Investigation of Machine Learning Techniques for Fault Diagnosis in the Semiconductor Manufacturing Process

Abubakar Abdussalam Nuhu

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Eastern Mediterranean University
February 2021
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Mechanical Engineering.

Prof. Dr. Hasan Hacisevki
Chair, Department of Mechanical Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Mechanical Engineering.

Assoc. Prof. Dr. Qasim Zeeshan
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Shaban Ismael Albrka

2. Assoc. Prof. Dr. Qasim Zeeshan

3. Asst. Prof. Dr. Mohammed Bsher A. Asmael

ABSTRACT

Industries are going through the fourth industrial revolution (Industry 4.0), where technologies like the Industrial Internet of Things (IIoT), Big Data Analytics and Machine Learning (ML) are being extensively employed for improving the productivity and efficiency of manufacturing systems. Recently, many researchers have demonstrated the ability of ML algorithms to meet various challenges presented by the next generation Smart Manufacturing Systems (SMSs). This work aims to investigate the applicability of several machine learning techniques for early fault diagnosis towards smart manufacturing process. Thus, in this thesis, we propose several fault diagnosis ML models for SMSs applications. A case study has been conducted on a dataset from a semiconductor manufacturing process. However, this dataset contains missing values, redundant and noisy features, and class imbalance problem. This imbalance problem makes it so difficult to accurately predict the minority class, due to the majority class size difference. Therefore, this work proposes and compares the effects of three synthetic data generation techniques to handle such class imbalance problem. To handle issues related to missing values and redundant features, we implemented and compared the performance of two missing values imputation techniques and two feature selection techniques using three adopted data synthetic generation techniques. We then developed and compared the performance of ten predictive machine learning models against the abovementioned proposed approaches. Experimental results across seven evaluation metrics of performance obtained from these models were significant. These results and a comparative analysis show the feasibility and validate the effectiveness of these proposed synthetic data generation techniques and the proposed methodologies. Some among the proposed

methodologies could produce an accuracy in the range of 99.9% to 100%. Furthermore, a comparative analysis has been conducted with similar models proposed in the literature. Based on the results, our proposed models outpace those proposed in the literature.

Keywords: Semiconductor Manufacturing Process, Fault Diagnosis, Imbalance Dataset, Synthetic Data Generation, Machine Learning.

ÖZ

Endüstriler, Endüstriyel Nesnelerin İnterneti (IIoT), Büyük Veri Analitiği ve Makine Öğrenimi (ML) gibi teknolojilerin üretim sistemlerinin üretkenliğini ve verimliliğini artırmak için yoğun bir şekilde kullanıldığı dördüncü endüstriyel devrimden (Endüstri 4.0) geçiyor. Son zamanlarda, birçok araştırmacı, ML algoritmalarının yeni nesil Akıllı Üretim Sistemleri (SMS'ler) tarafından sunulan çeşitli zorlukları karşılama becerisini göstermiştir. Bu çalışma, akıllı üretim sürecine yönelik erken arıza teşhisi için çeşitli makine öğrenme tekniklerinin uygulanabilirliğini araştırmayı amaçlamaktadır. Bu nedenle, bu tezde, SMS uygulamaları için çeşitli arıza teşhis ML modelleri öneriyoruz. Yarı iletken üretim sürecinden bir veri seti üzerinde bir vaka çalışması yapılmıştır. Bununla birlikte, bu veri kümesi eksik değerler, fazlalık ve gürültülü özellikler ve sınıf dengesizliği problemini içermektedir. Bu dengesizlik sorunu, çoğunluk sınıf büyüklüğü farkı nedeniyle azınlık sınıfını doğru bir şekilde tahmin etmeyi çok zorlaştırıyor. Bu nedenle, bu çalışma, bu tür bir sınıf dengesizliği sorununu ele almak için üç sentetik veri oluşturma tekniğinin etkilerini önermekte ve karşılaştırmaktadır. Eksik değerler ve gereksiz özelliklerle ilgili sorunları ele almak için, benimsenmiş üç veri sentetik oluşturma tekniğini kullanarak iki eksik değer atama tekniğinin ve iki özellik seçim tekniğinin performansını uygulayıp karşılaştırdık. Daha sonra on tahmine dayalı makine öğrenimi modelinin performansını yukarıda belirtilen önerilen yaklaşımlarla geliştirip karşılaştırdık. Bu modellerden elde edilen performansın yedi değerlendirme metriğine ilişkin deneysel sonuçlar anlamlıydı. Bu sonuçlar ve karşılaştırmalı bir analiz, bu önerilen sentetik veri oluşturma tekniklerinin ve önerilen metodolojilerin uygulanabilirliğini gösterir ve etkililiğini doğrular. Önerilen metodolojilerden bazıları, % 99,9 ila% 100 aralığında

bir doğruluk sağlayabilir. Ayrıca, literatürde önerilen benzer modellerle karşılaştırmalı bir analiz yapılmıştır. Sonuçlara göre, önerilen modellerimiz literatürde önerilenleri geride bırakıyor.

Anahtar Kelimeler: Yarıiletken Üretim Süreci, Hata Teşhisi, Dengesizlik Veri Seti, Sentetik Veri Üretimi, Makine Öğrenimi.

Dedicated to my late beloved GRANDPARENTS; may their souls Rest in JANNAH…

ACKNOWLEDGMENTS

This piece of work is made possible with the support and contribution of many individuals. So much has been contributed by some that they have to be appreciated, their names have to be mentioned, and they deserve recognition. Leading the list is my supervisor and a 'substitute father' ***Assoc. Prof. Dr. QASIM ZEESHAN,*** *that* played a prominent role during my time of studies, in my life at large, and in this work. Thanks for the countless hours of counseling, guidance, and motivation. You have "turned us on, turned us around, and boosted us up" to believe in ourselves, to believe that we can do and achieve so much so in this life when we put ourselves to it. So much has been learned from you. I am deeply grateful to you, endlessly I will be. Scholastically, you are such a role model! JAZAKALLAH AL KHAIR. Many thanks to you, ***ABDULLAHI ISMAIL, MSc***. Without your support, this work would not have been as great as it is. Thanks for the intro to the field of machine learning, I have learnt a lot and you have given me the key to this field at large. I am deeply grateful.

I am also thankful to the department of mechanical engineering and its entire staff members for granting me such an opportunity, a friendly and conducive environment to carry out this study. Especially, *Assoc. Prof. Dr. Qasim Zeeshan Assist. Prof. Dr. Mohammad Bsher A. Asmael, Assist. Prof. Dr. Devrim Aydin, Depart. Chair Prof. Dr. Hasan Hacisevki,* and *Assist. Prof. Dr. Babak Safaei.*

I am acknowledging the members of my Jury including *Assoc. Prof. Dr. Qasim Zeeshan, Assist. Prof. Dr. Mohammad Bsher A. Asmael,* and *Assoc. Prof. Dr. Shaban*

Ismael Albrka Ali for their comments and guidance that made this thesis greater. I am extremely thankful.

A very special thanks to my wonderful colleagues for their support, guidance, and advice over the years, most especially *Mr. Omer A. Kalaf* – you were such a great brother and office mate, *Mr. Mohammed Y. Alibar, Mr. Tauqir Nasir,* and *Mr. Hussain Ali Faraj M.,* I am grateful in every possible way to you all. To my friends*; Naziru Aliyu (AH)* – what a true friend and SA, *Umar Lawan* – DH, *Yazid Mustapha, Abdul Hakeem A. A.* – you helped me so much during the time of my studies, *Umar Saleh* – my proofreader, my great roomie *Sani M. Ahmad* – four years with you were just great, and *Sadisu Usman,* I thank you all and deeply appreciate your presence and support in my life.

Last but not the least, I thank my parents and family for your immeasurable love, immense prayers, and endless support. I thank ALLAH for being part of you and gifting me you in this gracious world. May HE the lord of the worlds grant you good here and hereafter.

To all that I have acknowledged, please accept my earnest thanks. To all you whom I have not mentioned, please just know that you are appreciated more than you could imagine, even though you are not mentioned in this piece of work.

Hello Reader; the fact that you are reading this work deserves thanks. I thank you.

God Bless and have a Great Forever !

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1NNC	1-Nearest Neighbor Classifier
ADASYN	Adaptive Synthetic Oversampling
ANN	Artificial Neural Network
BSMOTE-SVM	Borderline-SMOTE SVM
CCPR	Control Chart Pattern Recognition
CNN	Convolutional Neural Network
CV	Cross Validation
DT	Decision Tree
FPR	False Positive Rate
FS	Feature Scaling
GBM	Gradient Boosting Machine
k-NN	k-Nearest Neighbour
k-NNI	k-NN imputation
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MI	Mean imputation
MLP	Multilayer Perceptron
MRD	Modified Raw Dataset
PCA	Principal Component Analysis
RF	Random Forest
RFE	Recursive Feature Elimination
SDGT	Synthetic Data Generation Technique
sEMG	Surface Electromyography

SMOTE	Synthetic Minority Oversampling Technique
SVM	Support Vector Machines
TPR	True Positive Rate
UFS	Univariate Feature Selection

Chapter 1

INTRODUCTION

1.1 Background

Due to the advancement in manufacturing and manufacturing technologies plus the change of the global economic landscape, Smart Manufacturing Systems (SMSs) have become a general solution for both developed and developing countries to upgrade their manufacturing industries. Also, with the emergence of Industry 4.0 (I4.0), smart systems, Machine Learning (ML), predictive model are being applied extensively in manufacturing areas for monitoring the equipment status of industrial systems [1]. Moreover, the concept of prognostics and health management have become unavoidable trends in the framework of industrial big data and SMSs. I4.0 gives a reliable solution for monitoring equipment health status in industries.

Recent, many manufacturing systems are equipped with sensors, algorithms, technologies, and advanced methods in order to facilitate real-time monitoring of the production processes and to collect and extract data because they cannot anymore be processed using traditional technologies. I4.0 and its key technologies play an essential role in making industrial systems autonomous, hence enabling automatized big data collection from industrial machines/components [1]. Production-state and equipment-state sensors collect data that provides opportunity for efficient control and optimization. Unfortunately, such measurements of variables data formats from multiple sensors can be so overwhelming and poses a challenge for manufacturing data

analysis, and timely detection of any fault during the production process can be difficult [2]–[4].

Big data collected for ML contains very useful information and valuable knowledge that can improve the whole productivity of manufacturing processes and system dynamics. It can also be applied into decision support in several areas, such as manufacturing, maintenance and health monitoring [5]. Based on a collected data, a suitable ML models can be developed and be applied for automatic fault detection and diagnosis. The enormously available data generated from industries, ML techniques have been broadly applied in areas such as computer science, smart manufacturing systems and processes, and predictive maintenance of industrial systems [5], [6]. These advancements facilitate the development of manufacturing contexts into integrated networks of automation devices and allow the smart characteristics of being self-sensing, self-adaptive, and self-organizing. However, obtaining such advancements require addressing numerous challenges including data mining, data quality, data volume and merging [7].

1.2 Problem statement

Due to the recent technological advancements, the manufacturing process of semiconductor is becoming more complex, costly, and extremely interdisciplinary processes that involve several of stages [7]–[10]. Therefore, the yield in a semiconductor manufacturing process is affected greatly by several factors, most of which are being monitored using sensors and quality inspections. Managing these factors to enhance the yield has been an endless challenge in this manufacturing process [8], [11]–[13]. Moreover, sensory datasets acquired from semiconductor manufacturing process domain usually might contain outliers, missing values,

redundant and noisy features, and class imbalance problem. This imbalance problem makes it so difficult to accurately predict the minority class, due to the majority class size difference. This imbalance exists as a result of very low rate of finding defective products in manufacturing processes in practice. Also, handling and predicting the quality of a product in a semiconductor manufacturing process is an imbalanced problem. To handle such issue of imbalance, a suitable technique for handling imbalanced dataset needs to be considered to enhance the predictive model performance. To address the issues related to missing values and redundant features, we implement and compare the performance of two missing values imputation techniques and two feature selection techniques versus three propose data synthetic generation techniques.

1.3 Scope and aim of the thesis

The scope of this thesis is within manufacturing systems that uses predictive models for fault detection and diagnosis using machine learning algorithms. Also, this work tends to investigate and improve the efficiency and effectiveness of the predictive models within semiconductor manufacturing processes. The evaluation of the ML algorithms has been conducted by using a case study from a distinctive manufacturing domain using a real-world dataset from a semiconductor manufacturing process. The performance of the developed predictive models has been compared with the existing models in the literature to further analyze their accuracy and reliability. The outcomes of the research have contributed in developing efficient and effective ML prediction models for smart manufacturing processes.

Further exploration towards the applicability of ML algorithms in smart manufacturing systems/processes is the main objective of this study. We achieved this by developing

predictive models based on machine learning algorithms with special focus on manufacturing process.

First, detailed literature review of the relevant studies describing the applications of ML techniques in the field of smart semiconductor manufacturing process has been conducted, to get a more profound knowledge in the analyzed contexts, and to examine and highlight the gap in the existing studies, thus, create a firm foundation for further research.

1.4 Thesis contributions

The semiconductor manufacturing processes are complex, costly, and extremely interdisciplinary processes that involve several stages. Failures in the manufacturing stages may result in faulty products [7]. Consequently, feature extraction and early fault diagnosis are of great importance which can only be achieved through fully investigating the production stages and mining significant manufacturing features involved in the production line. Moreover, early fault diagnosis involves implementing predictive ML model within the manufacturing process for feature extraction and classification to improve the manufacturing process and productivity.

The key contributions of this thesis are as follows:

- Reviewing the applications of ML techniques towards semiconductor manufacturing processes with a special focus on studies reported that utilized the UCI machine learning repository semiconductor manufacturing process dataset.
- Proposal of a methodology for ML predictive models' development that comprises data cleaning, missing datapoints imputation techniques, most

potential features selection techniques, features normalizing technique, synthetic data generation techniques, model training and validation techniques, model development, and model performance evaluation.

- Adoption and comparison of two different missing datapoint imputation techniques including mean and k-NN imputation techniques.
- Adoption and comparison of two different features selection techniques including PCA and univariate features selection.
- Adoption and comparison of synthetic data generation techniques including SMOTE, BSMOTE-SVM and ADASYN for synthetic data generation to handle the class imbalance distribution of the dataset.
- Implementation and comparison of two different validation techniques for evaluating the performance of the developed predictive models.
- An extensive comparisons analysis between the results obtained in this thesis with those similar studies reported in the literature.
- Proposed methodologies and models evaluation on the UCI machine learning repository SECOM dataset.

1.5 Thesis outline

The remainder of the thesis is organized as bellow:

Firstly, in **Chapter 2,** the state-of-the-art theories and general information behind each topic considered in this thesis are presented. Focus is mainly on providing the state-of-the-art definitions, of Smart Manufacturing; various techniques and types of ML, alongside, their classifications and applications towards manufacturing systems. Moreover, the chapter emphasizes the semiconductor manufacturing dataset issues and descriptions used in developing and validating the developed classifier models.

Secondly, **Chapter 3** focuses on presenting some related works done in this field of study. Moreover, the section is aimed at providing recent advancements of ML techniques applied to SMSs found in literature from ample perspectives. The literature studies have been conducted in those areas of interest to further determine and define the current state-of-the-art knowledge, determine the feasibility of the research questions, and the research methodologies that have been applied in this thesis. Thirdly, **Chapter 4** presents the research methodology for ML models selection, development, and evaluation. Moreover, it emphasizes the descriptions of the datasets used, the methodology for data cleaning, features selection techniques, and synthetic data generation method. Then, in **Chapter 5** discussions on the results obtained from experimentation performed in this study are presented while highlighting main contributions of this work. Finally, **Chapter 6** presents the conclusions and future outlooks.

1.6 Summary of the chapter

To conclude, in this chapter a brief background in the advancements within smart manufacturing is presented followed by problem statements. Moreover, the main objective, aims of this work, and the thesis contributions are outlined. Finally, thesis structure is presented.

In the following section, the theories and general information regarding machine learning, smart manufacturing, and semiconductor manufacturing process are presented. Similarly, the issues and challenges within smart semiconductor manufacturing process are drawn. Then, the chapter reports the issues and description of the semiconductor manufacturing process dataset used in developing and validating the proposed methodologies and models in this thesis.

Chapter 2

BACKGROUND INFORMATION

This section provides the theories and general information regarding learning algorithms, smart manufacturing, and semiconductor manufacturing process. Furthermore, the issues and challenges within semiconductor manufacturing process are described.

2.1 Machine learning (ML)

Lately, ML within the contexts of AI [14] has appeared to be one of the most powerful tools that can be applied in several applications to develop intelligent predictive algorithms. It has been developed into a wide field of research over the past decades. ML can be defined as a technology by which the outcomes can be forecasted based on a model prepared and trained on past or historical input data and its output behavior [15]. ML approaches are known to have tremendous advantages, as they have the ability in handling multivariate, high dimensional data and can extract hidden relationships within data in complex, dynamic, and chaotic environments [6], [16], [17]. Selecting the most appropriate, simple and the most efficient ML algorithm could be of a great concern when building the predictive model. Similarly, when selecting the algorithm to use for a particular problem, it is significant to know the difference amongst ML categories and their types as well as their way of training and validation techniques in order to be able to investigate and prepare the data of choice correctly.

2.1.1 Machine learning types and classifications

ML algorithms are characterised into three different types including supervised, unsupervised, and reinforcement learning [6], [18], [19]. As stated in [6], different algorithms can be combined together in order to maximize the classification power. To add on, some among the ML algorithms are both applicable to unsupervised and supervised learning. Figure 1 shows the types and categorisation within ML algorithms. Also, they are categorized into three categories including classification, regression, and clustering.

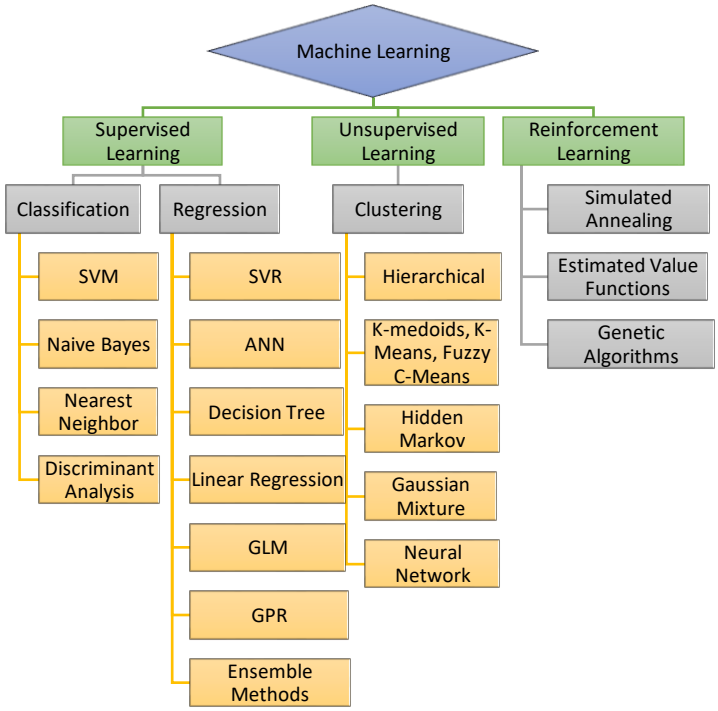

Figure 1: Classifications within Machine Learning Techniques.

ML algorithms usually require collecting huge amount of data of the failure status scenarios and the health conditions scenarios for model training. ML algorithm development covers the historical dataset selection, dataset preprocessing, model selection, training and validation, as shown in Figure 2.

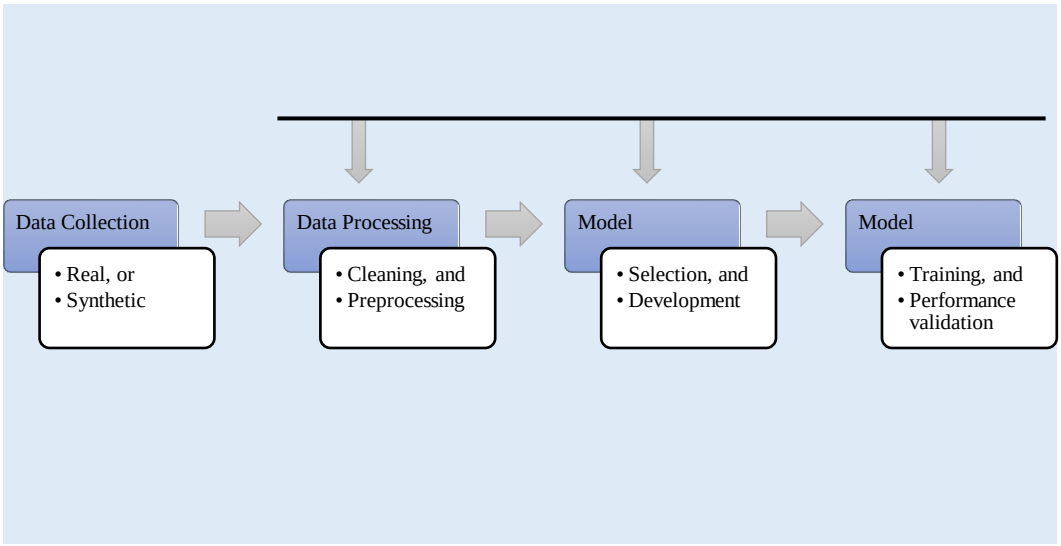

Figure 2: Main steps involved for ML model development.

2.1.1.1 Multi-layer perceptron (MLP)

MLP network is a supervised learning algorithm and a common type of ANN. MLP is one of the most common examples of feed-forward neural networks that has been applied in several practical applications [20], [21] MLP is considered as an efficient method of capturing non-linear relationships between the model parameters [21]. MLP consists of three layers as can be seen from Figure 3. These layers include; Input layer, Hidden layer, and Output layer [22]. MLP parameters including weight and bias values determine its outputs. Training process of MLP has to do with the creation of relationships between outputs the corresponding to inputs [23].

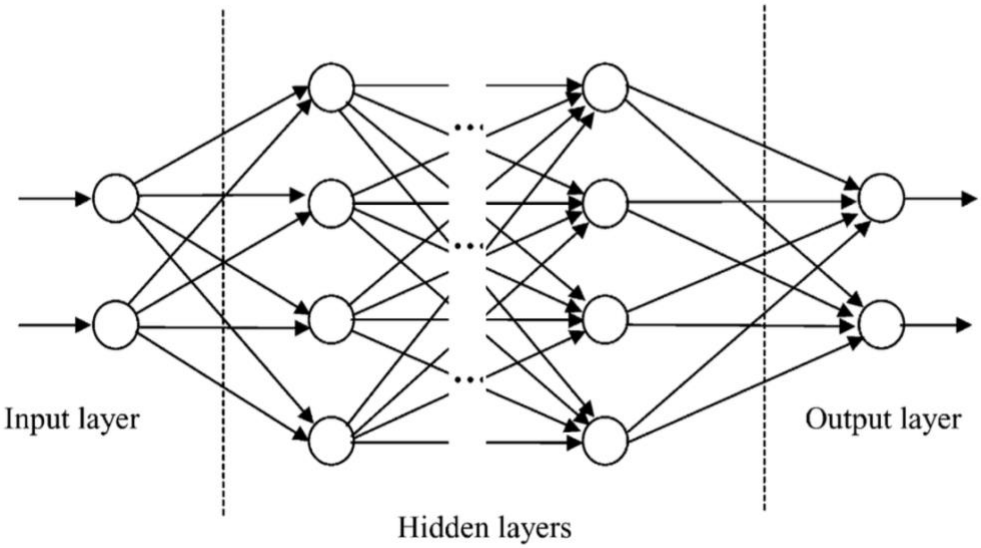

Figure 3: Multi hidden layer MLP [24].

2.1.1.2 Support vector machine (SVM)

SVM is a well-known ML technique which is widely used for both classification and regression analysis, due to its high accuracy [17], [25], [26]. SVM is defined as a statistical learning concept with an adaptive computational learning method. SVM learning algorithm is presented in Figure 4. SVM learning technique employs input vectors to map nonlinearly into a feature space whose dimension is high [27]–[29]. SVM is a supervised ML technique that can perform pattern recognition, classification, and regression analysis.

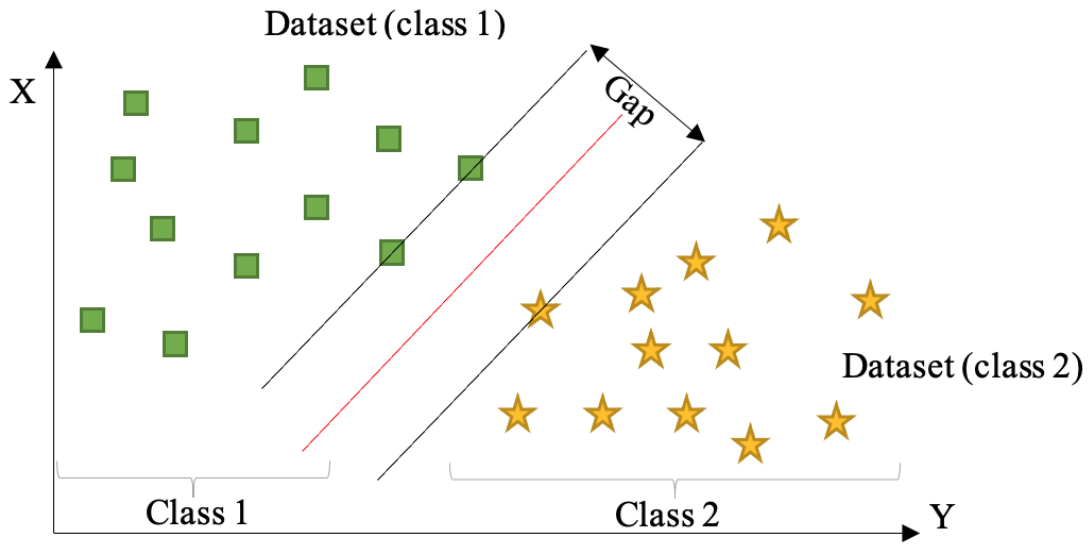

Figure 4: Support vector machine algorithm.

2.1.1.3 Random forest (RF)

RF was developed by Breiman, L. [30]. This is an ensemble learning algorithm made up of several DT classifiers, and the output category is determined collectively by these individual trees. When the number of trees in the forest increases, the fallacy in generalization error for forests converges. There are also important benefits of the RF. For example, it can manage high-dimensional data without choosing a feature; trees are independent of each other during the training process, and implementation is fairly simple; however, the training speed is generally fast and, at the same time, the generalization functionality is good enough [5]. Random forest algorithm for machine learning has tree predictions, and based on tree predictions, the RF provides random forest predictions [31]. The RF model is visualized in Figure 5.

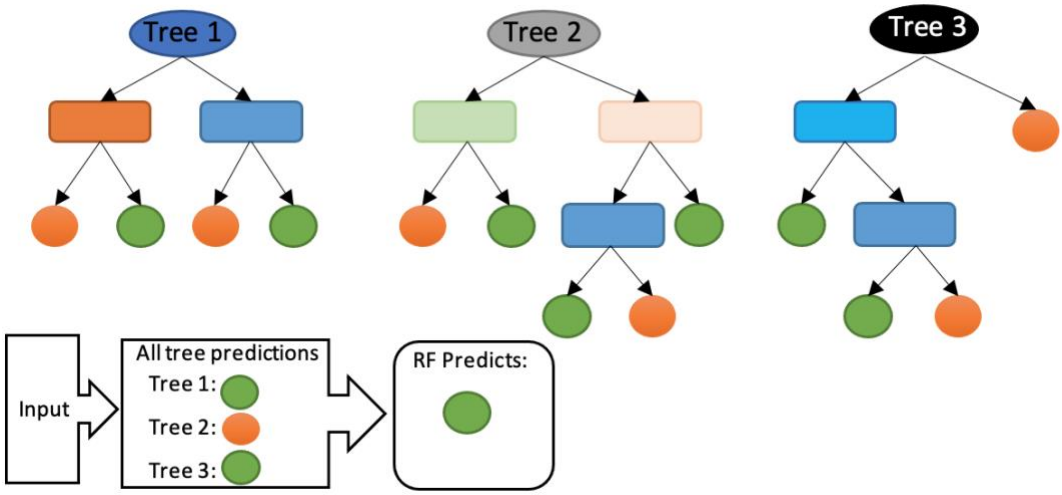

Figure 5: Random forest.

