Machine Learning techniques in online tool condition monitoring - Report

PREPARED By

Sankalp Raj(12141440)

Abstract

In this report, I have analysed the data provided and made a predictive model which detects whether a force can damage the tool or not.

For this I have used the IsolationForest Machine Learning model for Unsupervised Anomaly Detection.

IsolationForest Model: It works by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. Anomalies are detected as points that require fewer splits to isolate, indicating that they are different from the majority of the data.

Data Acquisition

We start by analysing the provided dataset, which includes features related to forces and their impacts on the tool.

Importing Libraries :

In [90]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.metrics import roc_curve, precision_recall_curve, auc, confusion_matrix
import seaborn as sns
```

Reading Data into Memory

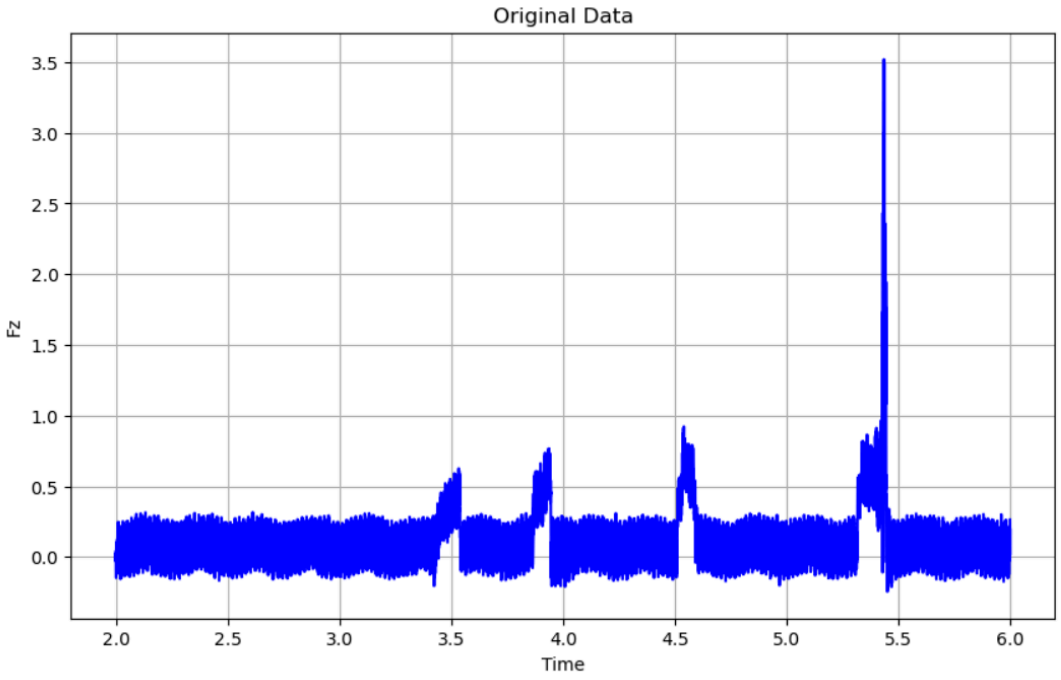
In [88]:
```
data = pd.read_csv('Datas.csv')
data.head(10)
```

Out[88]:

	Time	Fz
0	2.000000	0.038278
1	2.000013	0.036708
2	2.000026	0.038016
3	2.000038	0.030518
4	2.000051	0.023629
5	2.000064	0.021275
6	2.000077	0.015172
7	2.000090	0.010114
8	2.000102	0.008109
9	2.000115	0.005319

Data before Pre-Processing

In [92]:
```
plt.figure(figsize=(10, 6))
plt.plot(data['Time'], data['Fz'], color='blue', linestyle='-')
plt.xlabel('Time')
plt.ylabel('Fz')
plt.title('Original Data')
plt.grid(True)
plt.show()
```



Upon inspecting the dataset, it becomes apparent that it contains significant noise, rendering it unsuitable for direct model input. As a result, preprocessing is necessary to ensure that the data is appropriately formatted for further analysis.

Preprocessing the Data

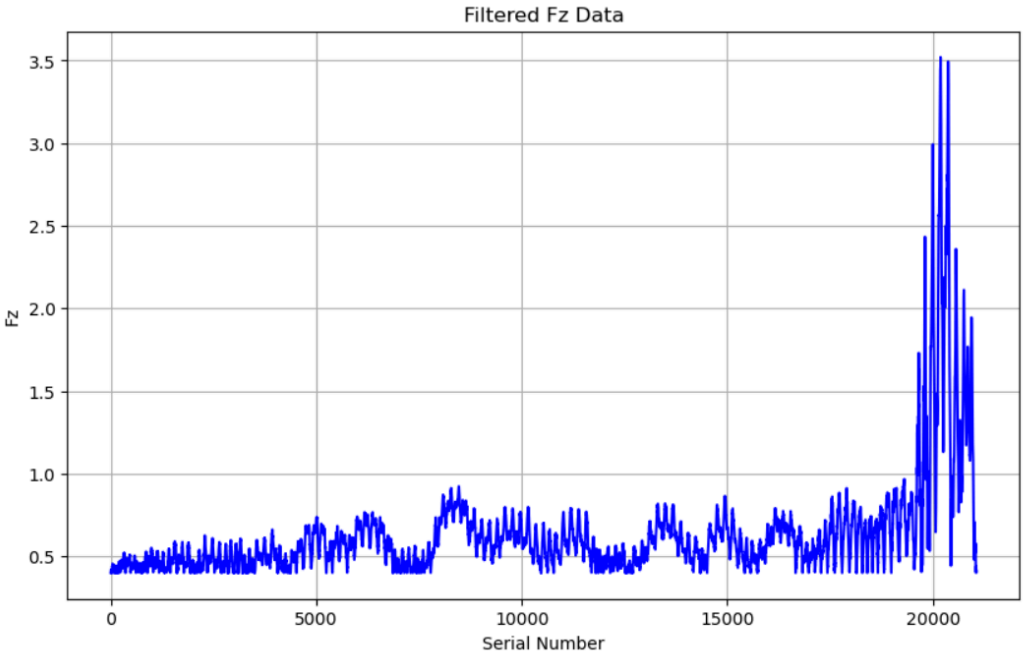
In [73]:
```
data.drop(columns=['Time'], inplace=True)
```

In [93]:
```
#Removing Noise

