TECHNICAL UNIVERSITY OF DORTMUND

Department of Mechanical Engineering

Institute of Production Systems (IPS)

MASTER THESIS

Unsupervised approaches for Anomaly detection in quality management of screw connections

First examiner: Prof. Dr.Christine Müller
Secondary examiner: M.Sc. Nikolai West

Submitted by: Naveen Kumar Bhageradhi
Matriculation number: 231678

Issued on: 05.02.2024
Submitted on: 05.08.2024

Dortmund, August 2024

Table of contents

List of abbreviations

ARI	Adjusted Rand Index
ANN	Artificial Neural Network
AUC	Area Under the Curve
CSV	Comma Seperated Values
CNN	Convolutional Neural Network
CV	Cross-Validation
DTW	Dynamic Time Warping
HMM	Hidden Markov Model
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbours
LSTM	Long Short-Term Memory
LOF	Local Outlier Factor
ML	Machine Learning
NMI	Normalized Mutual Information
OC-SVM	One-Class Support Vector Machine
ROCKET	RandOm Convolutional KErnel Transform
RF	Random Forest
ROC	Receiver Operating Curve
ResNet	Residual Neural Network
SOM	Self Organizing Maps
SVM	Support Vector Machine
TSF	Time Series Forest
TABL	Temporal Attention-Augmented Bilinear Network

List of Formulas

List of figures

List of tables

1 Introduction

In today's manufacturing industry, generating and storing vast amounts of data have become integral to all operations. With advancements in data collection technologies, such as sensors and IoT devices, industrial processes now produce extensive datasets that capture various parameters at different stages of production. As a result, manual validation and analysis are no longer feasible and become more complicated in the case of single or multivariable analysis. Hence, automated analysis and the use of industry-standard methods for data analysis offer deep insights into manufacturing processes, leading to improvements in efficiency, quality, and cost-effectiveness.

Detecting anomalies in the manufacturing industry is essential. Spotting deviations from standard operations guarantees the reliability and safety of products, avoids expensive downtimes, and preserves the integrity of the production process.

In the case of the automated screw-tightening process, screws create connections between two components. The quality of these connections depends on both the strength of the screws and the efficiency of the tightening process. Improperly tightened screws can result in damaged products and various issues when incorporated into the final product. Therefore, anomaly detection in automated screw tightening is essential to improve the reliability and safety of the end products. Sensors record the screw-tightening process as univariate or multivariate time series, and machine learning methods can accurately detect anomalies using this data.

Training supervised models requires a large amount of labeled data, which is generally not available in most of the industrial use cases. This thesis focuses on investigating and implementing semi-supervised and unsupervised approaches for anomaly detection.

1.1 Objective of the thesis

The primary objective of this thesis is to research and implement unsupervised approaches to cluster different anomalies in the screw-tightening process and map the clusters to specific anomaly categories in a labeled dataset. Additionally, it aims to build a general machine-learning prediction pipeline to determine if the process is anomalous by analyzing the torque sequence of the screw-tightening run. If an anomaly is detected, the system will inform the user of the type of anomaly and the probability that the anomaly belongs to that type through a web interface.

To achieve this, a univariate time series (torque data) is utilized. First, a semi-supervised approach is used to predict whether a screw-tightening run is anomalous or not. Given that most industrial data is unlabeled, this thesis explores the effectiveness of unsupervised approaches in identifying different types of anomalies. Finally, the results are compared with supervised approaches to highlight the advantages and limitations of unsupervised methods.

1.2 Structure of thesis

This thesis report aims to explain the research and analysis performed on anomaly detection in the screw-tightening process in detail. Including the introduction, this report consists of 8 sections in total. In section 2, the fundamental concepts related to time series, anomaly, and their detection methods are explained in detail. In addition, this section also provides a good overview of the need for anomaly detection methods in the manufacturing industry and the screw tightening process. The existing literature and research performed in the screw-tightening anomaly detection process are stated in section 3. This provides a foundation for further research performed in this thesis.

Section 4 consists of all the statistical models used for anomaly detection in this thesis. They are explained elaborately with formulas, pseudo-code, and hyperparameters. Moreover, evaluation metrics used to check the performance of models are also described in detail. In section 5, the anomaly detection architecture used in this thesis is explained using a flow chart diagram. Next, it provides an overview of the dataset used and explains different errors that are dealt with. Additionally, preliminary hypotheses are created based on the data analysis and explained using graphs.

In section 6, all the statistical methods explained in section 4 are applied to the screw-tightening data, and the findings are interpreted using appropriate visualizations. The study of this thesis is concluded in section 7 by summarizing all the key findings. This section details the advantages and constraints of the methods in unsupervised anomaly detection of time series. Moreover, it also provides some insights that can be carried out for future work. Finally, section 8 consists of a list of references for the articles and books used for research during this thesis.

2 Fundamentals

In this section, some fundamental concepts related to anomalies, time series, and the manufacturing industry are discussed. These concepts are essential for understanding the topic of the thesis and the motivation behind the need for anomaly detection in the manufacturing industry. The section begins by explaining the different types of anomalies in time series data and details the various anomaly detection methods available. Subsection 2.5 covers industrial applications of anomaly detection with examples. Additionally, the general procedure for anomaly detection is outlined, and the phases or steps in the screw-tightening process are explained.

2.1 Time series data

A sequence of data points measured at regular intervals of time is called time series data. It is chronologically ordered data recorded at discrete time stamps. Examples of time series data include temperature recorded every day, sensor measurements recorded every second of the

manufacturing process, etc. If only a single variable value is recorded at each time stamp, then it is called univariate time series, and if multiple variable values are recorded, then it is called multivariate (Haben et al., 2023, p. 55-66).

2.2 Outlier vs Anomaly

Outlier

Data points that deviate from normal data points and raise suspicion that they may follow a different distribution or may be generated by a different mechanism are called outliers (Olteanu et al., 2023). It can also be defined as a data point that is far away from the mean or median values of the dataset.

Anomaly

A data point that significantly deviates from normal data points can be defined as an anomaly. In the field of time series, an anomaly can be defined as a point or group of points that show a different pattern or behavior when compared to the data points from previous time stamps and are considered to be rare.

The line plots in figure 1 explain the difference between an outlier and an anomaly in time series data. Each plot consists of amplitude values (univariate time series) plotted in gray against time. The normal range of amplitude values falls within the blue-shaded region. In the left plot, the green point deviates from the normal data points but remains within the valid range of amplitude values. Therefore, it creates a suspicion of being an anomaly point but is considered an outlier. In the right plot, the red data point significantly deviates from the normal amplitude value range, following a different data distribution. Hence, it is considered an anomaly or point anomaly in time series (Choi et al., 2021).

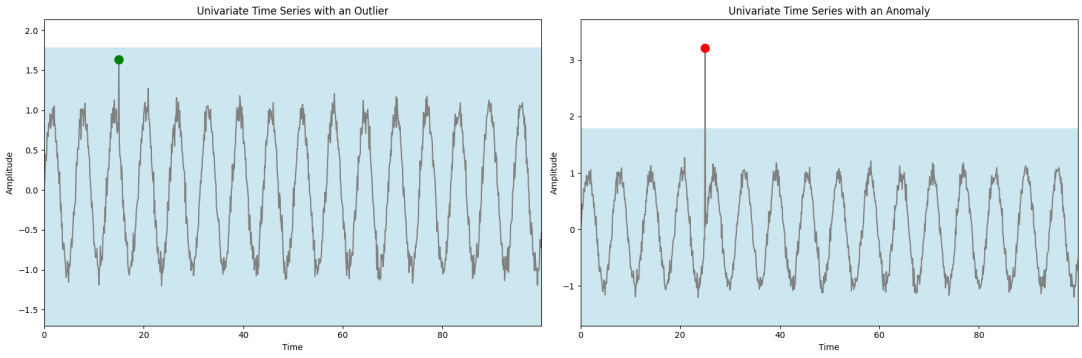


Figure 1: Outlier vs Anomaly.

2.3 Types of anomalies

In the time series domain, anomalies can be classified into 3 categories based on the different kinds of abnormality in the data.

Point anomaly: A data point that abruptly deviates from the normal data sequence or the norm can be termed a point anomaly. They appear in the form of temporal noise in the dataset.

Contextual anomaly: A short sequence of data points that has a different shape or pattern when compared to a normal sequence is termed a contextual anomaly. They are similar to the point anomaly but in this case, a short sequence of values doesn't deviate from the regular interval of time series. As a result, they are difficult to identify.

Collective anomaly: A collection of data points that show different patterns when compared to normal sequences over a period of time is called collective anomaly. These points are considered non-anomalous when looked at one specific time point but it is considered as an anomaly when looked at as a whole series or collection of points.

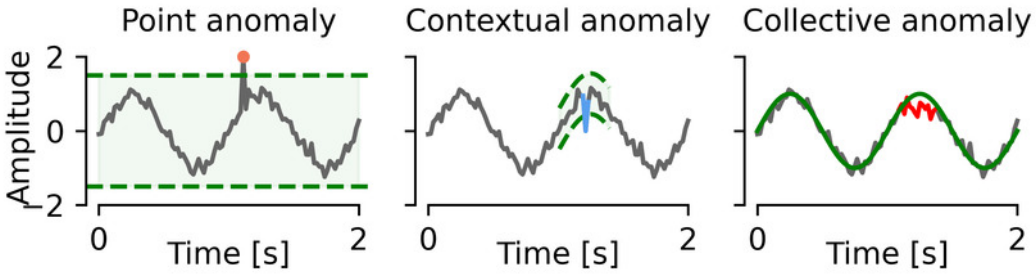


Figure 2: Different types of anomalies (Yan et al., 2023).

Figure 2 consists of 3 graphs, each graph is a line plot of amplitude values over time. In the left line plot, the green dotted lines on the top and bottom represent the range for normal amplitude values. It can be noticed that a single red point is significantly different from all the other points and it is outside of the amplitude range. This is an example of a point anomaly. The line plot in the middle is an example of contextual anomaly. A short sequence in blue color is significantly different from the normal pattern and it is in the same range.

Moreover, the line plot on the right is an example of collective anomaly. A red sequence can be observed in the graph which deviates from the normal sequence for a longer period (Choi et al., 2021).

To explain the definitions formally, let $X = \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \mathbb{R}^d$, $i \in \{1, 2, \ldots, n\}$. The variable n denotes the number of observations in the dataset, and d denotes the number of dimensions in the dataset. Point $x \in X$ or a subset of points $X_{\text{sub}} \subset X$ that are different from other data points in X are called anomalies. The methods used to detect X_{sub} from X are called anomaly detection methods (Huang, 2018, p. 1-2).

2.4 Types of anomaly detection methods

Machine learning for anomaly detection offers 3 different techniques, namely, supervised, semi-supervised, and unsupervised approaches. Each of these methods has its advantages and disadvantages. They can be noticed in figure 3 and also explained in detail below:

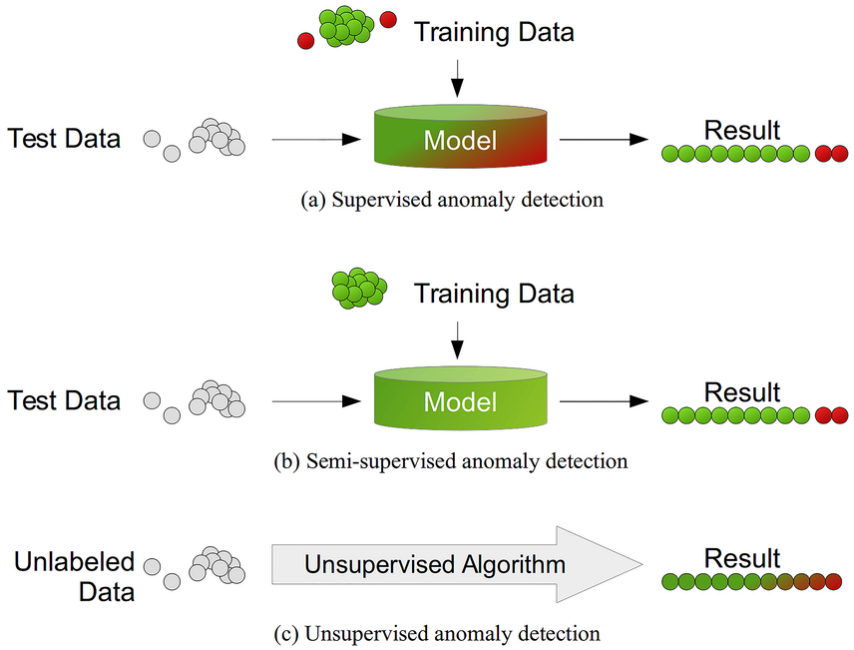


Figure 3: Different types of anomaly detection methods (Goldstein and Uchida, 2016).

Supervised anomaly detection: Supervised methods assume that the dataset is labeled. The dataset is split into training and test sets. Using the training set, general classification models like decision trees, multi-layer perceptron, etc are built to understand the patterns in the normal and anomaly data sequences. As the anomaly sequences are labeled, the classifiers can capture the anomaly patterns easily. Hence, models perform well on the test data. However, as the anomaly sequences are rare events, all the real-world datasets will have a high data imbalance between the classes. This can impact the performance of the classification models and hence supervised approaches are not very feasible for the anomaly detection process. Also, in most cases, the labels are not available in the real world and it can be too expensive or impossible to create the labeled datasets.

Semi-supervised anomaly detection: Similar to supervised methods, semi-supervised methods also contain training and test datasets, but, in this process, only normal instances of data are used for training the models. The basic idea of this approach is to learn the distribution of normal sequences and hence when using the test data set, the observations that deviate from the normal sequences are classified as anomalies. It can also be called a one-class classification approach. One-class SVM and autoenoders are some examples of this approach.

Unsupervised anomaly detection: In general, the most feasible solution for anomaly detection is using unsupervised approaches because it doesn't require any labels for the dataset. As a result, the training and test datasets are also not required. Several clustering approaches are

used to learn the distributions of the data or dissimilarities between the data points. However, as the prior information about the anomalies are unknown, modeling requires more effort and the results are less accurate when compared to the supervised methods.

In this thesis, semi-supervised and unsupervised methods are explored in detail to detect the anomalies and cluster different kinds of anomalies. Also, supervised models are used to compare the results from the unsupervised models (Goldstein and Uchida, 2016).

2.5 Industry applications of anomaly detection

Anomaly detection is used in different fields to find errors, spot unusual trends that could lead to other potential problems, and enhance current methods. The important advantages of identifying anomalies early include reduced costs associated with failures by preventing them beforehand, increased operational efficiency, and safety. Some examples of domains using anomaly detection are mentioned below.

Financial Sector

Finding anomalies is essential to exposing fraudulent activity in the financial sector. Credit card issuers and banks monitor the spending patterns of their customers to spot anomalies like irregular transactions or sudden spikes. These anomalies are flagged for further investigation to safeguard against fraud and protect the financial assets of the customers.

Cybersecurity

Anomaly detection is essential for network traffic monitoring and detecting possible incursions that might indicate cyber-attacks or other risks in the field of cybersecurity. It helps detect illegal access to systems or databases, and hence immediate action can be taken to prevent data breaches or the loss of sensitive information. Furthermore, constant network monitoring lowers downtime and aids in error detection, enhancing overall network security.

Healthcare

In the healthcare sector, anomaly detection is used to monitor patient health data for early signs of problems. By examining data from medical devices, healthcare professionals can spot irregular heartbeats or sudden drops in oxygen levels and respond swiftly. This proactive method improves patient care and safety by ensuring timely medical interventions when necessary.

Manufacturing Industry

In the manufacturing industry, sensors fitted to equipment collect data at regular intervals. Analyzing this data using anomaly detection techniques helps predict machine failures in advance and prevent the production of defective products. It also indicates when maintenance is needed, preventing device breakdowns and avoiding significant costs. Overall, anomaly detection improves the manufacturing process and ensures product quality (Blessing and Klaus, 2023).

This thesis focuses on anomaly detection in the manufacturing industry, specifically detecting anomalies in the automatic screw-tightening process. Since thousands of screws might be fixed automatically, identifying faulty screw runs is crucial to avoid errors during testing and save potential costs and additional work. By employing anomaly detection, manufacturers can ensure the reliability and efficiency of the screw-tightening process.

2.6 Need for anomaly detection in Manufacturing industry

Machines in the manufacturing industry run throughout the day to produce/fix different components. Over time, these machines need a maintenance check to ensure that they are functioning well. Inadequate maintenance checks can affect the overall productivity of the machines, and research shows that manufacturing industry organizations have to pay significant costs every year for unplanned halts or defects that lead to machine downtime. The different maintenance procedures that have evolved over time are explained below:

- **Reactive maintenance** – In this process, the machine is repaired after the failure has occurred.
- **Preventative maintenance** – In this process, regular maintenance is carried out to avoid failures. However, in regular maintenance, components are replaced periodically even if it is not actually needed at that time.
- **Rule-based predictive maintenance** – In this process, maintenance is carried out based on the alerts sent from hard coded threshold rules. Every time a measurement crosses the threshold, maintenance is carried out.
- **Machine Learning (ML) based predictive maintenance** – In this process, data analysis and machine learning are used to predict when the next maintenance is needed for the equipment. It can also be used to analyze the data from each fitting/manufacturing process and identify if the process is good or bad.

Hence, using ML-based methods for predictive maintenance can avoid unplanned downtime, increase productivity, optimize resources, and also increase customer satisfaction.

As the internet of things has increased significantly over recent years, it is used to enable communication between machines through the internet in real-time. Also, with the availability of sensors and data storage at cheaper costs, all devices in the manufacturing industry can be equipped with sensors to collect data periodically. This data can be used for ML based predictive maintenance (Kamat and Sugandhi, 2020). Hence, anomaly detection methods are important to overcome these problems in the manufacturing industry.

2.6.1 General anomaly detection procedure

In this section, the general steps taken for the anomaly detection process in the manufacturing industry are explained. The process consists of seven steps, detailed below:

- **Data gathering** – Relevant data from sensors and other devices are collected based on the task and stored in a database.
- **Data cleaning** – As raw data from multiple sources may contain noise and have different structures, it should be pre-processed into a consistent format. This step also handles missing and duplicate values and also outliers. Finally, the data is normalized to a single scale.
- **Feature extraction** – Depending on the methods to be applied, identify and extract features useful for spotting anomalies. If the dataset contains many features, dimensionality reduction methods may also be applied.
- **Model selection** – Based on the data, choose from a variety of anomaly detection methods from different families. This approach helps experiment with multiple models, avoiding bias and random selection of a specific model.
- **Model training and evaluation** – Train the models using the pre-processed data and record the evaluation metrics of each model. Select the best model by comparing the relevant metrics to determine which one outperforms the others.
- **Deployment** – Deploy the best model selected in step 5 to detect anomalies in the manufacturing process in real-time.
- **Monitoring and maintenance** – Regularly monitor the model to assess its performance and ensure the results are accurate. If there are any irregularities or data drifts, retrain and redeploy the model as needed.

(Srivastava and Bhambhu, 2023).

2.7 Different steps in screw tightening process

Different articles mention different processes in the screw-tightening. This subsection explains one such general process involved in screw tightening. This process is divided into three phases: screw seating, clamping, and screw stripping.

In **Phase I (screw seating)**, the screw is inserted into its base component by progressively increasing the screw-driving torque to overcome friction. The screw is considered properly seated when it reaches the set value of the seating torque.

In **phase II (clamping)**, the torque is sharply increased to tighten the screw. It is increased until it reaches the set value of the stripping torque.

In **phase III (screw stripping)**, the torque values are gradually reduced after reaching the stripping torque value. In this phase, the screw-driver is detached from the screw head, and the screw is properly fixed into the base component (Tor et al., 2020).

In this thesis, the screw-tightening process is considered to have 4 phases and these phases are explained in detail in sub-section 5.3.

3 Related work

Over the years, advancements in the manufacturing industry have led to the investigation and use of anomaly detection methods. The following literature provides a good overview of existing methods for anomaly detection in the automated screw-tightening process.

Ribeiro et al. (2021) compares various methods to identify abnormal screw-tightening processes by using angle-torque pairs. The research uses Local Outlier Factor (LOF), Isolation Forest, and deep learning autoencoder models from the unsupervised domain, as well as the Random Forest (RF) model from the supervised domain, to determine whether the screw-tightening process is anomalous or not. A threshold is used to separate the anomalies when using Isolation Forest and autoencoder models.

Yuki et al. (2023) proposed a method to detect screw-tightening defects in real-time. This approach uses position, velocity, and torque data from the R and Z axes as the dataset. A feature extraction method is used to train the Isolation Forest model using normal instances.

Cao et al. (2019) proposed a method based on an Long Short-Term Memory (LSTM) network that can automatically analyze the quality of screw-tightening. First, the dataset is labeled into four categories by identifying different patterns in the angle-torque curve. Then, the LSTM model is trained to learn patterns from each class. This process follows a supervised approach, and the results are compared with machine learning algorithms such as Support Vector Machine (SVM) and Random Forest.

West et al. (2023) used K-means algorithm to distinguish screw-tightening anomalies from normal ones. This paper addresses the anomaly detection issue using unsupervised methods, even though true labels are available. As a result, it avoids the manual effort required to generate labels and is useful for detecting errors that have never been seen before. The clusters created are assigned to normal or abnormal classes based on a set of rules.

West and Deuse (2024) also performs a comparative study of machine learning approaches for anomaly detection in screw-tightening. In this research, several machine learning algorithms from both supervised and unsupervised families are applied to identify anomalous screw-tightening runs. By comparing the results from both families, it is shown that unsupervised approaches can effectively detect both known and unknown anomalies.

Leporowski et al. (2021) proposed an application of time series classification models for anomaly detection in automatic screw driving. In this research, the machine learning algorithms Temporal

Attention-Augmented Bilinear Network (TABL) and Residual Neural Network (ResNet) are used on publicly available screw-tightening datasets. The dataset contains four different kinds of labeled anomalies. Experiments include both binary and multi-class classification.

In above mentioned research methods, anomaly detection in the screw-tightening process is addressed as a binary classification problem (anomaly/not an anomaly). Many methods from supervised, semi-supervised, and unsupervised domains are proposed and are in use. In the case of multi-class classification, or distinguishing between different anomalies, research has been conducted to a certain extent using supervised approaches but not using unsupervised approaches. As mentioned, most real-world data doesn't contain labels, so an efficient unsupervised method is required for anomaly detection in the automated screw-tightening process. This thesis focuses further on binary classification (anomaly/not an anomaly) and, for distinguishing different kinds of anomalies, an in-depth study in the unsupervised domain specifically using clustering methods are carried out.

4 Methods

In this section, all the statistical models and their evaluation metrics used in this thesis are explained in detail along with their mathematical notations. The data analysis and modeling code is written in Python language, software version 3.10.0 (Python Core Team, 2024).

4.1 Mann Whitney U-test

Mann Whitney U-test is a non-parametric test that aims to determine whether the data from two samples have a significant difference in their distributions. Unlike parametric tests like the t-test, it doesn't assume the data to follow a normal distribution. Hence, it is mainly used to compare two groups of non-normal distributions. If the two samples are of different sizes then Mann Whitney test is used to compare their median values.

The null hypothesis (H_0) assumes that there is no significant difference between the samples of the two distributions, and the alternative hypothesis (H_a) assumes that there is a significant difference between the samples of the two distributions.

Let $X = X_1, X_2, \ldots, X_n$ be a sample from one distribution and $Y = Y_1, Y_2, \ldots, Y_m$ be a sample from another distribution. N denotes the total number of observations from both samples. The formula to calculate Mann Whitney U-test statistic can be stated as follows:

$$U = mn + \frac{n(n+1)}{2} - T, \tag{1}$$

Where:

- U – represents the Mann Whitney test statistic.

- n – denotes the size of the first sample.
- m – denotes the size of the second sample.
- T - denotes the sum of ranks from the first sample (X).

This test statistic U computes the total number of times that an X_i precedes a Y_j in the classification in increasing order of the $(n+m)$ observations. For a test statistic, a corresponding p-value can be obtained from the table and if the p-value is less than the significance level ($\alpha = 0.05$) then the null hypothesis is rejected indicating the samples of 2 distributions are different (Dodge, 2008, p. 327-329).

4.2 Min-max normalization

A method used to transform the values of several features in a dataset to a common fixed scale is called min-max normalization. Using the transformed values for modeling or distance calculation ensures that each feature contributes equally. This linear transformation scales the data to the range [0, 1]. Additionally, min-max normalization preserves the relationships among the original values. The formula below is used for min-max transformation.

$$x_{sclaed} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{2}$$

Where:

- x_{scaled} – represents the transformed value of the original data point x.
- x_{min} – denotes the minimum value of a feature in the dataset.
- x_{max} – denotes the maximum values of a feature in the dataset.

(Ciaburro et al., 2018).

4.3 Linear interpolation

A method used to fill missing values between any two data points is called interpolation. In the case of linear interpolation, it uses a straight line to find the missing values by connecting two known data points. The missing values are estimated from the straight line created. This is well-suited for handling missing values in linear time series data. In the case of non-linear time series, it breaks the seasonal patterns. To estimate the missing values in a uni-variate time series sequence $x_1, \ldots, x_n$ using linear interpolation, the following formula can be used.

$$x_i = x_{i-1} + \frac{(x_{i+1} - x_{i-1})}{(t_{i+1} - t_{i-1})} \cdot (t - t_i), \tag{3}$$

Where:

- x_i - denotes the interpolated value at time t_i.
- x_{i-1} and x_{i+1} - denotes the known values before and after the missing value.
- t_i - denotes the timestamp of the missing value at index i.
- t_{i-1} and t_{i+1} - denotes the timestamp of known values before and after the missing value.

(Chapra and Canale, 2002, p. 491).

4.4 Ensemble learning

The process of creating multiple base learners and combining the predictions of these individual learners to create one final prediction is called ensemble learning. The underlying idea of this method is to overcome the limitations of individual models and improve the overall performance of the model. Bagging (bootstrapping) is a method of ensemble learning that combines the predictions of multiple learners. In this case, each base learner is trained using a subset of the training data, and their predictions are combined using strategies such as voting or averaging. It is widely used in solving problems of classification, regression, clustering, and anomaly detection (Jo, 2023, p. 83-109).

4.4.1 Weighted ensemble learning

In this ensemble learning strategy, each base learner or model prediction is multiplied by a specific weight and then added together to make the final prediction. The idea behind this approach is that it is valid to give higher importance to the models that have more predictive power. The final output or class label can be calculated using the following formula:

$$H(x) = c_{\underset{j}{\arg\max}} \sum_{i=1}^{T} w_i h_i^j(x), \tag{4}$$

Where:

- H(x) - represents the final class label obtained from weighted ensemble.
- h_i - denotes the model or base learner i.
- w_i - denotes the weight assigned for classifier $h(x)$.

The weights are normalized such that each model contributes to assigning the final class label ($w_j >= 0$), and they are constrained by the $\sum_{i=1}^{T} w_i = 1$. The predictions obtained using this strategy are better compared to the predictions of the best individual classifier and the majority voting method (Zhou, 2012, p. 74-75).

4.5 Dynamic Time Warping (DTW)

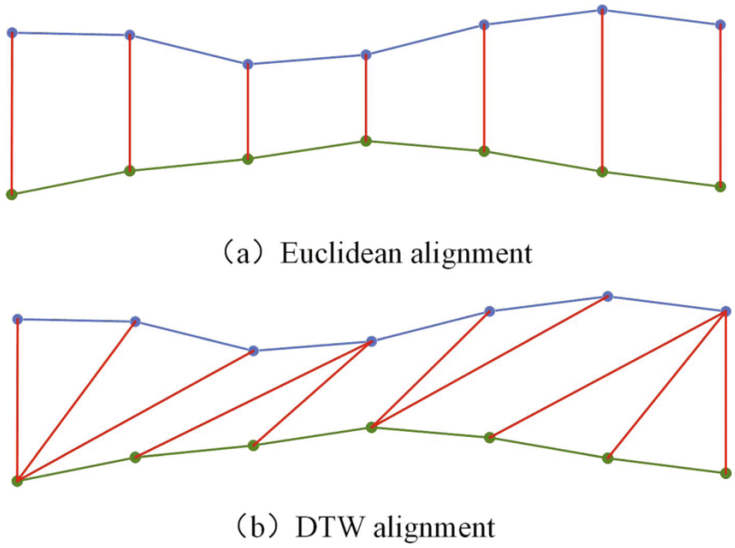


Figure 4: Comparing alignments using Euclidean and DTW. (a) Euclidean alignment and (b) DTW alignment.(Zhang et al., 2023).

One of the methods used to compute the similarity between two sequences is Dynamic Time Warping (DTW). Let $x = [x_1, x_2, ..., x_n]$ and $y = [y_1, y_2, ..., y_n]$ be two sequences. The problem with using general methods like Euclidean distance to compute the similarity between them is that it uses a one-to-one matching of points between the time series. This approach fails if the two sequences are not linearly aligned. DTW, on the other hand, uses a dynamic programming approach to compute the smallest cost between two sequences, aligning them in the best possible way. It can also be used to compare sequences of variable lengths (Lei and Sun, 2007).

The figure 4 shows the difference between sequence alignment using Euclidean distance and DTW. Euclidean distance only computes the distance between pairs of points that are aligned equally, using the formula $L_2(x, y) = \sqrt{(\sum_{i=1}^{n}(x_i - y_i)^2)}$. In contrast, DTW compares the sequences by non-linearly warping one onto the other, addressing problems that arise with time offsets (Zhang et al., 2023).

To determine the DTW distance between x and y, a cost matrix is first initialized using an $m \times n$ matrix. Each cell in the cost matrix represents the difference between two points. The optimal path between the two sequences is then computed using the formula below:

$$D(i,j) = \text{cost}(x_i, y_j) + \min \begin{cases} D_l(i-1, j) \\ D_l(i-1, j-1) \\ D_l(i, j-1) \end{cases}, \tag{5}$$

Here $\text{cost}(x_i, y_j)$ denotes the cost or difference between 2 points and can be computed using $\|x_i - y_i\|^2$ (Lei and Sun, 2007).

4.6 Semi-supervised models

4.6.1 Isolation Forest

In general, all the anomaly detection algorithms try to capture the information from the normal data and classify the new unseen data as an anomaly if their profile doesn't match with the normal data. Isolation forest follows a different approach where it tries to isolate (separate the point from other data points) anomalies, rather than focusing explicitly on normal data. It uses an ensemble approach to create multiple isolation trees for the provided dataset. Let $X = \{x_1, ..., x_n\}$ be a sample data set of n data points. Each isolation tree uses a random attribute q and a split value p to partition the data.

This method uses the known fact that anomalous points are rare and will have different characteristics when compared to the normal data. Each isolation tree repeatedly splits the data until every data point has been isolated. Since anomalies are rare, fewer partitions can be used to isolate them. As a result, these points will be close to the root of the tree and have shorter paths.

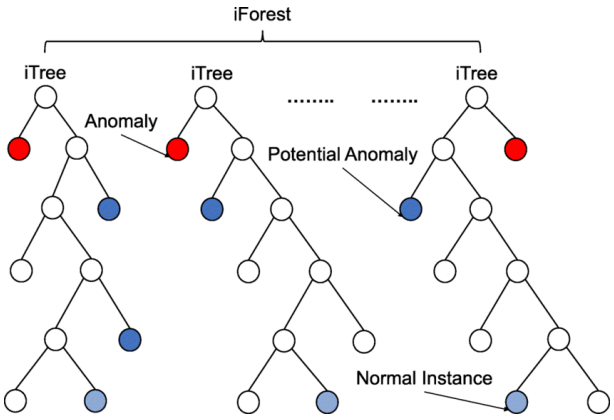


Figure 5: Collection of isolation trees (Regaya et al., 2021).

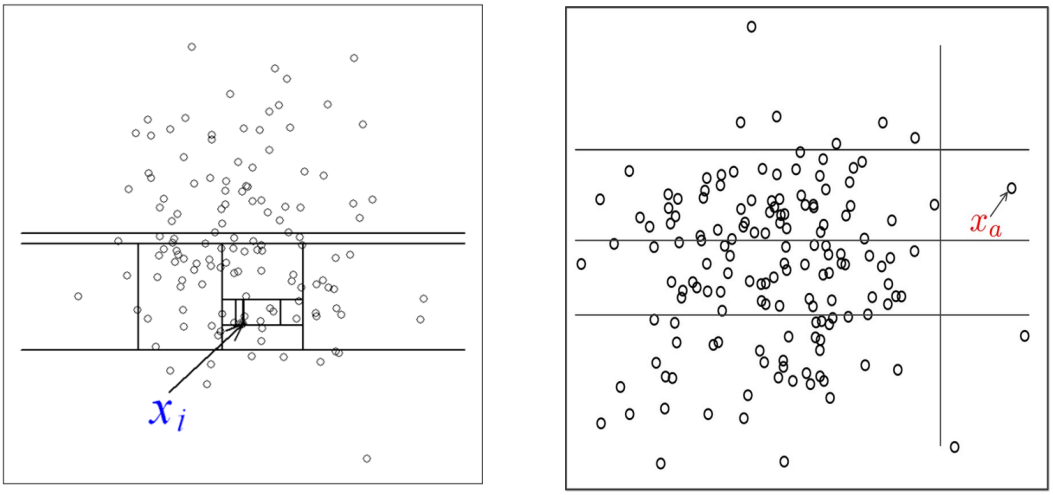
In Figure 5 the recursive partitioning is represented in the tree structure. The path length can be determined as the number of partitions required to separate a point and it is the path from root to the leaf node of a tree. Normal data points will have higher average path length when compared to anomaly points.

