POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Addressing Data Scarcity for Machine-Learning-based Failure Management in Microwave Networks

Tesi di Laurea Magistrale in
Telecommunication Engineering - Ingegneria delle Telecomunicazioni

Author: **Matteo Conci**

Student ID: 964105
Advisor: Prof. Francesco Musumeci
Co-advisors: Nicola Di Cicco, Mëmëdhe Ibrahimi
Academic Year: 2022-23

Abstract

Failure management in communication networks is a critical issue nowadays, as a single failure in the network can lead to service disruption for thousands or millions of users at the same time. Therefore, preventing failures from occurring is a crucial task for network operators to meet Service Level Agreements (SLAs) to its customers. For this purpose, collecting and analyzing data generated from the continuous monitoring of network parameters and alarms have become crucial to construct historical knowledge of network failures and drive future decisions on how to handle them. Nowadays, this analysis is carried out by domain experts who, based on their experience, identify the failures and engage proper countermeasures to mitigate them or to restore the service. The time required by humans to perform the analysis and engage the countermeasures is often not in line with the stringent time constraint to restore the service after a failure imposed by the SLAs. To overcome this problem, substantial help comes from Artificial Intelligence (AI) and Machine Learning (ML), through which it is possible to automate and speed up the whole network management process by leveraging all the data retrieved monitoring the network. In our work, we consider failure management in microwave networks, focusing on the failure-cause identification problem. Specifically, we use supervised machine learning models to address the classification of hardware failures in microwave networks. These models require large amount of labelled data to be trained, but gathering data from the field in an expensive and time-consuming process, so it is necessary to devise approaches that address the problem of having insufficient amount of data. This condition is known as "data scarcity" and the main contribution of this work is to identify and compare different ML methodologies to address this problem, such as the generation of synthetic data using Synthetic Minority Over-sampling TEchnique (SMOTE), the use of Transfer Learning, the use of Auxiliary-Task Learning, and the use of Denoising Autoencoders. Our numerical results show that, focusing the synthetic data generation on specific failure classes and deciding proper amounts of data to generate, SMOTE outperforms all other methodologies.

Keywords: microwave networks, machine learning, failure management, hardware failures, data scarcity, SMOTE, transfer learning, auxiliary-task learning, denoising autoencoders

Abstract in lingua italiana

La gestione dei guasti nelle reti di comunicazione è una tematica critica al giorno d'oggi, poiché un singolo guasto nella rete può portare all'interruzione del servizio per migliaia o milioni di utenti nello stesso momento. Pertanto, la prevenzione dei guasti è un compito cruciale per gli operatori di rete, al fine di soddisfare i Service Level Agreement (SLA) con i propri clienti. A tal fine, la raccolta e l'analisi dei dati generati dal monitoraggio continuo dei parametri di rete e degli allarmi sono diventati fondamentali per costruire una conoscenza storica dei guasti di rete e guidare le decisioni future su come gestirli. Al giorno d'oggi, questa analisi viene effettuata da esperti del settore, che, sulla base della loro esperienza, identificano i guasti e adottano le contromisure adeguate a mitigarli o per ripristinare il servizio. Il tempo richiesto dall'uomo per eseguire l'analisi e adottare le contromisure spesso non è in linea con i tempi stringenti di ripristino del servizio dopo un guasto imposti dagli SLA. Per mitigare questo problema, un aiuto sostanziale viene dall'Intelligenza Artificiale (AI) e dal Machine Learning (ML), grazie ai quali è possibile automatizzare e velocizzare l'intero processo di gestione della rete sfruttando tutti i dati recuperati monitorando la rete. Nel nostro lavoro prendiamo in considerazione la gestione dei guasti nelle reti a microonde, concentrandoci sul problema dell'identificazione delle cause dei guasti. In particolare, utilizziamo modelli di apprendimento automatico supervisionato per affrontare la classificazione dei guasti hardware nelle reti a microonde. Questi modelli richiedono una grande quantità di dati etichettati per essere addestrati, ma la raccolta di dati dal campo è un processo costoso e che richiede tempo, quindi è necessario ideare approcci che affrontino il problema della quantità insufficiente di dati. Questa condizione è nota come "data scarcity" (scarsità di dati) e il contributo principale di questo lavoro consiste nell'identificare e confrontare diverse metodologie di ML per affrontare questo problema, come la generazione di dati sintetici utilizzando SMOTE (Synthetic Minority Over-sampling TEchnique), l'uso di Transfer Learning, l'uso di Auxiliary-Task Learning e l'uso di Denoising Autoencoders. I nostri risultati numerici dimostrano che, concentrando la generazione di dati sintetici su specifiche classi di guasti e decidendo le quantità di dati da generare, SMOTE supera tutte le altre metodologie.

Parole chiave: reti a microonde, apprendimento automatico, gestione dei guasti, guasti hardware, scarsità di dati, SMOTE, transfer learning, auxiliary-task learning, denoising autoencoders

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

1 Introduction **1**

1.1 Thesis Contribution 3

1.2 Thesis Outline 3

2 Related Work **5**

2.1 Use of Machine Learning in Microwave Networks 5

2.2 Data Scarcity in Communication Networks 7

3 Background **11**

3.1 Microwave Network Technologies 11

3.2 Machine Learning Methodologies 14

3.2.1 Classification problems 16

3.2.2 Classification Algorithms 24

3.3 Hyperparameter Optimization 35

3.3.1 XGBoost 37

3.3.2 SVC 37

3.3.3 ANN 38

3.4 Methodologies to address Data Scarcity 39

3.4.1 Data Augmentation 39

3.4.2 Transfer Learning 41

3.4.3 Auxiliary-Tasks Learning 44

3.4.4 Denoising Autoencoders 46

4 Failure-Cause Identification in Microwave Networks **49**

4.1 Dataset Description 49
4.2 Data Preprocessing 54
4.3 Baseline Machine Learning Models 58
4.4 Machine Learning methodologies to address Data Scarcity 63
4.4.1 Synthetic Minority Over-sampling TEchnique 63
4.4.2 Transfer Learning 65
4.4.3 Auxiliary-Task Learning 69
4.4.4 Denoising Autoencoders 73

5 Numerical Results 77
5.1 Baseline Models 77
5.2 Methodologies to address Data Scarcity 78
5.2.1 SMOTE 79
5.2.2 Transfer Learning 80
5.2.3 Auxiliary-Task Learning 81
5.2.4 Denoising Autoencoders 82
5.3 Best Methodology to address Data Scarcity 82
5.4 SMOTE Analysis 84

6 Conclusion and Future Developments 109

Bibliography 113

List of Figures 119

List of Tables 121

Acknowledgements 123

1 | Introduction

Failure management in communication networks is a critical issue nowadays, as a single failure in the network can lead to service disruption for thousands or millions of users at the same time. Network failures can be distinguished according to their nature in hardware failures, which are related to equipment malfunctioning as fan failure, power supply issues, overheating, and in propagation-related failures, which are caused by environmental reasons (i.e., atmospheric events, physical obstacles), as deep fading, interference, extra attenuation. Regardless of their nature, preventing failures from occurring is a crucial task for network operators to meet Service Level Agreements (SLAs) to its customers.

A combination of proactive measures and reactive measures can be undertaken in network failure management. The former aim at ensuring network stability preventing service disruption by anticipating failure occurrence, the latter are used to quickly address failures by activating recovery procedures to repair or substitute the failed equipment in the shortest possible time. Proactive measures are typically implemented on-line by continuously monitoring transmission-quality parameters, such as Bit-Error-Rate (BER), Signal-to-Noise Ratio (SNR), signal strength, attenuation, delay, and others. An example of proactive measure is failure prediction and early-detection, that aim to predict that a failure is about to occur. Reactive measures are used to deal with hard failures and are typically implemented off-line by leverage failure recovery information retrieved by network monitors and/or alarms (e.g., which equipment as failed, etc.). Examples of reactive measures are failure detection (i.e., recognize anomalies due to failure occurrences), failure-cause identification (i.e., understand the actual cause of the failure), and failure localization (i.e., identify where the failure occurred in the network).

Both proactive and reactive measures require collecting data generated from the continuous monitoring of network parameters and alarms. In a network where a large amount of data is available, the challenge is to analyze this data quickly so that the necessary actions can be taken to avoid service interruption or to restore the service quickly. Nowadays, this analysis is carried out by domain experts, who based on their experience identify the failures and engage proper countermeasures to mitigate them or to restore the service.

The time required by humans to perform the analysis and engage the countermeasures is often not in line with the stringent time constraint to restore the service after a failure imposed by the SLAs. To overcome this problem, substantial help comes from Artificial Intelligence (AI) and Machine Learning (ML), through which it is possible to automate and speed up the whole network management process by leveraging all the data retrieved monitoring the network.

The most widely used ML techniques come from the field of supervised learning, which require large amount of labelled data in order to learn "signatures" from it, that can be then recognized in future occurrences of similar failures. In real network deployments, limited amount of labeled data is available for training, as manual labeling is performed by domain experts based on their knowledge and experience. So although failure analysis is automated by ML techniques, the need to have a large amount of data to be labelled remains. When there is not enough labelled data to efficiently train an ML algorithm, we are facing the problem of data scarcity. To improve the performance of a supervised ML algorithm affected by data scarcity, alternative ML methodologies can be adopted that do not require new data to be labelled from the network.

In fact, in recent years ML has been used extensively to solve numerous problems in the field of telecommunications, and in our case we apply it to address failure-cause identification in microwave networks. As collecting data from the field is not cheap, in this work we focus on finding various ML methodologies to solve the problem of the lack of abundant data, a.k.a. "data scarcity", in order to classify different types of hardware failures. We make use of data provided by SIAE Microelettronica [38], an Italian company that manufactures telecommunications equipment and specializes in microwave networks. SIAE was responsible for collecting and classifying the alarms that occurred on the hardware components of a microwave network into different classes of hardware failures. For some of them, mainly due to the low frequency with which the faults occurred, we have only limited information available. These classes are therefore affected by data scarcity problem, which results in greater difficulty in training ML algorithms to correctly classify such faults. From network operators' point of view, the use of ML algorithms that are poorly trained in classifying these types of failures, even though they occur less frequently, has serious consequences both from an economic point of view, as specialized maintenance teams are sent to repair the wrong hardware components or at the wrong sites, and from the point of view of costumers satisfaction, as the time to repair the fault will be increased as a result.

To address the data scarcity problem we have investigated the following methodologies: *Synthetic Minority Over-sampling TEchnique (SMOTE)*, *Transfer Learning*, *Auxiliary-*

Tasks Learning, and *Denoising Autoencoders*. Among the various methodologies addressed, we will focus on the use of synthetic data, which allows to automatically generate new labelled data that we will use to improve the performance of the ML algorithms performing failure-cause identification, in particular for the failure classes affected by data scarcity.

1.1. Thesis Contribution

The main goal of this thesis is to address the data scarcity problem in the field of failure management for microwave networks. More specifically, it addresses the issue of failure identification for the hardware components of a microwave network. Specifically, the main contributions of this work can be listed as follows:

- We model the hardware failure identification problem as a machine learning classification problem.
- We use different supervised learning models to identify classes (i.e., failure causes) where classification performance is poor, showing the correlation between data scarcity and poor model performance.
- We investigate different methodologies to deal with the data scarcity problem, to understand which is the most appropriate for our scope.
- We conduct an in-depth analysis on synthetic data generation as a methodology to address data scarcity, proposing a strategy to identify which classes to actually generate synthetic data on and with what percentage.

1.2. Thesis Outline

The remainder of the thesis is organized as follows.

In *Chapter 2* we present an overview of previous work on failure management in communication networks, on the use of machine learning in communication networks, and on the use of synthetic data generation in communication networks.

In *Chapter 3* we present the background knowledge on the microwave network application domain and ML methodologies used to solve the failure-cause identification problem.

In *Chapter 4* we describe the dataset and the processing phase we do on it. Then we describe the pipeline we used to do classification with the basic models and with the methodologies we defined to address the data scarcity problem.

In *Chapter 5* we present the dataset after the preprocessing phase and the performance obtained with the baseline model and with the different methodologies proposed in chapter 4. We compare their performance and then we conduct an in-depth analysis on synthetic data generation.

In *Chapter 6* we conclude the thesis and identify possible future development of this work.

2 | Related Work

In this chapter we provide an overview on related work in the field of microwave communication networks and Machine Learning applied to communication networks. In Section 2.1, we concentrate on works that adopt ML in microwave networks emphasizing those that relate to failure management, while in Section 2.2 we concentrate on works that address the data scarcity problem in communication networks.

2.1. Use of Machine Learning in Microwave Networks

Several works have investigated the use of ML techniques applied to the field of microwave networks. The following works focus on the specific topic of the failure management in microwave network. In [39] the authors give an overview on supervised and semi-supervised learning approaches for automated failure-cause identification in microwave networks. Using real-field data they firstly identify six categories of failure causes in microwave networks and show that supervised ML enables very accurate failure identification reaching 93% classification accuracy. Then, to solve the problem of limited amount of labeled data available for training supervised ML models, they investigate a semi-supervised learning approach to automate labeling procedure, based on autoencoders-like Artificial Neural Networks, to combine the knowledge of the few manually-labeled data with large unlabeled data. They show that data augmentation based on autoencoders can slightly improve failure-cause identification only when Artificial Neural Networks or Support Vector Machines are used, while accuracy slightly decreases when adopting Random Forest. In [55] the authors propose an efficient multi-link faults location algorithm based on Hopfield Neural Network (HNN). Exploiting network topology and the services information, they modeled the relationship between fault set and alarm set and thanks to these information the HNN analyze the uncertainty of faults and alarms and locate the failures' position. In [41] is presented a machine-learning-based proactive microwave link anomaly detection system that exploits both performance data and network topological information to detect microwave link anomalies that may eventually lead to actual failures. This system, called PMADS, encode topological information into features and using a novel active learning algorithm, called ADAL, update continuously the detection model at low cost by first applying unsupervised learning to separate anomalies as outliers from the training set. Machine learning for anomaly detection and classification in cellular networks was studied in [13], comparing different ML based techniques as decision trees, SVM, and neuronal networks. The authors of [52] adopted Federated Learning (FL) to perform failure-cause identification in microwave networks, emulating a multi-operator scenario in which one operator has partial knowledge of failure causes during the training phase. They designed an FL-based classification model using the FedAvg algorithm and applied it to automate the identification of six different failure causes in microwave networks. Two recent works, [3] and [4], use Explainable Artificial Intelligence (XAI) to explain machine learning models for failure identification in microwave networks. The first one shows that SHapley Additive exPlanations (SHAP)-assisted feature selection allows reducing the number of features more than tree-based feature selection approach, while achieving an improvement in terms of model's accuracy. The second one explore the use of SHAP and Local Interpretable Model-agnostic Explanations (LIME) to address important practical questions with the aim of achieving a trustable deployment of automated failure-cause identification in microwave networks. The questions are: "Are the lists of most important features the same among the ML models?", "Which features are most influencing model's decision for each failure class and how?", and "Can we determine why the model systematically misclassifies instances of one class as instances of another particular class?".