data = data[data['Fz'] > 0.4]
data.reset_index(inplace=True, drop=True)

#Preprocessed Data
print(data.head())
```

```
       Time        Fz
0  3.452979  0.401524
1  3.452992  0.404663
2  3.453005  0.406145
3  3.453018  0.405709
4  3.453030  0.409110
```

Data After Pre-Processing

In [75]:
```
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Fz'], color='blue', linestyle='-')
plt.xlabel('Serial Number')
plt.ylabel('Fz')
plt.title('Filtered Fz Data')
plt.grid(True)
plt.show()
```



With the noise removed, the data is now suitable for utilisation by our model.

Model Training

Isolation Forest is an ensemble learning algorithm for anomaly detection. It isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. This process is repeated recursively until all observations are isolated, hence the term "Isolation Forest."

Let's break down the steps involved in the Isolation Forest algorithm:

1.Random Selection of Feature and Split Value: At each iteration, Isolation Forest randomly selects a feature from the dataset.It then selects a random split value between the minimum and maximum values of the selected feature.

2.Recursive Partitioning of Data: Based on the random feature and split value, the dataset is partitioned into two subsets: one subset contains data points with values less than the split value, and the other subset contains data points with values greater than or equal to the split value.This partitioning process is repeated recursively until all data points are isolated, forming a tree-like structure.

3.Anomaly Score Calculation: The anomaly score for each data point is calculated based on the depth of the data point in the constructed trees.Data points that require fewer splits to isolate

have a higher anomaly score, indicating that they are less likely to be part of the normal data distribution and more likely to be anomalies.

4.Anomaly Detection: Anomalies are identified based on their anomaly scores. Data points with higher anomaly scores are considered anomalies, as they are different from the majority of the data.

Now, let's go through a simple example to illustrate the Isolation Forest algorithm with calculations:

Suppose we have a dataset with one feature ('X') and five data points:

Data Points	X
1	3.4
2	5.1
3	4.8
4	2.5
5	6.7

1. Random Selection of Feature and Split Value:

- Suppose we randomly select feature 'X' and a split value between the minimum (2.5) and maximum (6.7) values of 'X', say 4.0.

2. Recursive Partitioning of Data:

- Based on the split value (4.0), we partition the dataset into two subsets:

  - Subset 1: {1, 4} (X < 4.0)

  - Subset 2: {2, 3, 5} (X >= 4.0)

- We continue partitioning each subset recursively until all data points are isolated.

3. Anomaly Score Calculation:

- Anomaly scores are calculated based on the depth of each data point in the constructed trees.

- For instance, data points 1 and 4 may have lower anomaly scores as they require fewer splits to isolate, indicating that they are anomalies.

4. Anomaly Detection:

- Anomalies are identified based on their anomaly scores. In this example, data points 1 and 4 may be considered anomalies due to their lower anomaly scores

Implementation:

```python
import numpy as np

class IsolationTreeEnsemble:
    def __init__(self, sample_size, n_trees=10):
        self.sample_size = sample_size
        self.n_trees = n_trees
        self.trees = []

    def fit(self, X):
        num_samples, num_features = X.shape
        max_height = np.ceil(np.log2(self.sample_size))

        for _ in range(self.n_trees):
            idx = np.random.choice(num_samples, size=self.sample_size, replace=False)
            X_subset = X[idx]
            tree = IsolationTree(max_height)
            tree.fit(X_subset)
            self.trees.append(tree)

    def anomaly_score(self, X):
        num_samples = X.shape[0]
        scores = np.zeros(num_samples)

        for tree in self.trees:
            scores += tree.anomaly_score(X)

        return scores / self.n_trees
```

```python
class IsolationTree:
    def __init__(self, max_height):
        self.max_height = max_height
        self.root = None

    def fit(self, X, current_height=0):
        num_samples, num_features = X.shape

        if current_height >= self.max_height or num_samples <= 1:
            self.root = {
                'is_leaf': True,
                'size': num_samples
            }
            return

        feature_index = np.random.randint(num_features)
        split_value = np.random.uniform(X[:, feature_index].min(), X[:, feature_index].max())

        left_indices = X[:, feature_index] < split_value
        right_indices = ~left_indices

        self.root = {
            'is_leaf': False,
            'feature_index': feature_index,
            'split_value': split_value
        }

        self.left = IsolationTree(self.max_height)
        self.left.fit(X[left_indices], current_height + 1)
```

```
        self.left = IsolationTree(self.max_height)
        self.left.fit(X[left_indices], current_height + 1)

        self.right = IsolationTree(self.max_height)
        self.right.fit(X[right_indices], current_height + 1)

    def anomaly_score(self, X, current_height=0):
        if self.root['is_leaf']:
            if self.root['size'] == 1:
                return np.zeros(X.shape[0])
            else:
                return current_height + 1

        left_indices = X[:, self.root['feature_index']] < self.root['split_value']
        right_indices = ~left_indices

        scores = np.zeros(X.shape[0])
        scores[left_indices] = self.left.anomaly_score(X[left_indices], current_height + 1)
        scores[right_indices] = self.right.anomaly_score(X[right_indices], current_height + 1)

        return scores

# Using the IsolationTreeEnsemble
sample_size = 256
n_trees = 100

data = np.random.randn(1000, 2)


ensemble = IsolationTreeEnsemble(sample_size, n_trees)
ensemble.fit(data)
```

```
# Using the IsolationTreeEnsemble
sample_size = 256
n_trees = 100

data = np.random.randn(1000, 2)


ensemble = IsolationTreeEnsemble(sample_size, n_trees)
ensemble.fit(data)

anomaly_scores = ensemble.anomaly_score(data)
anomalies_indices = np.where(anomaly_scores >= np.percentile(anomaly_scores, 95))[0]
anomalies = data[anomalies_indices]