Figure 6 shows an example of normal and anomalous data points. In Figure 6(a) x_i is a normal point in the dataset, and the algorithm took many splits to isolate the point x_i, whereas, in Figure 6(b) the anomalous point x_a was isolated in fewer splits.

The anomaly score of a point can be calculated using the formula:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}, \tag{6}$$

where:



(a) Isolating x_i (b) Isolating x_a

Figure 6: Isolation tree data point partition. (a) Normal data point (Galante and Banisch, 2019) and (b) Anomaly data point (Liu et al., 2023).

- s - denotes the anomaly score.
- x - denotes an observation in dataset.
- n - total number of observations in the dataset.
- $h(x)$ - represents the path length.
- $c(n)$ - denotes average $h(x)$ given n. Since iTrees have equivalent structure to BST (Binary Search Trees), the average path length of unsuccessful search in BST is: $c(n) = 2H(n-1) - (2(n-1)/n)$, where $H(i)$ is the harmonic number which can be estimated by $ln(i) + 0.5772156649$.
- $E(h(x))$ - represents the average path length $h(x)$ from collection of isolation trees. In Equation (6),
  - when $E(h(x)) \to c(n), s \to 0.5$;
  - when $E(h(x)) \to 0, s \to 1$;
  - and when $E(h(x)) \to n-1, s \to 0$.

If the anomaly score for a particular data point is very close to 1, it is highly likely to be an anomaly. Conversely, if the score is much smaller than 0.5, the data point is considered normal. Moreover, if the anomaly scores for all points in the dataset are close to 0.5, it indicates that the dataset does not have distinct anomalies.

Steps to build the Isolation Forest model:

- **Inputs**: X – input data, t - number of trees, ψ - sub-sampling size. Based on the sub-sampling size, the height of the tree is estimated.

- The algorithm starts by selecting a random subset of features from the dataset for each isolation tree. At each step of recursive partitioning, a random feature with a split value is used to isolate the data points.
- Calculate the path length for each data point from different isolation trees. Anomaly points are expected to be shorter in length.
- Calculate the anomaly score of each data point. The anomaly scores of all isolated trees are averaged to create the final anomaly score.
- Sort the anomaly scores in descending order to find the top m anomalies in the data or threshold can be used to separate the anomaly points from normal points (Liu et al., 2009, p. 413-422).

The scikit-learn implementation of isolation forest provides multiple hyperparameters that can be experimented with different values and customized to provide the best performance for a specific dataset. *n_estimators* denotes the number of decision trees used in building the isolation forest model. The default value is set to 100. *max_samples* denotes the number of observations to be taken from the dataset to train each base learner. If the value is set to default then *max_samples* is taken as *max_samples = min(256, n_samples)*. Here, *n_samples* is the number of observations in the dataset. The *contamination* parameter is the proportion of anomalies present in the dataset. The value can be assigned in the range (0, 0.5] or it can be assigned *auto* providing flexibility to the model in determining the proportion of anomalies (Adari and Alla, 2024, p. 149-155).

4.6.2 Autoencoder

In the case of ordinary neural networks, the architecture consists of an input layer, an output layer, and one or more hidden layers. The neural network calculates the output as a weighted combination of inputs and activation functions. Backpropagation is used to adjust the weights such that the loss between the actual and predicted values is reduced. An autoencoder is a type of neural network designed to reconstruct the original data. By learning the underlying distribution of normal data, it can detect anomalies, which are data points that deviate from the normal data distribution.

Figure 7 shows the architecture of the Autoencoder. It consists of 3 components, namely, encoder, latent space, and decoder. The encoder part takes the input values and passes through hidden layers to down-sample and create the lower dimensional representation of the data. This representation exists in the form of encoded data in a latent space. The decoder part works in the reverse process by applying up-sampling to the encoded data in hidden layers to re-create the original data. If the architecture of the autoencoder is good, then it reconstructs the normal data accurately (Adari and Alla, 2024, p. 262-263).

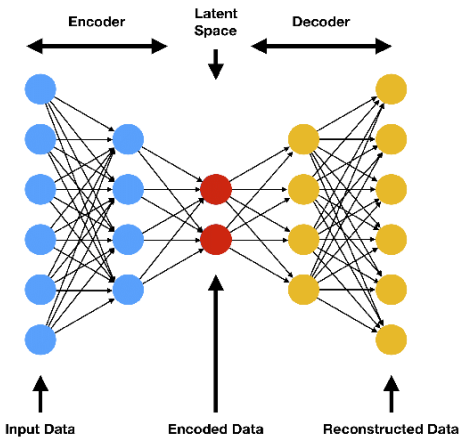


Figure 7: Autoencoder Architecture (Wang et al., 2021).

Consider an input vector, X with dimensions $(d_x, 1)$ where d_x is the number of features in X. In the encoder block, the input vector is passed through the sequence of hidden layers to create a latent representation Z of dimensions $(d_z, 1)$, where d_z is the number of features in latent space. In the decoder block, the latent representation (Z) is passed through another sequence of hidden layers to create the output vector X' with the help of the decoder function. The difference between the X and X' is known as reconstruction error (L). It works as a loss function and can be used to optimize the weights of the network. The loss function can be represented using the formula:

$$\min_{\theta} J_{AE}(\theta) = \min_{\theta} \sum_{i=1}^{n} l(x_i, x_i') = \min_{\theta} \sum_{i=1}^{n} l(x_i, g_\theta(f_\theta(x_i))), \tag{7}$$

Where:

- θ - denotes the weights in the hidden layers.
- $J_{AE}(\theta)$ - represents the objective function to reduce the loss.
- x_i - denotes the value of ith dimension in the input data.
- x_i' - denotes the value of ith dimension in the output data.
- l - represents the loss function.
- f_θ - represents the encoder function.
- g_θ - represents the decoder function.
- n - denotes the number of observations in the dataset.

For the input vector X, the encoder and decoder parts can be represented in the matrix notation using the below formulas:

$$\begin{aligned} Z &= f_\theta(X) = s(WX + b), \\ X' &= g_\theta(Z) = s(W'Z + b'), \end{aligned} \tag{8}$$

Where:

- b, b' - denotes the bias vectors.
- W, W' - denotes the weight matrices, with dimensions (d_z, d_x) and (d_x, d_z) respectively.
- s - denotes the non-linear activation function like ReLU or sigmoid.

In equation 7 the loss l denotes reconstruction error of the vector X. It can be calculated using the formula represented in matrix notation below:

$$L(X, X') = \sum_{i=1}^{n} \|X_i - X'_i\|^2, \tag{9}$$

Autoencoders have a wide range of hyperparameters and these values impact the performance of the model. Some parameters are fixed before the training process and others can be changed to improve the model performance. Relevant hyperparameters are explained below:

- **Hidden layers** – The layers between the input layer and the output layer are termed hidden layers, and it also determines the depth of the neural network. In each hidden layer, the input is combined with different weights and bias terms to transform the data into a latent representation. As the number of hidden layers increases, the model will capture the more complex relations in the data but also it will be difficult to train and optimize the model. Also, there is a possibility of overfitting. Hence, it is important to fix the optimal number of hidden layers before training the model.
- **Number of neurons in each layer** – The number of neurons can differ for different hidden layers. In the case of the encoder part, each successive hidden layer will have a decreasing number of neurons and is useful to capture data representation. In the decoder part, each successive hidden layer will have an increasing number of neurons as the model tries to reconstruct the data. More number of neurons can be useful for learning the input data better but it also increases the chances of overfitting.
- **Size of Latent space** - It determines the number of neurons in the latent representation layer to which the input data is compressed. A smaller size of latent space captures only limited information from the data, whereas, a larger size of latent space captures unwanted information (noise).
- **Activation function** - It is used to calculate the output of each neuron, and it introduces non-linearity into the neural network. The activation functions help learn different patterns in the data and can be chosen for each layer before the training. Some of the common activation functions used when training the neural networks are sigmoid, tanh, and ReLU. The sigmoid function transforms the output values in range 0 and 1. The ReLU activation function assigns negative values to 0 and positive values remain unchanged. Tanh function converts the values in the range from -1 to +1.

- **Learning rate** - It determines the proportion of weights and bias values to be updated in the backpropagation and also the speed of convergence of the objective function. The learning rate is assigned a default value for each optimization algorithm and they can be modified during the training to achieve a quicker or better convergence. The lower values of learning rate make the convergence process slower and the objective function to settle at local minima, whereas, the higher learning rate might overshoot and may never find the optimal solution.
- **Number of Epochs** - Passing the entire dataset through the network completes one epoch and hence, the number of epochs decides how many times the model processes and learns from the same dataset. In each epoch, the model tries to capture more information from the data and tries to reduce the error by updating the weights and bias terms. As the number of epochs increases, the model might overfit for the specific dataset. This can be controlled using the early stopping method. The size of the dataset and the task's complexity determines the number of epochs.
- **batch size** - The batch size refers to the number of observations from the dataset used for one forward and backward pass. It determines the count of samples processed by the model before updating the weights and bias terms. Smaller batch sizes save memory and increase speed, but may result in noisy gradient values. On the other hand, if the batch size is large, it might be slow and require more memory, but it leads to more stable gradient values.

(Berahmand et al., 2024, p. 11-15)

4.6.3 One-Class Support Vector Machine (OC-SVM)

The general support vector machine algorithm is a classification model that tries to separate the classes with the help of support vectors and a hyperplane. It belongs to the supervised learning method. OC-SVM is a modified version of SVM, which is a semi-supervised method for identifying anomalies in data. The model is trained only on the normal data points and it tries to create a hyperplane or optimal boundary, which has maximum data points only on one side of the hyperplane. Points that fall on the other side or outside the hyperplane are considered outliers. (Adari and Alla, 2024, p. 159-168)

Consider a dataset with input vector X. OC-SVM maps the data into feature space using a kernel function. If the dataset is completely pure (doesn't contain any anomalies), then OC-SVM creates a hyperplane with a maximum margin where the data is far from the origin.

From figure 8, we can notice two regions, the region marked with +1 are considered as normal data and are located far from the origin. The data points that are outside of this region are treated as anomalies. In general, these anomalous points are near to the origin as shown in the figure and assigned label -1.

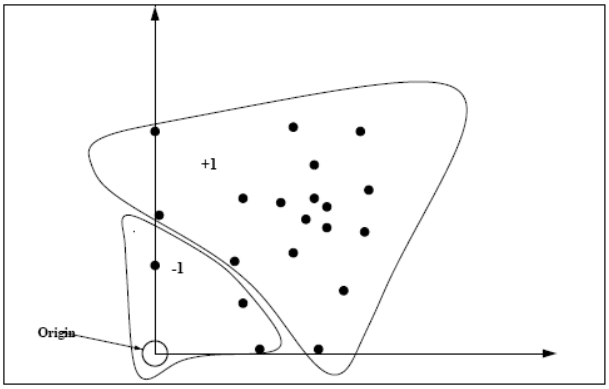


Figure 8: OC-SVM Classifier (Glavin, 2009).

The objective function to determine the optimal hyperplane can be represented as follows:

$$\begin{aligned} \min \frac{1}{2}\|\mathbf{w}\|^2 - \rho, \\ \text{subject to:} \quad \mathbf{w} \cdot \phi(\mathbf{x}_i) \geq \rho, \quad \forall \mathbf{x}_i \in X, \end{aligned} \tag{10}$$

Where:

- ϕ - represents the nonlinear mapping from input space to feature space ($X \to F$).
- $\mathbf{w}$ - denotes the weight vector of the feature space.
- ρ - denotes the offset of hyperplane.
- $\mathbf{x}_i$ - denotes the feature vector of ith observation in the dataset.

If the dataset doesn't contain any anomalies, then the maximum margin hyperplane will satisfy the condition ($\mathbf{w} \cdot \phi(\mathbf{x}_i) = \rho$) and the distance between the origin and hyperplane can be calculated as $\frac{\rho}{\|\mathbf{w}\|}$.

To classify the data points as normal by the model, the mapping function combined with weights should be greater than ρ. As there is a possibility of outliers in the data set, to make the model more robust to them, the smaller values of ($\mathbf{w} \cdot \phi(\mathbf{x}_i)$) should be penalized. This can be achieved by using the slack variable. Hence, the new objective function with constraints can be represented as follows:

$$\begin{aligned} \min \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i, \\ \text{subject to:} \quad \mathbf{w} \cdot \phi(\mathbf{x}_i) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \forall \mathbf{x}_i \in X, \end{aligned} \tag{11}$$

Where:

- ξ_i - denotes the slack variable.
- ν - represents the trade-off between errors and the maximum margin ($\nu \in (0, 1]$).

To solve the above constrained objective function, a lagrangian function is introduced as follows:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \quad - \sum_{i=1}^{n}\alpha_i[(\mathbf{w}\cdot\phi(\mathbf{x}_i)) - \rho + \xi_i] - \sum_{i=1}^{n}\beta_i\xi_i, \tag{12}$$

Here, α_i and β_i are called lagrange multipliers, and their values should be greater than or equal to 0. The above equation is maximized with respect to $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ and minimized with respect to $\mathbf{w}$, $\boldsymbol{\xi_i}$ and ρ for the given values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

Using the Karush-Kuhn-Tucker conditions, the variables $\mathbf{w}$, ξ_i, ρ, and β_i can be dropped, so the Lagrangian equations can be written in Wolfe dual form. The Wolfe dual form is expressed as a quadratic function of $\alpha_i's$. To avoid high dimensional feature space F, a feature space is chosen where the dot product can be computed directly using kernel K in the input space. The The Wolfe dual problem can be expressed as the following equation:

$$\begin{aligned} &\min \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j), \\ \text{subject to:}\quad &0 \le \alpha_i \le \frac{1}{\nu n}, \quad \sum_{i=1}^{n}\alpha_i = 1, \end{aligned} \tag{13}$$

Once the above-constrained optimization problem is solved, the decision function for any data point can be calculated using the formula below:

$$f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}\cdot\phi(\mathbf{x}) - \rho) = \operatorname{sgn}\left(\sum_{i=1}^{n}\alpha_i K(\mathbf{x}_i,\mathbf{x}) - \sum_{i=1}^{n}\alpha_i K(\mathbf{x}_i,\mathbf{x}_j)\right), \tag{14}$$

While training the model many kernel functions (K) can be used. Some of the common kernel functions are sigmoid, polynomial, and radial basis kernel (Liu et al., 2007, p. 516-518).

nu and *gamma* are 2 other hyperparameters that can be adjusted during the training of OC-SVM using scikit-learn. *nu* parameter is used to set the lower bound for the number of support vectors and the upper bound for the proportion of training errors. The value of *nu* is in the range (0, 1] and it is used to set the trade-off between the number of support vectors and the number of training errors. The smaller value of *nu* makes the model strict as it allows only a few training errors and support vectors, whereas, the larger value of *nu* makes the model a bit lenient as the model uses more support vectors and allows more training errors. This value of this parameter can significantly affect the results of the model and needs to be chosen carefully. It is recommended to retune the parameter again when the size of the training data changes.

The *gamma* parameter is specifically associated with the Radial Basis Kernel (RBF) function. It determines the amount of influence one training sample has on another. The smaller value of *gamma* indicates less influence and points are far from each other. This can result in a smoother decision boundary, but it may not capture all the information from the data to separate anomalies. A larger value of *gamma* indicates a higher influence between the points and can lead to a wiggly decision boundary. In this case, the model captures the complex patterns in

data along with the noise, which leads to overfitting (Adari and Alla, 2024, p. 175). Scikit-learn provides 3 choices for this parameter:

- *scale* - this is the default option, which uses the formula $\frac{1}{n\_features \times X.var()}$ to calculate the value of *gamma*.
- *auto* - the *gamma* value is calculated as $\frac{1}{n\_features}$.
- nu - floating point values greater than 0 can be manually assigned before training.

4.7 Supervised models

4.7.1 ROCKET classifier

RandOm Convolutional KErnel Transform (ROCKET) is designed using a huge number of random convolutional kernels with weights, bias, dilation, and padding to extract relevant features. These features are used to train a linear classifier to achieve state-of-art accuracy in time series classification.

Kernel

The kernel contains random weights in a vector. As the weights are sampled from a normal distribution ($w \sim \mathcal{N}(0, 1)$), most of the weights are small. The kernel size used (chosen randomly from the numbers 7, 9, and 11) is smaller than input size of the time series but has the same dimensions (uni-variate or multi-variate time series) as the input. The difference between the random convolution and normal convolution is that in the random convolution kernel, the weights are not typically learned. Each random convolutional kernel is combined with input time series using the dot product to extract features and identify different patterns in time series and added to the bias term. The bias term is sampled from a uniform distribution ($b \sim \mathcal{U}(-1, 1)$).

Dilation (d) refers to the gaps between the time series when convolved with the kernel. In the case of dilation one, all data point in the time series is convolved with the kernel whereas in the case of dilation two, every second data point is used for convolution operation. The dilation value is sampled using an exponential scale ($d = \lfloor 2^x \rfloor, x \sim \mathcal{U}(0, A)$), where $A = \log_2\left(\frac{l_{\text{input}}-1}{l_{\text{kernel}}-1}\right)$. Here l denotes the length.

The formula below is used to represent the convolutional operation applied to the time series:

$$X_i * \omega = \left(\sum_{j=0}^{l_{\text{kernel}}-1} X_{i+(j \times d)} \times \omega_j \right) + b, \tag{15}$$

Where:

- X_i - denotes a uni-variate time series at a particular point i.
- ω - denotes the weights in the kernel.

- l_{kernel} - represents the length of the kernel.
- b - denotes the bias term.
- $\times$ - represents the dot product operation.

The padding technique is used to append zeros at the start and end to ensure that the length of the convolutional output is the same as the input. The value of stride is always set to 1. Moreover, no non-linearity functions are applied to the results of the convolutions.

Feature extraction

From each feature map obtained from the convolutions, the ROCKET classifier calculates 2 features, namely, the maximum value and proportion of positive values (*ppv*). The maximum value is nothing but the global max pooling and is used to get the maximum feature value from the feature map.

The (*ppv*) is used to determine the proportion of input time series that matches the positive proportion of the output of convolutional operation (given pattern). The bias term acts as a threshold for calculating the *ppv* feature. The positive value of the bias term is highly relevant for the *ppv*. If the bias value is positive, even the weak matches between the input and the given pattern are captured by *ppv*, whereas, if the bias value is negative, only the strong matches between the input and the given pattern are captured. The *ppv* feature has more weightage in improving the accuracy of the classifier when compared to other features.

The formula to calculate the *ppv* can be represented as follows:

$$ppv(Z) = \frac{1}{n}\sum_{i=0}^{n-1}[z_i > 0], \tag{16}$$

Where:

- Z - represents the output of the convolutional operation.
- z_i - denotes the ith element in the output of convolutional operation.
- n - denotes the number of time series.

The features extracted are used to train any classifier but linear classifiers like logistic regression or ridge regression classifier are preferred in general. The logistic regression is suitable for very large datasets and the ridge regression classifier can be used on any dataset. In the case of the ridge regression classifier, the model is trained on each class using the one vs rest approach.

Even though ROCKET uses convolutional kernels as a neural network, it is different from them as it doesn't use hidden layers and the features extracted from each kernel are independent from each other. The ROCKET classifier implementation in *sktime* provides some hyperparameters: *num_kernels* are used to set the number of convolutional kernels to be used while training the classifier. The library has a default value of 10,000 kernels. *max_dilations_per_kernel* is used to set the maximum number of dilations per kernel (Dempster et al., 2019, p. 1-9).

4.7.2 Catch22 Features

The process aims to select the best 22 features out of 4791 features of the highly comparative time series analysis (hctsa) library based on their performance in 93 time series classification tasks. The selection procedure consists of 3 steps, namely, statistical prefiltering, performance filtering, and redundancy minimization. In the statistical prefiltering step, the features are evaluated based on consistent performance on a set of classification tasks. For the classification tasks, null accuracy distributions are generated using a permutation-based procedure. Next, the classification is performed on randomly shuffled class labels, and the p-value is estimated. In the second step, a subset of features is selected from the good-performing features in the first step by assigning ranks based on combined normalized accuracy. The third step aims to reduce the redundancy of features. In this step, a hierarchical clustering is performed using the Pearson correlation distance, and a single feature is selected to represent each cluster. The catch22 features are listed in the tables 60 and 61 in the appendix (Lubba et al., 2019).

4.7.3 Random Forest Classifier

The random forest algorithm is also an ensemble learning method that uses a bagging approach to train multiple decision trees. A random subset of features is selected to construct each decision tree. At each node, the best feature is selected to split the data into the child nodes. To build a random forest model for the training set (D), the parameters required are the number of base learners to construct (M), and the number of features to consider for each node (Vens, 2013, p. 1812-1813).

Algorithm 1 Random Forest algorithm

for $k = 1$ to the number of base learners (M) **do**
  Select a random subset of data (D_k) from the training set (D) using bootstrapping
  Build a base learner (h_k) using random feature selection (D_k, F)
end for
return final prediction by aggregating the predictions of the base learners ($\cup\, h_k$).

As the number of trees increase the predictive performance of the model and hence, there might not be a possibility of overfitting to the training dataset. The random forest algorithm also provides a list of attributes ranked according to their importance in predicting the target variable.

4.7.4 Shape based classifier

A subsequence that captures the local patterns or shapes in the time series data is termed a shaplet. The idea behind the process is that each class of time series will have a unique shape that distinguishes it from the other classes and this can be used as features for the classification

algorithm. The Shaplet discovery process consists of 3 steps, namely, candidate generation, shaplet distance calculation, and shaplet assessment.

In the first step, $(m-l)+1$ subsequences are generated for each time series in the minimum and maximum value of the intervals. Here, m denotes the length of time series and l denotes the length of shaplets. These subsequences are normalized and termed as candidates. In the second step, Euclidean distance is used to calculate the distance between 2 subsequences of length l. The minimum distance between a subsequence and all the subsequences of a particular time series (X_i) is defined as the distance between the time series and a subsequence. It can be calculated using the formula below.

$$d_{S,i} = \min_{R \in W_{i,l}} \text{dist}(S, R), \tag{17}$$

Where:

- S - denotes the subsequence of length l.
- $W_{i,l}$ - set of normalised sequences of length l for series X_i.
- R - denotes the subsequence of length l from the set $W_{i,l}$.
- $\text{dist}(S, R)$ - denotes the euclidean distance between S and R.
- $d_{S,i}$ - represents the minimum distance between the S and R in the set $W_{i,l}$.

Using the above procedure, n number of distances are generated between the shaplet S and all the time series in the dataset. In the third step, the shaplet quality is assessed using how well each class can be separated by a set of distances ($D_s =< d_{S,1}, d_{S,2}, ..., d_{S,n} >$). The list D_S is sorted, and split points (sp) are defined as the average of any two consecutive distances in the sorted list. In other words, if the distances in D_S are sorted as $d_{S,1} \leq d_{S,2} \leq ... \leq d_{S,n}$, then each sp is calculated as:

$$sp = \frac{d_{S,i} + d_{S,i+1}}{2}, \text{ for } i = 1, 2, ..., n-1, \tag{18}$$

The information gain metric is used to evaluate the quality of shaplets. Information gain is calculated for each split point in the sorted list of D_S using the formula:

$$IG(D_S, sp) = H(D_S) - \left(\frac{|A_S|}{|D_S|} H(A_S) + \frac{|B_S|}{|D_S|} H(B_S) \right), \tag{19}$$

Where:

- A_S - denotes the elements with $(D_S < sp)$.
- B_S - denotes the elements with $(D_S > sp)$.
- $H(D_S), H(A_S), H(B_S)$ - represents the entropy of D_S, A_S and B_S respectively.
- $|D_S|, |A_S|, |B_S|$ - denotes the cardinality of D_S, A_S and B_S respectively.

- $IG(D_S, sp)$ - represents the information gain from a particular split point sp.

The final information gain of shaplet S is calculated using the formula:

$$IG_S = \max_{sp \in D_S} IG(D_S, sp), \tag{20}$$

(Hills et al., 2013).

4.7.5 Time Series Forest (TSF) classifier

The TSF algorithm uses a tree ensemble approach similar to the random forest algorithm but it is specifically designed for time series classification. It uses a combination of entropy gain and distance measure (Entrance) to perform the splitting process.

Let $\{f_k^n(t_1, t_2), n \in 1, 2, ..., N\}$ denote the set of features for each node of a tree. The algorithm starts by calculating the features mean, standard deviation, and slope for a random set of intervals in the time series. The above-mentioned features can be calculated using the formulas below:

$$f_1(t_1, t_2) = \frac{\sum_{i=t_1}^{t_2} v_i}{t_2 - t_1 + 1}, \tag{21}$$

$$f_2(t_1, t_2) = \begin{cases} \sqrt{\frac{\sum_{i=t_1}^{t_2} (v_i - f_1(t_1,t_2))^2}{t_2 - t_1}} & \text{if } t_2 > t_1 \\ 0 & \text{if } t_2 = t_1 \end{cases}, \tag{22}$$

$$f_3(t_1, t_2) = \begin{cases} \hat{\beta} & \text{if } t_2 > t_1 \\ 0 & \text{if } t_2 = t_1 \end{cases}, \tag{23}$$

In the above formulas, t_1 and t_2 represent the start and end points of the intervals. f_1, f_2, and f_3 denote the function to calculate the mean, standard deviation, and slope between the intervals t_1 and t_2. $\hat{\beta}$ is the slope. v_i represents the value of the time series at point i.

Each node in the tree checks the condition $(f_k(t_1, t_2) \leq \tau)$ and it is used to split the node. Here, t represents the threshold value. The best threshold value for a feature f_k is obtained from the candidate threshold values range $\left[\min_{n=1}^{N}\left(f_k^n(t_1, t_2)\right), \max_{n=1}^{N}\left(f_k^n(t_1, t_2)\right)\right]$. The interval between the candidates is equally spaced and the number of candidates for the threshold is fixed and denoted by k.

The feature values that are less than or equal to the best threshold are placed in the left child node of a tree and the values greater than the best threshold are placed in the right child node of the tree.

In general, entropy is used to determine the best split, but in the time series data there is a possibility of having a large number of candidate splits, and most of the candidate splits might

have the same entropy. Hence, an additional measure is required to determine the best split. TSF uses Margin (it is the distance between the candidate threshold and the nearest feature value). The Margin value for the particular feature split ($f_k(t_1, t_2) \leq \tau$) is calculated using the below formula:

$$\text{Margin} = \min_{n=1,2,\ldots,N} |f_k^n(t_1, t_2) - \tau|, \tag{24}$$

In the above equation, n denotes the nth instance of the node. Hence, the new splitting criteria is a combination of entropy gain ΔEntropy and Margin. It can be termed as Entrance gain ($E = \Delta\text{Entropy} + \alpha \cdot \text{Margin}$).

Here, ΔE is the entropy gain and the α value is used to break ties in the case of equal entropy gain values. Hence, the maximum value of E is used to split the node. Based on this approach, multiple decision trees are trained using the training data. The predictions from each decision tree are used to make the final prediction for the time series using the max voting strategy (Deng et al., 2013).

4.7.6 Rotation Forest

Rotation forest is an ensemble learning method, as a first step it splits the features into K subsets and applies principal component analysis (PCA) to extract features. Consider a dataset X with N observations and n number of features. Let Y denote the class label vector with shape $N \times 1$. The parameter K is selected before the training phase, and each feature subset contains $M = n/K$ features. Finally, L denotes the number of base learners in the ensemble algorithm and $\mathbf{F}$ denotes the complete feature set. The features extracted from PCA are placed in the rotation matrix.

The dataset X is multiplied with R_i^a and each base learner takes the input (XR_i^a, Y) in the training phase. During the testing phase, each base learner (D_i) gives probability scores for an observation belonging to multiple classes $\{\omega_1, \ldots, \omega_c\}$. The final prediction is calculated using the formula below:

$$\mu_j(\mathbf{x}) = \frac{1}{L} \sum_{i=1}^{L} d_{i,j}(\mathbf{x}R_i^a), j = 1, \ldots, c, \tag{25}$$

Where:

- a - coefficients of principal components $a_{i,j}^{(1)}, \ldots, a_{i,j}^{(M_j)}$, each of size $M \times 1$.
- R_i^a - rearranged rotation matrix of size $(N \times n)$.
- $\mu_j(\mathbf{x})$ - denotes the class label predicted for $\mathbf{x}$.
- $d_{i,j}(\mathbf{x}R_i^a)$ - denotes the probability predicted by the base learner D_i given $\mathbf{x}$, to test the hypothesis that $\mathbf{x}$ comes from class ω_j.

Algorithm 2 Algorithm for creating Rotation matrix in Rotation Forest

for $i = 1$ to the number of base learners (L) **do**

Create K feature subsets $\mathbf{F}_{i,j}$ (j = 1 ... K)

for $j = 1$ to the number of feature subsets (K) **do**

Select a random subset ($X_{i,j}$) from the data set X and features $F_{i,j}$

Remove a random subset of classes from $X_{i,j}$

Select 75 % of observations from the subset $X_{i,j}$ using bootstrapping and denote it by variable $X'_{i,j}$

Apply Principal component analysis store feature coefficients in $C_{i,j}$

end for

Create a Rotation matrix (R_i) from $C_{i,j}$ of all the K feature subsets

Create R_i^a by rearranging the columns of the rotation matrix (R_i) such that the features match the order of the original feature set

end for

(Rodríguez et al., 2006).

4.7.7 Convolutional Neural Network (CNN)

A convolutional neural network (CNN) consists of neurons, weights, and biases similar to dense layers in a neural network, but they are not fully connected. Each CNN layer contains kernels with a length smaller than the input. These kernels capture temporal features in the time series by sliding over the input, calculating the dot product, and then applying activation functions like ReLU, sigmoid, or hyperbolic tangent. After activation, the output undergoes a pooling operation like max pooling or average pooling. The equation for calculating the dot product is as follows:

$$s(t,h) = (w_h \times x)(t) = \sum_{n=-\infty}^{\infty} x(n) w_h(t-n), \tag{26}$$

Where:

- w_h – denotes the weights of the kernel h.
- x – represents a time series of length N.
- T – denotes a point in time series and ranges from 1 to N.

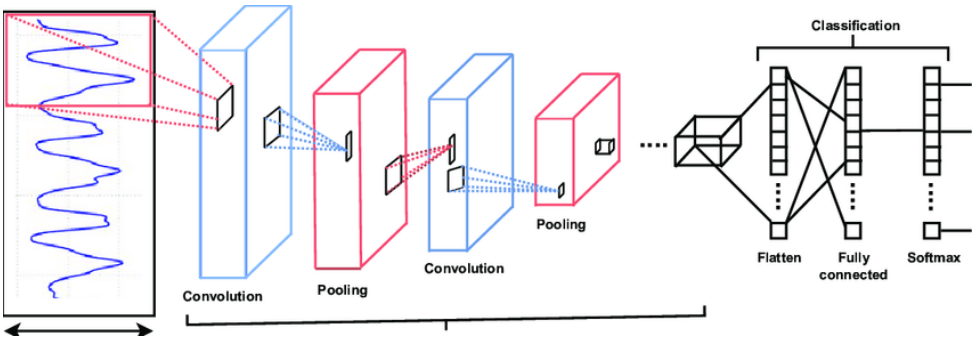


Figure 9: Convolutional Neural Network Architecture (Doniec et al., 2023).

Figure 9 shows the convolution operation performed on a time series data. It has two convolution layers each combined with a pooling layer. The output of the convolutional layers is flattened and passed through fully connected dense layers. Finally, the output layer uses the softmax function to classify the time series into a particular class (Arratia and Sepúlveda, 2020, p. 60-69).

4.7.8 KNN classifier

KNN is a non-parametric, simple algorithm used to predict the class of unknown data points using the nearest neighbor approach. To predict the class label for an observation in the test set, the algorithm computes the distance between the observation and all the observations in the training set. It then filters out the top k observations with the minimum distance to the test observation. The class label is assigned based on the class labels of these k nearest neighbors. For the KNN algorithm, it is assumed that the training data is labeled. The choice of the k value and the distance function plays an important role in the performance of the KNN classifier. The distance function must be selected so that the probability of k-nearest neighbors belonging to the same class is high. Generally, Minkowski, Manhattan, and Euclidean distances are used as distance functions.

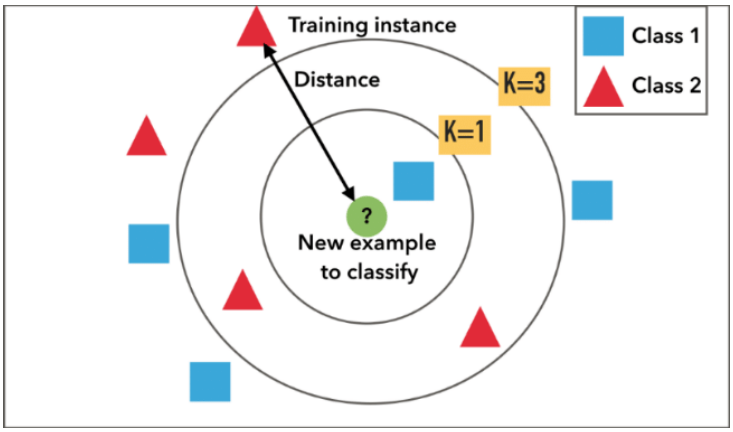


Figure 10: KNN classifier (Tarawneh, 2021).