2.1.1.4 Logistic regression (LR)

LR can be used to estimate the categorical variations with a given set of independent variables [32]. Figure 6 graphically illustrates the working principal of LR.

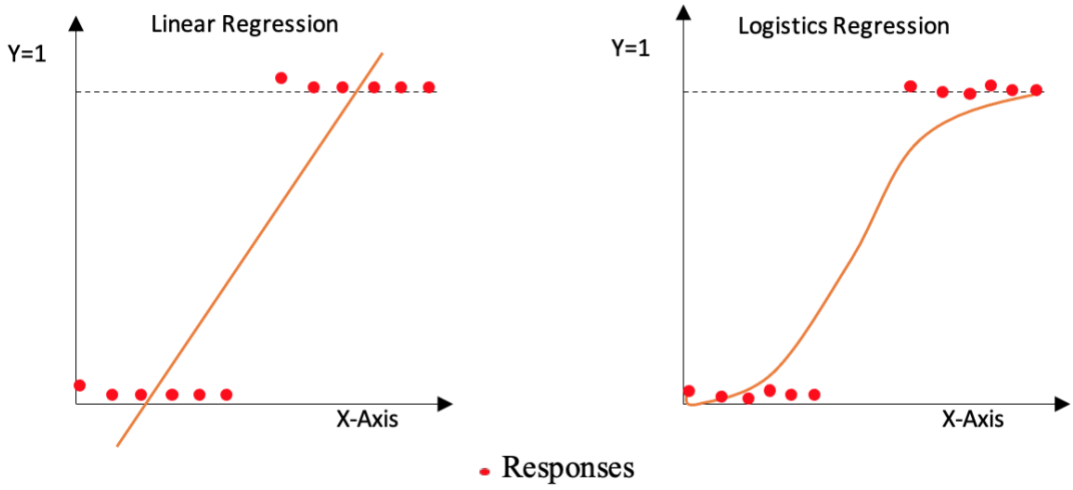

Figure 6: Linear and logistic regression.

2.1.1.5 Extreme gradient boosted trees (XGBoost)

XGBoost was developed by Chen, T. & Guestrin, C. [33], a scalable tree boosting system that is widely used by data scientists and provides state-of-the-art results on

many problems. Figure 7 presents the XGBoost model tree. XGB classification trees can be able to not only reveal the significant of dataset features but to also develop a robust classification model. XGBoost involves the development of multiple ensemble of weaker trees (small trees) [2].

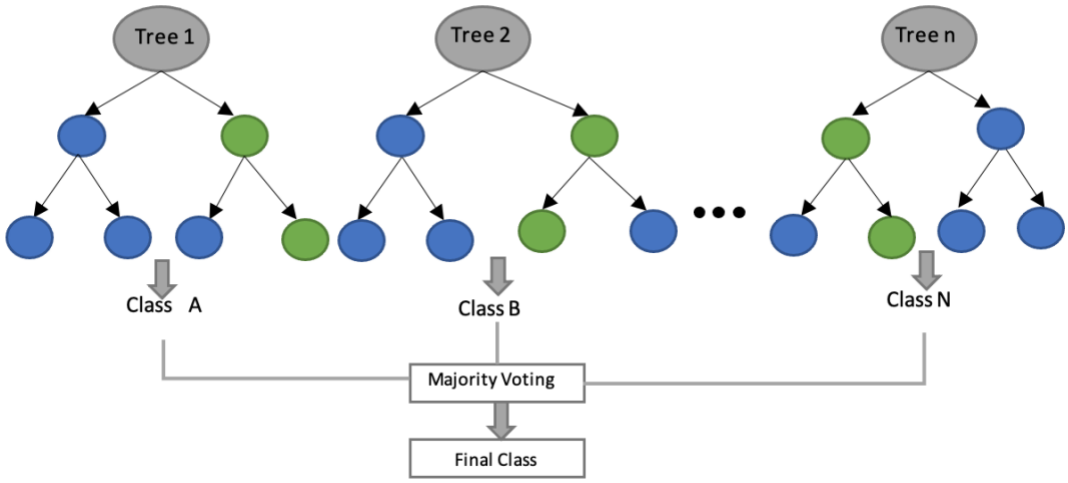

Figure 7: XGBoost algorithm tree.

2.1.1.6 Gradient boosting trees (GBT)

Boosting algorithms are methods that repetitively add several simple classification models known as weak learners to build a complex classification model with higher accuracy [34][35]. GBM achieves this by using a gradient optimization algorithm to reduce the loss function or extent of error [36]. The regression tree and the gradient boosting are combined into decision trees, with appropriate trimming. The algorithm consists of multiple decision trees, with each tree gradient down by learning from the n – 1 number of trees [36].

2.1.1.7 Decision tree (DT)

Decision Tree is a network system composed primarily of nodes and branches, and nodes comprising root nodes and intermediate nodes. The intermediate nodes are used

to represent a feature, and the leaf nodes are used to represent a class label [27]. DT can be used for feature selection [37]. DT algorithm is presented in Figure 8.

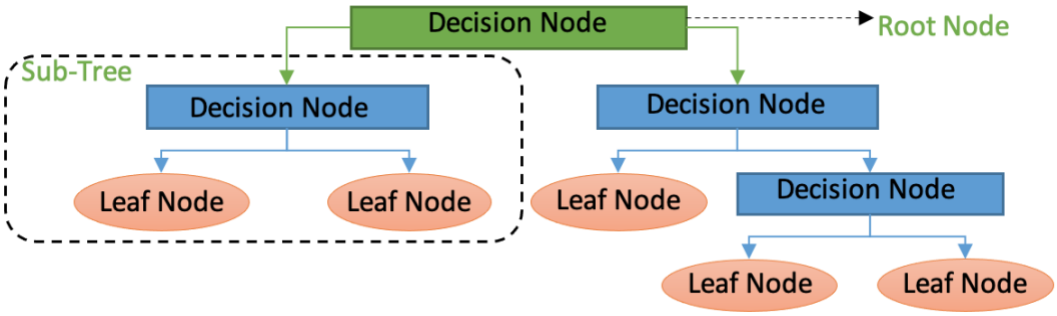

Figure 8: Decision tree algorithm.

DT classifiers have gained considerable popularity in a number of areas, such as character identification, medical diagnosis, and voice recognition. More notably, the DT model has the potential to decompose a complicated decision-making mechanism into a series of simplified decisions by recursively splitting covariate space into subspaces, thereby offering a solution that is sensitive to interpretation [39].

2.1.1.8 Linear discriminant analysis (LDA)

Linear discriminant analysis (LDA) is a generalization of Fisher's discriminant method used in statistics, pattern recognition and machine learning to find a linear combination of features that separates two or more classes of objects or events [40]. The resulting combination may be used as a linear classifier, or, more commonly, for dimension reduction before classification. LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data according to a defined objective [41]. Linear Discriminate Analysis (LDA) is one of the robust machine learning algorithms among the popular classifiers. The LDA projects the feature in the most discriminative ways for identification process [42]. LDA is a pattern recognition

method providing a classification model based on the combination of variables that best predicts the category or group to which a given compounds belongs. The basic theory of LDA is to classify the dependent by dividing an n-dimensional descriptor space into two regions that are separated by a hyperplane defined by a linear discriminant function [43]. The problem of finding the optimal be mathematically represented as the following maximization problem [44].

2.1.1.9 Adaptive boosting (AdaBoost)

AdaBoost, short for "Adaptive Boosting", is one of the ensembles boosting classifiers. AdaBoost is an iterative ensemble ML technique can that combine several ML classifiers to increase the classifier prediction accuracy and performance (see Figure 9). Also, it produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees, with each new model attempting to correct for the deficiencies in the previous model [45]. It has been widely used in classification and regression for its capability to improve learning quality of weak learning algorithms [46][47][48].

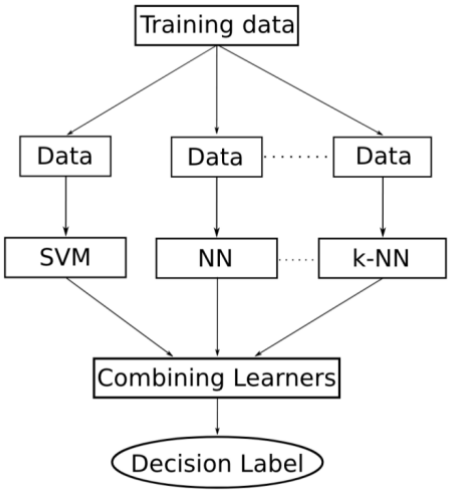

Figure 9: AdaBoost [49].

2.1.1.10 Naïve bayes (NB)

Naïve Bayes (NB) is among the simplest classifiers. It is based on the Bayes' theorem with independence prediction [42], [50]. It estimates the most likely anticipated class by analysing the probability of test features and it performs well at measuring the density a dataset [42]. Mathematically Bayes' theorem is stated as [51];

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \quad (1)$$

Where, P(A) is the probability of A, P(B) is the probability of B, P(A|B) is the conditional probability of A given B, and P(B|A) is the conational probability of B given A. When training naïve Bayes algorithm, the Bayes theorem provides a way of determining the posterior probability, $P(c|x)$, from $P(c)$, $P(x)$, and $P(x|c)$. Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence [49].

$$P(C|x) = \frac{P(x|C) \times P\ (C)}{P(x)} \quad (2)$$

Where, $P(c|x)$ is the posterior probability of *class* (*target*) given *predictor* (*attribute*). $P(c)$ is the prior probability of *class*. $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*. $P(x)$ is the prior probability of *predictor*.

2.2 Smart manufacturing and machine learning

In most of the manufacturing processes, product quality, cost and the time to deliver are the main key features for enterprises to attain long-term competition. Process engineers must be able to point out abnormalities in peculiar products features during the processes of manufacturing [4]. Nowadays, modern technologies in manufacturing and smart manufacturing systems enable to collect data measurements from the equipment sensors in real time process control, as it is very hard using traditional

process control techniques to control or monitor hundreds of processing stages within the manufacturing systems. With such high volume of data collected throughout the manufacturing processes or systems, effective monitoring and optimal process control can be carried-out by the process engineers by investigating and analyzing these datasets so that the monitoring would be much easier.

SMSs are defined as fully-integrated systems and as collaborative manufacturing systems that can respond in real-time to meet customers need, changing demands, and conditions in the factory and supply network [52]. SMSs aim at integrating big data, Industrial Internet of Things (IIoT), advanced analytics, and high-performance computing, into conventional manufacturing processes and systems to produce highly customizable products at very low cost and with higher quality, [53]. Moreover, the synthesis of those advanced technologies and the manufacturing capabilities can increase the he agility, productivity, and sustainability of SMSs [54]. The rapid developing AI technologies, ML and DL in particular, are one of the promising tools that further boost these SMSs industries [55]. In the environment of Internet of Things (IoT), Industry 4.0, Big Data, cloud computing, and other advanced technology provides great support in the growth of intelligent manufacturing [56]. Several algorithms and techniques have been developed such as ML algorithms to learn valuable information from the data produced in manufacturing sectors and to make the manufacturing smarter [57].

2.2.1 Semiconductor manufacturing process

With no intention of completeness, this section describes the fabrication process of semiconductor wafer. Interested reader is directed to [58] and a PhD thesis [59] for a detailed description.

The semiconductor industry is arguably at the forefront of the most cost intensive technological advanced industries. The pervasive nature of the semiconductor devices implies that they are widely used in every segment of our lives. With ever increasing demand of semiconductor devices, addressing production related problems and increasing output in the industry is getting more complex [60]. Over the past decades, developments in semiconductor technology have made electronic devices smaller, faster, cheaper, reliable and more advanced to handle huge amount of data with higher degree of complexity[61]. Semiconductor is often referred to as integrated circuit/microchip that is made from silicon/germanium. Semiconductor is a basic building blocks that is used to make electronic devices [61]. Figure 10 shows a typical silicon semiconductor wafer.

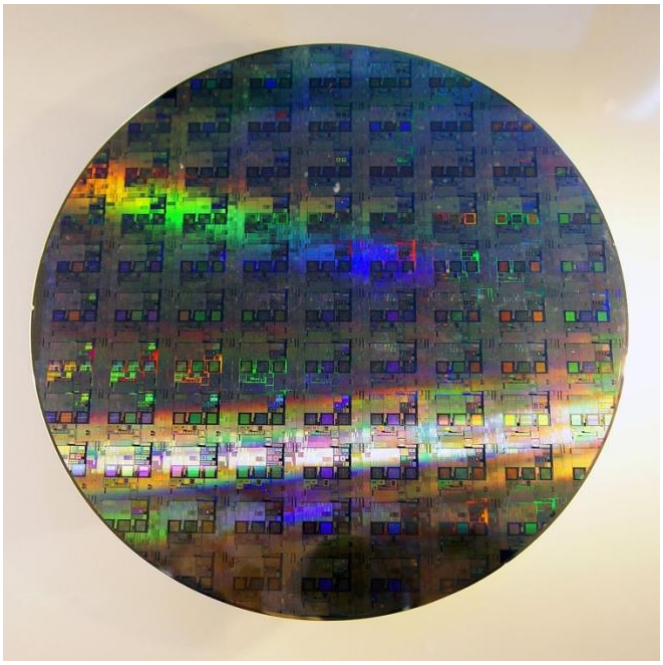

Figure 10: A typical silicon semiconductor wafer [62]

In the manufacturing of semiconductors, it entails numerous simultaneous processes involving several inter-operating machineries [63], [64]. Typically, more than 500-steps are required to fabricate each semiconductor wafer [65]. The sequence of

semiconductor manufacturing process involves the following main steps (Figure 11): the production of silicon wafers, integrated circuits fabrication onto the silicon wafers, assembly by putting the integrated circuit inside a package to form a ready-to-use product, and testing of the finished products/yield [4], [8], [12]. Semiconductor wafer fabrication process involves numerous complicated processes, such as oxidation, photolithography, cleaning, etching, and planarization, many among these processes are executed repeatedly. These basic concepts of semiconductor wafer fabrication are shown and explained in Figure 12.

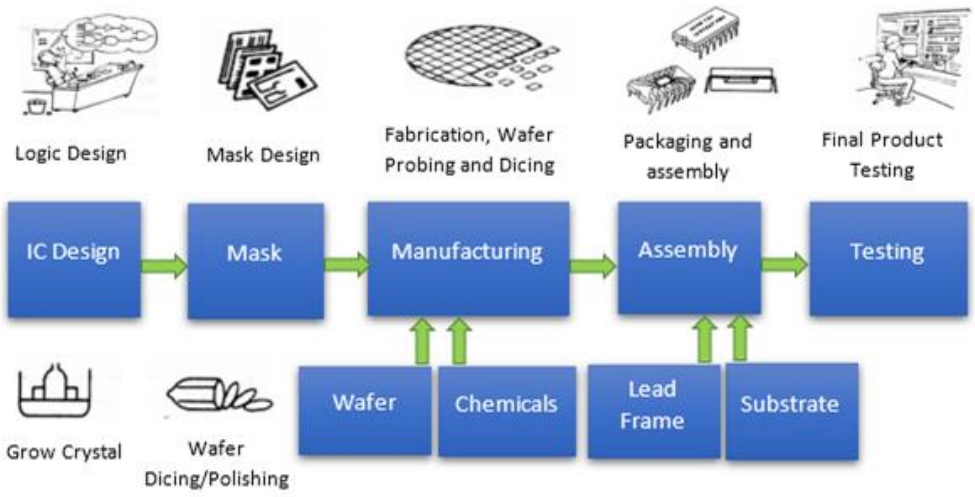

Figure 11: Overview of the main steps involved in the SECOM process [61].

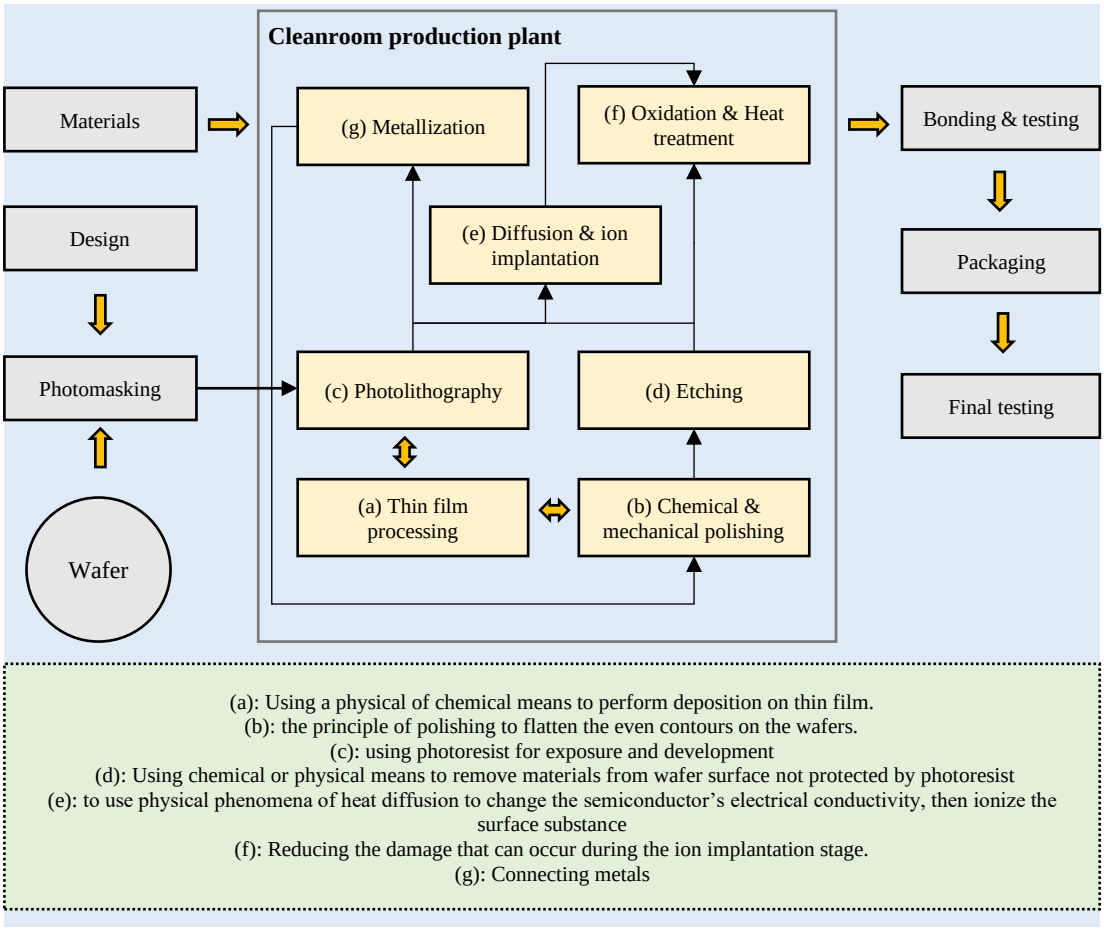

Figure 12: Overview of the detailed and explanation of SECOM fabrication process, adapted from [66].

2.2.1.1 SECOM dataset description

Equipment embedded with production sensors used in semiconductor fabrication process generates immense amounts of data in real-time. The amount of generated data is so overwhelming such that it makes preemptive detection of production faults is difficult to achieve [60]. The SECOM dataset analyzed in this study is gotten from the UCI machine learning repository [67]. The data has been analyzed with the methodology proposed in the previous section.

Semiconductor Manufacturing (SECOM) dataset is a public dataset [68] that is acquired from semiconductor manufacturing process. Hence, it has been used as a benchmarking dataset for assessing predictive ML models in the context of smart

manufacturing [2]. The data contains both semiconductor quality and manufacturing operations datasets. The dataset represents a selection of process related data taken from a production line. Within the production line there are several major checks points for in house line testing to ensure product functionality [64].

The dataset contains 591 features, among these attributes/features there is one response attribute that classifies the product as if it pass or fail the quality test. The data is composed of 1567 instances, each instance is recorded after the product has been tested as if it pass or fail the quality test using 590-sensor measurements, i.e., 1567 x 591 matrix. Amongst these 1567 instances, 104 where been classified as 'fail class' encoded as 1, whereas the rest have been classified as 'pass class' encoded as -1, see Table 1 and Figure 13A.

Table 1: SECOM dataset description.

#	**Raw data**
Number of Features	590
Number of classes (pass and fail)	2
Number of 'pass' instances	1463
Number of 'fail' instances	104
Total number of instances	1567

The following insights have been drawn after analyzing the SECOM dataset:

- The class imbalance distribution among the two classes with a ratio of 1:14 (failed to passed classes) is of a great concern. Because, the imbalance is a big issue when it comes to classification algorithms as some algorithms cannot handle such issue, thus, this issue of data imbalance has to be dealt with in order to develop a model that can classify the two classes without miss classifying the minority class as the majority class.

- Feature selection has to be carried out, because some of the recorded features might not be useful in developing the predictive model as some recorded no values and some have unique values.
- Approximately, 6.64% of the dataset are missing (see Figure 13B, the yellow shaded parts). We could say this missingness in the dataset is unknown. It could be due to the sensors or they were never measured. However, this missingness has to be dealt with using appropriate methods as some of the algorithms do not efficiently work with missing values.

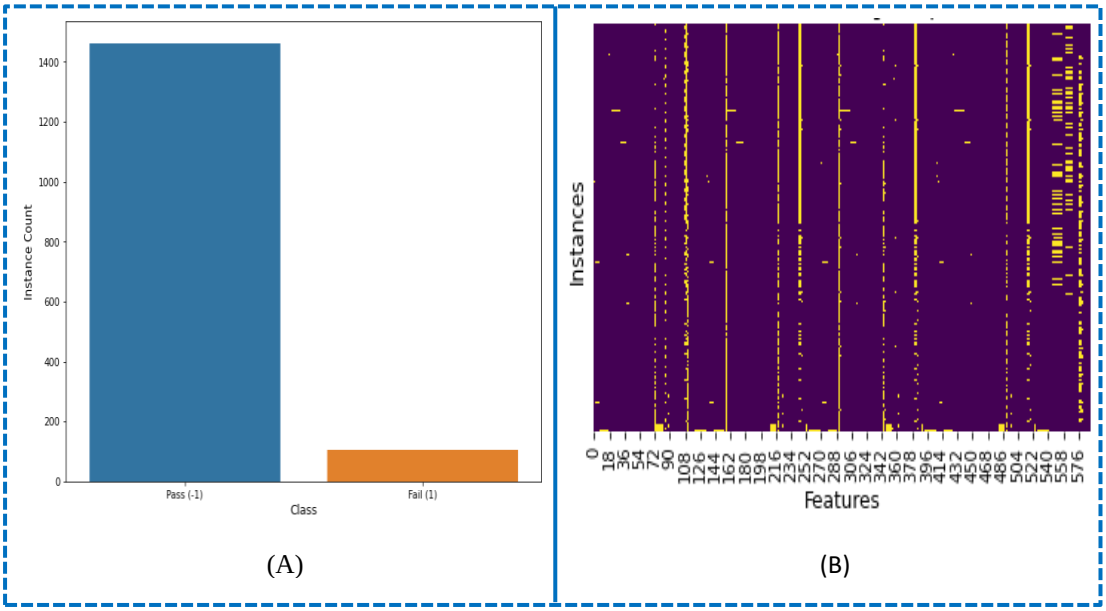

Figure 13: SECOM dataset description: (A) Instances distribution within the two classes. (B) Missing vs observed datapoint values.

2.3 Summary of the chapter

This chapter reports the general background info on machine learning, smart manufacturing and semiconductor manufacturing processes. The definition of machine learning, its types and the categories are discussed. Moreover, issues and challenges in the semiconductor manufacturing processes are detailed. Lastly, the chapter reported the issues and description of the semiconductor manufacturing

process dataset used in developing and validating the proposed methodologies and models in this thesis.

The next chapter reports the recent advancements in smart manufacturing with a main focus on semiconductor manufacturing processes. In the chapter, applications of numerous different techniques of machine learning within smart semiconductor manufacturing processes are reported and discussed. The chapter categorized the ML algorithms based on the ML technique considered, data preprocessing technique considered, feature selection technique considered, synthetic data generation technique considered, the model validation technique considered, model evaluation metrics considered, and the respective study key findings. The literature studies have been conducted in those areas of interest to further determine and define the current state-of-the-art knowledge, determine the feasibility of the research questions, and the research methodologies that have been applied in this thesis.

Chapter 3

LITERATURE REVIEW

From a comprehensive perspective, this thesis pinpointed and categorized the ML algorithms based on the ML technique considered, data preprocessing technique considered, feature selection technique considered, synthetic data generation technique considered, the model validation technique considered, model evaluation metrics considered, and the respective study key findings.

3.1 ML techniques applied to SMSs

SMSs, makes available of operational performance data that was previously not available for performance management. At one time, such data when collected, was used mostly for production control, or not collected at all [69]. Machine learning techniques, as emerging techniques, are been explored for a broad range of SMSs recently. They are been investigated widely in different stages of manufacturing including concept, design, operation, production, evolution, and sustainment [70] as shown in Figure 14. In this work, we have investigated the application of ML techniques applied to ‘materials, processing and manufacturing’.

Many manufacturing problems belong to the class of classification problems where the industrial domain experts are requested to assign a class to an object or dataset according to the state of the parameters of that object. Based on the experience made in this field, faults happen quite often in the process of production of any kind. Not being able to detect and correct those faults means increase of production costs and it

could even be a reason for production delay or complete standstill. These reasons led to in-creased interest of industry for machine learning techniques as a most efficient way to develop an expert knowledge-based system.

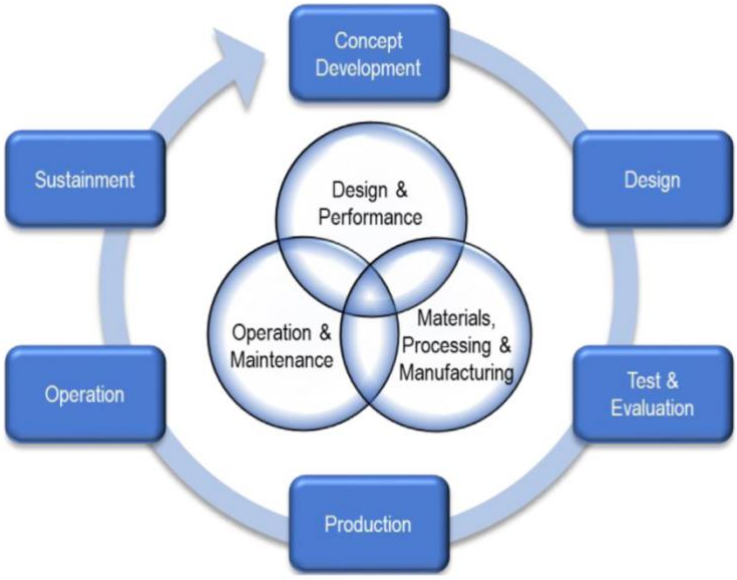

Figure 14: Application scenarios of ML applied to SMS [70].