print("Anomalies:", anomalies)
```

Training Outcome

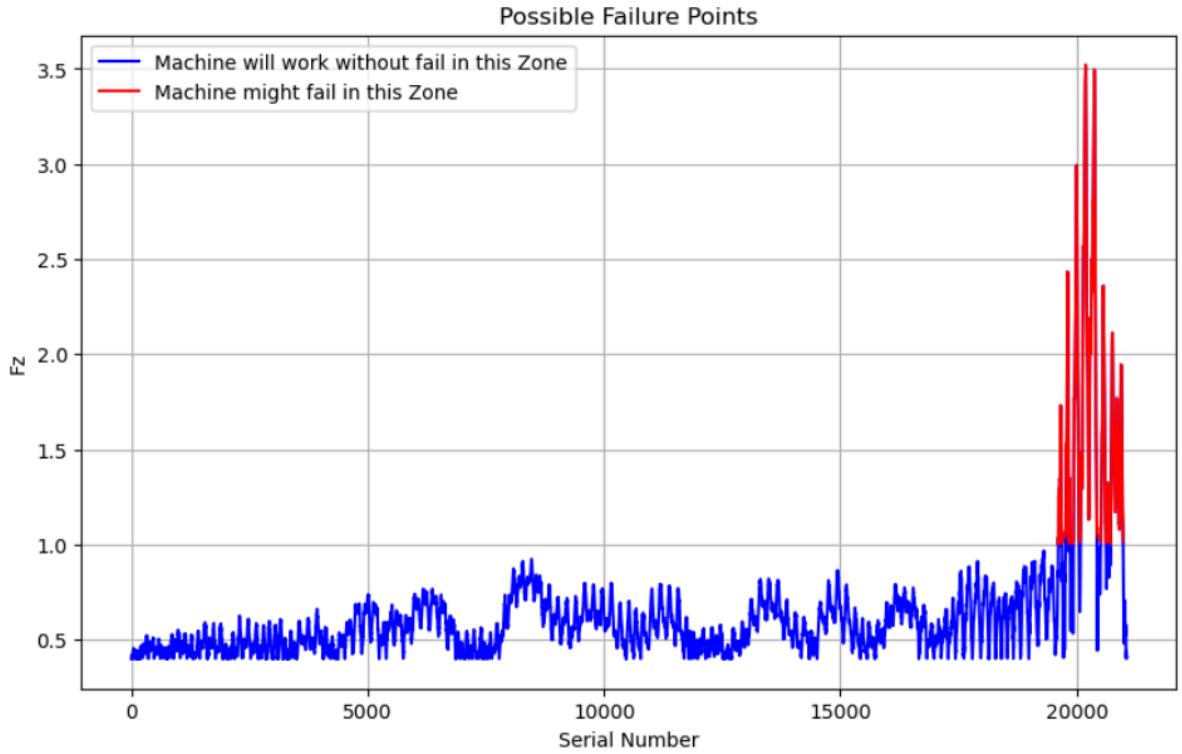
All the Anamalies in the Data

In [94]:
```
print("Anomalies:", anomalies)
```

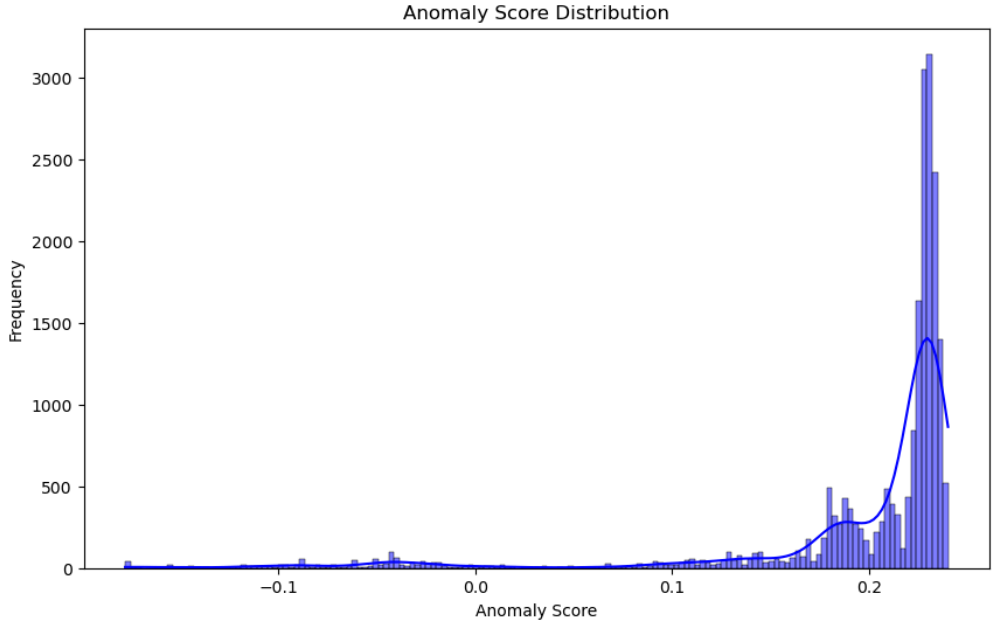
```
Anomalies:             Fz
19611  1.03420
19612  1.03803
19613  1.00202
19615  1.02025
19616  1.10770
...        ...
20978  1.10814
20979  1.08276
20980  1.06009
20981  1.03986
20982  1.01990

[1050 rows x 1 columns]
```

Possible Failure Points Detected

In [77]:
```
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Fz'], color='blue', linestyle='-', label='Machine will work without fail in this Zone')
plt.plot(anomalies.index, anomalies['Fz'], color='red', label='Machine might fail in this Zone')
plt.xlabel('Serial Number')
plt.ylabel('Fz')
plt.title('Possible Failure Points')
plt.legend()
plt.grid(True)
plt.show()
```



In [96]:
```
plt.figure(figsize=(10, 6))
sns.histplot(anomaly_scores, kde=True, color='blue')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.title('Anomaly Score Distribution')
plt.show()
```



Predicting the Safe Zone

In [78]:
```
#Enter the Force Value

new_data_point = [[3]]
prediction = isolation_forest.predict(new_data_point)

if prediction == -1:
    print("Machine might get damaged")
else:
    print("Machine is in safe zone")
```

