MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION, DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN

Thesis for the Degree of Bachelor in Computer Science 15.0 credits

MACHINE LEARNING BASED PREDICTIVE DATA ANALYTICS FOR EMBEDDED TEST SYSTEMS

Fayad Al Hanash
Fah20002@student.mdu.se

Examiner: Mobyen Uddin Ahmed
Mälardalen University, Västerås, Sweden

Supervisor: Shahina Begum, Md Alamgir Kabir & Md Rakibul Islam
Mälardalen University, Västerås, Sweden

Company Supervisor: Daniel Bengtsson
Nordic Engineering Partner, Stockholm, Sweden

29/09/2023

Abstract

Organizations gather enormous amounts of data and analyze these data to extract insights that can be useful for them and help them to make better decisions. Predictive data analytics is a crucial subfield within data analytics that make accurate predictions. Predictive data analytics extracts insights from data by using machine learning algorithms. This thesis presents the supervised learning algorithm to perform predicative data analytics in Embedded Test System at the Nordic Engineering Partner company. Predictive Maintenance is a concept that is often used in manufacturing industries which refers to predicting asset failures before they occur. The machine learning algorithms used in this thesis are support vector machines, multi-layer perceptrons, random forests, and gradient boosting. Both binary and multi-class classifier have been provided to fit the models, and cross-validation, sampling techniques, and a confusion matrix have been provided to accurately measure their performance. In addition to accuracy, recall, precision, f1, kappa, mcc, and roc auc measurements are used as well. The prediction models that are fitted achieve high accuracy.

Acknowledgements

I would like to thank my supervisors Shahina Begum, Md Alamgir Kabir & Md Rakibul Islam for their support and advice during the thesis. I would also like to thank Daniel Bengtsson for sharing the ETS data and knowledge that allowed me to work on the thesis. Finally, I would like to thank my family for their encouragement and support during my studies.

Table of Contents

List of Figures

List of Tables

List of abbreviations

AI	*Artificial Intelligence*
ANN	Artificial Neural Networks
CM	Confusion Matrix
CRISP-DM	Cross Industry Standard Process for Data Mining
CV	Cross Validation
EOL	End of Line
EQ	Measurement equal to the low limit
ETS	Embedded Test System
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GB	Gradient Boosting
GELE	Measurement >= low limit and measurement <= high limit
GTLT	Measurement > low limit and measurement < high limit
ICT	In-Circuit Test
LINQ	Language-Integrated Query
MCC	Matthews Correlation Coefficient
ML	Machine Learning
MLP	Multi-Layer Perceptron
NEP	Nordic Engineering Partner
NULL	No comparison
PCB	Printed Circuit Board
PCBA	Printed Circuit Board Assembly
PdM	Predictive Maintenance
PvM	Preventive Maintenance
R2F	Run-to-failure
RF	Random Forest
ROC	Receiver Operating Characteristic
ROC AUC	Area Under the Receiver Operating Characteristic Curve
SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
TPR	True Positive Rate
V	Volt
ÂµA	Microampere

1. Introduction

Nowadays, organizations gather enormous amounts of data and analyze these data to extract insights that can be useful for them and help them to make better decisions [1]. Predictive data analytics is a crucial subfield within data analytics that make accurate predictions. Predictive data analytics extracts insights from data by using machine learning algorithms. There are four types of machine learning algorithms such as supervised learning, unsupervised learning, semi-supervised learning, and reinforcement [1]. This thesis presents the supervised learning algorithm to perform predicative data analytics in Embedded Test System (ETS) at the Nordic Engineering Partner company (NEP). The increasing availability of data and the increasing capability of hardware have encouraged industries to apply machine learning (ML) approaches in maintenance management domains [2]. The concept Predictive Maintenance (PdM) is often used in manufacturing industries which refers to predicting the asset failures before they occur [3]. There are many advantages to using predictive maintenance in production environments, but there are also challenges to be overcome [4]. The PdM increases productivity and efficiency and decreases the system faults and unplanned downtimes [4]. The necessity of integrating data from diverse systems and sources within a facility to gather accurate data for creating prediction models is a challenge that has to be overcome [4]. Additionally, the use of artificial intelligence (AI) raises several challenges such as "obtaining training data, dealing with dynamic operating environments, selecting the ML algorithm that better fits to a given scenario, and the necessity of context-aware information, such as operational conditions, and production environment" [4]. Furthermore, the use of ML for predictive data analytics faces several challenges as well, such as extracting and selecting the features from the database and fitting and developing the models to predict the failure of ETS. In this thesis, the implementation of a ML-based application is carried out using the experimental and Cross Industry Standard Process for Data Mining (CRISP-DM) methodologies [1].

2. Background

2.1. Embedded Test System

The Nordic Engineering Partner is a company that develops and assembles advanced In-Circuit Test (ICT) fixtures executing through End of Line (EOL) tests on Printed Circuit Board Assemblies (PCBAs), these fixtures are called Embedded Test System. ICT is a type of electrical test and used to ensure that the PCB is functioning correctly and identify any potential defects. ICT uses a bed-of-nails test fixture that makes electrical contact to every net on the PCB [5]. EOL testing is a crucial part in ensuring the quality and reliability of PCBAs and represents the final functional check of PCBA before it is shipped to the customers [6]. PCBA is the process of attaching electronic components to a substrate that is printed with a particular circuit. PCBA ensures components are electrically interconnected and allows them to communicate with each other [7]. The Embedded Test System is shown in Figure 1 and is an "all-in-one solution" for testing electronic products. The ETS consist of a test fixture, a computer with operators panel and tester specific boards such as power supply, input/output channels etc.[1] ETS fixtures are hardware and software tools that are designed to automate tests on the PCBA, such as functional tests, stress tests, and environmental tests. ETS fixtures improve the quality and reliability of the PCBA by increasing test efficiency, reducing human error, and improving test coverage and accuracy.

[1] Embedded Test System. https://nepartner.se/products/embedded-test-system/

Figure 1 Embedded Test System

2.2. Machine Learning

Arthur Samuel coined the term "Machine Learning" in 1959 in the context of letting the machine solve the game of checkers. Machine learning refers to a computer program that has the ability to learn in order to produce behavior that was not explicitly programmed by the programmer, and it can display behavior that the programmer might not even be aware of [8][9]. The program learns the behaviors based on the following factors: the consumed data by the program, a metric that measures the error or the deviation between the ideal behavior and current behavior, and a feedback mechanism that utilizes the measured error to guide the program to produce better behavior in future occurrences [8]. ML is a subset of artificial Intelligence [10]. In contrast to AI-Applications, ML involves learning of hidden patterns within the data and uses these patterns for prediction or classification [10]. Since making inferences from samples is the core task of ML, the ML builds statical models by using the theory of statistics [11].

2.2.1. Supervised Learning

There are four types of machine learning algorithms such as supervised learning, unsupervised learning, semi-supervised learning, and reinforcement [1]. Supervised learning simply refers to learning from examples. Supervised learning techniques aim to learn a model using a training set of input-output pairs and later use this model to make predictions (predicate outputs) on a test set. The inputs are also called descriptive features and the output is called a target feature or label [1][12][13].
Let $(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$ represent a set of training examples where:

- x_i is a vector or tuple containing n numbers of descriptive features.
- y_i is a target feature.
- (x_i, y_i) is a pair of x_i and its corresponding y_i.

More specifically, the dataset represents a table in a relational database, and each row in the table is called a record or an example and represents such (x_i, y_i). In addition, each column in the table is called an attribute or a feature and represent such x_i or y_i. The test set is represented in the same way as the training set, but target feature y_i is not known. By applying the supervised learning, y_i can be predicted.

2.2.2. Machine Learning Algorithms

ML provides the knowledge that intelligent machines need to maintain and sustain their functionality. Essentially, ML algorithms are embedded into machines and data flows to extract knowledge and feed it into the system for managing the processes more rapidly and efficiently [10]. Machine learning algorithms utilize the patterns and relationships existing in massive amounts of multidimensional data to automatically learn how to anticipate outcomes, classify data, and perform other tasks. They gain knowledge from a training set of examples, and then they apply that knowledge to any new input [14]. There are a lot of ML algorithms, but in this thesis, support vector machines (SVM), random forest (RF), Multilayer perceptron (MLP) and gradient boosting (GB) were chosen to perform the work. The model's implementation was carried out using the Scikit library. Scikit is a library in Python that supports the implementation of those algorithms[2].

[2] https://scikit-learn.org/

2.2.2.1. Support Vector Machines

The support vector machine algorithm is designed to solve binary classification problems. The idea is to use a hyperplane that separates two classes and maximizes the margin between the classes' closest vectors [8] [15]. As shown in Figure 2 below, the hyperplane is the middle of the margin, and the support vectors are the points on the boundary [15]. When it is difficult to find a hyperplane between the two classes, the kernel approach can be used to separate the classes by projecting the support vectors into a higher-dimensional space [15]. SVM can be used to solve multi-class classification problems by separating the classes into several binary classifiers and solving the corresponding problem. This can be achieved via two techniques:

- One-against-all:
  The one-against-all technique splits the dataset into n-classes, then applies the SVM algorithm for each class against all other classes. This technique creates n-numbers of SVM [16]. For example, the ETS's dataset consists of four classes: passed, failed, error, and terminated. By applying this technique, four SVM binary classifications can be obtained, as follows:
    - Passed vs (failed, error, and terminated).
    - Failed vs (passed, error, and terminated).
    - Error vs (passed, failed, and terminated).
    - Terminated vs (passed, failed, and error).

- One-against-one:
  The one-against-one technique splits the dataset into n-classes, then applies the SVM algorithm for one class against each other classes. This technique creates (n(n-1)/2) numbers of SVMs [16]. By applying this technique to the ETS's dataset, six SVM binary classifications can be obtained, as follows:

    - Passed vs failed.
    - Passed vs error.
    - Passed vs terminated.
    - Failed vs error.
    - Failed vs terminated.
    - Error vs terminated.

  Scikit provides a great support for developing SVM[3]. Scikit uses the one-against-one technique to model SVM for multi-class strategies. The One-against-all matrix is constructed from the one-against-one matrix.

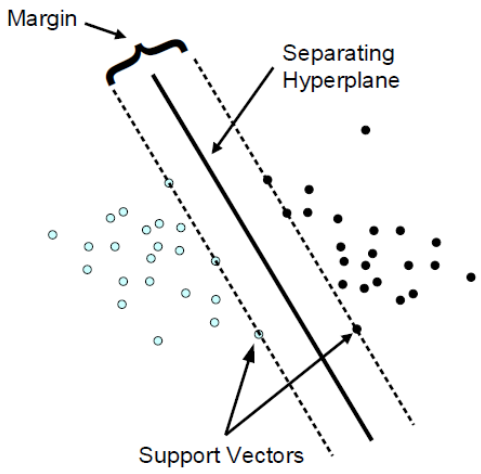

Figure 2 Support Vectors Classification [15]

[3] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

2.2.2.2. Random Forest

Random forest is a tree-based ensemble such that each tree depends on a set of random variables. RF can be used for classification purposes and is quite fast in training and predicting [17]. The approach, which creates many decision tree classifiers on various samples of the dataset while training the model, predicts the target feature on each classifier under the testing phase, then predicts the final target feature by voting for the majority of predicted target features. Figure 3 shows the process of random forest classification [18].

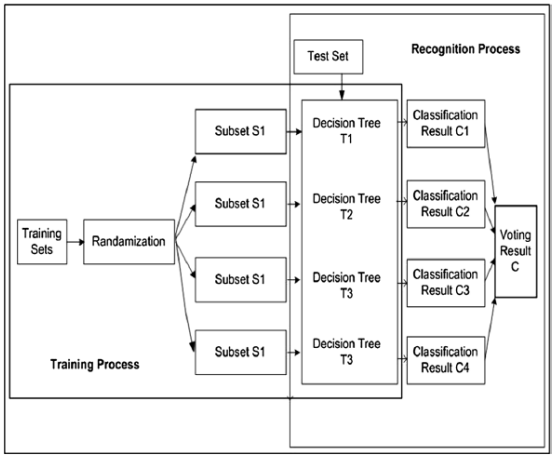

Figure 3 Random Forest Classification [18]

2.2.2.3. Multi-Layer Perceptron

Multi-Layer Perceptron is one of the most popular artificial neural networks (ANN) for solving classification problems. MLP is a feedforward network that consists of multiple layers of perceptrons, the input layer, the output layer, and hidden layers between the input and output layers. Figure 4 illustrates the MLP [19]. The size of hidden layers doesn't depend on the input and output. Each layer has several nodes or neurons that connect all layers with each other. MLP is described as fully connected, in which each layer must be connected to every node in the previous layer and every node in the next layer [8] [19] [20]. MLP is a supervised type and uses the backpropagation algorithm to train the model. The backpropagation algorithm works as follows: randomly initialize network weights; propagate the input through the network to get the output; calculate the error between the target output and the actual output; propagate the error back through the network; update the weights to reduce the error; repeat the steps (without initializing the weights again) until the error is as small as possible [19].

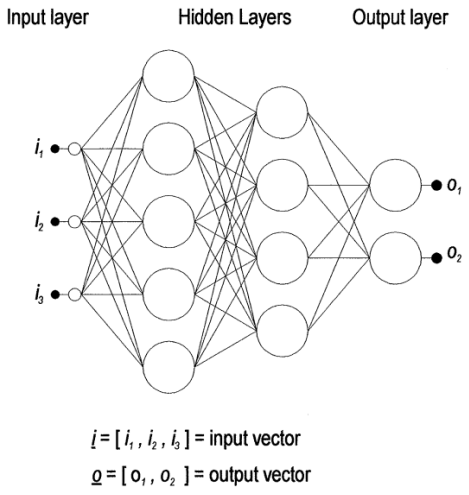

Figure 4 Multi-Layer Perceptron [19]

2.2.2.4. Gradient Boosting

Boosting is an effective technique introduced in the domain of machine learning and statistics for solving classification problems [21] [22]. The concept behind boosting is the combination of simple classifiers that a weak or base learner obtains to improve prediction performance instead of using a simple classifier alone. A weak learner is a learning algorithm that produces classifiers with better performance than random guessing [17]. Boosting is a type of supervised learning that aims to find an optimal function of variables to predict the output by reducing the loss function [21]. One of the earliest boosting algorithms suggested is AdaBoost, which was developed to solve binary classification problems and then extended to solve multi-class classification as well. AdaBoost is an algorithm of type gradient descent, thus stating boosting as a statistical estimation and numerical optimization. Gradient boosting is an extension of the AdaBoost algorithm utilizing a statistical framework [8] [23]. Gradient boosting uses decision trees as weak or base learners [23]. Figure 5 illustrates the GB [24].

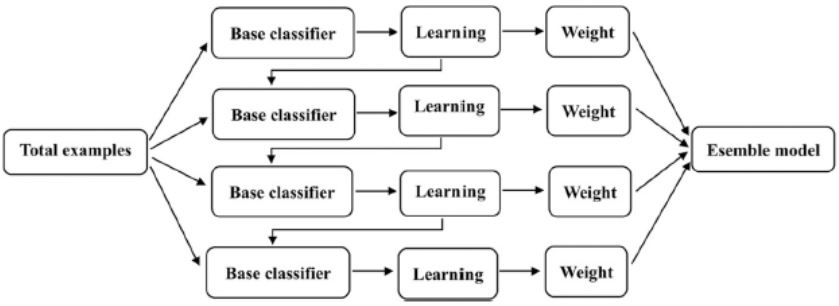

Figure 5 Gradient Boosting [24]

2.3. Predictive Data Analytics

Predictive data analytics is the technique of creating and utilizing models that make accurate predictions. Predictive data analytics analyses and extracts insights from current and previous data by using AI, ML, and statistics [1][11]. CRISP-DM is one of the most widely used methodologies for predictive data analytics. The CRISP-DM is preferred by data analytics practitioners because it is non-proprietary, explicitly includes both application-focused and technical perspectives in the data analytics process, and is neutral regarding application, industry, and tools [1][25]. Figure 6 illustrates the CRISP-DM. The phases of CRISP-DM are as follows:

- Business Understanding: addressing and determining business problems such as identifying the goal of building a prediction model and initializing and developing a plan.
- Data Understanding: understanding the available data sources and what they contain. In addition, addressing the data requirements such as the collection, description, exploration, and quality verification of the data.
- Data Preparation: preparing the data such as selecting the desired data, cleaning it, and organizing it into a specific structure.
- Modeling: building a predictive model by using different ML algorithms and selecting the best one.
- Evaluation: evaluating and testing the model to ensure that it fits the purpose (makes accurate predictions).
- Deployment: deploying and integrating the evaluated model into the process within an organization.