In [71], data mining techniques and their applications to manufacturing are investigated. The review covers the different categories of production processes, operations, fault detection, maintenance, decision sup-port, and product quality improvement [70]. Application schemes of ML in manufacturing are identified and summarized in [72], where they introduce the data quality problem in the context of supply chain management (SCM) and propose methods for monitoring and controlling data quality. Evolution and future of manufacturing are reviewed in [73], where in the study they highlighted the significance of data modelling and analysis in manufacturing intelligence. A study on big data analytics modeling in metal cutting industry has been reported in [74]. Major key technologies, overall concept of Smart Manufacturing, the key system structure, and each key technology were investigated

and reported in [75]. Moreover, the work identified and predicted the trends and future of Smart Manufacturing by conducting various analyses on the application areas and technology development levels. The advancements and development of machine learning in smart manufacturing is reviewed in [76].A comparative study was reported by [77] on ML algorithms for smart manufacturing (SM) of tool wear prediction using RF. The study introduced a RF-based prognostic technique for prediction of a tool wear, then the performance of the RF algorithms with feed-forward back propagation (FFBP) were compared to the SVR and ANN algorithms. Results from the conducted experiment have shown that, RF-algorithms outperformed SVR and FFBP ANNs with single-hidden layer in tool wear prediction. The algorithms were evaluated with experimental dataset collected from 315 milling tests, R-squared, training time, and mean squared error were measures of the performance.

[78] reported a comprehensive review of machine learning techniques used in manufacturing diagnosis. The study considered 20 articles published within the range of 2007 to 2017. Moreover, they only focused on the applications of four machine learning techniques applied to manufacturing diagnosis namely; ANN, Bayesian networks, SVM, and hidden Markov model.

Anomaly detection algorithm was proposed by [79] that can be used successfully to detect defects in nanofibrous materials. Conducted experiments on a sizable dataset of scanning electron microscope (SEM) images confirmed that, effectively tiny defects can be detected by the proposed algorithm, similarly, in a reasonable time the algorithm can processes images. Therefore, the proposed algorithm can be applied in SMS for nanofibrous material production, to control the quality of the produced material by spot checks. These checks allow to adjust the production process

parameters and, when regularly performed, to raise alerts when the production quality falls below a desired standard, yielding both economic and environmental advantages. The proposed method has been validated successfully over 45 images acquired from samples produced by a prototype electrospinning machine [80] proposed a framework for prognostics and health management applications toward SMS. In the study a detailed survey was carried out to gather the existing studies that deal with maintenance strategies and system failures in the field of SMS. Multi-agent reinforcement learning, and Deep Q-network (DQN) a new reinforcement-learning technique, have been proposed by [81] which can accommodate the characteristics of SM marketplace.

A comparative study was reported by [82] on Deep Learning (DL) method of H20 ML framework and another ML methods from Microsoft Azure where Letter Recognition Data Set from UCI Machine Learning Repository was utilized. The aim was to present the possibilities of DL algorithms and their applications in industrial environment. Moreover, [83] reported a comprehensive review survey in SM of commonly used DL algorithms and discussed their applications in that area. The evolvement of DL technologies and their advantages over conventional ML algorithms was discussed. Consequently, computational techniques based on DL are presented, particularly those that aim to improve the system performance in SM sectors. They concluded that DL provides advanced analytics and offers great potentials to SM in the age of big data.

Cost-sensitive CNN model was proposed by [84] for CCPR problem. The architecture of the proposed model was tuned for several abnormal pattern recognition problems. The model was compared with the standard/traditional and existing CNN models and

discover the robustness of cost-sensitive CNN for highly-imbalanced problem. Experimental studies were conducted using both real-world and simulated datasets.

A cost-effective SVM-based model has been developed by [85] for automated QMC system and was installed at the door-trim manufacturing process using the kiosk. A case study was performed and shows that the proposed methodology using the cost-effective SVM-based model can efficiently evaluate the real-time product quality with minimum number of defective parts by type-II errors. The proposed model can be applied as a complementary or an alternative tool for the conventional/semi-automated/manual QMC systems.

The effectiveness of application of the Statistical Learning Theory (SLT) to MS is illustrated by aiming to develop a predictive model for quality forecasting of products on an assembly line [86]. In their work they targeted to deliver a summary of the pros and cons of the SLT framework. The proposed SVM with a linear formulation was applied to a case study on an assembly line, where the model functions to provide a binary value, in the smallest-time possible, which then gives an alert if the final product fails to reach the required quality constraints. They concluded that SLT model has proven to be an effective method in several other applications such as density estimation, regression, multi-class classification, etc. Similarly, the proposed SLT model can be applied to other settings of MS.

An extensive review on ML applications in manufacturing has been reported by [60]. Where in the study the main focused was on reviewing ML applications towards SECOM production processes and assembly lines. In their paper relevant studies

describing the applications of machine learning techniques in these fields of manufacturing have been studied and reported.

3.1.1 ML techniques applied to the semiconductor manufacturing process dataset

Recently, modern technology in SECOM allows real time process control with the measured data gained from the equipment sensors installed in the production line. This recorded data of the entire manufacturing process, effective monitoring and optimal process control by investigating and analyzing these data are difficult work for process engineers. Traditional process control methodology like univariate and multivariate control charts is no longer an efficient method to control manufacturing systems with hundreds of processing stages. Instead, automatic and advanced process control method are required, for instance machine learning and deep learning techniques.

SECOM dataset is one of the most applied datasets in the context of manufacturing systems, as it has been applied for the purpose of benchmarking the machine learning algorithms developed in several fields of studies. Similarly, in this work, this dataset is used as a benchmarking data in validating the performance of our proposed models. A new data analytics frameworks model has been proposed in [2] for faults prediction in a large-scale manufacturing process, and SECOM dataset was used in validating the model. Where in their work the main focus is mainly on the approaches for identifying the important features from the dataset, feature selection, and an updated framework methodology and description. Two feature selection methods were used in the study including embedded and wrapper methods. XGBoost algorithm for prediction was developed and its performance has been compared with a RF-tree-based algorithm. The framework developed was able to effectively recognized the key features associated with product failure in each production line data.

An experimental study has been proposed [87] that evaluates different approaches including different levels for data imputation, data imbalance, feature selection, and classification techniques for the SECOM dataset. Moreover, the study proposed a novel process for data imputation that were inspired by image in-painting. Based on the obtained results, they were able to identify the suitable tools and stages for classifying the SECOM dataset. Results show that LR outperform the other classification algorithms and "In-painting KNN-Imputation" for data imputation, for synthetic data generation SMOTE was found to be the best and estimated false discovery rate for feature selection. Munirathinam and Ramadoss [3] proposed ML models including ANN, SVM, LR, NB, DT, and k-NN to automatically develop a predictive model that can predict equipment failures during SECOM process. Also, they constructed a decision model that helps in detecting equipment faults for maintaining higher process yields in manufacturing. Four feature selection techniques were considered including variable selection, correlation analysis, PCA, and average Diff method.

Kerdprasop and Kerdprasop [4] proposed models including k-NN, LR, NB, and DT that can automatically detect faults during wafer fabrication process. PCA, Gain ratio, and MeanDiff feature selection techniques were compared. Rare Case Boosting oversampling technique was used to handle the issue of imbalance in the dataset. The developed models were evaluated using TPR, TNR, precision, and F1 score metrics of performance. Experimental results show that right features selection and rare class oversampling enhance the model performance accuracy. They added, Oversampling technique applied to DT increase the performance of the model in terms of TPR.

Lee et al. [65] proposed critical process steps selection for SECOM process. Where in their work three data mining techniques were investigated under three missing value rates deletion. They applied meanDiff technique for feature selection and 75% dataset for training with the remaining 25 for testing the model. DR, LR, k-NN, and SVM were the considered algorithms, in where they have been evaluated across five metrics of performance including accuracy, precision, true positive rate (TPR), true negative rate (TNR), and F1_score.

Performance of six machine learning models including k-NN, RF, LR, DT, MLP, and AdaBoost is compared [88]. In the study, the algorithms were trained using dataset with and without dimensionality reduction, where three different feature selection techniques are considered including PCA, LDA, and SELECTFDR. They used SMOTE to address the issue of class imbalance in the dataset. Mean cross validation (CV) was used in training the algorithms with accuracy and ROC curve as the measure of performance. Based on their experimental results obtained, LR gave best performance with features selected using PCA.

Anghel et al. [89] proposed ML and DL techniques for error prediction in manufacturing process. Where MARS and SVM feature selection techniques are analyzed. The work developed and compares the performance of MARS + GBT and SVM + NN models. In comparison, SVM + NN deep learning technique outperforms MARS + GBT ML technique

A priori algorithm has been applied to SECOM dataset for mining the relations between operation parameters and quality outcomes [90]. In the data preprocessing stage, redundant features were discarded, PCA was used for important features

selection, a boosting technique was considered in sampling the minority class of the dataset. Some number of models are trained using 3:1 data split and their performance was evaluated using five metrics of performance.

Moldovan et al. [91] applied PCA, MARS and Boruta feature selection techniques while developing three machine learning models including RF, LR and GBT for fault detection and diagnosis. Several dataset cleaning techniques were considered involving removal of features with at least 55% missing values rates, missing values replacement with mean; mean heuristic; nearest neighbor heuristic; and numerical value. The performance of the models is analyzed under; unsampled dataset, oversampled and under sampled using SMOTE. k-Fold CV technique was used while training the models and five metrics of performance were considered to score the models. ANN, RF, LR, and DT prediction classification models are developed using 30% dataset holdout for testing [92]. Performance of SMOTE and ransom under-sampling techniques was compared using features selected from PCA. the models are evaluated using several metrics of performance.

Ko and Fujita [93] proposed an evidential analytics to disclose buried information in big data samples where UCI SECOM dataset is used as a case study. A framework model is proposed for model search using machine learning [94]. The proposed framework model was evaluated using two different datasets including SECOM dataset. An MLP model is proposed [95] that can be used to classify products as faulty or nonfaulty, the nodes of the algorithm were determined using Chicken Swarm Optimization (CSP) algorithm. Two different datasets were considered including SECOM and SETFI datasets to validate the proposed MLP+CSP model. Kim et al. [96] proposed a particle swarm optimization–deep belief network (PSO-DBN) for

minority classification where SECOM dataset was used in validating the model. Two feature selection techniques including standard deviation and Euclidean distance were used for selecting the most influential features from the dataset.

Table 2 gives a comprehensive summary of the most recent studies applied on the SECOM dataset. While reviewing the literature it has been pointed out that, SECOM dataset is one of the most widely explored public datasets in manufacturing domain, as it has been used in benchmarking several studies in the literature. Therefore, in this work, the aforementioned dataset has been implemented and trained ML algorithms for faults diagnosis in semiconductor manufacturing process.

3.2 Summary of the chapter

This chapter reports the literature review on the applications of machine learning techniques for smart semiconductor manufacturing processes. A special attention was on reviewing the recent advancements of machine learning techniques for semiconductor manufacturing processes, in where the reviewed studies are classified and reported in Table 2. The studies are categorized based on on the ML technique considered, data preprocessing technique considered, feature selection technique considered, synthetic data generation technique considered, the model validation technique considered, model evaluation metrics considered, and the respective study key findings. Next chapter presents the research methodology.

Table 2: Summary of the recent ML algorithms applied to SECOM dataset.

ML Algorithms	Data Cleaning/Preprocessing	Feature Selection	Sampling Technique	Validation Technique	Metrics of Evaluation	Key Findings	Reference
k-NN, RF, LR, DT, MLP and AdaBoost	• With and without dimensionality reduction	• PCA • LDA • SELECTFDR	• SMOTE	• mean CV	• ROC curve • accuracy	• RF performs poorly due to the data imbalance • LR + PCA performs better • LR + SELECTFDR is best suited for the dataset • Imbalance causes the models to deliver worse performance • SMOTE helps in RF performance enhancement • k-NN gave the highest mean-CV score	[88]
DT, LR, k-NN, SVM	EM imputation. 3 σ rule • 80%, • 50%, and • 20% deletion of the dataset	• MeanDiff	• SMOTE	• 75|25 split	• accuracy • precision • TPR • FPR • f-measure	• proposed critical process steps selection for SECOM • Implemented three data mining techniques • Investigated the SECOM dataset under three missing value rates • proposed methods show good classification performance	[65]
GBT and NN	• Removal of features with 0 standard deviation • Features with at least 55% missing values are removed	• MARS • SVM	• -	• 80|20 split	• Accuracy • TPR • FPR • Precision • F - measure	• Proposed ML and DL techniques for error prediction in manufacturing process • MARS and SVM features selection techniques are analyzed • MARS + GBT and SVM + NN models are developed • In comparison, SVM + NN deep learning technique outperforms MARS + GBT ML technique	[89]
Rough Set, SVM, NB, ID3, CART, C5.0, and A Priori	• Remove redundant features	• PCA	• Boosting	• 3:1 split	• Accuracy • TPR • Geometric mean • Balanced error rate • F1_score	• Introduced quality prediction modeling framework for MMS environment • A priori algorithm has been applied to SECOM dataset for mining the relations between operation parameters and quality outcomes • The methodology can help manufacturers to obtain real time quality info of manufacturing lines • The framework can help in manufacturing processes optimization	[90]

SVM, KNN, RF and LR	• Data pruning • In-painting k-NN-imputation	• UFS • selecting from a model • RFE • PCA	• SMOTE	• k-Fold CV	• AUC • TNR	• Proposed in-painting k-NN imputation • Analyzed feature importance from the best methodology perspective • LR outperforms all other algorithms • SMOTE was found to be the best for data generation • SELECTFDR features selection found to be the best • Proposed in-painting imputation technique shows good performance	[87]
RF, LR, and GBT	Features with at least 55% missing values are removed. Missing values replacement with: • mean • mean heuristic • nearest neighbor heuristic • numerical value	• Boruta • MARS • PCA	• SMOTE; under-sampling and oversampling • Unsampled	• k-Fold CV	• false positive rate • accuracy • precision • recall • f-measure	• applied MARS and Boruta features selection techniques • results show that, best model performance was obtained when the majority class are under-sampled; under-sampled + Boruta + RF • better precision is obtained with Boruta and MARS • better accuracy is obtained with unsampled data using RF and LR	[91]
ANN, RF, LR and DT	Missing values removal.	• PCA	• SMOTE • RUS (ransom under-sampling)	• 70|30 split	• accuracy • precision • sensitivity • specificity • f-measure	• applied SMOTE for data imbalance solving • proposed methodology manages to solve the imbalance between the two classes	[92]
RF and XGBoost	• assign missing instances with independent group missing value replacement with a unique value	• wrapper • embedded	• SMOTE	• k-Fold CV	• accuracy • sensitivity • specificity • f-measure	• proposed a new data analytics framework for faults prediction • focused on features selection and important features identification • XGBoost performance was compared with RF's • Proposed framework identified influential features successfully	[2]
ANN, SVM, LR, NB, DT and KNN	Cleansing procedures to discard missing values.	• variable selection • correlation analysis • PCA • Average Diff method	• -	• k-Fold CV	• TPR • FPR • precision • f-measure • MCC, ROC-area, PRC, PRC-area • recall	• proposed ML techniques to automatically develop a predictive model that can predict equipment failures during SECOM process • constructed a decision model that helps in detecting equipment faults for maintaining higher process yields in manufacturing	[3]
k-NN, LR, NB, and DT	• Features with single values are removed	• PCA • Gain ratio • *MeanDiff*	• Rare Case Boosting: oversampling	• k-Fold CV	• TPR • FPR • Precision	• Proposed models that can automatically detect faults during wafer fabrication process	[4]

	• Features with no value entry are removed • Features with at least 55% missing values are removed	•	• F1_score	• Proposed methods of features selection and oversampling • Results show that right features selection and rare class oversampling enhance the model performance accuracy • Developed NB model can classify the fault cases with a high rate, similarly false alarm rate • Oversampling technique applied to DT increase the performance of the model in terms of TPR.

PCA: Principal component analysis; SELECTFDR: False discovery rate; SMOTE: Synthetic minority oversampling technique; CV: Cross validation; MARS: Multivariate adaptive regression spline; UFS: Univariate feature selection; RFE: recursive feature elimination; EM: expectation Maximization;

Chapter 4

RESEARCH METHODOLOGY

This section describes the thesis methodology and the stages that were followed in detail for the prediction model development, and how each amongst these stages was carried-out explicitly. As seen in Chapter 2, literature studies have been carried-out in the areas of interest to determine and define the current state-of-the-art knowledge, determine the feasibility of the research questions and the research methodologies that have been applied in this thesis. This chapter comprises of three sections. First, section 4.1 presents the methodology for machine learning techniques development for classification. Then, section 4.2 presents the process of model selection, the followed model training and validation techniques. Finally, the concluding remarks of this chapter are reported in section 4.3.

4.1 ML techniques development for classification

The proposed methodology for classification model's development in this thesis is summarized. The following summarizes the sequence of the methodology followed:

A. Data identification and collection.
B. Data preprocessing stage; this is the step that clean and process the data for model training by performing:
    a) Data cleaning
    b) Feature selection
    c) Feature scaling
C. Data sampling: oversampling the data if it's imbalanced or highly imbalanced.

D. Model selection and training

E. Model validation and performance metrics:

   a) Confusion metrics.

   b) Accuracy

   c) Recall

   d) Precision

   e) Specificity

   f) F1_score

4.1.1 Proposed methodology

All the steps in model development are implemented in Google Colab Environment using Python Programming Language. Figure 15 shows the methodology framework followed while training the predictive models for SECOM dataset classifier models development.

The classification ML algorithms as mentioned in the previous subsection, namely; MLP (ANN), XGBoost, LR, DT, NB, LDA, RF, SVC, AdaBoost and GBT have been selected, trained and validated, and their performance metrics have been evaluated and compared. Moreover, while training the models, two approaches have been considered as follows:

1. model training with training/testing data split (80|20 split) validation approach. This technique of model training has been applied in [89], and [90].
2. model training with k-Fold cross validation approach. This methodology of validation has been applied in [2], [3], [4], [87] and [91].

Each of the approaches mentioned above has been considered in order to explore the applicability of the two methods, and also to discover which among the algorithms performs better against the two methods.

Firstly, for the model training, the dataset has been divided into training and testing sets, 80% for training and 20% for testing as the first validation technique. Throughout this work, all the algorithms considered are trained with the same ratio of data split.

Then, the models are trained using CV as the second validation technique. Six among the trained models are trained with 7-Fold, namely, MLP, NB, LDA, RF, AdaBoost and GBT. Two are trained with 3-Fold including LR and SVC. XGBoost is trained with 13-Fold, and DT with 10-Fold. For each model, the different number of k-Fold is selected because the model performs better with the selected number of k.

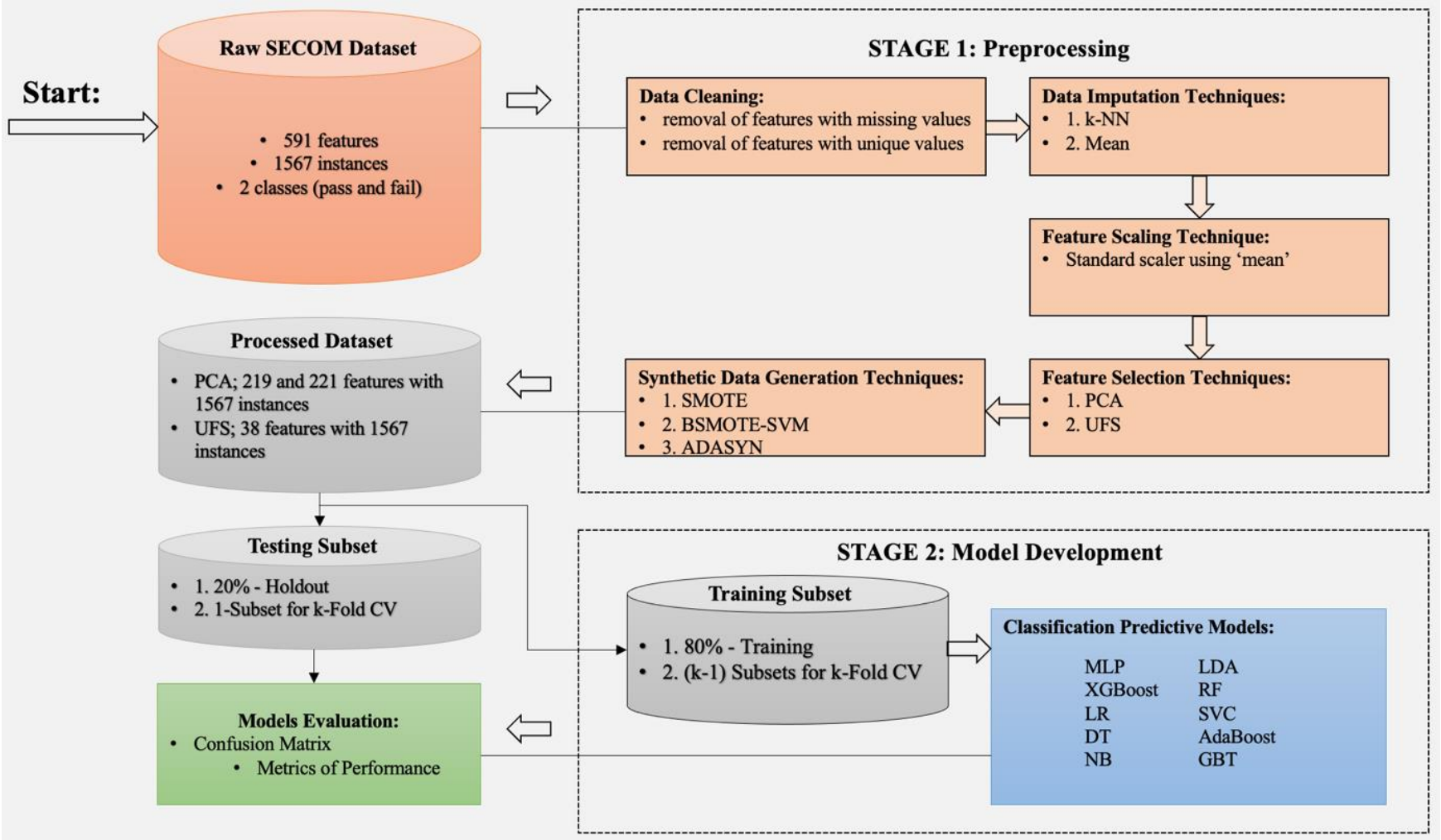

Figure 15: Overall framework of the proposed methodology for predictive ML development.

4.1.2 Dataset

As this work aims at further exploring the machine learning technologies applicable to smart manufacturing domains. It is important to find the suitable and available dataset to explore. While reviewing the literature, it has been pointed out that SECOM dataset is one of the most widely explored public datasets in manufacturing process, as it has been used in benchmarking several studies in the literature. Table 2 summarizes some of these studies that have utilized UCI SECOM dataset in benchmarking their proposed models. Therefore, in this work, the SECOOM dataset has been used to validate the proposed methodologies and to train ten ML algorithms for faults diagnosis in semiconductor manufacturing process.

4.1.3 Dataset preprocessing

Dataset obtained from manufacturing domain can contain little or large amount of redundant information that, if fed to the predictive ML model, can affect the performance of the model and result in an unreliable result [2]. Similarly, it is of great importance finding a suitable way on how to process the data or the dataset at hand (data preprocessing) when developing a ML model. Thus, the ML predictive model can be developed with that preprocessed data and evaluated.

4.1.3.1 Data cleaning

Data cleaning helps in handling data features like outliers and missing values so that a faultlessly organized dataset could be obtained. The following steps were followed in this work for the imbalanced datasets cleaning:

I. all features with empty/no values have been removed. Considered in [3].
II. all features with unique values have been removed. This method has been implemented in [2] and [4].

III. all features with less than half of their values have been removed. Applied in [4] and [89].

4.1.3.2 Data imputation

After data cleaning, there were still some number of cells with missing data points, thus, the cells with missing values had to be replaced and needed to be processed for ML development. From the literature there are several different techniques proposed that can be implemented to handle such cases of missingness. Removal of any feature with a missing value is one of the techniques that could be used and as proposed in some of the literatures. However, this approach will for certain remove some features from the dataset that may have the highest impact on the predictive model performance outcome. As a result, the efficiency of the model may be rendered. Similarly, substitution of the missing values with 'mean' is one of the proposed approaches for data imputation [87]. In this thesis, we adopted and applied mean and k-NN data imputation techniques, as they have been implemented for the missing values substitution.

4.1.3.2.1 Mean data imputation

Mean imputation is one of the most often used method for missing values imputation [97]. For a given instance or attribute, the method uses mean to replace the missing datapoint. This technique has been considered in [91].

4.1.3.2.2 k-NN data imputation

k-NN imputation [97]–[99] uses k-NN algorithm to estimate and substitute the required missing attribute. The importance of k-NN imputation include the creation of predictive model for each attribute with missing datapoints is not required, k-NN can handle instances with several missing datapoints, and it can also predict both continues

and discrete attributes. Moreover, k-NN algorithm can be used to get weighted vectors for attributes in a dataset [99].

4.1.3.3 Feature scaling or normalizing

Feature scaling is a method that can be applied to dataset in order to normalize it. The main goal of normalizing the data with the help of feature scaling is to transform the continuous input and output data to a linear scale values that ranges from -1 to 1 or 0 to 1. -1 to 1 scaling method is considered in this work. Feature scaling help the model to easily learn the important features from the data. This means that all the features are normalized to the values ranging from -1 to 1. The standard scaling using mean and standard deviation is implemented in this work. The standard score of a sample feature is calculated using equation:

$$Xnormalized = \frac{X - Xmean}{Xstd} \quad (3)$$

where x is the feature data to be normalized, x_{mean} is the mean of the feature data or 0 if the data has no mean value, and x_{std} is the standard deviation of the training feature data sample or 1 if it has no standard deviation.

4.1.3.4 Feature selection

Selection of the most crucial features is what helps the most when developing ML predictive model. Therefore, it is important to use the most appropriate and best method in selecting the features for model development as those features influence the performance of the model having no features with missing data points as they have been filled from the data imputation using 'mean' substitution of the feature columns. The lack of missingness in the dataset allows us to proceed with feature selection. Feature selection techniques applied in this work is Principal Component Analysis and Univariate feature selection (SELECTFDR).

4.1.3.4.1 Principal component analysis (PCA)

PCA is a multivariate technique that analyzes a data table in which observations are described by several inter-correlated quantitative dependent variables. Its goal is to extract the important information from the table, to represent it as a set of new orthogonal variables called principal components, and to display the pattern of similarity of the observations and of the variables as points in maps [100]. Mathematically, PCA relies upon the singular value decomposition of rectangular and the eigen-decomposition of positive semi-definite matrices. Similarly, the technique lessens the dimension of dataset by calculating the covariance and then perform eigen value decomposition of the covariance matrix. In this work, the method of selecting all the features with eigen values of at least greater than 1 is applied. This technique has been applied in [3], [4], [88], [90], [87], [91] and [92].

4.1.3.4.2 Univariate feature selection (UFS)

This feature selection technique selects the best features with the highest score after performing univariate statistical tests on the input dataset [87]. In this thesis the scoring function used is estimated false discovery rate (SELECTFDR). This technique is considered because it has been applied in [87] and has shown promising results. In fact, it performed better than the other UFS selection techniques they considered in their work.

4.1.3.5 Synthetic data generation techniques (SDGT)

In a dataset if there is any class or group with small or very large number of instances compared to another class, that data is referred to as imbalanced dataset, and it is a common problem in real data and data mining [65]. Most classification algorithms provide poor metrics of performance when trained with an imbalanced dataset.

Synthetic sampling data generation methods are applied in a wide range of studies to solve such type of problem.

When a dataset has passed preprocessing - the stages of cleaning, data points imputation (if it has missing values) and feature scaling and selection, then, if there is an imbalance or a high imbalance between the desired output classes, the synthetic data sampling generation method can be applied in order to generate synthetic data to balance the imbalance within the two classes, so that the model could learn better from the dataset.

Sampling method help in creating a synthetic data by oversampling the minority of the desired output class. Synthetic Minority Oversampling Technique (SMOTE) [101] and SMOTE with Selective Synthetic Sampling Generation are among the techniques used in generating synthetic data for imbalanced classification problems. SMOTE has been considered in several studies including [2], [65], [87], [88], [91] and [92]. To the best of our knowledge, no one has reported a study on BSMOTE-SVM and ADASYN using SECOM dataset.

