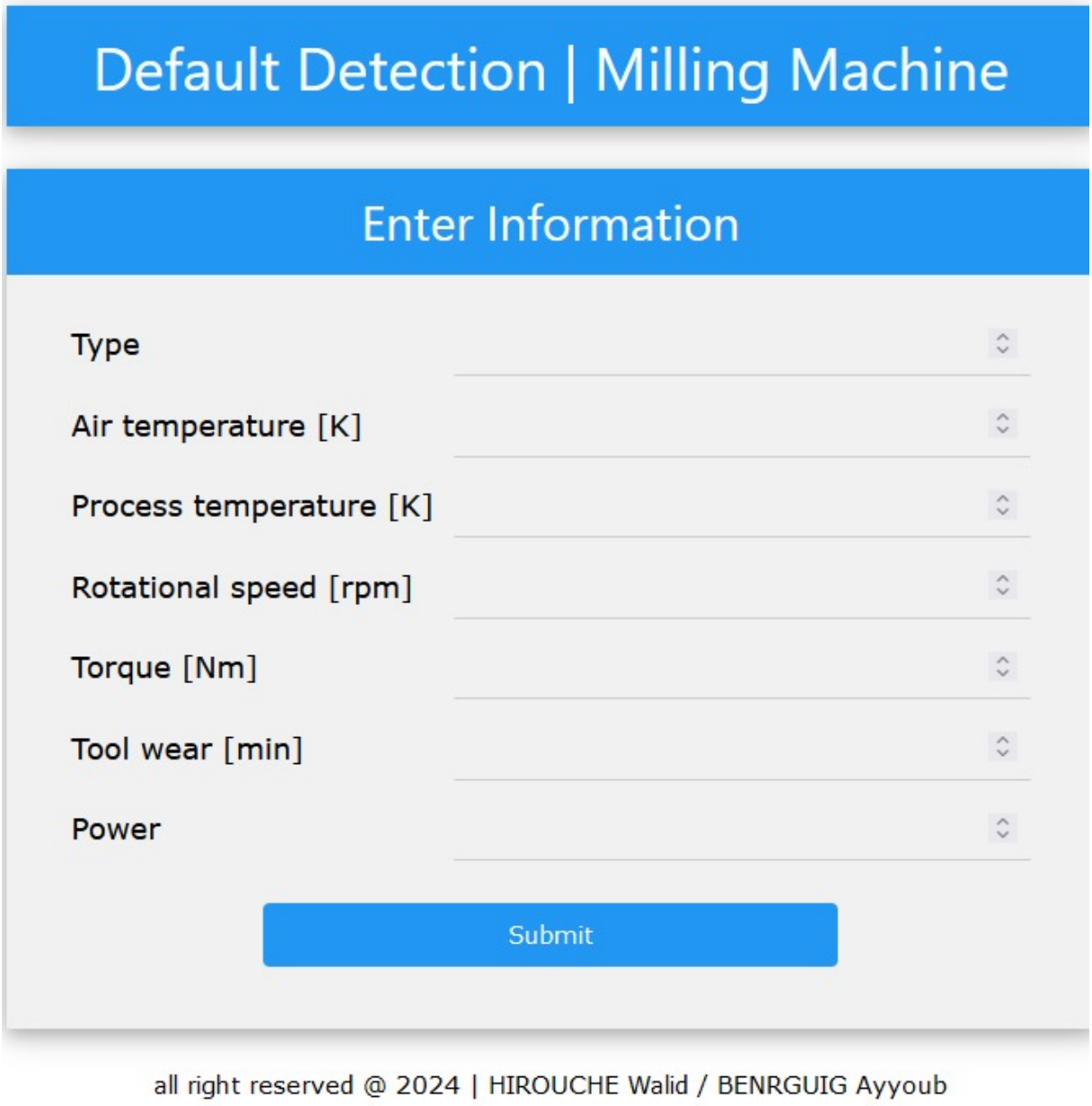

FIGURE 31 – Application Flask pour la prédiction de défaillances de machines"