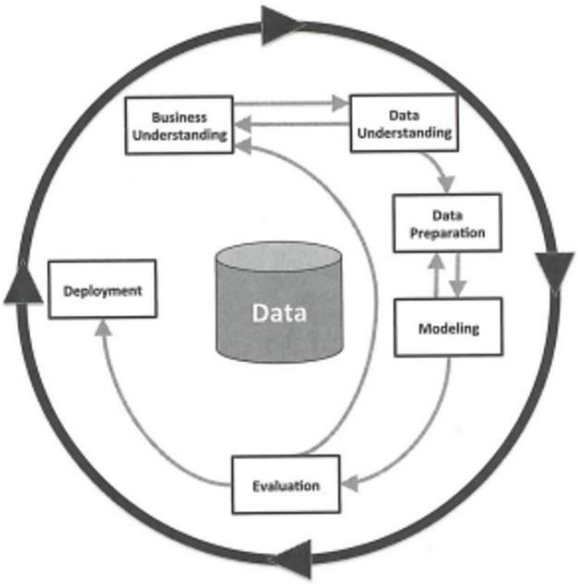

Figure 6 A diagram of the CRISP-DM [1]

2.4. Predictive Maintenance

Due to the increased availability of data, the industries started to apply a new approach called predictive maintenance to improve their products. This concept refers to the idea that each production component can inform when it will fail [26] [27]. According to a common estimate [3], downtime costs almost every company between 5% and 20% of its production, and maintenance costs can be enormous. By reducing downtime, industry can become more efficient [3]. In addition, industries can provide efficient solutions for managing the maintenance and improving quality by applying machine learning techniques. In order of rising complexity and efficiency, the maintenance management approaches have been categorized as follow:

- Run-to-failure (R2F): is reactive maintenance, which the maintenance is performed after the occurrence of failures and issues. This technique for maintenance is the easiest and the least efficient because the maintenance actions will be applied after the breakdown occurs and the cost will be higher than planned actions [26][27].
- Preventive maintenance (PvM): The maintenance is performed before the occurrence of failures by using a planned schedule that is "based on time or process iterations". Failures are usually prevented using this technique, also known as scheduled maintenance. This technique may increase costs by unnecessarily maintaining a component that still works and has not failed [26][27].
- Predicative maintenance (PdM): Also known as "condition-based maintenance" or "on-line monitoring" or "risk-based maintenance" [28]. The maintenance is performed before the occurrence of failures by utilizing and monitoring the health status of the equipment. PdM applies statistical and machine learning techniques to automatically collect and analyze equipment data to predict failures [26][27]. With the help of the integrated sensors, PdM can reduce breakdowns and costs, increase efficiency, address different issues related to equipment replacement, and identify the underlying cause of the failure [28]. Both PvM and PdM may act in the same way to schedule the maintenance activities before failure occurrences, but the PdM approach is based on data gathered from sensors and analysis algorithms [28].

2.4.1. Machine learning for PdM

ML-based PdM categorizes into two classes: supervised and unsupervised. In supervised class, the modeling dataset contains information about failure occurrences, while in unsupervised class, the modeling dataset contains information about the logistic and/or process but doesn't contain information about maintenance [27]. The availability of maintenance data depends on the type of existing maintenance management approach. However, in the run-to-fail approach, the supervised technique is suitable due to the availability of maintenance data such as the production process between two failure occurrences. On the other hand, in the preventive maintenance approach, the unsupervised technique is suitable because the failure maintenance is performed before actual failures occur, which leads to a lack of full availability of the maintenance data [27]. Due to the availability of datasets provided by NEP, the supervised approach is considered in this thesis. Two classes of supervised problems can exist depending on the type of outcome: a classification problem if the outcome is considered to have categorical values and a regression problem if the outcome is considered to have continuous values [27].

2.5. PdM based ML for ETS

The NEP is trying to apply the ML algorithms within ETS to develop a system that can monitor the electrical performance of the test fixtures during testing and identify any changes in the test results that may indicate a problem with the bed-of-nails pins. For example, if the electrical resistance of the pins increases over time, it may indicate that the pins are becoming worn or contaminated and may need to be cleaned or replaced. This thesis can provide the essential steps to achieve the proposed system since the limitations of this thesis are limited to only classification purposes. To fully achieve the proposed system, the thesis can be extended for regression purposes as well. Furthermore, this thesis presents suitable ML algorithms that can be applied. Moreover, this thesis can provide a tool that is ready to extract the features, tune the parameters, fit ML models, evaluate them and plot the results. The tool can predict and classify the status of new ETS tests as well.

3. Related Work

The increasing availability of data and the increasing capability of hardware have recently encouraged industries to apply ML approaches in maintenance management domains. The application of ML approaches in maintenance management has made PdM a popular topic in manufacturing research [28][2]. PdM based ML applications have been a potential tool in preventing and predicting equipment failures. The performance of PdM applications is tied to the selection of the ML algorithm [29]. The research presented by Franco and Figueiredo [26], develops a PdM system in a mobile edge computing architecture to combine the low cost of embedded systems and the advantages of security with the needs of PdM. The research presented shows that the developed system provides an accuracy of 100%, whereas the ML algorithm was RF. According to Dey et al. [30], the performance of classification tasks that work online will be required by intelligent embedded systems in the future. SVM is one of the different classification techniques that has received a lot of interest. SVM appears to be a better option than traditional neural networks and provides "remarkable results both in classification and regression applications" [30]. Additionally, Chigurupati, Thibaux and Lassar [14] illustrate and propose that the SVM is a crucial algorithm for achieving very accurate prediction. The study developed by Castanheira et al. [31] shows that MLP is used to classify the incipient errors in power transformers. The provided procedure was evaluated using data collected from chromatographic tests of the transformers. The accuracy of MLP is between 75 and 90%. Gkamas et al. [32] proposed a ML framework based on a MLP classifier that can solve classification problems, analyze big data in a relative amount of time, is suitable for low-resource embedded systems, and is a utility for crucial event detection and industrial maintenance. The proposed framework can even be trained online. According to Carvalho et al. [29], research suggests a fault detection system that detects real-time faults on the hard disk drive. The suggested system is divided into two phases: batch training, where RF is used to train models, and real-time prediction, where the estimations are performed using data collected from the end-user device. However, the accuracy of this system is 85%. Furthermore, Carvalho et al. argue that RF is the most used and compared machine learning algorithm in predictive maintenance applications and the key reason is that RF is based on decision trees, which generate a huge number of observations that can be used in prediction. According to Liulys [22], recent developments in statistical and computer science domains have led to the manufacturing industry focusing on increasing the data from industrial

repositories in a sustainable manner. An exciting development is machine learning, where the GB is one of the most popular machine learning algorithms for analyzing big datasets.

4. Problem Formulation

The increasing complexity of ETS has made it difficult for engineers to ensure their reliability and performance. To address this issue, ML has been proposed as a potential solution for predicative data analytics in ETS. However, the adoption of ML-based predictive analytics in ETS is limited due to the lack of a systematic approach to implement ML-based predictive analytics in ETS and the lack of relevant research. A PdM-based ML approach allows engineers to ensure the performance of the ETS to maintain and replace pins frequently to prevent and avoid unexpected pin failures. This helps in reducing costs and unexpected failures, thus increasing the efficiency and reliability of ETS. This work aims to develop a systematic approach to implementing ML-based predictive analytics in ETS that can facilitate NEP's engineers' ability to predict the potential failures of ETS.

4.1. Research questions

1. How to extract and select the key Features from ETS dataset?
2. How to implement ML model with optimize the parameters to predict ETS failures?
3. How to develop a framework to be evaluated using real-world ETS dataset?

4.2. Limitations

The limitation of this thesis is that the dataset used in the implementation is limited to one year of data collection, which may not be enough to study all possible ETS tests. Additionally, this thesis is limited to the classification and prediction of the ETS results.

5. Method

5.1. Research Methodology

The main objective of the thesis was to apply ML algorithms to real-world ETS data to perform predictive analysis on the data that could facilitate NEP's engineers' ability to predict the potential failures on ETS. However, the developed approach could extract the features, prepare them for modeling, tunning the parameters and train the models and evaluate them. An experimental approach was carried out to address the research questions. The methodological steps were applied as follows:

- A brief understanding of the concepts, technologies, and processes of both ETS and PCBA was made such as understanding the test fixture, electronic components, test objects, and interactions between them.
- A literature study was carried out to review the existing literature and research in the domains of predictive data analytics, predictive maintenance, and embedded systems as well.
- A current state-of-the-art study within the machine learning domain was also carried out to identify ML algorithms that fit the application.
- An experimental implementation of the work was conducted as follows:
    - Collecting and preparing the data.
    - Medeling different ML-algorithms.
    - Evaluating and testing the performance of the built models.
- The conclusion was drawn and discussed.

6. Ethical and Societal Considerations

There are ethical considerations to consider while conducting the thesis. The data that is provided by NEP is confidential and considered to be used exclusively at Mälardalen University campuses and only for research purposes. A non-disclosure agreement has been signed with the NEP, which includes the

data sharing agreement and publication regulation. The data will be deleted as soon as the work is finished.

7. Experimental study and implementation

The experimental work was carried out following the CRISP-DM approach [1]. Figure 7 illustrates the core implementation of the work. The implementation consists of three phases: the data preparation phase, the modeling phase, and the evaluation phase. In the data preparation phase, feature extraction and normalization of the features have been carried out; more details provided in sections 7.1, 7.2, and 7.3. In the modeling phase, the steps are carried out as follows: 1. splitting the data set into an 80 percent train dataset and a 20 percent test dataset; 2. utilizing the train dataset to train the model; 3. selecting the type of classifier whether a binary classifier or a multi-class classifier; 4. selecting the classifier, such as SVM, RF, MLP, or GB; 5. selecting whether the dataset should be imbalanced or balanced. If the selected dataset is balanced, then resampling and cross validating the features, otherwise training the model with an imbalanced dataset. More detail about the modeling phase is described in section 7.4. In the evaluating phase, the evaluation of the trained model is carried out by using the test dataset to predict the outcomes and comparing them with the actual outcomes. More details are described in section 7.5.

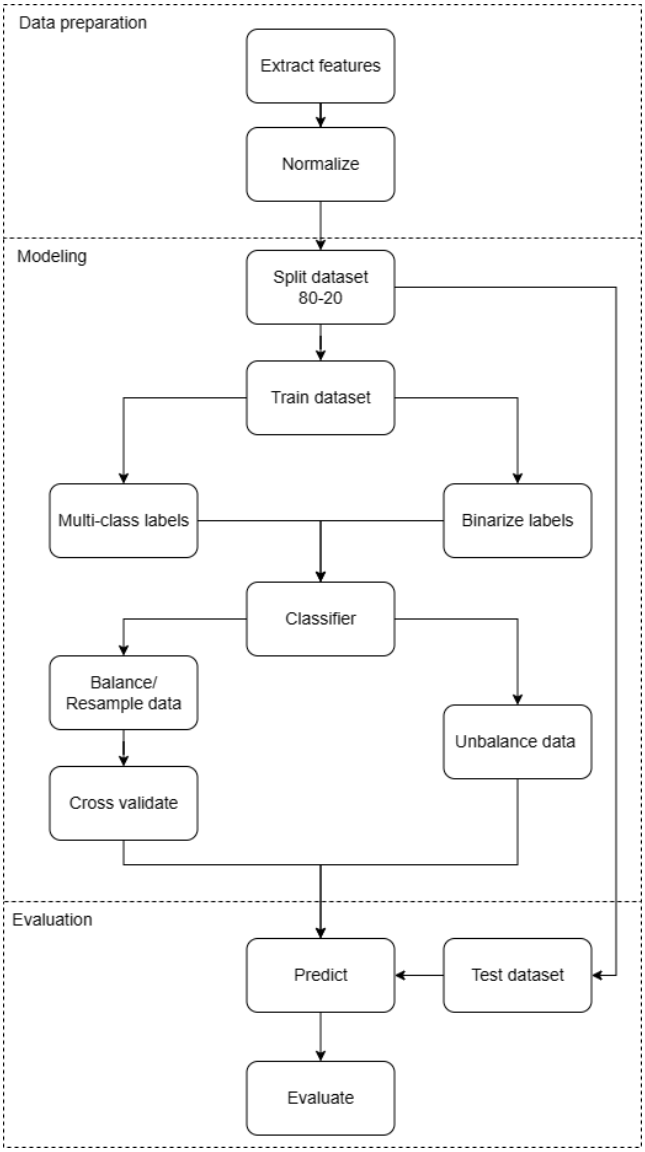

Figure 7 Experimental flow diagram

7.1. Data collecting

The dataset gathered from NEP and compiled from ETS tests that contains details about the EOL tests of PCBA. The dataset consists of two csv files: test result and measurement result. The test result provides information on complete tests, namely, each record in the test file is a complete test sequence of an EOL test of a single PCBA, while the measurement test provides in detail how a test sequence is processed. As shown in Tables 1 and 2, the test result has 4 features and contains 42 992 records, while the measurement has 11 features and contains 1 473 741 records.

Table 1 Details of test result features

Header	*Description*
Test result ID	The ID of a test sequence
Timestamp	The execution time and data for each sequence
Execution time	The execution time of a test sequence
status	The status of a test (passed, failed, error, terminated)

Table 2 Details of measurement result features

Header	*Description*
Measurement result ID	The measurement ID for each test in a test sequence
Test result ID	The ID of a test sequence
Name	The name of each test in a test sequence
Timestamp	The execution time and data for each test in a test sequence
Execution time	The execution time of a test sequence
Numeric time	The value of a measurement test
Numeric low limit	The lowest value accepted
Numeric high limit	The highest value accepted
Comparator	The comparator of a test
Unit	The unit of the measured test
Status	The status of a test (passed, failed, error, terminated)

7.2. Data preparing

Data preparation and feature extraction were carried out using the .NET[4] framework and the LINQ[5] library. .NET is a free open-source cross-platform to build modern applications. LINQ stands for language-integrated query and refers to a set of technologies based on the direct integration of query capabilities into the C# language. With LINQ, the dataset can be easily filtered, ordered, and grouped with a minimum of code. The code for feature extraction is provided in Appendix A and on GitHub[6] as well. The dataset was prepared into a format that could be used for training models. At the beginning, the two files have been merged into one file by including all features from the measurement file and replacing the status feature from the test file based on the test ID. Then a new dataset (format) was extracted, as shown in Table 3, some features were taken from the measurement file, and new features were added as follows:

- Grouping the tests where they had the same test ID.
- Calculating the minimum, maximum, average, median, and standard deviation of each execution time, numeric data, numeric low limit, and numeric high limit.
- Counting the frequency of comparators and units. The comparator and unit have different abbreviations, some of them are GELE, EQ, GTLT, LOG, V, ÂµA, and NULL.
- Counting the frequency of test names and measurement IDs.
- The missing values (NaN) have been replaced in the dataset.