The importance of the k-value is illustrated using the above figure. In figure 10, square and rectangle points belong to class 1 and class 2, respectively, of a training set. After computing

distances between all the points, if k = 1, then class 1 is assigned to the new observation, and if k = 3, class 2 is assigned. If the k value is too small, it is considered weak classification because there are only a few neighbors, and more instances of the training set are not considered for prediction. If the k-value is too large and most of the training instances belong to the same class, the prediction might be biased. Hence, the optimal value of k is chosen based on the results from the validation set (Mucherino et al., 2009, p. 83-84).

Algorithm 3 KNN algorithm

for $i = 1$ to the number of samples in test set (test) **do**
  for $j = 1$ to the number of samples in training set (training) **do**
    Calculate the distance between training(i) and test(j)
  end for
  Select the k samples from training(i) with smallest distance.
  Assign label to test(j) based on the majority labels of k nearest neighbors from training(i).
end for

4.8 Unsupervised models

4.8.1 Spectral Clustering

In general, simple clustering algorithms like K-means perform poorly when data points are not linearly separable. Spectral clustering is based on graph theory and uses linear algebra methods to separate data into clusters more efficiently. The steps in spectral clustering include constructing a similarity matrix, computing the graph Laplacian, and finding the eigenvectors. These eigenvectors represent the decomposed representation of the original data and are used as input for the final clustering algorithm.

Similarity matrix

Let $x_1, \ldots, x_n$ be a set of data points and the similarity between two points x_i and x_j be s_{ij}. The idea behind spectral clustering is to partition the data into multiple groups using the similarity information between the data. Similar data points are expected to be in the same group whereas dissimilar data points will be in different groups and hence the similarity s_{ij} is always greater than or equal to 0.

In spectral clustering graph $G = (V, E)$ is used to represent the similarity between the data points. V denotes a set of vertices. In this case, vertices v_i and v_j are data points x_i and x_j. The edges represent the connection between vertices and are weights of the edges w_{ij} represent the similarity (s_{ij}) between the data points x_i and x_j. The edges of the nodes within the group will have higher weights because of the higher similarity between them and the edges between different groups will have lower weights. As the graph is undirected, $w_{ij} = w_{ji}$.

In the case of a fully connected graph, the weighted adjacency matrix (W) contains the similarity information and can be calculated using the formula below:

$$W_{ij} = \begin{cases} s(x_i, x_j) & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}, \tag{27}$$

The similarity $s(x_i, x_j)$ can be computed using the Gaussian or radial basis function (RBF) and it can be represented using the formula:

$$s(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{28}$$

Where $\|x_i - x_j\|^2$ represents the Euclidean distance between the points x_i and x_j. σ represents the ϵ neighborhood graph. It is used to connect the data points with pairwise distances less than ϵ.

The degree (d_i) of the vertex $v_i \in V$ can be calculated using the formula $d_i = \sum_{j=1}^{n} w_{ij}$. The degree matrix D is used to represent the degree of all vertices and it is a diagonal matrix.

Graph laplacian and eigen vectors

The graph Laplacian matrix can be defined as $L = D - W$. To be a graph Laplacian matrix, the matrix L should satisfy the conditions below:

- For an arbitrary eigenvector, $f \in \mathbb{R}^n$, it should satisfy the condition $f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$.
- L is a positive semi-definite matrix and symmetric.
- The smallest possible eigenvalue of L is 0 and its corresponding eigenvector should have all the values assigned to 1.
- The matrix L contains real values and non-negative eigenvalues satisfying the condition $(0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n)$.

For a small eigenvalue λ the Laplacian graph matrix is associated with an eigenvector f. For a pair of data points x_i and x_j, if the similarity between them is high then $(f_i - f_j)$ is very small. Now, for the first $k(k \leq n)$ eigenvectors of the k smallest eigenvalues are selected as an input for the simple clustering algorithms like k-means clustering. Also, k value can be used to specify the number of clusters in k-means algorithm (Luxburg, 2007, p. 1573-1375).

Algorithm 4 Spectral clustering algorithm

Require: Input: set of data points $\{x_1, \ldots, x_n\}$, similarity function $s(x_i, x_j)$, number of clusters (k)

1: Create a similarity graph G from $\{x_1, \ldots, x_n\}$ and $s(x_i, x_j)$
2: Create weighted adjacency matrix (W) and degree matrix (D) from G
3: Calculate graph Laplacian matrix $L = D - W$
4: Compute first k eigenvectors $f_1, \ldots, f_k$ from L and let $U \in \mathbb{R}^{n \times k}$ represent the matrix of k eigenvectors
5: For $i = 1, \ldots, n$ let $\mathbf{z}_i \in \mathbb{R}^n$ represent the ith row in U
6: Fit k-means algorithm to $\mathbf{z}_i$ and assign the clusters from $C_1, \ldots, C_k$
7: **return** Cluster assignments

4.8.2 K-means clustering

K-means is a simple clustering algorithm that divides the entire dataset into a set of K non-overlapping clusters. The value of K determines the number of clusters and it should be specified before the start of the algorithm. In the first step, depending on the number of clusters, K random cluster centers are initialized and each observation is assigned to its closest cluster center. Let $C_1, \ldots, C_K$ denote cluster variables of K clusters, and n denote the total number of observations in the dataset then the following 2 properties are valid:

- $C_1 \cup C_2 \cup \ldots \cup C_K = n$. This says that the union of all the observations from K clusters is equal to the total observations in the dataset and it also denotes that each observation belongs to at least 1 cluster.
- $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$. This says that there are no common observations between any 2 clusters.

The amount of deviation of a data point to all other data points within the cluster can be defined as the within-cluster variation of a cluster. For cluster C_K, it can be denoted using $W(C_k)$. For a good clustering result obtained from the K-means algorithm, the within-cluster variation will be as small as possible. Common distance functions like squared Euclidean distance, and Manhattan distance can be used to calculate the within-cluster variance of the data points. In the case of time series data, dynamic time warping is used as a distance metric. The formula to calculate within-cluster variation using squared Euclidean distance can be represented as follows:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2, \tag{29}$$

Where:

- $|C_k|$ - denotes the total number of observations in clusters C_k.
- i, i' – denote the data points of the cluster C_k.

- p - denotes the number of features or parameters in the dataset.

At each iteration, the algorithm calculates $W(C_k)$ and updates the cluster centers by computing the mean of all the observations within the cluster. The process is repeated until it reaches the specified number of iterations or until convergence. Hence, the goal is to reduce the within-cluster variation of all the clusters. This can be considered as an optimization problem and can be denoted using the formula below:

$$\min_{C_1,\ldots,C_K} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}, \tag{30}$$

The convergence of the algorithm depends on the initialization of cluster centers, and the algorithm may converge at the local optimum. To handle this problem, the K-means algorithm is run multiple times with different initializations. In the end, the model with the smallest within-cluster variance is selected as the best model (James et al., 2021, p. 521-525).

Algorithm 5 K-means clustering algorithm

Input: number of clusters (K), dataset D with n observations
Initialize K random points as cluster centers
repeat
for each observation i in the dataset D **do**
    Assign observation i to its nearest cluster center
end for
update cluster centers by calculating the mean of all observations in the cluster
Until maximum iteration number or convergence
return Cluster results

(Jin and Han, 2017, p. 695-697)

4.8.3 K-shape clustering

The K-shape clustering algorithm is used to partition the data into K clusters using a shape-based distance metric. It is similar to the K-means algorithm as it uses an iterative approach to minimize the sum of squared distances to create well-separated clusters. The uniqueness of the K-Shape algorithm lies in formulating the distance measure and centroid computation, which captures and preserves the shape of the time series. Also, the K-Shape algorithm scales linearly with the increase in the number of time series sequences. K-shape is specifically designed to efficiently cluster time series data, and it is invariant to scaling and translation in the time series.

Distance Measure

To capture shape-based information, distance measures should be robust to changes in the amplitude and phase of the time series sequences. DTW (Dynamic Time Warping) is one

of the best distance metrics for handling these changes, but it is computationally expensive. Therefore, the k-shape algorithm uses a normalized version of the cross-correlation measure to capture shape-based similarity. Consider two sequences $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{y} = (y_1, \ldots, y_m)$. To determine the similarity between $\vec{x}$ and $\vec{y}$, the cross-correlation measure keeps the sequence $\vec{y}$ static and slides sequence $\vec{x}$ over $\vec{y}$ to compute the inner product of $\vec{y}$ with each shift s in $\vec{x}$. This helps capture the similarity even if one sequence is shifted relative to the other. The shift (s) in sequence $\vec{x}$ can be denoted using the formula below:

$$\vec{x}_{(s)} = \begin{cases} (\overbrace{0, \ldots, 0}^{|s|}, x_1, x_2, \ldots, x_{m-s}), & s \geq 0 \\ (x_{1-s}, \ldots, x_{m-1}, x_m, \underbrace{0, \ldots, 0}_{|s|}), & s < 0 \end{cases}, \tag{31}$$

In the above formula, m denotes the number of points in a sequence. If $s \geq 0$, then it denotes the shift is positive. The sequence $\vec{x}$ is shifted to the right by placing s 0's in the front of the sequence and removing s points at the end of the sequence. If $s < 0$, it denotes that the shift is negative. The sequence $\vec{x}$ is shifted left by removing the first s 0 points of the sequence and simultaneously s points are added at the end of the sequence. If all possible shifts are considered, then s belongs to $[-m, +m]$. The cross-correlation of sequences $\vec{x}$ and $\vec{y}$ is defined using $CC_w(\vec{x}, \vec{y}) = R_{w-m}(\vec{x}, \vec{y})$, where $w \in [1, 2, \ldots, 2m-1]$. Here, the index w represents different shifts of the sequence $\vec{x}$. This can be calculated using the formula below

$$R_k(\vec{x}, \vec{y}) = \begin{cases} \sum_{l=1}^{m-k} x_{l+k} \cdot y_l, & k \geq 0 \\ R_{-k}(\vec{y}, \vec{x}), & k < 0 \end{cases}, \tag{32}$$

In the above formula, $k = w - m$, k can have both positive and negative values which denote the shift of the sequence to the right and left. Based on the k value, the above formula is used to compute the inner product of y with shifted sequence x. The goal is to find the index w at which $CC_w(\vec{x}, \vec{y})$ is maximum. Based on the value of w, the optimal shift value s of sequence $\vec{x}$ with respect to $\vec{y}$ can be calculated as $s = w - m$.

Moreover, based on the field of application, a specific normalization method has to be applied to the values $CC_w(\vec{x}, \vec{y})$. In this case, coefficient normalization (NCC_c)is used. The coefficient normalization divides the cross-correlation measure with the geometric mean of autocorrelations of individual sequences.

Shape-based distance (SBD): A cross-correlation measure with coefficient normalization is used to create the shape-based distance metric. The values of $NCC_c(\vec{x}, \vec{y})$ range from -1 to +1. Finally, the shape-based distance metric can be represented using the formula below:

$$\mathrm{SBD}(\vec{x}, \vec{y}) = 1 - \max_w \left(\frac{\mathrm{CC}_w(\vec{x}, \vec{y})}{\sqrt{R_0(\vec{x}, \vec{x}) \cdot R_0(\vec{y}, \vec{y})}} \right), \tag{33}$$

In the above equation, $R_0(\vec{x}, \vec{x})$ and $R_0(\vec{y}, \vec{y})$ denotes the geometric mean of autocorrelation of $\vec{x}$ and $\vec{y}$ respectively. The values of SBD range from 0 to 2, where 0 indicates a perfect similarity between the sequences.

Shape extraction

In general, an average time series is used to make observations on a set of time series sequences. In clustering, this average can be denoted as a centroid. However, for time series, the computed average doesn't efficiently capture the underlying class characteristics. To address this, the centroid computation in k-shape clustering is treated as an optimization task where the goal is to minimize the sum of squared distances to all other sequences. Since cross-correlation measures the similarity of time series rather than dissimilarity, the objective function is changed to maximize the sum of squared similarities to all other sequences. As the clustering algorithm is iterative, the centroids from the previous iteration are used to reassign the sequences to the nearest centroid.

In the first step, the algorithm randomly assigns a cluster number from 1 to K to all the time series. In the second step, the cluster centers are computed using the shape extraction method. After computing the centroids, the cluster memberships are refined using the shape-based distance measure. Next, the algorithm iteratively performs two steps, namely, the assignment step and the refinement step. In the assignment phase, each time series is compared with all the computed cluster centroids, and cluster membership is reassigned to the closest centroid. In the refinement phase, as the cluster membership has changed for the data, the cluster centroids are computed and updated. These two steps are repeated until there are no changes in the cluster membership (the algorithm has reached convergence) or until the maximum number of iterations specified. The output of the algorithm contains final cluster assignments for each time series and the centroid values of each cluster (Paparrizos and Gravano, 2016).

4.8.4 Affinity propagation

The affinity propagation algorithm belongs to a family of message passing algorithms. Here, each point in the dataset is considered a node and messages are passed through the edges in the network until a good set of clusters are created. The cluster centers are called exemplars and are chosen in such a way that they represent the characteristics of other data points in the cluster. The algorithm works very differently compared to normal clustering algorithms. In the initial phase, each point in the dataset is considered an exemplar and the algorithm automatically determines the number of clusters for the provided dataset.

As the algorithm uses the graph structure, it expects the input as a similarity matrix. Let i and k be two data points. The similarity $s(i, k)$ denotes how well the data point at index k is suited to be an exemplar for the data point i. In general, negative squared Euclidean distance is used as a metric to compute the similarity between the points and it is calculated using the equation $s(i, k) = -\|x_i - x_k\|^2$.

For each data point, $s(k,k)$ computes the self-similarity. This is important because the data points with larger self-similarity values are most likely to be chosen as exemplars. The self-similarity values are called preferences and are also used to determine the number of clusters. If all the points in the dataset are equally suitable to be an exemplar, the preferences are set to a common value like the median. Two kinds of messages are passed between the data points, namely, responsibility and availability.

Responsibility

The responsibility message is passed from the data point i to the potential exemplar candidate k. This message provides evidence to show how well suited the point k is to serve as an exemplar for point i when compared to other candidate exemplars for i. The responsibility can be computed using the formula below:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} \left\{ a(i,k') + s(i,k') \right\}, \tag{34}$$

Where:

- $r(i,k)$ – denotes the responsibility computation between the point i and exemplar k.
- $s(i,k)$ – denotes the similarity between point i and exemplar k.
- $s(i,k')$ – denotes the similarity between the point i and other exemplars k'.
- $a(i,k')$ – denotes the availability computation between the point i and other exemplars k'.

In the first iteration, the responsibility and availability matrices are initialized to 0. Then, the responsibility is calculated using the above formula. As the availability matrix is 0 for the first iteration, the responsibility is computed by subtracting the maximum value of the similarity between the point i and other candidate exemplars k' from $s(i,k)$. The self-responsibility $s(k,k)$ is computed by subtracting $s(i,k')$ from $s(k,k)$. The self-responsibility shows evidence of how ill-suited it is to assign an exemplar k to another exemplar.

Availability

The availability message is passed from the potential exemplar k to the data point i. This message provides evidence to show how appropriate it is to choose the point k as an exemplar for point i by considering the support from all other points that have chosen k as its exemplar. The availability can be computed using the formula below:

$$a(i,k) \leftarrow \min \left\{ 0, r(k,k) + \sum_{i' \notin \{i,k\}} \max \left\{ 0, r(i',k) \right\} \right\}, \tag{35}$$

Where:

- $a(i,k)$ – denotes availability computation between the point i and k.
- $r(k,k)$ – denotes self-responsibility of the candidate exemplar k.

- $R(i', k)$ – denotes responsibility of the other points i' to the exemplar k.

At each iteration, the responsibility and availability matrices are updated based on the current similarity and availability values. To identify the exemplars, the availability and responsibility values are combined. For a point i, the maximum value of $a(i, k) + r(i, k)$ either denotes an exemplar k if $k = i$ or finds an exemplar point of the data point i. The algorithm stops when the message passing falls below a certain threshold, when there are no changes in the exemplars of the data points for a certain period of time, or when the specified number of iterations is reached (Frey and Dueck, 2007).

4.8.5 Self Organizing Maps (SOM)

Self-organizing maps (SOM) are a special type of artificial neural network primarily used for tasks like dimensionality reduction and clustering. The goal of SOM is to transform high-dimensional data into lower dimensions, typically 2D, without losing the relationships between the input data. The main advantages of SOMs are that they do not assume variables in the dataset follow a certain distribution or that variables are independent. They perform well on non-linear datasets and are easy to implement. It is an unsupervised approach and they are also called Kohonen maps.

The architecture of a SOM network consists of just an input layer and an output layer. The input layer is directly connected to the output layer without any hidden layers. Consider a data set X with m features. Each row in the dataset can be represented as an m dimensional input vector $x(t) = (x_1(t), x_2(t), \ldots, x_m(t))'$ at iteration t, and hence, the number of neurons in the input layer is equal to the number of features in the dataset (m). The number of neurons in the output layer is equal to the number of clusters specified by the user. This is a fully connected network, and weights are initialized randomly. Let $w(t) = (w_{i1}(t), w_{i2}(t), \ldots, w_{im}(t))$ where $i = 1, 2, \ldots, n$ denotes the number of nodes in the output layer. The SOM model is trained based on three characteristic processes: competition, cooperation, and adoption.

Competition: In this process, all the output neurons compete with each other to learn and represent the input data in the best possible way. The neuron with the best representation is declared the winner. Representations of each output neuron are compared with input neuron using functions like Euclidean distance, and the neuron with the weight vector closest to the input vector is determined as the best matching unit.

The Euclidean distance to calculate the similarity between the input vector and each output node is represented using the formula below:

$$d_i(t) = \|x(t) - w_i(t)\| = \sqrt{\sum_{j=1}^{m}(x_{tj} - w_{ij}(t))^2}, \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m, \tag{36}$$

Where:

- t denotes the iteration number.
- $x(t)$ represents the input vector at iteration at t.
- $w_i(t)$ represents the weight vector of the ith output node at iteration t.
- j denotes a specific feature from the dataset.
- $x_j(t)$ and $w_{ij}(t)$ are the j-th components of the input vector and the weight vector of the i-th output neuron at iteration t, respectively.

Finally, the formula for winning node c can be represented as follows:

$$c(t) = \arg\min_i \{\|x(t) - w_i(t)\|\}, \tag{37}$$

Cooperation: SOM output neurons are topologically structured so that neurons capturing similar properties from input data are located near each other. This is achieved by using neighborhood information. Here, the winner node determines the neighborhood of the similar or cooperating nodes.

Adaptation: The winner node and its neighboring nodes weight vectors are adjusted to become more similar to the input sample. As a result, the node that captures a particular input information precisely will also capture data similar to this input. The weights are updated based on the hyperparameters learning rate (α) and neighborhood size.

The learning rate is inversely proportional to the number of iterations (t), i.e., as the number of iterations increases, the learning is decreased linearly. Let $w_i(t)$ be the weight vector of the winner node in iteration t. Then, the weight vector is updated at iteration $t + 1$ using the formula below:

$$w_i(t+1) = w_i(t) + \alpha(t)[x(t) - w_i(t)], \tag{38}$$

As the algorithm uses neighborhood information to preserve the topological information of the input data, the neighborhood in iteration t is calculated using the formula below:

$$h_{ci}(t) = \exp\left(-\frac{d_{ci}^2}{2\sigma^2(t)}\right), \tag{39}$$

Where:

- $h_{ci}(t)$ - represents the neighborhood function.
- d_{ci}^2 - represents the distance between winning neuron c and excited neuron i.
- $\sigma(t)$ - denotes the effective radius of neighborhood in iteration t.

(Asan and Ercan, 2012)

4.9 Cluster ensemble methods

The process of applying multiple clustering algorithms to the same dataset and combining the results to create final clusters is called cluster ensemble. Just as ensemble models have proven successful in improving the results of supervised classifiers, the same idea is used to enhance overall clustering results. Cluster ensemble methods consist of two steps: generation and consensus function. In the generation step, several clustering algorithms are applied to the same dataset, and the labels from each method are stored. In the consensus function step, the partitions obtained from different clustering algorithms are combined to create new cluster labels or partitions. The new clusters obtained may or may not be better, as there is no ground truth for comparison. It is based on the assumption that combining labels from different algorithms could reduce the errors made by using only a single algorithm, making the results more reliable. In this thesis, co-association matrix methods and relabel and maximum voting methods are used for cluster ensemble in the consensus step (Vega-Pons and Ruiz-Shulcloper, 2011).

4.9.1 Co-association matrix method

In the consensus step of the co-association matrix method, first, a binary co-occurrence matrix is created for each clustering algorithm in the generation step. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a dataset with n observations, and $\mathbb{C}^{\text{in}} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$ be a set of m clustering algorithms applied on X. For each clustering algorithm, the binary co-occurrence matrix is $n \times n$ matrix, which indicates whether two observations x_i and x_j belong to the same cluster. The values of the binary co-occurrence matrix for algorithm $\mathcal{C}_k$ can be represented using the method below:

$$\mathcal{C}_k(i,j) = \begin{cases} 1, & \text{if } x_i \text{ and } x_j \text{ are clustered together} \\ 0, & \text{otherwise} \end{cases}, \tag{40}$$

The co-association matrix of size $n \times n$ is created by taking the average of all the binary co-occurrence matrices. The formula for this is as follows:

$$M(i,j) = \frac{1}{m} \sum_{k=1}^{m} C_k(i,j), 1 \leq i \leq n, 1 \leq j \leq n, \tag{41}$$

In the above formula, $M(i,j)$ denotes the co-association matrix of m clustering algorithms. In the co-association matrix, the observations that belong to the same cluster will have values close to 1, while the observations that belong to different clusters will have values close to 0. This method is simple and avoids relabelling problems when combining labels from different algorithms. This co-association matrix serves as a similarity matrix for final clustering algorithms like hierarchical clustering or spectral clustering (Hu et al., 2018, p. 599-613).

4.9.2 Relabel and maximum voting method

The consensus step of the relabel and voting method consists of two steps: relabeling the cluster assignments and the voting step. Since the cluster labels generated by different clustering algorithms have no relation between them, it is crucial to relabel them for use in the voting method. This relabeling task can be seen as a label correspondence problem and is particularly challenging in unsupervised domains. In this thesis, the Hungarian algorithm is used to address this label correspondence problem through a maximum likelihood approach. The goal of the Hungarian algorithm is to minimize disagreements between pairwise mappings of cluster labels or maximize total agreements between them. For efficient implementation of this approach, it is essential that different clustering algorithms yield the same number of clusters. The results of the Hungarian algorithm are meaningful only when there exists a relationship between the labels assigned by different clustering algorithms (Vega-Pons and Ruiz-Shulcloper, 2011).

Consider two sets A and B each containing n elements. A cost matrix C is created such the rows contain the elements from set A and the columns contain elements from set B. An element in the cost matrix C_{ij} denotes the cost of assigning the ith element in A to the jth element in B. This matrix is used to find the assignment that minimizes the total cost.

Steps in the Hungarian Algorithm:

- In the cost matrix C, find the minimum value in each row and subtract it from all the elements in the row. Similarly, find the minimum value in each column and subtract it from all the elements in the column. As a result, some values in the cost matrix are assigned to value 0.
- Determine the minimum number of lines for both rows and columns such that all the zeros are covered.
- If the number of lines is less than n, find the minimum of values that are not covered by the lines. Subtract the minimum value from the uncovered elements and add the minimum value to the intersection elements. The elements that contain a row line and a column line are considered as an intersection element. Repeat the above steps until the number of lines is equal to n.
- If the number of lines is equal to n, then the algorithm has reached the optimum solution for the assignment between the sets A and B.

The Hungarian algorithm is suitable only for medium-scale assignment problems because of its time complexity (Gil-Aluja, 1998, p. 148-158).

Once the optimal mappings are obtained from the Hungarian algorithm, the cluster labels from individual clustering algorithms are relabeled accordingly. Subsequently, the voting method is applied to determine the final cluster assignment for each data point. Each data point is assigned to the cluster that receives the most votes. This voting mechanism resolves conflicts in

assignments from different algorithms and ensures that each data point is assigned to a single cluster (Vega-Pons and Ruiz-Shulcloper, 2011).

4.10 Evaluation Metrics

4.10.1 Confusion matrix

A matrix used to evaluate the predictions obtained from the classification model is termed a confusion matrix. It consists of N rows and N columns, with the rows representing the actual classes and the columns representing the predicted classes. Here, N denotes the number of classes in the dataset. For N = 2 classes, the confusion matrix can be represented using the table below:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Table 1: Confusion Matrix

From table 1, the definitions of the elements TP, TN, FP, and FN are as follows:

True Positive (TP): count of positive values predicted by the model, and the ground truth is also positive.

True Negative (TN): count of negative values predicted by the model, and their ground truth is also negative.

False Positive (FP): count of positive values predicted by the model, and their ground truth is negative.

False Negative (FN): count of negative values predicted by the model, and their ground truth is positive (Dalianis, 2018, p. 46-53).

4.10.2 Classification report

Performance metrics like recall, precision, F1 score, and accuracy can be calculated using the elements of confusion matrix. These metrics provide different information about the classification model, and their collective can be termed as a classification report.

Accuracy: The count of correct predictions divided by the total number of predictions made by the model is termed accuracy. It is calculated using the formula:

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

Precision: The count of true positive predictions divided by the total number of positive predictions made by the model is termed precision. It can be calculated using the formula:

$$Precision = \frac{TP}{TP + FP},$$

Recall: The count of true positive predictions divided by the total number of actual positive values is termed as the recall of the model. It is also called sensitivity or true positive rate. It can be calculated using the formula:

$$Recall = \frac{TP}{TP + FN},$$

Specificity: The count of true negative predictions divided by the total number of actual negative values can be termed as specificity or True Negative Rate (TNR). It can be calculated using the formula:

$$Specificity = \frac{TN}{TN + FP},$$

F1-score: The harmonic mean between the precision and recall values can be termed F1-score. It can be calculated using the formula:

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall},$$

(Dalianis, 2018, p. 46-53).

4.10.3 Hyperparameter tuning

In general, the variables learned during model training are called parameters. For example, weights in a neural network. Hyperparameters are different from normal parameters because these values are assigned by the user before training starts. Every machine learning model comes with some hyperparameters, and it is important to experiment and identify the best hyperparameters as they can improve the model's performance.

Grid search is one approach to hyperparameter tuning, where multiple values are provided for each hyperparameter in grid form, and the model is trained on each possible combination of hyperparameters. The advantage of this approach is that it extensively searches all combinations to find the best hyperparameters that improve the model's performance. As a result, we get the best model, but the disadvantage is that as the number of values for hyperparameters increases, the time taken for computation increases exponentially (Ippolito, 2022, p. 231-251).

4.10.4 Cross-validation

In this method, the dataset is split into k folds. During each iteration, the model is trained using k-1 folds while one fold is kept aside for validation. This procedure is iterated k times to

evaluate the model's performance across various subsets of the dataset. To calculate the final result, the model results from each iteration are averaged. As different subsets of data are used to train the model in each iteration, the performance metrics help to identify the variability in model performance and also avoid overfitting. After capturing the best hyperparameters from the cross-validation, the final model is trained using the hyperparameters on the entire dataset (Ippolito, 2022, p. 231-251).

4.10.5 AUC-ROC curve

The Receiver Operating Characteristic (ROC) curve is used to visualize the performance of the classifier for different thresholds. The x-axis of the ROC curve represents the False positive rate (1-specificity) and the y-axis represents the True positive rate (sensitivity). The empirical ROC curve is generated by connecting the data points obtained for each threshold. A better ROC curve will have the values closer to the top-left corner of the graph. This shows that the model has high TPR and low FPR. The ROC curve provides the trade-off between the sensitivity and specificity (Zhou et al., 2011, p. 24).

The area under the ROC curve is used to quantify the performance of the classification model. AUC can take the values from 0 to 1. AUC value 1 indicates the classifier is perfect with 0 false positive rate, whereas the AUC value 0 indicates that all positive observations in the dataset are classified as negative and all negative observations are classified as positive. In real-world datasets, both AUC = 0 and 1 are highly unlikely and hence, AUC values range from 0.5 to 1. If the AUC value is 0.5, then, the ROC curve is a diagonal line indicating the model has no predictive power and is similar to random guessing. Therefore, a higher AUC value indicates the higher predictive power of the classifier (Zhou et al., 2011, p. 28).

4.10.6 Adjusted Rand Index (ARI)

The Rand Index is an external cluster evaluation metric used when true labels of the dataset are available. It measures the similarity between cluster assignments and true class labels using pairwise comparisons. Consider a dataset X with n observations. Let C and D be two clustering results with r and s clusters, respectively. A contingency matrix of shape $r \times s$ is used to determine the relationship between the two cluster sets. The element in the ith row and the jth column of the contingency matrix represents the cardinality of $C_i \cap Dj$. The Rand Index is calculated from the summary of four statistics computed from the contingency table.

a: Number of pairs of elements that are the same sets in both C and D.

b: Number of pairs of elements that are in the same set in C but are in different sets in D.

c: Number of pairs of elements that are in different sets in C but are in the same set in D.

d: Number of pairs of elements that are in different sets in both C and D.

All the described summary statistics are computed using the formulas below:

$$
\begin{aligned}
a &= \frac{(\sum_{i=1}^{r}\sum_{j=1}^{s} n_{ij}^2) - n}{2}, \\
b &= \frac{\sum_{i=1}^{r} n_{i+}^2 - \sum_{i=1}^{r}\sum_{j=1}^{s} n_{ij}^2}{2}, \\
c &= \frac{\sum_{j=1}^{s} n_{+j}^2 - \sum_{i=1}^{r}\sum_{j=1}^{s} n_{ij}^2}{2}, \\
d &= \frac{(\sum_{i=1}^{r}\sum_{j=1}^{s} n_{ij}^2) + n^2 - \sum_{i=1}^{r} n_{i+}^2 - \sum_{j=1}^{s} n_{+j}^2}{2},
\end{aligned}
\tag{42}
$$

Here, $n_{i+} = \sum_{j=1}^{s} n_{ij}$ denote the row wise totals $(n_{1+}, \ldots, n_{r+})$ and $n_{+j} = \sum_{j=1}^{r} n_{ij}$ denote the column wise totals $(n_{+1}, \ldots, n_{+s})$. Finally, the rand index can be computed using the variables in summary statistics as follows:

$$
RI = \frac{a+d}{a+b+c+d}, \tag{43}
$$

Moreover, if $N = a+b+c+d$, then the Rand Index can be represented as $a + d/N$. The values of the Rand Index range from 0 to 1, where 0 indicates random labeling and 1 indicates a perfect agreement between the clusters. One drawback of using the Rand Index is that as the number of clusters increases, it provides a high score for random cluster assignments. Therefore, the Rand Index is not useful for evaluating clusters accurately. The Adjusted Rand Index (ARI) is used to correct this by normalizing the Rand Index score. The formula used to calculate the Adjusted Rand Index is represented as follows:

$$
ARI = \frac{RI - \mathbb{E}[RI]}{max(RI) - \mathbb{E}[RI]}, \tag{44}
$$

Where:

- RI – denotes the rand index.
- $\mathbb{E}[RI]$ – represents the expected value of the rand index in case of random cluster assignments.
- $max(RI)$ – denotes the maximum value of the rand index, i.e., 1.

(Chacón and Rastrojo, 2023, 125-133).

5 Project workflow

This section of the thesis report provides a brief overview of the entire workflow and the tasks performed on the screw-tightening dataset. In sub-section 5.1, the architecture of the project is described in detail using a flowchart. Sub-section 5.2 provides an overview of the dataset and describes all the screw errors in a table. Sub-section 5.3 explains the pre-processing steps performed to standardize the dataset and make it suitable for modeling. Furthermore, based

on analysis and statistical tests, some hypotheses are formulated to be verified using modeling techniques in the next section.

5.1 Screw tightening Anomaly detection Architecture

Figure 11 illustrates the proposed architecture for detecting anomalies in screw-tightening processes in the manufacturing industry using unsupervised approaches. This flowchart diagram includes various components. Both model training and prediction on new test data follow a similar procedure, as shown in the figure below. In this subsection, each component of the architecture is explained in detail in general terms.