Implementation of the aforementioned sampling techniques was applied to balance the minority class, and to ensure that the minority class is as sufficient as the majority class for the classification model to learn from, and to be able to classify the two classes easily without misclassifying the minority class.

4.1.3.5.1 Synthetic minority oversampling technique (SMOTE)

SMOTE [101] uses k-NN to create new instances in order to maintain the balance between the two classes [96]. Based upon the amount of data required to be over-sampled randomly, neighbors from the k nearest neighbors are selected [101]. The class with minority instances is oversampled by taking or considering each class in the

minority sample and introducing synthetic examples along the line segments joining any or all of the k minority class nearest neighbors. Figure 16 illustrates the procedure involved in SMOTE data generation. SMOTE data generation consists of the following steps:

a. A sample from the minority class will be selected at random
b. k number of nearest neighbors near to that selected minority sample will be selected. Usually, k is set at 5.
c. Generation of the synthetic samples between the selected minority sample and that 5 nearest neighbors' samples. These steps will be repeated till the number of minority samples equal to that of the majority samples.

Creating the synthetic data by oversampling the minority class helps in making the minority class equal to the majority class, so that any classification algorithm that may have difficulty handling the imbalanced class and learning from the data could easily and efficiently classify the two classes. SMOTE increases the amount of data without altering the variation or information of the data, or feeding new information or variation to the learning model [102].

4.1.3.5.2 Borderline-SMOTE SVM (BSMOTE-SVM)

BSMOTE-SVM [103] uses a decision boundary to generate the synthetic data between the two classes, unlike SMOTE that creates synthetic data at random. Also, BSMOTE-SVM uses ensemble SVM algorithm to identify the misclassification and generate more samples in the minority class. The technique uses SVMs to generate the samples near the decision boundary. As shown in Figure 17, samples near decision boundary can be roughly characterized from support hyperplane learned by the first SVM [103]. The generated synthetic minority sample tend to correct the skewness distribution

within the two class samples finely, as the decision boundary skew towards the minority. Simply put, it gives more attention to where the dataset is separated.

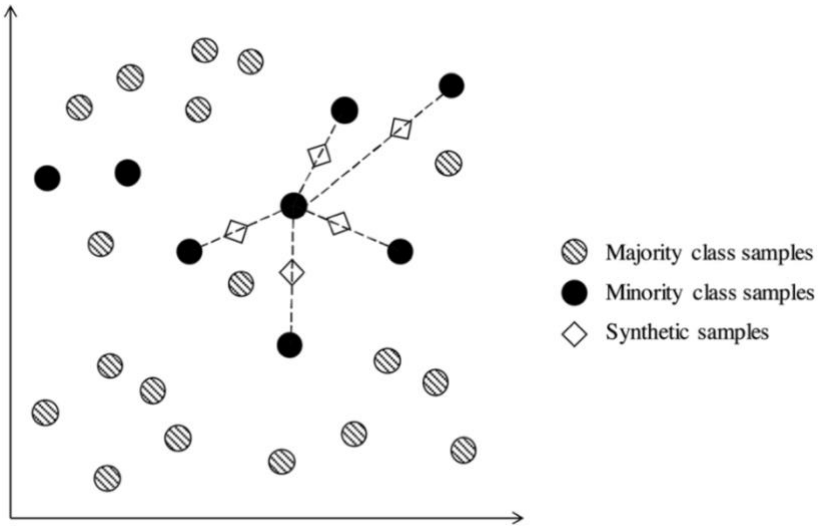

Figure 16: Illustration of SMOTE data generation technique [65].

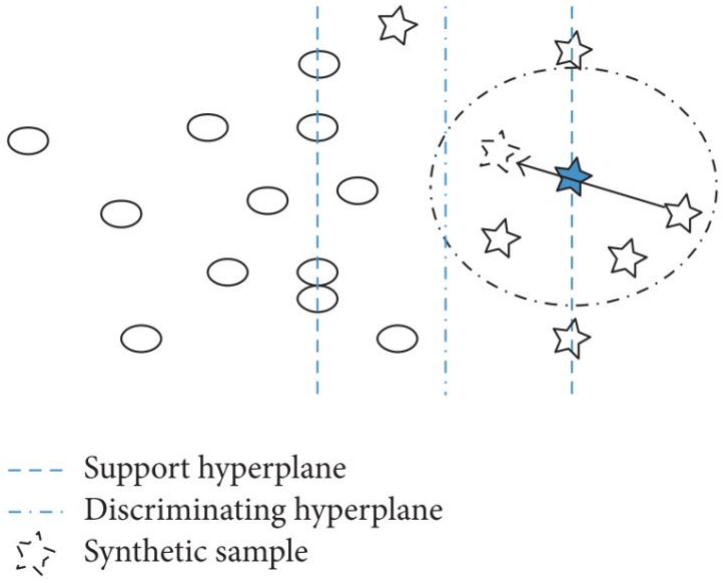

Figure 17: Illustration of BSMOTE-SVM data generation technique [103].

4.1.3.5.3 Adaptive synthetic oversampling (ADASYN)

ADASYN [104] is an extension of SMOTE that generates synthetic data samples in the minority class according to their weighted difference distribution. In regard to that,

ADASYN focuses more on the minority class samples that are harder to learn than those minority samples that are easier to learn. It helps in reducing the learning bias introduced by the imbalance amongst the classes, and also it adaptively shifts the decision boundary to focus on those minority samples that are difficult to learn.

4.2 Predictive model selection, training and validation

The selected algorithms to be trained are listed in the previous section (section 4.1.1). These classifiers are selected, trained and validated in the Google Colab Environment using Python Programming Language.

The technique for validating the selected classifiers used in this work is Cross validation (CV) technique. Among the methods of cross validation, the one used in this thesis is k-Fold cross validation [105] as it is the most commonly used method and it is used for validating most of the classifier models. The main objective of applying CV is to let the model learn from all the training data by dividing the dataset into a number of k-subsets as k-Fold subsets. One subset is used for validation and the remaining k-1 subsets are used in training the model [106], [107]. Moreover, CV helps in increasing the performance of classifiers.

Note: Sometimes, k-Fold method is used in a situation where the training dataset seems to be small or the number of attributes is insufficient for the classifier model to learn from. Thus, k-Fold can be applied to divide the dataset into k number of subsets and the model will take one subset from the k-number of subsets with which to test itself after training from the rest of (k-1)-subsets. It will keep doing so until it train and test itself on the k-number of divided subsets. For instance, say the number of k is 3. This means 3-Fold. So, the classifier will be trained with 2 data subsets from the 3

subsets and be tested on the remaining 1 data subset. Similarly, the same will be done with the other two subsets, until it is done three times, with each subset being used as a testing data and the rest for training.

4.2.1 Predictive model evaluation metrics

The classifiers have to be evaluated with the right evaluation metrics in order to avoid bias in the model performance. Thus, if you choose the wrong metric in evaluating the classifier models, it is most likely that you could be misled about the expected performance of your classifier model. Figure 15 shows the schematic process of the classifier models development.

Selecting an appropriate classifier for your model could be challenging in in general. Similarly, when dealing with an imbalanced dataset, problems, particularly classification problems involving imbalanced dataset, tend to be tedious to handle in terms of evaluating the classifier. Because most of the standard metrics used assume a balanced class distribution, and of course not all the classes are distributed equally, not all the performance metrics for evaluation could be useful for imbalanced classification. Classification accuracy and error are among the most widely applied standard metrics for evaluating classification models. However, these metrics evaluate the classifiers considering the classes as equally important. If the dataset is imbalanced, then the accuracy matric will not be used for evaluation. Hence, precision and recall metrics must be used in this case in order to have non bias evaluation.

Such metrics might be needed as give a focus on the minority class, because it is from the minority class that we lack enough features required to train and get an effective predictive model. There are tens of evaluation metrics from which to choose in order to evaluate or measure the performance of the classifier model. To mention a few,

classifier models can be assessed with the use of standard statistical metrics including accuracy, specificity, sensitivity, precision, etc. [2].

4.2.1.1 Confusion matrix

These performance measure evaluation metrics would best understand with the help of a confusion matrix. Confusion matrix gives more insight into the performance of a predictive ML model. Similarly, the matrix identifies and tells which classes are being correctly and incorrectly predicted and classified. Table 3 below shows the summary on how confusion matrix is and how each cell in the table has a specific and well-understood name.

Table 3: Confusion Matrix for Predictive Model Evaluation.

		Predicted Class	
		Positive Prediction	Negative Prediction
True Class	Positive Class (Pass)	True Positive (TP)	False Negative (FN)
	Negative Class (Fail)	False Positive (FP)	True Negative (TN)

The cells in the confusion matrix could be defined as follows; TP refers to true positive or the number of fail classes that are identified accurately as fail classes, FP refers to false positive or the number of pass class that are identified incorrectly as fail class, FN refers to false negative or the number of fail classes that are misclassified as pass classes, and TN refers to true negative or the number of pass classes that are classified correctly as pass classes.

4.2.1.2 Accuracy performance metric

Accuracy is the most widely used threshold metric. However, this metric may not generally be appropriate for imbalanced classification. Because, a high accuracy from a model could be achieved by only predicting the majority class without being able to predict the minority class. Accuracy is defined as correct predictions divide by the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

4.2.1.3 Sensitivity and specificity performance metrics

Sensitivity-Specificity and Precision-Recall metrics are the two metrics groups that are considered in this work plus the accuracy of the predictive model. Sensitivity means the true positive rate and states how well the positive class is predicted; whereas, specificity is the true negative rate, and it summarizes how well the negative class is predicted. Note that sensitivity is more preferred over specificity for imbalanced problems classification.

$$\text{Specificity} = \frac{TN}{FP + TN} \quad (5)$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad (6)$$

4.2.1.4 Precision, recall and F1_score performance metrics

Precision gives summary of the fraction between examples of the positive class that belong to the positive class. Recall tells how well the examples of the positive class were predicted. Sensitivity gives the same measurements as recall. On the other hand, F1_score incorporates both precision and recall to balance the tradeoff [108].

$$\text{Precision} = \frac{TP}{TP + FP} \quad (7)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (8)$$

$$\text{F1\_score} = \frac{2TP}{2TP + FP + FN} \quad (9)$$

4.3 Summary of the chapter

Throughout this chapter, methodologies followed in this thesis have been presented. These methodologies comprise data cleaning, missing datapoints imputation techniques, most potential features selection techniques, features normalizing

technique, synthetic data generation techniques, model training and validation techniques, model development, and model performance evaluation.

Firstly, two missing datapoints imputation techniques were presented: missing datapoints imputation using mean and using k-NN, followed with a feature normalizing technique. Secondly, two feature selection known as dimensionality reduction techniques were presented: principal component analysis and univariate feature selection. Thirdly, three synthetic data oversampling techniques were proposed including SMOTE and its two variants: Borderline-SMOTE SVM and ADASYN. These techniques were proposed so as to handle the issue of class imbalance in the semiconductor manufacturing dataset, investigate their feasibility, and compare their performance in the semiconductor manufacturing processes. Then, two model validation techniques were considered; 80|20 split and k-Fold cross validation. Finally, seven metrics of performance evaluation were presented so as to measure the performance of the proposed methodologies using the proposed algorithms. The next chapter reports the experimental results. In the chapter, a case study has been conducted using dataset from a semiconductor manufacturing process.

Chapter 5

EXPERIMENTAL RESULTS

The experimental results using the proposed methodologies are reported in this chapter. The chapter contains two sections, section 5.1 and 5.2.

Section 5.1 reports the experimental results on a case study conducted using a dataset from semiconductor manufacturing process. The section is divided into six subsections. First, section 5.1.1 presents the analysis, results, and influence of the main steps from the proposed methodologies on the considered predictive models using the SECOM dataset. These steps include the investigation of: starting from the effect of data cleaning, the effect of datapoints imputation techniques, the effect of datapoints imputation techniques with feature selection, the effect of mean imputation technique and feature normalizing and selection using PCA, then the effect of synthetic minority oversampling technique on the SECOM modified raw dataset. Second, section 5.1.2 presents the experimental results obtained using the overall proposed methodologies using SMOTE. Third, section 5.1.3 reports the general experimental results obtained using Borderline-SMOTE SVM on the proposed methodologies. Fourth, section 5.1.4 reports ADASYN experimental results using the proposed methodologies. Then, section 5.1.5 reports the experimental results general comparison analysis considering several perfectives. Lastly, section 5.1.6 gives the comparative analysis between our experimental results obtained and similar studies reported experimental results. Section 5.2 reports the concluding remarks of this chapter.

5.1 A case study on semiconductor manufacturing process

This section analyzes the semiconductor manufacturing process dataset by applying the proposed methodologies from the previous chapter using ten machine learning algorithms for fault detection in the semiconductor manufacturing processes.

5.1.1 ML development using SECOM dataset

5.1.1.1 SECOM dataset cleaning

Dataset obtained from manufacturing domain can contain some amount of redundant information, that if fed to the predictive ML model, can affect the performance of the model and result in an unreliable outcome, as stated above. At first, we explore the dataset, thus, some features have been removed, because some of them contain no data-points, unique value, or little number of data-points. Amongst 590 features about 140 were deleted, see Table 4.

Table 4: Dataset description before and after each step.

#number of:	Raw data	MRD	MRD + MI + PCA	MI+ S + PCA	MRD + MI + SMOTE
features	590	450	17	221	450
'pass' instances	1463	1463	1463	1463	1463
'fail' instances	104	104	104	104	1463
instances	1567	1567	1567	1567	2926

MRD: Modified raw dataset; MI: Mean imputation; S: Scaling;

5.1.1.1.1 Model development using (MRD) modified raw dataset

In this section, ten machine learning algorithms are selected for the predictive models' development. 80|2 split validation technique was used for training and building the models. That is, 80% of the dataset is used for model development and the remaining is used to validate the performance of the proposed models. Confusion matrix of the experimental results obtained is shown in Table 10. Only XGBoost was able to generate results whereas, rest of the models failed to generate any. This happens because the dataset is noisy as it contains missing values. Similarly, Table 5 shows the

performance measures obtained when the four models are trained with raw and modified dataset.

5.1.1.1.2 Effect of datapoints imputation

This section analyzes the effect of data points imputation on the modified raw data. Two imputation methods have been adopted and their effect on the prediction models' performance has been evaluated. These imputation techniques include mean and k-NN missing datapoints imputation techniques, and their experimental results are shown in Table 10 and Table 5. Note that, the same number of features from previous section is used to train the models, that is 450 features.

As clearly seen from the results, all the developed models were able to generate some results when the missing datapoints in the dataset are substituted using those imputation techniques, unlike in the previous section. However, most of the models (including XGBoost, LR, RF, SVC, and GBT) were not able to classify the fail class with an exception of MLP, DT, NB, LDA, and AdaBoost, each of which managed to classify few when trained using dataset with missing datapoints mean imputation. XGBoost, RF, and SVC produce similar performance using both techniques of imputation. GBT managed to classify the fail class correctly 2 times when trained with dataset using k-NN imputation.

5.1.1.1.3 Effect of datapoints imputation and features selection without scaling

Then, we tried to analyze the effect of features selection (using PCA) on the modified raw dataset in this section, but we did not perform any normalizing on the features. Also, only mean imputation is considered in this section. After PCA, 17 features were selected from 450, see Table 4. The parameter setting of PCA are set to select features with 0.99 variance. As a result, 17 features were selected from those 450 features. That

happened because the dataset is noisy and had missing values and it had not been normalized. Table 5 shows the experimental results obtained when the models were trained with the 17 features selected using PCA. Results obtained clearly show that feature selection has no or has little effect on the models' performance, because similar results are obtained in the previous section (5.1.1.1.2), but slightly different.

5.1.1.1.4 Effect of mean imputation, features scaling and selection

To increase the efficiency and accuracy of the proposed classifier models, in this section we analyzed the effect of features scaling and feature selection. Moreover, these features are real values and the feature interval values differ from one another. Thus, there is a need for feature normalization called feature scaling. Standard scaler is used to normalize the features to a range of around -1 to 1. All the features are normalized by subtracting the mean of the feature value and dividing it by standard deviation of the feature values, and thus, they range between -1 to 1. PCA was used for features selection. As a result, 221 features were selected by PCA as the most important features. Refer to Table 4 to see the description of the dataset used in this section. Moreover, features with 0.99 variance are selected (using PCA). Normalizing increases the number of features selected by PCA technique when compared with 17 features that are selected when the dataset has not been normalized. Then, these features with mean imputation are considered in developing the ML models.

The results obtained are shown in Table 10 and Table 5. Similar results were obtained as in previous sections (5.1.1.1.2 and 5.1.1.1.3). This shows that, features scaling has no or little impact on the models' performance. However, features normalizing helps PCA to select a greater number of features for the model trained.

5.1.1.1.5 Effect of SMOTE, without features scaling and selection

In this section, SMOTE has been applied in order to generate more synthetic data within the minority class to equal the distribution between the two classes as they have a ratio of 1:14. This technique has been applied in order to solve this problem of imbalance between the two classes and to determine its effect on the model's performance. Also, it has been applied to the whole dataset, not just the training subset as what many researchers have considered. After implementing SMOTE, the two classes are balanced and now the models training stage can be carried out. See Table 4 for the dataset description used in this section.

Table 10 and Table 5 report the results of the performance metrics obtained when the algorithms are trained using the modified raw dataset with SMOTE synthetic data generation, and shows the confusion matrix results obtained. The tables clearly show how the performance of the models is improved as they were able to classify the fail class. RF outperforms all the developed models followed by XGBoost. SVC delivers least metrics of performance in comparison to the other models, because it is sensitive to unnormalized data.

5.1.1.1.6 Discussion on the results obtained

This section discusses and compares the results obtained from the previous sections (5.1.1.1.1 to 5.1.1.1.5). First, we tried to analyze the effect of each step in the preprocessing stage. We did so in order to verify and figure out the step that has the main impact on the performance of the proposed models. It has been clearly seen from this analysis that, most of the steps have no or little impact on the model development when trained with imbalanced dataset. However, the performance of the algorithms significantly increases when trained with dataset after SMOTE implementation.

Consequently, the imbalance of the dataset has the greatest impact on the models' performance. Table 5 reports the comparison amongst the experimental results obtained in this section.

Table 5: Metrics of performance results obtained before and after each step.

Methods:	**Accuracy**						**Recall**						**Precision**						**Specificity**						**F1_score**					
	MRD	MRD + MI	MRD + k-NNI	MRD + MI + PCA	MRD + MI + FS + PCA	MRD + MI + SMOTE	MRD	MRD + MI	MRD + k-NNI	MRD + MI + PCA	MRD + MI + FS + PCA	MRD + MI + SMOTE	MRD	MRD + MI	MRD + k-NNI	MRD + MI + PCA	MRD + MI + FS + PCA	MRD + MI + SMOTE	MRD	MRD + MI	MRD + k-NNI	MRD + MI + PCA	MRD + MI + FS + PCA	MRD + MI + SMOTE	MRD	MRD + MI	MRD + k-NNI	MRD + MI + PCA	MRD + MI + FS + PCA	MRD + MI + SMOTE
MLP	NAN	0.7293	0.9140	0.9268	0.9268	0.9215	NAN	0.2500	0.0000	0.0000	0.0500	0.9486	NAN	0.0667	0.0000	0.0000	0.3333	0.8994	NAN	0.7619	0.9762	0.9898	0.9932	0.8946	NAN	0.1053	0.0000	0.0000	0.0870	0.9233
XGBoost	0.9363	0.9363	0.9363	0.9299	0.9299	0.9676	0.0000	0.0000	0.0000	0.0000	0.0000	0.9589	NAN	NAN	NAN	0.0000	NAN	0.9756	1.0000	1.0000	1.0000	0.9932	1.0000	0.9762	0.0000	0.0000	0.0000	0.0000	0.0000	0.9672
LR	NAN	0.9299	0.9299	0.9363	0.9363	0.7235	NAN	0.0000	0.0000	0.0000	0.2500	0.7329	NAN	0.0000	0.0000	NAN	0.2500	0.7181	NAN	0.9932	0.9932	1.0000	0.9490	0.7143	NAN	0.0000	0.0000	0.0000	0.2500	0.7254
DT	NAN	0.8885	0.8949	0.8503	0.8503	0.9078	NAN	0.1500	0.2500	0.0000	0.1500	0.9589	NAN	0.1429	0.2174	0.0000	0.1364	0.8696	NAN	0.9388	0.9388	0.9082	0.9354	0.8571	NAN	0.1463	0.2326	0.0000	0.1429	0.9121
NB	NAN	0.2070	0.1815	0.9013	0.9013	0.5734	NAN	0.7000	0.7500	0.0500	0.0500	0.9623	NAN	0.0545	0.0562	0.0769	0.0588	0.5404	NAN	0.1735	0.1429	0.9592	0.9456	0.1871	NAN	0.1011	0.1045	0.0606	0.0541	0.6921
LDA	NAN	0.9108	0.9204	0.9236	0.9236	0.9147	NAN	0.4500	0.4000	0.0000	0.2000	1.0000	NAN	0.3462	0.3810	0.0000	0.3333	0.5838	NAN	0.6422	0.9558	0.9864	0.9728	0.8299	NAN	0.3913	0.3902	0.0000	0.2500	0.9211
RF	NAN	0.9363	0.9363	0.9363	0.9363	0.9846	NAN	0.0000	0.0000	0.0000	0.0000	0.9726	NAN	NAN	NAN	NAN	NAN	0.9965	NAN	1.0000	1.0000	1.0000	1.0000	0.9966	NAN	0.0000	0.0000	0.0000	0.0000	0.9844
SVC	NAN	0.9363	0.9363	0.9363	0.9363	0.6775	NAN	0.0000	0.0000	0.0000	0.0000	0.7979	NAN	NAN	NAN	NAN	NAN	0.6419	NAN	1.0000	1.0000	1.0000	1.0000	0.5578	NAN	0.0000	0.0000	0.0000	0.0000	0.7115
AdaBoost	NAN	0.9236	0.9299	0.9236	0.9236	0.9266	NAN	0.0500	0.2000	0.0000	0.0000	0.9349	NAN	0.1667	0.4000	0.0000	0.0000	0.9192	NAN	0.9830	0.9796	0.9864	0.9762	0.9184	NAN	0.0769	0.2667	0.0000	0.0000	0.9270
GBT	NAN	0.9331	0.9299	0.9331	0.9331	0.9642	NAN	0.0000	0.1000	0.0000	0.0000	0.9589	NAN	0.0000	0.3333	0.0000	0.0000	0.9689	NAN	0.9966	0.9864	0.9966	0.9898	0.9694	NAN	0.0000	0.1538	0.0000	0.0000	0.9639

MRD: Modified raw data; MI: Mean imputation; FS: Features scaling;

5.1.2 Proposed methodology with SMOTE

In this section, the overall proposed methodology is analyzed considering the implementation of SMOTE synthetic data generation technique. Refer back to RESEARCH METHODOLOGY and Figure 15 to see the overall methodology. Moreover, the effect of mean (5.1.2.1) and k-NN (5.1.2.2) imputation techniques are considered separately in the following subsections. Table 6 reports the description of the dataset used while training the models. Two validation techniques are considered including 80|20 split and k-Fold cross validation. All the ten models are trained using 80|20 split validation technique. However, different number of k is considered while training using k-Fold cross validation technique. 7-Fold is considered for MLP, NB, LDA, RF, SVC, AdaBoost, and GBT; 13-Fold for XGBoost; 3-Fold for LR and 10-Fold for DT.

Table 6: Datasets characteristic used for ML model development after preprocessing with and without SDGT.

	Methods:				
#number of:	MI + PCA - without SDGT	UFS - without SDGT	MI + PCA - with SDGT	k-NNI + PCA - with SDGT	k-NNI + UFS - with SDGT
features	221	38	221	219	38
'pass' instances	1463	1463	1463	1463	1463
'fail' instances	104	104	1463	1463	1463
instances	1567	1567	2926	2926	2926

SDGT: Synthetic data generation technique; MI: Mean imputation; k-NNI: k-NN imputation;

Six methodologies are proposed including MI + PCA + SMOTE + 80|20 split, MI + PCA + SMOTE + CV, k-NNI + PCA + SMOTE + 80|20 split, k-NNI + UFS + SMOTE +80|20 split, k-NNI + PCA + SMOTE + CV, and k-NNI + UFS + SMOTE +CV. Ten machine learning for fault diagnosis models are trained using these proposed methodologies. The respective results obtained from the experimentation are reported in the following subsections.

5.1.2.1 Mean imputation with SMOTE

First, we consider training the models by substituting the missing datapoints using mean imputation technique together with selecting the most important features using PCA. The experimental results obtained are shown in Table 11 (confusion metric results) and Table 7 (metrics of performance). These tables report the results obtained when the models are trained with 80|20 – split validation technique and the results obtained when the models are trained with CV technique. Moreover, Figure 18 A and B graphically illustrate the summary of the experimental results.

5.1.2.2 k-NN imputation with SMOTE

Then, we consider training the models by substituting the missing datapoints using k-NN imputation technique together with selecting the most important features using PCA and UFS. Table 6 shows the characteristic of the dataset used after preprocessing and feature selection.

Table 7 summarizes the results obtained when the models are trained with PCA and UFS features selection techniques when the models are trained with 80|20 – split validation technique. Similarly, the same table reports the results obtained when the models are trained with CV technique. Figure 18 C, D, E, and F show the results obtained using this method versus two considered feature selection techniques and two considered validation techniques. As can be seen from Table 11 and Table 7, the models performed better when trained with features selected using PCA because, PCA selects 219 as important features, whereas UFS selects 38. Thus, the reduction of the features selected caused a huge impact on the performance of the models. Possibly most important features that the models learn from are discarded when UFS is applied. From the classifiers considered, almost all the models performed better when validated with k-Fold cross validation technique. For instance, XGBoost plus PCA features

selection, where the model classifies the two classes (pass and fail) way better than it performed when trained with 80|20 split validation technique.

5.1.2.3 Discussion on the results obtained using SMOTE

Table 11 and Table 7 summarize the results obtained using two different features selection techniques as well as the comparison between the two validation techniques. Likewise, the comparison of these models has been reported based on which model among the models handled the dataset well in giving better performance. Figure 18 shows the summary of the achieved results from the developed classifier models using six proposed methodologies: MI + PCA + SMOTE + 80|20 split, MI + PCA + SMOTE + CV, k-NNI + PCA + SMOTE + 80|20 split, k-NNI + UFS + SMOTE +80|20 split, k-NNI + PCA + SMOTE + CV, and k-NNI + UFS + SMOTE +CV.

Based on the results obtained, a drastic performance decrement was observed when the models are trained with the features selected using UFS (SELECTFDR) (Figure 18 D and F), with the exception of MLP, in which the model performs equally in terms of recall using 80|20 split technique, and in terms of specificity and precision using CV technique. SVC and RF performed really well considering both validation techniques with a slight difference in their metrics of performance.

Amongst all the models trained, NB failed to deliver good performance metrics when the model is trained using features selected from PCA features selection plus 80|20 split technique. Similarly, it failed to deliver good results when trained with features selected from UFS, but with a better performance only in terms of specificity of 0.9422.

It is observed that, SVC and RF are the best performing models developed from both validation techniques considered when features are selected using PCA features selection technique. GBT and AdaBoost gave an approximate similar trend performance in all the four approaches examined (PCA + 80|20 split PCA + CV, UFS + 80|20 split UFS + CV). However, GBT outperforms AdaBoost in all the cases.

Moreover, Figure 18 shows the results comparison. From the figure it can be clearly seen that almost all the models performed when trained with features selected from PCA. RF, MLP XGBoost and GBT. RF and MLP outperformed all the other models trained than any other model on the four techniques considered. This shows that the two models can be trained with features selected from both PCA and UFS. Since there is no much differences in their performance metrics measures.