[4] https://dotnet.microsoft.com/en-us/
[5] https://learn.microsoft.com/en-us/dotnet/csharp/linq/
[6] https://github.com/FayadHanash/FeatureExtractionETS

Table 3 Details of extracted features

Header	*Description*
TestResultID	The ID of a test sequence
Execution Time Total	The execution time for a test sequence
Execution Time Min	The minimum execution time for a test sequence
Execution Time Max	The maximum execution time for a test sequence
Execution Time Avg	The average execution time for a test sequence
Execution Time Median	The median execution time for a test sequence
Execution Time StdDev	The standard deviation execution time for a test sequence
Numeric Data Min	The minimum numeric data for a test sequence
Numeric Data Max	The maximum numeric data for a test sequence
Numeric Data Avg	The average numeric data for a test sequence
Numeric Data Median	The median numeric data for a test sequence
Numeric Data StdDev	The standard deviation numeric data for a test sequence
Numeric Data Low Limit Min	The minimum numeric data low limit for a test sequence
Numeric Data Low Limit Max	The maximum numeric data low limit for a test sequence
Numeric Data Low Limit Avg	The average numeric data low limit for a test sequence
Numeric Data Low Limit Median	The median numeric data low limit for a test sequence
Numeric Data Low Limit StdDev	The standard deviation numeric data low limit for a test sequence
Numeric Data High Limit Min	The minimum numeric data high limit for a test sequence
Numeric Data High Limit Max	The maximum numeric data high limit for a test sequence
Numeric Data High Limit Avg	The average numeric data high limit for a test sequence
Numeric Data High Limit Median	The median numeric data high limit for a test sequence
Numeric Data High Limit StdDev	The standard deviation numeric data high limit for a test sequence
Name Frequency	The frequency of the name for each test within a test sequence
Measurement ID Frequency	The frequency of the measurement ID for each test within a test sequence
GELE Frequency	The frequency of GELE for a test sequence
EQ Frequency	The frequency of EOF for a test sequence
LOG Frequency	The frequency of LOG for a test sequence
GTLT Frequency	The frequency of GTLT for a test sequence
V Frequency	The frequency of V for a test sequence
ÂµA Frequency	The frequency of ÂµA for a test sequence
NULL Frequency	The frequency of NULL for a test sequence
Status	The status for a test sequence

After exploring all extracted features, the most varied features were selected to train the model. The feature variety improves the performance of ML models and avoids data overfitting. As shown in Table 4, the selected features are the total, maximum, average, and standard deviation of the execution time and the minimum, average, median, and standard deviation of the numeric data. In addition, the status feature is selected since it is the target feature. The name of each test in a test sequence is not considered while only its frequency is considered, due to the fact that the names are the same and frequently repeated in each sequence, which does not make any sense for modeling's purpose to have them in the dataset. Furthermore, the measurement ID is not considered in the dataset since the measurement ID is frequently increased in each test. Moreover, the timestamp feature for each test in a sequence test is not considered because the execution time total already represent the total time for a test sequence. Finally, all other features that are presented in Table 3 and are not presented in Table 4 have been ignored since they do not vary and are almost the same in each test sequence. The dataset has been divided into two sets: one for the training phase, which includes 80% of records, and the second for the testing phase, which includes 20% of records.

Table 4 The selected features

Header	*Description*
Execution Time Total	The execution time for a test sequence
Execution Time Max	The maximum execution time for a test sequence
Execution Time Avg	The average execution time for a test sequence
Execution Time StdDev	The standard deviation execution time for a test sequence
Numeric Data Min	The minimum numeric data for a test sequence
Numeric Data Avg	The average numeric data for a test sequence
Numeric Data Median	The median numeric data for a test sequence
Numeric Data StdDev	The standard deviation numeric data for a test sequence
Status	The status for a test sequence

7.3. Normalization

Normalization is a technique used to scale the values in a given range. It is helpful for classification and predication purposes and facilitates the modeling process for machine learning algorithms. Min-Max normalization, also known as range normalization, is a normalization technique that fits the original data into a specific range. The Min-Max normalization can be calculated by using the following equation:

$$a' = \frac{a_i - \min(a)}{\max(a) - \min(a)} * (high - low) + low \quad \textbf{(1)}$$

Where a' is the normalized value, a is the actual value, min is the minimum value of a, max is the maximum value of a, low and high are the boundaries of the desired range. [1] [33]. In the extracted features, the feature ranges were extremely high. The normalization phase was carried out using the Min-Max technique, and the features were scaled into the range [0,1].

7.4. Modeling

The modeling implementation was carried out using Python and libraries such as Scikit[7], Imbalanced-learn[8], Matplotlib[9], NumPy[10], Pandas[11] and Seaborn[12]. Scikit is a comprehensive library for predictive data analysis and machine learning. It provides a wide range of tools and algorithms for classification, model evaluation, etc. Imbalanced-learn is a library that provides re-sampling techniques. Matplotlib is a plotting and visualization library. Pandas is a library for data analysis and manipulation. Seaborn is a visualization library based on Matplotlib. The code for normalization, modeling, and evaluating is provided in Appendix B and on GitHub[13] as well. The final dataset contains 42 844 records, and after counting insight into the target set, the class that is "passed" has 39855 records, the class that is "failed" has 378, the class that is "error" has 2518 records, and the class that is "terminated" has 93 records. Since the class "passed" is significantly higher than the other classes, the modeling phase was trained in two different phases: the binary classification phase and the multi-class classification phase. In the binary classification phase, the target set has been binarized into two classes: the "passed" class and the "rest" class, while in the multi-class classification, all the classes have been kept. As illustrated in Figure 7, the developed framework allows to select either binary classification or multi-class classification, then select a classifier, and after that, select whether the dataset should be imbalanced or balanced. If the selected dataset is balanced, then the features will be resampled and cross-validated. Then, the modeling process continues fitting until it is done and evaluates the performance of the modeled classifier. Parameter

[7] https://scikit-learn.org/
[8] https://imbalanced-learn.org/
[9] https://matplotlib.org/
[10] https://numpy.org/
[11] https://pandas.pydata.org/
[12] https://seaborn.pydata.org/
[13] https://github.com/FayadHanash/MachineLearningBasedPredictiveDataAnalytics

tunning is carried out using the GridSearchCV[14] approach, which is a method provided by Scikit that allows to search for the best parameters that fit the model.

7.5. Evaluation

In the evaluation phase, the combination of cross validation, sampling techniques, and confusion matrix were applied to the dataset for determining the classifier performance.

7.5.1. Cross Validation

Cross validation (CV) is a way to ensure how a model performs on unseen data. It is a procedure for evaluating robust models. The repeated stratified k-fold cross validation has been used to validate the balanced dataset. The repeated stratified k-fold cross validation is a version of k-fold cross validation that uses stratified random sampling to create the folds. The dataset was split into five folds and repeated three times; at each repeat, the training set was split into one testing fold and four training folds. This method can yield correct values if it is applied efficiently [34].

7.5.2. Data balancing

A dataset is imbalanced where the distribution of classes is significantly skewed, which leads to the classification of the majority classes being more accurate better than the minority classes or the minority classes being ignored at all. A way to address this issue is by re-sampling the dataset. Over sampling technique duplicates the records in the minority classes and replicates them in the dataset. Under sampling technique delates the records from the majority classes. Synthetic Minority Oversampling Technique (SMOTE) is a technique that creates synthetic examples to over-sample the minority classes [35]. By combining the SOMTE and the under-sample, a classifier can perform better [35]. In this experiment, the data balancing was carried out by pipelining the SMOTE and Random Under Sampler techniques. Figure 8 shows the distribution of the classes before and after balancing the dataset in both binary and multi-class cases.

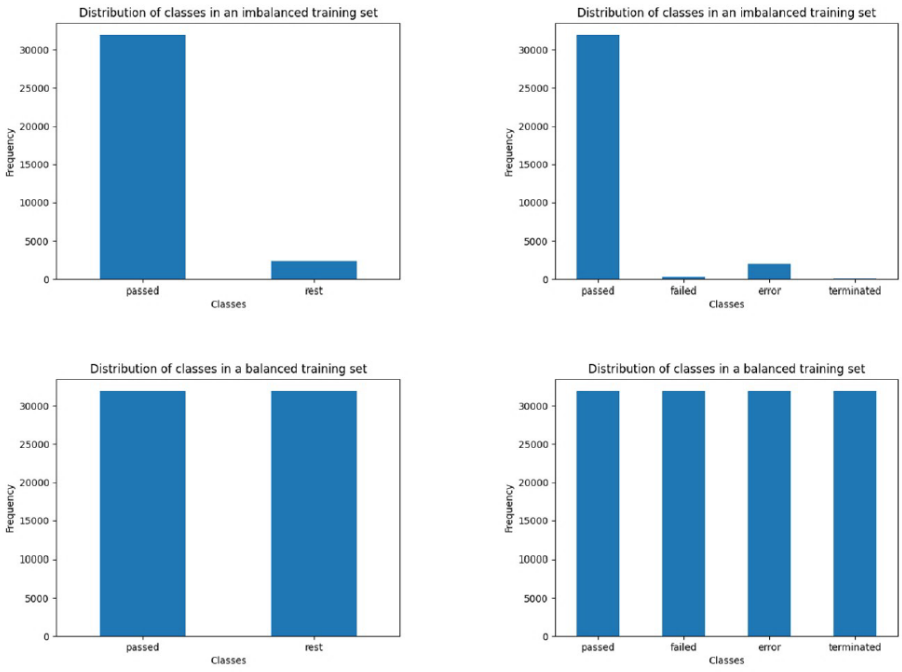

Figure 8 Distribution of the classes

[14] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

7.5.3. Performance Measures

A confusion matrix (CM) was carried out to evaluate the performance of the algorithms. The CM as shown in Table 5 is a matrix containing the predicted class as columns and the actual class as rows.

Table 5 Confusion matrix

	Predicted Negative	*Predicted Positive*
Actual Negative	True Negatives	False Positives
Actual Positive	False Negatives	True Positives

True Negatives (TN): represent the number of correctly classified negative examples. False Positives (FP): represent the number of falsely classified positive examples. False Negatives (FN): represent the number of falsely classified negative examples. True Positives (TP): represent the number of correctly classified positive examples [35]. Accuracy represents the proportion of correct prediction examples of the total prediction examples and measures as the ratio between sum of correct classification and total number of classifications. The accuracy can be calculated by using the following equation:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad \textbf{(2)}$$

Recall represents the proportion of correctly predicted positive examples of all actual positive examples and measures as the ratio between of the true positive examples and the sum of true positive and false negative examples. The recall can be calculated by using the following equation:

$$Recall = \frac{TP}{TP + FN} \quad \textbf{(3)}$$

Precision represents the proportion of correctly predicted positive examples of all predicted positive examples and measures as the ratio of the true positive examples and the sum of true positive and false negative examples. The precision can be calculated by using the following equation:

$$Precision = \frac{TP}{TP + FP} \quad \textbf{(4)}$$

F1-score represents the harmonic mean of recall and precision. The f1-score can be calculated by using the following equation:

$$F1 = \frac{2 * precision * recall}{precision + recall} \quad \textbf{(5)}$$

The Kappa coefficient estimates the degree of agreement between a pair of raters. The kappa, also known as Cohen's kappa, was introduced by Jacob Cohen. The Kappa can be used as performance measure for both binary and multi-class classification. The value of kappa is a range between [-1, +1], where -1 indicates to wrong predication and +1 perfect predication [36]. The kappa can be calculated by using the following equation:

$$Kappa = \frac{2 * (TP * TN - FP * FN)}{(TP + FP) * (FP + TN) + (TP + FN) * (FN + TN)} \quad \textbf{(6)}$$

Matthews Correlation Coefficient (MCC), also known as the phi (ϕ) coefficient in 2 * 2 confusion matrices [36]. MCC can be used to evaluate the performance of binary and multi-class classifications. The value of MCC is a range between [-1, +1], where -1 indicates to wrong predication and +1 perfect predication [36]. The MCC can be calculated by using the following equation:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN * FN)}} \quad \textbf{(7)}$$

The Receiver Operating Characteristic curve (ROC curve) is a plot that separates two classes in a binary classifier, such as a form of threshold. The ROC curve summarizes the classification performance between the true positive rate (TPR) and false positive rate (FPR) [35]. TPR and FPR are used to plot the ROC curve. The area under the ROC curve, often known as the ROC AUC, is a metric provided by the ROC curve, and it is a measurement that is considered the most essential property of the classifier. The larger the area, the better the performance of the classifier. TPR represents the same thing as Recall, also known as Sensitivity. FPR, also known as Specificity, represents the proportion of incorrectly predicted positive examples of all actual negative examples and measures as the ratio of the false positive examples and the sum of false positive and true negative examples [8]. The TPR and FPR can be calculated by using the following equations:

$$TPR = \frac{TP}{TP + FN} \quad \textbf{(8)}$$

$$FPR = \frac{FP}{FP + TN} \quad \textbf{(9)}$$

8. Results

The result of the experimental work is divided into two sections: binary classification and multi-class classification, and each section is divided into two subsections as well: imbalanced dataset and balanced dataset. The measurements that are used in the binary classifications to perform the evaluation are accuracy, precision, recall, f1-score, kappa, mcc, roc auc. In contrast, accuracy, precision, recall, and f1-score are used for the multi-class classification. Furthermore, the results of the CV are estimated only for the balanced dataset.

8.1. Binary Classification

8.1.1. Imbalanced dataset

Figure 9 presents the CM of the SVM, RF, MLP, and GB classifiers, where the dataset was unbalanced and binarized to ‘pass’ and ‘rest’ classes. Table 6 shows the measurement as follows: The accuracy and precision of the classifiers are almost 100% where the data is unbalanced. The recall, F1-score, kappa, and MCC measurements are highest for the RF classifier and lowest for the SVM classifier. The ROC AUC measurement is highest for RF and GB at 100%, as plotted in Figure 10.