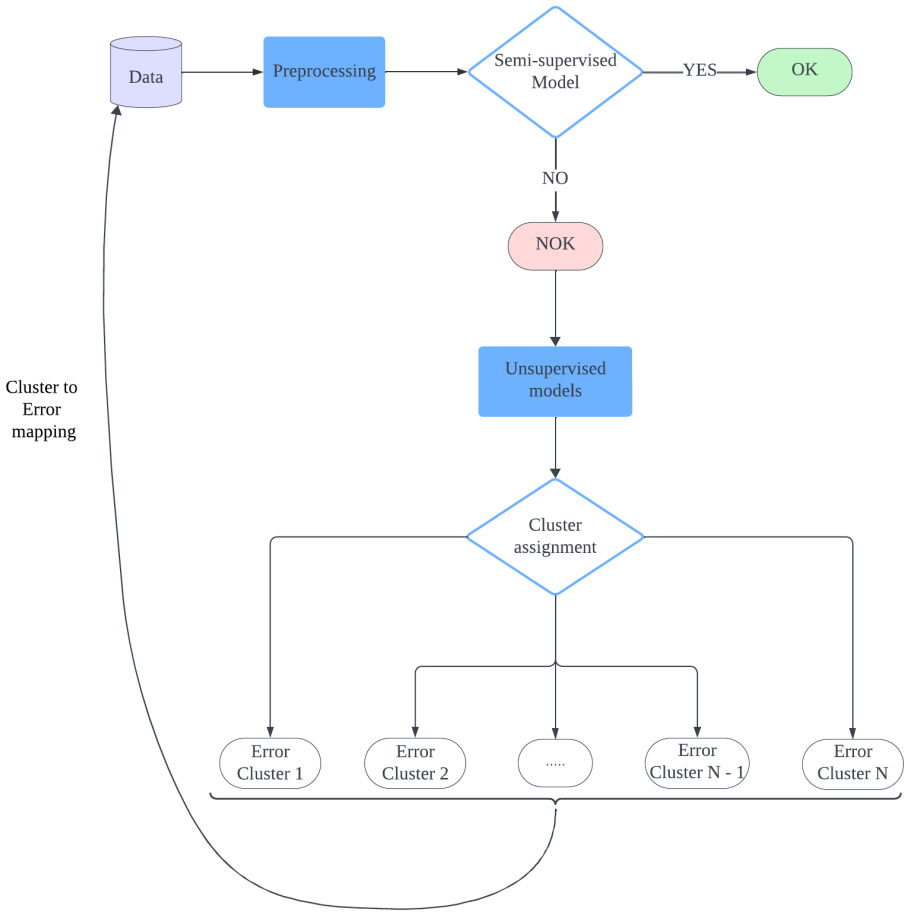


Figure 11: Anomaly detection Architecture.

5.1.1 Training pipeline

The training pipeline consists of components such as data, pre-processing, semi-supervised anomaly classification, cluster assignment for different error categories using clustering, and mapping the clusters to error categories in the dataset.

Data component: It is a database component and also acts as input component. It contains the raw data used for subsequent tasks in the pipeline. In the context of a screw-tightening

process, this component includes data from sensor measurements of angle values, torque values, and gradient values collected from each screw-tightening process at regular intervals of time.

pre-processing: This step involves preparing raw data for analysis and modeling. Generally, this step can include processes like cleaning and handling missing values in the data, normalizing or standardizing the data, and other techniques to ensure the data is suitable for modeling. In this thesis, this component is used to perform tasks specifically like removing duplicates and handling missing values in each screw-tightening process, cutting/padding the time series sequences to have a constant sequence length in all sequences, and applying min-max normalization to normalize torque sequences.

Semi-supervised model: This component in the flow chart represents a point where a decision needs to be made. It typically has two or more branches coming out of it, representing the different possible outcomes of the decision. The decision box usually contains a question or condition, and the branches are labeled with possible answers, such as 'Yes' or 'No'. Based on the answer, the flowchart will follow a different path, directing the process to the appropriate next step. This helps to visualize how different choices or conditions affect the flow of the process.

In this thesis, this component applies several semi-supervised models such as One-class SVM, Isolation Forest, and Autoencoders to predict whether the screw-tightening process is anomalous or not. A weighted ensemble learning method combines the predictions of these three models to make a final decision (Anomaly / Not an anomaly). Therefore, this step branches into two paths based on the decision made.

OK: This is an output/terminal component. Depending on the decision made in the previous step, the process halts if it follows this path. In this thesis, the process reaches this stage if the prediction indicates that a specific screw-tightening process is not an anomaly. This confirms there are no errors in the process and the process stops here.

NOK: This is also an output component. The process reaches this stage if the prediction indicates that this specific screw-tightening process is an anomaly. Further tasks are then performed to identify the type of anomaly.

Unsupervised models: This process component is executed if the decision from the previous step is NOK. Here, several unsupervised models are applied to the dataset to identify patterns or group similar points into clusters without any prior knowledge of the classes. In this thesis, if the screw-tightening process is identified as an anomaly, clustering methods such as Spectral Clustering, Affinity Propagation, K-means, K-shape, Self-Organizing Maps (SOM), and cluster ensemble methods are utilized to cluster similar error sequences in this step.

Cluster assignment: This component also involves decision-making. Based on the outcomes from multiple unsupervised models, the time series sequences are assigned to N clusters based on their similarities. Each cluster represents a group of similar anomalies. In this thesis, a com-

bination of predictions from multiple algorithms using cluster ensemble methods or predictions form the best-performing cluster algorithm can be used in this decision-making process.

Error class 1, . . . , **N:** Each of the blocks 1 to N in the flowchart represents an output component of the cluster assignment. These clusters represent the errors identified by unsupervised models, with each cluster corresponding to a specific type of anomaly in the screw-tightening process.

Finally, the line or arrow from cluster blocks 1 to N to the data component represents the mapping of each cluster obtained in the unsupervised process to a specific error category in the labeled dataset. This mapping helps identify the types of anomalies or specific error categories present in each cluster. It is useful for improving data pre-processing techniques, as well as the semi-supervised and unsupervised models used in the training pipeline. Overall, the results or accuracy scores obtained from this mapping can be used to improve the anomaly detection process as needed. This training pipeline is not only applicable to the field of screw-tightening but also to any anomaly detection process in the manufacturing industry that utilizes semi-supervised and unsupervised models.

5.1.2 Prediction pipeline

The prediction/testing pipeline is similar to the training pipeline described in the previous subsection. During the testing phase, a single JSON file containing recorded sensor data from the screw-tightening process serves as the test data. The file undergoes specified pre-processing steps, and a prediction from a weighted ensemble learning model determines whether the screw-tightening test run is anomalous. If the process is not anomalous, it concludes here. If it is anomalous, the process proceeds to predict the type of anomaly. The predict method from the best-performing cluster model is then used to predict the cluster for the new test data. After obtaining the cluster, the probabilities of the test data belonging to each error category are specified.

The web application created using Streamlit is used to display the results appropriately. First, a Plotly graph (time vs torque line plot) is shown to visualize the input test data. Subsequently, the results from the semi-supervised models are displayed (Anomaly / Not anomaly). In cases where the process is flagged as anomalous, the K-shape algorithm predicts the cluster for the anomaly. Since clusters may contain sequences from multiple error categories due to their similar patterns, the prediction displays the probability of the test sequence belonging to each error category within the specific cluster. These probabilities are crucial as they quantify the likelihood of belonging to different error categories in cases where it is not definitively possible to determine a single error category.

(a) Screw-tightening machine

(b) Screw-fixing component

Figure 12: Industry machine for automated screw-tightening

5.2 Screw-tightening machine

Figure 13: Base component.

The images in figure 12 show the machine and screw-fixing component used for screw-tightening in the industry. Figure 12(a) displays the complete machine, where a base component will be placed on the left side before the machine is turned on. When the machine starts running, the base component moves inside the machine. Simultaneously, a screw is automatically inserted into the fixing component shown in Figure 12(b) through a tube. When the base component reaches the correct position inside the machine, the fixing component places the screw in the base component and tightens it by applying torque.

Figure 13 shows the base component, which contains two holes in the center. These are the exact spots where screws are tightened by the screw-fixing component. Once the screws are tightened into the respective holes, the base component is sent out on the right side of the machine. The

entire process, including inserting the base component into the machine and retrieving it, takes about 30 seconds. The actual screw-tightening takes only 2-3 seconds.

5.3 Description of the dataset

The time series data for each screw-tightening process is recorded using sensors and provided in the form of JSON files. All necessary variable information is extracted from each JSON file, and a CSV file is created, which includes data from all screw runs. In each screw-tightening procedure, the sensors record the timestamp (in milliseconds), angle value, torque value, and gradient value every 12 milliseconds. This CSV file is used as the dataset for all the analysis and modeling techniques carried out in this thesis. A brief description of each feature in the CSV file is stated in table 2. A total of 1342 time series sequences (screw-tightening runs) are used for analysis and modeling in this thesis.

Variable name	**Description**
Angle value	It denotes the rotation angle of the screw being tightened. It is used to determine how much the screw has been turned from its original position.
Torque value	It is the rotational force applied to the screw during the tightening process. It is measured in Newton-meters (Nm) and shows how much force was applied to turn the screw.
Gradient value	It refers to the rate of change of torque with respect to the angle during the tightening process.
Timestamp	It denotes the timestamp at which the observations are recorded.
Error_category	It is a categorical variable and denotes the specific error category in the screw-tightening process.

Table 2: Variables in the dataset

From all the features extracted, only the torque values from the screw-tightening process are used for all the analysis and modeling procedures.

5.3.1 Different phases in the screw-tightening procedure

Each normal screw-tightening process takes about 0.6 to 0.9 seconds to complete. In the case of some faulty runs, the tightening process may run for a longer period, so a maximum time limit of 1.5 seconds is specified. After this time, the process is automatically stopped. Figure 14 is a line plot showing the torque vs time values recorded for a specific screw-tightening process in OK category. In each screw-tightening process, the screw undergoes four different phases: finding, thread forming, pre-tightening, and tightening 1.4. These phases mentioned can be noticed in the figure 14 and are also explained in detail below.

Finding: The phase in which the tightening machine looks for a screw in the storage place and attaches itself to the head of the screw is called the finding phase. This phase usually takes 0.1 to 0.12 seconds of the entire tightening process. This is the first phase of the process.

Thread forming: This is the second phase of the process. In this phase, the machine tries to place the screw into a base component and rotates the screw such that the threads of the screw are moved into the component. This phase usually takes place between 0.1 to 0.5 seconds of the tightening process. During this phase, the torque increases steadily.

pre-tightening: In the third phase, the machine tries to tighten the entire screw into the base component. This phase usually occurs in the time range of 0.5 – 0.7 seconds of the tightening process and the screw is perfectly fixed into the component.

Tightening 1.4: This is the final phase of the tightening process and in this phase, the machine all of a sudden tries to detach itself from the screw head. During this process, the torque values decrease gradually until they reach 0. This phase is carried out usually in the time range of 0.7 – 0.9 seconds of the entire tightening process.

Figure 14: Torque vs time line plot.

5.3.2 Different errors in screw-tightening

The dataset used in this thesis consists of 15 different error categories (NOK categories) and one category denoting the perfect screw-tightening process (baseline/OK category). Each error category contains 50 - 100 screw-tightening runs, all of which are manually labeled. Each error category is created by manipulating the screw or base component or screw tightening device in certain way and they are explained below:

- **Adhesive-thread** - A metal adhesive (foreign material) is applied in the center part of the screw. This material blocks the screw connection with the base component.

- **Deformed-thread** - A tool is used to deform 3 to 4 threads in the center of the screw. As a result, some material is removed from the screw and also some parts of the screw are misshapen.
- **M3-half-washer-in-upper-part** - A half M3 polyamide washer is attached in the upper part of the screw. As a result, the screw distance is shortened when fastened using machine.
- **M3-washer-in-upper-part** - A M3 polyamide washer is attached in the upper part of screw. As a result, the screw distance is shortened when fastened using machine.
- **M4-washer-in-upper-part** - A M4 polyamide washer is attached in the upper part of screw. As a result, the screw distance is shortened when fastened using machine.
- **Material-in-lower-part** - A plastic adhesive is applied to the lower part of the screw connector device. It gives more resistance in the core screw hole.
- **Material-in-screw-head** - A plastic adhesive material is applied on the top of the screw (specifically in the cross-shaped part of screw head). As a result, the connector slips when tries to make contact with screw head.
- **Offset-of-screw-hole** - A ployamide washer is used to create a horizontal offset of screwdriver to the joining part. The washer reduces the hole size and hence the screwdriver first hits the washer instead of lower part.
- **Offset-of-work-piece** - A foreign material between the component and tool carrier is used to create offset between the screw axis and the screw in tube. As a result, an inclined screw connection is created.
- **Shortening-the-screw** - A saw is used to remove 2 threads at the end of the screw. As a result, the screw is shortened.
- **Surface-lubricant** - An oil based lubricant is used to reduce the friction in the upper part.
- **Surface-sanded-40** - A sandpaper with 40 grit roughness is used in the upper part to increase surface friction.
- **surface-sanded-400** - A sandpaper with 400 grit roughness is used in the upper part to increase surface friction.
- **Surface-used** - An upper part that had already been bolted 25 times was used during assembly and the lower part was completely new, so the defect occurs due to surface effects.
- **Tearing-off-screw** - A saw is used to completely cut the screw exactly under the screw head. This will result a complete failure during assembly.

For the OK category, the data is recorded in four different experimental settings, resulting in four sub-categories. No error characteristics were introduced in these four sub-categories. Each sub-category contains 100 - 200 screw-tightening runs, all of which are manually labeled. The experimental setting used to create each sub-category of baseline are explained below:

- **Baseline** - Observations recorded during the generation of the defects without any manipulations.
- **Baseline-abrasion** - Observations recorded during the investigation of influence of abrasion caused by thread cutting. No error characteristics are introduced.
- **Baseline-friction** - Observations recorded during the investigation of influence of friction in the head support. No error characteristics are introduced.
- **Baseline-extra** - Observations without errors are generated for additional reference.

Figure 15 shows the bar plot for the number of time series recorded in the OK and NOK categories. It can be noted that in the OK category, only the *baseline-abrasion* sub-category has 200 time series sequences, while all other sub-categories have 100 sequences recorded. In the NOK categories, only the *deformed-thread* and *surface-used* categories have 100 time series sequences. Most of the other categories have 50 sequences, with two categories having fewer than 50 sequences. Specifically, *shortening-the-screw* category has 47 sequences recorded, and the *tearing-off-screw* category has only 45 sequences.

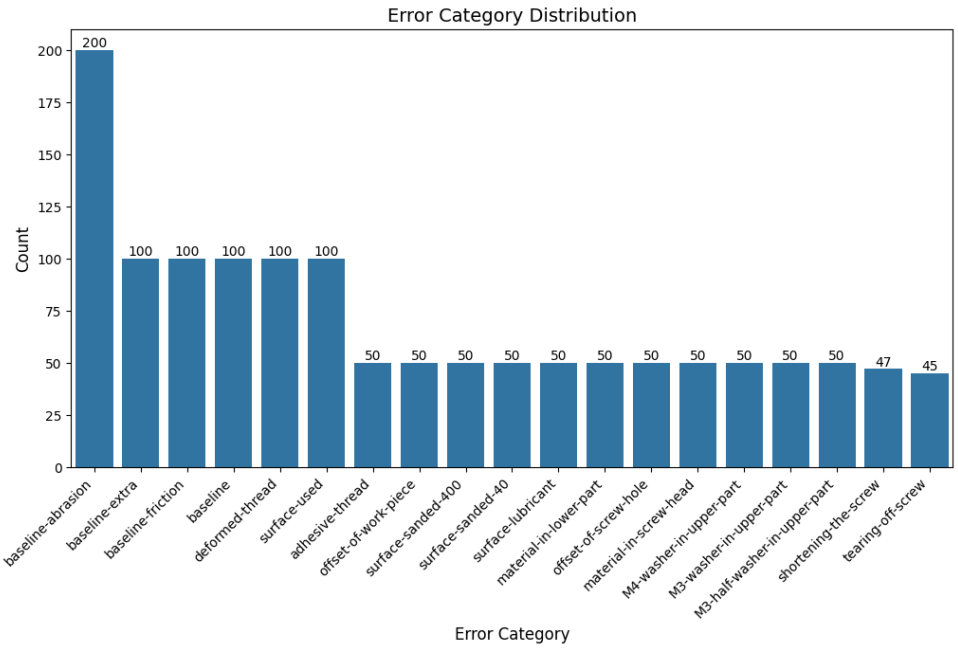


Figure 15: Error categories bar plot.

5.4 Data pre-processing

In this sub-section, data pre-processing methods such as handling duplicate and missing values, cutting/padding of time series sequences, and min-max normalization are applied to the raw time series sequences. The insights from these processes are explained in detail using appropriate visualizations.

5.4.1 Handling duplicate values

If multiple torque, angle, and gradient values are recorded at the same timestamp, these timestamps are said to have duplicate values. Figure 16 shows the histogram of duplicate values with respect to timestamps, plotted from 1342 screw-tightening processes.

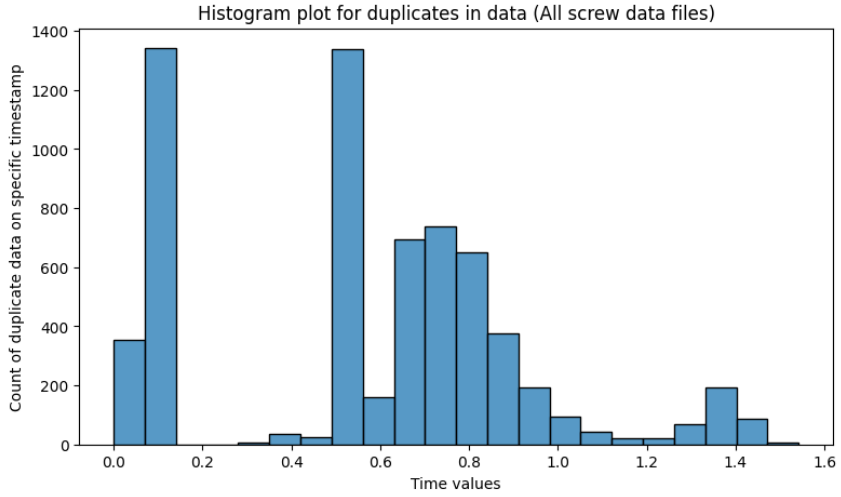


Figure 16: Duplicate values histogram plot.

It can be observed that most duplicate values are recorded around the timestamps 0.1 and 0.5, as the histogram bars peak with a maximum value close to 1400. The timestamp 0.1 is considered the starting point of the *thread-forming* phase, while timestamp 0.5 marks the beginning of the *pre-tightening* phase. Overall, 6447 duplicate values are recorded from 1342 sequences. Although this is not an excessively high number of duplicate values, it is still significant. Moreover, a higher proportion of the duplicate values are recorded in the *pre-tightening* and *tightening 1.4* phases. The possible reason for this could be that as the torque is very high and increases quickly, the sensors might capture multiple observations at the same timestamp.

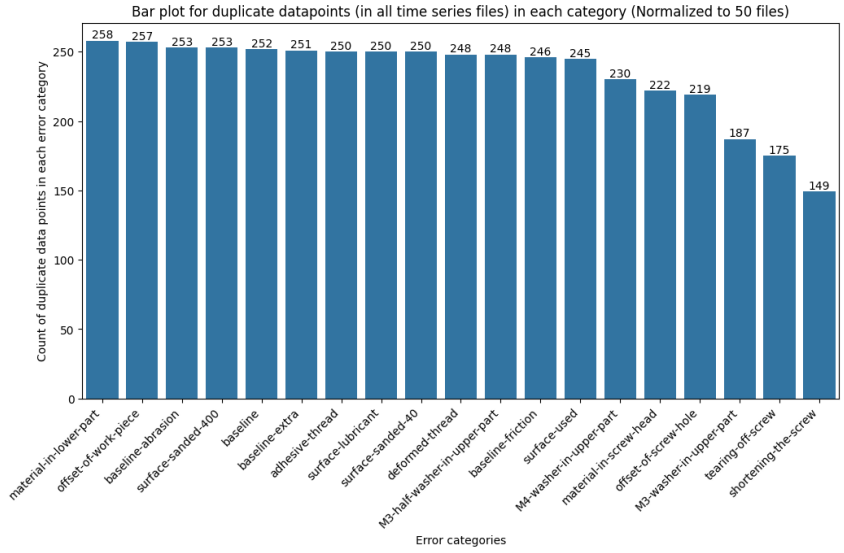


Figure 17: Count of duplicate values in each error category.

Figure 17 shows a bar plot where each bar represents the number of duplicate values observed in each error category. Since the number of sequences in each error category is not the same, all duplicate values from each category have been normalized to 50 sequences. The count of duplicate values ranges from 149 to 258 across all error categories. It can be observed that the error category *material-in-lower-part* has the highest number of duplicate values (258), while *shortening-the-screw* has the least number of duplicate values (149). Moreover, most error categories have an almost equal number of duplicate values in their time series sequences. Only the error categories *M3-washer-in-upper-part*, *tearing-off-screw*, and *shortening-the-screw* have fewer duplicate values compared to other categories. Additionally, it can be seen that the sub-categories of OK data have almost the same number of duplicate values (ranging from 246 to 253).

To handle the duplicate values, if multiple observations are recorded at the same timestamp, only the first recorded observation is considered for analysis. All other observations from the same timestamp are discarded.

5.4.2 Handling missing values

The screw-tightening process records an observation every 12 milliseconds. If the gap between any two consecutive timestamps is greater than 12 milliseconds, it indicates missing data between these timestamps. Figure 18 shows the histogram plot for missing data with respect to timestamps, plotted from 1342 screw-tightening processes.

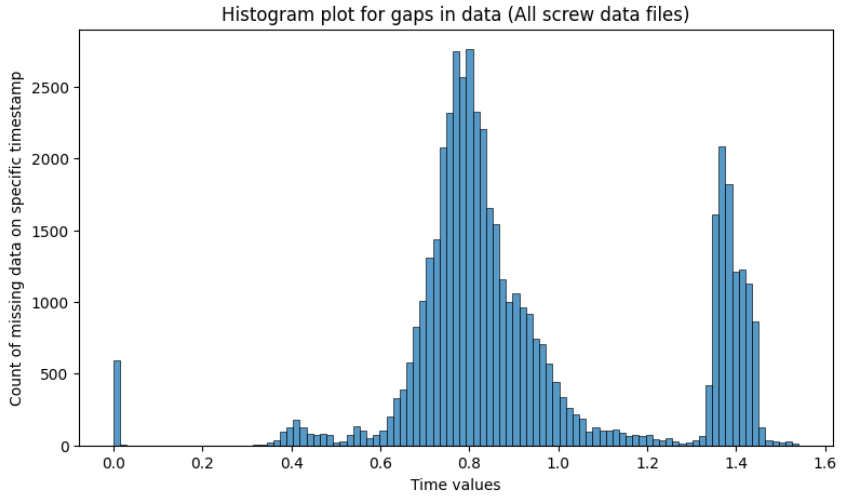


Figure 18: Missing values histogram plot.

From the histogram, it can be observed that almost all the missing values occur either at the beginning or at the end of the screw-tightening process, as indicated by the high bin size at timestamp 0 and the two bell-shaped curves between the time ranges 0.6 to 1 and 1.3 to 1.5. Since most screw-tightening runs end in the range of 0.7 to 0.9, the first bell curve clearly denotes that the missing values are at the end of the process. Additionally, some screw-tightening

processes are designed with defects, causing them to last longer than usual, which explains the second bell curve (1.3 to 1.5), indicating missing values at the end. Overall, 48,732 missing values are recorded from 1342 sequences. Although this is a significant number, since most missing values occur at the end of the process when the tightening device is trying to detach itself from the screw, it can be concluded that there is no substantial loss of information, and this can be handled using simple techniques. The bar plot in figure 19 shows the count of missing values recorded in each phase of screw-tightening process and it clearly evident that most of the missing values occur at the end of the process (*tightening 1.4*).

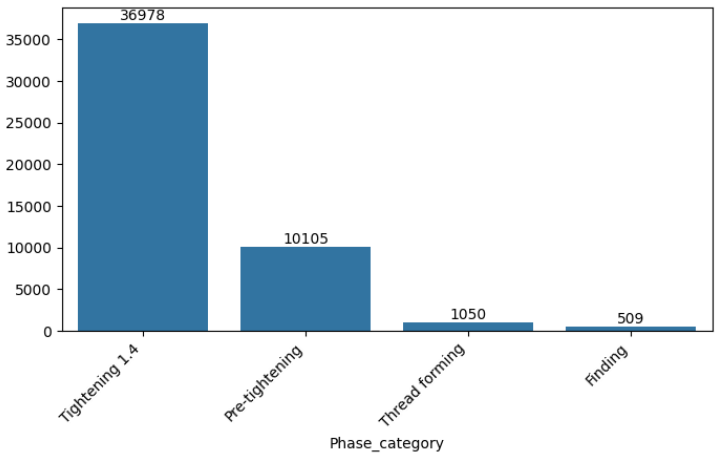


Figure 19: Missing values in each phase bar plot.

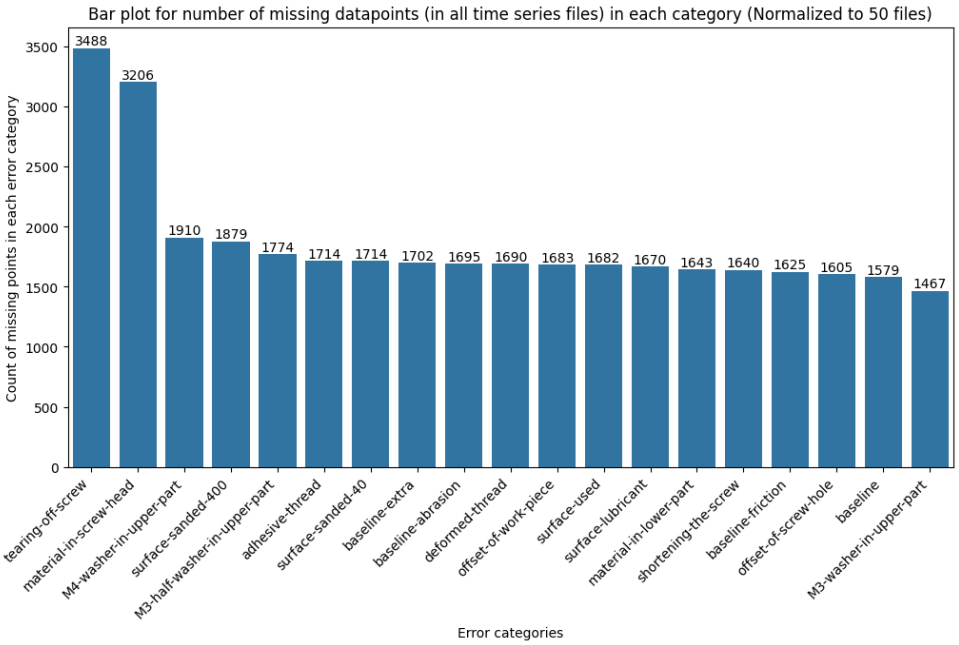


Figure 20: Count of missing values in each error category.

Additionally, in the histogram plot (figure 18), a small bell-shaped curve can be noticed between the time range of 0.3 to 0.5 seconds. Upon closer inspection, it is observed that almost all of the observations in this range belong to the error category *shortening-the-screw*. Due to the

manipulation in the process, some tightening experiments end during the *thread-forming* phase, contributing to the missing values in this part of the histogram.

Figure 20 shows a bar plot where each bar represents the number of missing values observed in each error category. Since the number of sequences in each error category is not the same, all missing values from each category have been normalized to 50 sequences for better comparison. The error category *tearing-off-screw* has the highest number of missing values, with a count of 3488, while the error category *M3-washer-in-upper-part* has the lowest number of missing values, with a count of 1467. Additionally, the error category *material-in-screw-head* has a high number of missing values (3206), similar to *tearing-off-screw*. All other error categories have a similar count of missing values, ranging from 1600 to 1700. All sub-categories of OK data also fall within this range.

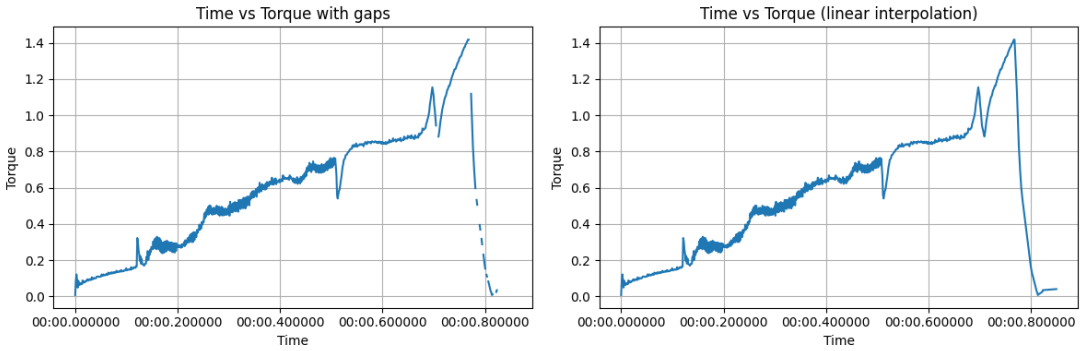


Figure 21: Linear interpolation to fill missing values

As the observations are recorded at very short time intervals (12 milliseconds) and the time series is linear, the linear interpolation method is used to fill the gaps or missing values in the dataset. The line plots in figure 21 show the torque values against time before and after applying the linear interpolation method. Figure 21(a) shows the time series with missing values, and it can be noticed that gaps occur only at the end of the process or in the *tightening 1.4* phase. Figure 21(b) shows the time series after filling in the missing values.

5.4.3 Cutting/padding time series sequences

As each screw-tightening process does not follow a perfectly constant time to fit the screw into the base component, and due to manipulations made to create errors, each screw-fitting process takes different amounts of time to complete. The table 4 below shows the minimum, maximum, and median values for the recorded time series in each error category. It can be noted from the table that the longest time series sequence contains 1,286 data points recorded during the tightening process, while the shortest time series sequence contains about 299 data points. The longest sequence is recorded in the *offset-of-screw-hole* error category, and the shortest sequence is recorded in the *shortening-the-screw* error category.

Error category	Minimum	Maximum	Median
Adhesive-thread	634	1212	706
Deformed-thread	600	821	684.5
M3-half-washer-in-upper-part	474	1213	709.5
M3-washer-in-upper-part	1143	1174	1157
M4-washer-in-upper-part	467	1214	686
Material-in-lower-part	773	1141	935.5
Material-in-screw-head	601	1220	1201
Offset-of-screw-hole	1145	1286	1160
Offset-of-work-piece	586	859	742
Shortening-the-screw	299	1212	376
Surface-lubricant	697	864	771.5
Surface-sanded-40	640	1202	730
Surface-sanded-400	576	1213	719
Surface-used	581	1211	712
Tearing-off-screw	347	1215	1212

Table 3: Error categories time series sequence lengths

From table 3, we can also notice that in the error categories *shortening-the-screw* and *tearing-off-screw*, the range of time series sequence lengths is higher compared to other categories. Some time series sequences are very short with 300 observations, while others are very long with 1,215 observations. Moreover, in the error categories *M3-washer-in-upper-part* and *offset-of-screw-hole*, all the time series are longer compared to other error categories, with sequences ranging from 1,143 to 1,286.

baseline sub-category	Minimum	Maximum	Median
Baseline	557	814	670
Baseline-abrasion	547	775	661
Baseline-friction	613	852	672
Baseline-extra	663	958	820

Table 4: Baseline sub-categories sequence lengths

The table 4 contains information about the minimum, maximum, and median values of the sub-categories of OK data. In the case of baseline sub-categories, the time series sequences range from 547 to 958. Additionally, the sub-categories *baseline-extra* and *baseline-friction* generally have longer sequence lengths compared to sub-categories *baseline* and *baseline-abrasion*.

For modeling time series data, it is preferred to have all the time series sequences at the same length. Hence, a cutting/padding technique is used to trim or add extra observations to each time series. Based on the analysis, a value of 1,200 is decided upon as the constant time series length. By setting this constant length, not much information is lost from the longer sequences, which is useful for distinguishing the error categories in clustering approaches. Therefore, sequences longer than 1,200 are cut at the end, and sequences shorter than 1,200 have a constant value of 0 added at the beginning. Adding 0 at the beginning does not change the time series but simply shifts it slightly to the right compared to other sequences. Distance metrics such as DTW are

used in the modeling approach to effectively handle time-varying series and group them into similar error categories.

5.4.4 Torque normalization

The torque sequences used for modeling are normalized using min-max normalization. The minimum and maximum values are determined based on all the time series sequences used in the process, establishing global minimum and maximum values for normalization. The minimum torque value is set to 0, as the torque begins to increase from 0 when the process starts. The maximum torque value is determined to be 1.574 from all the time series sequences. This normalization helps improve model performance, prevents issues like vanishing gradients during neural network training, and facilitates faster convergence.

5.5 Data analysis and hypothesis

In this section, time series from different categories are compared with other categories to determine whether their data distributions are similar. Specifically, QQ plots and the Mann-Whitney U test are used to compare the time series sequences, and the results are interpreted in detail. Moreover, hypotheses are formulated to verify using the results of various semi-supervised and unsupervised models in the next section.