However, SVC, LR, LDA and NB tend to show differences in their metrics of performance as the models are trained with different features selected from the two features selection techniques. Similarly, the performance metrics of SVC + PCA excelled the performance of SVC + UFS in both two validation techniques. Thus, we could conclude that SVC is more compatible with the features selected using PCA, as it performs better with those features selected than with the features selected using UFS. DT and AdaBoost models performed moderately better on the four techniques followed.

Table 7 reports the overall summary of the results obtained using SMOTE synthetic data generation technique against six different methodologies. These methodologies include MI + PCA + SMOTE + 80|20 split; MI + PCA + SMOTE + CV; k-NNI + PCA + SMOTE + 80|20 split; k-NNI + UFS + SMOTE + 80|20 split; k-NNI + PCA +

SMOTE + CV; and k-NNI + UFS + SMOTE + CV. Th accuracy metric of performance tells the overall score of a developed classification model. From Table 7, in terms of accuracy, RF outpaces all the other developed models considering all the six methodologies considered. However, with an exception of 'k-NNI + PCA + SMOTE + 80|20 split' technique, where SVC slightly performs better than RF with a score of 0.9983. NB records the least performance in all cases (all six methodologies).

MLP, XGBoost RF, and GBT performed best with MI + PCA + SMOTE + 80|20 split technique and least with k-NNI + UFS + SMOTE + CV. A reduction in performance is experienced the moment CV validation technique was applied in all the case rather than with 80|20 split validation technique, and a further reduction when UFS features selections was used. In contrast to MI, k-NN produces worse results when MLP was trained. This shows that, MLP learns better with features that are preprocessed with mean imputation.

LR, DT, NB, LDA, SVC, and AdaBoost performed best with k-NNI + PCA + SMOTE + 80|20 split technique and least with k-NNI + UFS + SMOTE + CV technique. This means that, these models learn better from the features preprocessed using k-NN imputation and selected using PCA and when trained with 80|20 split validation technique rather with CV. k-NN imputation works better for these models, 80|20 split works better compared to CV and PCA for features selection.

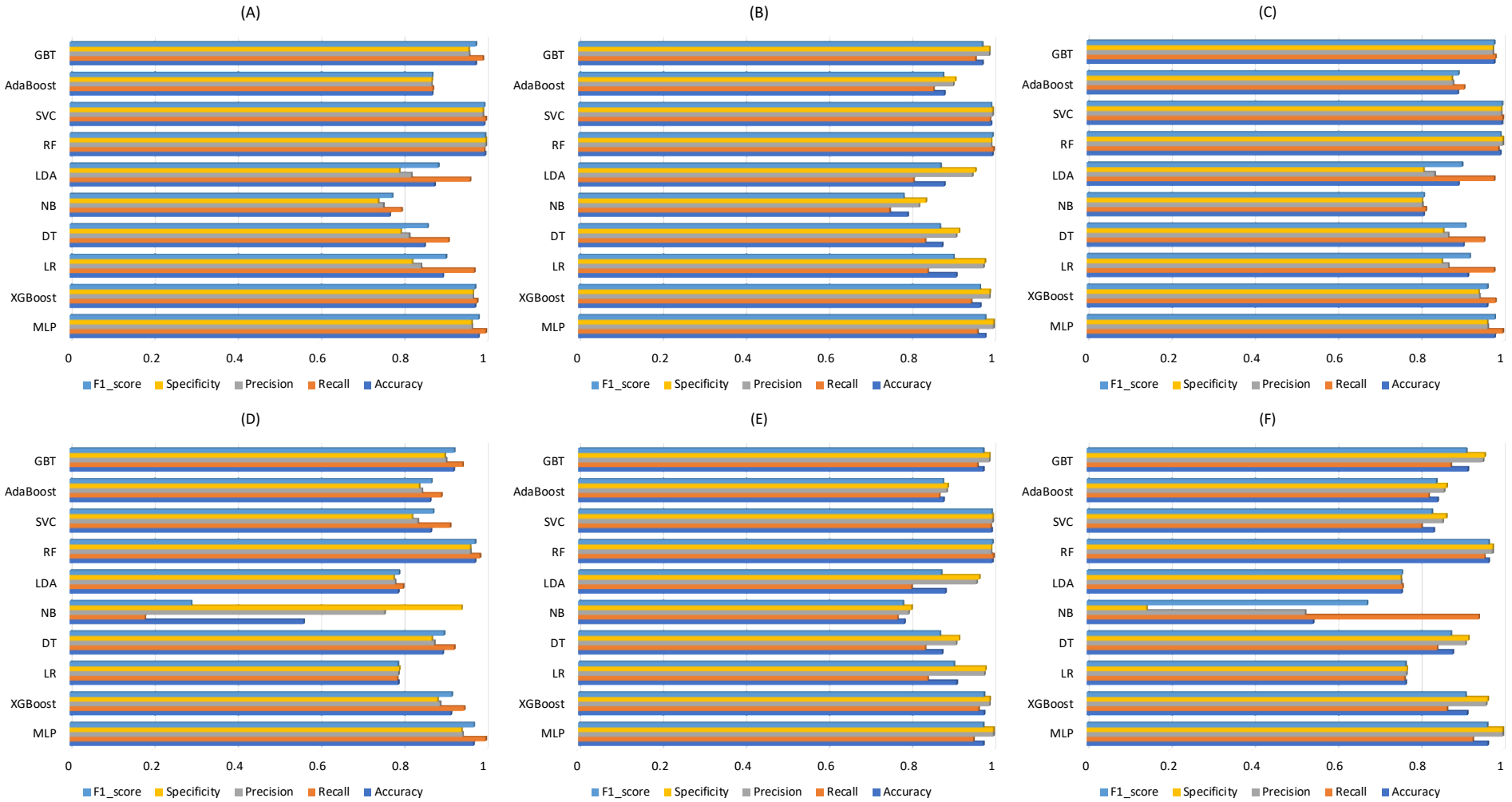

Figure 18: Results summary - (A) MI + PCA + SMOTE + 80|20 split. (B) MI + PCA + SMOTE + CV. (C) k-NNI + PCA + SMOTE + 80|20 split. (D) k-NNI + UFS + SMOTE +80|20 split. (E) k-NNI + PCA + SMOTE + CV. (F) k-NNI + UFS + SMOTE +CV.

5.1.3 Proposed methodology with Borderline-SMOTE SVM

In this section, the overall proposed methodology is analyzed considering the implementation of Borderline SMOTE SVM synthetic data generation technique. Also, the effect of mean and k-NN imputation techniques are discussed separately in the following subsections. Table 6 reports the dataset descriptions used in this section.

Similarly, like from the previous section (section 5.1.2), in this section too, two validation techniques are considered including 80|20 split and k-Fold cross validation. All the ten models are trained using 80|20 split validation technique. However, different number of k is considered while training using k-Fold cross validation technique. 7-Fold is considered for MLP, NB, LDA, RF, AdaBoost, and GBT; 13-Fold for XGBoost; 3-Fold for LR and SVC using PCA; 7-Fold using UFS and 10-Fold for DT. The methodologies proposed and studied in this section involve MI + PCA + BSMOTE-SVM + 80|20 split, MI + PCA + BSMOTE-SVM + CV, k-NNI + PCA + BSMOTE-SVM + 80|20 split, k-NNI + UFS + BSMOTE-SVM + 80|20 split, k-NNI + PCA + BSMOTE-SVM + CV, and k-NNI + UFS + BSMOTE-SVM + CV.

5.1.3.1 BSMOTE-SVM with mean imputation

First, we used mean imputation to replace the missing values then applied the proposed methodologies and then, trained the models. Table 6 reports the dataset descriptions used. PCA is used in selecting the most important features and for synthetic data generation within the minority class, borderline SMOTE-SVM is used. The models are using 80|20 slip validation technique and k-Fold CV technique. Confusion matrix and performance metrics of the results obtained are shown in Table 11and Table 7, respectively. Figure 19 A and B illustrate the models' metrics of performance results using mean imputation.

5.1.3.2 BSMOTE-SVM with k-NN imputation

Then, the effects of k-NN missing datapoints imputation is analyzed alongside BSMOTE-SVM, and PCA and UFS features selection techniques using both two validation techniques involving 80|20 split and k-Fold CV. The dataset characteristics used is reported in Table 6. Ten classifier models were developed and analyzed including MLP, XGBoost, LR, DT, NB, LDA, RF, SVC, AdaBoost, and GBT. Confusion matrix obtained from the experiments are given in Table 11 and Table 7 report the experimental results of 80|20 – split validation technique and k-Fold CV. Figure 19 C – F graphically show the experimental results obtained using k-NNI and BSMOTE-SVM synthetic datapoints generation technique.

5.1.3.3 Discussion on the results obtained using BSMOTE-SVM

Table 7 and Figure 19 report the overall summary of the results obtained using BSMOTE-SVM synthetic data generation technique against five different methodologies. These methodologies include MI + PCA + BSMOTE-SVM + 80|20 split; k-NNI + PCA + BSMOTE-SVM + 80|20 split; k-NNI + UFS + BSMOTE-SVM + 80|20 split; k-NNI + PCA + BSMOTE-SVM + CV; and k-NNI + UFS + BSMOTE-SVM + CV. Contrary to the performance of the developed models, NB failed to deliver good performance in all five considered techniques, as it gave worst performance in all cases, like from the previous section.

SVC was found to be the best performing algorithm with overall accuracy score of 1.0000 when trained using k-NNI + UFS + BSMOTE-SVM + CV technique. It also outpaces all other algorithms in terms of accuracy when ‘MI + PCA + BSMOTE-SVM + 80|20 split’ and ‘k-NNI + PCA + BSMOTE-SVM + 80|20 split’ techniques are considered. RF outperformed all other algorithms when k-NNI + UFS + BSMOTE-SVM + 80|20 split technique was applied. MLP and RF performed equally with an

overall accuracy score of 0.9639 using k-NNI + UFS + BSMOTE-SVM + CV technique. Similarly, they outperformed the other algorithms. MLP, XGBoost, LR, LDA, RF, and GBT produced worse performance when trained with k-NNI + UFS + BSMOTE-SVM + CV technique. NB, SVC, and AdaBoost deliver worst performance with k-NNI + UFS + BSMOTE-SVM + 80|20 split, and DT with k-NNI + PCA + BSMOTE-SVM + 80|20 split.

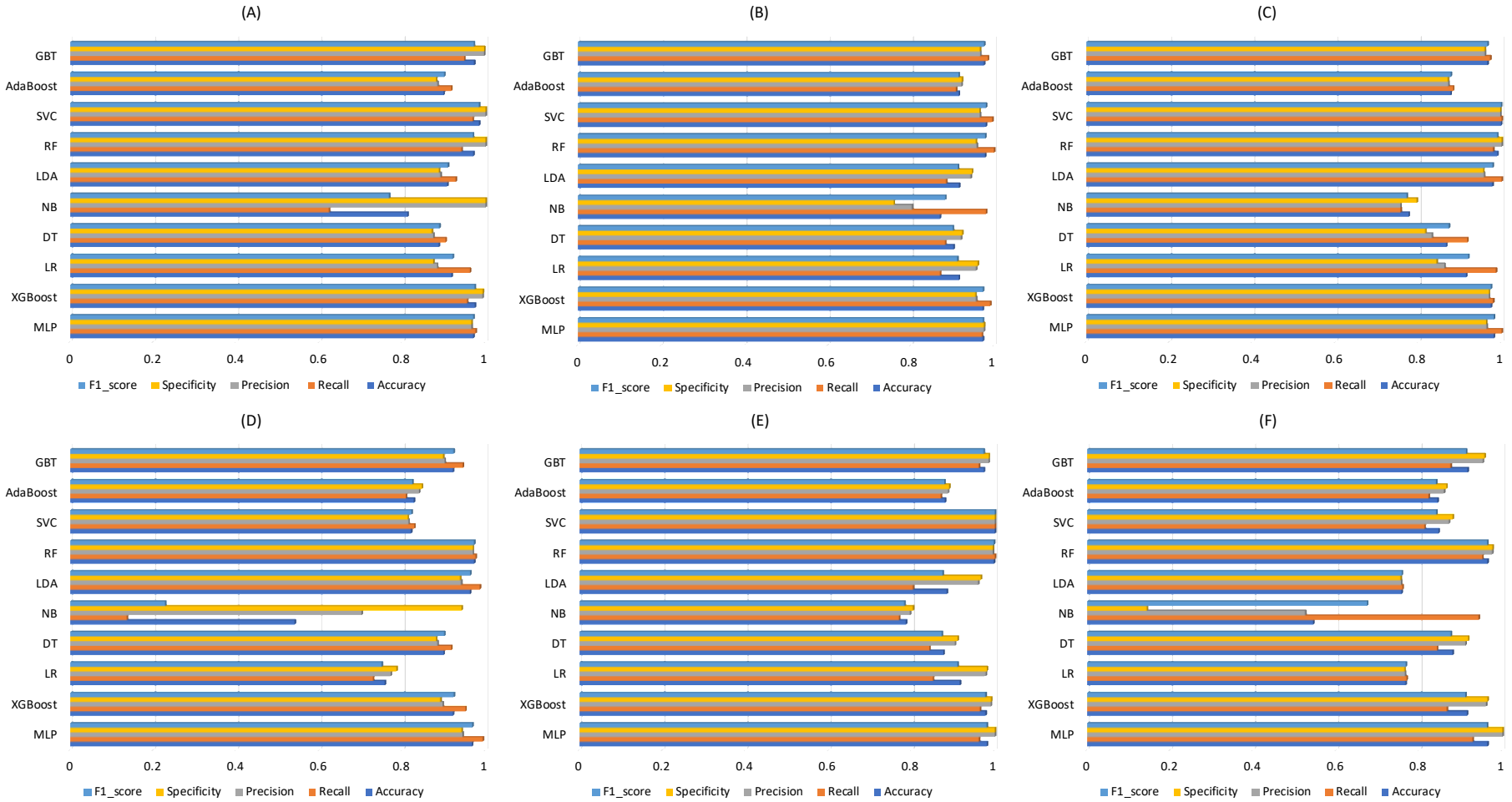

Figure 19: Results summary – (A) MI + PCA + BSMOTE-SVM + 80|20 split. (B) MI + PCA + BSMOTE-SVM + CV. (C) k-NNI + PCA + BSMOTE-SVM + 80|20 split. (D) k-NNI + UFS + BSMOTE-SVM + 80|20 split. (E) k-NNI + PCA + BSMOTE-SVM + CV. (F) k-NNI + UFS + BSMOTE-SVM + CV.

5.1.4 Proposed methodology with ADASYN

In this section the effects of ADASYN synthetic data generation are investigated. Similarly, the effects of two different features selection techniques on this method of synthetic data generation are investigated as in the previous sections. Table 6 shows the dataset characteristics used in training the classifier models in this section.

Similarly, the same approach was followed as in the previous sections (section 5.1.2and 5.1.3). Two validation techniques are considered including 80|20 split and k-Fold cross validation. All the ten models are trained using 80|20 split validation technique. For k-Fold CV, different number of k is considered while training the models. 7-Fold is considered for MLP, NB, LDA, RF, SVC, AdaBoost, and GBT. 13-Fold for XGBoost, 3-Fold for LR, and 10-Fold for DT. These algorithms are trained using six proposed methodologies including MI + PCA + ADASYN + 80|20 split, MI + PCA + ADASYN + CV, k-NNI + PCA + ADASYN + 80|20 split, k-NNI + UFS + ADASYN + 80|20 split, k-NNI + PCA + ADASYN + CV, and k-NNI + UFS + ADASYN + CV.

5.1.4.1 ADASYN with mean imputation

First, the effects of ADASYN are investigated using mean imputation technique. Table 6 shows the dataset characteristic used in training the classifier models. Moreover, PCA was used to select the most important features. Table 11 shows results of the confusion matrix obtained. Table 7 reports the performance metrics of the developed models. It is clearly shown that, the models performed well with ADASYN synthetic data generation. Where, SVC and RF performed significantly well and equally, and then followed by MLP, XGBoost, and GBT. Figure 20 A and B graphically compare and show how these three models and the other seven performed.

5.1.4.2 ADASYN with k-NN imputation

Lastly, the effects of k-NN missing datapoints imputation were analyzed alongside ADASYN synthetic data generation, and PCA and UFS features selection techniques using 80|20 split and CV validation techniques. The dataset characteristics used is reported in Table 6. Ten classifier models were developed and analyzed including MLP, XGBoost, LR, DT, NB, LDA, RF, SVC, AdaBoost, and GBT. Experimental results are reported in Table 11 and Table 7, including the confusion matrix and the performance metrics of the developed models. Figure 20 C to F graphically illustrate the comparison of the results obtained from the developed models using these techniques.

5.1.4.3 Discussion on the results obtained using ADASYN

Table 7 reports the overall summary of the results obtained using ADASYN synthetic data generation technique against six different methodologies proposed in this section. From Table 7, SVC reports an overall accuracy of 1.000 when trained with k-NNI + PCA + ADASYN + 80|20 split technique followed by RF with k-NNI + UFS + ADASYN + 80|20 split technique. Similarly, in this section NB fails to deliver good performance.

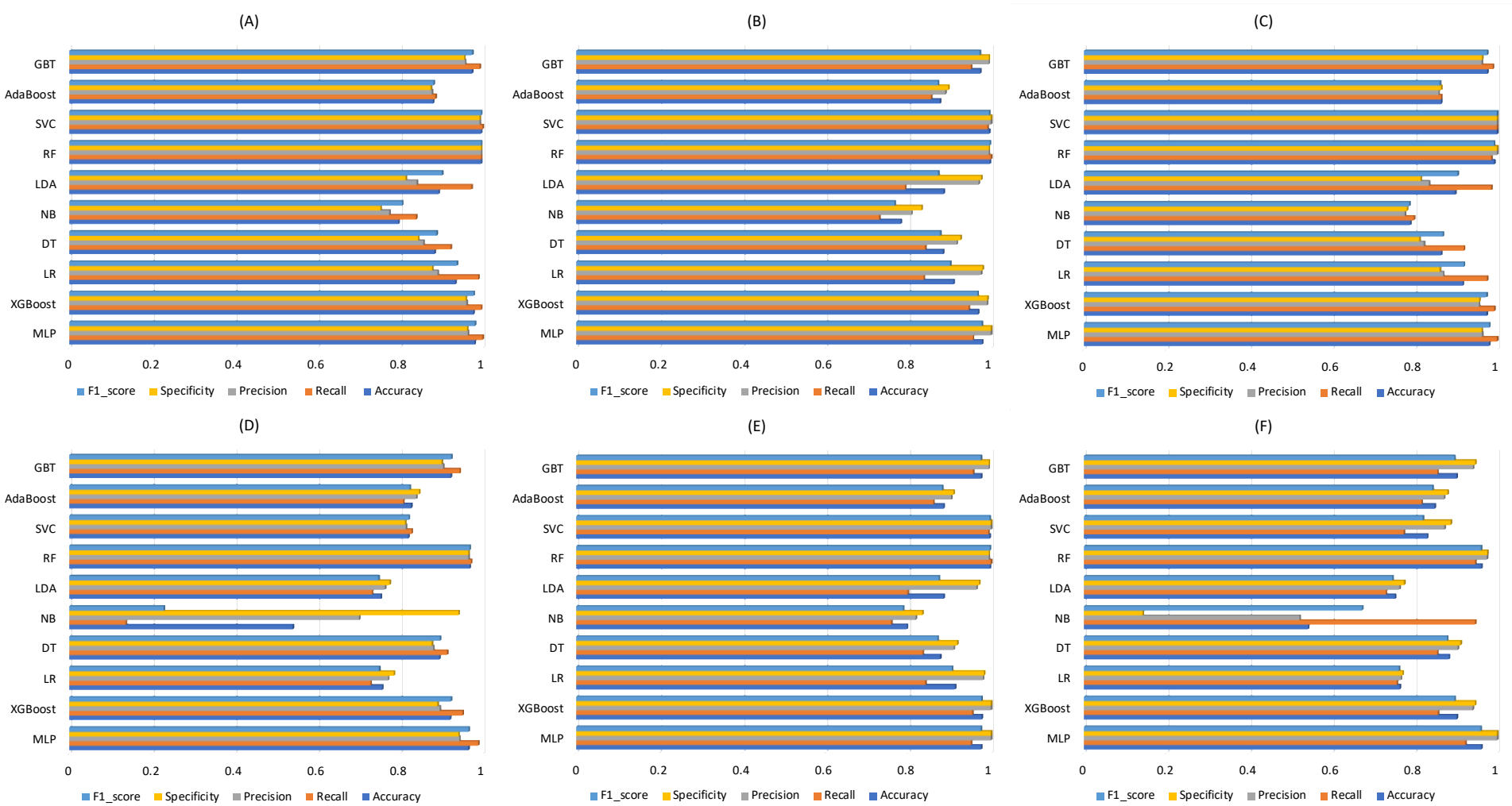

Figure 20: Results summary – (A) MI + PCA + ADASYN + 80|20 split. (B) MI + PCA + ADASYN + CV. (C) k-NNI + PCA + ADASYN + 80|20 split. (D) k-NNI + UFS + ADASYN + 80|20 split. (E) k-NNI + PCA + ADASYN + CV. (F) k-NNI + UFS + ADASYN + CV.

Table 7: Metrics of performance overall experimental results obtained using three different SDGT.

		Accuracy						Recall						Precision						Specificity						F1_score					
SDGT	Models	***MI + PCA + 80|20 split***	***MI + PCA + CV***	***k-NNI + PCA + 80|20 split***	***k-NNI + UFS + 80|20 split***	***k-NNI + PCA + CV***	***k-NNI + UFS + CV***	***MI + PCA + 80|20 split***	***MI + PCA + CV***	***k-NNI + PCA + 80|20 split***	***k-NNI + UFS + 80|20 split***	***k-NNI + PCA + CV***	***k-NNI + UFS + CV***	***MI + PCA + 80|20 split***	***MI + PCA + CV***	***k-NNI + PCA + 80|20 split***	***k-NNI + UFS + 80|20 split***	***k-NNI + PCA + CV***	***k-NNI + UFS + CV***	***MI + PCA + 80|20 split***	***MI + PCA + CV***	***k-NNI + PCA + 80|20 split***	***k-NNI + UFS + 80|20 split***	***k-NNI + PCA + CV***	***k-NNI + UFS + CV***	***MI + PCA + 80|20 split***	***MI + PCA + CV***	***k-NNI + PCA + 80|20 split***	***k-NNI + UFS + 80|20 split***	***k-NNI + PCA + CV***	***k-NNI + UFS + CV***
SMOTE	MLP	0.9829	0.9808	0.9812	0.9710	0.9760	0.9639	**1.0000**	0.9615	**1.0000**	**1.0000**	0.9519	0.9279	0.9669	**1.0000**	0.9637	0.9450	**1.0000**	**1.0000**	0.9660	**1.0000**	0.9626	**0.9422**	**1.0000**	**1.0000**	0.9832	0.9804	0.9815	0.9717	0.9754	0.9626
	XGBoost	0.9744	0.9688	0.9625	0.9164	0.9775	0.9148	0.9795	0.9464	0.9829	0.9486	0.9640	0.8661	0.9695	0.9907	0.9441	0.8907	0.9907	0.9604	0.9694	0.9911	0.9422	0.8844	0.9910	0.9640	0.9744	0.9680	0.9631	0.9187	0.9772	0.9108
	LR	0.8976	0.9107	0.9164	0.7901	0.9117	0.7669	0.9726	0.8419	0.9795	0.7877	0.8419	0.7639	0.8452	0.9762	0.8693	0.7904	0.9785	0.7686	0.8231	0.9795	0.8537	0.7925	0.9815	0.7700	0.9045	0.9041	0.9211	0.7890	0.9051	0.7662
	DT	0.8532	0.8767	0.9061	0.8976	0.8767	0.8801	0.9110	0.8356	0.9555	0.9247	0.8356	0.8425	0.8160	0.9104	0.8692	0.8766	0.9104	0.9111	0.7959	0.9178	0.8571	0.8707	0.9178	0.9178	0.8608	0.8714	0.9103	0.9000	0.8714	0.8754
	NB	0.7696	0.7938	0.8106	0.5631	0.7861	0.5444	0.7979	0.7500	0.8151	0.1815	0.7692	0.9426	0.7540	0.8211	0.8068	0.7571	0.7960	0.5253	0.7415	0.8373	0.8061	**0.9422**	0.8029	0.1442	0.7754	0.7839	0.8109	0.2928	0.7824	0.6747
	LDA	0.8771	0.8822	0.8942	0.7901	0.8846	0.7572	0.9623	0.8077	0.9795	0.8014	0.8029	0.7596	0.8216	0.9492	0.8363	0.7826	0.9598	0.7560	0.7925	0.9567	0.8095	0.7789	0.9663	0.7548	0.8864	0.8727	0.9022	0.7919	0.8743	0.7578
	RF	**0.9983**	**0.9976**	0.9949	**0.9744**	**0.9976**	**0.9663**	0.9966	**1.0000**	0.9897	0.9863	**1.0000**	**0.9567**	**1.0000**	0.9952	**1.0000**	**0.9632**	0.9952	0.9755	**1.0000**	0.9952	**1.0000**	0.9626	0.9952	0.9760	**0.9983**	**0.9976**	0.9948	**0.9746**	**0.9976**	**0.9660**
	SVC	0.9966	0.9949	**0.9983**	0.8686	0.9959	0.8347	**1.0000**	0.9918	**1.0000**	0.9144	0.9938	0.8049	0.9932	0.9979	0.9966	0.8370	0.9979	0.8559	0.9932	0.9979	0.9966	0.8231	0.9979	0.8645	0.9966	0.9949	**0.9983**	0.8740	0.9959	0.8296
	AdaBoost	0.8720	0.8822	0.8925	0.8669	0.8801	0.8438	0.8733	0.8558	0.9075	0.8938	0.8702	0.8221	0.8703	0.9036	0.8804	0.8474	0.8873	0.8593	0.8707	0.9087	0.8776	0.8401	0.8900	0.8654	0.8718	0.8790	0.8938	0.8700	0.8786	0.8403
	GBT	0.9761	0.9736	0.9795	0.9232	0.9760	0.9161	0.9932	0.9567	0.9829	0.9452	0.9615	0.8750	0.9603	0.9900	0.9762	0.9049	0.9901	0.9529	0.9592	0.9904	0.9762	0.9014	0.9904	0.9569	0.9764	0.9731	0.9795	0.9246	0.9756	0.9123
BSMOTE-SVM	MLP	0.9710	0.9736	0.9812	0.9676	0.9808	**0.9639**	**0.9760**	0.9712	**1.0000**	**0.9932**	0.9615	0.9279	0.9661	**0.9758**	0.9637	0.9446	**1.0000**	**1.0000**	0.9660	**0.9760**	0.9626	0.9422	**1.0000**	**1.0000**	0.9710	0.9735	0.9815	0.9683	0.9804	0.9626
	XGBoost	0.9744	0.9732	0.9744	0.9215	0.9775	0.9148	0.9555	0.9911	0.9795	0.9521	0.9640	0.8661	0.9929	0.9569	0.9695	0.8968	0.9907	0.9604	0.9932	0.9554	0.9694	0.8912	0.9910	0.964	0.9738	0.9737	0.9744	0.9236	0.9772	0.9108
	LR	0.9181	0.9158	0.9147	0.7577	0.9159	0.7668	0.9623	0.8706	0.9863	0.7295	0.8510	0.7692	0.8836	0.9571	0.8623	0.7717	0.9779	0.7656	0.8741	0.9610	0.8435	0.7857	0.9808	0.7644	0.9213	0.9118	0.9201	0.7500	0.9100	0.7674
	DT	0.8874	0.9031	0.8669	0.8993	0.8763	0.8801	0.9041	0.8828	0.9178	0.9178	0.8425	0.8425	0.8742	0.9209	0.8323	0.8845	0.9044	0.9111	0.8707	0.9236	0.8163	0.8810	0.9103	0.9178	0.8889	0.9014	0.8730	0.9008	0.8723	0.8754
	NB	0.8123	0.8702	0.7765	0.5410	0.7861	0.5444	0.6233	0.9808	0.7568	0.1370	0.7692	0.9426	**1.0000**	0.8031	0.7565	0.7018	0.7960	0.5253	**1.0000**	0.7596	0.7959	0.9422	0.8029	0.1442	0.7679	0.8831	0.7714	0.2292	0.7824	0.6747
	LDA	0.9078	0.9161	0.9778	0.9625	0.8846	0.7572	0.9281	0.8852	**1.0000**	0.9863	0.8029	0.7596	0.8914	0.9439	0.9574	0.9412	0.9598	0.7560	0.8878	0.9471	0.9558	0.9388	0.9663	0.7548	0.9094	0.9136	0.9782	0.9632	0.8743	0.7578
	RF	0.9710	0.9784	0.9898	**0.9727**	0.9976	**0.9639**	0.9418	**1.0000**	0.9795	0.9760	**1.0000**	**0.9519**	**1.0000**	0.9587	**1.0000**	**0.9694**	0.9952	0.9754	**1.0000**	0.9567	**1.0000**	**0.9694**	0.9952	0.9760	0.9700	0.9789	0.9896	**0.9727**	0.9976	**0.9635**
	SVC	**0.9846**	**0.9805**	**0.9983**	0.8208	**1.0000**	0.8465	0.9692	0.9959	**1.0000**	0.8288	**1.0000**	0.8125	**1.0000**	**0.9661**	0.9966	0.8148	**1.0000**	0.8711	**1.0000**	0.9651	0.9966	0.8129	**1.0000**	0.8804	0.9843	**0.9808**	**0.9983**	0.8217	**1.0000**	0.8408
	AdaBoost	0.8993	0.9159	0.8771	0.8276	0.8801	0.8438	0.9178	0.9087	0.8836	0.8082	0.8702	0.8221	0.8845	0.9220	0.8716	0.8399	0.8873	0.8593	0.8810	0.9231	0.8707	0.8469	0.8900	0.8654	0.9008	0.9153	0.8776	0.8237	0.8786	0.8403
	GBT	0.9727	0.9760	0.9659	0.9215	0.9736	0.9161	0.9486	0.9856	0.9726	0.9452	0.9615	0.8750	0.9964	0.9670	0.9595	0.9020	0.9852	0.9529	0.9966	0.9663	0.9592	0.8980	0.9856	0.9569	0.9719	0.9762	0.9660	0.9231	0.9732	0.9123
ADASYN	MLP	0.9815	0.9787	0.9812	0.9659	0.9762	0.9617	**1.0000**	0.9567	**1.0000**	**0.9897**	0.9519	0.9231	0.9645	**1.0000**	0.9637	0.9444	**1.0000**	**1.0000**	0.9628	**1.0000**	0.9626	0.9422	**1.0000**	**1.0000**	0.9819	0.9779	0.9815	0.9666	0.9754	0.9600
	XGBoost	0.9782	0.9693	0.9746	0.9215	0.9777	0.9022	0.9967	0.9464	0.9931	0.9521	0.9550	0.8571	0.9613	0.9907	0.9565	0.8968	1.0000	0.9412	0.9595	0.9914	0.9571	0.8912	1.0000	0.9469	0.9787	0.9680	0.9744	0.9236	0.9770	0.8972
	LR	0.9345	0.9100	0.9171	0.7577	0.9134	0.7646	0.9900	0.8378	0.9757	0.7295	0.8419	0.7577	0.8916	0.9761	0.8700	0.7717	0.9809	0.7672	0.8784	0.9801	0.8614	0.7857	0.9838	0.7714	0.9382	0.9017	0.9198	0.7500	0.9061	0.7624
	DT	0.8840	0.8847	0.8646	0.8959	0.8776	0.8836	0.9231	0.8414	0.9201	0.9144	0.8356	0.8552	0.8571	0.9173	0.8230	0.8812	0.9104	0.9051	0.8446	0.9267	0.8119	0.8776	0.9189	0.9116	0.8889	0.8777	0.8689	0.8975	0.8714	0.8794
	NB	0.7966	0.7825	0.7902	0.5410	0.7976	0.5431	0.8395	0.7308	0.7986	0.1370	0.7596	**0.9471**	0.7747	0.8085	0.7770	0.7018	0.8187	0.5225	0.7534	0.8326	0.7822	0.9422	0.8349	0.1429	0.8058	0.7677	0.7877	0.2292	0.7880	0.6735
	LDA	0.8941	0.8863	0.8985	0.7543	0.8860	0.7530	0.9732	0.7933	0.9861	0.7329	0.7990	0.7308	0.8410	0.9706	0.8353	0.7643	0.9653	0.7638	0.8142	0.9766	0.8152	0.7755	0.9717	0.7751	0.9023	0.8730	0.9045	0.7483	0.8743	0.7469
	RF	**0.9966**	0.9976	0.9932	**0.9693**	0.9976	0.9617	0.9967	**1.0000**	0.9861	0.9726	**1.0000**	**0.9471**	**0.9967**	0.9952	**1.0000**	**0.9660**	0.9952	0.9752	0.9966	0.9953	**1.0000**	**0.9660**	0.9953	0.9762	0.9967	0.9976	0.9930	**0.9693**	0.9976	0.9610
	SVC	**0.9966**	**0.9960**	**1.0000**	0.8208	**0.9969**	**0.8311**	**1.0000**	**0.9918**	**1.0000**	0.8288	0.9938	0.7741	0.9934	**1.0000**	**1.0000**	0.8148	**1.0000**	0.8727	0.9932	**1.0000**	**1.0000**	0.8129	**1.0000**	**0.8878**	0.9967	**0.9959**	**1.0000**	0.8217	**0.9969**	**0.8205**
	AdaBoost	0.8807	0.8771	0.8646	0.8276	0.8857	0.8489	0.8863	0.8558	0.8646	0.8086	0.8612	0.8173	0.8775	0.8900	0.8586	0.8399	0.9045	0.8718	0.8750	0.8977	0.8647	0.8469	0.9100	0.8804	0.8819	0.8725	0.8616	0.8237	0.8824	0.8437
	GBT	0.9748	0.9740	0.9763	0.9232	0.9762	0.9019	**0.9933**	0.9519	0.9896	0.9452	0.9569	0.8558	0.9581	0.9950	0.9628	0.9049	0.9950	0.9418	0.9561	0.9953	0.9637	0.9014	0.9953	0.9476	0.9754	0.9730	0.9760	0.9246	0.9756	0.8967