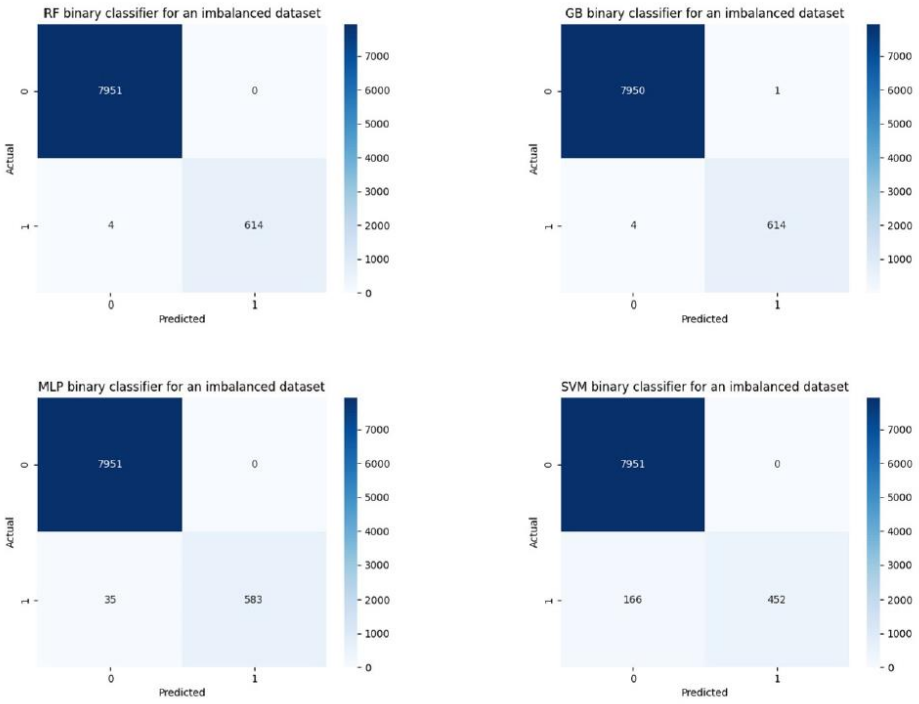

Figure 9 Confusion matrix of the binary classifiers for the unbalanced dataset

Table 6 Accuracy, Precision, Recall, F1, Kappa, MCC, and ROC_AUC measurements of the binary classifiers for the unbalanced dataset

Classifier Algorithm	*Accuracy*	*Precision*	*Recall*	*F1-Score*	*Kappa*	*MCC*	*ROC_AUC*
SVM	98.06	1	0.73	0.84	0.83	0.85	0.97
RF	99.95	1	0.99	0.98	0.98	0.98	1
MLP	99.59	1	0.94	0.97	0.97	0.97	0.98
GB	99.94	0.998	0.994	0.996	0.996	0.996	1

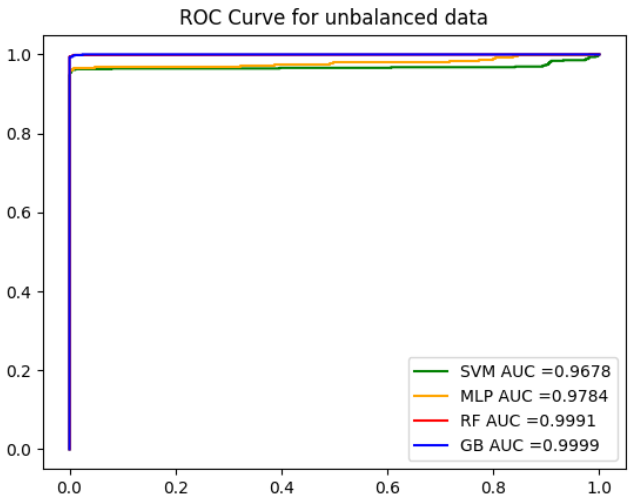

Figure 10 Roc curve for the binary classifiers (unbalanced)

8.1.2. Balanced dataset

Figure 11 presents the CM of the SVM, RF, MLP, and GB classifiers, where the dataset was balanced and binarized to 'pass' and 'rest' classes. Table 8 shows that almost all the measurements are equal. The ROC AUC measurement is highest for RF and GB at 100%, as plotted in Figure 12. Table 7 shows the average of the CV scores that are in the range [97%, 100%], and the best classifier was RF under training the model.

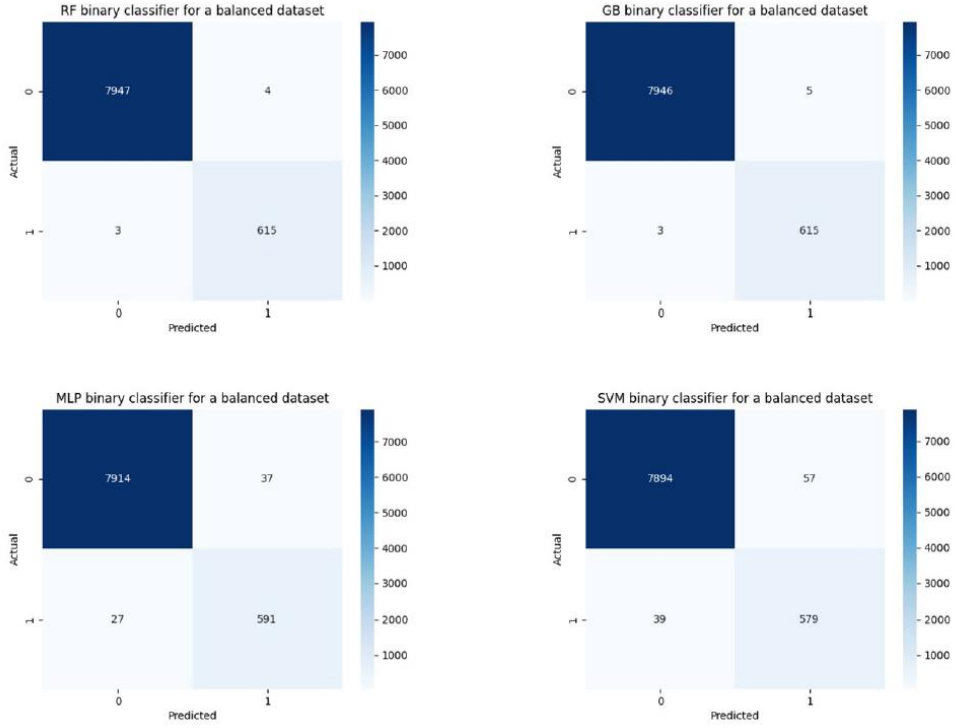

Figure 11 Confusion matrix of the binary classifiers for the balanced dataset

Table 7 Cross validation scores of the binary classifiers

Classifier Algorithm	*Cross validation score*
SVM	97.07
RF	99.94
MLP	98.44
GB	99.90

Table 8 Accuracy, Precision, Recall, F1, Kappa, MCC, ROC_AUC measurements of the binary classifiers for the balanced dataset

Classifier Algorithm	*Accuracy*	*Precision*	*Recall*	*F1-Score*	*Kappa*	*MCC*	*ROC_AUC*
SVM	98.88	0.91	0.937	0.923	0.917	0.918	0.971
RF	99.92	0.994	0.995	0.994	0.994	0.994	1
MLP	99.25	0.94	0.956	0.949	0.945	0.945	0.998
GB	99.91	0.992	0.995	0.994	0.993	0.993	1

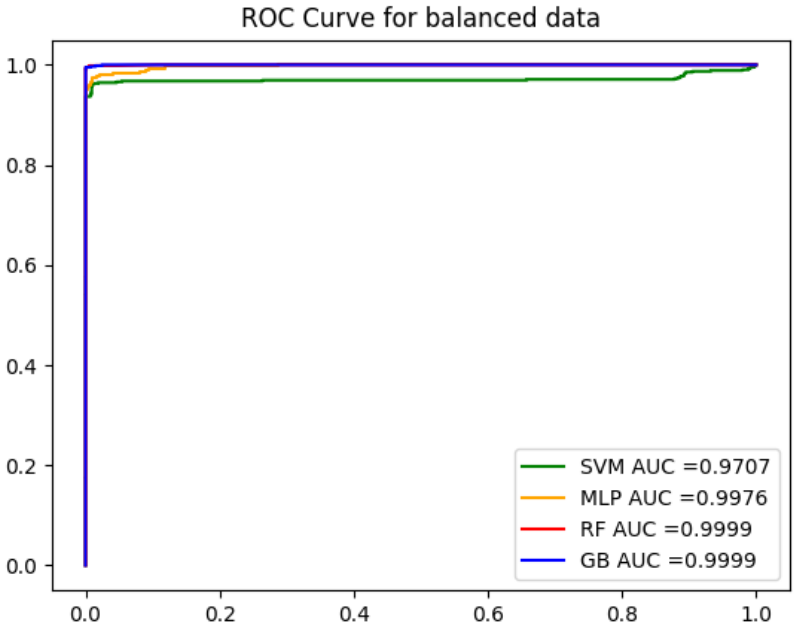

Figure 12 Roc curve for the binary classifiers (balanced)

8.2. Multi-class Classification

8.2.1. Imbalanced dataset

Figure 13 presents the CM of the SVM, RF, MLP, and GB classifiers where the dataset was unbalanced. The f1-score is focused here because the f1-score is between recall and precision. Tables 9, 10, 11, and 12 show the accuracy, precision, recall, and f1-score of the SVM, RF, MLP, and GB classifiers, respectively. Table 9 shows that the SVM classifier classified the 'passed' class almost correctly while failing to classify the 'terminated' class, which classified one sample correctly of the total terminated 15 samples. The 'failed' and 'error' classes were classified with 0.7 and 0.9, respectively. Tables 10, 11, and 12 show that the RF, MLP, and GB classifiers classified almost all the classes correctly.

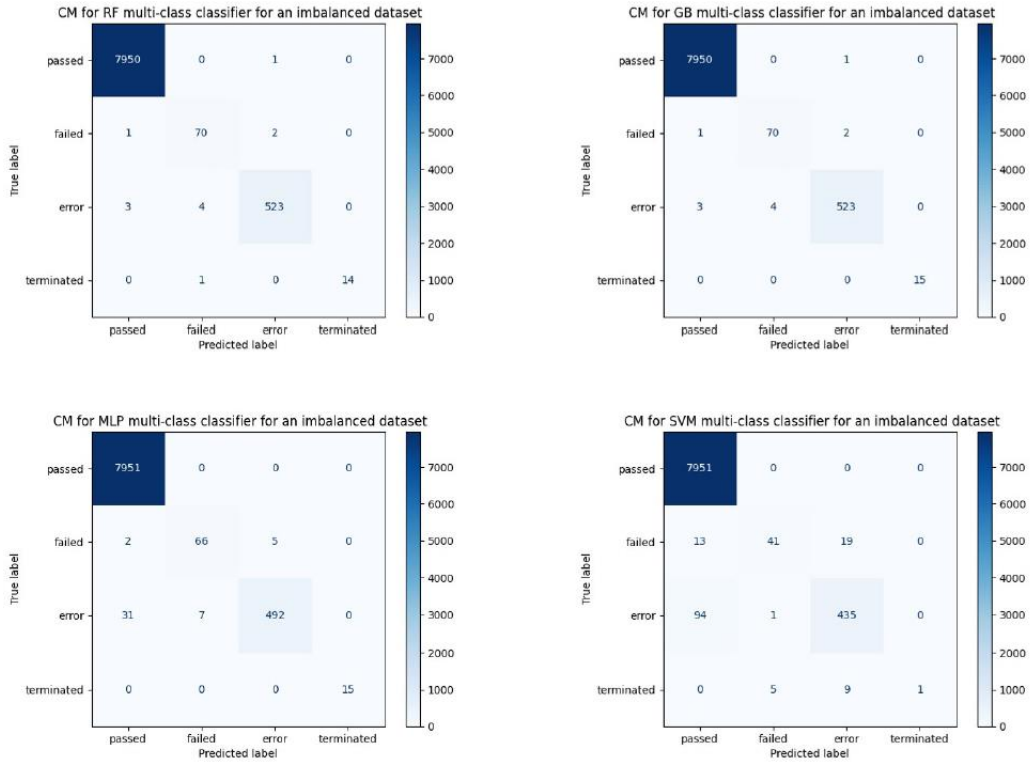

Figure 13 Confusion matrix of the multi-class classifiers for the unbalanced dataset

Table 9 Accuracy, Precision, Recall, and F1 measurements of the multi-class SVM for the unbalanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	98.75	0.987	1	0.993
Failed	99.56	0.872	0.562	0.683
Error	98.56	0.94	0.82	0.876
Terminated	99.84	1	0.0667	0.125

Table 10 Accuracy, Precision, Recall, and F1 measurements of the multi-class RF for the unbalanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	99.94	0.999	1	1
Failed	99.91	0.93	0.959	0.946
Error	99.88	0.994	0.987	0.991
Terminated	99.99	1	0.933	0.966

Table 11 Accuracy, Precision, Recall, and F1 measurements of the multi-class MLP for the unbalanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	99.61	0.996	1	0.998
Failed	99.84	0.904	0.904	0.904
Error	99.50	0.99	0.928	0.958
Terminated	100	1	1	1

Table 12 Accuracy, Precision, Recall, and F1 measurements of the multi-class GB for the unbalanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	99.94	0.999	1	1
Failed	99.92	0.946	0.959	0.952
Error	99.88	0.994	0.987	0.991
Terminated	100	1	1	1

8.2.2. Balanced dataset

Figure 14 presents the CM of the SVM, RF, MLP, and GB classifiers where the dataset was balanced. The f1-score is focused here also. Tables 14, 15, 16, and 17 show the accuracy, precision, recall, and f1-score of the SVM, RF, MLP, and GB classifiers, respectively. Table 15 shows the improvement of the SVM classifier with balanced data over unbalanced data, which this time misclassified only one sample of the 'terminated' class. While still the RF, MLP, and GB correctly classified all the classes, with a nuance improvement compared to the unbalanced data, as shown in tables 15, 16, and 17, respectively. Table 13 shows the average of the CV scores that are in the range [95%, 100%], and the best classifier was RF under training the model.

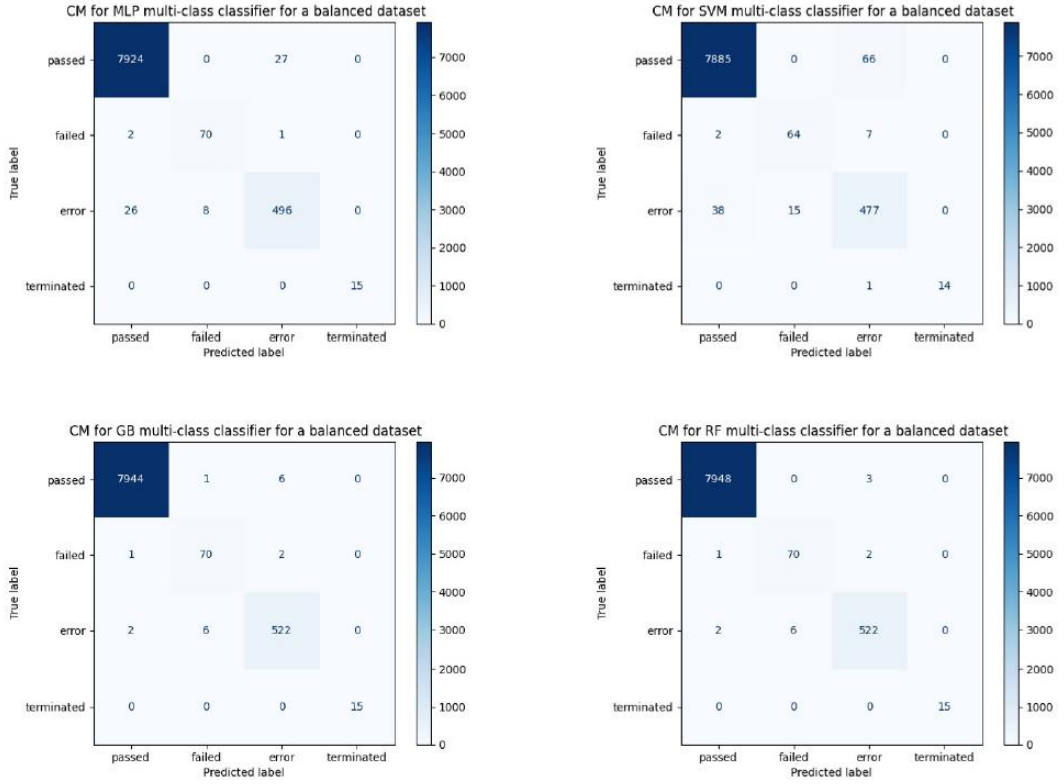

Figure 14 Confusion matrix of the multi-class classifiers for the balanced dataset

Table 13 Cross validation scores of the multi-class classifiers

Classifier Algorithm	*Cross validation score*
SVM	94.63
RF	99.97
MLP	98.81
GB	99.93

Table 14 Accuracy, Precision, Recall, and F1 measurements of the multi-class SVM for the balanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	98.76	0.995	0.992	0.993
Failed	99.72	0.81	0.877	0.842
Error	98.52	0.866	0.9	0.883
Terminated	99.99	1	0.93	0.966

Table 15 Accuracy, Precision, Recall, and F1 measurements of the multi-class RF for the balanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	99.93	1	1	1
Failed	99.89	0.921	0.959	0.94
Error	99.85	0.991	0.985	0.988
Terminated	100	1	1	1

Table 16 Accuracy, Precision, Recall, and F1 measurements of the multi-class MLP for the balanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	99.36	0.996	0.997	0.997
Failed	99.87	0.897	0.959	0.927
Error	99.28	0.947	0.936	0.941
Terminated	1	1	1	1

Table 17 Accuracy, Precision, Recall, and F1 measurements of the multi-class GB for the balanced dataset

Class	*Accuracy*	*Precision*	*Recall*	*F1-Score*
Passed	99.88	1	0.999	0.999
Failed	99.88	0.909	0.959	0.933
Error	99.81	0.985	0.985	0.985
Terminated	100	1	1	1

9. Discussion

The measurement of the experiment is very high and accurate, which indicates doubts about it being unbelievable and not trustable. The SVM seems to be more acceptable than the RF, MLP, and GB, even though the RF, MLP, and GB are perfect. In contrast, it is also good to mention that when training the SVM, where the regularization factor was set to 10, the training phase was done for the binary SVM, where the data was balanced and unbalanced, and for the multi-class SVM, where the data was only unbalanced. The modeling time for multi-class SVM where the data was balanced, has taken 5 days without finishing the modeling phase, which leads to remodeling the SVM, but this time the regularization factor was set to 5 and 4 for binary SVM and multi-class SVM, respectively. The measurements reflect that the training is being overfitted despite the normalization and sampling techniques. One of the reasons why the measurements are high is the data itself, because the distribution of the features is not varied and the classes in each feature are discriminated, which leads to the classifier's bias. Figure 15 presents the distribution of the scaled features in the training set. Figure 16 shows the correlation matrix of features in the training set. The matrix shows that feature 7 is highly correlated to feature 5 with a value of 0.93 and to feature 1 with a value of 0.81. Feature 3 is highly

correlated to features 1 and 2, with values 0.82 and 0.82, respectively. In addition, as early mentioned in the limitation section, that the dataset is only for one-year collection, which means that after exploring the dataset and reviewing the document provided by NEP, not all the components are included in the dataset, especially the comparator and unit components. Regarding the research questions:

1. How to extract and select the key Features from ETS dataset?
   The answer to this question is explained in the data collection and preparation sections, section 7.1 and 7.2, respectively.
2. How to implement ML model with optimize the parameters to predict ETS failures?
   The answer to this question is explained in the modeling section, section 7.4.
3. How to develop a framework to be evaluated using real-world ETS dataset?
   The answer to this question is explained in the evaluation section, section 7.5.