5.5.1 Comparison between baseline sub-categories

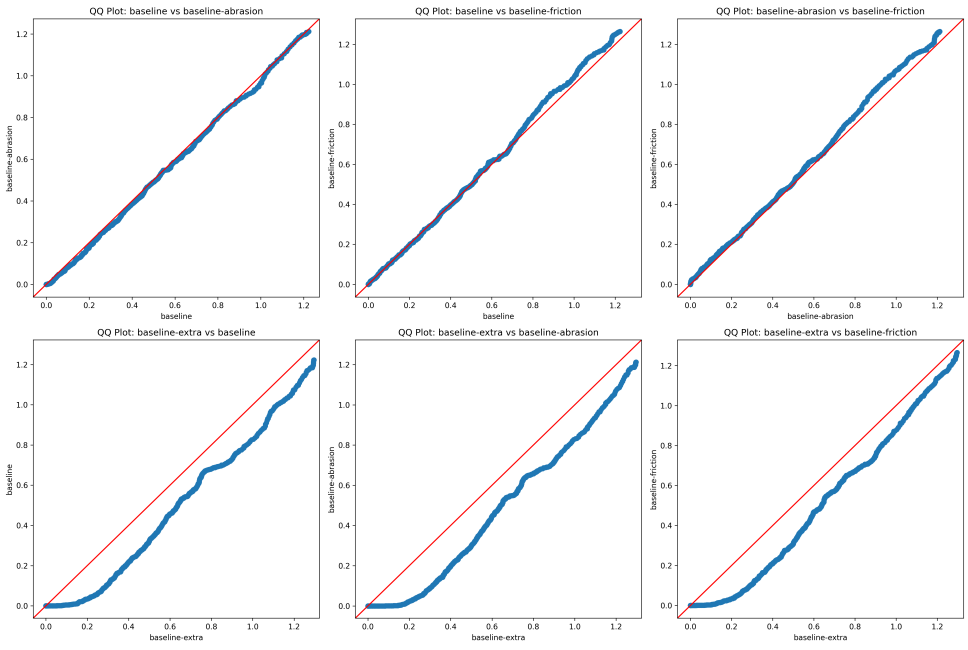


Figure 22: QQ-plots for sub-categories of OK sequences

Figure 22 consists of multiple QQ plots. Each QQ plot represents the comparison between pairs of sub-categories in the OK sequences. For example, in the top left plot, the x-axis contains the average values of time series from the *baseline* sub-category, and the y-axis contains the average values of time series from the *baseline-abrasion* sub-category. If the time series data from both distributions follow the same distribution, the blue line will overlap with the diagonal red line. If they don't follow the same distribution, the blue line will deviate from the diagonal line.

The QQ plots in the first row show that the pairs of sub-categories — *baseline*, *baseline-abrasion*, and *baseline-friction* are generated from the same distribution and are similar. In the second row, the *baseline-extra* sub-category is compared with the other three sub-categories of OK data. We can see that the blue line significantly deviates from the diagonal line, indicating that the data distribution of *baseline-extra* is different from the other sub-categories of OK data.

To verify these observations, a Mann-Whitney U test is also performed. In the case of the Mann-Whitney U test, if the p-value is less than 0.05, there is enough evidence to say that the data distributions of the two groups are different, and the null hypothesis is rejected.

sub-category 1	**sub-category 2**	**MW U-test statistic**	**P-value**
Baseline	Baseline-abrasion	516445.5	2.008587e-01
Baseline	Baseline-friction	492989.0	5.863207e-01
Baseline-abrasion	Baseline-friction	475947.5	6.166942e-02
Baseline-extra	Baseline	620158.0	1.254169e-20
Baseline-extra	Baseline-abrasion	634188.0	2.379743e-25
Baseline-extra	Baseline-friction	615914.5	2.701107e-19

Table 5: Mann Whitney test results for baseline sub-categories

Table 5 presents the results of the Mann-Whitney U test performed on all possible pairs of sub-categories of OK data. It can be observed that the p-values for the first three rows are greater than 0.05, indicating that the data from the sub-categories *baseline*, *baseline-abrasion*, and *baseline-friction* follow the same distribution. Moreover, in the last three rows, a very low p-value (< 0.05) is observed, indicating that the data distribution of *baseline-extra* is completely different from the other sub-categories of OK data.

From the above observations, we can hypothesize that if *baseline-extra* is used as OK data for training semi-supervised models, the model might not capture the patterns from OK sequences efficiently. As there are two different distributions of OK sequences, a significant number of misclassifications might occur during the testing phase, with some proportion of anomaly sequences being classified as OK sequences.

5.5.2 OK category vs multiple surface error categories

Here, OK time series data is compared with several surface-related error categories, namely surface-*sanded-40*, *surface-sanded-400*, *surface-lubricant*, and *surface-used*. In the case of OK sub-categories, *baseline-extra* has been observed to have a different data distribution and is,

therefore, eliminated from this part of the analysis. In this context, the term baseline denotes the average value of all time series sequences from *baseline*, *baseline-abrasion*, and *baseline-friction*.

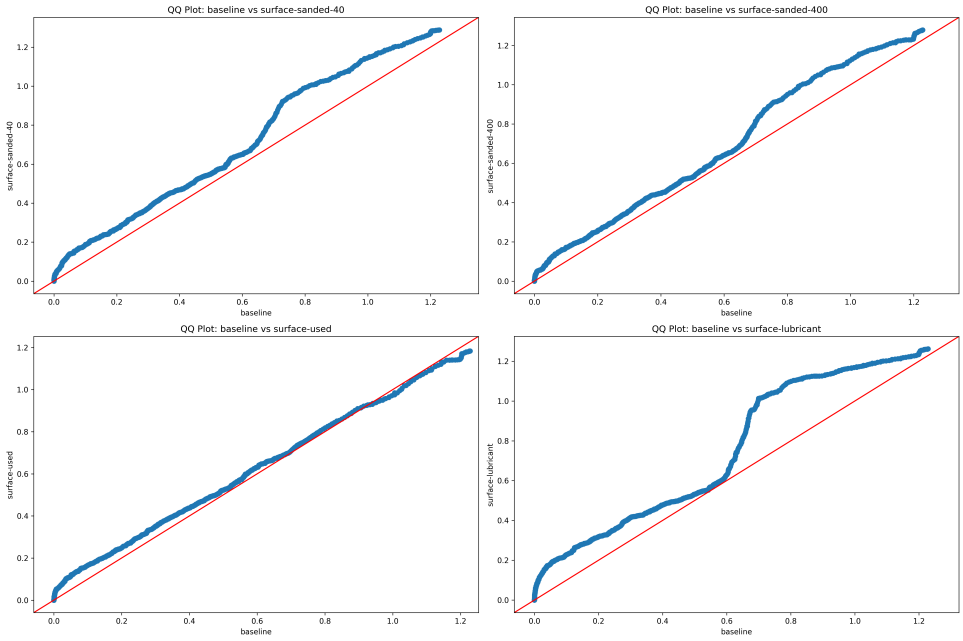


Figure 23: QQ-plots for baseline vs surface error categories

Figure 23 shows four QQ plots, where each surface-related error category is compared against the baseline to check if they follow the same distribution. In the first row, the baseline category is compared with *surface-sanded-40* and *surface-sanded-400*, showing that they follow a nearly similar distribution from the start to the middle of the time series sequence. However, the distribution differs later on. In the QQ plot of *baseline* vs. *surface-used*, a perfect overlap of the blue line with the diagonal line is noticed, indicating that the two distributions are similar. Finally, the QQ plot between the *baseline* and *surface-lubricant* shows that they are entirely different distributions with very small similarities in the middle of the sequence.

Ok category	**Error category**	**MW U-test statistic**	**P-value**
Baseline	Surface-sanded-40	422465.0	1.907281e-09
Baseline	Surface-sanded-400	437486.0	1.284930e-06
Baseline	Surface-used	449111.5	8.093488e-05
Baseline	Surface-lubricant	440943.0	4.651647e-06

Table 6: Mann Whitney test results for baseline vs surface error categories

From table 6, we can see that the p-values for all comparisons are less than 0.05, indicating that surface-related error categories have different distributions compared to the baseline. This can be observed from the QQ plots as well. However, in the case of *baseline* vs. *surface-used*, even though the test indicates they are different, the QQ plots clearly show they are similar.

From the above observations, we can hypothesize that semi-supervised models might struggle a bit in detecting *surface-sanded-40* and *surface-sanded-400* time series as anomalies. We can

also expect a lot of misclassifications in the case of *surface-used*, where a significant proportion of *surface-used* sequences might be classified as OK sequences.

6 Validation and Results

In this subsection, various supervised, semi-supervised, and unsupervised methods are applied to the screw-tightening dataset described above. The results are visualized with appropriate graphs and tables and are interpreted in detail.

6.1 Binary classification (Semi-supervised methods)

In this part, the results of the semi-supervised models, namely, Isolation Forest, One-Class SVM, and Autoencoder, are explained in detail. All the models are trained twice: once including *baseline-extra* in the OK sequence data, and once without *baseline-extra*. In addition, the results of the weighted ensemble model are presented and compared with the results of the individual models.

6.1.1 Isolation forest results

In the training phase of the Isolation Forest, the model is trained with the OK screw-tightening sequences to learn the data distribution. To find the best hyperparameters for the model, a grid search CV is used with the set of values for each hyperparameter mentioned in table 7.

Hyperparameter	**values**
n_estimators	[80, 100, 200]
max_samples	['auto', 0.5, 0.7]
contamination	['auto']
bootstrap	[True, False]
n_jobs	[None,1,2,3,4]
random_state	[None,10,42]
warm_start	[True,False]

Table 7: Isolation forest hyperparameters.

After evaluating all possible combinations, the best parameters identified for the Isolation Forest model were: *bootstrap* set to *True*, *contamination* set to *auto*, *max_samples* set to *auto*, *n_estimators* set to 80, *n_jobs* set to *None*, *random_state* set to *None*, and *warm_start* set to *True*. These parameters were found to provide the optimal performance for anomaly detection on the screw-tightening dataset.

Without baseline-extra

Table 8 below shows the classification report of the Isolation Forest model trained without the *baseline-extra* sub-category. The overall accuracy and macro average F1 score of the model

	Precision	Recall	F1-score	Support
Error class	0.92	0.77	0.84	842
Normal class	0.64	0.85	0.73	400
Accuracy			0.80	1242
Macro avg	0.78	0.81	0.78	1242
Weighted avg	0.83	0.80	0.80	1242

Table 8: Isolation forest classification report (without baseline-extra).

are 0.80 and 0.78, respectively. For the error class, the precision is very high (0.92) and the recall is also good (0.77). For the normal class, the recall score is high (0.85) but the precision is comparatively low (0.64). A significant proportion of anomalies are classified as normal, affecting the precision for this class.

Table 9 shows the error categories and their counts that are classified as normal even though they are anomalies. As expected, nearly half of the sequences from each surface-related error category (except *surface-lubricant*) are misclassified. A large proportion of the *deformed-thread* error category is also classified as normal.

Error Category	Count
Deformed-thread	47
Surface-used	46
Surface-sanded-40	23
Surface-sanded-400	22
M3-half-washer-in-upper-part	17
Offset-of-work-piece	17
Material-in-screw-head	13
M4-washer-in-upper-part	7
Adhesive-thread	2

Table 9: Isolation forest misclassification of anomaly sequences (without baseline-extra).

Error Category	Count
Baseline-abrasion	24
Baseline-friction	19
Baseline	16

Table 10: Isolation forest misclassification of normal sequences (without baseline-extra).

Table 10 shows the count of normal sequences classified as anomalies. Even though the model is trained using these sequences, because of the contamination rate and slight differences between the sequences, they are classified as anomalies. The count does not seem very high, as only 23% of all OK sequences are classified as anomalies.

With baseline-extra

To determine if the results of the models are affected by including *baseline-extra* in the dataset, the Isolation Forest model is trained with the same parameters again, and the results are provided below. Table 11 shows the classification report of the Isolation Forest model trained with all the time series sequences including *baseline-extra*. From the accuracy score, it is evident that including *baseline-extra* negatively impacts the overall results of the model, as the accuracy scores dropped from 80% to 67%. A significant drop in the recall score of the error class and the precision score of the normal class can also be observed.

	Precision	Recall	F1-score	Support
Error class	0.85	0.57	0.68	842
Normal class	0.54	0.83	0.65	500
Accuracy			0.67	1342
Macro avg	0.69	0.70	0.67	1342
Weighted avg	0.73	0.67	0.67	1342

Table 11: Isolation forest classification report (with baseline-extra).

Error Category	Count
Surface-used	81
Deformed-thread	80
Surface-sanded-40	40
Surface-sanded-400	38
Offset-of-work-piece	35
M3-half-washer-in-upper-part	26
Material-in-screw-head	18
Surface-lubricant	18
M4-washer-in-upper-part	12
Adhesive-thread	7
Material-in-lower-part	4

Table 12: Isolation forest misclassification of anomaly sequences (with baseline-extra).

Error Category	Count
Baseline-extra	66
Baseline-abrasion	10
Baseline	6
Baseline-friction	4

Table 13: Isolation forest misclassification of normal sequences (with baseline-extra).

Moreover, from table 13, we can see that 66 out of 100 sequences of *baseline-extra* are classified as anomalies, and the misclassifications in other sub-categories are reduced compared to the previous model results. However, in the case of error categories, the misclassifications increased significantly. This can be observed from the values in table 12. 80% of time series sequences from *surface-used*, *surface-sanded-40*, and *deformed-thread* anomalies are classified as normal. Additionally, 65-70% of *surface-sanded-400* and *offset-of-work-piece* sequences are classified as normal. The number of misclassifications in other error categories also increased.

6.1.2 One-class SVM results

Hyperparameter	values
kernel	['rbf', 'poly', 'sigmoid']
gamma	['auto', 'scale']
nu	[0.1, 0.2, 0.5]
degree	[2, 3, 4, 5]
shrinking	[True, False]

Table 14: One-class SVM hyperparameters.

Similar to the Isolation Forest, the One-Class SVM model is also trained on the OK samples. During the testing phase, both OK and anomalous samples are used as input to the model to

obtain the final results. To find the best hyperparameters for accurately classifying the anomaly and OK sequences, a list of values mentioned in table 14 was used in the grid search CV. After evaluating all possible combinations with 5-fold cross-validation, the best hyperparameter values identified are as follows: *degree* set to 2, *gamma* set to *auto*, *kernel* set to *rbf*, *nu* set to 0.1, and *shrinking* set to *True*. Using these optimal hyperparameters, a final One-Class SVM model was trained on the training data.

Without baseline-extra

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.94	0.67	0.78	842
Normal class	0.57	0.91	0.70	400
Accuracy			0.75	1242
Macro avg	0.75	0.79	0.74	1242
Weighted avg	0.82	0.75	0.76	1242

Table 15: One-class SVM classification report (without baseline-extra).

The classification report in table 15 shows the results of the One-Class SVM model trained on OK samples without *baseline-extra*. The overall accuracy of the model is 0.75, indicating the model correctly predicts 3 out of 4 times. The model shows a high precision score (0.94) in the error class, denoting strong confidence in identifying anomalous sequences. However, the recall score is low (0.67), indicating it misses identifying some anomalies in the dataset. Conversely, for the normal class, the model shows a high recall (0.91) and low precision (0.57). This shows that while it correctly identifies most normal sequences, it also incorrectly predicts a significant proportion of anomalies as normal.

Error Category	**Count**
Deformed-thread	67
Surface-used	65
Offset-of-work-piece	30
M3-half-washer-in-upper-part	26
Surface-sanded-400	24
Surface-sanded-40	22
Material-in-screw-head	18
Surface-lubricant	9
Adhesive-thread	8
M4-washer-in-upper-part	5
Material-in-lower-part	2

Table 16: One-class SVM misclassification of anomaly sequences (without baseline-extra).

Error Category	**Count**
Baseline-abrasion	22
Baseline-friction	8
Baseline	8

Table 17: One-class SVM misclassification of normal sequences (without baseline-extra).

Tables 16 and 17 show the misclassifications, making it easier to see which error categories the model struggles with most. Compared to the Isolation Forest model, the overall count of misclassifications is higher for the One-Class SVM. The model struggles significantly to identify

anomalies in the categories of *deformed-thread*, *surface-used*, *offset-of-work-piece*, and *M3-half-washer-in-upper-part*. In contrast, fewer misclassifications are observed for *M4-washer-in-upper-part* (5) and *material-in-lower-part* (2). Misclassifications in the OK samples are very low overall, with very few errors in the *baseline-friction* (8) and *baseline* sub-categories (8).

With baseline-extra

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.87	0.38	0.53	842
Normal class	0.46	0.90	0.61	500
Accuracy			0.58	1342
Macro avg	0.66	0.64	0.57	1342
Weighted avg	0.72	0.58	0.56	1342

Table 18: One-class SVM classification report (with baseline-extra).

Table 18 shows the classification report of the One-Class SVM model trained with the same hyperparameters but including the *baseline-extra* subcategory in the training sample. The overall accuracy is 0.58, which is 17% lower than the One-Class SVM model without *baseline-extra*, indicating a significant decrease in performance. The recall score for the error class is very low (0.38), and there is a significant drop in the precision score for the normal class (0.46).

Moreover, table 20 shows that the overall misclassifications of the normal class. The values are lower compared to the Isolation Forest model. However, the misclassifications in several error categories (in table 19) are very high. Almost all sequences from *deformed-thread*, *surface-used*, *offset-of-work-piece*, *surface-sanded-400*, *surface-sanded-40*, *M3-half-washer-in-upper-part*, and *surface-lubricant* are misclassified, indicating very poor model performance. Fewer misclassifications are observed in the error category *material-in-lower-part*, with a count of 17.

Error Category	**Count**
Deformed-thread	93
Surface-used	91
Offset-of-work-piece	48
Surface-sanded-400	46
Surface-sanded-40	45
Surface-lubricant	42
M3-half-washer-in-upper-part	41
Adhesive-thread	34
M4-washer-in-upper-part	22
M3-washer-in-upper-part	21
Material-in-screw-head	20
Material-in-lower-part	17

Table 19: One-class SVM misclassification of anomaly sequences (with baseline-extra).

Error Category	**Count**
Baseline-extra	23
Baseline-abrasion	20
Baseline	5
Baseline-friction	2

Table 20: One-class SVM misclassification of normal sequences (with baseline-extra).

6.1.3 Autoencoder results

The autoencoder uses the training data to learn the data representation by compressing it into a latent representation and then reconstructing the data in such a way that reconstruction errors are minimal. During the testing phase, anomalous sequences will have higher reconstruction errors since they do not follow the normal data representation and can thus be identified as anomalies based on a threshold value. The hyperparameters used to train the autoencoder model are listed in table 21. After evaluation using grid search CV, the best hyperparameters identified are: *batch_size* set to 32, *epochs* set to 100, *loss* set to *mse*, and *optimizer* set to *adam*. Using these optimal hyperparameters, the autoencoder model was retrained on the OK samples, and the optimal threshold value was determined based on the AUC-ROC curve.

Hyperparameter	values
loss	['mse', 'binary_crossentropy', 'mae']
optimizer	[Adam, SGD, RMSprop]
batch_size	[16, 32, 64]
epochs	[50, 100]
learning_rate	[0.001, 0.01]
activation	['relu', 'tanh', 'sigmoid']

Table 21: Autoencoder hyperparameters.

Without baseline-extra

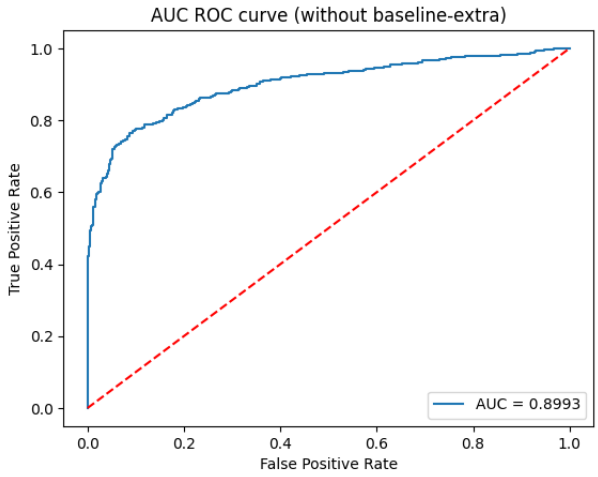


Figure 24: AUC ROC curve for Autoencoder model (without baseline-extra).

	Precision	Recall	F1-score	Support
Error class	0.95	0.77	0.85	842
Normal class	0.65	0.91	0.76	400
Accuracy			0.81	1242
Macro avg	0.80	0.84	0.80	1242
Weighted avg	0.85	0.81	0.82	1242

Table 22: Autoencoder classification report (without baseline-extra).

Figure 24 shows the AUC-ROC curve plotted for the autoencoder model (without *baseline-extra*). It can be observed that the area under the curve is 0.899, which is very good, and based on the maximum difference between the true positive rate and false positive rate, the best threshold is decided to separate the anomalous data from normal data. The optimal threshold was found to be 0.022. The classification report in table 22 shows the results from the autoencoder model trained by excluding *baseline-extra* from the OK samples. The overall accuracy of the model is 0.81, which is higher compared to the Isolation Forest and One-Class SVM models. In general, the results of the autoencoder model are similar to those of the Isolation Forest model (without *baseline-extra*), but the autoencoder model outperforms the Isolation Forest model. The precision score for the error class is the highest at 0.95, and the recall score (0.77) also looks good for the error class. For the normal class, the recall score is very high at 0.91, though the precision score is a bit lower at 0.65. For our use case, a model with the highest recall for the error class is preferred because we want to identify as many anomalies as possible in screw-tightening. As a result, we might expect more misclassifications in the normal class.

Error Category	**Count**
Surface-used	54
Deformed-thread	46
Offset-of-work-piece	26
Surface-sanded-40	25
Surface-sanded-400	22
Material-in-screw-head	16
Material-in-lower-part	4
M3-half-washer-in-upper-part	1

Table 23: Autoencoder misclassification of anomaly sequences (without baseline-extra).

Error Category	**Count**
Baseline-abrasion	16
Baseline-friction	13
Baseline	8

Table 24: Autoencoder misclassification of normal sequences (without baseline-extra).

Tables 23 and 24 show the misclassification counts in the OK sequences and different error categories. Nearly half of the time series sequences from *surface-used*, *deformed-thread*, *surface-sanded-40*, and *offset-of-work-piece* are misclassified. Additionally, only one sequence from *M3-half-washer-in-upper-part* is misclassified. In general, the number of misclassifications in the OK samples is very low.

With baseline-extra

Figure 25 shows the AUC-ROC curve plotted for the autoencoder model (with *baseline-extra*). It can be observed that the area under the curve is 0.8615, and the optimal threshold to separate the anomalous data is found to be 0.019. Table 25 shows the results of the autoencoder model trained by including *baseline-extra* in the OK samples. The results indicate that the precision, recall, and accuracy scores are not significantly different from the autoencoder model (without *baseline-extra*). Hence, it can be stated that the autoencoder model is more stable even when including data from *baseline-extra* compared to other models.

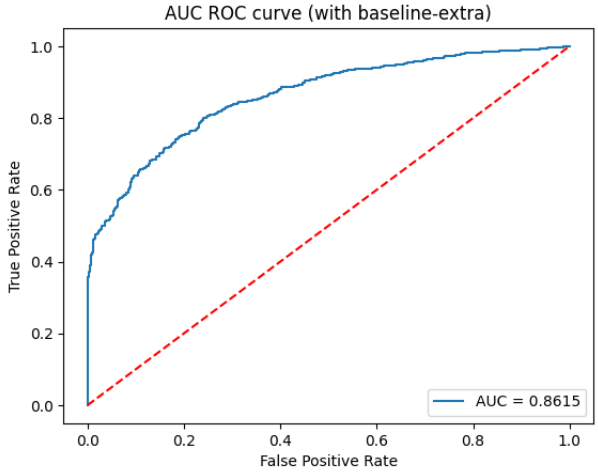


Figure 25: AUC ROC curve for Autoencoder model (with baseline-extra).

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.85	0.80	0.82	842
Normal class	0.69	0.76	0.73	500
Accuracy			0.79	1342
Macro avg	0.77	0.78	0.77	1342
Weighted avg	0.79	0.79	0.79	1342

Table 25: Autoencoder classification report (with baseline-extra).

The misclassification of error categories can be observed in table 26. The error categories and their counts are similar to those of the autoencoder model (without *baseline-extra*). Additionally, the misclassifications of the OK samples are represented in table 27, and all sub-categories have slightly more misclassifications compared to the other autoencoder model. Interestingly, 73 sequences from *baseline-extra* were classified as anomalies. This is a positive sign because, given that *baseline-extra* has a different data distribution than the other baseline categories, the autoencoder treats them as anomalies in general. Hence, even though it is used in training, the autoencoder model does not regard them as normal data.

Error Category	**Count**
Surface-used	47
Deformed-thread	45
Surface-sanded-400	23
Offset-of-work-piece	22
Surface-sanded-40	17
Material-in-screw-head	11
Material-in-lower-part	2
M3-half-washer-in-upper-part	1

Table 26: Autoencoder misclassification of anomaly sequences (with baseline-extra).

Error Category	**Count**
Baseline-extra	73
Baseline-abrasion	19
Baseline-friction	18
Baseline	10

Table 27: Autoencoder misclassification of normal sequences (with baseline-extra).

6.1.4 stacked bar charts and color categories

Figures 26 and 27 present stacked bar charts depicting the recall values for each error category across three different semi-supervised models, trained with and without *baseline-extra*, respectively. The recall value for each error category is obtained by testing the model with a dataset that includes all the OK sequences and only one error category sequence at a time. This means that for each error category, a specific test set is created, combining all the OK sequences and sequences from only that particular error category.

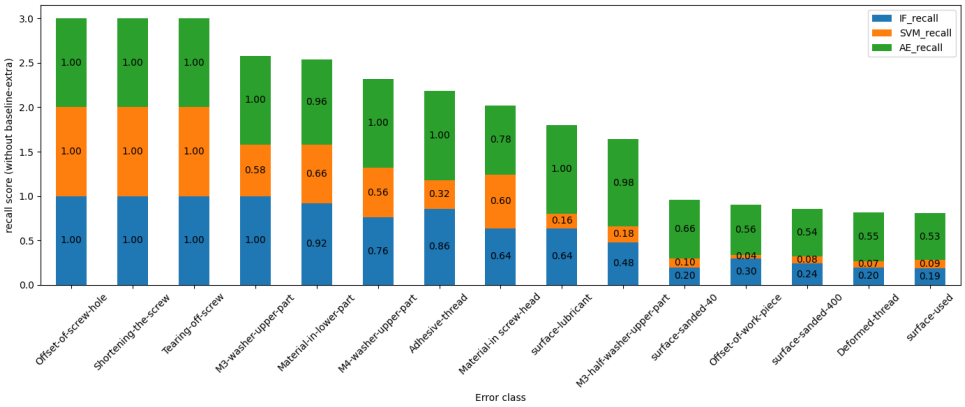


Figure 26: Stacked bar chart (with baseline-extra).

In figure 26 (recall scores with *baseline-extra*), we observe that the error categories *offset-of-screw-hole*, *shortening-the-screw*, and *tearing-off-screw* achieve a recall score of 1 across all three semi-supervised models. For the *M3-washer-in-upper-part* error category, the autoencoder and isolation forest models both achieve a recall score of 1, whereas the one-class SVM achieves only 0.58. The autoencoder model performs well in identifying the other errors like *M3-washer-in-upper-part*, *adhesive-thread*, *surface-lubricant*, *M3-half-washer-in-upper-part*, and *material-in-lower-part*. However, all models perform poorly in identifying error categories such as *surface-sanded-40*, *offset-of-work-piece*, *surface-sanded-400*, *deformed-thread*, and *surface-used*, as indicated by their low recall values. Among these, the one-class SVM performs the worst, with recall values below 0.10.

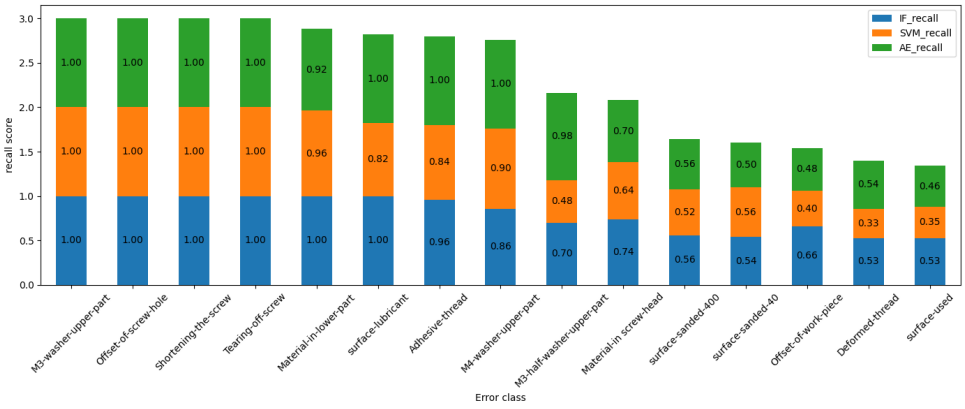


Figure 27: Stacked bar chart (without baseline-extra).

Figure 27 shows the recall values estimated using the same approach, but this time *baseline-extra* is eliminated from the OK sequences. It is evident that the recall scores improve significantly, as indicated by the longer bars. All three models perform exceptionally well in identifying error categories such as *M3-washer-in-upper-part*, *offset-of-screw-hole*, *shortening-the-screw*, *tearing-off-screw*, *material-in-lower-part*, *surface-lubricant*, *adhesive-thread*, and *M4-washer-in-upper-part*, with recall scores ranging from 0.82 to 1. However, the models still struggle to perfectly identify error categories such as *surface-sanded-40*, *offset-of-work-piece*, *surface-sanded-400*, *deformed-thread*, and *surface-used*, with recall values around 0.50 or less. Nonetheless, the scores are better compared to those in figure 26. These results indicate that *baseline-extra* has a different data distribution, affecting the overall model performance, and therefore, it is removed from the dataset.

Green	**Orange**	**Red**
M3-washer-in-upper-part	Adhesive-thread	Offset-of-work-piece
Offset-of-screw-hole	M4-washer-in-upper-part	Surface-sanded-400
Shortening-the-screw	M3-half-washer-in-upper-part	Deformed-thread
Tearing-off-screw	Material-in-screw-head	Surface-used
Surface-lubricant	Material-in-lower-part	Surface-sanded-40

Table 28: Color categories

For an easier interpretation of the results, the error categories are classified into three color categories: green, orange, and red. The green category includes errors that are easily identified as anomalies, the orange category includes errors that are somewhat difficult to identify, and the red category includes errors that are very hard to distinguish from the OK samples. The recall scores are used to separate the errors into these color categories. From the stacked bar chart in figure 27, if two out of three models have a recall score of 1 for an error category, it is classified as green. If the recall score is between 0.7 and 0.99, it is classified as orange, and if the recall value is less than 0.7, it is classified as red. The table 28 details the error categories and their corresponding color groups.

6.1.5 Weighted ensemble learning results

Isolation Forest, one-class SVM, and Autoencoder models trained on the dataset without *baseline-extra* are used for ensemble learning. To create the weighted ensemble model, we first obtain the anomaly scores from the decision functions of Isolation Forest and one-class SVM. In these models, more negative scores indicate a higher likelihood of being an anomaly. To bring both decision function scores to the same scale, min-max normalization is applied to convert the values to a range of 0 to 1.

Next, we obtain the reconstruction errors from the Autoencoder model. Since higher reconstruction errors indicate a higher likelihood of being an anomaly, which is the opposite of the decision function scores of Isolation Forest and one-class SVM, we apply min-max normalization to these

errors and then subtract the normalized values from 1. This inversion ensures that all values are now on the same scale, where lower values indicate a higher likelihood of being an anomaly.

To find the best possible weights for each model, we experiment with all possible combinations of values from 0 to 1 for the three models, selecting the combination that maximizes the macro-average F1 score. Additionally, we determine the best threshold for classifying points as anomalous or non-anomalous, again aiming to maximize the macro-average F1 score.

The optimal weights for each model are determined to be: one-class SVM – 0.1, Isolation Forest – 0.1, and Autoencoder – 0.8. The best threshold obtained is 0.95. Therefore, final scores below 0.95 are considered anomalies, and scores above 0.95 are considered normal sequences.

After obtaining the optimal weights and threshold values, the models are combined using the weighted ensemble method to test their performance in predicting anomalies. The weighted ensemble model is evaluated using datasets created based on color categories, where each dataset contains the OK samples and the error categories belonging to a specific color category. Finally, the model is tested with a comprehensive dataset that includes all OK samples and all error categories. The classification reports from these tests are displayed in the tables below.

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.80	1.00	0.89	242
Normal class	1.00	0.84	0.92	400
Accuracy			0.90	642
Macro avg	0.90	0.92	0.90	642
Weighted avg	0.92	0.90	0.90	642

Table 29: Weighted ensemble classification report on Green category.

Table 29 shows the classification report for the green color category. The model performs exceptionally well in identifying anomalies in this category, achieving an accuracy of 0.90. The recall for the error class is 1, indicating that the model can classify these errors without a single misclassification. Moreover, the precision for the normal class is 1, demonstrating the model's confidence in predicting sequences from the OK class. Other scores are also greater than 0.80, further indicating strong model performance.

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.80	0.95	0.87	250
Normal class	0.97	0.85	0.91	400
Accuracy			0.89	650
Macro avg	0.88	0.90	0.89	650
Weighted avg	0.90	0.89	0.89	650

Table 30: Weighted ensemble classification report on Orange category.