ML is also applied in microwave networks to address different topics than the failure management. In [7] the authors propose a survey on ML-based approaches for radio resource management (RRM) in fifth-generation and beyond (5G/B5G) wireless cellular networks, pointing out which ML algorithms should be used in different RRM sub-problems (e.g., unsupervised clustering algorithms in RN selection, DNNs in subcarrier allocation and power management, etc.). In [50] ML-based approaches combined with software-defined network are suggested as solutions for Handover (HO) management in the 5G HetNet system, i.e., a system where small cells are suitable to operate by using mmWave due to its short-range, while macrocells are liable to use long-range radio waves. In [18] the authors applied a ML technique based on gradient boosting tree (GBT) to synthesis a linear millimetre wave (mmWave) phased array antenna and estimate the phase values of a 16-element array antenna to generate different far-field radiation patterns for various 5G mmWave transmission scenarios such as multicast, unicast, broadcast and unmanned aerial vehicle (UAV) applications. In [32] the performance of a throughput based adaptive beamforming framework in 5G massive multiple-input multiple-output (MIMO) millimeter wave cellular networks is evaluated with and without the aid of the ML. The results showed that ML-assisted beamforming framework can improve energy efficiency with reduced algorithmic complexity compared to the non-ML case, depending on the tolerable amount of blocking probability. The authors in [5] used an unsupervised clustering approach to reduce latency and select which low power nodes in 5G networks should be converted to fog nodes. ML is also applied to ensure and improve network security: in [26] successful detection of known and unseen attacks using both anomaly and misuse detection simultaneously has been reached building a real-time intrusion detection systems based on Recirculation Neural Networks, while in [35] the naıve Bayes, k-nearest neighbor and support vector machine classifiers has been used for prevent security problem and solve network congestion caused from anomalies in the network traffic.

Unlike previous work, the novelty we introduce in this paper is the use of different ML methodologies to solve the failure-cause identification problem in microwave networks to classify hardware failures occurring on a microwave network of which there is little data available. This problem is known as “data scarcity”, and related works on this topic are introduced in the next section.

2.2. Data Scarcity in Communication Networks

Authors in [10] introduce the problem of data scarcity, with the objective of understanding how to generate sufficient amounts of labeled data when it is sparse in order to build effective ML models. They proposed two approaches for learning from data that is dominantly unlabeled. In the first approach, the k-NN algorithm is applied to pre-label the unlabeled data. A multilayer perceptron is then used to classify the pre-labeled data. In the second approach, a prototypicality rule based on FCM is used to pre-label unlabeled data before training the MLP classifier. They showed that unlabeled data enhances the accuracy of the neural classifier. In [31] authors present an overview describes various semi-supervised learning techniques for classifying data streams with limited labeled data. Specifically, they used Decision Tree based techniques, Clustering based techniques, Ensemble based techniques, Expectation Maximization (EM) techniques, and Graph based techniques.

To address data scarcity problem different methodologies can be applied. Some of them are based on Data Augmentation, which involves generating additional data from the existing data by applying transformations such as rotation, scaling, or flipping. This can increase the size of the dataset and help to reduce the risk of overfitting. In [2] authors proposed two Non Parametric Data Augmentations techniques to improve Deep-Learning models for brain tumor segmentation. These techniques are the mixed structure regularization (MSR) and shuffle pixels noise (SPN). Another data augmentation technique is “SMOTE” (Synthetic Minority Over-sampling Technique), which generates synthetic examples of a class by interpolating between existing examples of that class. SMOTE has been used in several works, as in [47] where the authors developed a machine learning algorithms for the prediction of machine failures, using a synthesized dataset balanced with the use of SMOTE, and in [57] where SMOTE was tested over the Three-phase Flow Facility (TFF) , that is an experimental simulation object for the research on the fault diagnosis of complex industrial processes. In [48] the authors proposed an approach to suggest a value of the parameter k, used in SMOTE to define the neighborhood of samples to use to generate the synthetic samples, adopting a Natural Neighbor algorithm. In [22] SMOTE has been compared with other techniques to understand which one performs better over an imbalanced dataset. The authors of [44] proposed a hybrid method to classify binary imbalanced data using SMOTE followed by Extreme Learning Machine (ELM) is proposed. The authors in [6] proposed two novel SMOTE methods for solving imbalanced classification problems, known as Center point SMOTE (CP-SMOTE) method and Inner and outer SMOTE (IO-SMOTE) method. The CP-SMOTE method generates new samples based on finding several center points, then linearly combining the minority samples with their corresponding center points. The IO-SMOTE method divides minority samples into inner and outer samples, and then uses inner samples as much as possible in the subsequent process of generating new samples. Instead authors in [23] proposed a modified SMOTE approach called MSMOTE, which not only considers the distribution of minority class samples, but also eliminates noise samples by adaptive mediation.

Other methodologies to address data scarcity are based on transfer learning (TL), which involves using a pre-trained model that has been trained on a larger dataset to initialize the weights of a model for a smaller dataset. This can help to improve the performance of the model by leveraging the knowledge learned from the larger dataset. This approach has been used in [58], where with limited measurement data a TL ensemble model was adopted for evaluating power transformer health conditions. Specifically, a target model parameters are initialized with those of the source model and fine-tuned using the target dataset, then a transfer strategy is proposed to decide what and where the diagnostic knowledge should be transferred from the source domain to the target domain. Finally, an ensemble model is built on the basis of a series of target models with different transfer strategies, which can further alleviate the overfitting issue and improve the generalization ability.

In [36] this methodology was combined with another methodology that is the data synthesis, which involves creating synthetic data to supplement the existing data. In particular, TL was combined with the generative adversarial network (GAN) to investigates the problem of data scarcity in spectrum prediction. Other works investigated this methodology, as in [25], where GAN was used to deal data scarcity problem in photonic-based microwave frequency measurement. With GAN authors were able to augment the 75 sets of experimental data into 5000 sets of data for training the model, effectively reducing the amount of experimental data needed by 98.75%, and reducing frequency estimation error by 10 times. Another way to perform data synthesis is the use of Variational Autoencoders (VAEs). They have been used in [43], to predict the performance of a double-T monopole antenna, and in [20], to define a nonlinear model compression scheme for microwave data inversion. ì

Another methodology that we can cite to address data scarcity is the use of Active learning, which involves selecting the most informative examples to label and adding them to the dataset, rather than labeling all examples. This can help to reduce the cost of labeling and improve the quality of the dataset by focusing on the most informative examples. This techniques was used in [21] where the authors make a perspective study on overcoming the data scarcity in smart grid. The active learning strategy they proposed provide a feasible solution for addressing the data scarcity challenge. In addition, they give a discussion on current state-of-the-art and the limitations in previous work.

Different from the other works mentioned above, in this paper we try to compare different methodologies for dealing with the data scarcity problem to classify different types of hardware failures in microwave networks. We proposed a TL-based method, an Auxiliary-Task Learning-based method, a Denoising Autoencoder-based method and finally a SMOTE-based method for generating synthetic data. All these methods were compared with the performance obtained from basic classifiers, such as XGBoost, Support Vector Classifier and Artificial Neural Network, showing which are the most suitable and which are the least suitable for addressing the data scarcity problem. Finally, a more in-depth study was done on SMOTE, as it was found to be the best among the considered methodologies. A final note on this work is that here, unlike most other work, SMOTE is used as a specific technique for solving data scarcity problem and not as a specific technique for solving class imbalanced problem.

3 | Background

In this chapter we provide background information on technological and theoretical aspects covered in this thesis. Specifically, Chapter-3 is organized as follows: in Section 3.1, we introduce the hardware components of a microwave network, in Section 3.2 we discuss general principles of Machine Learning, with emphasis on classification problems and algorithms used to deal with it. In Section 3.3 we describe the methods used to optimize hyperparameters. Finally, in Section 3.4 we introduce the concept of "data scarcity" and discuss several approaches that we used to address it.

3.1. Microwave Network Technologies

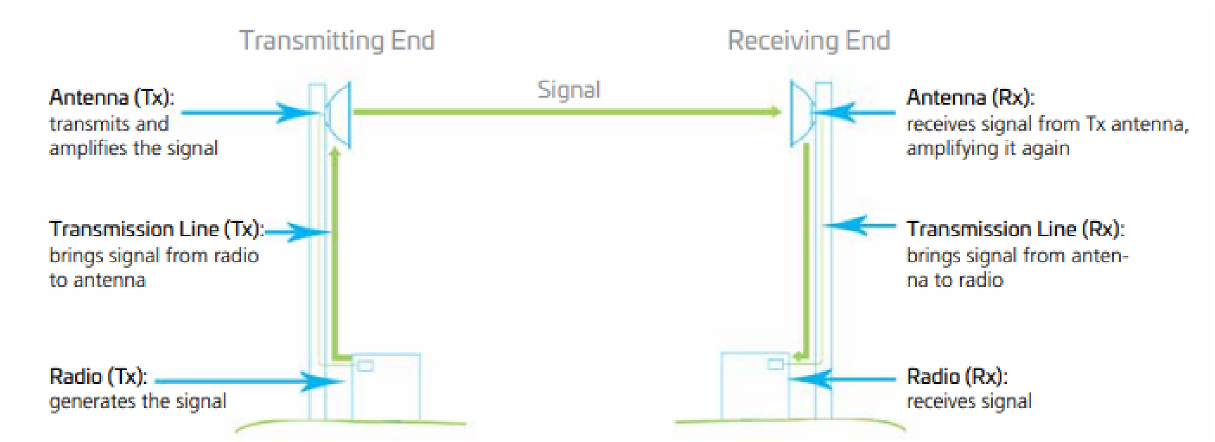

Transmitting End
Receiving End
Signal
Antenna (Tx): transmits and amplifies the signal
Transmission Line (Tx): brings signal from radio to antenna
Radio (Tx): generates the signal
Antenna (Rx): receives signal from Tx antenna, amplifying it again
Transmission Line (Rx): brings signal from antenna to radio
Radio (Rx): receives signal

Figure 3.1: Basic components of a microwave link: Microwave Radio, Transmission Line, and Antenna [16].

A microwave network is a communication network that uses microwave radio frequencies, that range from 300 MHz to 300 GHz (1 m - 1 mm wavelength), to transmit and receive data between two or more points [49]. Microwaves are characterized by high-frequency and small wavelength. High frequency allows them to provide high-speed wireless connections capable of sending and receiving voice, video, and data information. Microwave connections have 30 times the bandwidth of the rest of the underlying radio spectrum.

Instead, the small wavelength allows properly sized antennas to direct the waves into narrow beams that can be pointed directly at the receiving antenna, making this technology suitable for point-to-point communications. This allows nearby microwave equipment to use the same frequencies without interfering with each other, as low-frequency radio waves do.

Figure 3.1 shows the basic structure of a microwave link, constituted by the following three main building blocks:

- Microwave Radio: at each side of the microwave link we have a Microwave Radio typically with both transmission (TX) and receiving (RX) capabilities. On the transmitter side the Microwave Radio is responsible of generating the analog signal, while on the receiving side it is responsible of demodulating the signal. The Microwave Radio has three basic configuration used in microwave communications systems [16]:
  - *Full Indoor*: all active components are located inside a building or shelter, allowing easy maintenance and upgrades, without requiring tower climbs, for instance. Being farther from the antenna may introduce higher transmission line losses than other configurations.
  - *Full Outdoor*: all electronic devices are mounted outside, eliminating the need and cost for indoor space. However, because they are located on the tower, they can be difficult to access for maintenance or upgrades, requiring tower climbs. In some cases, rooftop access mitigates this challenge.
  - *Split-Mount*: electronic devices are distributed into an outdoor unit (ODU) and indoor unit (IDU), eliminating transmission line losses with easy maintenance of the IDU. However, it also combines the disadvantages of the other two configurations by requiring indoor storage and tower climbs for the ODU. The ODU is responsible for the signal generation, while the IDU is responsible for receiving the microwave signals from the ODU, demodulating them, and processing them into a form that can be used by the local network or devices. In figure 3.2 an example of Split-Mount Microwave Equipment installation. This is the configuration adopted in the microwave network used for this thesis work.