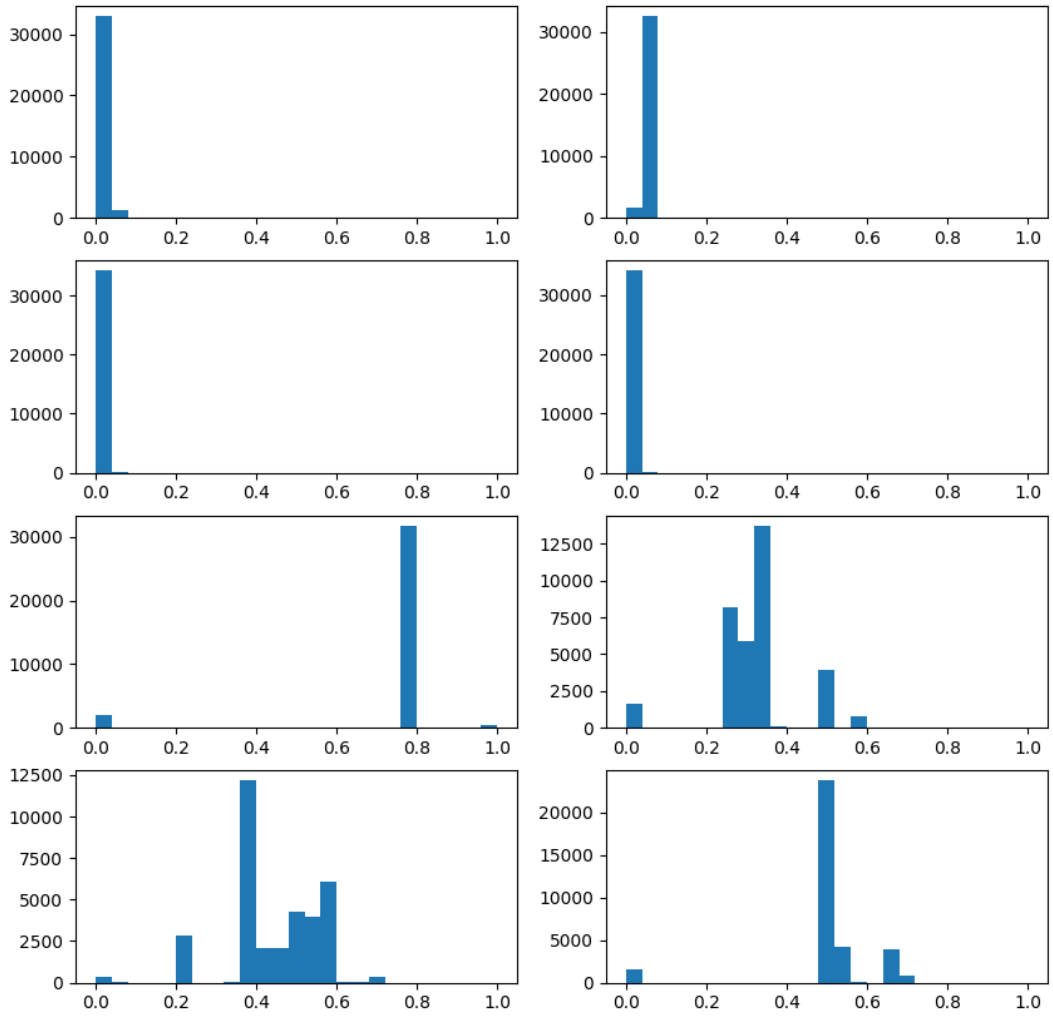

Figure 15 Distribution of the scaled features in the training set.

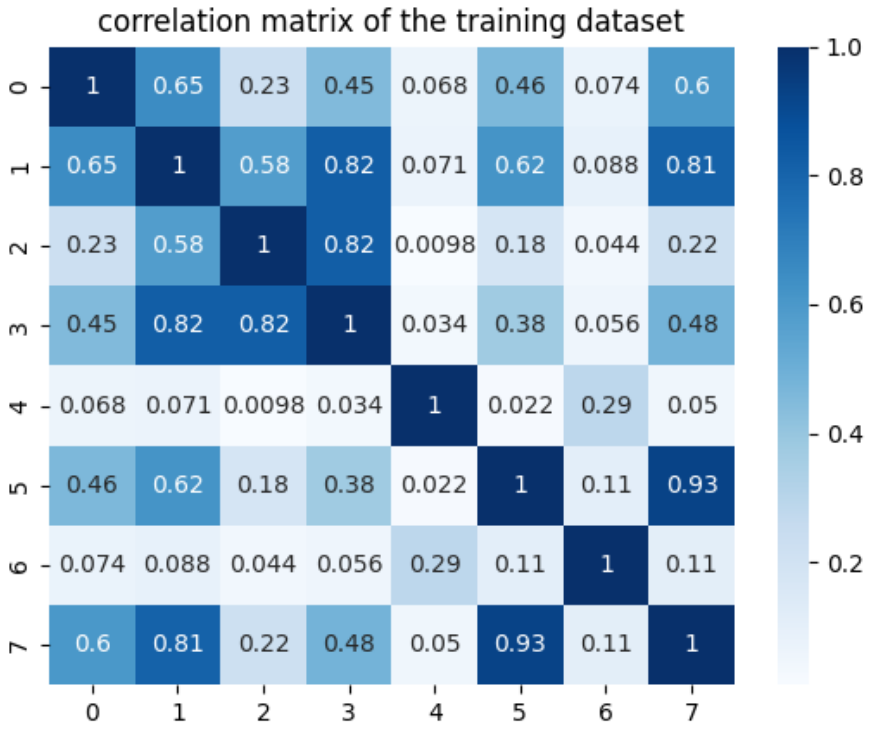

Figure 16 Correlation matrix of the training dataset

10. Conclusions

In this thesis, supervised machine learning algorithms were used to predict and classify the data that was collected from embedded test system. The machine learning algorithms were support vector machines, multi-layer perceptrons, random forests, and gradient boosting. Predicative maintenance is often used in manufacturing industries to predict failures before they occur. Predictive maintenance increases productivity and efficiency and decreases system failures and unplanned downtimes. Predictive data analytics is a crucial subfield within data analytics that make accurate predictions. Predictive data analytics extracts insights from data by using machine learning algorithms. The use of machine learning for predictive data analytics faces several challenges, such as extracting and selecting the features from the database and fitting and developing the models to predict the failure of ETS. The CRISP-DM was used in the experiment. Both binary and multi-class classifiers have been provided to fit the models, and cross-validation, sampling techniques, and a confusion matrix have been provided to accurately measure their performance. In addition to accuracy, recall, precision, f1, kappa, mcc, and roc auc measurements are also used. The random forest and gradient boosting performed best in all cases in this experiment. The trained models were supposed to be overfitted and biased, despite all the sampling techniques and normalization.

11. Future Work

In this experiment, only one single classifier is used to train the model. For future work, multiple different classifiers can be used at the same time to train the model by using the model ensemble technique, such as boosting or bagging techniques. The model ensemble technique is a technique that allows to combine a set of classification algorithms for building the models and then aggregates the prediction results of these models. Furthermore, the experimental work can also be extended to regression rather than just classification tasks.

References

[1] J. D. Kelleher, B. M. Namee & A. D'Arcy, *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. Cambridge, MA, England: The MIT Press, 2015.
[2] G. A. Susto, J. Wan, S. Pampuri, M. Zanon, A. B. Johnston, P. G. O'Hara & S. McLoone, *An Adaptive Machine Learning Decision System for Flexible Predictive Maintenance*. 2014 IEEE International Conference on Automation Science and Engineering (CASE). New Taipei. Taiwan. 2014. pp. 806-811. [online]. Available: https://doi.org/10.1109/CoASE.2014.6899418. Accessed: September. 27, 2023.
[3] A. Kane, A. Kore, A. Khandale, S. Nigade & P. P. Joshi, *Predictive Maintenance using Machine Learning*. arXiv.org. 2022. [online]. Available: https://doi.org/10.48550/arXiv.2205.09402. Accessed: September. 27, 2023.
[4] J. Dalzochio, R. Kunst, E. Pignaton, A. Binotto, S. Sanyal, J. Favilla & J. Barbosa, *Machine learning and reasoning for predictive maintenance in industry 4.0: Current status and challenges*. Elsevier. 2020. [online]. Available: https://doi.org/10.1016/j.compind.2020.103298. Accessed: September. 27, 2023.
[5] A. J. Albee, *The Evolution of ICT: PCB Technologies, Test Philosophies, and Manufacturing Business Models Are Driving In-Circuit Test Evolution and Innovations*. IPC APEX EXPO. Conf. 2013. Available: https://www.circuitinsight.com/pdf/evolution_ict_changing_pcb_technologies_ipc.pdf. Accessed: September. 27, 2023.
[6] V. Hirsch, P. Reimann, O. Kirn & B Mitschang, *Analytical Approach to Support Fault Diagnosis and Quality Control in End-Of-Line Testing*. Proceda CIRP. 2018. [online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827118301240. Accessed: September. 27, 2023.
[7] A. Maria & K. Srihari, *A Review of Knowledge-Based Systems in Printed Circuit Board Assembly*. The international journal of advanced manufacturing technology. 1992. [online]. Available: https://link.springer.com/article/10.1007/BF03500680. Accessed: September. 27, 2023.
[8] A. V. Joshi, *Machine Learning and Artificial Intelligence*. Second Edition Cham. Switzerland: Springer. 2023. [online]. Available: https://doi.org/10.1007/978-3-031-12282-8 Accessed: September. 27, 2023.
[9] K. Malhotra & Y. Kumar, *Challenges to implement Machine Learning in Embedded Systems*. 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN). Greater Noida. India. 2020. pp. 477-481. [online]. Available: https://doi.org/10.1109/ICACCCN51052.2020.9362893. Accessed: September. 27, 2023.
[10] M. W. Berry, A. Mohamed & B. W. Yap, *Supervised and Unsupervised Learning for Data Science*. Cham. Switzerland: Springer. 2020. [online]. Available: https://doi.org/10.1007/978-3-030-22475-2_1. Accessed: September. 27, 2023.
[11] E. Alpaydin, *Introduction to Machine Learning*. Third Edition. The MIT Press. 2014.
[12] C. Elkan, *Predictive analytics and data mining*. San Diego. University of California. 2013. [online]. Available: https://cseweb.ucsd.edu/classes/sp15/cse190-c/files/elkan_dm.pdf. Accessed: September. 27, 2023.
[13] S. J. Russell & P. Norvig, *Artificial Intelligence: A Modern Approach*. Harlow. England: Pearson Education. 2022.
[14] A. Chigurupati, R. Thibaux & N. Lassar, *Predicting Hardware Failure Using Machine Learning*. 2016 Annual Reliability and Maintainability Symposium (RAMS). 2016. [online]. Available: https://doi.org/10.1109/RAMS.2016.7448033. Accessed: September. 27, 2023.
[15] D. Mayer, *Support Vector Machines: The Interface to libsvm in package e1071*. FH Technikum Wien. Austria. 2015. [online]. Available: https://vps.fmvz.usp.br/CRAN/web/packages/e1071/vignettes/svmdoc.pdf. Accessed: September. 27, 2023.
[16] C. Stoean & R. Stoean, *Support Vector Machines and Evolutionary Algorithms for Classification Single or Together?*. Springer. 2014. pp. 23. [online]. Available: https://doi.org/10.1007/978-3-319-06941-8. Accessed: September. 27, 2023.
[17] C. Zhang & Y. Ma, *Ensemble Machine Learning Methods and Applications*. Springer. 2012. pp. 35-163. [online]. Available: https://doi.org/10.1007/978-1-4419-9326-7. Accessed: September. 27, 2023.