Table 30 contains the classification report for the orange color category. The overall accuracy is 0.89, just 1% less than the green category. The precision and recall values are similar but slightly

lower than those for the green category. The recall scores for the error and normal classes are 0.95 and 0.85, respectively. This indicates that the model performs well in detecting anomalies from the orange category, with only a few errors.

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.75	0.59	0.66	350
Normal class	0.70	0.83	0.76	400
Accuracy			0.72	750
Macro avg	0.73	0.71	0.71	750
Weighted avg	0.73	0.72	0.72	750

Table 31: Weighted ensemble classification report on Red category.

Table 31 provides the classification report for the red color category. The recall score for the error class is 0.59, slightly higher than 0.50. This suggests that the model struggles to correctly identify anomalies in this category, as the sequences are more similar to the OK samples. The precision for the error and normal classes are 0.75 and 0.70, respectively. The overall accuracy is 72%, which is lower compared to the other color categories. In summary, the model performs very well in detecting anomalies in the green and orange categories but performs poorly in the red category.

	Precision	**Recall**	**F1-score**	**Support**
Error class	0.92	0.81	0.86	842
Normal class	0.68	0.86	0.76	400
Accuracy			0.83	1242
Macro avg	0.80	0.83	0.81	1242
Weighted avg	0.85	0.83	0.83	1242

Table 32: Weighted ensemble classification report on entire dataset.

Finally, Table 32 shows the overall results of the weighted ensemble model when tested on the entire dataset. The overall accuracy is 0.83, which is 2% higher than our best-performing autoencoder model. Additionally, the precision for the normal class and recall for the error class have improved, while the other scores have dropped slightly. Therefore, the weighted ensemble model can be used as the final model for classifying screw-tightening sequences into anomalous or non-anomalous, as it is more stable and effectively combines predictions from multiple models.

6.2 Supervised methods

In this subsection, the results of all the supervised models used for multi-class error classification in this thesis are presented through appropriate plots and interpreted in detail. Although the primary focus of this thesis is on evaluating different unsupervised approaches for multiple errors or anomaly detection, a brief analysis of supervised methods is conducted to compare these results with those of the unsupervised methods in the next subsection.

Specifically, six supervised models are chosen and implemented in this thesis. These models are selected to represent each family of supervised models. The KNN DTW model is chosen from the distance-based methods, while the Time Series Forest (TSF) model represents the interval-based methods. The ROCKET classifier is selected from the kernel-based methods, and the catch22 classifier is chosen from the feature-based methods. The shape transform classifier is taken from the shapelet-based methods, and finally, the CNN classifier represents the deep learning methods. This comparative analysis aims to provide a comprehensive understanding of how supervised models perform in multi-class error classification on our screw-tightening dataset.

For all supervised models, the dataset is split such that 80% of the data is used for training and 20% is used for testing. To ensure that the train and test sets have the same proportion of each error class, the *stratify* parameter is used in the *train_test_split* function.

6.2.1 KNN DTW results

In the training phase of the KNN DTW classifier, a grid search cross-validation (CV) is performed using the following list of values for the hyperparameters: *n_neighbors* is experimented with values [1, 5, 10], and *weights* is experimented with values *['uniform', 'distance']*. After a 5-fold cross-validation in grid search CV, the best hyperparameters are found to be *n_neighbors*: 10 and *weights*: *distance*. The classification report for multi-class error classification using this distance-based method is shown in figure 28. The overall accuracy of the model is 0.56, which is considered as low.

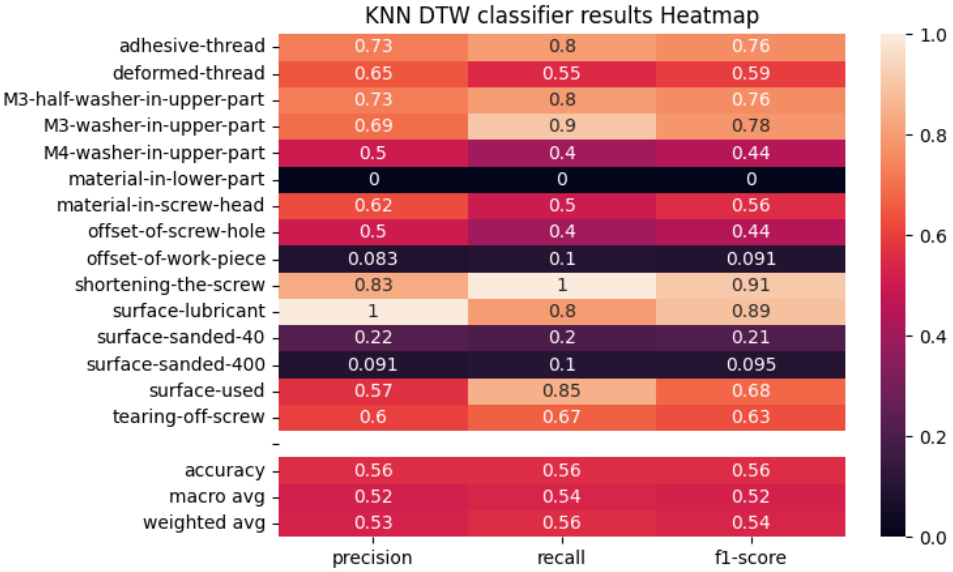


Figure 28: KNN DTW classification report.

The error categories *surface-lubricant*, *adhesive-thread*, *shortening-the-screw*, and *M3-half-washer-in-upper-part* have higher precision and recall scores. This indicates that the model was able to capture the patterns from these error categories more efficiently. Conversely, the error categories *offset-of-work-piece*, *surface-sanded-400*, and *surface-sanded-40* have very low precision

and recall scores. Additionally, the model was not able to learn anything from the error category *material-in-lower-part*, as both precision and recall scores are 0. In the case of *surface-used*, the precision score is low (0.57) but the recall score is high (0.85).

Overall, it can be stated that the distance-based method performs well in predicting a few error classes but performs poorly in classifying many other error categories, with the overall macro average F1 score being just above 50%.

6.2.2 Time Series Forest (TSF) results

Hyperparameter	values
n_estimators	[10, 30, 50, 100]
min_interval	[3, 5, 10]
inner_series_length	[10, 20, 30]
random_state	[42]

Table 33: TSF hyperparameters.

During the training phase of this interval-based method, the list of values used for hyperparameters in the grid search CV are mentioned in table 33. The best hyperparameter values obtained are as follows: *inner_series_length*: 30, *min_interval*: 3, and *n_estimators*: 50. The model was then trained again with this optimal set of hyperparameters, and the results are shown in figure 29.

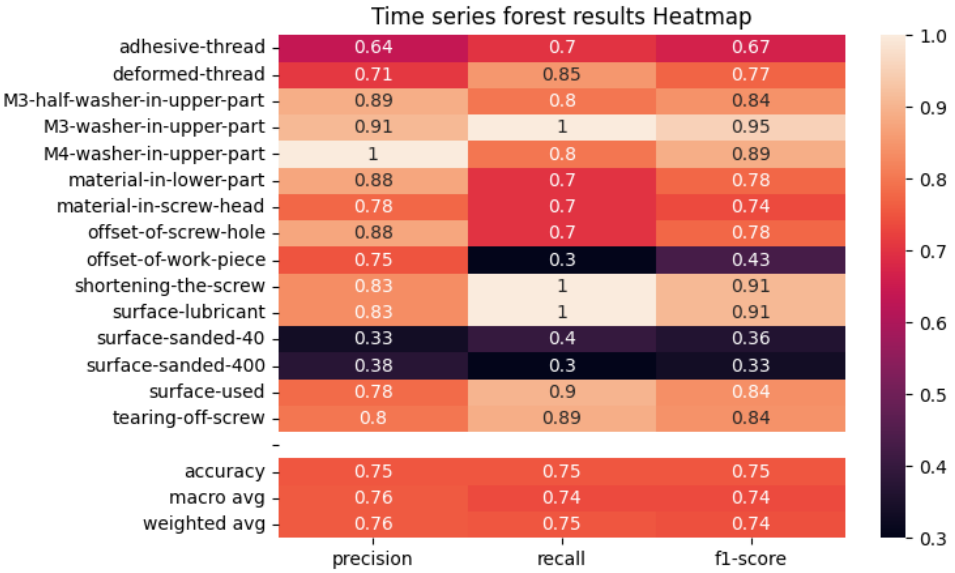


Figure 29: Time Series Forest classification report.

From the classification report in figure 29, it can be observed that the overall accuracy of the TSF classifier is 0.75, which is significantly better compared to the distance-based classifier. However, for the error categories *surface-sanded-40* and *surface-sanded-400*, the precision and recall values are very low compared to other categories. This indicates that similar to the distance-based

classifier, the interval-based method also struggles to correctly identify these two classes. In the case of *offset-of-work-piece*, the precision score is good at 0.75, but the recall value is very low at 0.3. A precision value of 1 is obtained only in one category (*M4-washer-in-upper-part*), whereas the recall score of 1 is obtained in three error categories (*M4-washer-in-upper-part*, *shortening-the-screw*, and *surface-lubricant*). In all other categories, the precision and recall scores are observed to be similar.

Overall, the TSF classifier results are better, with good precision and recall scores for most categories except for *surface-sanded-40* and *surface-sanded-400*.

6.2.3 ROCKET classifier results

During the training phase of the kernel-based method, only the list of values for the hyperparameter *rocket_transform* is used. It is experimented with three different values: *['rocket', 'minirocket', 'multirocket']*, and the *num_kernels* hyperparameter is set to 5000. After experimenting with different *rocket_transform* methods with a fixed value of *num_kernels*, the best value for *rocket_transform* is found to be *multirocket*. The classification report created using the test set is shown in figure 30. The overall accuracy of the ROCKET model is 0.81.

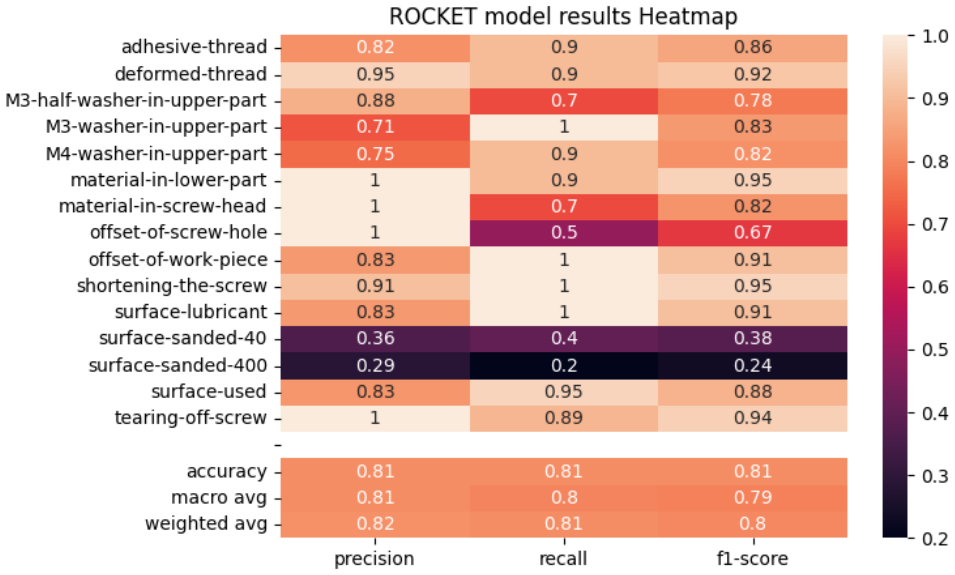


Figure 30: ROCKET classification report.

It can be seen that even the kernel-based methods struggle to perfectly identify the *surface-sanded-40* and *surface-sanded-400* error categories, with both precision and recall values less than 0.4. Looking into the recall values of each error category, it is observed that, apart from the mentioned surface error categories, the recall value is low only for *offset-of-screw-hole* (0.50). Moreover, in a few categories like *offset-of-work-piece*, *shortening-the-screw*, and *surface-lubricant*, the recall score is 1. Similarly, the precision scores are also high for most error categories. For the error categories *material-in-lower-part*, *material-in-screw-head*, and *offset-*

of-screw-hole, the precision is 1. Overall, the ROCKET classifier predictions are extremely good compared to other classifier results.

6.2.4 Catch22 results

During the training phase, the 22 features listed in Appendix tables 60 and 61 are extracted from the time series sequences. The catch22 classifier utilizes the Random Forest classifier internally to train the classification model. For the Random Forest classifier, the hyperparameters used are as follows: *n_estimators*: 100, *outlier_norm*: *True*, and *random_state*: 42. The trained model's performance is evaluated on 20% of the dataset (test set), and the classification report from the test set is visualized in figure 31. The overall accuracy of the model is observed to be 0.67.

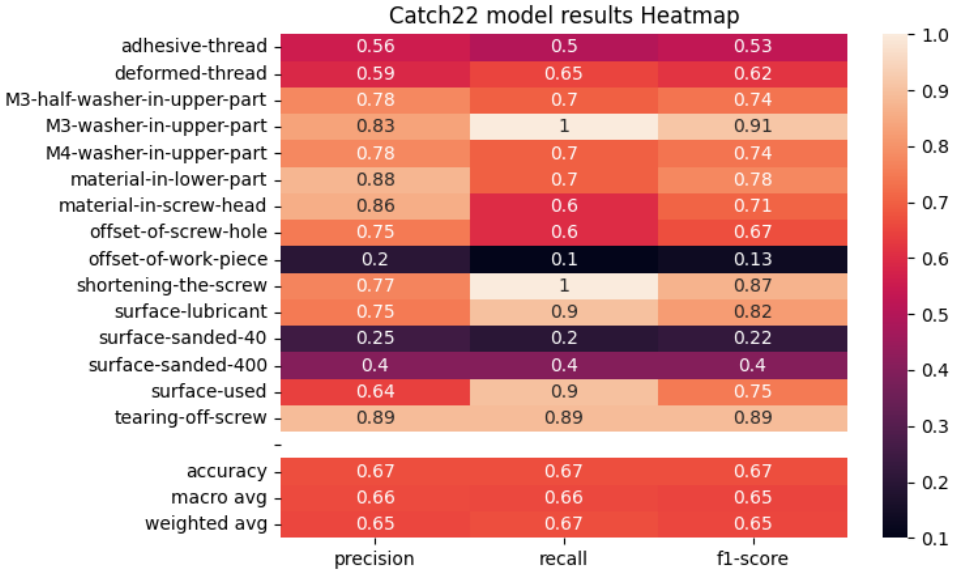


Figure 31: Catch22 classification report.

This feature-based classifier struggles to accurately classify the error classes *offset-of-work-piece* and *surface-sanded-40*, as both precision and recall values are very low. Similar challenges are noticed in *surface-sanded-400* and *adhesive-thread*, although the precision and recall scores are slightly better in comparison. Apart from these error categories, all other categories exhibit high precision and recall values. Notably, *M3-washer-in-upper-part* and *shortening-the-screw* achieve a perfect recall score of 1. Moreover, in the *tearing-off-screw* error category, both the precision and recall values are exactly 0.89.

Overall, considering the precision, recall, and accuracy scores of the feature-based classifier, we can conclude that its performance is above average and better than that of the distance-based classifier.

6.2.5 Shaplet transform classifier results

In the training phase of the shapelet transform classifier, the rotation forest algorithm is used to train the model. The list of values for each hyperparameter used in the grid search CV is mentioned in table 34. After the grid search, the best parameters are found to be: *batch_size*: 10, *estimator___n_estimators*: 20, *max_shapelet_length*: *None*, *max_shapelets*: 5, and *n_shapelet_samples*: 100. The shapelet transform model is then retrained with this optimal set of hyperparameters. The test dataset is used as input for the final trained model, and the results from the classification report are visualized in figure 32. The overall accuracy of the shapelet transform classifier is observed to be 0.70.

Hyperparameter	values
estimator___n_estimators	[5, 10, 20]
n_shapelet_samples	[50, 100, 150]
max_shapelets	[5, 10, 20]
max_shapelet_length	[None, 20, 50]
batch_size	[10, 20, 50]

Table 34: Shaplet transform hyperparameters.

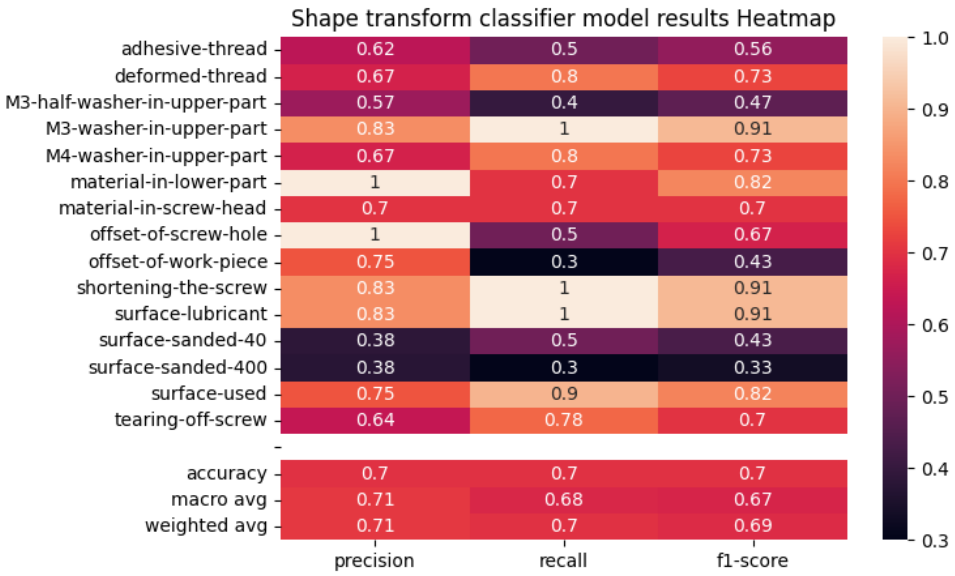


Figure 32: Shapelet transform classification report.

The precision for the *material-in-lower-part* and *offset-of-screw-hole* is 1, but the recall value for *offset-of-screw-hole* is very low (0.5) compared to precision. Similar to other classifiers, the precision and recall values are low for *surface-sanded-40* and *surface-sanded-400*. Additionally, for the error categories *M3-washer-in-upper-part*, *shortening-the-screw*, and *surface-lubricant*, the recall value is exactly 1. Lower recall values are also observed in the error categories *adhesive-thread* (0.5), *M3-half-washer-in-upper-part* (0.4), and *offset-of-work-piece* (0.3). For all other error categories, the precision and recall scores are either high or above average. Overall, it can be said that the results of the shapelet-based classifier are better than the distance-based and

feature-based classifiers, and it competes with the results of the Time Series Forest (TSF) and ROCKET classifiers, though with slightly lower precision and recall scores.

6.2.6 CNN results

The convolutional neural network is also trained using a large list of hyperparameters, and grid search CV is used to find the best parameters. The list of hyperparameter values is specified in table 35, and the best value for each hyperparameter is found to be as follows: *activation*: *tanh*, *batch_size*: 32, *epochs*: 50, *learning_rate*: 1.0, *loss*: *SparseCategoricalCrossentropy()*, *optimizer*: *Adadelta*. The overall accuracy of the CNN classifier is 0.56, which is the same as the KNN DTW classifier, indicating that this classifier performs poorly compared to other classifiers.

Hyperparameter	**values**
epochs	[50, 100]
batch_size	[16, 32]
activation	['relu', 'tanh']
loss	[SparseCategoricalCrossentropy(), CategoricalCrossentropy()]
learning_rate	[0.01, 0.1, 1.0]
optimizer	[Adadelta, Adam, SGD]

Table 35: CNN hyperparameters.

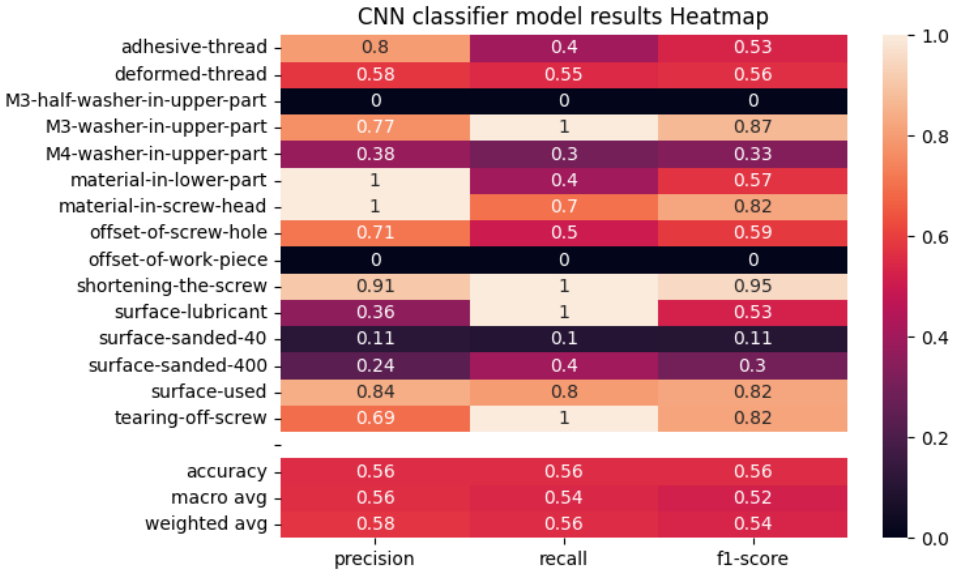


Figure 33: CNN classification report.

The main observation from the classification report in figure 33 is that for *M3-half-washer-in-upper-part* and *offset-of-work-piece*, the precision and recall values are 0. This indicates that the CNN model was not able to capture any patterns from these error classes and always misclassifies these categories. Moreover, the precision and recall values for *surface-sanded-40* are nearly 0. Apart from these poor values, the classifier also performs better in a few error categories. The

precision value is 1 in the error categories *material-in-lower-part* and *material-in-screw-head*. Additionally, the recall value is 1 for the error categories *M3-washer-in-upper-part*, *shortening-the-screw*, *surface-lubricant*, and *tearing-off-screw*.

Overall, it can be stated that even though the model has some good precision and recall values for a few error categories, the overall performance of the model is poor. The CNN model is not able to accurately capture the patterns of different error categories efficiently.

6.3 Supervised models combined results

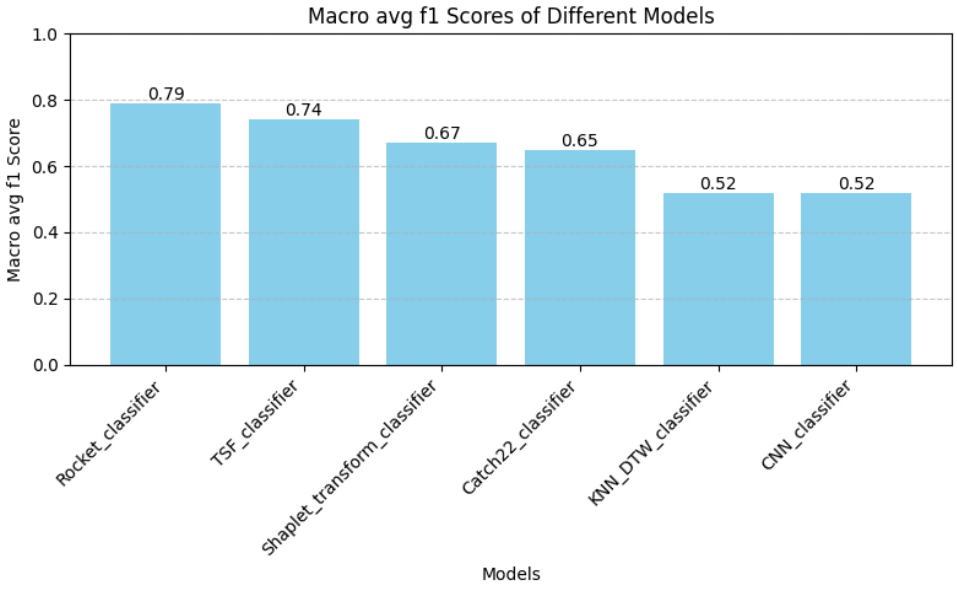


Figure 34: Bar plot for Macro avg F1 scores of different (supervised) models.

The bar plot in figure 34 shows the macro average F1 scores of each supervised classification model. It can be seen that out of the six classification models, the ROCKET classifier performs the best with a macro average F1 score of 0.79. It is followed by the Time Series Forest (TSF) classifier with a macro average F1 score of 0.74. The distance-based model (KNN DTW classifier) and the deep learning model (CNN classifier) perform poorly compared to other classifiers, each with a macro average F1 score of 0.52. Furthermore, the results of Shapelet Transform classifier and catch22 classifier are above average with macro-average F1 scores ranging from 0.65 to 0.67. Hence, it can be concluded that if different screw anomalies need to be determined using the supervised approach, the kernel-based approach (ROCKET classifier) can be used as it performs very well in classifying multiple errors.

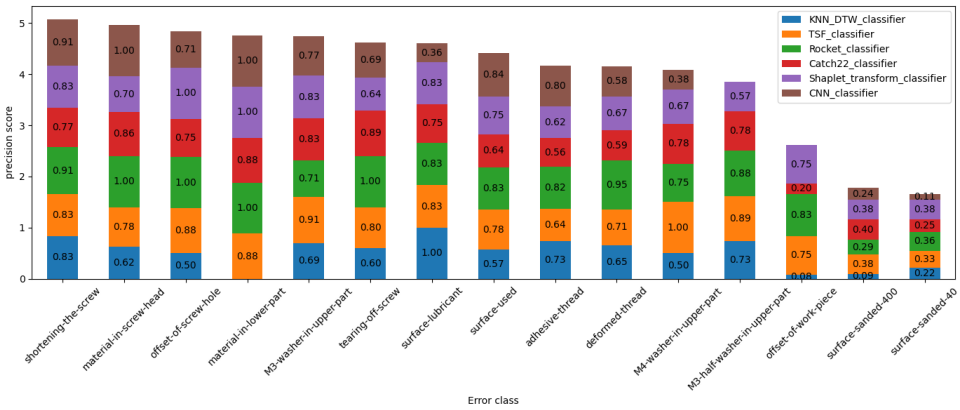


Figure 35: Stacked bar chart for precision scores of each error category (Supervised models).

Figure 35 shows the stacked bar chart combining the precision scores of all the classification models. This makes it easier to compare the results of each error category across multiple models. It can be observed that precision scores are lower for the error categories *offset-of-work-piece*, *surface-sanded-40*, and *surface-sanded-400*, as the length of the bars is comparatively shorter. For *surface-lubricant*, all classifiers have good precision scores except for the CNN classifier. In general, all the models have good precision scores for the error categories *shortening-the-screw*, *material-in-screw-head*, *offset-of-screw-hole*, *M3-washer-in-upper-part*, *material-in-lower-part*, *tearing-off-screw*, and *surface-used*.

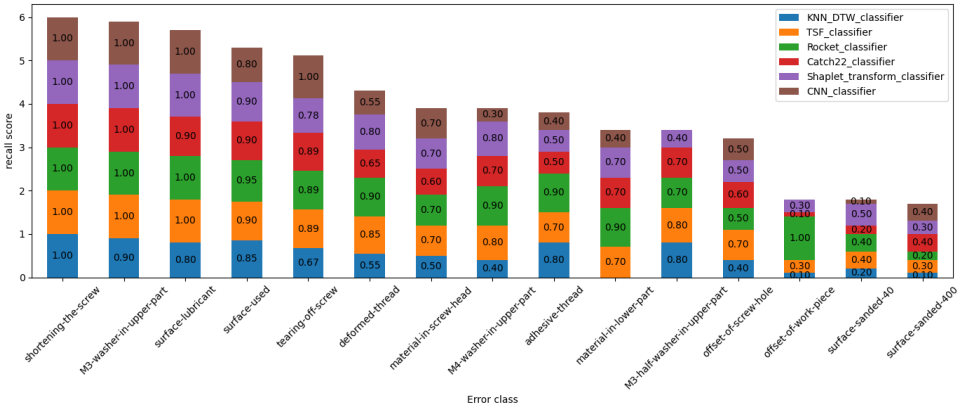


Figure 36: Stacked bar chart for recall scores of each error category (Supervised models).

Figure 36 shows a stacked bar chart of the recall scores for all classification models combined. In general, the bar lengths for each error category in the recall values chart are shorter compared to the precision values chart, indicating that recall scores are generally lower than precision scores. All other observations for the recall values chart are similar to those for the precision values chart.

6.4 Unsupervised methods

In this subsection, the results of all unsupervised models used for multi-class error clustering in this thesis are presented in tabular form and explained in detail. A variety of clustering algorithms from different families are experimented and their performance is assessed. From simple clustering methods, time series k-means (distance-based approach), and DBSCAN (density-based method) are used.

From advanced clustering algorithms, k-shape clustering (shape-based method), spectral clustering (graph-based method) is selected, and self-organizing maps and CNN autoencoder approaches are used from the deep learning family. Affinity propagation is used from the similarity-based approach. For the probabilistic-based approach, Hidden Markov Models (HMM) and Bayesian Gaussian mixture modeling are utilized. In the decomposition-based approach, matrix factorization was employed.

Due to the page limit of the master thesis report, it is not possible to explain all the clustering results. Therefore, results from k-means is detailed from the simple clustering method. From the advanced clustering methods, the results from the best models, i.e., K-shape, spectral clustering, affinity propagation, and self-organizing maps are explained in detail. Additionally, cluster ensemble methods like the co-association matrix method and the relabel and maximum voting method were used to determine if the results were better than any individual clustering method.

Since the clustering algorithms only output labels indicating which cluster each time series belongs to, additional logic is used to map these cluster labels to the true error categories, given that the true label for each screw-tightening process is known. The basic idea is that each error category will have the maximum number of observations in just one cluster, with very few observations assigned to other clusters. Therefore, each error category is mapped to the cluster with the highest number of observations from that category.

In some cases, certain clusters might remain unassigned. These clusters are identified and mapped to the error category that has the most observations in that cluster. Thus, each cluster can be assigned to a particular error category. To enhance stability, accuracy and find the best hyperparameters, 5-fold cross-validation is used. In each fold, the model was trained with 4 folds, and the last fold was used as a validation set. Adjusted Rand Index (ARI) is used as metric to evaluate and compare the cluster results from each fold. Finally, the clusters with the best ARI score are selected as the final clusters.

Each clustering algorithm was applied to errors in the green, orange, and red error categories separately, including the OK samples (without *baseline-extra*) as an additional category. Finally, the clustering algorithm was applied to the whole dataset (including all error categories and OK samples without *baseline-extra*). For better understanding, this clustering result is referred to as the ‘black color category’.

6.4.1 K-means clustering results

K-means clustering is applied to four different color categories (green, orange, red, and black) separately, and the results are tabulated below. For the green, orange, and red color categories, the number of clusters is fixed at 8. This number is chosen to represent 5 error categories and 3 sub-categories in the baseline. For the black color category, the number of clusters is set to 18, representing the entire screw-tightening dataset, which includes 15 different error categories and 3 sub-categories of the baseline. This cluster numbers are consistent across other clustering algorithms as well. In the results, the category labeled *baseline* represent all the sub-categories within the OK category (*baseline*, *baseline-abrasion*, and *baseline-friction*). As they are OK categories, only one true label is assigned to all these categories in the dataset.

The *TimeSeriesKMeans* function from the *sktime* library is used to train the k-means clustering model, which offers various hyperparameters. The *init_algorithm* hyperparameter, which defines the method for initializing the clusters, is tested with both *kmeans++* and *random*. Cross-validation reveals that *kmeans++* provides the best initialization. The other hyperparameters are set as follows: *n_init* to 1, *max_iter* to 10, and metric to *dtw*. These parameters are kept consistent across all color categories to ensure a fair comparison of the results.

Green category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	**400**	0	0	0	0	0	0	0
M3-washer-in-upper-part	0	0	0	3	0	**47**	0	0
Offset-of-screw-hole	**23**	0	4	2	0	19	0	2
Shortening-the-screw	**45**	0	0	0	0	0	0	2
Tearing-off-screw	4	5	0	0	10	8	16	2
Surface-lubricant	**50**	0	0	0	0	0	0	0

Table 36: Error category to cluster mappings for green category (K-means clustering)

Table 36 presents the results for the green color category. It can be observed that most error categories are assigned to cluster 0, indicating that the algorithm is not performing well in separating the error categories. The DTW distances among these time series are small, leading the algorithm to consider all the time series similar and assign them to a single cluster. However, the k-means algorithm performs well in identifying one particular error category (*M3-washer-in-upper-part*) as these observations are assigned to a different cluster (cluster 5). Additionally, the observations from the error category *tearing-off-screw* are split across several clusters, with clusters 4 and 6 containing most observations from this category. The ARI score for this clustering result is 0.478. The highlighted numbers in the table denote the mapping of clusters to error categories based on the logic explained earlier. Since most error categories are mapped to a single cluster, the accuracy metric for the mapping is very high even though the results are not very good.

Table 37 shows the k-means clustering results for the orange color category, and similar observations to the green category can be noticed. The predictions for the error category *material-in-screw-head* are spread across multiple clusters and all other error categories are assigned to cluster 0. This indicates that k-means does not effectively separate the error categories in the orange category. The ARI for this clustering result is 0.135 which is very low.