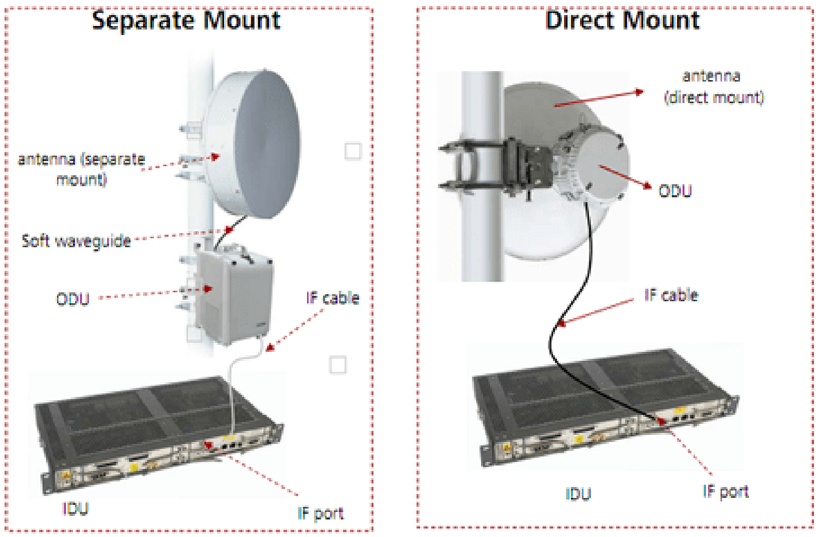


Figure 3.2: Split-Mount Microwave Equipment installation, with two possible configuration: on the left side the "Separate Mount" configuration where the ODU is located saparate from the Antenna, while on the right side the "Direct Mount" configuration where the ODU is directly mounted behind the Antenna. [17]

- Transmission Line: it is the physical media connecting the radio and the directional antenna. Because of the amount of signal loss they can introduce, the choice of transmission line type is determined largely by the frequencies in use. The two possible implementations are [16]:
  - *Coaxial Cable*: is suitable in applications using frequencies up to—or just above—2 GHz. Above this range, most lengths become too lossy to be of practical use.
  - *Waveguide*: is suitable for higher frequencies. Elliptical waveguide features an elliptical cross section and can support frequencies up to around 40 GHz, however it is rarely used in applications above 13 GHz.

  To avoid the need for transmission line, frequently in the split-mount radio configuration the ODU is mounted directly to the antenna with a special interface plate, as shown in figure 3.2 (Direct Mount).

- Antenna: is the device that transmits and receives electromagnetic waves. A microwave communication system usually uses directional antenna with parabolic shape to allow the greatest focus of energy possible in a single beam. There are different types of antennas, each characterized by:
  - Gain: is a measure of the ability of an antenna to focus electromagnetic radia-

tion in a particular direction, compared to an isotropic radiator (a theoretical antenna that radiates equally in all directions). It is expressed as the ratio of the radiation intensity in a given direction to the radiation intensity that would be produced if the power accepted by the antenna were isotropically radiated [1].

  - Size: is the physical dimensions of an antenna dish. Larger antennas generally have higher gain and can transmit signals over longer distances, but they may be more difficult to install and may require more precise alignment. Smaller antennas are typically easier to install and align, but they may have lower gain and transmit signals over shorter distances.
  - Directivity function: it describes how much power an antenna radiates in a particular direction compared to the power it would radiate if it were an isotropic radiator. It is expressed as the ratio of the radiation intensity in a given direction from the antenna to the radiation intensity averaged over all directions [1].

For this thesis work, we focus on hardware failures affecting microwave networks. Thanks to the use of a Network Management System (NMS) of SIAE Microelettronica, the network infrastructure has been monitored, all the alarms have been recorded and classified by domain experts. The next step, is the use of Machine Learning to train a classification algorithm able to automatically classify these types of failures.

3.2. Machine Learning Methodologies

The goal of Machine Learning is to develop models that can extract information from available data and make decisions or predictions based on the learned information. Machine learning algorithms are broadly categorized into the following learning strategies:

- *Supervised learning*: it consists of training a model based on “labeled” data, that means the data set used for training consists of input data, also known as features, and corresponding output data, also known as labels. The goal is to train a model that can accurately map inputs to outputs, so that it can make accurate predictions or decisions when presented with new unseen input data. Depending on the output there are two main types of supervised learning problems:
  - *Regression problem* :here the output to predict is a continuous numerical value. Regression algorithms are used to predict the value of an unknown variable based on input features. An example can be predicting the price of an house

given some characteristics such as the size, the location, the floor, the number of bedrooms, and other relevant features. Another example can be predicting the temperature based on wind speed, atmospheric pressure, humidity, etc.

- *Classification problem* : here the output to predict is a discrete (or even "categorical") value. Classification algorithms are used to classify input data into different categories based on their features. Examples can be classify if a financial transaction is fraudulent or not fraudulent, or classify an email as spam or not spam, or classify an object in one of the possible type of object.

- *Unsupervised learning* : consists in training a model based on "unlabeled" data, that means the data set used for training consists only of input data, also known as features. The goal of unsupervised learning is to discover hidden structures or relationships in the data that can be used to gain insights, make predictions, or inform decision-making. Unsupervised learning can be used for tasks such as clustering, anomaly detection, and dimensionality reduction.

- *Semi-Supervised learning*: consists of training a model based on a combination of "labeled" and "unlabeled" data, exploiting the characteristics of both supervised and unsupervised learning. Semi-supervised learning is useful when labeled data is limited or expensive to obtain; by incorporating unlabeled data into the training process, the model can potentially learn more about the underlying structure of the data and improve its performance on the labeled data.

- *Self-Supervised learning*: is a type of unsupervised learning, since it is based on "unlabeled" data, where the model itself defined and solves a supervised learning task based on the data it's given. Given to the model only a training set of input data, it has to find structure in the data in order to learn from it unobserved or hidden part of the input data itself. The typical Self-Supervised learning model is based on an artificial neural network composed of two network:

  - **Encoder**: it translates the original high-dimension input x into the latent low-dimensional representation c(x).

  - **Decoder**: it reconstruct the input x from that representation c(x), with larger and larger output layers.

  An ANN with this structure is called Auto-Encoders [8]. Self-Supervised learning can be summarize the into three general categories [37], as shown in Figure 3.3:

  - *Generative*: train an encoder to encode input x into an explicit vector z and a decoder to reconstruct x from z.

  - *Contrastive*: train an encoder to encode input x into an explicit vector z to measure similarity.
  - *Generative-Contrastive (Adversarial)*: train an encoder-decoder to generate fake samples and a discriminator to distinguish them from real samples (e.g., GAN).

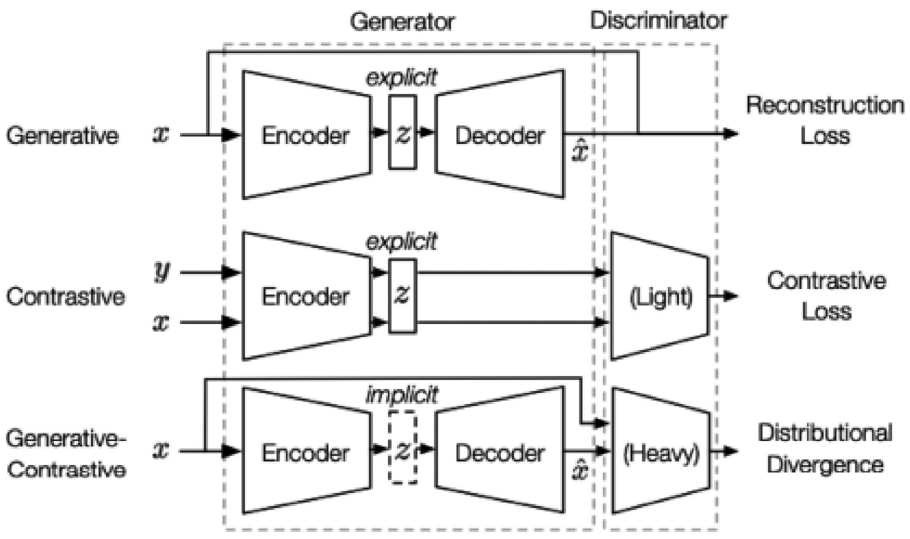


Figure 3.3: Conceptual comparison between Generative, Contrastive, and Generative-Contrastive methods [37].

- *Reinforcement learning* : here an agent learns to make decisions by interacting with an environment, whit the goal of learning a policy, or a set of actions, that maximize a reward signal over time, also known as the return. To reach this scope the agent receives feedback in the form of rewards or punishments based on the actions it takes in the environment.

In the following section, we detail classification as failure-cause identification falls in this category of machine learning algorithms.

3.2.1. Classification problems

In this thesis we model the failure-cause identification as a classification problem. A classification problem has the goal to learn a model that can assign a class or category label to input data based on a set of labeled examples. There are two main types of classification problems:

- *Binary Classification*: with the goal of classifying input data into one of two classes or categories, such as true/false or yes/no.

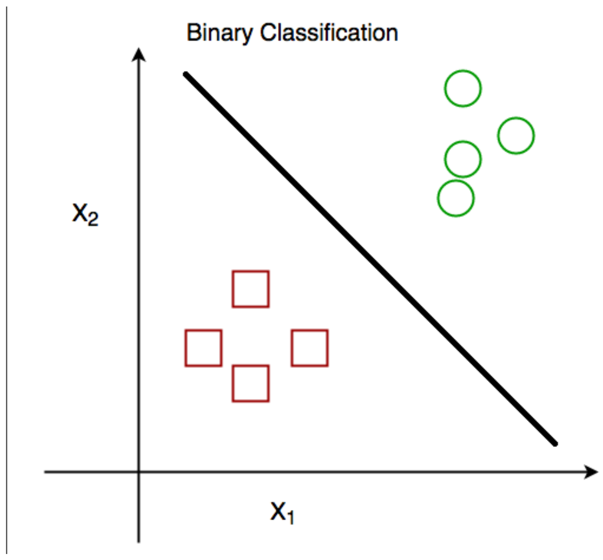


Figure 3.4: Data classification in two classes, i.e., red square or green circle.

- *Multiclass Classification*: with the goal of classifying input data into one of N-classes or categories, such as different types of failure in microwave networks.

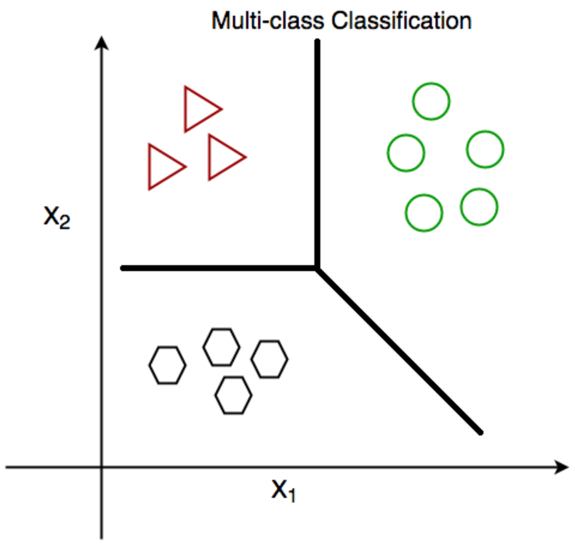


Figure 3.5: Data classification in 3-classes, i.e., red square, green circle or black hexagon.

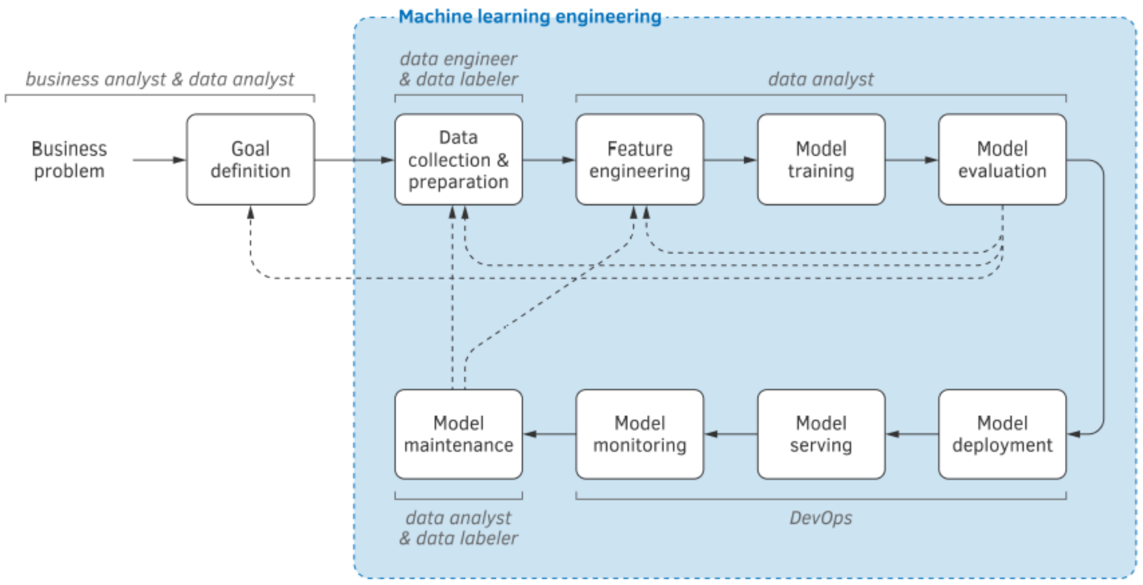


Figure 3.6: Machine learning project life cycle. The scope of machine learning engineering is limited by the blue zone. The solid arrows show a typical flow of the project stages. The dashed arrows indicate that at some stages, a decision can be made to go back in the process [12].

As described in [12] and shown in Figure 3.6, a machine learning project life cycle consists of the following stages:

1. **Goal Definition**: in this first phase usually a business analyst works with the client and the data analyst to transform a business problem into an engineering project. Once an engineering project is defined, the machine learning part can start. The first task is the goal definition: is a specification of what a statistical model receives as input, what it generates as output, and the criteria of acceptable (or unacceptable) behavior of the model;

2. **Data collection and preparation**: in this phase firstly a consisting of input data and corresponding output data (labels) is collected and then it is prepared (or pre-processed) for the target problem.

   - Data Collection: this phase is critical for the success of the ML tasks. A successful data collection contains enough information that can be used for modeling, has good coverage of what we want to do with the model, and reflects real inputs that the model will see in production. It is as unbiased as possible and not a result of the model itself, has consistent labels, and is big enough to allow generalization.

- Data Preparation: after data collection, the dataset to be suitable for the machine learning algorithms must be prepared/preprocessed. This preparation phase could consist of feature analysis, dataset cleaning (outlier removal), handling missing values, and performing other data transformations as needed, like data augmentation and data partitioning in training set, validation set, and test set. To obtain a good partition of your entire dataset into training, validation and test sets, the process of partitioning must satisfy several conditions:
  - Data was randomized before the split.
  - Split was applied to raw data.
  - Validation and test sets follow the same distribution.
  - Leakage was avoided.

3. **Feature Engineering**: this phase is made of all the operations on the dataset that will help the algorithm to learn and make accurate predictions. During this phase feature selection, dimensionality reduction, feature scaling and normalization are usually done.