[18] J. Hemanth, X. Fernando, P. Lafata & Z. Baig, *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*. Springer. 2019. pp. 760. [online]. Available: https://doi.org/10.1007/978-3-030-03146-6. Accessed: September. 27, 2023.
[19] M. W. Gardner & S. R. Dorling, *Artificial Neural Networks (The Multilayer Perceptron) – A review of applications in the atmospheric sciences*. Atmospheric Environment Elsevier. 1998. pp. 2627-2636. [online]. Available: https://doi.org/10.1016/S1352-2310(97)00447-0. Accessed: September. 27, 2023.
[20] M. M. Ali, B. K. Paul, K. Ahmed, F. M. Baui, J. M. W. Quinn & M. A. Moni, *Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison. Computers in Biology and Medicine*. Elsevier. 2021. [online]. Available: https://doi.org/10.1016/j.compbiomed.2021.104672. Accessed: September. 27, 2023.
[21] K. Li, S. Yao, Z. Zhang, B. Cao, C. M. Wilson, D. Kalos, P. f. Kuan, R. Zhu & X. Wang, *Efficient gradient boosting for prognostic biomarker discovery*. Bioinformatics. 2022. pp. 1631-1638. [online]. Available: https://doi.org/10.1093/bioinformatics/btab869. Accessed: September. 27, 2023.
[22] K. Liulys, *Machine Learning Application in Predictive Maintenance*. 2019 Open Conference of Electrical, Electronic and Information Sciences. Vilnius. Lithuania. 2019. pp. 1-4. [online]. Available: https://doi.org/10.1109/eStream.2019.8732146. Accessed: September. 27, 2023.
[23] G. Biau, B. Cadre & L. Rouviere, *Accelerated gradient boostin. Machine Learning 2019*. Springer. 2019. [online]. Available: https://doi.org/10.1007/s10994-019-05787-1. Accessed: September. 27, 2023.
[24] X. W. Liu, Z. L. Long, W. Zhang & L. M. Yang, *Key feature space for predicting the glass-forming ability of amorphous alloys revealed by gradient boosted decision trees model. Journal of Alloys and Compounds*. Elsevier. 2022. [online]. Available: https://doi.org/10.1016/j.jallcom.2021.163606. Accessed: September. 27, 2023.
[25] D. L. Olson & D. Delen, *Advanced Data Mining*. Springer. 2008. [online]. Available: https://doi.org/10.1007/978-3-540-76917-0. Accessed: September. 27, 2023.
[26] I. T. Franco & R. M. d. Figueiredo, *Predictive Maintenance: An Embedded System Approach*. Journal of Control, Automation and Electrical Systems. 2023. [online]. Available: https://doi.org/10.1007/s40313-022-00949-4. Accessed: September. 27, 2023.
[27] G. A. Susto, A. Schirru & S. Pampuri, *Machine Learning for Predictive Maintenance: A Multiple Classifier Approach*. IEEE Transactions on Industrial Informatics. 2015. [online]. Available: https://doi.org/10.1109/TII.2014.2349359. Accessed: September. 27, 2023.
[28] M. Calabrese, M. Cimmino, M. Manfrin, F. Fiume, D. Kapetis, M. Mengoni, S. Ceccacci, E. Frontoni, M. Paolanti, A. Carrotta & G. Toscano, *An Event Based Machine Learning Framework for Predictive Maintenance in Industry 4.0*. IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. Anaheim. California. USA. 2019. [online]. Available: https://doi.org/10.1115/DETC2019-97917. Accessed: September. 27, 2023.
[29] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto & S. G. S. Alcala, *A systematic literature review of machine learning methods applied to predictive maintenance. Computers & Industrial Engineering*. Elsevier. 2019. [online]. Available: https://doi.org/10.1016/j.cie.2019.106024. Accessed: September. 27, 2023.
[30] S. Dey, M. Kedia, N. Agarwal & A. Basu, *Embedded Support Vector Machine: Architectural Enhancements and Evaluation*. 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07). Bangalore. India. 2007. pp. 685-690. [online]. Available: https://doi.org/10.1109/VLSID.2007.73. Accessed: September. 27, 2023.
[31] L. G. Castanheira, J. A. d. Vasconcelos, A. J. R. Reis, P. H. V. Magalhaes & S. A. L. d. Silva, *Application of Neural Networks in the Classification of Incipient Faults in Power Transformers: A Study of Case*. The 2011 International Joint Conference on Neural Networks. San Jose. CA. USA. 2011. pp. 3099-3104. [online]. Available: https://doi.org/10.1109/IJCNN.2011.6033631. Accessed: September. 27, 2023.
[32] T. Gkamas, S. Kontogiannis, V. Karaiskos, C. Pikridas & I. A. Karolos, *Proposed Cloud-assisted Machine Learning Classification Process implemented on Industrial Systems: Application to Critical Events Detection and Industrial Maintenance*. 2022 5th World Symposium on Communication Engineering (WSCE). Nogoya. Japan. 2022. pp. 95-99. [online]. Available: https://doi.org/10.1109/WSCE56210.2022.9916043. Accessed: September. 27, 2023.
[33] S. G. K. Patro & K. K. Sahu, *Normalization: A Preprocessing Stage*. arXiv preprint. 2015. [online]. Available: https://doi.org/10.48550/arXiv.1503.06462. Accessed: September. 27, 2023.
[34] D. V. Souza, J. X. Santos, H. C. Vieira, T. L. Naide, S. Nisgoski & L. E. S. Oliveira, *An automatic recognition system of Brazilian flora species based on textural features of macroscopic images of wood*.

Wood Science and Technology. 2020. [online]. Available: https://doi.org/10.1007/s00226-020-01196-z. Accessed: September. 27, 2023.
[35] N. V. Chawla, K. W. Bowyer, L. O. Hall & W. P. Kegelmeyer, *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research. 2002. [online]. Available: https://doi.org/10.48550/arXiv.1106.1813. Accessed: September. 27, 2023.
[36] D. Chicco, M. J. Warrens & G. Jurman, *The Matthews Correlation Coefficient (MCC) is More Informative Than Cohen's Kappa and Brier Score in Binary Classification Assessment*. IEEE. 2021. [online]. Available: https://doi.org/10.1109/ACCESS.2021.3084050. Accessed: September. 27, 2023.

Appendices

Appendix A: Feature extraction

```
// Extract.cs
// Author: Fayad Al Hanash
namespace ExtractFeatures;
/// <summary>
/// A class to extract features from the measurement results and test results.
/// </summary>
2 references
public class Extract
{
    /// <summary>
    /// Method that extracts the features
    /// Calls the ReadFile method to get the measurements and test result arrays.
    /// Calls GetMeasurementResultFeatures and GetTestResultFeatures methods to create list of
    /// MeasurementResultFeatures and TestResultFeatures objects
    /// Calls the GetMergedList method to merge the lists into one list of MeasurementResultFeatures objects
    /// Groups the measurement result features based on the test result ID
    /// Creates a new list of ExtractedFeatures object
    /// Iterates through each group to calculate the measurements and frequencies
    /// Calls the WriteFile method to save the features in the file
    /// </summary>
    1 reference
    public bool StartExtracting(string measuremetPath, string testPath, string desPath)
    {
        (string[] measurementArray, string[] testArray) = FileUtility.ReadFile(measuremetPath, testPath);
        if (measurementArray == null) return false;
        List<MeasurementResultFeatures> measurementResultList = GetMeasurementResultFeatures(measurementArray);
        List<TestResultFeatures> testResultList = GetTestResultFeatures(testArray);
        List<MeasurementResultFeatures> mergerdList = GetMergedList(measurementResultList, testResultList);
        var groups = mergerdList.GroupBy(b => b.TestResultID);
        List<ExtractedFeatures> featureAfterList = new List<ExtractedFeatures>();
        foreach (var g in groups)
        {
            var ExtQuery = from b in g let execTime = b.ExecutionTime select execTime;
            var NumericDataQuery = from b in g let numericData = b.NumericData select numericData;
            var NumericLowLimitQuery = from b in g let numericLowLimit = b.NumericLowLimit select numericLowLimit;
            var NumericHighLimitQuery = from b in g let numericHighLimit = b.NumericHighLimit select numericHighLimit;
            var measurementIDQuery = from b in g let measurementID = b.MeasurementResultID select measurementID;

            var nameQuery = from n in g let name = n.Name select name;
            var comparator = from b in g let comp = b.Comparator select comp;
            var unit = from b in g let u = b.Unit select u;
            var GELE = from c in comparator where c.GELE == 1 select c.GELE;
            var EQ = from c in comparator where c.EQ == 1 select c.EQ;
            var LOG = from c in comparator where c.LOG == 1 select c.LOG;
            var GTLT = from c in comparator where c.GTLT == 1 select c.GTLT;
            var V = from u in unit where u.V == 1 select u.V;
            var µA = from u in unit where u.µA == 1 select u.µA;
            var NULL = from u in unit where u.NULL == 1 select u.NULL;

            float extMin = ExtQuery.Min();
            float numDMin = NumericDataQuery.Min();
            float numLMin = NumericLowLimitQuery.Min();
            float numHMin = NumericHighLimitQuery.Min();

            ExtractedFeatures after = new ExtractedFeatures()
            {
                TestResultID = g.Key,
                MeasurementTestFrequency = measurementIDQuery.Count(),
                NameFrequency = nameQuery.Count(),
                ExecTimeTotal = g.ElementAt(0).ExecutionTimeAll,
                ExecTimeMes = new Measurement
                {
```

```
                    Avg = ExtQuery.Average(),
                    Max = ExtQuery.Max(),
                    Min = ExtQuery.OrderBy(x => x != 0 && extMin != 0).First(),
                    Median = Helper.MEDIAN(ExtQuery),
                    StdDev = Helper.STDDEV(ExtQuery)
                },
                NumericDataMes = new Measurement
                {
                    Avg = NumericDataQuery.Average(),
                    Max = NumericDataQuery.Max(),
                    Min = NumericDataQuery.OrderBy(x => x != 0 && numDMin != 0).First(),
                    Median = Helper.MEDIAN(NumericDataQuery),
                    StdDev = Helper.STDDEV(NumericDataQuery)
                },
                NumericLowLimitMes = new Measurement
                {
                    Avg = NumericLowLimitQuery.Average(),
                    Max = NumericLowLimitQuery.Max(),
                    Min = NumericLowLimitQuery.OrderBy(x => x != 0 && numLMin != 0).First(),
                    Median = Helper.MEDIAN(NumericLowLimitQuery),
                    StdDev = Helper.STDDEV(NumericLowLimitQuery)
                },
                NumericHighLimitMes = new Measurement
                {
                    Avg = NumericHighLimitQuery.Average(),
                    Max = NumericHighLimitQuery.Max(),
                    Min = NumericHighLimitQuery.OrderBy(x => x != 0 && numHMin != 0).First(),
                    Median = Helper.MEDIAN(NumericHighLimitQuery),
                    StdDev = Helper.STDDEV(NumericHighLimitQuery)
                },
                GELEFreq = GELE.Count(),
                EQFreq = EQ.Count(),
                LOQFreq = LOG.Count(),
                GTLTFreq = GTLT.Count(),
                VFreq = V.Count(),
                µAFreq = µA.Count(),
                NULLFreq = NULL.Count(),
                Status = g.ElementAt(0).Status
            };

            featureAfterList.Add(after);
        }
        return FileUtility.WriteFile(featureAfterList, desPath);
    }

    /// <summary>
    /// Method that returns a list of MeasurementResultFeatures object
    /// </summary>
    1 reference
    List<MeasurementResultFeatures> GetMeasurementResultFeatures(string[] arr) =>
            arr
            .Skip(1)
            .Select(line =>
            {
                string[] col = line.Split(',');
                return new MeasurementResultFeatures
                {
                    MeasurementResultID = int.Parse(col[0]),
                    TestResultID = int.Parse(col[1]),
                    Name = col[2],
                    TimeStamp = col[3].Substring(0, col[3].LastIndexOf(':')),
                    ExecutionTime = float.Parse(col[4]),
                    NumericData = float.Parse(col[5]),
                    NumericLowLimit = float.Parse(col[6]),
                    NumericHighLimit = float.Parse(col[7]),
                    Comparator = Helper.GetComparator(col[8]),
                    Unit = Helper.GetUnit(col[9]),
                    Status = Helper.GetStatus(col[10])
                };
            })
```

```
            .ToList();

    /// <summary>
    /// Method that returns a list of TestResultFeatures object
    /// </summary>
    1 reference
    List<TestResultFeatures> GetTestResultFeatures(string[] arr) =>
    arr
    .Skip(1)
    .Select(line =>
    {
        string[] col = line.Split(',');
        return new TestResultFeatures
        {
            TestResultID = int.Parse(col[0]),
            Timestamp = col[1].Substring(0, col[1].LastIndexOf(':')),
            ExecutionTime = float.Parse(col[2]),
            Status = Helper.GetStatus(col[3])
        };
    })
    .ToList();

    /// <summary>
    /// Method that merges list of MeasurementResultFeatures and TestResultFeatures lists and returns a new list
    /// of MeasurementResultFeatures
    /// </summary>
    1 reference
    List<MeasurementResultFeatures> GetMergedList(List<MeasurementResultFeatures> mRList, List<TestResultFeatures> tRList)
    {

        var list = (from mR in mRList
                    join tR in tRList
                    on mR.TestResultID equals tR.TestResultID
                    into temp
                    from tR in temp.DefaultIfEmpty()
                    select new MeasurementResultFeatures
                    {
                        MeasurementResultID = mR.MeasurementResultID,
                        TestResultID = mR.TestResultID,
                        Name = mR.Name,
                        TimeStamp = mR.TimeStamp,
                        ExecutionTime = mR.ExecutionTime,
                        ExecutionTimeAll = tR.ExecutionTime,
                        NumericData = mR.NumericData,
                        NumericLowLimit = mR.NumericLowLimit,
                        NumericHighLimit = mR.NumericHighLimit,
                        Comparator = mR.Comparator,
                        Unit = mR.Unit,
                        Status = tR.Status
                    }).ToList();

        return list;
    }
}
```

Extract.cs

```
// TestResultFeatures.cs
namespace ExtractFeatures;
/// <summary>
/// A test result features class represents the features of the test results.
/// </summary>
4 references
public class TestResultFeatures
{
    2 references
    public int TestResultID { get; set; }
    1 reference
    public string Timestamp { get; set; }
    2 references
    public float ExecutionTime { get; set; }
    2 references
    public int Status { get; set; }
}
```

TestResultFeatures.cs

```cs
// MeasurementResultFeatures.cs
namespace ExtractFeatures;
/// <summary>
/// A measurement result features class represents the features of the measurement results.
/// </summary>
7 references
public class MeasurementResultFeatures
{
    4 references
    public int MeasurementResultID { get; set; }
    5 references
    public int TestResultID { get; set; }
    4 references
    public string Name { get; set; }
    3 references
    public string TimeStamp { get; set; }
    4 references
    public float ExecutionTime { get; set; }
    2 references
    public float ExecutionTimeAll { get; set; }
    4 references
    public float NumericData { get; set; }
    4 references
    public float NumericLowLimit { get; set; }
    4 references
    public float NumericHighLimit { get; set; }
    4 references
    public Comparator Comparator { get; set; }
    4 references
    public Unit Unit { get; set; }
    3 references
    public int Status { get; set; }
}
```

MeasurementResultFeatures.cs

```cs
// Comparators.cs
namespace ExtractFeatures;
/// <summary>
/// A comparator record represents the comparators used to evaluate a test.
/// </summary>
7 references
public record Comparator()
{
    4 references
    public int GELE { get; set; }
    4 references
    public int EQ { get; set; }
    4 references
    public int LOG { get; set; }
    4 references
    public int GTLT { get; set; }

    0 references
    public override string ToString() => $"{GELE},{EQ},{LOG},{GTLT}";
}
```

Comparators.cs

```cs
// Unit.cs
namespace ExtractFeatures;
/// <summary>
/// A unit record represents the unit of the measured values.
/// </summary>
6 references
public record Unit
{
    4 references
    public int V { get; set; }
    4 references
    public int µA { get; set; }
    4 references
    public int NULL { get; set; }
    0 references
    public override string ToString() => $"{V},{µA},{NULL}";
}
```

Unit.cs

```cs
// Measurement.cs
namespace ExtractFeatures;
/// <summary>
/// A measurement class represents the calculated Min, Max, Mean, Median,
/// and StdDev of the numerical and execution time measurments.
/// </summary>
8 references
public class Measurement
{
    6 references
    public float Min { get; set; }
    6 references
    public float Max { get; set; }
    8 references
    public float Avg { get; set; }
    7 references
    public float Median { get; set; }
    8 references
    public float StdDev { get; set; }

    0 references
    public override string ToString() =>
        $"{Min},{Max},{Avg},{Median},{StdDev}";
    1 reference
    public string ExecToString() => $"{Max},{Avg},{StdDev}";
    1 reference
    public string NumDataToString() => $"{Min},{Avg},{Median},{StdDev}";
    1 reference
    public string NumDLowToString() => $"{Avg},{Median},{StdDev}";
    0 references
    public string NumDHighToString() => NumDLowToString();
}
```