Orange category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	**400**	0	0	0	0	0	0	0
Adhesive-thread	**49**	1	0	0	0	0	0	0
M4-washer-in-upper-part	**45**	0	3	0	0	0	0	2
M3-half-washer-in-upper-part	**46**	0	1	0	0	0	0	3
Material-in-screw-head	**23**	0	5	0	12	10	0	0
Material-in-lower-part	**50**	0	0	0	0	0	0	0

Table 37: Error category to cluster mappings for orange category (K-means clustering)

Table 38 shows the results for the red color category. It can be noticed that the results are significantly worse compared to the green and orange categories, as all observations from all error categories are assigned to cluster 5. A single time series from the error categories *surface-sanded-400*, *surface-sanded-40*, and *surface-used* is assigned to cluster 1, indicating these three time series are very similar and distinct from all other time series. The Adjusted Rand Index for this clustering result is 0.006 which is almost 0 and proves the cluster results are worse.

Red category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	0	0	0	0	0	**400**	0	0
Offset-of-work-piece	0	0	0	0	0	**50**	0	0
Surface-sanded-400	0	1	0	0	0	**49**	0	0
Deformed-thread	0	0	0	0	0	**100**	0	0
Surface-used	0	1	0	0	0	**99**	0	0
Surface-sanded-40	0	1	0	0	0	**49**	0	0

Table 38: Error category to cluster mappings for Red category (K-means clustering)

Finally, the table 39 shows the clustering results obtained from modeling the entire dataset. The highlighted numbers represent the good performance of the k-means clustering algorithm. The model captures the distinct pattern in the *tearing-off-screw* category, as most observations are assigned to clusters 6 and 9, which contain very few observations from other categories. A similar observation can be made for the error category *M3-washer-in-upper-part*, where all observations are assigned to clusters 1 and 10. A proportion of observations from the *offset-of-screw-hole* category is also assigned to cluster 10, indicating these time series are similar to *M3-washer-in-upper-part*. Additionally, a proportion of observations from *material-in-screw-head* is assigned to cluster 7. All other observations from multiple error categories are assigned to cluster 0. It

can be noted that clusters 5, 8 and 15 are not included in the table as none of the observations are assigned to these clusters.

Overall, the performance of the k-means clustering is poor across all color categories, except for capturing a few distinct time series from some error categories. The ARI score for clustering using the entire dataset is 0.089.

Black category results

Cluster Label / True Label	0	1	2	3	4	6	7	9	10	11	12	13	14	16	17
Baseline	400	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Adhesive-thread	49	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Deformed-thread	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M3-half-washer	46	0	0	0	0	0	0	0	0	0	0	1	0	0	3
M3-washer	0	**7**	0	0	0	0	0	0	**43**	0	0	0	0	0	0
M4-washer	38	0	2	0	0	0	0	0	9	0	0	0	0	0	1
Material-in-lower-part	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Material-in-screw-head	23	0	0	0	2	0	**15**	7	3	0	0	0	0	0	0
Offset-of-screw-hole	18	3	0	4	0	0	0	0	**21**	1	1	0	0	2	0
Offset-of-work-piece	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Shortening-the-screw	44	0	0	0	1	0	0	0	1	0	0	0	0	0	1
Surface-lubricant	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Surface-sanded-40	49	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Surface-sanded-400	49	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Surface-used	99	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Tearing-off-screw	3	0	1	0	0	**8**	0	**25**	5	0	1	1	1	0	0

Table 39: Error category to cluster mappings for Black category (K-means clustering)

6.4.2 K-shape clustering results

The *KShape* function from the *tslearn* library is used to implement the shape-based clustering algorithm K-shape. The library offers several hyperparameters that can be set before training. The *n_clusters* parameter is set to 8 or 18, depending on the color category being clustered. The *n_init* hyperparameter, which denotes the number of times the K-shape algorithm runs with different centroid initializations, is experimented with [5, 10, 15] using cross-validation, and it is found that the best number of initializations is 5. The *init* parameter, which denotes the method for initialization, is set to *random*. The *max_iter* parameter, which denotes the maximum number of iterations, is set to 100.

Table 40 shows the results for the green category. The observations from different error categories are well separated into different clusters, with some misclassifications. Most of the observations from the *M3-washer-in-upper-part* category are assigned to cluster 6. The observations from the *surface-lubricant* category are perfectly assigned to cluster 1, which does not contain observations from any other error categories. This shows that the *surface-lubricant* error has a unique shape

of time series, making it easily separable. In the case of *shortening-the-screw*, 38 observations are assigned to cluster 7, which is also very good. The observations from the *offset-of-screw-hole* category are split into two clusters (cluster 5 and cluster 6). It can be seen that half of the observations from this category are in the same cluster as *M3-washer-in-upper-part*, indicating that some time series from *offset-of-screw-hole* have the same shape as *M3-washer-in-upper-part*.

Green category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	**209**	0	0	**161**	30	0	0	0
M3-washer-in-upper-part	0	0	0	1	9	0	**40**	0
Offset-of-screw-hole	0	0	0	0	5	**20**	25	0
Shortening-the-screw	0	0	7	0	2	0	0	**38**
Tearing-off-screw	1	0	**23**	2	16	0	1	2
Surface-lubricant	0	**50**	0	0	0	0	0	0

Table 40: Error category to cluster mappings for green category (K-shape clustering)

Additionally, almost half of the observations from the *tearing-off-screw* belong to cluster 2. Cluster 4 contains observations from all error categories except *surface-lubricant*, indicating that some time series have similar shapes across all error categories. The baseline observations are also well separated from multiple error categories, mostly being assigned to clusters 0 and 3. The Adjusted Rand Index (ARI) for the K-shape clustering in the green category is 0.46. The overall accuracy of the cluster-to-error category mapping is 93%, indicating that the K-shape clustering performance is excellent in separating the errors in the green category.

Orange category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	30	0	**171**	30	**169**	0	0	0
Adhesive-thread	0	**44**	0	0	1	1	0	4
M4-washer-in-upper-part	2	4	0	4	1	10	0	**29**
M3-half-washer-in-upper-part	4	12	0	2	1	5	0	**26**
Material-in-screw-head	11	10	7	2	11	0	5	4
Material-in-lower-part	14	1	0	0	0	**35**	0	0

Table 41: Error category to cluster mappings for orange category (K-shape clustering)

Table 41 represents the results from the orange color category. It can be observed that the error category *adhesive-thread* is well separated into cluster 1, but also contains some misclassifications from other error categories. Similarly, *material-in-lower-part* observations are mostly assigned to cluster 5. Half of the observations from *M4-washer-in-upper-part* and *M3-half-washer-in-upper-part* are assigned to cluster 7, indicating that these error categories have the same shapes. *Material-in-screw-head* observations are spread across all the clusters, indicating the model was not able to properly capture the shape pattern from this error category. The *baseline* observations are mostly limited to clusters 2 and 4 and are well separated from errors. The ARI

score for this clustering is 0.337. The overall accuracy of the cluster-to-error category mapping is 80%, which is also very good and far better than the k-means results.

Table 42 shows the results from the red category. At a glance, it can be noticed that all the clusters have observations from multiple error categories, as the numbers are spread across the table. The main observations are as follows: In the case of *deformed-thread*, 50% of observations are separated into cluster 6 with very few observations from other errors. Similarly, 50% of observations from *surface-sanded-40* and *surface-sanded-400* are assigned to cluster 2, and these observations overlap with each other and with a few other error categories. The *offset-of-work-piece* and *surface-used* errors are spread across all the clusters. Moreover, the *baseline* observations are also spread across all the clusters, but a majority of them belong to clusters 1, 5, and 7. The ARI score for this clustering is 0.124, which is very low compared to the green and orange categories. This shows that multiple error categories in red have the same shape, making it very hard to separate them into individual clusters. The overall accuracy of the cluster-to-error category mapping is 68%.

Red category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	13	**106**	30	30	29	**77**	1	**114**
Offset-of-work-piece	20	0	14	0	4	5	3	4
Surface-sanded-400	13	0	**24**	2	4	1	1	5
Deformed-thread	3	1	9	2	5	16	**49**	15
Surface-used	37	19	3	11	9	8	0	13
Surface-sanded-40	11	0	**23**	1	7	3	3	2

Table 42: Error category to cluster mappings for Red category (K-shape clustering)

Black category results

Table 43 shows the results from the black category obtained from K-shape clustering. The highlighted numbers indicate the unique patterns captured by K-shape, which are well separated into different clusters compared to others. The numbers from the *tearing-off-screw*, *shortening-the-screw*, and *surface-lubricant* categories show that these are well separated, similar to the results from the green color category. This is valid when the model is tested with the whole dataset as well. Additionally, the highlighted numbers from *M3-half-washer-in-upper-part* (20) and *M4-washer-in-upper-part* (24) show that they belong to the same cluster (cluster 2) and overlap, as observed in the orange category results. More than 60% of observations from *adhesive-thread* belong to cluster 9, overlapping with the observations from *deformed-thread*. Also, more than half of the observations from *material-in-lower-part* are assigned to cluster 1 and appear well separated. All the other error categories have their observations spread across the clusters. In the case of the *baseline* category, most observations are assigned to clusters 0, 4, and 11.

The overall accuracy of the cluster-to-error category mapping is 55%, with a 0.25 ARI score for the clustering result. It can be stated that K-shape clustering performs very well in the green

Cluster Label / True Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Baseline	**72**	0	0	20	**72**	0	25	2	0	0	3	**157**	9	5	0	0	0	35
Adhesive-thread	2	0	4	0	0	0	0	0	10	**33**	0	0	0	0	0	0	1	0
Deformed-thread	31	0	5	5	12	0	2	0	2	**31**	0	5	1	0	0	0	0	6
M3-half-washer	0	2	**20**	1	0	0	2	4	3	2	1	0	0	0	9	1	4	1
M3-washer	0	0	1	0	0	0	0	0	13	5	0	0	0	0	0	31	0	0
M4-washer	0	0	**24**	0	1	0	4	1	1	0	1	0	0	0	5	8	5	0
Material-in-lower-part	0	**28**	0	3	0	0	0	14	2	0	0	0	1	0	0	2	0	0
Material-in-screw-head	1	1	2	4	8	5	2	3	0	2	1	1	0	1	0	0	14	5
Offset-of-screw-hole	0	21	0	0	0	0	1	0	3	0	0	0	0	0	0	17	8	0
Offset-of-work-piece	5	0	2	16	4	0	0	11	0	0	4	2	0	0	0	0	0	6
Shortening-the-screw	0	0	0	0	0	**45**	0	0	0	0	0	0	0	0	0	0	2	0
Surface-lubricant	0	0	0	0	0	0	0	0	0	0	8	0	0	0	**42**	0	0	0
Surface-sanded-40	1	1	0	8	5	0	0	7	0	0	16	0	0	0	4	0	1	7
Surface-sanded-400	3	0	1	6	4	0	1	8	0	1	11	0	0	0	1	0	1	13
Surface-used	9	0	0	19	5	1	4	11	0	0	0	22	17	6	0	0	1	5
Tearing-off-screw	0	0	5	0	0	10	1	0	0	2	0	0	0	0	0	0	**27**	0

Table 43: Error category to cluster mappings for Black category (K-shape clustering)

and orange categories but performs poorly in the red category. The accuracy on the overall dataset is just above 50%, which is similar to some classification models in supervised methods.

6.4.3 Spectral clustering results

The steps involved in spectral clustering, such as constructing the similarity matrix, Laplacian transformation, and computing eigenvalues and eigenvectors, are performed using the appropriate libraries. The obtained eigenvectors are then used as input for the K-means clustering algorithm from *sklearn* library. A CV approach is used to determine the best values for the hyperparameters *init* and *algorithm* in the K-means clustering. The *init* parameter, which denotes the method for initializing the centroids, is experimented with *['k-means++', 'random']*. The algorithm parameter, which denotes the type of algorithm to use for K-means clustering, is experimented with *['lloyd', 'elkan']*. After hyperparameter tuning, the best parameters found are as follows: *init*: *k-means++*, *algorithm*: *lloyd*, *max_iter*: 300, *n_init*: *auto*.

Table 44 shows the results from the green color category. It can be observed that the error categories *surface-lubricant shortening-the-screw*, and *tearing-off-screw*, are assigned to clusters 4, 5, and 6 respectively, and are well separated. The *M3-washer-in-upper-part* observations are assigned to cluster 0, and half of the observations from *offset-of-screw-hole* overlap with this error category. Additionally, only 7 observations from *offset-of-screw-hole* are assigned to cluster 7, and this cluster doesn't contain any other observations. The *baseline* observations are well separated and assigned to clusters 1, 2, and 3. The overall accuracy of the cluster-to-error category mapping is 93%, which is similar to K-shape results. However, in the case of *tearing-*

off-screw, spectral clustering has an edge over K-shape. The ARI score for the clustering is 0.45.

Green category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	1	**213**	**35**	**150**	1	0	0	0
M3-washer-in-upper-part	**47**	2	0	0	0	1	0	0
Offset-of-screw-hole	**27**	1	0	0	13	2	0	**7**
Shortening-the-screw	0	0	0	0	0	**38**	9	0
Tearing-off-screw	1	0	1	0	0	3	**40**	0
Surface-lubricant	0	2	8	0	**40**	0	0	0

Table 44: Error category to cluster mappings for green category (Spectral clustering)

Table 45 shows the results from the orange color category. It can be seen that almost all observations from the *baseline* category are assigned to clusters 0, 1, and 3. Similar to the K-shape results, the observations from *M4-washer-in-upper-part* and *M3-half-washer-in-upper-part* overlap, which is evident from the numbers in cluster 5. Spectral clustering performs well in identifying the error category *material-in-screw-head*, as more than 50% of observations are assigned to cluster 6. Additionally, most observations from *material-in-lower-part* and *adhesive-thread* are assigned to clusters 2 and 7, respectively, and are well separated. The overall accuracy for the cluster-to-error category mapping is 86%, and the ARI score is 0.252.

Orange category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	**148**	**108**	0	**108**	33	2	0	1
Adhesive-thread	1	1	0	0	6	1	1	**40**
M4-washer-in-upper-part	1	4	10	1	4	**27**	3	0
M3-half-washer-in-upper-part	1	0	3	4	2	**32**	4	4
Material-in-screw-head	11	3	0	6	2	0	**27**	1
Material-in-lower-part	6	0	**33**	8	2	0	0	1

Table 45: Error category to cluster mappings for orange category (Spectral clustering)

Red category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	15	**87**	**94**	24	2	25	33	**120**
Offset-of-work-piece	15	1	8	13	3	0	5	5
Surface-sanded-400	11	0	7	16	2	2	6	6
Deformed-thread	2	7	16	8	**44**	1	6	16
Surface-used	23	19	12	3	0	10	14	19
Surface-sanded-40	10	0	15	18	1	1	3	2

Table 46: Error category to cluster mappings for Red category (Spectral clustering)

Table 46 shows the results from the red color category. The overall accuracy for the cluster-to-error category mapping is 63%, and the ARI score is 0.1, which is very low. It is evident from table 46 that all observations for all categories are distributed across clusters. The observations from the *baseline* category are assigned to clusters 1, 2, and 7, and all these clusters contain some observations from other error categories as well. Additionally, the model assigns 44% of observations from *deformed-thread* to cluster 4.

Table 47 shows the results from the black color category of spectral clustering. The overall accuracy of the cluster-to-error category mapping is 62%. The important values are highlighted in the table. Spectral clustering perfectly assigns the *M3-washer-in-upper-part* to a single cluster (cluster 10), which also contains 28 observations from *offset-of-screw-hole*. Cluster 12 contains 32 observations from *material-in-lower-part* and very few observations from other error categories. Furthermore, in the *baseline* category, the majority of observations are assigned to clusters 1, 3, 14, and 15. The ARI score for the spectral clustering performed on the entire dataset is 0.165. The clustering results on the entire dataset are similar to K-shape clustering but with a higher accuracy score and a lower ARI score.

Black category results

Cluster Label / True Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Baseline	2	**68**	3	**92**	0	25	0	0	33	0	0	17	0	0	**75**	**58**	0	27
Adhesive-thread	0	0	2	1	1	0	0	1	2	0	0	2	0	**40**	1	0	0	0
Deformed-thread	0	7	2	6	0	1	0	1	7	0	0	6	0	21	17	17	0	15
M3-half-washer	0	0	19	0	3	0	2	17	2	0	1	1	2	0	0	1	1	1
M3-washer	0	0	0	0	0	0	0	0	0	0	**50**	0	0	0	0	0	0	0
M4-washer	2	0	15	0	3	0	1	8	6	0	11	2	0	0	1	1	0	0
Material-in-lower-part	0	1	0	0	0	0	0	2	2	0	0	9	**32**	1	1	0	0	2
Material-in-screw-head	0	1	1	1	22	1	0	0	2	0	0	3	1	0	2	7	4	5
Offset-of-screw-hole	0	0	0	0	2	0	0	1	0	6	**28**	0	13	0	0	0	0	0
Offset-of-work-piece	0	2	0	1	0	0	0	1	5	0	0	19	1	1	5	6	0	9
Shortening-the-screw	19	3	0	0	2	0	16	0	0	0	0	0	0	0	0	0	7	0
Surface-lubricant	0	0	16	0	0	0	0	22	8	0	0	4	0	0	0	0	0	0
Surface-sanded-40	0	0	6	0	1	0	0	8	3	0	0	16	1	0	5	5	0	5
Surface-sanded-400	0	0	5	0	1	1	0	4	6	0	0	17	0	0	2	7	0	7
Surface-used	1	14	0	22	1	9	0	0	14	0	0	5	0	0	11	9	0	14
Tearing-off-screw	1	0	0	0	19	0	1	0	0	0	2	0	0	0	0	0	22	0

Table 47: Error category to cluster mappings for Black category (Spectral clustering)

6.4.4 Self-Organizing Maps results

During the cross-validation of the SOM model, the list of hyperparameter values used for grid search CV is mentioned in Table 48. Here, x represents the SOM kernel map width and y represents the SOM kernel map height. The best hyperparameters found are as follows: x: 15, y: 15, *sigma*: 0.5, *learning_rate*: 0.5, *neighborhood_function*: *triangle*, *activation_distance*:

gaussian. These best hyperparameters are used to train SOM models for green, orange, red, and black color categories, and the results are provided in the tables below.

Hyperparameter	**values**
x	[10, 15, 20]
y	[10, 15, 20]
sigma	[0.5, 1.0]
neighborhood_function	['gaussian', 'triangle']
activation_distance	['euclidean', 'cosine']

Table 48: SOM hyperparameters.

Table 49 shows the results from the green color category. The numbers from the error categories *M3-washer-in-upper-part*, *offset-of-screw-hole*, and *shortening-the-screw* are similar to spectral and K-shape clustering results. The SOM model doesn't perform well in uniquely identifying the error category *surface-lubricant*, as all the observations from this error category overlap with the *baseline* category, which can be seen in clusters 1, 6, and 7. The error category *tearing-off-screw* is well identified, as most of the observations are assigned to cluster 5. The *baseline* observations are assigned to clusters 1, 2, 6 and 7. The overall accuracy of the cluster-to-error category mapping is 78%, and the ARI score is 0.27, which is low compared to the results from spectral and K-shape clustering methods in the green category.

Green category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	0	**33**	**123**	0	0	0	**151**	**93**
M3-washer-in-upper-part	5	0	0	**45**	0	0	0	0
Offset-of-screw-hole	2	0	0	**26**	20	2	0	0
Shortening-the-screw	**35**	0	0	0	0	9	3	0
Tearing-off-screw	2	0	0	1	0	**41**	0	1
Surface-lubricant	0	8	0	0	0	0	17	**25**

Table 49: Error category to cluster mappings for green category (SOM)

Orange category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	**149**	0	**101**	0	0	33	**115**	2
Adhesive-thread	13	0	0	1	0	6	**23**	7
M4-washer-in-upper-part	**14**	0	4	3	**11**	8	10	0
M3-half-washer-in-upper-part	**19**	0	2	4	1	2	17	5
Material-in-screw-head	11	**2**	2	**25**	0	3	7	0
Material-in-lower-part	3	0	0	0	0	1	4	**42**

Table 50: Error category to cluster mappings for orange category (SOM)

Table 50 shows the results from the orange color category. The SOM model performs well in identifying the error category *material-in-lower-part*, as most of the observations belong to

cluster 7. The results from the error categories *M4-washer-in-upper-part* and *M3-half-washer-in-upper-part* are similar to spectral and K-shape clustering results. Clusters 0, 2, and 6 contain most of the observations from the *baseline* category. Moreover, SOM assigns 50% of the observations from *material-in-screw-head* to cluster 3 and almost 50% of the observations from *adhesive-thread* to cluster 6. The overall accuracy of the cluster-to-error category mapping is 64%, and the ARI score is 0.135, which is also low compared to spectral and K-shape clustering.

Red category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	8	**82**	**103**	61	33	**86**	0	27
Offset-of-work-piece	13	1	3	8	5	20	0	0
Surface-sanded-400	12	4	0	11	5	16	1	1
Deformed-thread	18	6	5	22	7	**41**	0	1
Surface-used	9	11	16	9	14	**30**	1	10
Surface-sanded-40	10	1	1	10	1	**26**	1	0

Table 51: Error category to cluster mappings for Red category (SOM)

Black category results

Cluster Label / True Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Baseline	**77**	0	20	0	**63**	**74**	0	0	0	0	32	0	10	**95**	0	27	2	0
Adhesive-thread	0	0	21	1	1	0	0	0	0	0	6	0	7	2	0	0	12	0
Deformed-thread	6	0	26	0	19	2	0	0	0	0	7	0	11	18	0	1	10	0
M3-half-washer	0	0	7	4	1	0	1	0	0	0	2	0	6	13	0	0	12	4
M3-washer	0	24	0	0	0	0	0	0	0	0	0	20	0	0	6	0	0	0
M4-washer	0	8	3	3	0	0	2	0	0	0	6	1	1	11	1	0	14	0
Material-in-lower-part	0	0	3	0	0	0	17	0	0	0	1	0	7	0	0	0	0	22
Material-in-screw-head	2	0	4	19	7	1	0	8	0	0	3	0	2	3	0	1	0	0
Offset-of-screw-hole	0	22	0	2	0	0	8	0	0	7	0	2	0	0	2	0	0	7
Offset-of-work-piece	0	0	14	0	11	1	0	0	0	0	3	0	13	5	0	0	0	3
Shortening-the-screw	0	0	0	2	0	0	0	7	**35**	0	0	0	0	0	3	0	0	0
Surface-lubricant	0	0	25	0	0	0	0	0	0	0	8	0	10	0	0	0	7	0
Surface-sanded-40	0	0	15	1	9	0	1	0	0	0	2	0	6	11	0	0	4	1
surface-sanded-400	1	0	9	1	12	0	0	0	0	0	6	0	9	7	0	1	4	0
Surface-used	11	0	22	1	7	15	0	0	0	0	10	0	15	9	0	9	0	1
Tearing-off-screw	0	1	1	**29**	0	0	0	12	2	0	0	0	0	0	0	0	0	0

Table 52: Error category to cluster mappings for Black category (SOM)

Table 51 shows the results from the red color category. The results are spread across all clusters, with cluster 5 containing most of the observations from multiple error categories as well as the *baseline* category. Specifically, 30 observations from *surface-used*, 26 observations from *surface-sanded-40*, and 41 observations from *deformed-thread* are assigned to cluster 5. Observations from the error categories, *offset-of-work-piece* and *surface-sanded-400* are spread across

all clusters, but cluster 5 contains a few more observations compared to other clusters. The overall accuracy of the cluster-to-error category mapping is 61%, and the ARI score is 0.03.

Table 52 shows the results from the black color category. Interesting numbers are highlighted in the table. The *baseline* observations are mostly assigned to clusters 0, 4, 5, and 13. Additionally, 35 observations from the *shortening-the-screw* error category are assigned to cluster 8, and this cluster doesn't contain observations from any other error category except for 2 observations from *tearing-off-screw*. Most of the observations from *tearing-off-screw* (29) are assigned to cluster 3, which also contains observations from multiple error categories. All other error categories are assigned to a greater number of clusters, with no particularly interesting patterns. The overall accuracy of the cluster-to-error category mapping is 56%, and the ARI score is 0.133.

6.4.5 Affinity propagation results

The *AffinityPropagation* function from the *sklearn* library is used to implement the affinity propagation clustering algorithm. Unlike other clustering methods, affinity propagation automatically determines the number of clusters, so fixed cluster numbers are not applicable. For the green category, the algorithm produces 8 clusters, matching the number of clusters assigned in other clustering algorithms. For the orange and red categories, it produces 9 clusters. In the black category, the algorithm assigns data points to 22 clusters, which is 4 more than the clusters assigned by other methods.

The hyperparameters used to train the affinity propagation algorithm are as follows: *damping* is set to 0.5, *max_iter* to 100, *convergence_iter* to 15, and *affinity* is set to *precomputed*. The pairwise kernels method is used to compute the similarity matrix, similar to the approach used in spectral clustering. These hyperparameters are consistent across all color categories, and the results are tabulated below.

Green category results

Cluster Label / True Label	0	1	2	3	4	5	6	7
Baseline	**189**	**151**	27	33	0	0	0	0
M3-washer-in-upper-part	0	0	2	0	0	**43**	2	3
Offset-of-screw-hole	0	0	0	0	**20**	**26**	0	4
Shortening-the-screw	0	0	0	0	0	0	**38**	9
Tearing-off-screw	0	0	1	0	0	2	2	**40**
Surface-lubricant	**25**	17	0	8	0	0	0	0

Table 53: Error category to cluster mappings for green category (Affinity propagation)

Table 53 shows the results from the green category. The results are somewhat similar to those of the SOM algorithm for this category. For instance, half of the observations from the *surface-lubricant* category overlap with the *baseline* category and are assigned to cluster 0. Observations from the *offset-of-screw-hole* category are assigned to clusters 4 and 5, with cluster 5 also containing most observations from the *M3-washer-in-upper-part* category. Additionally, the

shortening-the-screw and *tearing-off-screw* categories are assigned to clusters 6 and 7, respectively. The ARI score for this clustering is 0.33, and the overall accuracy of cluster-to-error category mapping is 92%. Affinity propagation performs well in separating errors in the green category, similar to K-shape and spectral clustering methods.

Orange category results

Cluster Label / True Label	0	1	2	3	4	5	6	7	8
Baseline	**144**	24	33	48	**137**	14	0	0	0
Adhesive-thread	0	0	4	**23**	2	17	1	0	3
M4-washer-in-upper-part	3	5	6	8	2	13	3	10	0
M3-half-washer-in-upper-part	2	2	1	18	1	19	4	1	2
Material-in-screw-head	3	1	2	5	10	1	**27**	1	0
Material-in-lower-part	0	0	1	11	0	6	0	0	**32**

Table 54: Error category to cluster mappings for orange category (Affinity propagation)

Table 54 shows the results from the orange category. Observations from the *M4-washer-in-upper-part* and *M3-half-washer-in-upper-part* categories are spread across all clusters. The *baseline* category is mostly assigned to clusters 0 and 4, with a few observations in clusters 1, 2, and 3. Nearly half of the observations from the *adhesive-thread* category are mapped to cluster 3, and more than half of the observations from the *material-in-screw-head* category are mapped to cluster 6. The overall accuracy of cluster-to-error category mapping is 0.71, and the ARI score is 0.204. This ARI score is lower than those for spectral and K-shape clustering but higher than for SOM clustering.

Table 55 shows the results from the red category. All error categories, as well as the *baseline* category, have some observations assigned to all 9 clusters. For the *baseline* category, only cluster 8 doesn't have any observations, with most observations mapped to clusters 0 and 4. In the *offset-of-work-piece* error category, all clusters except 5 and 8 have some observations mapped to them. The ARI score for this clustering result is 0.076, and the overall accuracy of the cluster-to-error category mapping is 0.49. This accuracy is very low compared to other clustering results in the red category.

Red category results

Cluster Label / True Label	0	1	2	3	4	5	6	7	8
Baseline	**100**	44	11	79	**106**	26	32	2	0
Offset-of-work-piece	2	10	15	5	4	0	5	9	0
Surface-sanded-400	0	11	9	7	5	1	6	10	1
Deformed-thread	7	25	23	16	11	1	7	10	0
Surface-used	16	10	11	20	14	9	13	6	1
Surface-sanded-40	1	9	18	8	3	0	2	8	1

Table 55: Error category to cluster mappings for red category (Affinity propagation)

Tables 56 and 57 show the results from the black category. Due to the high number of clusters (21), the results are split into two tables. Table 56 contains results from clusters 0 to 9, while table 57 contains results from clusters 10 to 21. From table 56, we notice that most *baseline* observations are limited to clusters 0 to 7. Additionally, 36 observations from the *M3-washer-in-upper-part* category are assigned to cluster 8, and half of the observations from the *surface-lubricant* category are assigned to cluster 3. All other clusters contain a small number of observations from each error category.

Black category results

Cluster Label / True Label	0	1	2	3	4	5	6	7	8	9
Baseline	**97**	**57**	**86**	24	26	33	**69**	3	0	0
Adhesive-thread	0	0	0	20	0	6	2	9	0	0
Deformed-thread	4	2	22	26	1	7	16	10	0	0
M3-half-washer	0	0	1	8	0	2	10	13	0	0
M3-washer	0	0	0	0	0	0	0	0	**36**	3
M4-washer	0	0	0	3	0	6	8	13	1	3
Material-in-lower-part	0	0	1	7	0	1	0	0	0	0
Material-in-screw-head	1	3	7	3	1	2	4	0	1	0
Offset-of-screw-hole	0	0	0	0	0	0	0	0	5	0
Offset-of-work-piece	1	0	10	16	0	5	5	2	0	0
Shortening-the-screw	0	0	0	0	0	0	0	0	0	0
Surface-lubricant	0	0	0	**25**	0	8	0	8	0	0
Surface-sanded-40	0	0	8	20	0	2	7	2	0	0
Surface-sanded-400	0	0	10	10	1	6	6	4	0	0
Surface-used	18	8	14	21	9	14	7	0	0	0
Tearing-off-screw	0	0	0	1	0	0	0	0	1	0

Table 56: Error category to cluster mappings for Black category - I (Affinity propagation)

In Table 57, no significant numbers are observed, with nearly half of the values being zero. This shows that most observations are mapped to clusters 0 to 9. The overall accuracy of cluster-to-error category mapping is 0.58, and the ARI score is 0.134. This accuracy score is similar to those of K-shape and SOM clustering but lower than that of spectral clustering.

Cluster Label / True Label	10	11	12	13	14	15	16	17	18	19	20	21
Baseline	0	0	0	0	0	0	0	0	5	0	0	0
Adhesive-thread	0	1	3	0	0	0	0	0	8	0	0	1
Deformed-thread	0	0	0	0	0	0	0	0	12	0	0	0
M3-half-washer	1	3	1	0	0	0	1	0	5	0	2	3
M3-washer	0	0	0	0	0	0	0	11	0	0	0	0
M4-washer	2	0	0	0	5	2	0	0	2	2	2	1
Material-in-lower-part	0	18	16	0	0	0	0	0	7	0	0	0
Material-in-screw-head	10	0	0	0	0	0	0	0	1	0	0	17
Offset-of-screw-hole	0	7	7	6	1	5	5	12	0	0	0	2
Offset-of-work-piece	0	0	0	0	0	0	0	0	11	0	0	0
Shortening-the-screw	7	0	0	0	0	0	0	0	0	19	19	2
Surface-lubricant	0	0	0	0	0	0	0	0	9	0	0	0
Surface-sanded-40	0	0	1	0	0	0	0	0	9	0	0	1
Surface-sanded-400	0	0	0	0	0	0	0	0	12	0	0	1
Surface-used	0	1	0	0	0	0	0	0	6	0	1	1
Tearing-off-screw	31	0	0	0	0	1	0	0	0	1	1	9

Table 57: Error category to cluster mappings for Black category - II (Affinity propagation)

6.5 Unsupervised models combined results

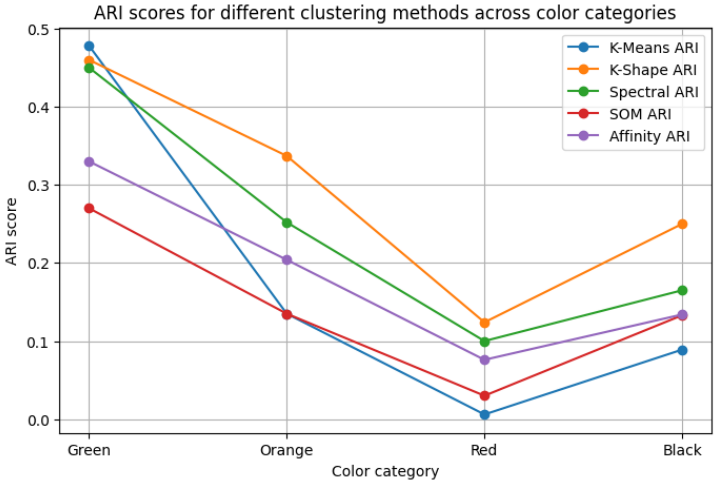


Figure 37: ARI scores lineplot for differnt clustering methods.