4. **Model Training**: in literature an huge amount of classification algorithms are available. The goal of this phase is selecting an appropriate machine learning model for the problem at hand, based on factors such as the size of the dataset, the complexity of the problem, and the desired prediction accuracy. The selected machine learning model has two important components, that are the:

   - Model parameters: are variables that define the model trained by the learning algorithm. They are directly modified by the learning algorithm based on the training data. The goal of learning is to find such values of parameters that minimize the prediction error. Examples of parameters are weights (W) and biases (b) in the equation of linear regression y = Wx+b, where x is the input of the model, and y is its output (the prediction). However, not every ML model is parametric, as Decision Trees and K-means.
   - Model hyperparameters: are inputs of machine learning algorithms that influence the performance of the model. They are not learned by the model during training, but instead are set before training begins and determine how the model is trained. Example of hyperparameters are the maximum depth of

the tree in the decision tree learning algorithm, the misclassification penalty in support vector machines, k in the k-nearest neighbors algorithm, the learning rate, batch size, and number of epochs, and many others.

In this phase to find the best configuration values for the learning algorithm (best hyperparameters), the training and validation sets are used. There are available different techniques for validating a model, all with the goal of generalize well the results to new and unseen data. The choice of which technique to use depends on factors such as the size of the dataset, the complexity of the algorithm, and the available computing resources. The technique used in this work is known as *K-Fold Cross-Validation*: it consists in splitting the data into k non-overlapping parts or "folds", train the algorithm on k-1 folds, and validate its performance on the remaining fold. This procedure is repeated k times, using each fold as the validating set once. This can provide a more statistically unbiased estimate of the algorithm's performance, as it validates the algorithm on multiple subsets of the data. With the Stratified K-Fold Cross-Validation is possible to apply on this techniques the benefits carried by the stratification.

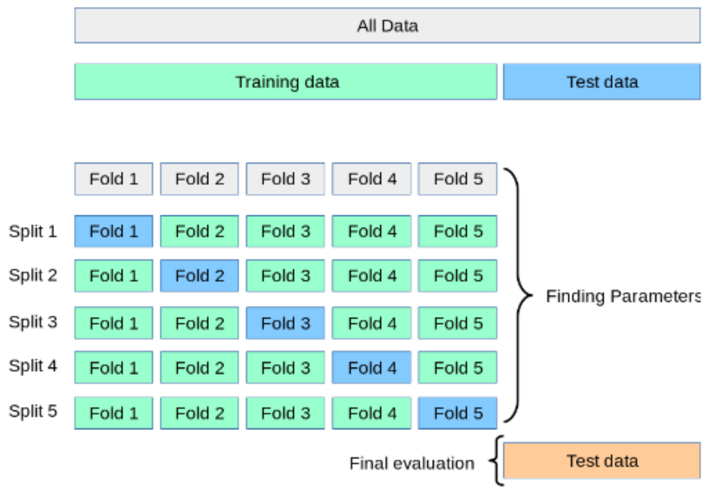


Figure 3.7: K-Fold Cross-Validation [33].

There are other possible techniques for validating a model, as:

- *Holdout Validation*: it consists in splitting the training set into training and validation set and use the last one to test the algorithm's performance after it has been trained on the training set with a configuration of hyperparameters. To get better performance we can use on it the stratification, that allows to maintain in both the train and validation set the same distribution of classes

The main advantage of holdout validation is that it allows the algorithm to be tuned on a separate validation set, which helps to prevent overfitting to the training data.

- *Leave-one-out cross-validation (LOOCV)*: it is a special case of cross-validation where k is set to the number of data points (N), making the algorithm training on all but one of the data points (N-1), and validating its performance on the left-out point. This methods can be computationally expensive, but it can provide a less biased estimate of the algorithm's performance than other validation techniques such as k-fold cross-validation, especially when the dataset is small.
- *Bootstrapping Validation*: it consists in combining bootstrapping and holdout validation techniques. In this way we can get estimating of the algorithm's performance with less variance than in holdout validation. The main difference with respect to K-Fold Cross-Validation is that in Bootstrapping Validation, doing resampling with replacement can lead to have the same point repeated in the training set, or used in more than one test sets. This is not possible using the K-Fold Cross-Validation techniques.

5. **Model Evaluation**: the performance of the trained model is evaluated on the test set, that is the part of the dataset that has not been used during the model training phase, so it contains labelled data that have never been seen by the model. To assess the performance, different metrics calculated over the test data can be used: the most widely used are accuracy, precision, recall, and f1-score. These metrics can be calculated through the confusion matrix, that summarizes how successful the classification model is at predicting examples belonging to various classes. Figure 3.8 shows the confusion matrix used for multiclass classification: it is a table where on one axis there is the class that the model predicted (usually the x axis), on the other axis there is the actual label (usually the y axis), and the cells represent the number of samples that fall into each of these categories:
   - True positive (TP): the model correctly predicts that a sample belongs to the positive class.
   - True negative (TN): the model correctly predicts that a sample belongs to the negative class.
   - False positive (FP): the model predicts that a sample belongs to the positive class when it actually belongs to the negative class.

- False negative (FN): the model predicts that a sample belongs to the negative class when it actually belongs to the positive class.

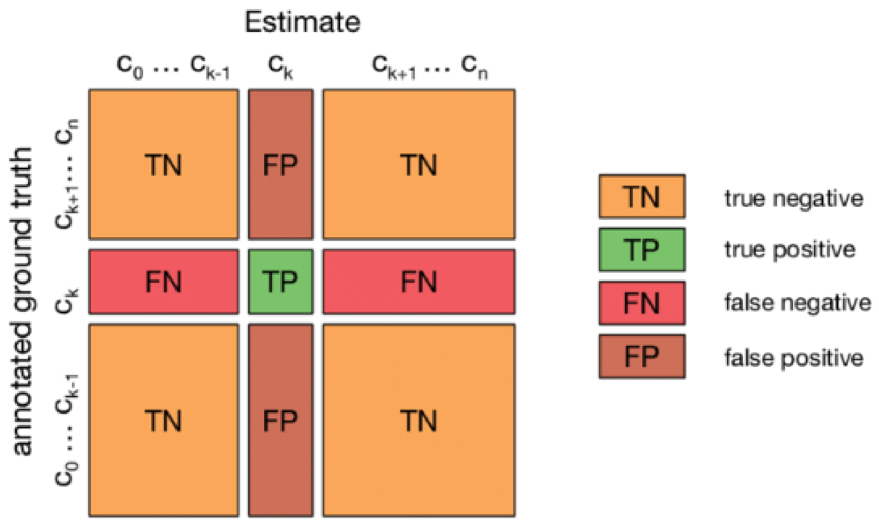


Figure 3.8: Confusion Matrix for Multiclass Classification

Once the confusion matrix is built we can calculate the different performance metrics as:

- **Accuracy**: it is defined as the ratio between the correctly classified samples to the total number of samples, i.e., (3.1):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

  Intuitively, accuracy measures how well the model can correctly classify both positive and negative instances in a dataset.

  Its complement is called Error Rate (ERR) and is calculated as EER = 1-Acc. It represents the number of misclassified points in both side positive and negative class.

- **Precision**: it is the ratio of true positive predictions to the overall number of positive predictions, i.e., (3.2):

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

  Intuitively, precision measures how well the model can correctly identify positive instances without falsely labeling negative instances as positive. A high precision score indicates that the model is making fewer false positive predictions, which can be useful in applications where false positives are costly.

- **Recall**: it is the ratio of true positive predictions to the overall number of positive examples, i.e., (3.3):

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

  It is also known as sensitivity or true positive rate and intuitively it measures how well the model can identify all positive instances in the dataset, without missing any positive instance. A high recall score indicates that the model is making fewer false negative predictions, which can be useful in applications where it is important to identify all positive instances.

- **F1-Score**: it is an harmonic mean of precision and recall and provides a balance between these two metrics, i.e., (3.4):

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{3.4}$$

  Intuitively, F1-score measures the trade-off between precision and recall, and provides a single value that summarizes the overall performance of the model. A high F1-score indicates that the model is able to achieve both high precision and high recall, which can be useful in applications where both false positives and false negatives are costly.

However, especially when the classes are imbalanced or when the cost of false positives and false negatives is different, considering just one metric alone may not be the best choice, so for this reason they are usually compared together.

Once, the metrics results on the test set are available, it is necessary to validate them, using the techniques described above, but considering the train and test sets instead of the train and the validation sets. This will allow to get an error rate on the considered performance metrics that are as near as possible to the real error rate of the model.

6. **Model Deployment**: in this phase the trained model is ready to be inserted in the production environment, where it can be employed by the users to make predictions on new, unseen data. A trained model can be deployed in various ways. It can be deployed on a server, or on a user's device. It can be deployed for all users at once,

or to a small fraction of users.

7. **Model Serving**: in this phase the machine learning model can be served in either batch mode, when it is applied to large quantities of input data, or on-demand mode either a human client or a machine.

8. **Model Monitoring**: in this phase the model is monitored to make sure that it is served correctly, and the performance of the model remains within acceptable limits.

9. **Model Maintenance**: during this phase the model that is in the production environment is regularly updated to correct possible prediction errors, thanks to the availability of new training data, and in general to be always "fresh" and so useful to the clients.

3.2.2. Classification Algorithms

In literature many supervised learning algorithms for solving classification problems are available. In this section are described the ones used in the thesis, focusing on the theory behind them.

XGBoost (eXtreme Gradient Boosting)

eXtreme Gradient Boosting (XGBoost) is a gradient boosting algorithm that uses decision trees ensembles as base learners, i.e., an individual learner of the ensemble, to iteratively improve the predictions of the model. To understand this definition let analyse the key concepts of XGBoost: decision trees ensembles and gradient boosting.

Figure 3.9 shows the structure of a decision tree model, where each internal node represents a feature, each branch represents a decision, and each leaf node represents a class label. An instance is classified by starting at the root node of the tree, testing the feature specified by this node, then moving down the tree branch corresponding to the value of the feature in the given example. This process is then repeated for the subtree rooted at the new node. In this way, tree-based methods partition the feature space into a set of rectangles, where in classification problem at each rectangle a label is assigned.

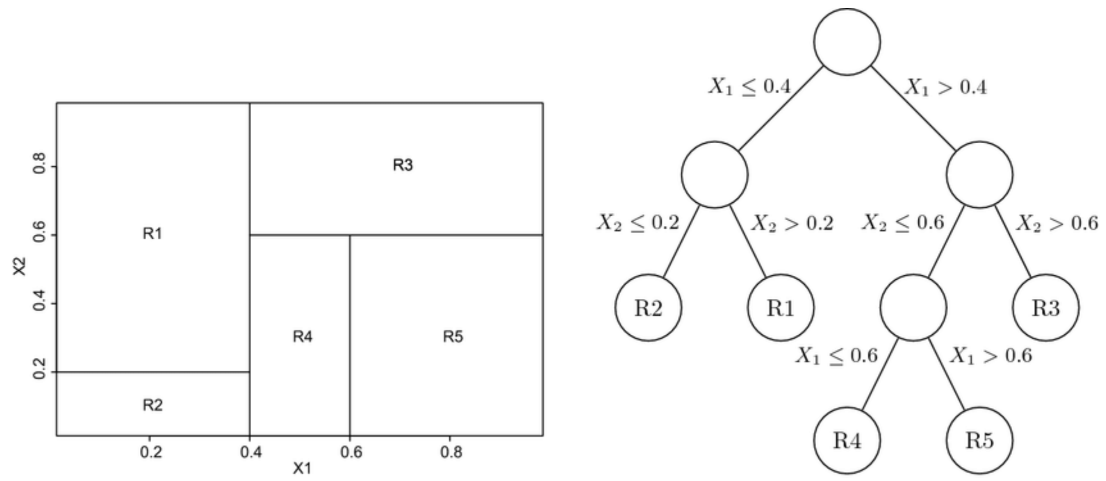


Figure 3.9: Decision tree example.

The training process for a decision tree can be summarized as follows:

1. Start with the entire training dataset at the root node of the tree.
2. Select a feature or attribute that best separates the data into different classes or categories. This is done by calculating a splitting criterion for each feature, such as Gini impurity, Entropy and Information Gain.
3. Split the data into two or more subsets based on the selected feature and the splitting criterion.
4. Create a new internal node for the selected feature, and add child nodes for each subset created in step 3.
5. Repeat steps 2-4 for each child node, until a stopping criterion is met. The stopping criterion may be based on a predefined maximum depth of the tree, a minimum number of samples required to split a node, or a minimum improvement in the splitting criterion.
6. Assign a class label or a numeric value to each leaf node based on the majority class label or the average numeric value of the samples in the node.

Decision trees can be prone to overfitting if they are too complex, and may not perform well on datasets with imbalanced class distributions or noisy data. Therefore, it is important to use regularization techniques, such as pruning, to prevent overfitting and improve the generalization performance of the decision tree. Pruning involves removing nodes from the tree that do not improve its predictive performance on the validation dataset. There are two types of pruning:

- *Pre-pruning* : involves stopping the growth of the tree based on a stopping criterion, such as a maximum depth, a minimum number of samples required to split a node, or a minimum improvement in the splitting criterion.
- *Post-pruning* : involves growing the tree to its maximum size and then removing nodes from the bottom up based on their impact on the validation error.

Figure 3.10 shows another way to deal with bias and overfitting that is the use of decision tree ensembles, which combines several decision trees to produce better predictive performance than utilizing a single decision tree. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner.