Measurement.cs

```
// ExtractedFeatures.cs
namespace ExtractFeatures;
/// <summary>
/// A feature class represents the extracted features (features after extraction).
/// </summary>
5 references
public class ExtractedFeatures
{
    1 reference
    public int TestResultID { get; set; }
    1 reference
    public int MeasurementTestFrequency { get; set; }
    1 reference
    public int NameFrequency { get; set; }
    2 references
    public float ExecTimeTotal { get; set; }
    2 references
    public  Measurement ExecTimeMes { get; set; }
    2 references
    public  Measurement NumericDataMes { get; set; }
    1 reference
    public  Measurement NumericLowLimitMes { get; set; }
    1 reference
    public  Measurement NumericHighLimitMes { get; set; }
    1 reference
    public int GELEFreq { get; set; }
    1 reference
    public int EQFreq { get; set; }
    1 reference
    public int LOQFreq { get; set; }
    1 reference
    public int GTLTFreq { get; set; }
    1 reference
    public int VFreq { get; set; }
    1 reference
    public int µAFreq { get; set; }
    1 reference
    public int NULLFreq { get; set; }
    2 references
    public int Status { get; set; }

    /*public override string ToString() => $"{TestResultID},{MeasurementTestFrequency}," +
    $"{NameFrequency},{ExecTimeTotal},{ExecTimeMes},{NumericDataMes},{NumericLowLimitMes}," +
    $"{NumericHighLimitMes},{GELEFreq},{EQFreq},{LOQFreq},{GTLTFreq},{VFreq},{µAFreq}, " +
    $" {NULLFreq},{(int)Status}";*/
    1 reference
    public override string ToString() => $"{ExecTimeTotal},{ExecTimeMes.ExecToString()}," +
        $"{NumericDataMes.NumDataToString()},{(int)Status}";
}
```

ExtractedFeatures.cs

```
// Helper.cs
namespace ExtractFeatures;

13 references
public class Helper
{
    /*public static string ExtractedFeaturesNames = "TestResultID,MeasurementTestFrequency,NameFrequency," +
        "ExecTimeTotal,ExecMin,ExecMax,ExecAvg,ExecMedian,ExecStdDev,NumDataMin,NumDataMax,NumDataAvg," +
        "NumDataMedian,NumDataStdDev,NumDataLowLimMin,NumDataLowLimMax,NumDataLowLimAvg,NumDataLowLimMedian," +
        "NumDataLowLimStdDev,NumDataHighLimMin,NumDataHighLimMax,NumDataHighLimAvg,NumDataHighLimMedian," +
        "NumDataHighLimStdDev,GELEFreq,EQFreq,LOQFreq,GTLTFreq,VFreq,µAFreq,NULLFreq,Status";*/
    1 reference
    public static string ExtractedFeaturesNames = "ExecTimeTotal,ExecMax,ExecAvg,ExecStdDev,NumDataMin," +
        "NumDataAvg,NumDataMedian,NumDataStdDev,Status";
    0 references
    public static string OrginalFeaturesNames = "measurement_result_id,test_result_id,name,timestamp," +
        "execution_time,numeric_data,numeric_low_limit,numeric_high_limit,comparator,unit,status";

    /// <summary>
    /// Method that returns a new instance of comparator and sets the value of the comparator based on the string value.
    /// </summary>
    1 reference
    public static Func<string, Comparator> GetComparator = x => x switch
    {
        "GELE" => new() { GELE = 1 },
        "EQ" => new() { EQ = 1 },
        "LOG" => new() { LOG = 1 },
        "GTLT" => new() { GTLT = 1 },
        _ => new(),
    };

    /// <summary>
    /// Method that returns a new instance of unit and sets the value of the unit based on the string value.
    /// </summary>

    1 reference
    public static Func<string, Unit> GetUnit = x => x switch
    {
        "V" => new Unit { V = 1 },
        "µA" => new Unit { µA = 1 },
        "NULL" => new Unit { NULL = 1 },
        _ => new Unit()
    };

    /// <summary>
    /// Method that returns the status of a test as an integer based on the string value.
    /// </summary>
    2 references
    public static int GetStatus(string status)
    {
        return status switch
        {
            "passed" => 0,
            "failed" => 1,
            "error" => 2,
            "terminated" => 3,
            _ => 4,
        };
    }

    /// <summary>
    /// Method that returns the status of a test as a string based on the integer value.
    /// </summary>
```

```
0 references
public static string GetStatus(int status)
{
    return status switch
    {
        0 => "passed",
        1 => "failed",
        2 => "error",
        3 => "terminated",
        _ => "unknown",
    };
}

/// <summary>
/// Method that calculates the standard deviation.
/// </summary>
4 references
public static float STDDEV(IEnumerable<float> s)
{
    if (s.Count() > 0)
    {
        float x = (float) Math.Sqrt(s.Sum(d => Math.Pow(d - s.Average(), 2)) / (s.Count() - 1));
        if (double.IsNaN(x))
            return 0;
        else
            return x;
    }
    else return 0;
}

/// <summary>
/// Method that calculates the median
/// </summary>
4 references
public static float MEDIAN(IEnumerable<float> s) =>
    s.Count() > 0 ? s.OrderBy(x => x).ElementAt((int)Math.Ceiling((double)(s.Count() - 1) / 2)) : 0;

}
```

Helper.cs

```
// FileUtility.cs
namespace ExtractFeatures;

2 references
public class FileUtility
{
    /// <summary>
    /// Method that reads the measurement results and test results from the csv files,
    /// returns a tuple of string arrays containing the measurement results and test results.
    /// </summary>
    1 reference
    public static (string[], string[]) ReadFile(string measuremetPath, string testPath)
    {
        try
        {
            if (!File.Exists(measuremetPath))
                throw new FileNotFoundException("File not found", measuremetPath);

            if (testPath == null)
            {
                return (File.ReadAllLines(measuremetPath), null);
            }
            else
            {
                if (!File.Exists(testPath))
                    throw new FileNotFoundException("File not found", testPath);
                return (File.ReadAllLines(measuremetPath), File.ReadAllLines(testPath));
            }

        }
        catch (Exception e)
        {
            throw new Exception(e.ToString());
        }
        return (null, null);
    }
```

```
36      /// <summary>
37      /// Method that writes the extracted features to a csv file.
38      /// </summary>
        1 reference
39      public static bool WriteFile(List<ExtractedFeatures> afterExtracts, string path)
40      {
41          try
42          {
43              using (StreamWriter sw = new StreamWriter(path))
44              {
45                  sw.WriteLine(Helper.ExtractedFeaturesNames);
46                  afterExtracts.ForEach(x => sw.WriteLine(x.ToString()));
47                  return true;
48              }
49          }
50          catch (Exception e)
51          {
52              throw new Exception(e.ToString());
53          }
54          return false;
55      }
56  }
57
```

FileUtility.cs

Appendix B: Modeling & Evaluation

```
1   # main.py
2   # Author: Fayad Al Hanash
3   # Date: 2023-08-30
4
5   # This file is the main file for the project
6   # import libraries
7   import numpy as np
8   from helper import Helper
9   import sklearn.model_selection as ms
10  from file_utility import FileUtility
11  from parameters_tunning import ParametersTunning
12  from binary_classification import BinaryClassification
13  from multi_classification import MultiClassification
14
15  # method that runs the binary classification for the given algorithm and dataset
16  def binary_classification( alg, is_blanced, is_modeled, x_train, x_test, y_train, y_test):
17      cl = BinaryClassification(alg,is_blanced, x_train, x_test, y_train, y_test)
18      if is_modeled:
19          mod = FileUtility.load_model(f'{alg}_bin_{is_blanced}_balancing')
20          cl.set_model(mod)
21      else:
22          cl.fit()
23          FileUtility.save_model(cl.get_model(), f'{alg}_bin_{is_blanced}_balancing')
24      cl.display_matrix(f"{alg} bin for {is_blanced} dataset")
25      cl.plot()
26
```

```python

# method that runs the multi classification for the given algorithm and dataset
def multi_classification( alg, is_blanced, is_modeled, x_train, x_test, y_train, y_test):
    cl = MultiClassification(alg,is_blanced, x_train, x_test, y_train, y_test)
    if is_modeled:
       mod = FileUtility.load_model(f'{alg}_mul_{is_blanced}_balancing')
       cl.set_model(mod)
    else:
        cl.fit()
        FileUtility.save_model(cl.get_model(), f'{alg}_mul_{is_blanced}_balancing')
    cl.display_confusion_matrix(f"{alg} multi for {is_blanced} dataset")
    cl.plot()


# method that runs the parameters tunning for the given algorithm and dataset
def tune_parameters(alg,is_balanced,cl_type, x_train, y_train):
    pt = ParametersTunning(alg, x_train, y_train)
    para = pt.tune_parameters(is_balanced)
    FileUtility.save_text_json(para, f'{alg}_{cl_type}_{is_balanced}_balancing.txt')

# method that runs the print the cross validation for an already saved cross validation
def print_cross_validation(alg,is_balanced,cl_type):
    cr = FileUtility.load_model(f'{alg}_{cl_type}_{is_balanced}_balancing_cross')
    score = np.mean(cr['test_score'])
    print(f'Mean Accuracy: {score:.2%}')


# methid that runs the main method
def main():

    # read the data from the file
    xData, yData = FileUtility.read_file('data/extracted.csv')
    # split the data into train and test
    x_train, x_test, y_train, y_test = ms.train_test_split(xData, yData, train_size=0.80,
                                                           test_size=0.20, random_state=0)
    # scale the data
    x_train, x_test = Helper.scale_min_max(x_train, x_test)


    algorithm = "svm"
    is_blanced = False
    cl_type = 'bin'
    is_modeled = True

    if cl_type == 'bin':
        binary_classification(algorithm, is_blanced, is_modeled, x_train, x_test, y_train, y_test)
    else:
        multi_classification(algorithm, is_blanced, is_modeled, x_train, x_test, y_train, y_test)


# run the main method
if __name__ == "__main__":
    main()
```

main.py

```python
# model.py
# import libraries
from numpy import mean
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import cross_validate , RepeatedStratifiedKFold

# The class Model which is used to build the model
class Model:
    # method that initializes the model
    def __init__(self, x_train, x_test, y_train, y_test):
        self.model = None
        self.algorithm = None
        self.score = None
        self.x_train = x_train
        self.x_test = x_test
        self.y_train = y_train
        self.y_test = y_test
        self.y_predict = None
        self.y_predict_proba = None
        self.cross_validation = None

    # method that fits the model
    def fit(self):
        pass

    # method that fits the model with unbalanced data
    def fit_unbalanced_data(self):
        self.model = self.algorithm.fit(self.x_train, self.y_train)
        self.score = self.model.score(self.x_test, self.y_test)
        self.y_predict = self.model.predict(self.x_test)
        self.y_predict_proba = self.model.predict_proba(self.x_test)

```

```python
    # method that fits the model with balanced data
    def fit_balanced_data(self):
        try:
            oversample = SMOTE()
            undersample = RandomUnderSampler()
            steps = [('o', oversample), ('u', undersample)]
            pipeline = Pipeline(steps=steps)
            x_train_res, y_train_res = pipeline.fit_resample(self.x_train, self.y_train)
            cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
            self.cross_validation = cross_validate(self.algorithm, x_train_res, y_train_res,
                                                   scoring='accuracy', cv=cv, return_estimator=True)
            self.score = mean(self.cross_validation['test_score'])
            print(f'Mean Accuracy: {self.score:.2%}')
            #self.model = cross_val['estimator'][0].fit(x_train_res, y_train_res)
            self.model = self.algorithm.fit(x_train_res, y_train_res)
            self.y_predict = self.model.predict(self.x_test)
            self.y_predict_proba = self.model.predict_proba(self.x_test)
        except Exception as ex:
            raise ex

    # method that predicts and returns the result
    def predict(self, x):
        return self.model.predict(x)

    # method that predicts and returns the probability
    def predict_proba(self, x):
        return self.model.predict_proba(x)

    # method that returns the model's score
    def get_score(self):
        return self.score

    # method that returns the model
    def get_model(self):
        return self.model

    # method that sets the model and predicts the result and probability for the test data
    def set_model(self,model):
        self.model = model
        self.y_predict = self.model.predict(self.x_test)
        self.y_predict_proba = self.model.predict_proba(self.x_test)

    # method that sets the cross validation
    def set_cross_validation(self,cross):
        self.cross_validation = cross

    # method that returns the cross validation
    def get_cross_validation(self):
        return self.cross_validation

```

model.py

```python
# binary_classification.py
# Importing the libraries
from plot import Plot
from sklearn import svm
from model import Model
from helper import Helper
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,precision_score
from sklearn.metrics import recall_score,f1_score,roc_auc_score,matthews_corrcoef,cohen_kappa_score, roc_curve

# The class BinaryClassification which is used to build the binary classification model
# inherits the Model class
class BinaryClassification(Model):
    # method that initializes the binary classification model
    # calls the super class constructor
    # calls the binarize method from the helper class to binarize the target features
    def __init__(self, alg, is_blanced, x_train, x_test, y_train, y_test):
        self.is_blanced = is_blanced
        super().__init__(x_train, x_test, Helper.binarize(y_train), Helper.binarize(y_test))
        self.set_algorithm(alg)

    # method that initializes the algorithm
    def set_algorithm(self, alg):
        if alg == "mlp":
            if self.is_blanced == False:
                self.algorithm = MLPClassifier(activation='relu',hidden_layer_sizes=(200,),
                                               learning_rate='constant',solver='adam')
            else:
                self.algorithm = MLPClassifier(activation='relu',hidden_layer_sizes=(50,50),
                                               learning_rate='adaptive',solver='adam', max_iter=300)
        elif alg == "gb":
            if self.is_blanced == False:
                self.algorithm = GradientBoostingClassifier(n_estimators=300,criterion='squared_error',
                                                            loss='deviance',max_features='sqrt')
            else:
                self.algorithm = GradientBoostingClassifier(n_estimators=300,criterion='friedman_mse',
                                                            loss='log_loss',max_features='log2')
        elif alg == "svm":
            self.algorithm = svm.SVC( C=5,degree=3,gamma='scale',kernel='poly' ,probability=True)
        elif alg == "rf":
            if self.is_blanced == False:
                self.algorithm = RandomForestClassifier(n_estimators=100,criterion='log_loss',max_features='sqrt')
            else:
                self.algorithm = RandomForestClassifier(n_estimators=200,criterion='gini',max_features='sqrt')
        else:
            raise ValueError("Invalid algorithm")