SDGT: Synthetic data generation technique; MI: Mean imputation; k-NNI: k-NN imputation; PCA: Principal component analysis; UFS: Univariate features selection; 80|20 split: 80|20 split validation technique; CV: k-Fold cross validation technique;

5.1.5 Discussion and comparison on the overall experimental results

In this work, ten prediction machine learning classifiers have been developed, namely; MLP (ANN), XGBoost, LR, DT, NB, LDA, RF, SVC, AdaBoost, and GBT. Semiconductor manufacturing process dataset was used to evaluate and validate the proposed diagnosis models. Moreover, this section discusses and compares the results obtained from different perspectives. Note that, this section discusses and compares the performance of the developed models only based on their accuracy. This is because, accuracy tells the overall performance of a prediction classifier.

5.1.5.1 Effects of MRD, dataset preprocessing and SMOTE-based proposed methodology

Table 9 reports an accuracy-based results comparison amongst MRD, MRD with main steps effects of dataset preprocessing, and proposed methodology using SMOTE and 80|20 split validation technique. For better visualization, Figure 21 graphically shows how the performance of the models is affected with each and every step of data preprocessing starting from when trained with modified raw data. At first, when the algorithms were trained with raw and modified raw datasets, they failed to deliver any result, as the dataset contained missing values, however, with exception of XGBoost that managed to deliver some performance. Second, all the algorithms managed to deliver some performance when the missing values were substituted using missing values imputation methods. Third, the effect of features selection technique using PCA was analyzed on the modified raw dataset. However, the results obtained were similar with those obtained from previous steps. That is, at this moment PCA has no significant effect on the developed models' performance. This is due to the dataset containing imbalanced classes. Consequently, the trained algorithms in the aforementioned steps failed to classify the minority classes. Later, the effect of

synthetic data generation was analyzed. SMOTE was applied in order to generate synthetic data within the minority class, so as to balance this problem of imbalance. As a result, changes were observed in the models' performance. Refer to section 5.1.1.1.1 through 5.1.1.1.5 to see the overall results and analysis. In terms of accuracy (Table 9 and Figure 21), MLP, XGBoost, SVC, and RF are the best examples that clearly show the effect of each and every step together with their accuracy metric of performance.

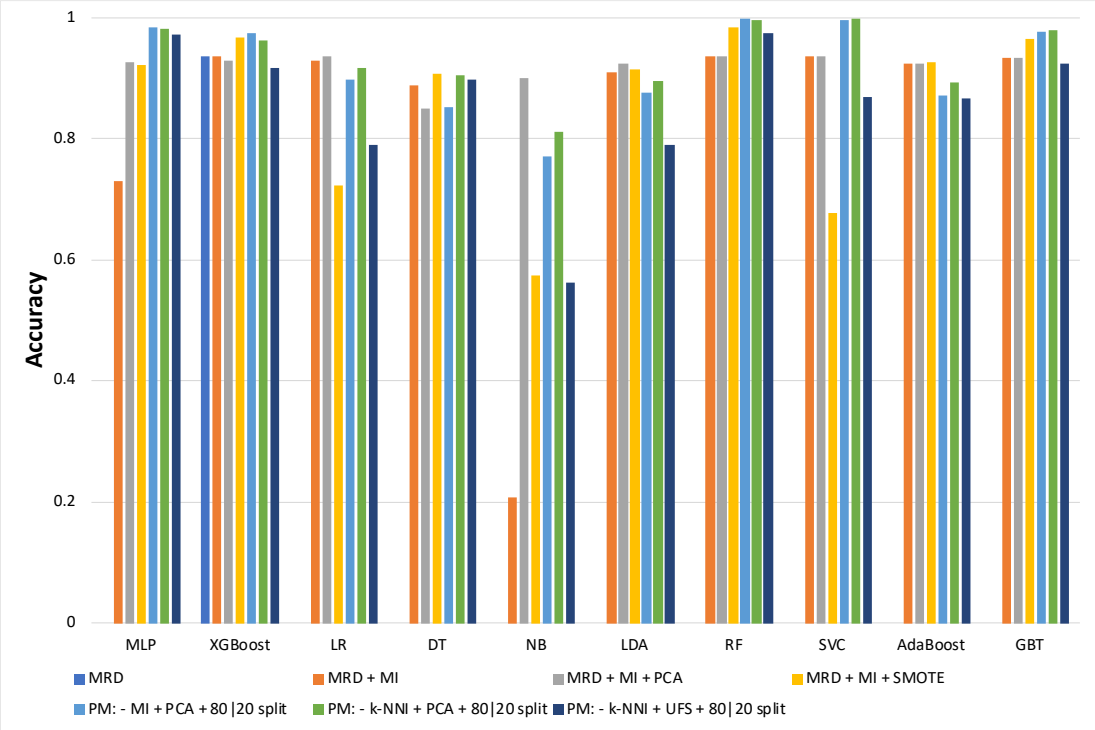

Figure 21: Results comparison – MRD, MRD + effects of data preprocessing steps, and proposed methodology with SMOTE using 80|2-split.

5.1.5.2 Overall results comparison within the proposed methodologies

Table 7 and Table 11 report the comparison summary of the experimental results obtained using SMOTE, BSMOTE-SVM and ADASYN respectively. Based on the results obtained from these tables (Table 7 and Table 11), Figure 22 was developed. The figure reports the overall comparison amongst six considered methodologies

versus three adopted synthetic data generation techniques (including SMOTE, BSMOTE-SVM, and ADASYN) on ten ML models.

The overall results of MLP are shown in Figure 22A. The model performs best with MI + PCA + 80|20 split using SMOTE followed by ADASYN. However, with the same training technique, a drastic performance decrement was seen when BSMOTE-SVM technique was applied. All the three synthetic data generation techniques performed equally with 'k-NNI + PCA + 80|20 split' that produced an accuracy score of 0.9812. With k-NNI + UFS + 80|20 split technique, ADASYN produced least score of 0.9659 followed by BSMOTE-SVM. SMOTE gave the best performance with the same technique. Surprisingly, BSMOTE-SVM outpaces SMOTE with k-NNI + PCA + CV by producing an accuracy score of 0.9808 and 0.976 for SMOTE. SMOTE and BSMOTE-SVM performed equally with k-NNI + UFS + CV.

As illustrated in Figure 22B, k-NNI + PCA + 80|20 split with SMOTE reported least result when XGBoost was trained, where, with the same technique, BSMOTE-SVM and ADASYN reported a score of 0.9744 and 0.9746, respectively, in where they outperformed SMOTE. BSMOTE-SVM and ADASYN performed equally and outpaced SMOTE with k-NNI + UFS + 80|20 split. SMOTE and BSMOTE-SVM performed equally with k-NNI + PCA + CV and k-NNI + UFS + CV techniques. In contrast, XGBoost performed best with k-NNI + PCA + CV technique using both SMOTE and BSMOTE-SVM.

Figure 22C shows the overall experimental results comparison for LR. The algorithm performed best with k-NNI + PCA + 80|20 split technique, in where all the three synthetic data generation techniques produced excellent performance with a slight

difference in their overall accuracy score. SMOTE reports best score with k-NNI + UFS + 80|20 split, while BSMOTE-SVM and ADASYN produced lower results and performed equally with the same technique. Similarly, similar results were produced with k-NNI + UFS + CV using SMOTE and BSMOTE-SVM.

The overall results of DT are shown in Figure 22D. Best score was obtained when SMOTE was applied using k-NNI + PCA + 80|20 split. When DT was trained with k-NNI + UFS + 80|20 split, BSMOTE-SVM performed best, then SMOTE, followed by ADASYN. With k-NNI + PCA + CV, both SMOTE and BSMOTE-SVM slightly performed differently, and performed equally with k-NNI + UFS + CV.

Figure 22E shows the overall results of NB. SMOTE with k-NNI + PCA + 80|20 split gave the best performance, while k-NNI + UFS + 80|20 split with BSMOTE-SVM and ADASYN gave the least performance. BSMOTE-SVM and ADASYN produced equal result with k-NNI + PCA + CV, and also with k-NNI + UFS + CV. However, in comparison with k-NNI + PCA + CV, both synthetic data generation techniques produce lower accuracy with k-NNI + UFS + CV technique. In summary, SMOTE gave better performance when the algorithm was trained with these techniques of synthetic data generation. Moreover, BSOMTE-SVM performed equally with SMOTE using k-NNI + PCA + CV. NB experienced an increase in its performance when trained with features selected from UFS.

Figure 22F illustrates the overall experimental results obtained when LDA was trained. Amazingly, with LDA, BSMOTE-SVM outpaces SMOTE and ADASYN using k-NNI + PCA + 80|20 split and k-NNI + UFS + 80|20 split techniques. Least performance was obtained when ADASYN was used with k-NNI + UFS + 80|20 split.

With k-NNI + PCA + CV technique, both SMOTE and BSMOTE-SVM performed equally with an accuracy overall score of 0.8846. Likewise, they performed equally with k-NNI + UFS + CV, but, with lower accuracy score of 0.7572. based on the results, BSMOTE-SVM works better for LDA and managed to outpace all the other techniques considered while training the algorithm for fault diagnosis using SECOM dataset.

Comparison of the results illustration of RF is shown in Figure 22G. From the figure, it is clearly shown that, MI + PCA + 80|20 split with SMOTE works best for RF, followed by k-NNI + PCA + CV with SMOTE and BSMOTE-SVM where both techniques performed the same. With the implementation of MI + PCA + 80|20 split, BSMOTE-SVM produced least accuracy score. Similarly, BSMOTE-SVM produced lower score with k-NNI + UFS + CV technique compared to SMOTE.

Figure 22H shows the results comparison of SVC. SMOTE and ADASYN performed the dame with MI + PCA + 80|20 split followed by BSMOTE-SVM. Using k-NNI + PCA + 80|20 split technique provides the highest accuracy score, with a slight difference in the overall accuracy score of the three synthetic data generation technique. ADASYN produced the best performance with k-NNI + PCA + 80|20 split and BSMOTE-SVM with k-NNI + PCA + CV, where they generate an accuracy of 1.0000. Both BSMOTE-SVM and ADSYN produced least performance with k-NNI + UFS + 80|20 split. With k-NNI + UFS + CV, BSMOTE-SMOTE generates better performance compared to SMOTE.

Least performance was generated when AdaBoost was trained with k-NNI + UFS + 80|20 split using BSMOTE-SVM and ADASYN, see Figure 22I. The best accuracy

score was achieved when k-NNI + PCA + 80|20 split was applied using SMOTE. Equal performance was achieved from both SMOTE and BSMOTE-SVM using k-NNI + PCA + CV and k-NNI + UFS + CV techniques.

Lastly, Figure 22J shows GBT experimental results. Best performance was achieved with k-NNI + PCA + 80|20 split using SMOTE, and least using BSMOTE-SVM. Considering k-NNI + PCA + CV, SMOTE slightly performs better than BSMOTE-SVM. Worse results were obtained with k-NNI + UFS + CV suing SMOTE and BSMOTE-SVM, where both synthetic data generation techniques performed equally.

5.1.6 Experimental results comparison with similar studies from the literature

Comparison between results obtained from this work with similar works from the literature has been summarized, tabulated and reported in Table 8. Best models with highest performance metrics were selected from the literature and are compared with similar models from this thesis. It can clearly be seen that, from the table, our proposed models outsmart the proposed classifiers from the literature in terms of accuracy metric of performance. Based on the comparative analysis, RF and SVC turned out to be the best developed classifiers models, as they beat all the proposed models in this work and also when compared with recent studies from the literature.

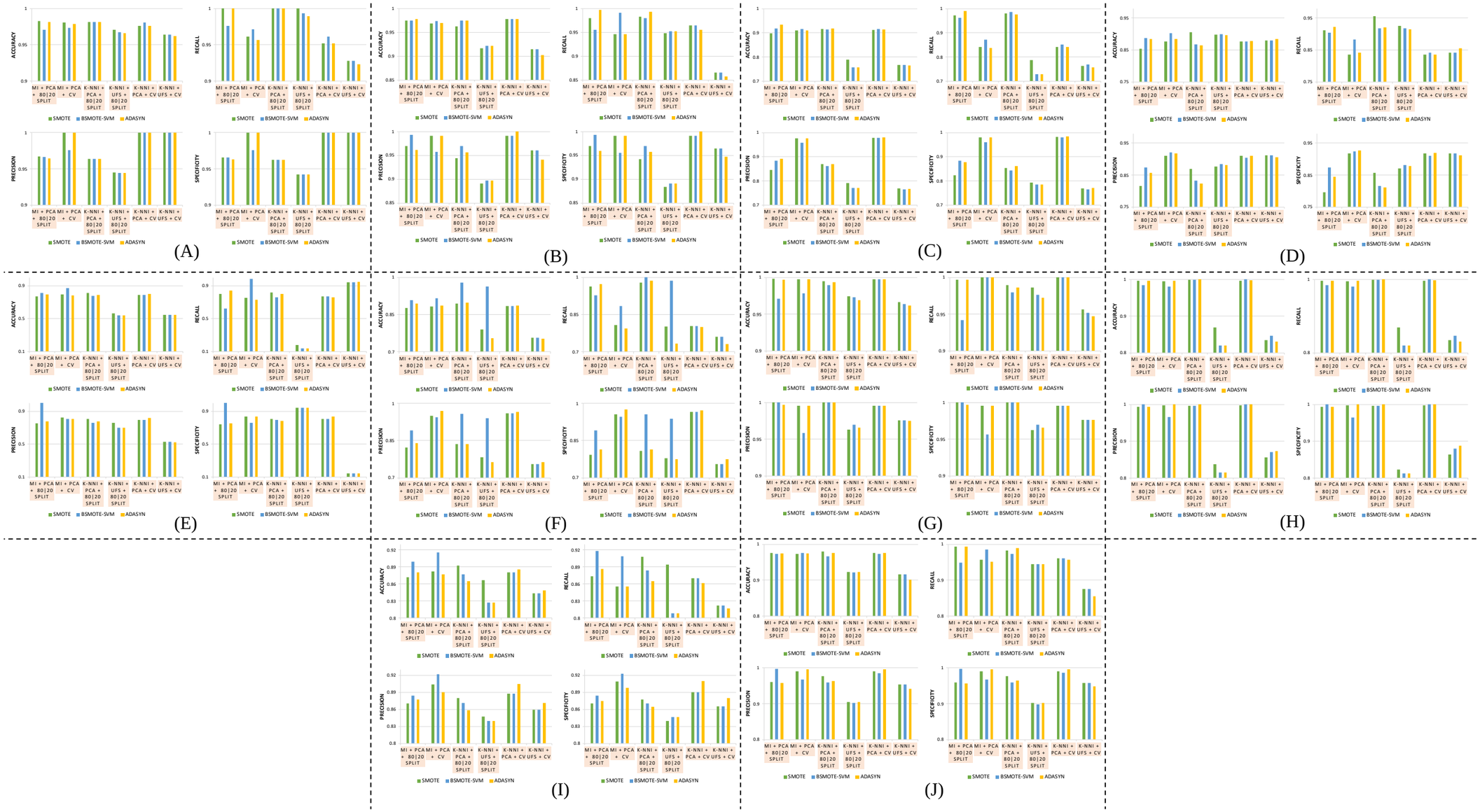

Figure 22: Overall comparison of the experimental results; (A) MLP; (B) XGBoost; (C) LR; (D) DT; (E) NB; (F) LDA; (G) RF; (H) SVC; (I) AdaBoost; (J) GBT.

Table 8: Results comparison with recent similar works form the literature.

Classifier Model	Reference	Validation Technique	Imputation	Features Selection	SDGT	Accuracy	Recall	Precision	Specificity	F1_score
LR	[87]	10-Fold	In-painting	SELECTFDR	SMOTE	0.7441	0.6538	0.1570	0.7505	-
	[88]	5-Fold	-	-	SMOTE	0.8469	-	-	-	-
	[3]	3-Fold	-	PCA	-	0.7100	0.3100	**1.0000**	**1.0000**	0.4600
	[4]	10-Fold	-	PCA	Rare case boosting	-	**1.0000**	0.3010	0.3350	0.4630
	this work	3-Fold	k-NN	PCA	SMOTE	0.9117	0.8419	0.9785	0.9815	0.9051
	this work	80|20 split	k-NN	PCA	SMOTE	0.9164	0.9795	0.8693	0.8537	0.9211
	this work	80|20 split	k-NN	PCA	BSMOTE-SVM	0.9147	0.9863	0.8623	0.8435	0.9201
	this work	7-Fold	k-NN	PCA	BSMOTE-SVM	0.9159	0.8510	0.9779	0.9808	0.9100
	this work	80|20 split	k-NN	PCA	ADASYN	**0.9171**	0.9757	0.8700	0.8614	0.9198
	this work	80|20 split	k-NN	SELECTFDR	ADASYN	0.7577	0.7295	0.7717	0.7857	0.7500
SVM	[87]	10-Fold	In-painting	SELECTFDR	SMOTE	-	0.6442	0.1572	0.7546	-
	[3]	3-Fold	-	PCA	-	0.4400	0.2300	0.6200	0.7700	0.3400
	[90]	train|test split	-	PCA	Boosting	0.6830	0.7120	-	-	0.6590
	this work	80|20 split	k-NN	PCA	SMOTE	0.9983	**1.0000**	0.9966	0.9966	0.9983
	this work	3-Fold	k-NN	PCA	SMOTE	0.9959	0.9938	0.9979	0.9979	0.9959
	this work	80|20 split	k-NN	PCA	BSMOTE-SVM	0.9983	**1.0000**	0.9966	0.9966	0.9983
	this work	7-Fold	k-NN	PCA	BSMOTE-SVM	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
	this work	80|20 split	k-NN	PCA	ADASYN	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
	this work	80|20 split	k-NN	SELECTFDR	ADASYN	0.8208	0.8288	0.8148	0.8129	0.8217
RF	[87]	10-Fold	In-painting	SELECTFWE	SMOTE	-	0.4615	0.1337	0.7874	-
	[92]	70|30 split	-	-	SMOTE	0.8930	0.2330	0.2060	0.9380	0.2190
	[88]	5-Fold	-	-	SMOTE	0.9439	-	-	-	-
	this work	80|20 split	k-NN	PCA	SMOTE	0.9949	0.9897	**1.0000**	**1.0000**	0.9948
	this work	7-Fold	k-NN	PCA	SMOTE	**0.9976**	**1.0000**	0.9952	0.9952	0.9976
	this work	80|20 split	mean	PCA	BSMOTE-SVM	0.9710	0.9418	**1.0000**	**1.0000**	0.9700
	this work	7-Fold	k-NN	PCA	BSMOTE-SVM	**0.9976**	**1.0000**	0.9952	0.9952	0.9976
	this work	80|20 split	k-NN	PCA	ADASYN	0.9932	0.9861	**1.0000**	**1.0000**	0.9930
	this work	80|20 split	mean	PCA	ADASYN	0.9966	0.9967	0.9967	0.9966	0.9967
DT	[3]	3-Fold	-	PCA	-	0.8600	0.4700	**1.0000**	**1.0000**	0.6400
	[4]	10-Fold	-	PCA	Rare case boosting	-	**1.0000**	0.4720	0.1610	0.6410
	this work	10-Fold	k-NN	PCA	SMOTE	0.8767	0.8356	0.9104	0.9178	0.8714
	this work	80|20 split	k-NN	PCA	SMOTE	**0.9061**	0.9555	0.8692	0.8571	0.9103
	this work	80|20 split	k-NN	PCA	BSMOTE-SVM	0.8669	0.9178	0.8323	0.8163	0.8730
	this work	80|20 split	k-NN	SELECTFDR	BSMOTE-SVM	0.8993	0.9178	0.8845	0.8810	0.9008
	this work	80|20 split	k-NN	PCA	ADASYN	0.8646	0.9201	0.8230	0.8119	0.8689
	this work	80|20 split	k-NN	SELECTFDR	ADASYN	0.8959	0.9144	0.8812	0.8776	0.8975
NB	[3]	3-Fold	-	PCA	-	0.6600	0.2300	0.7500	0.9500	0.3600
	[90]	train|test split	-	PCA	Boosting	0.6810	0.6970	-	-	0.6650
	[4]	10-Fold	-	PCA	Rare case boosting	-	0.7460	0.2340	0.3520	0.3560

	this work	80|20 split	k-NN	PCA	SMOTE	**0.8106**	0.8151	0.8068	0.8061	0.8109
	this work	7-Fold	k-NN	PCA	SMOTE	0.7861	0.7692	0.7960	0.8029	0.7824
	this work	80|20 split	k-NN	PCA	BSMOTE-SVM	0.7765	0.7568	0.7565	0.7959	0.7714
	this work	7-Fold	k-NN	PCA	BSMOTE-SVM	0.7861	0.7692	0.7960	0.8029	0.7824
	this work	80|20 split	k-NN	PCA	ADASYN	0.7902	0.7986	0.7770	0.7822	0.7877
	this work	80|20 split	k-NN	SELECTFDR	ADASYN	0.5410	0.1370	0.7018	0.9422	0.2292
MLP (ANN)	[88]	5-Fold	-	-	SMOTE	0.8893	-	-	-	-
	this work	80|20 split	k-NN	PCA	SMOTE	**0.9812**	**1.0000**	0.9637	0.9626	0.9815
	this work	7-Fold	k-NN	PCA	SMOTE	0.9760	0.9519	**1.0000**	**1.0000**	0.9754
	this work	80|20 split	mean	PCA	BSMOTE-SVM	0.9710	0.9760	0.9661	0.9660	0.9710
	this work	80|20 split	k-NN	PCA	BSMOTE-SVM	**0.9812**	**1.0000**	0.9637	0.9626	0.9815
	this work	80|20 split	k-NN	PCA	ADASYN	**0.9812**	**1.0000**	0.9637	0.9626	0.9815
	this work	80|20 split	k-NN	SELECTFDR	ADASYN	0.9659	**0.9897**	0.9444	0.9422	0.9666
NN	[109]	80|20 split	-	SVM	-	0.9360	0.9180	0.9970	0.0810	0.957
GBT	[109]	80|20 split	-	MARS	-	0.9000	0.9000	0.8840	0.7080	0.8910
	this work	80|20 split	k-NN	PCA	SMOTE	**0.9795**	0.9829	0.9762	0.9762	0.9795
	this work	7-Fold	k-NN	PCA	SMOTE	0.9760	0.9615	0.9901	0.9904	0.9756
	this work	80|20 split	k-NN	PCA	BSMOTE-SVM	0.9659	0.9726	0.9595	0.9592	0.9660
	this work	7-Fold	k-NN	SELECTFDR	BSMOTE-SVM	0.9161	0.8750	0.9529	0.9569	0.9123
	this work	80|20 split	k-NN	PCA	ADASYN	0.9763	0.9896	0.9628	0.9637	0.9760
	this work	80|20 split	k-NN	SELECTFDR	ADASYN	0.9232	0.9452	0.9049	0.9014	0.9246
PSO-DBN	[96]	70|30 split	-	MEDSD	SMOTE	0.8659	0.9831	-	0.8485	-

SDGT: Synthetic data generation technique;

5.2 Summary of the chapter

The experimental results obtained in this work using 18 proposed methodologies are reported in this chapter. The proposed methodologies are: MI + PCA + 80|20 split, MI + PCA + CV, k-NNI + PCA + 80|20 split, k-NNI + UFS + 80|20 split, k-NNI + PCA + CV, and k-NNI + UFS +CV versus three SDGT (including SMOTE, BSMOTE-SVM, and ADASYN). Before the ML predictive models' development, the dataset has gone through stages of preprocessing, missing data points imputation, feature selection, feature normalization and data sampling technique of oversampling the minority class (fail class). Moreover, 80|20 holdout-split and k-Fold cross validation were applied to evaluate the performance of the developed models.