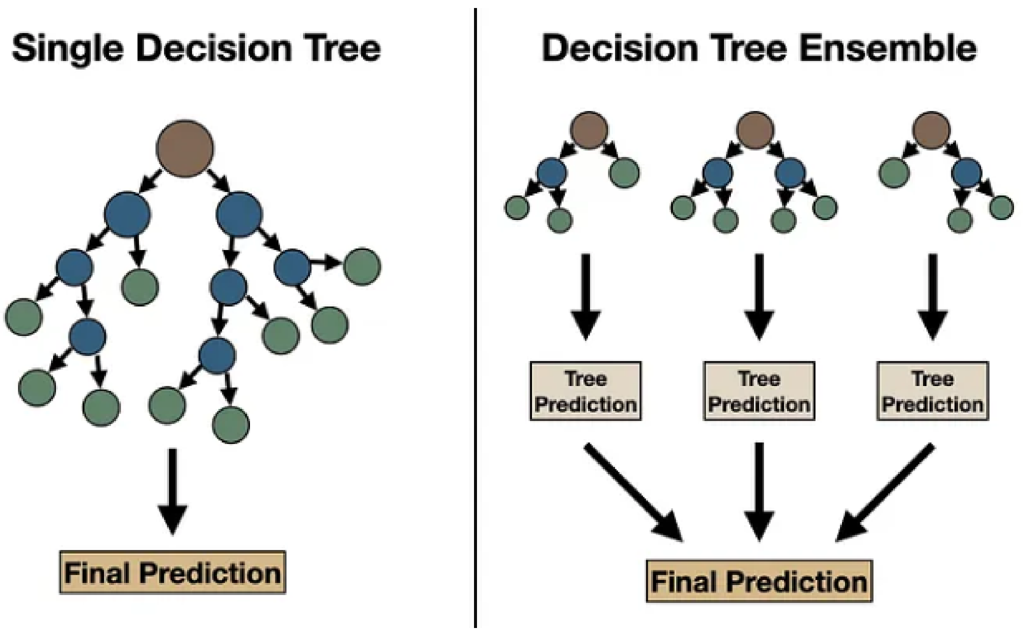


Figure 3.10: Decision tree ensambles.

Two popular methods to combine a set of decision trees are bagging and boosting:

- *Bagging*: stands for bootstrap aggregating and is the techniques used in Random Forest (RF). It consists in randomly sampling with replacement the initial training dataset to derive a set of training datasets used each to grow a decision tree. The set of decision trees forms a decision tree ensemble whose predictions can be aggregated (via mean, median, etc.) to obtain better prediction performance. This technique is shown in Figure 3.11

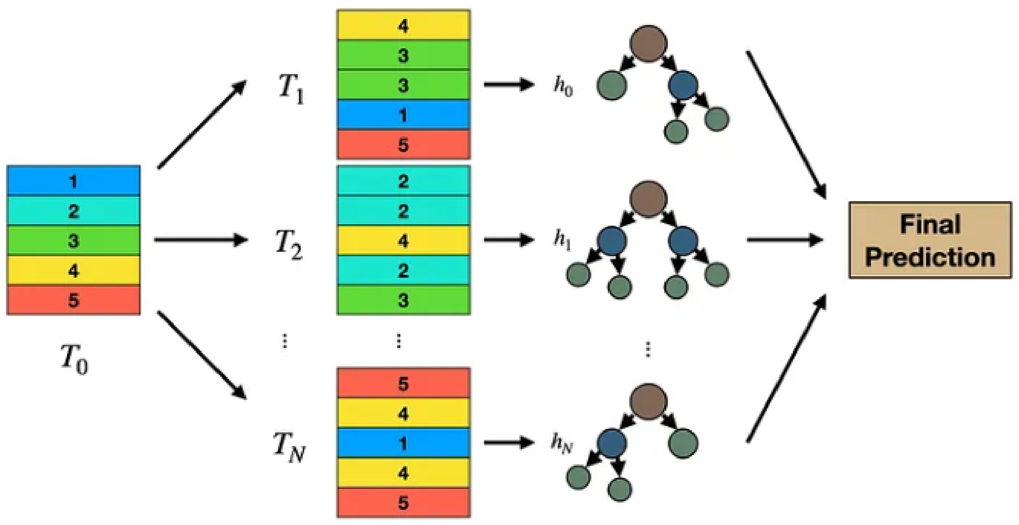


Figure 3.11: Bagging techniques.

This helps improves model performance minimizing the variance and overfitting, because it avoids the model over-optimizing to any single dataset.

- *Boosting*: consists in adding trees to correct the errors (also called residuals) made by existing tree. Trees are added sequentially until no further improvements can be made. Figure 3.12 shows an example of this technique. Starting with a single training dataset it is used to grow a single decision tree. Afterward, the training dataset is re-weighted so that records with incorrectly predicted targets receive more weight. The new training dataset, then trains another decision tree. Re-weighting and training processes are repeated until a set of decision trees are built. The set of decision trees forms a decision tree ensemble whose predictions can be aggregated (via mean, median, etc.) to obtain better prediction performance.

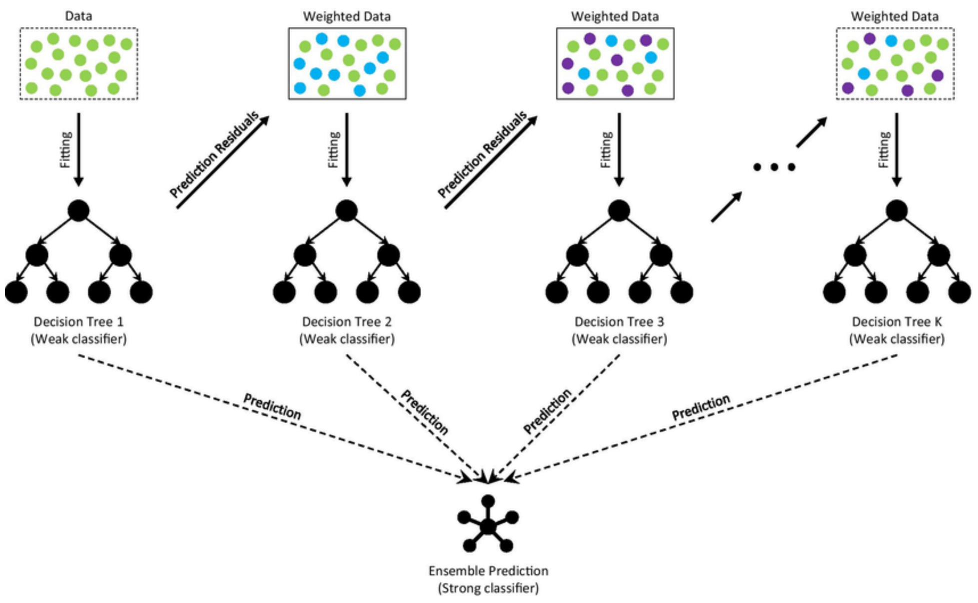


Figure 3.12: Boosting techniques.

This helps improves model performance minimizing the bias and underfitting.

These are the general concepts upon which XGBoost is built. Specifically XGBoost works in this way:

1. **Model initialization**: XGBoost starts by initializing a single decision tree as the base model. The trees used by XGBoost are a bit different than the traditional ones. They are called CART trees (Classification and Regression trees) (3.13) and instead of containing a single decision in each "leaf" node, they contain real-value scores of whether an instance belongs to a group. After the tree reaches max depth, the decision can be made by converting the scores into categories using a certain threshold.

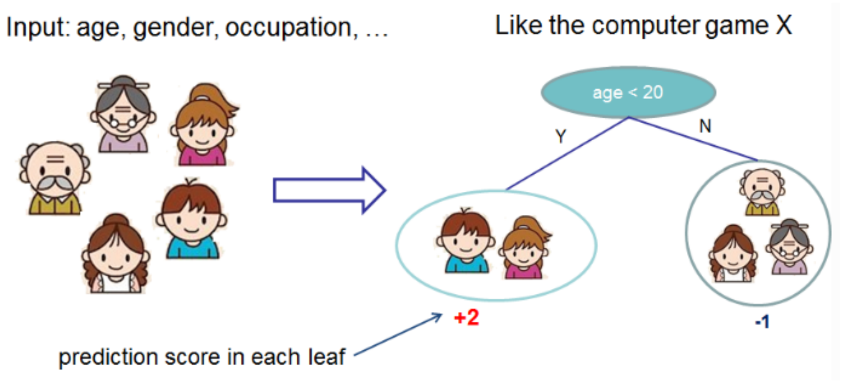

Input: age, gender, occupation, ...
Like the computer game X
age < 20
Y
N
+2
-1
prediction score in each leaf

Figure 3.13: Cart (Classification and Regression trees) [15].

2. **Loss computation**: the loss function L is shown in (3.5) and is used by XGBoost to measure the difference between the predicted values and the true values of the target variable in the training set.

$$L = \sum_{i=1}^{N} l(y_i, \hat{y}_i) \tag{3.5}$$

where y_{i} is the true value of the target variable for the i-th sample, and $\hat{y}_{\mathrm{i}}$ is the predicted value of the target variable for the i-th sample. The function l is a differentiable loss function that measures the difference between the true and predicted values, such as squared loss, logistic loss, or hinge loss.

3. **Gradient and Hessian computation**: XGBoost computes the gradient and the Hessian of the loss function with respect to the predicted values, as in (3.6) and (3.7) respectively. These values are used to optimize the objective function during the training process.

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i} \tag{3.6}$$

and

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2} \tag{3.7}$$

4. **Tree growing**: XGBoost fits a decision tree to the negative gradient values $-g_{\mathrm{i}}$, where each leaf node of the tree represents a predicted value. XGBoost uses the Hessian values h_{i} to compute the weights of the leaf nodes, which can be interpreted as a measure of the uncertainty of the predictions.

5. **Model updating**: XGBoost adds the new decision tree to the ensemble using Gradient Boosting, that does not generate subsequent trees through re-weighting the training data but rather through a gradient descent algorithm to minimize the loss L on training set when adding new models.

6. **Repeat**: steps 2 to 5 are repeated until the desired number of trees are added to the ensemble or until the loss stops decreasing.

7. **Make prediction**: to make a prediction, the model uses the values of the features to traverse the decision trees and compute the weighted sum of the leaf node values. Mathematically, the predicted value of the k-th tree $f_{\mathrm{k}}(x)$ can be computed by recursively adding the predictions of the previous trees, as in (3.8):

$$f_k(x) = f_{k-1}(x) + h_k(x) \tag{3.8}$$

where $h_k(x)$ is the prediction of the k-th tree for input x.

In summary, XGBoost is a gradient boosting algorithm that uses decision trees as base learners to iteratively improve the predictions of the model. It optimizes the model parameters by minimizing a differentiable loss function using gradient descent, and updates the model by fitting new decision trees to the negative gradients of the loss function. XGBoost, works via approximations that make gradient boosting more computationally efficient than standard gradient boosting. XGBoost strengths are its the ability to handle large datasets with a mix of categorical and numerical features, its speed, its ability to handle missing data and the presence of several regularization techniques to prevent overfitting, such as L1 and L2 regularization and early stopping.

SVC (Support Vector Classifier)

Support Vector Classifier (SVC) is a specific type of SVM (Support Vector Machine) that is optimized for classification tasks with linearly separable data. The goal of SVC is to find the optimal hyperplane that separates the data into different classes by maximizing the margin, that is the maximum distance between data points of different classes. Taking as example a binary classification problem in a 2-dimensional space (2 features) as shown in Figure 3.14, the hyperplane is a line and the two classes can be separated by an infinite number of lines.

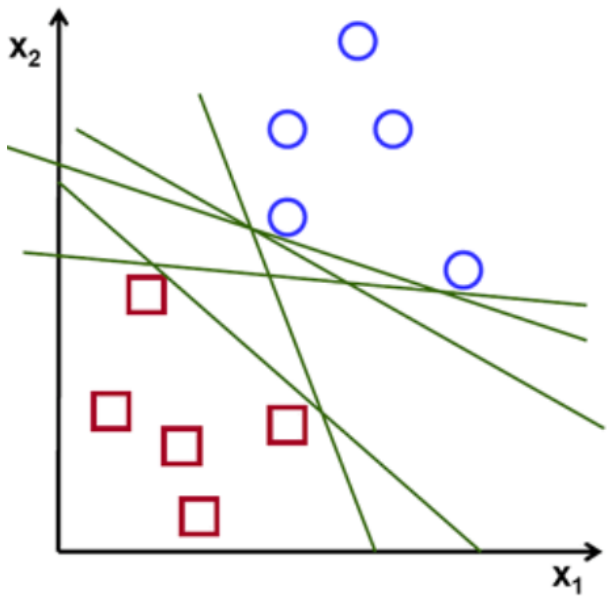


Figure 3.14: Linear hyperplane between two classes [40].

The points that are closer to the hyperplane are called support vectors because they influence the position and orientation of the hyperplane. Using the support vectors is possible to maximize the margin of the classifier. In SVC the loss function that is used to maximize the margin is the hinge loss. Figure 3.15 shows the optimal hyperplane found between two classes.

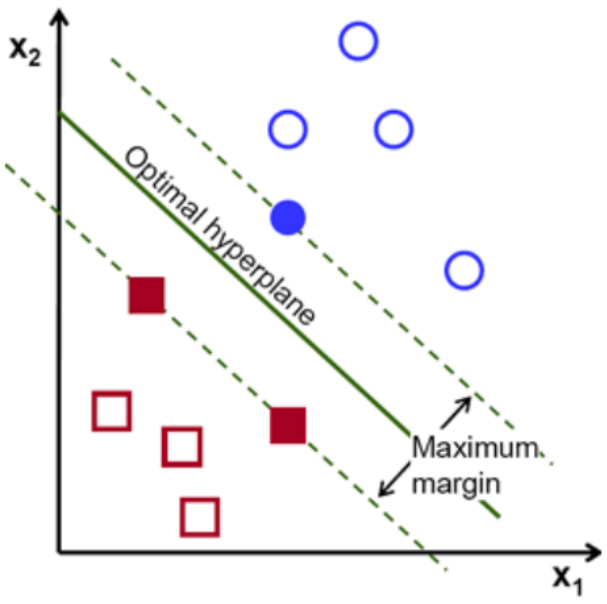


Figure 3.15: Optimal hyperplane between two classes [40].

Once the optimal the hyperplane is found, it can be used to make predictions on new, unlabeled data by determining which side of the hyperplane the data point falls on.

Mathematically speaking, the hyperplane is defined as in (3.9):

$$w^T x + b = 0 \tag{3.9}$$

where w is defined as weights, x as the observations and b as bias.