```

```python
    # method that fits the model
    def fit(self):
        if self.is_blanced == False:
            self.fit_unbalanced_data()
        else:
            self.fit_balanced_data()

    # method that returns the confusion matrix
    def get_confusion_matrix(self):
        return confusion_matrix(self.y_test, self.y_predict)

    # method that returns the accuracy score
    def get_accuracy(self):
        return accuracy_score(self.y_test, self.y_predict)

    # method that returns the precision score
    def get_precision(self):
        return precision_score(self.y_test, self.y_predict)

    # method that returns the recall score
    def get_recall(self):
        return recall_score(self.y_test, self.y_predict)

    # method that returns the f1 score
    def get_f1_score(self):
        return f1_score(self.y_test, self.y_predict)

    # method that returns the roc auc score
    def get_roc_auc_score(self):
        return roc_auc_score(self.y_test, self.model.predict_proba(self.x_test)[:,1])

    # method that returns the roc curve
    def get_roc_curve(self):
        return roc_curve(self.y_test, self.model.predict_proba(self.x_test)[:,1])

    # method that returns the matthews corrcoef score
    def get_matthews_corrcoef(self):
        return matthews_corrcoef(self.y_test, self.y_predict)

    # method that returns the cohens kappa score
    def get_cohen_kappa_score(self):
        return cohen_kappa_score(self.y_test, self.y_predict)

    # method that displays the measures
    def display_matrix(self, title):
        print("*******************************************************")
        print(f"Confusion Matrix: {title}\n\n{self.get_confusion_matrix()}\n")
        print(f"Accuracy:\t{(self.get_accuracy()):.2%}")
        print(f"Precision:\t{(self.get_precision()):.3}")
        print(f"Recall:\t\t{(self.get_recall()):.3}")
        print(f"F1 Score:\t{(self.get_f1_score()):.3}")
        print(f"Kappa:\t\t{(self.get_cohen_kappa_score()):.3}")
        print(f"ROC AUC:\t{(self.get_roc_auc_score()):.3}")
        print(f"MCC:\t\t{(self.get_matthews_corrcoef()):.3}")
        print("*******************************************************")

    # method that plots the confusion matrix and roc curve
    def plot(self):
        Plot.plot_confusion_matrix_binary(self.get_confusion_matrix(), f"Confusion Matrix binary
                                          {self.algorithm.__class__.__name__}")
        Plot.plot_roc_curve(self.get_roc_curve()[0], self.get_roc_curve()[1], self.get_roc_auc_score(),
                            f"ROC Curve binary {self.algorithm.__class__.__name__}")
```

binary_classification.py

```python
# multi_classification.py
# Importing the libraries
import numpy as np
from model import Model
from helper import Helper
from plot import Plot
from sklearn import svm
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import multilabel_confusion_matrix, classification_report

# The class MultiClassification which is used to build the multi classification model
# inherits the Model class
class MultiClassification(Model):

    # method that initializes the multi classification model
    # calls the super class constructor
    def __init__(self, alg,is_blanced, x_train, x_test, y_train, y_test):
        super().__init__(x_train, x_test, y_train, y_test)
        self.is_blanced = is_blanced
        self.set_algorithm(alg)
        self.num_classes = len(np.unique(y_test))

    # method that initializes the algorithm
    def set_algorithm(self, alg):
        if alg == "mlp":
            if self.is_blanced == False:
                self.algorithm = MLPClassifier(activation='relu',hidden_layer_sizes=(50,50),
                                               learning_rate='adaptive',solver='adam',max_iter=200)
            else:
                self.algorithm = MLPClassifier(activation='relu',hidden_layer_sizes=(200,),
                                               learning_rate='adaptive',solver='adam')
        elif alg == "gb":
            if self.is_blanced == False:
                self.algorithm = GradientBoostingClassifier(n_estimators=300,criterion='squared_error',
                                                            loss='log_loss',max_features='sqrt')
            else:
                self.algorithm = GradientBoostingClassifier(n_estimators=300,criterion='friedman_mse',
                                                            loss='deviance',max_features='sqrt')
        elif alg == "svm":
            if self.is_blanced == False:
                self.algorithm = svm.SVC(C=5,degree=4,gamma='scale',kernel='poly' , probability=True)
            else:
                self.algorithm = svm.SVC(C=3.5,degree=3, gamma='scale',kernel='poly', probability=True)
        elif alg == "rf":
            if self.is_blanced == False:
                self.algorithm = RandomForestClassifier(n_estimators=200,criterion='log_loss',max_features='sqrt')
            else:
                self.algorithm = RandomForestClassifier(n_estimators=300,criterion='log_loss',max_features='sqrt')
        else:
            raise ValueError("Invalid algorithm")

```

```
    # method that fits the model
    def fit(self):
        if self.is_blanced == False:
            self.fit_unbalanced_data()
        else:
            self.fit_balanced_data()

    # method that returns the multilabel confusion matrix
    def get_multilabel_confusion_matrix(self):
        return multilabel_confusion_matrix(self.y_test, self.y_predict)

    def get_classifcation_report(self):
        return classification_report(self.y_test, self.y_predict,
                                     target_names=["passed","failed","error","terminated"])

    # method that returns the accuracy, precision, recall and f1 score
    def get_accuracy_recall_precision_f1(self,arr):
        tn = arr[0][0]
        fp = arr[0][1]
        fn = arr[1][0]
        tp = arr[1][1]

        precision = 0
        recall = 0
        accuracy = 0
        f1 = 0
        # ignore the divide by zero warning
        np.seterr(divide='ignore', invalid='ignore')

        if(tp+tn+fp+fn == 0):
            accuracy = 0
            #raise ZeroDivisionError("Accuracy is undefined when TP+TN+FP+FN=0")
        else:
            accuracy = (tp+tn)/(tp+tn+fp+fn)
        if(tp == 0):
            precision = 0
        elif (tp+fp == 0):
            precision = 0
            #raise ZeroDivisionError("Precision is undefined when TP+FP=0")
        #elif (fp == 0):
         #   precision = "precision is not estimated when FP=0, the value is 1"
        else:
            precision = tp/(tp+fp)
        if (tp + fn == 0):
            recall = 0
            #raise ZeroDivisionError("Recall is undefined when TP+FN=0")
        #elif (fn == 0):
         #   recall = "recall is not estimated when FN=0, the value is 1"
        else:
            recall = tp/(tp+fn)

        f1 = 2 * (precision * recall) / (precision + recall)

        return accuracy,precision,recall,f1

    # method that returns the kappa and mcc
    def get_kappa_mcc(self, arr):
        tn = arr[0][0]
        fp = arr[0][1]
        fn = arr[1][0]
        tp = arr[1][1]

```

```
        # ignore the divide by zero warning
        np.seterr(divide='ignore', invalid='ignore')

        kappa_divisor = (tp+fp)*(fp+tn)+(tp+fn)*(fn+tn)
        if (kappa_divisor == 0):
            kappa = 0
            #raise ZeroDivisionError("Kappa is undefined when divisor=0")
        else:
            kappa = (tp+tn)/kappa_divisor
        mcc_divisor = np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
        if (mcc_divisor == 0):
            mcc = 0
            #raise ZeroDivisionError("MCC is undefined when divisor=0")
        else:
            mcc = (tp*tn-fp*fn)/mcc_divisor
        return kappa,mcc

    # method that displays the confusion matrix
    def display_confusion_matrix(self,title):
        print("******************************************************")
        cm = self.get_multilabel_confusion_matrix()
        print(f"Confusion Matrix for {title}:\n",cm)
        for i in range(self.num_classes):
            print(f"Class: ({Helper.get_status(i)})")
            accuracy, precision, recall, f1 = self.get_accuracy_recall_precision_f1(cm[i])
            kappa, mcc = self.get_kappa_mcc(cm[i])
            print(f"Accuracy:\t{accuracy:.2%}\nPrecision:\t{precision:.3}\nRecall:\t\t{
                recall:.3}\nF1 Score:\t{f1:.3}\nKappa:\t\t{kappa:.3}\nMCC:\t\t{mcc:.3}")
            #(f"Cohen's Kappa Score:\t{kappa:.3}\nMatthews Correlation Coefficient:\t{mcc:.3}")
        print("******************************************************")

    # method that plots the confusion matrix
    def plot(self):
        Plot.plot_confusion_matrix_multi_class(self.y_predict,self. y_test,
                                               f"Confusion Matrix multi-class {self.algorithm.__class__.__name__}")

    # method that displays the classification report
    def display_report(self):
        print("******************************************************")
        print("Classification Report:\n",self.get_classifcation_report())
        print("******************************************************")

```

multi_classification.py

```python
# plot.py
# The class Plot which is used to plot the data
# import the libraries
import numpy as np
import pandas as pd
import seaborn as sns
from helper import Helper
from matplotlib import pyplot as plt

class Plot:

    # method that plots the confusion matrix
    @staticmethod
    def plot_confusion_matrix_binary(cm,title):
        cm = cm
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
        plt.title(title)
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.show()

    # method that plots the roc curve
    @staticmethod
    def plot_roc_curve(fpr, tpr, roc_auc_score, title):
        fpr = fpr
        tpr = tpr
        plt.plot(fpr, tpr, label=f'ROC Curve: {roc_auc_score:.3}')
        plt.plot([0,1],[0,1], 'k--', label='Random Guess')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.legend(loc='lower right')
        plt.title(title)
        plt.show()

    # method that plots the confusion matrix for multi-class classification
    @staticmethod
    def plot_confusion_matrix_multi_class(y_pred, y_test, title):
        fig, ax = plt.subplots(figsize=(7,5))
        from sklearn.metrics import ConfusionMatrixDisplay
        ConfusionMatrixDisplay.from_predictions(y_test, y_pred, ax=ax, cmap='Blues')
        ax.xaxis.set_ticklabels(Helper.get_the_classes())
        ax.yaxis.set_ticklabels(Helper.get_the_classes())
        _=ax.set_title(title)
        plt.show()

    # method that plots the correlation matrix
    @staticmethod
    def plot_correlation_matrix(data, title):
        corr = np.corrcoef(data, rowvar=False)
        sns.heatmap(corr, annot=True, cmap='Blues')
        plt.title(title)
        plt.show()

    # method that plots the distribution of the classes both binary and multi-class classificatio
    @staticmethod
    def plot_class_distribution(x,str):
        class_count = pd.DataFrame(x).value_counts(sort = False)
        class_count.plot(kind = 'bar', rot=0)
        plt.title(f"{str}")
        plt.xlabel("Classes")
        plt.ylabel("Frequency")
        if len(class_count) <= 2 :
            plt.xticks(range(2), Helper.get_the_binary_classes())
        else:
            plt.xticks(range(4), Helper.get_the_classes())
        plt.show()

```

```
    # method that plots the histogram of each feature
    @staticmethod
    def plot_one_feature_histogram(data, title, xlabel, ylabel, bins=10):
        plt.barh(data,np.arange(len(data)))
        #plt.hist(data, bins=bins)
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.show()

    # method that plots the histogram of all features
    @staticmethod
    def plot_features_histogram(data):
        num_features = data.shape[1]
        num_rows = (num_features +1) // 2
        fig, axes = plt.subplots(num_rows, 2, figsize=(10, 6 * num_rows))
        for i in range(num_features):
            ax = axes[i // 2, i % 2]
            ax.hist(data.iloc[:, i], bins=25)
            #ax.set_title(f"feature {i}")
            #ax.title.set_position([.5, 1.05])
            #ax.set_xlabel(xlabel)
            #ax.set_ylabel(ylabel)
        for i in range(num_features, num_rows * 2):
            fig.delaxes(axes.flatten()[i])
        plt.show()

```

plot.py

```
# helper.py
# A helper class for the main program
class Helper:

    # method that binarizes the data
    @staticmethod
    def binarize(y_data):
        for i in range(len(y_data)):
            if y_data[i] == 0:
                y_data[i] = 0
            else:
                y_data[i] = 1
        return y_data


    # method that returns the classes of the target feature as a list
    @staticmethod
    def get_the_classes():
        return ['passed','failed','error','terminated']

    # method that returns the binary classes of the target feature as a list
    @staticmethod
    def get_the_binary_classes():
        return ['passed','rest']


    # method that normalizes the data using the min max scaler
    # returns the normalized train and test data
    @staticmethod
    def scale_min_max(x_train, x_test):
        from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler(feature_range=(0, 1))
        s_train = scaler.fit_transform(x_train)
        s_test = scaler.transform(x_test)
        return s_train, s_test
```

```
36
37      # method that returns the status of the target feature
38      @staticmethod
39      def get_status(i):
40          if i ==0:
41              x = "passed"
42          elif i ==1:
43              x = "failed"
44          elif i ==2:
45              x = "error"
46          elif i ==3:
47              x = "terminated"
48          else:
49              x = "other"
50          return x
```

helper.py

```
1   # file_utility.py
2   # FileUtility class to read and write files
3   import json
4   import pickle
5
6   class FileUtility:
7
8       # method that reads file and returns x_data and y_data
9       @staticmethod
10      def read_file(filename):
11          try:
12              x_data = []
13              y_data = []
14              with open(filename, 'r') as f:
15                  lines = f.readlines()
16                  for line in lines[1:lines.__len__()]:
17                      line = line.strip().split(',')
18                      x_data.append([float(line[0]), float(line[1]), float(line[2]),float(line[3]),
19                                     float(line[4]),float(line[5]),float(line[6]),float(line[7])])
20                      y_data.append(int(line[8]))
21              return x_data, y_data
22          except IOError as e:
23              print("read method error", e.strerror)
24
25      # method that writes x_data and y_data to file
26      @staticmethod
27      def write_to_file(x_train, x_test, filename):
28          try:
29              with open(filename, 'w') as f:
30                  for i in range(len(x_train)):
31                      for j in range(len(x_train[i])):
32                          f.write(str(x_train[i][j]))
33                          f.write(',')
34                      #f.write(str(y_train[i]))
35                      f.write('\n')
36                  for i in range(len(x_test)):
37                      for j in range(len(x_test[i])):
38                          f.write(str(x_test[i][j]))
39                          f.write(',')
40                      #f.write(str(y_test[i]))
41                      f.write('\n')
42          except IOError as e:
43              print("write method error", e.strerror)
44
```

```
45
46      # method that saves the model
47      @staticmethod
48      def save_model(model, filename):
49          try:
50              with open(f"modeled/{filename}", 'wb') as f:
51                  pickle.dump(model, f)
52          except IOError as e:
53              print("save error", e.strerror)
54
55      # method that loads the model
56      @staticmethod
57      def load_model(filename):
58          try:
59              with open(f"modeled/{filename}", 'rb') as f:
60                  return pickle.load(f)
61          except IOError as e:
62              print("load error", e.strerror)
63
64      # method that saves texts as json format
65      @staticmethod
66      def save_text_json(data, filename):
67          try:
68              with open(f"parameters/{filename}", 'w') as f:
69                  json.dump(data,f)
70          except IOError as e:
71              print("save json error", e.strerror)
```

file_utility.py

```python
# parameters_tunning.py
# import libraries
from sklearn import svm
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

# The class ParametersTunning which is used to tune the parameters of the model
class ParametersTunning:
    # method that initializes the parameters tunning
    def __init__(self, alg_name, x_train, y_train):
        self.x_train = x_train
        self.y_train = y_train
        self.algorithm = None
        self.parameters = {}
        self.set_algorithm(alg_name)

    # method that initializes the algorithm and the parameters
    def set_algorithm(self, alg):
        if alg == "mlp":
            self.algorithm = MLPClassifier()
            self.parameters = {'hidden_layer_sizes': [(50,), (100,),(200,),(50,50)], 'activation':
                               ['relu','tanh','sigmoid'], 'learning_rate': ['constant','adaptive'],
                               'solver': [ 'sgd', 'adam'],'max_iter':[200,300,400,500]}
        elif alg == "gb":
            self.algorithm = GradientBoostingClassifier()
            self.parameters = { 'n_estimators':[100,200,300],'criterion' :
                              ['friedman_mse','squared_error'], 'max_features': [ 'sqrt', 'log2','auto'],
                                   'loss': [ 'log_loss', 'deviance','exponential']}
        elif alg == "svm":
            self.algorithm = svm.SVC()
            self.parameters = { 'C': [0.1,1,1.5,2,2.5,3,4,4.5,5], 'kernel': [ 'rbf','poly'],
                               'degree' : [3,4],'gamma': ['scale', 'auto']}
        elif alg == "rf":
            self.algorithm = RandomForestClassifier()
            self.parameters = { 'n_estimators':[100,200,300],'class_weight' :
                              ['balanced_subsample','balanced'], 'max_features': [ 'sqrt', 'log2','None'],
                                  'criterion' : ['gini','entropy','log_loss']}
        else:
            raise ValueError("Invalid algorithm")

```

```python
    # method that tunes the parameters of the model and returns the best parameters
    def tune_parameters(self, is_balanced):
        if is_balanced == False:
            return self.tune_parameters_unbalanced()
        else:
            return self.tune_parameters_balanced()

    # method that tunes the parameters of the model with unbalanced data and returns the best parameter
    def tune_parameters_unbalanced(self):
        cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
        #grid_search = GridSearchCV(estimator=self.algorithm, param_grid=self.parameters, cv=cv)
        grid_search = GridSearchCV(estimator=self.algorithm, param_grid=self.parameters)
        grid_search.fit(self.x_train, self.y_train)
        print(grid_search.best_params_)
        return grid_search.best_params_


    # method that tunes the parameters of the model with balanced data and returns the best parameters
    def tune_parameters_balanced(self):
        oversample = SMOTE()
        undersample = RandomUnderSampler()
        steps = [('o', oversample), ('u', undersample)]
        pipeline = Pipeline(steps=steps)
        x_train_res, y_train_res = pipeline.fit_resample(self.x_train, self.y_train)
        cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
        grid_search = GridSearchCV(estimator=self.algorithm, param_grid=self.parameters, cv=cv)
        grid_search.fit(x_train_res, y_train_res)
        print(grid_search.best_params_)
        return grid_search.best_params_

```

parameters_tunning.py