Firstly, the effect of each proposed data preprocessing step is analyzed. Section 5.1.1 reported the experimental results. Table 5 showed the detailed results obtained in that section. Secondly, section 5.1.2 presented the experimental results obtained using SMOTE on the six proposed methodologies. Thirdly, section 5.1.3 reported the general experimental results obtained using Borderline-SMOTE SVM on the six proposed methodologies. Fourthly, section 5.1.4 reported ADASYN experimental results using the proposed methodologies. Then, section 5.1.5 reported the experimental results general comparison analysis considering several perfectives. Lastly, section 5.1.6 gave the extensive comparative analysis between our obtained experimental results and similar studies reported experimental results (Table 8).

Overall, this work shows the feasibility of the two variants of synthetic minority oversampling technique (SMOTE) using 18 different methodologies on ten fault diagnosis machine learning models in the semiconductor manufacturing processes.

Moreover, the performance of these models developed using both SDGT techniques (BSMOTE-SVM and ADASYN) has been compared with the models developed using 6 different methodologies with the utilization of SMOTE. See Table 7 and Figure 22. From the extensive results obtained and the models' performance, similar trends have been observed within these three adopted minority oversampling techniques, however with some little differences from some of the proposed models and methodologies. The next chapter presents the conclusions and the future directions of this work.

Chapter 6

CONCLUSIONS and FUTURE WORKS

6.1 Conclusions

In this work, machine learning-based methodologies for fault diagnosis towards noisy, and imbalanced dataset within smart manufacturing systems for the semiconductor manufacturing process were proposed. These proposed methodologies consider the effects of missing values, redundant and noisy features, and class imbalance problem. The key contribution of this work relies on the implementation and comparison of two different missing datapoint imputation techniques including mean and k-NN imputation techniques; implementation and comparison of two different features selection techniques including PCA and univariate feature selection; and; implementation and comparison of three synthetic data generation techniques including SMOTE, BSMOTE-SVM, and ADASYN for synthetic data generation to handle the class imbalance distribution of the dataset. Ten prediction machine learning classifiers have been developed, namely; MLP (ANN), XGBoost, LR, DT, NB, LDA, RF, SVC, AdaBoost, and GBT. Their performance has been validated and compared on 18 different proposed methodologies using two different validation techniques involving 80|20-split and k-Fold cross-validation.

SECOM dataset was used as a case study to investigate the influence of these proposed methodologies and models. Experimental results across seven evaluation metrics of performance obtained from these models and methodologies were significant.

Moreover, the extensive experimental results obtained from this work were compared with recent similar studies reported from the literature to further validate the feasibility of these proposed models and methodologies. Based on the extensive results obtained and the analysis of the comparison, it has been proven that the methodologies and models proposed in this work outperformed the methodologies and models proposed from similar studies.

6.2 Future works

For future research directions, the following are suggested:

- Some of the developed machine learning models require hyperparameters tuning based on the results obtained from these models, as they produced very low performance. However, we could not do so due to some limitations regarding computational resources. To improve the performance of some of the models proposed like DT and NB, etc. Hyperparameter-tuning could be implemented using optimization-based techniques like random and grid search algorithms, genetic algorithm, particle swamp optimization, and simulated annealing.
- The influence of some other different techniques of datapoint imputation and feature selection techniques on both BSMMOTE-SVM and ADASYN could be investigated to further validate the robustness of these synthetic data generation techniques on the SECOM dataset, similarly, on any dataset from any domain with similar problem.
- These suggested methodologies could be extended further to analyze their feasibility on various redundant, noisy, and imbalanced datasets from any domain, particularly, smart manufacturing domains.

REFERENCES

[1] Z. M. Çinar, A. A. Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, and B. Safaei, "Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0," *Sustain.*, vol. 12, no. 19, 2020, doi: 10.3390/su12198211.

[2] C. M. Carbery, R. Woods, and A. H. Marshall, "A new data analytics framework emphasising preprocessing of data to generate insights into complex manufacturing systems," *Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci.*, vol. 233, no. 19–20, pp. 6713–6726, 2019, doi: 10.1177/0954406219866867.

[3] S. Munirathinam and B. Ramadoss, "Predictive Models for Equipment Fault Detection in the Semiconductor Manufacturing Process," *Int. J. Eng. Technol.*, vol. 8, no. 4, pp. 273–285, 2016, doi: 10.7763/ijet.2016.v8.898.

[4] K. Kerdprasop and N. Kerdprasop, "A data mining approach to automate fault detection model development in the semiconductor manufacturing process," *Int. J. Mech.*, vol. 5, no. 4, pp. 336–344, 2011.

[5] W. Zhang, D. Yang, and H. Wang, "Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2213–2227, 2019, doi: 10.1109/jsyst.2019.2905565.

[6] T. Wuest, D. Weimer, C. Irgens, and K. D. Thoben, "Machine learning in

manufacturing: Advantages, challenges, and applications," *Prod. Manuf. Res.*, vol. 4, no. 1, pp. 23–45, 2016, doi: 10.1080/21693277.2016.1192517.

[7] M. Ghahramani, Y. Qiao, M. C. Zhou, A. O. Hagan, and J. Sweeney, "AI-based modeling and data-driven evaluation for smart manufacturing processes," *IEEE/CAA J. Autom. Sin.*, vol. 7, no. 4, pp. 1026–1037, 2020, doi: 10.1109/JAS.2020.1003114.

[8] S. Kang, D. An, and J. Rim, "Incorporating virtual metrology into failure prediction," *IEEE Trans. Semicond. Manuf.*, vol. 32, no. 4, pp. 553–558, 2019, doi: 10.1109/TSM.2019.2932377.

[9] A. J. Su, J. C. Jeng, H. P. Huang, C. C. Yu, S. Y. Hung, and C. K. Chao, "Control relevant issues in semiconductor manufacturing: Overview with some new results," *Control Eng. Pract.*, vol. 15, no. 10 SPEC. ISS., pp. 1268–1279, 2007, doi: 10.1016/j.conengprac.2006.11.003.

[10] C. A. Mack, "FiftyYears of Moore ' s Law," *IEEE Fellow*, vol. 24, no. 2, p. 2008, 2011.

[11] N. Kumar, K. Kennedy, K. Gildersleeve, R. Abelson, C. M. Mastrangelo, and D. C. Montgomery, "A review of yield modelling techniques for semiconductor manufacturing," *Int. J. Prod. Res.*, vol. 44, no. 23, pp. 5019–5036, 2006, doi: 10.1080/00207540600596874.

[12] C. K. Shin and S. C. Park, “A machine learning approach to yield management in semiconductor manufacturing,” *Int. J. Prod. Res.*, vol. 38, no. 17, pp. 4261–4271, 2000, doi: 10.1080/00207540050205073.

[13] C. F. Chien, W. C. Wang, and J. C. Cheng, “Data mining for yield enhancement in semiconductor manufacturing and an empirical study,” *Expert Syst. Appl.*, vol. 33, no. 1, pp. 192–198, 2007, doi: 10.1016/j.eswa.2006.04.014.

[14] N. Amruthnath and T. Gupta, “A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance,” *2018 5th Int. Conf. Ind. Eng. Appl. ICIEA 2018*, no. August 1993, pp. 355–361, 2018, doi: 10.1109/IEA.2018.8387124.

[15] V. Mathew, T. Toby, V. Singh, B. M. Rao, and M. G. Kumar, “Prediction of Remaining Useful Lifetime (RUL) of turbofan engine using machine learning,” *IEEE Int. Conf. Circuits Syst. ICCS 2017*, vol. 2018-Janua, no. Iccs, pp. 306–311, 2018, doi: 10.1109/ICCS1.2017.8326010.

[16] M. C. Testik, “Expert Systems with Applications A review of data mining applications for quality improvement in manufacturing industry,” vol. 38, pp. 13448–13467, 2011, doi: 10.1016/j.eswa.2011.04.063.

[17] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto, and S. G. S. Alcalá, “A systematic literature review of machine learning methods applied to predictive maintenance,” *Comput. Ind. Eng.*, vol. 137, no.

September, p. 106024, 2019, doi: 10.1016/j.cie.2019.106024.

[18] M. E. Centre, “Machine-learning techniques and their applications in manufacturing,” vol. 219, pp. 395–412, 2005, doi: 10.1243/095440505X32274.

[19] L. Monostori, “AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing,” in *Engineering Applications of Artificial Intelligence*, 2003, vol. 16, no. 4, pp. 277–291, doi: 10.1016/S0952-1976(03)00078-2.

[20] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, “Machine learning for internet of things data analysis: a survey,” *Digit. Commun. Networks*, vol. 4, no. 3, pp. 161–175, 2018, doi: 10.1016/j.dcan.2017.10.002.

[21] B. Mohammadi et al., “Developing Novel Robust Models to Improve the Accuracy of Daily Streamflow Modeling,” *Water Resour. Manag.*, vol. 34, no. 10, pp. 3387–3409, 2020, doi: 10.1007/s11269-020-02619-z.

[22] B. Mohammadi, Y. Guan, R. Moazenzadeh, and M. J. S. Safari, “Implementation of hybrid particle swarm optimization-differential evolution algorithms coupled with multi-layer perceptron for suspended sediment load estimation,” *Catena*, no. October, p. 105024, 2020, doi: 10.1016/j.catena.2020.105024.

[23] B. Turkoglu and E. Kaya, “Training multi-layer perceptron with artificial algae algorithm,” *Eng. Sci. Technol. an Int. J.*, no. xxxx, 2020, doi: 10.1016/j.jestch.2020.07.001.

[24] S. A. Kalogirou, “Artificial neural networks in renewable energy systems applications: A review,” *Renewable and Sustainable Energy Reviews*, vol. 5, no. 4. pp. 373–401, 2000, doi: 10.1016/S1364-0321(01)00006-5.

[25] T. Sexton, M. P. Brundage, M. Hoffman, and K. C. Morris, “Hybrid datafication of maintenance logs from AI-assisted human tags,” *Proc. - 2017 IEEE Int. Conf. Big Data, Big Data 2017*, vol. 2018-Janua, pp. 1769–1777, 2017, doi: 10.1109/BigData.2017.8258120.

[26] C. C. Chang and C. J. Lin, “LIBSVM: A Library for support vector machines,” *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, 2011, doi: 10.1145/1961189.1961199.

[27] W. M. T. W. Ahmad, N. L. A. Ghani, and S. M. Drus, “Data mining techniques for disease risk prediction model: A systematic literature review,” *Adv. Intell. Syst. Comput.*, vol. 843, pp. 40–46, 2019, doi: 10.1007/978-3-319-99007-1_4.

[28] U. B. Parikh, B. Das, and R. Maheshwari, “Fault classification technique for series compensated transmission line using support vector machine,” *Int. J. Electr. Power Energy Syst.*, vol. 32, no. 6, pp. 629–636, 2010, doi: 10.1016/j.ijepes.2009.11.020.

[29] DataFlair Team, “Support Vector Machines Tutorial – Learn to implement SVM in Python,” Data Flair, 2019.

[30] L. Breiman, Random forests. 2001.

[31] S. Polamuri, “How Does the Random Forest Algorithm Work in Machine Learning,” 2017. .

[32] ODSC, “Logistic Regression with Python,” 2019. .

[33] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.

[34] J. Friedman, “Greedy Function Approximation : A Gradient Boosting Machine Author (s): Jerome H . Friedman Source : *The Annals of Statistics* , Vol . 29 , No . 5 (Oct ., 2001), pp . 1189-1232 Published by : *Institute of Mathematical Statistics Stable* URL : http://www,” Ann. Stat., vol. 29, no. 5, pp. 1189–1232, 2001.

[35] S. Asante-Okyere, C. Shen, Y. Y. Ziggah, M. M. Rulegeya, and X. Zhu, “A Novel Hybrid Technique of Integrating Gradient-Boosted Machine and Clustering Algorithms for Lithology Classification,” *Nat. Resour. Res.*, vol. 29, no. 4, pp. 2257–2273, 2020, doi: 10.1007/s11053-019-09576-4.

[36] Y. C. Chang, K. H. Chang, and G. J. Wu, “Application of eXtreme gradient

boosting trees in the construction of credit risk assessment models for financial institutions," *Appl. Soft Comput. J.*, vol. 73, pp. 914–920, 2018, doi: 10.1016/j.asoc.2018.09.029.

[37] V. Sugumaran, V. Muralidharan, and K. I. Ramachandran, "Feature selection using Decision Tree and classification through Proximal Support Vector Machine for fault diagnostics of roller bearing," *Mech. Syst. Signal Process.*, vol. 21, no. 2, pp. 930–942, 2007, doi: 10.1016/j.ymssp.2006.05.004.

[38] S. Rasoul and L. David, "A Survey of Decision Tree Classifier Methodology," *IEEE Trans. Syst. Man. Cybern.*, vol. 21, no. 3, pp. 660–674, 1991.

[39] R. Silipo, "From a Single Decision Tree to a Random Forest," *Towar. Data Sci.*, 2019.

[40] A. Lasisi and N. Attoh-Okine, "Principal components analysis and track quality index: A machine learning approach," *Transp. Res. Part C Emerg. Technol.*, vol. 91, no. July 2017, pp. 230–248, 2018, doi: 10.1016/j.trc.2018.04.001.

[41] A. M. Martinez and A. C. Kak, "PCA versus LDA," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 2, pp. 228–233, 2001, doi: 10.1109/34.908974.

[42] J. Too, A. R. Abdullah, and N. M. Saad, "Classification of Hand movements based on discrete wavelet transform and enhanced feature extraction," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 6, pp. 83–89, 2019, doi:

10.14569/ijacsa.2019.0100612.

[43] Y. Ren, H. Liu, C. Xue, X. Yao, M. Liu, and B. Fan, “Classification study of skin sensitizers based on support vector machine and linear discriminant analysis,” *Anal. Chim. Acta*, vol. 572, no. 2, pp. 272–282, 2006, doi: 10.1016/j.aca.2006.05.027.

[44] Y. Lu and Q. Tian, “Discriminant subspace analysis: An adaptive approach for image classification,” *IEEE Trans. Multimed.*, vol. 11, no. 7, pp. 1289–1300, 2009, doi: 10.1109/TMM.2009.2030632.

[45] G. Wang, T. Xu, H. Wang, and Y. Zou, “AdaBoost and Least Square Based Failure Prediction of Railway Turnouts,” *Proc. - 2016 9th Int. Symp. Comput. Intell. Des. Isc. 2016*, vol. 1, pp. 434–437, 2016, doi: 10.1109/ISCID.2016.1107.

[46] P. Tavallali, M. Yazdi, and M. R. Khosravi, “Robust cascaded skin detector based on AdaBoost,” *Multimed. Tools Appl.*, vol. 78, no. 2, pp. 2599–2620, 2019, doi: 10.1007/s11042-018-6385-7.

[47] F. Wang, D. Jiang, H. Wen, and H. Song, “Adaboost-based security level classification of mobile intelligent terminals,” *J. Supercomput.*, vol. 75, no. 11, pp. 7460–7478, 2019, doi: 10.1007/s11227-019-02954-y.

[48] R. E. Schapire, “Explaining adaboost,” Empir. Inference Festschrift Honor

Vladimir N. Vapnik, pp. 37–52, 2013, doi: 10.1007/978-3-642-41136-6_5.

[49] D. S. Sayad, “Naive Bayesian,” pp. 1–4, 2010, Accessed: Nov. 24, 2020. [Online]. Available: https://www.saedsayad.com/naive_bayesian.htm.

[50] M. H. Jopri, M. R. Ab Ghani, A. R. Abdullah, T. Sutikno, M. Manap, and J. Too, “Naïve bayes and linear discriminate analysis based diagnostic analytic of harmonic source identification,” *Indones. J. Electr. Eng. Comput. Sci.*, vol. 20, no. 3, pp. 1626–1633, 2020, doi: 10.11591/ijeecs.v20.i3.pp1626-1633.

[51] G. H. John and P. Langley, “Estimating Continuous Distributions in Bayesian Classifiers,” pp. 338–345, 1995.

[52] P. Zheng and A. S. Sivabalan, “A generic tri-model-based approach for product-level digital twin development in a smart manufacturing environment,” *Robot. Comput. Integr. Manuf.*, vol. 64, no. August 2019, p. 101958, 2020, doi: 10.1016/j.rcim.2020.101958.

[53] D. Wu, C. Jennings, J. Terpenny, R. Gao, and S. Kumara, “Data-driven prognostics using random forests: Prediction of tool wear,” *ASME 2017 12th Int. Manuf. Sci. Eng. Conf. MSEC 2017 collocated with JSME/ASME 2017 6th Int. Conf. Mater. Process.*, vol. 3, pp. 1–9, 2017, doi: 10.1115/MSEC2017-2679.

[54] M. Helu and T. Hedberg, “Enabling Smart Manufacturing Research and

Development using a Product Lifecycle Test Bed," *Procedia Manuf.*, vol. 1, no. Wolf 2009, pp. 86–97, 2015, doi: 10.1016/j.promfg.2015.09.066.

[55] W. Tao, Z. H. Lai, M. C. Leu, and Z. Yin, "Worker Activity Recognition in Smart Manufacturing Using IMU and sEMG Signals with Convolutional Neural Networks," *Procedia Manuf.*, vol. 26, pp. 1159–1166, 2018, doi: 10.1016/j.promfg.2018.07.152.

[56] J. Wang and D. Li, "Adaptive computing optimization in software-defined network-based industrial internet of things with fog computing," *Sensors* (Switzerland), vol. 18, no. 8, 2018, doi: 10.3390/s18082509.

[57] K. Nagorny, P. Lima-Monteiro, J. Barata, and A. W. Colombo, "Big Data Analysis in Smart Manufacturing: A Review," *Int. J. Commun. Netw. Syst. Sci.*, vol. 10, no. 03, pp. 31–58, 2017, doi: 10.4236/ijcns.2017.103003.

[58] M. Quirk and J. Serda, "Semiconductor Manufacturing Technology [slides]," 2001, [Online]. Available: http://jupiter.math.nctu.edu.tw/~weng/courses/IC_2007/PROJECT_MATH_CLASS3/device/integrated circuit technique/integrated circuit technique/%A5b%BE%C9%C5%E9%BBs%B5%7B2%A4%B64.pdf%5Cnpapers3://publication/uuid/DC7DF81C-A448-4E9D-9DEE-93953E3ADF8C.

[59] G. A. Susto, "Statistical Methods for Semiconductor Manufactureing," 2013.

[60] D. Stanisavljevic and M. Spitzer, “A review of related work on machine learning in semiconductor manufacturing and assembly lines,” *CEUR Workshop Proc.*, vol. 1793, 2017.

[61] “State of semiconductor industry - Blogs - Televisory.” https://www.televisory.com/blogs/-/blogs/state-of-semiconductor-industry (accessed Feb. 28, 2021).

[62] “File:12-inch silicon wafer.jpg - Wikimedia Commons.” https://commons.wikimedia.org/wiki/File:12-inch_silicon_wafer.jpg (accessed Feb. 28, 2021).

[63] T. Pfingsten, D. J. L. Herrmann, T. Schnitzler, A. Feustel, and B. Schölkopf, “Feature selection for troubleshooting in complex assembly lines,” *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 3, pp. 465–469, 2007, doi: 10.1109/TASE.2006.888054.

[64] M. Mccann, Y. Li, L. Maquire, and A. Johnston, “Causality Challenge: Benchmarking relevant signal components for effective monitoring and process control,” *J. Mach. Learn. Res. Work. Conf. Proc.*, vol. 6, no. February, pp. 277–288, 2010, [Online]. Available: http://www.causality.inf.ethz.ch/data/SECOM.zip%5Cnhttp://jmlr.csail.mit.edu/proceedings/papers/v6/.

[65] D. H. Lee, J. K. Yang, C. H. Lee, and K. J. Kim, “A data-driven approach to

selection of critical process steps in the semiconductor manufacturing process considering missing and imbalanced data," *J. Manuf. Syst.*, vol. 52, no. May, pp. 146–156, 2019, doi: 10.1016/j.jmsy.2019.07.001.

[66] J. C. Chien, M. T. Wu, and J. Der Lee, "Inspection and classification of semiconductor wafer surface defects using CNN deep learning networks," *Appl. Sci.*, vol. 10, no. 15, pp. 1–13, 2020, doi: 10.3390/APP10155340.

[67] "UCI Machine Learning Repository: SECOM Data Set." https://archive.ics.uci.edu/ml/datasets/SECOM (accessed Jan. 17, 2021).

[68] M. Mccann, Y. Li, L. Maquire, and A. Johnston, "Causality Challenge: Benchmarking relevant signal components for effective monitoring and process control," *J. Mach. Learn. Res. Work. Conf. Proc.*, vol. 6, pp. 277–288, 2010, [Online]. Available: http://www.causality.inf.ethz.ch/data/SECOM.zip%5Cnhttp://jmlr.csail.mit.edu/proceedings/papers/v6/.

[69] D. Kibira, K. C. Morris, and S. Kumaraguru, "Methods and tools for performance assurance of smart manufacturing systems," *J. Res. Natl. Inst. Stand. Technol.*, vol. 121, pp. 282–313, 2016, doi: 10.6028/jres.121.013.

[70] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *J. Manuf. Syst.*, vol. 48, pp. 144–156, 2018, doi: 10.1016/j.jmsy.2018.01.003.

[71] J. A. Harding, M. Shahbaz, Srinivas, and A. Kusiak, “Data mining in manufacturing: A review,” *J. Manuf. Sci. Eng. Trans. ASME*, vol. 128, no. 4, pp. 969–976, 2006, doi: 10.1115/1.2194554.

[72] B. T. Hazen, C. A. Boone, J. D. Ezell, and L. A. Jones-Farmer, “Data quality for data science, predictive analytics, and big data in supply chain management: An introduction to the problem and suggestions for research and applications,” *Int. J. Prod. Econ.*, vol. 154, pp. 72–80, 2014, doi: 10.1016/j.ijpe.2014.04.018.

[73] B. Esmaeilian, S. Behdad, and B. Wang, “The evolution and future of manufacturing: A review,” *J. Manuf. Syst.*, vol. 39, pp. 79–100, 2016, doi: 10.1016/j.jmsy.2016.03.001.

[74] S. J. Shin, J. Woo, and S. Rachuri, “Predictive analytics model for power consumption in manufacturing,” *Procedia CIRP*, vol. 15, pp. 153–158, 2014, doi: 10.1016/j.procir.2014.06.036.

[75] H. S. Kang et al., “Smart manufacturing: Past research, present findings, and future directions,” *Int. J. Precis. Eng. Manuf.* - Green Technol., vol. 3, no. 1, pp. 111–128, 2016, doi: 10.1007/s40684-016-0015-5.

[76] M. Sharp, R. Ak, and T. H. Jr, “A survey of the advancing use and development of machine learning in smart manufacturing,” *J. Manuf. Syst.*, vol. 48, pp. 170–179, 2018, doi: 10.1016/j.jmsy.2018.02.004.

[77] D. Wu, C. Jennings, J. Terpenny, R. X. Gao, and S. Kumara, “A Comparative Study on Machine Learning Algorithms for Smart Manufacturing: Tool Wear Prediction Using Random Forests,” *J. Manuf. Sci. Eng. Trans. ASME*, vol. 139, no. 7, pp. 1–9, 2017, doi: 10.1115/1.4036350.

[78] E. Mora, P. Gaiardelli, B. Resta, and D. Powell, “A Review of Current Machine Learning Techniques Used in Manufacturing Diagnosis,” *APMS 2017 Adv. Prod. Manag. Syst. Path to Intelligent, Collab. Sustain. Manuf.*, vol. 513, no. ii, pp. 127–134, 2017, doi: 10.1007/978-3-319-66923-6.

[79] D. Carrera, F. Manganini, G. Boracchi, and E. Lanzarone, “Defect detection in SEM images of nanofibrous materials,” *IEEE Trans. Ind. Informatics*, vol. 13, no. 2, pp. 551–561, 2017, doi: 10.1109/TII.2016.2641472.

[80] I. Shin et al., “A Framework for Prognostics and Health Management Applications toward Smart Manufacturing Systems,” *Int. J. Precis. Eng. Manuf. - Green Technol.*, vol. 5, no. 4, pp. 535–554, 2018, doi: 10.1007/s40684-018-0055-0.

[81] M. Moghaddam, A. Jones, and T. Wuest, “Design of Marketplaces for Smart Manufacturing Services,” *Procedia Manuf.*, vol. 39, no. 2019, pp. 194–201, 2019, doi: 10.1016/j.promfg.2020.01.312.

[82] M. Miškuf and I. Zolotová, “Comparison between multi-class classifiers and deep learning with focus on industry 4.0,” *2016 Cybern. Informatics*, K I 2016

- Proc. 28th Int. Conf., pp. 1–5, 2016, doi: 10.1109/CYBERI.2016.7438633.

[83] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, “Deep learning for smart manufacturing : Methods and applications,” *J. Manuf. Syst.*, vol. 48, pp. 144–156, 2018, doi: 10.1016/j.jmsy.2018.01.003.

[84] D. Fuqua and T. Razzaghi, “A cost-sensitive convolution neural network learning for control chart pattern recognition,” *Expert Syst.* Appl., vol. 150, 2020, doi: 10.1016/j.eswa.2020.113275.

[85] Y. G. Oh, M. Busogi, K. Ransikarbum, D. Shin, D. Kwon, and N. Kim, “Real-time quality monitoring and control system using an integrated cost effective support vector machine,” *J. Mech. Sci. Technol.*, vol. 33, no. 12, pp. 6009–6020, 2019, doi: 10.1007/s12206-019-1145-9.

[86] L. Oneto, I. Orlandi, and D. Anguita, “Performance assessment and uncertainty quantification of predictive models for smart manufacturing systems,” *Proc. - 2015 IEEE Int. Conf. Big Data, IEEE Big Data 2015*, pp. 1436–1445, 2015, doi: 10.1109/BigData.2015.7363904.