Figure 37 shows the line plot of the ARI scores for different clustering methods across all color categories. From the plot, it is evident that K-means has a very high ARI score in the green category but drops significantly in the other categories. The ARI scores from the K-shape algorithm consistently dominate all other clustering methods across all color categories, followed by spectral clustering and affinity propagation. This indicates that the K-shape algorithm forms better clusters compared to the other methods.

The clusters formed by the SOM algorithm are not as well-defined, as indicated by its lower ARI scores compared to spectral, K-shape, and affinity propagation. In the orange category,

the ARI scores of SOM and K-means are the same. From the ARI scores, we can infer that the clusters obtained in the green and orange categories are well-separated compared to those in the red category. The ARI scores are very low for all models in the red category, which is expected, as the data from all error categories are very similar and cannot be easily separated into distinct clusters. The ARI scores for the black category are still low but better than those for the red category.

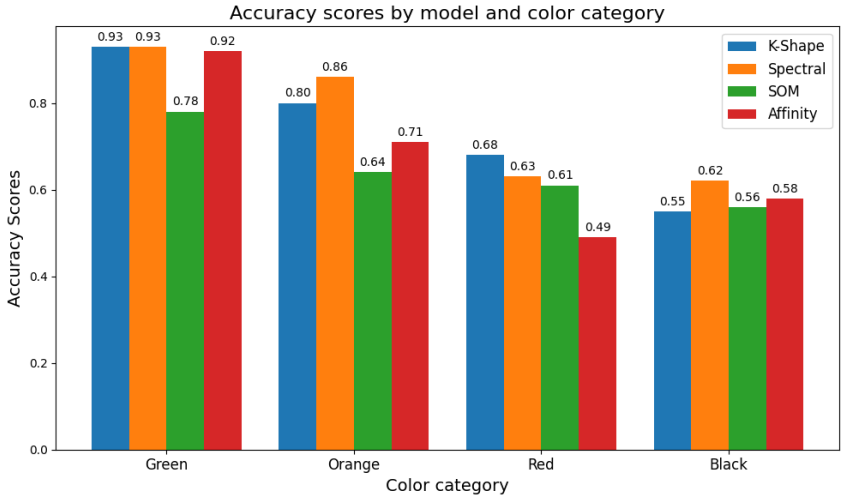


Figure 38: Accuracy scores multi-bar chart for differnt clustering methods.

Figure 38 shows a multi-bar chart for the accuracy scores obtained after mapping the clusters to their respective error categories. The K-means accuracy results are not shown in the plot because all the observations are mapped to the same cluster, resulting in artificially high accuracy scores that are not comparable to the results from other algorithms.

The results confirm that clustering methods perform well in identifying screw errors in the green and orange categories, as these categories have some unique patterns. In the green category, only the SOM method has a lower accuracy of 78%, whereas all other methods achieve an accuracy of 93%. In the orange category, spectral clustering dominates other algorithms with an 86% accuracy.

As expected, poor accuracy results are observed in the red category. Affinity propagation performs the worst, with an overall accuracy of 49%. Additionally, all clustering methods achieve an accuracy of 55% to 62% on the entire dataset (black category). This indicates that unsupervised methods can only achieve a maximum of 62% accuracy in identifying multiple error categories, which is significantly lower compared to supervised models. The best accuracy from a supervised model is 79%, achieved using the ROCKET classifier. This suggests that unsupervised models struggle to identify multiple error categories in the screw-tightening process.

6.6 Cluster ensemble results

The two cluster ensemble approaches explained in section 4.18 are used to combine the results from different clustering methods. For the ensemble method, only K-shape clustering and spectral clustering are used because they perform better on our screw-tightening dataset compared to other clustering methods, as shown in the ARI and accuracy scores graphs. The cluster ensemble approach is performed only on the black color category (entire screw-tightening dataset).

6.6.1 Co-association matrix method

First, a binary co-occurrence matrix is created separately for the K-shape and spectral clustering methods. Then, the co-association matrix is created by averaging all binary co-occurrence matrices. This matrix serves as the similarity matrix for our final spectral clustering algorithm. The cluster-to-error category mapping matrix is shown in table 58.

Cluster Label / True Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Baseline	39	0	33	27	0	13	**86**	15	0	0	0	**70**	0	0	3	37	64	13
Adhesive-thread	1	0	2	0	0	2	1	0	0	**40**	1	1	1	1	0	0	0	0
Deformed-thread	11	0	7	1	0	3	5	10	0	20	1	28	2	0	1	6	2	3
M3-half-washer	0	0	2	2	2	1	0	0	1	0	17	0	10	4	9	1	0	1
M3-washer	0	0	0	0	0	0	0	0	**50**	0	0	0	0	0	0	0	0	0
M4-washer	1	0	6	3	0	2	0	0	11	0	8	0	10	3	5	1	0	0
Material-in-lower-part	0	0	2	0	**30**	8	0	2	2	1	2	0	0	0	0	0	1	2
Material-in-screw-head	1	5	2	2	0	3	1	3	0	2	0	1	2	17	0	7	1	3
Offset-of-screw-hole	0	0	0	0	19	0	0	0	**28**	0	1	0	0	2	0	0	0	0
Offset-of-work-piece	3	0	5	0	1	13	1	3	0	0	1	5	0	0	0	5	2	11
Shortening-the-screw	0	**42**	0	0	0	0	0	0	0	0	0	0	0	2	0	0	3	0
Surface-lubricant	0	0	8	0	0	4	0	0	0	0	22	0	0	0	16	0	0	0
Surface-sanded-40	3	0	3	0	1	14	0	4	0	0	8	1	0	1	6	5	0	4
surface-sanded-400	0	0	6	1	0	16	0	5	0	0	4	3	1	1	4	6	0	3
Surface-used	2	0	14	10	0	4	21	8	0	0	0	9	0	1	0	8	12	11
Tearing-off-screw	0	**10**	0	1	0	0	0	0	2	2	0	0	5	**25**	0	0	0	0

Table 58: Error category to cluster mappings for Co-association matrix cluster ensemble method

From Table 58, we notice that even after using the cluster ensemble approach, the observations from each error category are scattered across the clusters. For the *baseline* category, most observations are in clusters 6 and 11, but some are also in clusters 15, 16, and 17. In the case of the *material-in-lower-part* category, 30 out of 50 observations are mapped to cluster 4. Half of the observations from the *tearing-off-screw* category are mapped to cluster 13, with about 10 observations mapped to cluster 1. Additionally, we observe that 42 observations from the *shortening-the-screw* category belong to cluster 1.

On the other hand, the *M3-washer-in-upper-part* category is perfectly mapped to cluster 8 and, as expected, has significant overlap with the *offset-of-screw-hole* error category. Finally, 80%

of the observations from the *adhesive-thread* category are limited to cluster 9. The ARI of the cluster mapping is 0.160, and the overall accuracy of cluster-to-error-category mapping is 57.5%. The ARI score is lower than that of the K-shape method but similar to that of spectral clustering, whereas the mapping accuracy score is similar to K-shape but lower than spectral clustering.

6.6.2 Relabel and maximum voting method

Table 59 shows the results from the relabel and maximum voting method of the cluster ensemble approach. As explained, the cost matrix is calculated for the process of relabeling and aligning the K-shape clustering results with the spectral clustering results. Next, the majority voting approach is used to decide the final cluster mapping for each observation in the dataset.

Cluster Label / True Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Baseline	2	**76**	3	**103**	0	25	0	0	1	2	0	14	0	0	**97**	32	0	15
Adhesive-thread	0	0	5	1	1	0	10	1	2	0	0	0	0	30	0	0	0	0
Deformed-thread	0	8	6	6	0	1	2	1	9	0	0	4	0	26	28	5	0	4
M3-half-washer	0	0	**29**	0	4	2	1	8	2	0	1	0	2	0	0	1	0	0
M3-washer	0	0	1	0	0	0	13	0	0	0	**36**	0	0	0	0	0	0	0
M4-washer	2	0	**29**	0	5	2	1	2	0	0	8	0	0	0	1	0	0	0
Material-in-lower-part	0	1	0	0	0	0	2	2	3	12	2	0	**28**	0	0	0	0	0
Material-in-screw-head	5	1	3	1	17	2	0	0	6	2	0	2	0	2	1	6	0	2
Offset-of-screw-hole	0	0	0	0	8	1	3	1	0	6	18	0	13	0	0	0	0	0
Offset-of-work-piece	0	2	2	1	0	0	0	1	20	8	0	9	0	0	6	0	0	1
Shortening-the-screw	**45**	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
Surface-lubricant	0	0	16	0	0	0	0	**28**	2	0	0	4	0	0	0	0	0	0
Surface-sanded-40	0	0	6	0	1	0	0	8	10	2	0	13	1	0	4	2	0	3
surface-sanded-400	0	0	5	0	1	1	0	4	11	2	0	15	0	0	3	4	0	4
Surface-used	2	27	0	16	1	9	0	0	22	9	0	2	0	0	6	2	0	4
Tearing-off-screw	10	0	5	0	**27**	1	0	0	0	0	0	0	0	2	0	0	0	0

Table 59: Error category to cluster mappings for Relabel and maximum voting cluster ensemble method

From Table 59, we notice that most observations from the *baseline* category are assigned to clusters 1, 3, and 14. Interestingly, exactly 29 observations from both error categories (*M3-half-washer-in-upper-part* and *M4-washer-in-upper-part*) are assigned to cluster 2. For the error categories *shortening-the-screw* and *tearing-off-screw*, the results and mappings are similar to those from the co-association matrix method. Moreover, 28 observations from *surface-lubricant* are mapped to cluster 7, 36 observations from *M3-washer-in-upper-part* are assigned to cluster 10, and 28 observations from *material-in-lower-part* are assigned to cluster 12.

The ARI score for this cluster ensemble result is 0.191, which is lower than that of the K-shape method but higher than that of spectral clustering. The overall accuracy of cluster-to-error-category mapping is 47.5%, which is lower than both individual clustering methods.

From these results, we can state that the cluster ensemble approaches do not improve the mapping of error categories to individual clusters. In the case of the relabel and maximum voting method, the accuracy is significantly lower compared to the individual accuracy of the models.

6.7 Prediction using Streamlit

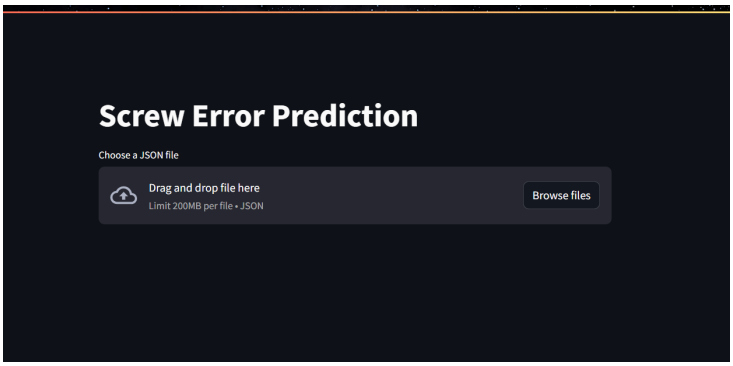


Figure 39: Streamlit web page.

The error category predicted for a new screw-tightening JSON file is displayed using a Streamlit web application. The results for anomaly and non-anomaly data are shown in the images below. Figure 39 displays the start page of the Streamlit app, where the user can upload the JSON file of a specific screw-tightening run. Once the JSON file is received, a preprocessing code file is triggered, which performs all the preprocessing steps mentioned in subsection 5.4 on the torque data extracted from the JSON file.

Figure 40: Streamlit prediction for OK data.

Next, semi-supervised models mentioned in subsection 6.1 are applied to the torque data. Figure 40 shows the output of the weighted ensemble model prediction when the screw-tightening is not an anomaly. First, the torque values from the JSON file are displayed using a line plot which also has different colors for each phase in screw-tightening, providing an overview of the screw-tightening run to the user. Next, information on the file name, normalized torque, and anomaly prediction is displayed in a table. If the screw run is not an anomaly, the prediction will be 0, and this is also indicated below the table.

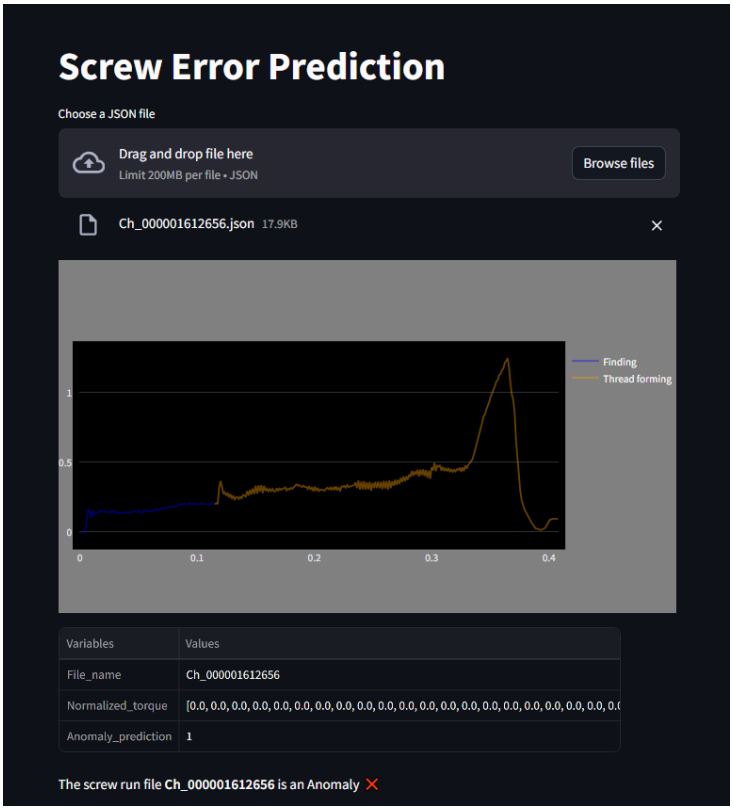


Figure 41: Streamlit prediction for anomaly data.

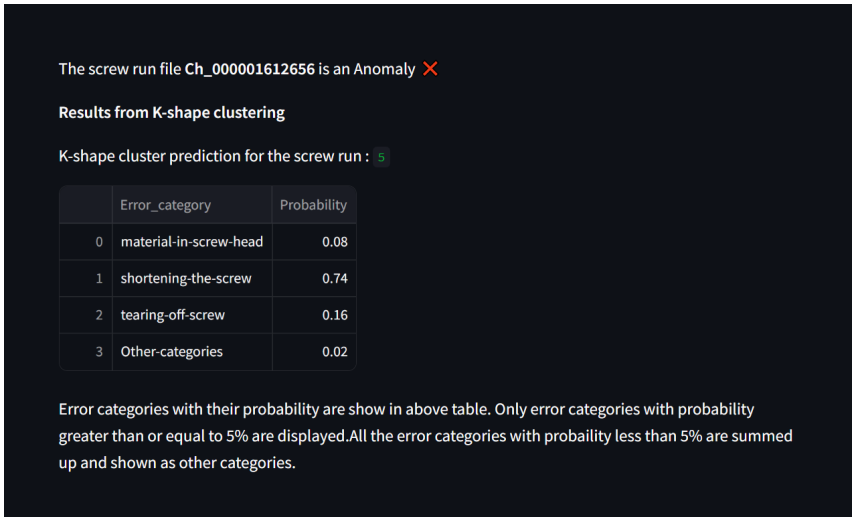


Figure 42: K-shape model results for anomaly screw run prediction.

Figures 41 and 42 show the results displayed when the screw run is classified as an anomaly. The K-shape clustering algorithm, is used in the prediction pipeline because of its stable result and ease of use. Figure 41 is similar to figure 40, with the only difference being the prediction result. If the file is classified as an anomaly, the torque sequence is passed to the K-shape algorithm to get the cluster label for the error category. Since the predicted cluster might contain errors from multiple categories, it is helpful for the user to see the probability of the screw run belonging to a particular cluster.

In this case, the file belonging to the error category *shortening-the-screw* is given as input and the predicted cluster label for this file is 5, as shown in the figure 42. This is followed by a table showing the probabilities of the file belonging to particular error classes. It can be noticed that *shortening-the-screw* has a higher probability (0.74) compared to all other error categories. Finally, if the probability of the screw-tightening file belonging to different error categories is less than 5%, these probabilities are summed up and shown in the last row as the probability of the screw run belonging to 'Other-categories'.

7 Conclusion and Future Work

In this thesis, a detailed investigation was conducted on unsupervised and semi-supervised approaches to detect anomalies in the screw-tightening dataset. Anomalies can occur in various forms: point, contextual, and collective anomalies. This thesis primarily focused on collective time series anomalies, with the aim of distinguishing whether an entire screw-tightening run was an anomaly or not by analyzing the complete sequence. Additionally, a brief exploration of supervised models was carried out.

The dataset provided included 15 error categories and 4 different sub-categories of baseline (OK sequences). The provided dataset is small as there are only 1342 time series. Torque data was used for all analysis and modeling.

In the initial phase, data pre-processing steps were performed to clean the torque data. This included removing duplicate values and filling in missing values using linear interpolation. Since the time series did not all have a constant length, a detailed analysis was conducted to determine an appropriate constant length, which was set to 1200. This length was chosen to capture patterns in the screw-tightening runs effectively without losing significant information. For shorter time series, a padding method was applied to fill in zeros at the beginning, shifting the series but preserving its features. Finally, min-max normalization was applied to standardize the torque data.

In the subsequent phase, semi-supervised models were employed to classify whether the screw-tightening sequences were anomalies or not. Models such as Isolation Forest, OC-SVM, and Autoencoder were utilized. The results indicated that the Isolation Forest and Autoencoder models performed similarly and were highly effective in anomaly detection. The Autoencoder model demonstrated greater stability with changes in the dataset. The OC-SVM model also

performed well, but it was not as effective as the other two models, with more frequent misclassifications. Further analysis revealed that the *baseline-extra* sub-category of OK samples had a different data distribution, affecting the results when included. Consequently, the *baseline-extra* category was removed from the dataset.

Based on the recall scores, color categories (green, orange, and red) were defined, with green indicating anomalies that were easy to identify and red indicating anomalies that were more challenging to detect. A weighted ensemble approach was used to combine the results from the three semi-supervised models. The classification report for the entire dataset showed an accuracy of 83%, which was a slight improvement over the Autoencoder model alone. The precision and recall scores also improved, resulting in fewer misclassifications. Thus, the task of determining whether the screw-tightening process was an anomaly was achieved with good results.

In the third step, if the tightening process was predicted to be an anomaly, several clustering methods were used to determine the type of anomaly or the error category to which this particular anomaly belonged. Since the industry datasets are mostly unlabelled, a detailed study of unsupervised methods was performed to group similar anomalies into the same cluster. The results showed that the Simple K-means algorithm performed the worst across all color categories, as all error categories were grouped into a single cluster. The Self-Organizing Maps (SOM) clustering results were similar to K-means in terms of the Adjusted Rand Index (ARI) score but with a slight improvement across all color categories. On the other hand, the K-shape and Spectral clustering methods performed notably better compared to other algorithms. They excelled at differentiating errors in the green and orange error categories but performed poorly with the red color category. Since the time series data in the red category were nearly identical, it was not possible to separate them using any clustering algorithm. For the black color category, the highest accuracy was achieved by Spectral clustering, with an accuracy of 0.62. This indicated that although these models were better compared to other clustering models, the overall results were still not very high. Additionally, intermediate performance was observed with the Affinity Propagation clustering method. A prediction pipeline was created, following the architecture shown in figure 11 and utilizing a Streamlit app to display the results with probabilities.

A brief investigation into supervised models was conducted by applying six classification models, each from a different family. The worst accuracy results were obtained from the KNN DTW and CNN classifiers, both achieving an accuracy of 0.52. Conversely, the best accuracy was achieved by the ROCKET and TSF classifiers, with accuracy ranging from 0.74 to 0.79. The Shape Transform and Catch22 classifiers yielded average accuracy results, ranging from 0.65 to 0.67.

The highest accuracy from supervised models was 0.79, while the highest from unsupervised models was 0.62. This demonstrated that supervised models performed significantly better compared to unsupervised models.

The work of this thesis primarily focused on semi-supervised and unsupervised methods for anomaly detection. Given that the small study on supervised methods indicated better per-

formance, further research into supervised models can be carried out. Additionally, a hybrid approach combining supervised and unsupervised methods can be developed for screw-tightening anomaly detection. Increasing the dataset size can also be beneficial for training supervised models and achieving more stable results. Finally, in the initial step, errors can be categorized based on the shape of the time series. This dataset can be used for further investigation. By exploring these areas, a more robust anomaly detection process can be developed, which would be useful and easily applicable for any anomaly detection use case in the manufacturing industry.

8 List of references

Adari, S. K. and Alla, S. (2024), *Beginning Anomaly Detection Using Python-Based Deep Learning*, 2nd edn, Apress Berkeley, CA.

Arratia, A. and Sepúlveda, E. (2020), Convolutional neural networks, image recognition and financial time series forecasting, *in* V. Bitetta, I. Bordino, A. Ferretti, F. Gullo, S. Pascolutti and G. Ponti, eds, 'Mining Data for Financial Applications', Springer International Publishing, Cham, pp. 60–69.

Asan, U. and Ercan, S. (2012), *An Introduction to Self-Organizing Maps*, pp. 299–319.

Berahmand, K., Daneshfar, F., Salehi, E. S. and et al. (2024), 'Autoencoders and their applications in machine learning: a survey', *Artificial Intelligence Review* .

Blessing, E. and Klaus, H. (2023), 'Anomaly detection techniques to identify outliers or anomalies in datasets, which could be indicative of errors or noteworthy events.', **3481**, 18.

Cao, X., Liu, J., Meng, F. and et al. (2019), Anomaly detection for screw tightening timing data with lstm recurrent neural network, *in* '2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)', pp. 348–352.

Chacón, J. E. and Rastrojo, A. I. (2023), 'Minimum adjusted rand index for two clusterings of a given size', *Advances in Data Analysis and Classification* **17**(1), 125–133.
URL: *https://doi.org/10.1007/s11634-022-00491-w*

Chapra, S. C. and Canale, R. P. (2002), *Numerical Methods for Engineers*, McGraw-Hill.

Choi, K., Yi, J., Park, C. and Yoon, S. (2021), 'Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines', *IEEE Access* **PP**, 1–1.

Ciaburro, G., Ayyadevara, V. K. and Perrier, A. (2018), *Hands-On Machine Learning on Google Cloud Platform*, Packt Publishing.

Dalianis, H. (2018), *Evaluation Metrics and Evaluation*, Springer International Publishing, Cham, pp. 45–53.
URL: *https://doi.org/10.1007/978-3-319-78503-5_6*

Dempster, A., Petitjean, F. and Webb, G. I. (2019), 'ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels', *CoRR* **abs/1910.13051**.
URL: *http://arxiv.org/abs/1910.13051*

Deng, H., Runger, G. C., Tuv, E. and et al. (2013), 'A time series forest for classification and feature extraction', *CoRR* **abs/1302.2277**.
URL: *http://arxiv.org/abs/1302.2277*

Dodge, Y. (2008), *The Concise Encyclopedia of Statistics*, Springer New York, New York, NY, pp. 327–329.
URL: *https://doi.org/10.1007/978-0-387-32833-1_243*

Doniec, R., Konior, J., Sieciński, S. and et al. (2023), 'Sensor-based classification of primary and secondary car driver activities using convolutional neural networks', *Sensors* **23**, 5551.

Frey, B. and Dueck, D. (2007), 'Clustering by passing messages between data points', *Science (New York, N.Y.)* **315**, 972–6.

Galante, L. and Banisch, R. (2019), A Comparative Evaluation of Anomaly Detection Techniques on Multivariate Time Series Data, PhD thesis.

Gil-Aluja, J. (1998), *The Hungarian assignment algorithm*, Springer US, Boston, MA, pp. 148–158.
URL: *https://doi.org/10.1007/978-1-4613-3329-6_24*

Glavin, F. (2009), A One-Sided Classification Toolkit with Applications in the Analysis of Spectroscopy Data, PhD thesis.

Goldstein, M. and Uchida, S. (2016), 'A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data', *PloS one* **11**, e0152173.

Haben, S., Voss, M. and Holderbaum, W. (2023), *Time Series Forecasting: Core Concepts and Definitions*, Springer International Publishing, Cham, pp. 55–66.
URL: *https://doi.org/10.1007/978-3-031-27852-5_5*

Hills, J., Lines, J., Baranauskas, E. and et al. (2013), 'Classification of time series by shapelet transformation', *Data Mining and Knowledge Discovery* **28**.

Hu, M., Deng, X. and Yao, Y. (2018), A sequential three-way approach to constructing a co-association matrix in consensus clustering, *in* H. S. Nguyen, Q.-T. Ha, T. Li and M. Przybyła-Kasperek, eds, 'Rough Sets', Springer International Publishing, Cham, pp. 599–613.

Huang, C. (2018), Featured Anomaly Detection Methods and Applications, PhD thesis, University of Exeter.

Ippolito, P. P. (2022), *Hyperparameter Tuning*, Springer International Publishing.
URL: *https://doi.org/10.1007/978-3-030-88389-8_12*

James, G., Witten, D., Hastie, T. and et al. (2021), *An Introduction to Statistical Learning: with Applications in R*, Springer Texts in Statistics, 2 edn, Springer New York, NY. Springer Texts in Statistics.
URL: *https://doi.org/10.1007/978-1-0716-1418-1*

Jin, X. and Han, J. (2017), *K-Means Clustering*, Springer US, Boston, MA, pp. 695–697.
URL: *https://doi.org/10.1007/978-1-4899-7687-1_431*

Jo, T. (2023), *Ensemble Learning*, Springer International Publishing, Cham, pp. 83–109.
URL: *https://doi.org/10.1007/978-3-031-32879-4_4*

Kamat, P. and Sugandhi, R. (2020), Anomaly detection for predictive maintenance in industry 4.0-a survey, Vol. 170.

Lei, H. and Sun, B. (2007), A study on the dynamic time warping in kernel machines, *in* '2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System', pp. 839–845.

Leporowski, B., Tola, D., Hansen, C. and et al. (2021), 'Detecting faults during automatic screwdriving: A dataset and use case of anomaly detection for automatic screwdriving'.
URL: *https://arxiv.org/abs/2107.01955*

Liu, D., Fei, S., Hou, Z. and et al. (2007), *Advances in Neural Networks - ISNN 2007*, 1st edn, Springer Berlin, Heidelberg.

Liu, F. T., Ting, K. and Zhou, Z.-H. (2009), Isolation forest, pp. 413 – 422.

Liu, S., Li, Z., Wang, G. and et al. (2023), 'Spectral-spatial feature fusion for hyperspectral anomaly detection'.

Lubba, C. H., Sethi, S. S., Knaute, P. and et al. (2019), 'catch22: Canonical time-series characteristics', *CoRR* **abs/1901.10200**.
URL: *http://arxiv.org/abs/1901.10200*

Luxburg, U. (2007), 'A tutorial on spectral clustering', *Statistics and Computing* **17**(4), 395–416.
URL: *https://doi.org/10.1007/s11222-007-9033-z*

Mucherino, A., Papajorgji, P. J. and Pardalos, P. M. (2009), *k-Nearest Neighbor Classification*, Springer New York, New York, NY, pp. 83–106.
URL: *https://doi.org/10.1007/978-0-387-88615-2_4*

Olteanu, M., Rossi, F. and Yger, F. (2023), 'Meta-survey on outlier and anomaly detection', *Neurocomputing* **555**, 126634.
URL: *http://dx.doi.org/10.1016/j.neucom.2023.126634*

Paparrizos, J. and Gravano, L. (2016), 'k-shape: Efficient and accurate clustering of time series', *ACM SIGMOD Record* **45**, 69–76.

Python Core Team (2024), *Python: A dynamic, open source programming language*, Python Software Foundation, Version: 3.10.0. (visited on 25th July 2024).
URL: *https://www.python.org*

Regaya, Y., Fadli, F. and Amira, A. (2021), 'Point-denoise: Unsupervised outlier detection for 3d point clouds enhancement', *Multimedia Tools and Applications* **80**, 1–17.

Ribeiro, D., Matos, L. M., Cortez, P. and et al. (2021), A comparison of anomaly detection methods for industrial screw tightening, *in* 'Computational Science and Its Applications – ICCSA 2021', Springer International Publishing, Cham, pp. 485–500.

Rodríguez, J., Kuncheva, L. and Alonso, C. (2006), 'Rotation forest: A new classifier ensemble method', *IEEE transactions on pattern analysis and machine intelligence* **28**, 1619–30.

Srivastava, D. and Bhambhu, L. (2023), Anomaly detection and time series analysis.

Tarawneh, A. (2021), Deep Learning Applications in Computer Vision: Experimental and Comparative Study, PhD thesis.

Tor, O., Birinci, E., Hu, L. and et al. (2020), 'Effects of pilot hole diameter and depth on screw driving torques in plywood', *Bioresources* **15**, 8121–8132.

Vega-Pons, S. and Ruiz-Shulcloper, J. (2011), 'A survey of clustering ensemble algorithms.', *IJPRAI* **25**, 337–372.

Vens, C. (2013), *Random Forest*, Springer New York, New York, NY.
URL: *https://doi.org/10.1007/978-1-4419-9863-7_612*

Wang, Z., Li, K., Xia, S. and et al. (2021), 'Economic recession prediction using deep neural network'.

West, N. and Deuse, J. (2024), A comparative study of machine learning approaches for anomaly detection in industrial screw driving data.

West, N., Schlegl, T. and Deuse, J. (2023), 'Unsupervised anomaly detection in unbalanced time series data from screw driving processes using k-means clustering (pre-print)'.

Yan, P., Abdulkadir, A., Rosenthal, M. and et al. (2023), 'A comprehensive survey of deep transfer learning for anomaly detection in industrial time series: Methods, applications, and directions'.

Yuki, S., Yoshiyuki, N. and Masayuki, S. (2023), A case study of real-time screw tightening anomaly detection by machine learning using real-time processable features.
URL: *https://api.semanticscholar.org/CorpusID:265030666*

Zhang, Q., Zhang, C., Cui, L. and et al. (2023), 'A method for measuring similarity of time series based on series decomposition and dynamic time warping', *Applied Intelligence* **53**, 6448–6463.

Zhou, X.-H., Obuchowski, N. A. and McClish, D. K. (2011), *Statistical Methods in Diagnostic Medicine*, 2nd edn, John Wiley & Sons, New York.

Zhou, Z.-H. (2012), *Ensemble Methods: Foundations and Algorithms*, 1st edn, Chapman & Hall/CRC.

Appendix

A Additional Tables

No.	Catch22 feature name	Description
1	DN_HistogramMode_5	Mode value of z-score distribution in 5 bin histogram
2	DN_HistogramMode_10	Mode value of z-score distribution in 10 bin histogram
3	SB_BinaryStats_mean_longstretch1	Longest sequence of positive values above mean
4	DN_OutlierInclude_p_001_mdrmd	Time period between consecutive extreme events above the mean value
5	DN_OutlierInclude_n_001_mdrmd	Time period between consecutive extreme events below the mean value
6	CO_f1ecac	The first value of $1/e$ exceeding the autocorrelation function
7	CO_FirstMin_ac	First minimum value of the autocorrelation function
8	SP_Summaries_welch_rect_area_5_1	Total power value in the lowest fifth of frequencies in Fourier power spectrum
9	SP_Summaries_welch_rect_centroid	Centroid value of Fourier power spectrum
10	FC_LocalSimple_mean3_stderr	Mean error value of a rolling 3-sample mean forecasting
11	CO_trev_1_num	Time reversibility statistic
12	CO_HistogramAMI_even_2_5	Automutual information
13	IN_AutoMutualInfoStats_40_gaussian_fmmi	First minimum value of the automutual information function
14	MD_hrv_classic_pnn40	Proportion of successive differences exceeding 0.04σ
15	SB_BinaryStats_diff_longstretch0	Longest period of consecutive increases and decreases
16	SB_MotifThree_quantile_hh	Shannon entropy of two successive letters in equiprobable 3-letter symbolization
17	FC_LocalSimple_mean1_tauresrat	Change in correlation length after iterative differencing

Table 60: Catch22 Features and descriptions - I

No.	Catch22 feature name	Description
18	CO_Embed2_Dist_tau_d_expfit_meandiff	Exponential fit to successive distances in 2-d embedding space
19	SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1	Proportion of slower timescale fluctuations that scale with DFA
20	SC_FluctAnal_2_rsrangefit_50_1_logi_prop_r1	Proportion of slower timescale fluctuations that scale with linearly rescaled range fits
21	SB_TransitionMatrix_3ac_sumdiagcov	Trace of covariance of transition matrix between symbols in 3-letter alphabet
22	PD_PeriodicityWang_th0_01	Periodicity measure

Table 61: Catch22 Features and descriptions - II

Eidesstattliche Versicherung

(Affidavit)

Bhageradhi, Naveen kumar

Name, Vorname
(surname, first name)

231678

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Unsupervised approaches for anomaly detection in the quality management of screw connections

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 05.08.2024

Ort, Datum
(place, date)

B. Naveen

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 05.08.2024

Ort, Datum
(place, date)

B. Naveen

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**