The support vectors are defined as in (3.10):

$$w^T x + b = 1 \tag{3.10}$$

The model training consists in find the optimal hyperplane that is the one that maximize the margin. To maximize the margin it is necessary to minimize $||w||$ (the Euclidean norm of the weight vector), with the condition that there are no datapoints between the support vectors. This is a constrained optimization problem that can be solved by the Lagrangian multipler method. It is also a quadratic problem, so the surface is a paraboloid, with just a single global minimum.

SVC can work also with non linearly separable data, thanks to the use of kernel functions that transform the data into a higher-dimensional space where it can be linearly separable. Some common kernel functions used in SVC are radial basis function (RBF), polynomial, and sigmoid. The type of kernel to use depends on the specific problem and data, so the kernel function is one of the hyperparameter that must be choose in SVC.

ANN (Artificial Neural Network)

Artificial Neural Network (ANN) is a this type of model is inspired by the structure and function of the human brain, particularly in how neurons are connected and communicate with each other. Figure 3.17 shows a specific architecture of Neural Network, that is Fully-Connected Neural Networks, where it is possible to distinguish the base elements that compose an ANN:

- **Neurons**: a neuron, also called perceptron, corresponds to circles and boxes in the representation and it is the basic element of the ANN. A perceptron receives in input a vector of real-valued inputs, calculates a linear combination of them, adds a baising term, then an activation function is applied to the net input to produce the output. An activation function maps any real input into a usually bounded range, often 0 to 1 or -1 to 1, and can be linear or nonlinear; common activation functions are: linear or identity, hyperbolic tangent, logistic, threshold, gaussian, etc.. In figure 3.16 a simple non linear perceptron with logistic activation function is represented:

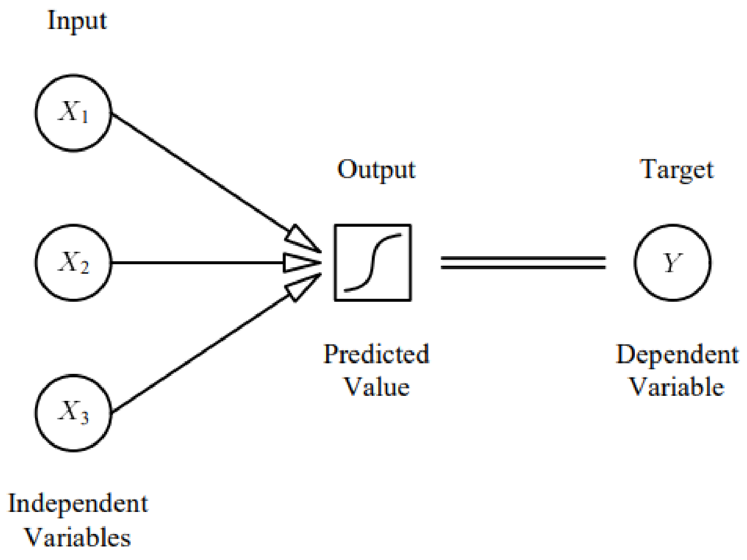


Figure 3.16: Simple non linear perceptron with logistic activation function [45].

The perceptron's output (O) can be defined as in (3.11):

$$O = f(net) = f(\bar{W} \cdot \bar{X}) = f(\sum_{j=1}^{n+1} W_j X_j) = f(\sum_{j=1}^{n} W_j X_j + \theta) \tag{3.11}$$

where $\bar{X}$ is the input vector, $\bar{W}$ is the weight vector associated to the arrows, f is the activation function and θ is the bias, that is a tunable parameter that allows the network to make predictions that are not solely dependent on the input data.

- **Layers**: a layer is a group of interconnected neurons that receive the input data, perform a computation on that input, and then produce an output. Layers are stacked on top of each other to form the overall network architecture. There are different types of layers:
  - Input Layer: it is the first layer of the network and so the one that receives the input data. Each neuron that compose this layer represents a feature or attribute of the input data. From the input layer the input data are passes the first hidden layer.
  - Hidden Layer: between the input layer and the output layer there be one or more hidden layers. Each neuron in a hidden layer receives inputs from the previous layer and applies an activation function to produce an output. The output of each neuron in a hidden layer is passed as input to the neurons in the next layer, that can be another hidden layers or the final output layer. An

ANN with two or more hidden layers is called DNN (Deep Neural Network).

  – Output Layer: it is the final layer of the network and so the one that produces the final result. Each neuron in the output layer represent one predicted values or class.

The number and size of the layers in a neural network can vary depending on the specific problem being solved and the complexity of the input data.

- **Connections**: between neurons there are connections, represented by the arrows in figure 3.17, that determine how the information flows through the network. Each connection is characterized by a weight, that represents its strength, and during the training process this weight is adjusted to improve the accuracy of the network.

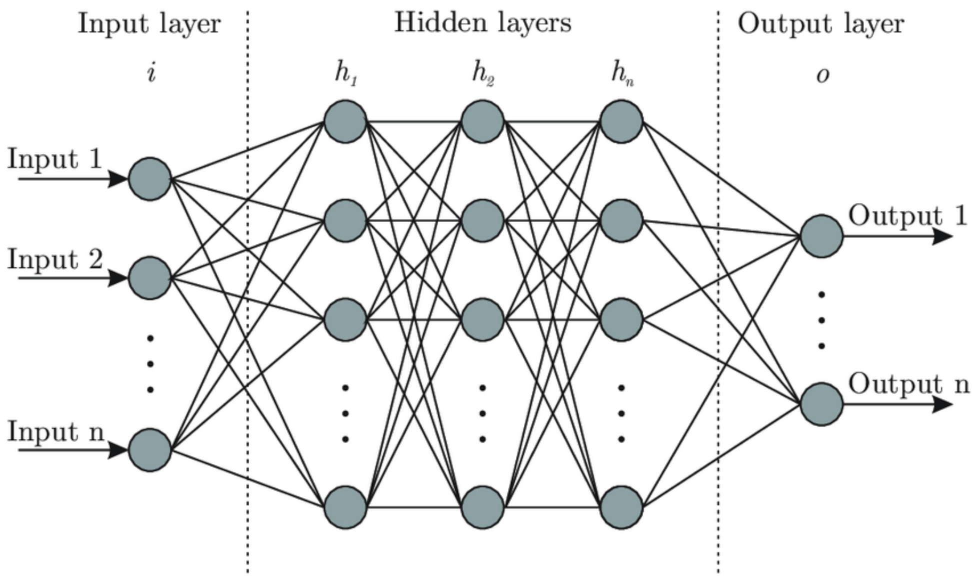


Figure 3.17: Artificial Neural Network architecture [11].

To train an ANN is necessary to define a cost function, that is a mathematical function that measures the difference between the predicted output and the real output (so the error) for the entire training set (while a loss function measures the error for a single training example), and an optimization algorithm such as gradient descent to minimize it. Depending on the type of problem it must be solved there are different cost function that can be used. For example a common cost function for binary classification problems is the binary cross-entropy, for regression problems is the mean squared error and for multi-class classification problems is the categorical cross-entropy. The training process consists in:

1. Initialize the weights and biases of the neurons in the network with random

values

2. Forward Propagation: the input data is propagated from the input layer of the network, through all the hidden layers and finally to the output layer where the predicted output is computed.

3. The network predicted output is compared to the expected output (ground truth) using the loss function.

4. Back Propagation: the error is propagated back through the network, and the weights and biases are adjusted based on the error gradient. This involves computing the derivative of the loss function with respect to each weight and bias in the network.

5. The weights and biases are updated using an optimization algorithm such as stochastic gradient descent. This involves multiplying the gradient by a learning rate and subtracting the result from the current weight or bias value.

6. Steps 2-5 are repeated for many iterations or epochs until the network converges to a good solution.

Once the network is trained, it can be used to make predictions on new, unseen data.

3.3. Hyperparameter Optimization

As introduced in the previous section, hyperparameters are all the parameters that cannot be learned by a ML model directly from the input data, but they must be set before training the model. Hyperparameter optimization is the process of selecting the best set of hyperparameters for a machine learning algorithm, that are the ones that will give the best performance of the model on the test set. This process will help to improve model performance, as it can find the best configuration for the model, which can result in better accuracy, faster convergence, and better generalization to new data. It can also help to reduce the training time by finding the best configuration for the model that requires fewer training epochs or less computational resources.

Hyperparameter optimization is typically done with the following steps:

1. Definition of hyperparameter space: this is the set of all the hyperparameters we want to search, specifying the range of values that each hyperparameter can take.

2. Chosen of a search method: is the method used to explore the hyperparameter

space. They will be described in the next.

3. Model training and evaluation: for each set of hyperparameters, the model is trained on the training set and evaluated on the validation set to obtain a performance metric, such as accuracy or mean squared error. In this phase it is also possible to used K-fold cross-validation to obtain a more accurate performance metric.

4. Selection of the best hyperparameters: the sets with best performance metric is selected to train the final model on the entire training set.

To select the best hyperparameters there are several search methods [9], the one used in this work is called *HyperOpt*. HyperOpt is based on "Bayesian Optimization" and supported by the SMBO (Sequential Model-Based Global Optimization) methodology adapted to work with different algorithms such as: Tree of Parzen Estimators (TPE), Adaptive Tree of Parzen Estimators (ATPE) and Gaussian Processes (GP). In bayesian optimization the performance of a model for a given set of hyperparameters are predicted using a probabilistic approach through a method known as surrogate optimization. A surrogate function is an approximation of the objective function (i.e., the model's performance metric) as a function of the hyperparameters. The most commonly used surrogate model is the Gaussian Process (GP). Another key component in this method is the acquisition function, a function that, based on the surrogate model's predictions, quantifies the usefulness of a set of hyperparameters. There are different types of acquisition functions all with the goal of balancing exploration and exploitation in different ways, i.e., they seek to explore the unexplored regions of the hyperparameter space (exploration) while also exploiting the promising regions (exploitation). This method can efficiently explore the hyperparameter space and converge to the optimal set of hyperparameters with fewer evaluations.

Other search methods are:

- *Manual Search*: in manual search the hyperparameters are set manually by the user based on their experience, intuition, or prior knowledge of the problem. Here there is no optimization due to the use of a search algorithm.

- *Grid Search*: in grid search all the hyperparameters defined by the users are test by the search algorithm, so the model is trained and evaluated for each combination of hyperparameters in the search space.

- *Random Search*: in random search, instead of evaluating all the combination of hyperparameters defined by the users, a random combinations of them are used to find the best solution.

- *Genetic Algorithms*: in the genetics algorithm there are chromosomes encoding the hyperparameters, and a population of chromosomes is iteratively evolved through selection, mutation, and crossover operations. The fitness function is the performance of the model for a given set of hyperparameters.

The choice of search method depends on the size of the hyperparameter space, the computational resources available, and the complexity of the model.

For each classification algorithm introduced in Section 3.2.2, an hyperparameter space has been defined, as presented in the following.

3.3.1. XGBoost

Hyperparameter search for XGBoost model focuses on the following hyperparameters [56]:

- *learning_rate*: also known as *eta*, is the step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and *eta* shrinks the feature weights to make the boosting process more conservative.
- *max_depth*: is the maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. Beware that XGBoost aggressively consumes memory when training a deep tree. The default value is 6, the range is $[0, \infty]$ and 0 indicates no limit on depth.
- *n_estimators*: is the number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance. The default value is 100, the range is $[1, \infty]$.
- *subsample*: is the subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees and this will prevent overfitting. Subsampling will occur once in every boosting iteration. The default value is 1, the range is $(0, 1]$.
- *gamma*: also known as min_split_loss, is the minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be. The default value is 0, the range is $[0, \infty]$.

3.3.2. SVC

Hyperparameter search for Support Vector Classifier model focuses on the following hyperparameters [46]:

- *kernel*: specifies the kernel type to be used in the algorithm. If none is given, 'rbf' (radial basis function) will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).
- *gamma*: is the kernel coefficient for 'rbf', 'poly' (polynomial) and 'sigmoid'. If gamma = 'scale' (default) is passed then it uses 1/(n_features * X.var()) as value of gamma, if gamma='auto' is passed then it uses 1/n_features.
- *C*: is the regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

3.3.3. ANN

Hyperparameter search for Artificial Neural Network model focuses on the following hyperparameters [27], [28], [29], [30]:

- *activation*: is the node function used to produce its output, and can be linear or non-liner.
- *layers*: are the internal (hidden) layers of multilayer networks, the ones between the input layer and output layer. They learn to represent intermediate features that are useful for learning the target function and that are only implicit in the network inputs.
- *ratio_input*: this parameter is used to establish how many neurons are there in the first layer.
- *epochs*: is the number of epochs to train the model. An epoch is an iteration over the entire X and y data provided.
- *batch*: is the number of samples per gradient update. The default value for the learning rate is 32.
- *learning_rate*: for this work we used as optimizer Adam. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The default value for the learning rate is 0.001.
- *dropout_rate*: is the fraction of the input units to drop in from the Dropout layer. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

- *l2_value*: is a layer weight regularizers that allow to apply penalties on layer parameters or layer activity during optimization. These penalties are summed into the loss function that the network optimizes. The L2 regularization penalty is computed as: loss = l2 * reduce_sum(square(x)). The default value for the learning rate is 0.01.

3.4. Methodologies to address Data Scarcity

When there is an insufficient amount of data available to train a machine learning model we are facing data scarcity. There are several methodologies to address this problem, we report here the ones used for this work.

3.4.1. Data Augmentation

Class imbalance occurs when one class in a dataset has significantly fewer samples than the other classes. The first step in handling class imbalance is to compute the imbalance ratio I.R [44], [47], as seen in the following equation (3.12):

$$I.R. = \frac{\#(S_{maj})}{\#(S_{min})} \tag{3.12}$$

where $\#(S_{\text{maj}})$ represents the frequency of samples in the majority class and $\#(S_{\text{min}})$ represents the frequency of samples in the minority class. The goal is to resample the classes such that there is equibalance between the majority and minority classes with I.R.= 1.

Depending on the imbalance ratio I.R, two different alternatives can be adopted to achieve balance:

- if I.R. is low, then undersampling can be performed by reducing the number of samples of the major class.
- if I.R. is high, then undersampling should be avoided as there will be a loss in data when a large number of majority class samples are removed. Instead, oversampling deals with synthesizing samples of the minority class in order to match the number of samples of the majority class.