[87] M. Salem, S. Taheri, and J.-S. Yuan, “An Experimental Evaluation of Fault Diagnosis from Imbalanced and Incomplete Data for Smart Semiconductor Manufacturing,” *Big Data Cogn. Comput.*, vol. 2, no. 4, p. 30, 2018, doi: 10.3390/bdcc2040030.

[88] A. Chazhoor, Y. Mounika, M. Vergin Raja Sarobin, M. V Sanjana, and R. Yasashvini, “Predictive Maintenance using Machine Learning Based Classification Models,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 954, p. 012001, 2020, doi: 10.1088/1757-899x/954/1/012001.

[89] I. Anghel, T. Cioara, D. Moldovan, I. Salomie, and M. M. Tomus, “Prediction of Manufacturing Processes Errors: Gradient Boosted Trees Versus Deep Neural Networks,” *Proc. - 16th Int. Conf. Embed. Ubiquitous Comput.* EUC 2018, pp. 29–36, 2018, doi: 10.1109/EUC.2018.00012.

[90] H. A. Kao, Y. S. Hsieh, C. H. Chen, and J. Lee, “Quality prediction modeling for multistage manufacturing based on classification and association rule mining,” *MATEC Web Conf.*, vol. 123, 2017, doi: 10.1051/matecconf/201712300029.

[91] D. Moldovan, T. Cioara, I. Anghel, and I. Salomie, “Machine learning for sensor-based manufacturing processes,” *Proc. - 2017 IEEE 13th Int. Conf. Intell. Comput. Commun. Process. ICCP 2017*, pp. 147–154, 2017, doi: 10.1109/ICCP.2017.8116997.

[92] J. Kim, Y. Han, and J. Lee, “Data Imbalance Problem solving for SMOTE Based Oversampling: Study on Fault Detection Prediction Model in Semiconductor Manufacturing Process,” vol. 133, pp. 79–84, 2016, doi: 10.14257/astl.2016.133.15.

[93] Y. C. Ko and H. Fujita, “An evidential analytics for buried information in big data samples: Case study of semiconductor manufacturing,” *Inf. Sci. (Ny).*, vol. 486, pp. 190–203, 2019, doi: 10.1016/j.ins.2019.01.079.

[94] Y. Takahashi, M. Asahara, and K. Shudo, “A Framework for Model Search Across Multiple Machine Learning Implementations,” arXiv, 2019.

[95] D. Moldovan, V. Chifu, C. Pop, T. Cioara, I. Anghel, and I. Salomie, “Chicken Swarm Optimization and Deep Learning for Manufacturing Processes,” *Proc. - 17th RoEduNet IEEE Int. Conf. Netw. Educ. Res.* RoEduNet 2018, pp. 18–23, 2018, doi: 10.1109/ROEDUNET.2018.8514152.

[96] J. K. Kim, Y. S. Han, and J. S. Lee, “Particle swarm optimization–deep belief network–based rare class prediction model for highly class imbalance problem,” *Concurr. Comput.* , vol. 29, no. 11, 2017, doi: 10.1002/cpe.4128.

[97] G. E. A. P. A. Batista and M. C. Monard, “A study of k-nearest neighbour as an imputation method,” *Front. Artif. Intell. Appl.*, vol. 87, no. June, pp. 251–260, 2002.

[98] O. Troyanskaya et al., “Missing value estimation methods for DNA microarrays,” *Bioinformatics*, vol. 17, no. 6, pp. 520–525, 2001, doi: 10.1093/bioinformatics/17.6.520.

[99] H. De Silva and A. S. Perera, “Missing data imputation using Evolutionary k-

Nearest neighbor algorithm for gene expression data," *16th Int. Conf. Adv. ICT Emerg. Reg. ICTer 2016 - Conf. Proc.*, pp. 141–146, 2017, doi: 10.1109/ICTER.2016.7829911.

[100] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdiscip. Rev. Comput. Stat.*, vol. 2, no. 4, pp. 433–459, 2010, doi: 10.1002/wics.101.

[101] W. P. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002, [Online]. Available: https://arxiv.org/pdf/1106.1813.pdf%0Ahttp://www.snopes.com/horrors/insects/telamonia.asp.

[102] "5 SMOTE Techniques for Oversampling your Imbalance Data | by Cornellius Yudha Wijaya | Towards Data Science." https://towardsdatascience.com/5-smote-techniques-for-oversampling-your-imbalance-data-b8155bdbe2b5 (accessed Dec. 27, 2020).

[103] Q. Wang, Z. Luo, J. Huang, Y. Feng, and Z. Liu, "A Novel Ensemble Method for Imbalanced Data Learning," *Comput. Intell. Neurosci.*, vol. 2017, pp. 1–11, 2017.

[104] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," *Proc. Int. Jt. Conf. Neural Networks*, no. 3, pp. 1322–1328, 2008, doi: 10.1109/IJCNN.2008.4633969.

[105] S. Arlot and A. Celisse, “A survey of cross-validation procedures for model selection,” *Stat. Surv.*, vol. 4, pp. 40–79, 2010, doi: 10.1214/09-SS054.

[106] C. Hu, B. D. Youn, P. Wang, and J. Taek Yoon, “Ensemble of data-driven prognostic algorithms for robust prediction of remaining useful life,” *Reliab. Eng. Syst. Saf.*, vol. 103, pp. 120–135, 2012, doi: 10.1016/j.ress.2012.03.008.

[107] A. Saxena, K. Goebel, D. Simon, and N. Eklund, “Damage propagation modeling for aircraft engine run-to-failure simulation,” *2008 Int. Conf. Progn. Heal. Manag. PHM 2008*, 2008, doi: 10.1109/PHM.2008.4711414.

[108] Y. Zhou, T. A. Mazzuchi, and S. Sarkani, “M-AdaBoost-A based ensemble system for network intrusion detection,” *Expert Syst. Appl.*, vol. 162, no. August, p. 113864, 2020, doi: 10.1016/j.eswa.2020.113864.

[109] I. Anghel, T. Cioara, D. Moldovan, I. Salomie, and M. M. Tomus, “Prediction of Manufacturing Processes Errors: Gradient Boosted Trees Versus Deep Neural Networks,” *Proc. - 16th Int. Conf. Embed. Ubiquitous Comput. EUC 2018*, no. February 2020, pp. 29–36, 2018, doi: 10.1109/EUC.2018.00012.

APPENDICES

Appendix A: Tabular supplementary experimental results

Table 9: Accuracy-based results comparison amongst MRD, MRD with main data preprocessing step effects, and PM using SMOTE.

Methods:	**Accuracy:**						
	MRD + 80|20 split	***MRD + MI + 80|20 split***	***MRD + MI + FS+ PCA+ 80|20 split***	***MRD + MI + SMOTE + 80|20 split***	***PM: - MI + PCA + SMOTE + 80|20 split***	***PM: - k-NNI + PCA + SMOTE + 80|20 split***	***PM: - k-NNI + UFS + SMOTE + 80|20 split***
MLP	NAN	0.7293	0.9268	0.9215	0.9829	0.9812	0.9710
XGBoost	0.9363	0.9363	0.9299	0.9676	0.9744	0.9625	0.9164
LR	NAN	0.9299	0.9363	0.7235	0.8976	0.9164	0.7901
DT	NAN	0.8885	0.8503	0.9078	0.8532	0.9061	0.8976
NB	NAN	0.2070	0.9013	0.5734	0.7696	0.8106	0.5631
LDA	NAN	0.9108	0.9236	0.9147	0.8771	0.8942	0.7901
RF	NAN	0.9363	0.9363	0.9846	0.9983	0.9949	0.9744
SVC	NAN	0.9363	0.9363	0.6775	0.9966	0.9983	0.8686
AdaBoost	NAN	0.9236	0.9236	0.9266	0.8720	0.8925	0.8669
GBT	NAN	0.9331	0.9331	0.9642	0.9761	0.9795	0.9232

MRD: Modified raw dataset; MI: Mean imputation; k-NNI: k-NN imputation; PCA: Principal component analysis; 80|20 split: 80|20 split validation technique; PM: Proposed methodology;

Table 10: Confusion matrix results obtained before and after each step.

	MRD				**MRD + MI**				**MRD + k-NNI**				**MRD + MI + PCA**				**MRD + MI + FS + PCA**				**MRD + MI + SMOTE**			
	TP	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**
MLP	-	-	-	-	224	5	70	15	287	0	7	20	291	0	3	20	292	1	2	19	263	277	31	15
XGBoost	294	0	0	20	294	0	0	20	294	0	0	20	292	0	2	20	294	0	0	20	287	280	7	12
LR	-	-	-	-	292	0	2	20	292	0	2	20	294	0	0	20	279	5	15	15	210	214	84	78
DT	-	-	-	-	276	3	18	17	276	5	18	15	267	0	27	20	275	3	19	17	252	280	42	12
NB	-	-	-	-	51	14	243	6	42	15	252	5	282	1	12	19	278	1	16	19	55	281	239	11
LDA	-	-	-	-	277	9	17	11	281	8	13	12	290	4	0	20	286	4	8	16	244	292	50	0
RF	-	-	-	-	294	0	0	20	294	0	0	20	294	0	0	20	294	0	0	20	293	284	1	8
SVC	-	-	-	-	294	0	0	20	294	0	0	20	294	0	0	20	294	0	0	20	164	233	130	59
AdaBoost	-	-	-	-	289	1	5	19	288	4	6	16	290	0	4	20	287	0	7	20	270	273	24	19
GBT	-	-	-	-	293	0	1	20	290	2	4	18	293	0	1	20	291	0	3	20	285	280	9	12

MRD: Modified raw dataset; MI: Mean imputation; k-NNI: k-NN imputation; PCA: Principal component analysis; TP: True positive; TN: True negative; FP: False positive; FN: False negative;

Table 11: Confusion matrix of the overall experimental results obtained.

		MI + PCA + 80|20-split				MI + PCA + CV				k-NNI + PCA + 80|20-split				k-NNI + UFS + 80|20-split				k-NNI + PCA + CV				k-NNI + UFS + CV			
SDGT	Model	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**	**TP**	**TN**	**FP**	**FN**
SMOTE	MLP	284	292	10	0	200	209	0	8	283	292	11	0	277	292	17	0	198	209	0	10	193	208	0	15
	XGBoost	285	286	9	6	106	111	1	6	277	287	17	5	260	277	34	15	107	110	1	4	97	107	4	15
	LR	242	284	52	8	410	477	10	77	251	286	43	6	233	230	61	62	410	478	9	77	372	375	112	115
	DT	234	266	60	26	125	133	12	21	252	279	42	13	256	270	38	22	122	134	12	24	123	134	12	23
	NB	218	233	76	59	156	175	34	52	237	238	57	54	277	53	17	239	160	167	41	48	197	30	178	12
	LDA	233	281	61	11	168	199	9	40	238	286	56	6	229	234	65	58	167	201	7	41	158	157	51	50
	RF	294	291	0	1	208	207	1	0	294	289	0	3	283	288	11	4	208	207	1	0	199	203	5	9
	SVC	292	292	2	0	483	486	1	4	293	292	1	0	242	267	52	25	484	486	1	3	392	421	66	95
	AdaBoost	256	255	38	37	178	189	19	30	258	265	36	27	247	261	47	31	181	186	23	27	171	180	28	37
	GBT	282	290	12	2	199	206	2	9	287	287	7	5	265	276	23	16	200	206	2	8	182	200	9	26
BSMOTE-SVM	MLP	284	285	10	7	202	203	5	6	283	292	11	0	277	290	17	2	200	209	0	8	193	203	0	15
	XGBoost	292	279	2	13	111	107	5	1	285	286	9	6	262	278	32	14	107	110	1	4	97	107	4	15
	LR	257	281	37	11	424	468	19	63	248	288	46	4	231	213	63	79	177	204	4	31	160	159	49	48
	DT	256	264	38	28	128	133	11	17	240	268	54	24	259	268	35	24	123	132	13	23	123	134	12	23
	NB	294	182	0	110	204	158	50	4	234	221	60	71	277	40	17	252	160	167	41	48	197	30	178	12
	LDA	261	271	33	21	185	197	11	24	281	292	13	0	276	288	18	4	167	201	7	41	158	157	51	50
	RF	294	275	0	17	209	199	9	0	294	286	0	6	285	285	9	7	208	207	1	0	198	203	5	10
	SVC	294	283	0	9	485	470	17	2	293	292	1	0	239	242	455	50	208	208	0	0	169	184	25	39
	AdaBoost	259	268	35	24	189	192	16	19	256	258	38	34	249	236	45	56	181	186	23	27	171	180	28	37
	GBT	293	277	1	15	205	201	7	3	282	284	12	8	264	276	30	16	200	205	3	8	182	200	9	26
ADASYN	MLP	285	299	11	0	199	215	0	9	283	292	11	0	277	289	17	3	198	212	0	10	192	210	0	16
	XGBoost	284	298	12	1	106	115	1	6	290	286	13	2	262	278	32	14	106	113	0	5	96	107	6	16
	LR	260	296	36	3	408	492	10	79	261	281	42	7	231	213	63	79	410	487	8	77	369	378	112	118
	DT	250	276	46	23	122	139	11	23	246	265	57	23	258	267	36	25	122	136	12	24	124	134	13	21
	NB	223	251	73	48	152	179	36	56	237	230	66	58	277	40	252	17	158	177	35	50	197	30	180	11
	LDA	241	291	55	8	165	209	5	43	247	284	56	4	228	214	66	78	167	206	6	42	152	162	47	56
	RF	295	298	1	1	208	214	1	0	303	284	0	4	284	284	10	8	208	211	1	0	197	205	5	11
	SVC	294	299	2	0	483	502	0	4	302	288	1	0	239	242	55	50	484	495	0	3	377	435	55	110
	AdaBoost	259	265	37	34	178	193	22	30	262	249	41	39	249	236	45	56	180	192	19	29	170	184	25	38
	GBT	283	297	13	2	198	214	1	10	292	285	11	3	265	276	29	16	200	210	1	9	178	199	11	30

SDGT: Synthetic data generation technique; MI: Mean imputation; k-NNI: k-NN imputation; PCA: Principal component analysis; UFS: Univariate features selection; 80|20 split: 80|20 split validation technique; CV: k-Fold cross validation technique; TP: True positive; TN: True negative; FP: False positive; FN: False negative;

Appendix B: Codes

Libraries used

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_
score, recall_score
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
import pandas as pd
from tensorflow import keras
import numpy as np # linear algebra
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score, cross_val_predict
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score,
precision_score, recall_score, classification_report
from keras.utils import to_categorical
```

Secom Data Exploration

```
Data = pd.read_csv('/content/uci-secom_v2.csv')
# displaying the dataset
Data.head()
#checking for any missing datapoints
Data.isnull().any().any()
#reclassifying the classes with 0 and 1, pass and fail class respectivel
y
Data = Data.drop(['Time'], axis=1)
Data.loc[(Data['Pass/Fail'] == -1),'Pass/Fail'] = 0
Data.head()
#Dropping the output column
features = Data.drop(['Pass/Fail'],axis=1)
features_labels = Data['Pass/Fail']
features.shape
# Get the counts for each class
alabel_count = Data['Pass/Fail'].value_counts()
print(alabel_count)
# Plot the results
plt.figure(figsize=(10,8))
sns.barplot(x=alabel_count.index, y= alabel_count.values)
plt.title('Number of labels', fontsize=14)
plt.xlabel('label type', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xticks(range(len(alabel_count.index)), ['Fail', 'Pass'])
plt.show()
# Missing datapoints vs Observed
import seaborn as sns
```

```
import matplotlib.pyplot as plt
sns.heatmap(Data.isnull(),yticklabels=False,cbar=False,cmap='viridis')

plt.title('Observed vs Missing datapoints', fontsize=12)
plt.xlabel('Features', fontsize=12)
plt.ylabel('Instances', fontsize=12)
#plt.xticks(range(len(alabel_count.index)), ['Features'])
plt.show()
```

SECOM Data Preprocessing

```
# finding missing and categorical values
col_missing_values = [col for col in features.columns
                      if features[col].isnull().any() and features[col].d
type]
# col wit numerical values which is null in this project
col_numerical_values = [col for col in features.columns if features[col]
.dtype in ['int64', 'float64']]
# col with categorical values
col_categorical_values = [col for col in features.columns
                          if features[col].dtype == 'object']
#missing values cout per col
missing_val_count_by_column = (features.isnull().sum())
print(missing_val_count_by_column[missing_val_count_by_column > 0])
print(col_categorical_values)
print(col_numerical_values)
```

Imputation

```
#Data imputation using KNN
from sklearn.impute import KNNImputer
imputer = KNNImputer()
imputed_features = pd.DataFrame(imputer.fit_transform(features))
# Fill in the lines below: imputation removed column names; put them bac
k
imputed_features.columns = features.columns

features = imputed_features
features.head()
#Checking for any missing cell value
features.isnull().any().any()
#Importing features to csv file
features.to_csv('features.csv')
features_labels.to_csv('features_labels.csv')
```

Feature scaling and selection

PCA

```
# Feature scaling using mean
Scaler =StandardScaler()
features =Scaler.fit_transform(features)
# Feature selection using Principal Component Analysis (PCA)
pca = PCA(n_components=0.99, whiten=True)
# Conduct PCA
features = pca.fit_transform(features)
```

```
# Show results
print("Original number of features:", Data.shape[1])
print("Reduced number of features:", features.shape[1])
# Dataset dimension visualization
features.shape
features
```

UFS

```
# Feature selection using UFS
from sklearn.feature_selection import SelectFdr
from sklearn.feature_selection import f_classif , chi2
# Create a UFS
UFS = SelectFdr(score_func=f_classif)
features = UFS.fit_transform(features, features_labels)
# Show results
print("Original number of features:", Data.shape[1])
print("Reduced number of features:", features.shape[1])
# Feature scaling using mean
Scaler = StandardScaler()
features = Scaler.fit_transform(features)
```

Model evaluation

```
def get_Evaluation_matrics(preds, orig_test_labels):
  # Get the confusion matrix
  cm  = confusion_matrix(orig_test_labels, preds)
  plt.figure()
  plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True,cmap=plt.cm.B
lues)
  plt.xticks(range(2), ['Fail', 'Pass'], fontsize=16)
  plt.yticks(range(2), ['Fail', 'Pass'], fontsize=16)
  plt.show()

  # Calculate the metrics
  tn, fp, fn, tp = cm.ravel()
  Accuracy = (tp+tn)/(tp+tn+fp+fn)
  precision = tp/(tp+fp)
  recall = tp/(tp+fn)
  specificity = tn/(tn+fp)
  F1_score = 2*tp/(2*tp+fp+fn)

  print("Accuracy of the model is {:.4f}".format(Accuracy))
  print("Recall of the model is {:.4f}".format(recall))
  print("Precision of the model is {:.4f}".format(precision))
  print("specificity of the model is {:.4f}".format(specificity))
  print("F1_score of the model is {:.4f}".format(F1_score))
```

Model prediction

```
def get_Model_prediction(model, test_data, test_labels):
  import numpy as np
  # # Evaluation on test dataset
  # test_loss, test_score = model.evaluate(test_data, test_labels, batch
_size=32)
  # print("Loss on test set: ", test_loss)
```

```
  # print("Accuracy on test set: ", test_score)

  preds = model.predict(test_data, batch_size=16)
  preds = np.argmax(preds, axis=-1)
  # orig_test_labels = np.argmax(test_labels, axis=-1)
  print(test_labels)
  print(preds)
  return preds, test_labels
```

Plotting model accuracy

```
def plot_Model_Accuracy(History, epoch):
  history_dict2 = History.history
  acc_values2 = history_dict2['accuracy']
  val_acc_values2 = history_dict2['val_accuracy']
  epochs = range(1, epoch + 1)
  plt.plot(epochs, acc_values2, 'b-', label='training Accuracy')
  plt.plot(epochs, val_acc_values2, 'r-', label='validation Accuracy')
  plt.title('trainin/validation Accuracy')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.rcParams['axes.facecolor'] = 'white'
  plt.rcParams['axes.edgecolor'] = 'white'
  plt.rcParams['axes.grid'] = True
  plt.rcParams['grid.alpha'] = 1
  plt.rcParams['grid.color'] = "#cccccc"
  plt.show()
```

Model building using SMOTE data generation technique
80|20 - split validation technique
ANN

```
early_stopping_cb = keras.callbacks.EarlyStopping(patience=5,
                                                  restore_best_weights=T
rue)


from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)

X = pd.DataFrame(X)
y = pd.DataFrame(y)

train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
train_data, valid_data, train_labels, valid_labels = train_test_split(tr
ain_data, train_labels, train_size = 0.85, random_state = 123)

train_data = np.array(train_data)
valid_data = np.array(valid_data)
test_data = np.array(test_data)
train_labels = np.array(train_labels)
valid_labels = np.array(valid_labels)
```

```
test_labels = np.array(test_labels)

y.iloc[:,0].value_counts()


from tensorflow.keras import models
from tensorflow.keras import layers
from imblearn.over_sampling import SMOTE
model = models.Sequential()
model.add(layers.Dense(1024, input_shape = (219,), activation='relu'))
# model.add(layers.Dense(128, activation='relu'))
# model.add(layers.Dense(512, activation='relu'))
# model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
# model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model_history = model.fit(train_data,
                          train_labels,
                          epochs=30,
                          batch_size=16,
                          validation_data=(valid_data, valid_labels),
                          callbacks=None
                          )
# Evaluating my model using CM, precsion, and recal
preds, original_test_labels = get_Model_prediction(model, test_data, tes
t_labels)
get_Evaluation_matrics(preds, original_test_labels)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier as kNN
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = kNN(leaf_size=10)
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)

# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

MLP

```
from sklearn.neural_network import MLPClassifier as MLP
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
```

```
X = pd.DataFrame(X)
y = pd.DataFrame(y)

train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = MLP()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

XGBOOST

```
from xgboost.sklearn import XGBClassifier
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = XGBClassifier()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

LR

```
from sklearn.linear_model import LogisticRegression as LR
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = LR(max_iter=100)
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

DT

```
from sklearn.tree import DecisionTreeClassifier as DT
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
```

```
model = DT()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

NB

```
from sklearn.naive_bayes import GaussianNB as NB
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = NB()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = LDA()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

RF

```
from sklearn.ensemble import RandomForestClassifier as RF
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = RF()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
```

```
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

SVC

```
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = SVC()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)

# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

ADABOOST

```
from sklearn.ensemble import AdaBoostClassifier as ADB
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = ADB()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

GBT

```
from sklearn.ensemble import GradientBoostingClassifier as GBT
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_sample(features, features_labels)
X = pd.DataFrame(X)
y = pd.DataFrame(y)
train_data, test_data, train_labels, test_labels = train_test_split(X, y
, test_size = 0.20, random_state = 123)
model = GBT()
model.fit(train_data, train_labels)
preds = model.predict(test_data)
get_Evaluation_matrics(preds, test_labels)
# plotting the AUC for all my classes
get_model_ROC_AUC_curve(preds, test_labels)
```

k-FOLD cross validation (CV)

```
from sklearn.model_selection import KFold, cross_val_score, cross_val_pr
edict, cross_validate
from sklearn.pipeline import make_pipeline
features, features_labels = oversample.fit_resample(features, features_l
abels)
# Create standardizer
standardizer = StandardScaler()
# Create logistic regression object
from sklearn.svm import SVC
logit = SVC()
# Create a pipeline that standardizes, then runs logistic regression
pipeline = make_pipeline(standardizer, logit)
# Create k-Fold cross-validation
kf = KFold(n_splits=7, shuffle=True, random_state=1)
# Conduct k-fold cross-validation
cv_results = cross_val_score(logit, # Pipeline
                             features, # Feature matrix
                             features_labels, # Target vector
                             cv=kf, # Cross-validation technique
                             scoring="precision", # Loss function
                             n_jobs=-1) # Use all CPU scores
# Calculate mean
cv_results.mean()
```

k-Fold CV metrics of performance

```
tp = 0
tn = 162
fp = 0
fn = 162

Accuracy = (tp+tn)/(tp+tn+fp+fn)
#precision = tp/(tp+fp)
recall = tp/(tp+fn)
specificity = tn/(tn+fp)
F1_score = 2*tp/(2*tp+fp+fn)

print("Accuracy of the model is {:.4f}".format(Accuracy))
print("Recall of the model is {:.4f}".format(recall))
#print("Precision of the model is {:.4f}".format(precision))
print("specificity of the model is {:.4f}".format(specificity))
print("F1_score of the model is {:.4f}".format(F1_score))
print("")
```

MLP

```
from sklearn.neural_network import MLPClassifier as MLP
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = MLP()
conf_matrix_list_of_arrays = []
```

```
kf = KFold(7, shuffle=True, random_state=123)

for train_index, test_index  in kf.split(train_data):

   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

XGBOOST

```
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()

train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = XGBClassifier()
conf_matrix_list_of_arrays = []
kf = KFold(13, shuffle=True, random_state=123)
for train_index, test_index  in kf.split(train_data):
   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

LR

```
from sklearn.linear_model import LogisticRegression as LR
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = LR(max_iter=200)
conf_matrix_list_of_arrays = []
kf = KFold(3, shuffle=True, random_state=123)

for train_index, test_index  in kf.split(train_data):
   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal(
mean_of_conf_matrix_arrays.T
```

DT

```
from sklearn.tree import DecisionTreeClassifier as DT
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = DT()
conf_matrix_list_of_arrays = []
kf = KFold(10, shuffle=True, random_state=123)
for train_index, test_index  in kf.split(train_data):
   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
```

```
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

NB

```
from sklearn.naive_bayes import GaussianNB as NB
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = NB()
conf_matrix_list_of_arrays = []
kf = KFold(7, shuffle=True, random_state=123)
for train_index, test_index  in kf.split(train_data):

   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)


pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as
LDA
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = LDA()
```

```
conf_matrix_list_of_arrays = []
kf = KFold(7, shuffle=True, random_state=123)
for train_index, test_index  in kf.split(train_data):
   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

RF

```
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = RF()
conf_matrix_list_of_arrays = []
kf = KFold(7, shuffle=True, random_state=123)
for train_index, test_index  in kf.split(train_data):

   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

SVC

```
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = SVC()
conf_matrix_list_of_arrays = []
kf = KFold(3, shuffle=True, random_state=123)

for train_index, test_index  in kf.split(train_data):

   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

ADBOOST

```
from sklearn.ensemble import AdaBoostClassifier as ADB
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)

#Create algorithm object
model = ADB()
conf_matrix_list_of_arrays = []
kf = KFold(7, shuffle=True, random_state=123)
for train_index, test_index  in kf.split(train_data):
   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
```

```
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```

GBT

```
from sklearn.ensemble import GradientBoostingClassifier as GBT
from sklearn.model_selection import KFold
from museotoolbox.charts import PlotConfusionMatrix

from imblearn.over_sampling import SMOTE
oversample = SMOTE()
train_data, train_labels = oversample.fit_resample(features, features_la
bels)
#Create algorithm object
model = GBT()
conf_matrix_list_of_arrays = []
kf = KFold(7, shuffle=True, random_state=123)

for train_index, test_index  in kf.split(train_data):

   train_data1, test_data1 = train_data[train_index], train_data[test_in
dex]
   train_labels1, test_labels1 = train_labels[train_index], train_labels
[test_index]

   model.fit(train_data1, train_labels1)
   conf_matrix = confusion_matrix(test_labels1, model.predict(test_data1
))
   conf_matrix_list_of_arrays .append(conf_matrix)
mean_of_conf_matrix_arrays = np.mean(conf_matrix_list_of_arrays, axis=0)
.astype(np.int16)
pltCM = PlotConfusionMatrix(mean_of_conf_matrix_arrays.T) # Translate fo
r Y = prediction and X = truth
pltCM.add_text()
# pltCM.add_f1()
pltCM.color_diagonal()
mean_of_conf_matrix_arrays.T
```