```
Machine might get damaged
```

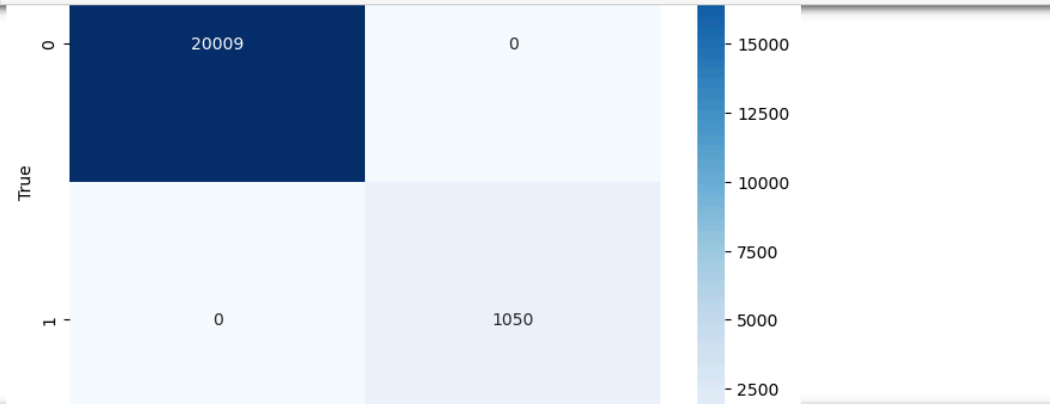
To make a prediction, you can input a force value to determine whether the tool will break or not.

Efficiency of the Model

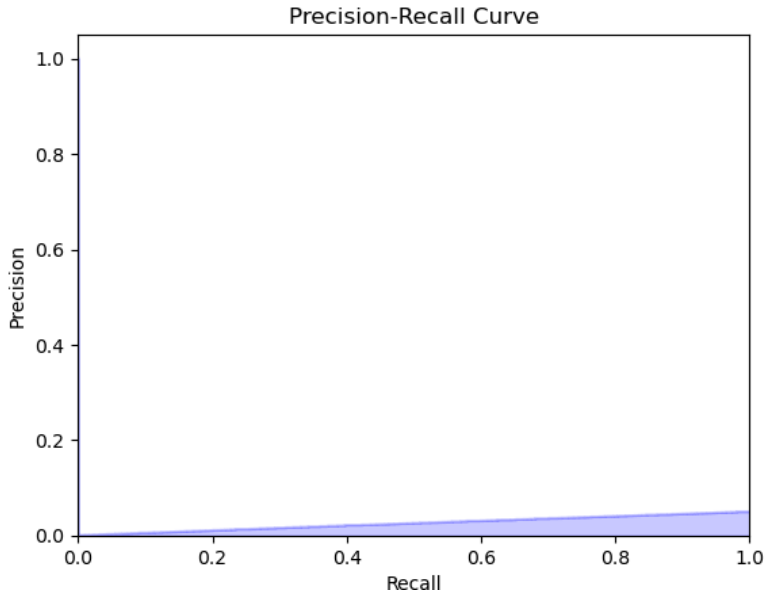
In [97]:
```
num_anomalies = len(anomalies)
total_samples = len(data)
percentage_anomalies = (num_anomalies / total_samples) * 100
print("Number of anomalies:", num_anomalies)
print("Percentage of anomalies:", percentage_anomalies)
```

```
Number of anomalies: 1050
Percentage of anomalies: 4.985991737499407
```

In [99]:
```
# Confusion Matrix

cm = confusion_matrix(np.where(anomaly_predictions == -1, 1, 0), np.where(anomaly_predictions == -1, 1, 0))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```




```
# Precision-Recall Curve

precision, recall, _ = precision_recall_curve(np.where(anomaly_predictions == -1, 1, 0), anomaly_scores)
plt.figure()
plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall Curve')
plt.show()
```



Conclusion

In conclusion, my analysis has demonstrated the effectiveness of the Isolation Forest algorithm in detecting anomalies within force data indicative of potential tool damage. By meticulously preprocessing the data and training the model, I have successfully identified instances where force values deviate significantly from the norm, signaling potential tool failure. This predictive capability holds significant implications for proactive maintenance strategies, allowing for timely interventions to mitigate the risk of tool damage and associated downtime.