Since we deal with a dataset characterized by an high I.R. values, we will adopt an oversampling technique named *Synthetic Minority Over-sampling TEchnique (SMOTE)* [14]. It is a data augmentation technique used in machine learning to address the problem of class imbalance. Data augmentation is the process of increasing training dataset size and diversity generating new data samples from existing data. The aim is to improve the performance and generalization of machine learning models by exposing them to more diverse examples.

SMOTE works by creating synthetic samples from the minority class, that is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen. Synthetic samples are generated in the following way, as in equation (3.13): compute the difference between the feature vector (sample) under consideration (x_{i}) and its nearest neighbor (x_{zi}), multiply this difference by a random number between 0 and 1 (λ), and add it to the feature vector under consideration. This process is repeated for identified feature vector.

$$x_{new} = x_i + \lambda * (x_{zi} - x_i) \tag{3.13}$$

This causes the selection of a random point along the line segment between two specific features, as shown in Figure 3.18. This approach effectively forces the decision region of the minority class to become more general.

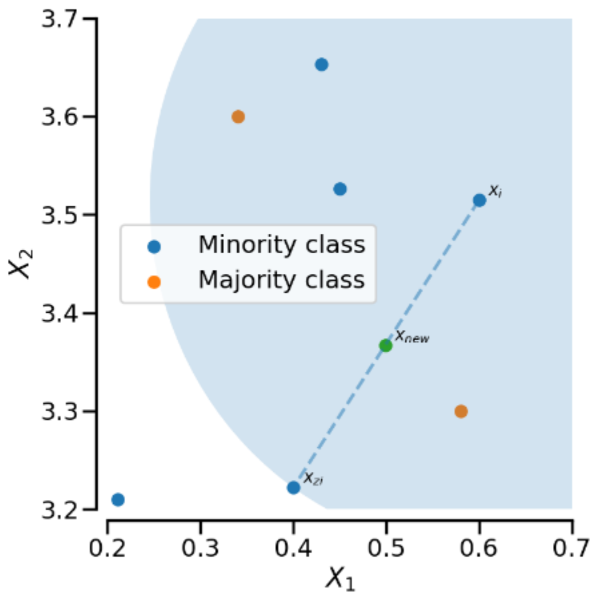


Figure 3.18: Over-sampling generation [24].

By creating new synthetic samples, SMOTE effectively increases the size of the minority class and helps to balance the class distribution. This can improve the performance of machine learning models by reducing bias towards the majority class and increasing the accuracy of the model on the minority class. However, it is possible to customize SMOTE to generate synthetic data over user-defined classes. It is important to note that SMOTE should only be used on training data and not on the test data, as it can lead to overfitting.

3.4.2. Transfer Learning

The concept of transfer learning was introduced to the field of educational psychology by psychologist C.H.Judd, who argued that transfer learning is the result of generalizing experience gained on one activity to another activity. For example, a person who has experience playing the piano will learn faster to play another instrument, such as the guitar, than a person who has never played an instrument, as both are musical instruments that share some common knowledge. An important requirement is that there be a connection between two learning activities, otherwise this process can be unsuccessful. For example, a person who can play an instrument cannot use this knowledge of his or her to learn to drive a car [59].

The concept of knowledge transfer is also applied in the field of machine learning where, unlike in traditional machine learning models which are trained from scratch, here the knowledge acquired by a model on one or more source tasks is transferred to a target task, as shown in Figure 3.19.

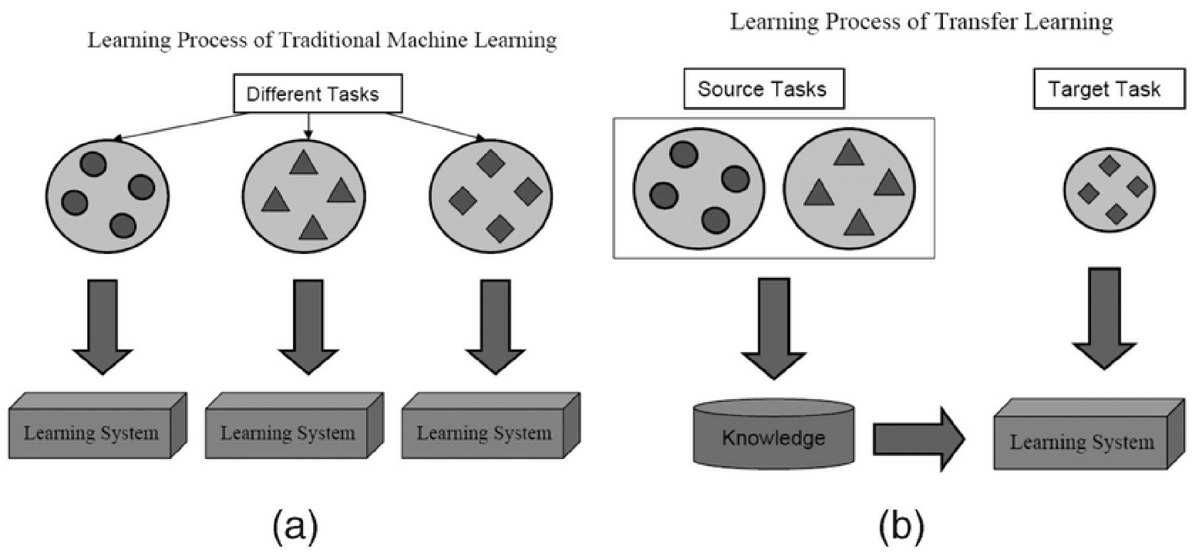


Figure 3.19: Different learning processes between (a) traditional machine learning and (b) transfer learning [42].

This methodology, called *Transfer Learning* (TL), finds wide application in domains where having insufficient data for training is an inescapable situation. Therefore, this methodology can be used to solve the problem of data scarcity by transferring knowledge gained from a model in a source domain, where a lot of labelled data are available, to a target domain, where little training data are available, relaxing the assumption that training data and test data must be independent and identically distributed (i.i.d).

To define formally TL, we need to introduce the concept of domain and task [59]:

- A domain D is composed of two parts, that are a feature space X and a marginal distribution P(X), where X denotes an instance set, which is defined as X = $\{\text{x}|x_\text{i} \in X, \text{i} = 1, \ldots, \text{n}\}$. In other words, $D = \{X, \text{P(X)}\}$.
- A task T consists of a label space Y and a decision function f, that is, $T = \{Y, f\}$. The decision function f is an implicit one, which is expected to be learned from the sample data.

Formally TL is defined as: given some/an observation (s) corresponding to $m^S \in \text{N}^+$ source domain(s) and task(s), and some/an observation(s) about $m^T \in \text{N}^+$ target domain(s) and task(s), transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions on the target domain(s).

Pan et al. in their survey on TL [42] categorize TL under three subsettings, as shown in Figure 3.20:

- *Inductive transfer learning*: in this setting the target task is different from the source task, regardless of whether the source and target domains are the same or not. Here, is required to have some labeled data in the target domain to induce an objective predictive function to use with target data. Depending on the presence or absence of labeled data in the source domain, we can further categorize the inductive transfer learning setting into two cases:
  - if a lot of labeled data in the source domain are available, then the inductive transfer learning setting is similar to the multitask learning setting, where multiple learning tasks are solved at the same time.
  - if no labeled data in the source domain are available, then the inductive transfer learning setting is similar to the self-taught learning setting, where semi-supervised learning is combined with TL.
- *Transductive transfer learning*: in this setting the source and target tasks are the same, while the source and target domains are different. Here, no labeled data in

the target domain are available while a lot of labeled data in the source domain are available.

- *Unsupervised transfer learning*: in this setting the target task is different from but related to the source task. Here, there are no labeled data available in both source and target domains in training. The scope is to solve unsupervised learning tasks in the target domain, such as clustering, dimensionality reduction, and density estimation.

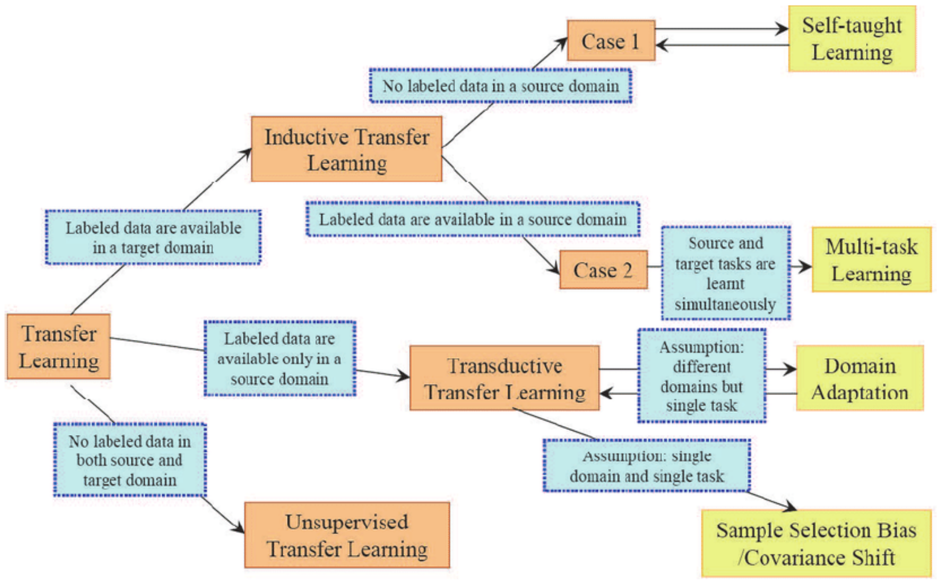

Case 1
Self-taught Learning
No labeled data in a source domain
Inductive Transfer Learning
Labeled data are available in a target domain
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Transfer Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
No labeled data in both source and target domain
Assumption: single domain and single task
Unsupervised Transfer Learning
Sample Selection Bias /Covariance Shift

Figure 3.20: Different settings of transfer [42].

By their natures, artificial neural networks are well suited for transfer learning. In [51], Tan et al. identify four different approaches to implementing transfer learning with deep neural networks that are:

- *Instances-based deep transfer learning*: here partial instances from the source domain are selected, weighted and assigned as supplements to the training set in the target domain.
- *Mapping-based deep transfer learning*: here the instances from the source domain and target domain are mapped into a new data space on which the deep neural network is trained.
- *Network-based deep transfer learning*: here is reused partial network that pre-trained in the source domain, including its network structure and connection parameters, transfer it to be a part of deep neural network which used in target domain, as in

Figure 3.21.

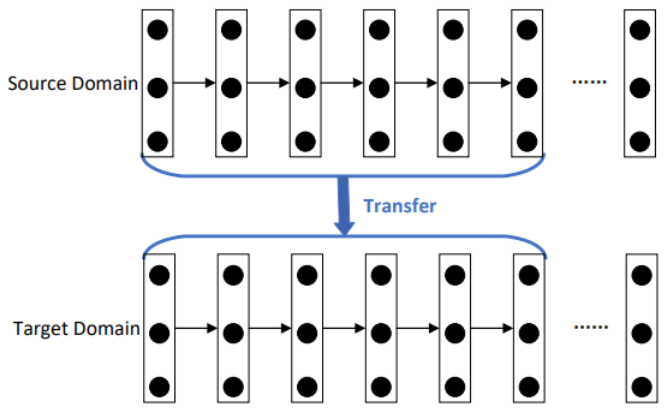


Figure 3.21: Network-based deep transfer learning schema [51].

This is the approach that we will used in Section 4.4.2, to address the problem of data scarcity.

- *Adversarial-based deep transfer learning*: here adversarial technology inspired by generative adversarial nets (GAN) are introduced to find transferable representations that is applicable to both the source domain and the target domain.

3.4.3. Auxiliary-Tasks Learning

Auxiliary-Tasks Learning was firstly introduced by Liebel and Korner in [34], as an extension of Multi-task Learning. The goal of Multi-Task Learning is to learn several outputs from a single input simultaneously, finding a common representation in the earlier layers of the network and completing the individual jobs in their own single-task branches at the later phases of the network, as in Figure 3.22. This is typically implemented as an encoder-decoder structure, where each atomic task represents a specialized decoder to the representation provided by the common encoder. While each type of label favors the learning of certain features in the common part, some of them can be exploited by other tasks as well. This structure can thus help to boost the performance of the atomic tasks.

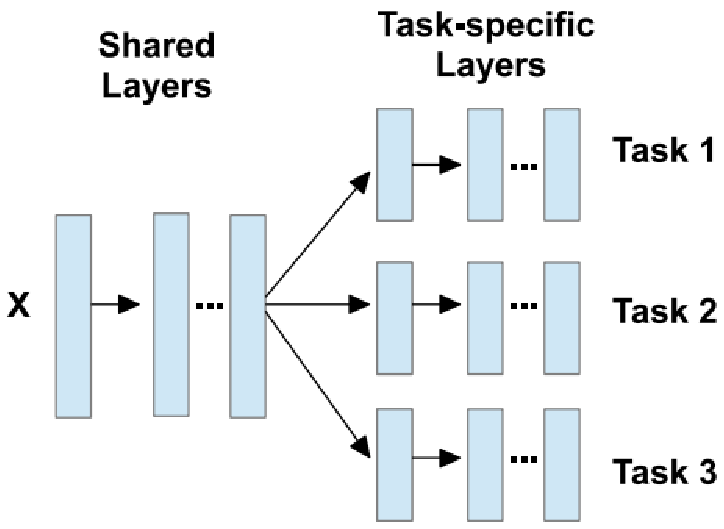


Figure 3.22: Multi-Task Learning Network structure with 3 main-task [53].

In Auxiliary-Task Learning, the main-tasks, i.e., the single-task producing the required output, are flanked by the auxiliary-task, that are easy to learn tasks of minor importance. The main-tasks and the auxiliary-tasks loss function are then combined into a final multi-task loss function, that allow to benefit from both contributions, as shown in Figure 3.23. The final loss function is a weighted sum of the losses for each task, where the weights determine the relative importance of each task. During training, the model updates its parameters to minimize this overall loss.

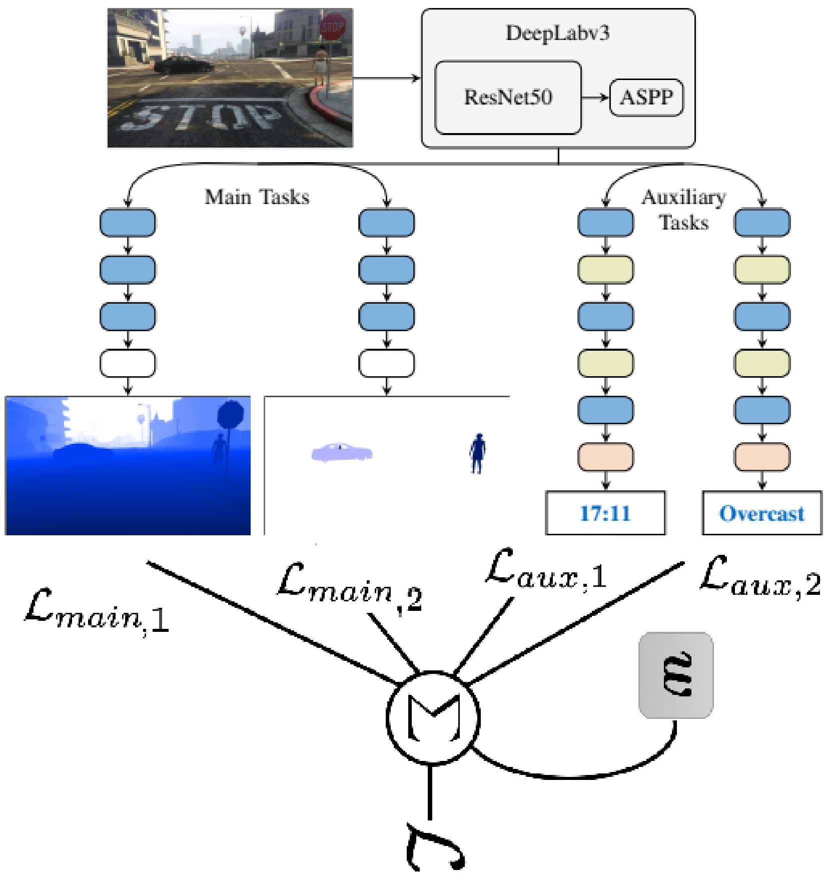


Figure 3.23: Auxiliary-Task in Multitask-Learning with 2 main-task and 2 auxiliary-task.

There are two main benefits from training the model on multiple tasks:

1. The model is forced to learn more generalizable characteristics that can be used on other tasks and new data. This can lessen the possibility of overfitting and enhance the model's generalizability.
2. The model can learn more from the same amount of data. This is particularly useful in applications where data is scarce or expensive to obtain. This is the reason why we will adopt this technique in Section 4.4.3.

3.4.4. Denoising Autoencoders

To understand what a denoising autoencoders is, we must first introduce the concept of an autoencoder. An autoencoder (AE) is a self-supervised learning model trained to encode the input x into some representation c(x) so that the input can be reconstructed from that representation. Hence the target output of the auto-encoder is the auto-encoder input itself. Autoencoders present an efficient way to learn a representation of the input data, which helps with tasks such as dimensionality reduction or feature extraction.

A denoising autoencoder (DAE) [54] is an AE that receives as input corrupted data by adding noise. There are several ways to add noise to the input data in a DEA, one of them is to chose for each input a fixed random number of components and force their value to 0, while the others are left untouched. Another approach can be, in case of binary features, to flip the values for a fixed random number of components of each input, while the others are left untouched. The aim is to reconstruct the original data removing the noise. In order to do so, the latent space of this self-supervised method must learn not only a compressed representation of the features but also the structural relationships within them. Figure 3.24 shows an illustration of denoising autoencoder model architecture.

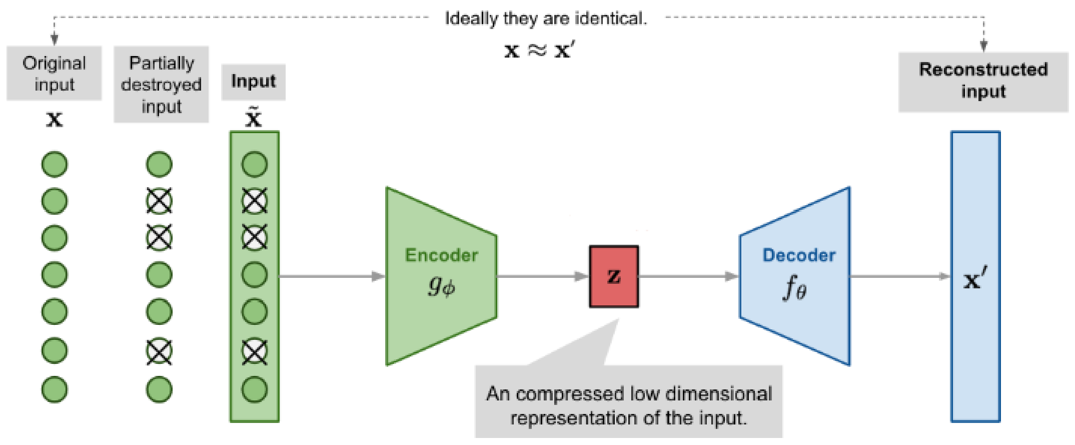

Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Original input
$\mathbf{x}$
Partially destroyed input
Input
$\tilde{\mathbf{x}}$
Encoder
g_ϕ
$\mathbf{z}$
Decoder
f_θ
Reconstructed input
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 3.24: Denoising autoencoder model architecture.

Denoising autoencoders (DEAs) can help address data scarcity by leveraging unsupervised learning, which does not require labeled data, to learn to extract useful features or representations of the data, which can be used for downstream tasks. This is the reason why we will adopt this methodology in Section 4.4.4.

4 | Failure-Cause Identification in Microwave Networks

In this chapter we describe failure-cause identification in microwave networks. The chapter is organized as follows: we first describe the dataset in Section 4.1, then we discuss the data preprocessing phase in Section 4.2. We describe the baseline machine learning models used in this work in Section 4.3, and finally we detail the methodologies used to address data scarcity in Section 4.4.

4.1. Dataset Description

In this work, we make use of data provided by SIAE Microelettronica [38], an Italian company that manufactures telecommunications equipment and specializes in microwave networks. This dataset is a collection of hardware failures states taken from a real microwave network, where devices alarms status is monitored by a Network Management System (NMS) with 15-minutes intervals. During the entire monitored period, 1051 hardware failures have been recorded considering windows of 15 minutes (900 seconds), featured by the information of all alarms states in the windows. To perform data labelling, these failures events have been analysed by domain experts and labelled in 4 macro-categories, i.e., the ones in Table 4.1, and 22 micro-categories of failures, as presented in Table 4.2. Due to confidentiality reasons, we do not report the description of the specific failures and alarms, and instead provide a masked representation.

MACRO-CATEGORY	**DATA POINTS**
0	402
1	314
2	70
3	265

Table 4.1: Macro-categories of hardware failures and number of data points in each macro-category.

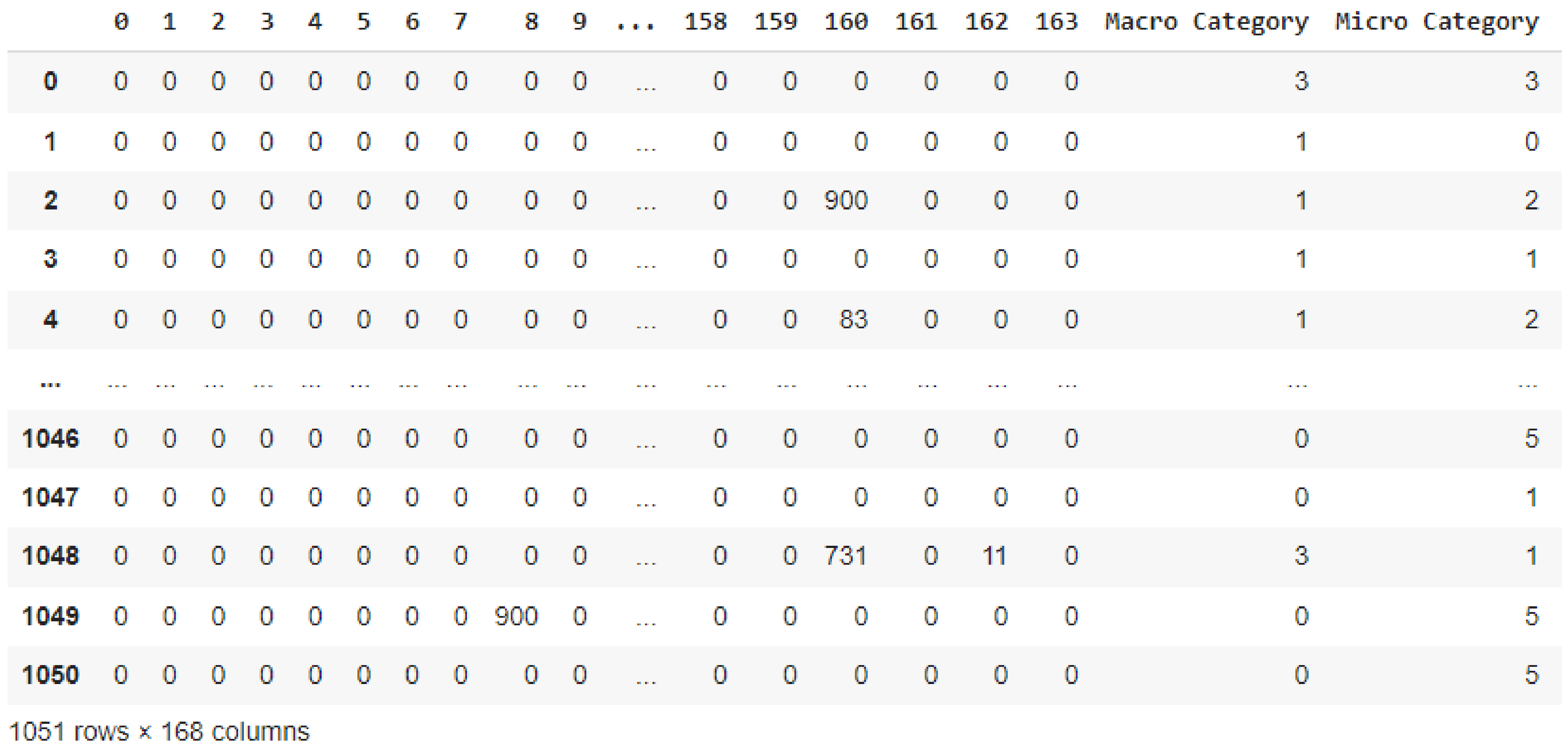
MACRO CATEGORY	MICRO CATEGORY	DATA POINTS
0	0	13
0	1	7
0	2	21
0	3	8
0	4	264
0	5	89
1	0	87
1	1	6
1	2	130
1	3	91
2	0	29
2	1	20
2	2	2
2	3	15
2	4	2
2	5	1
2	6	0
2	7	1
3	0	43
3	1	39
3	2	150
3	3	33

Table 4.2: Micro-categories of hardware failures and number of data points in each micro-category.

Figure 4.1 shows a description of the dataset used in this thesis: it consists of 1051 rows (i.e., 1051 different failure events) each characterized by the following information, distributed across 168 columns of the dataset[1]:

[1]Note that, due to confidentiality reasons, we provide a synthetic, non-detailed description of the used alarms.

- an identifier of the link where the failure has occurred.
- the failure date and time.
- 164 alarms raised on the network components used in the failed link. For each alarm, a value ranging from 0 (alarm not triggered) to 900 (alarms triggered for 900 seconds) is provided, representing the number of seconds the alarm was on during a 15-minutes window.
- two labels corresponding to the failure macro-category and micro-category, respectively.



	0	1	2	3	4	5	6	7	8	9	...	158	159	160	161	162	163	Macro Category	Micro Category
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	3	3
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	900	0	0	0	1	2
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	1
4	0	0	0	0	0	0	0	0	0	0	...	0	0	83	0	0	0	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1046	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	5
1047	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1
1048	0	0	0	0	0	0	0	0	0	0	...	0	0	731	0	11	0	3	1
1049	0	0	0	0	0	0	0	0	900	0	...	0	0	0	0	0	0	0	5
1050	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	5

1051 rows × 168 columns

Figure 4.1: Portion of raw dataset. All the sensitive information like alarms name, links name and timestamps have been removed due to confidentiality reasons.

4.2. Data Preprocessing

As described in Chapter 3.1, once the data collection phase is over, the dataset must be prepared to be suitable for the machine learning algorithms. During this phase the following actions were carried out on the raw dataset:

- Link ID and failure date/time information have been removed because they do not provide useful information for the objective of this thesis work.
- All columns containing alarms always set to 0 have been removed, because they do not carry relevant information for the classification task.
- All rows belonging to a class with a number of occurrences (number of hardware failures) less than 6 have been removed. This dataset truncation has been performed in order to have at least one occurrence in each fold when applying stratified K-Fold Cross-Validation with k=5.
- Information regarding the macro and micro category into which each failure is classified has been merged into a new category that encompasses them both (Table 4.3).
- The features (X), i.e., the alarms, have been split from the label (y), i.e., the classes of failure.
- All features have been converted from [0,900] to binary features with the following meaning:
  - **0**: alarm not triggered over the entire 15-minutes window.
  - **1**: alarm triggered for at least one second over the entire 15-minutes window.

MACRO CATEGORY	MICRO CATEGORY	CATEGORY	DATA POINTS
0	0	**0**	13
0	1	**1**	7
0	2	**2**	21
0	3	**3**	8
0	4	**4**	264
0	5	**5**	89
1	0	**6**	87
1	1	**7**	6
1	2	**8**	130
1	3	**9**	91
2	0	**10**	29
2	1	**11**	20
2	3	**12**	15
3	0	**13**	43
3	1	**14**	39
3	2	**15**	150
3	3	**16**	33

Table 4.3: Mapping between "MACRO CATEGORY" and "MICRO CATEGORY" into the new "CATEGORY" label obtained after data preprocessing. Column "CATEGORY" reports all new 17 labels used for the classification task and for each label we show the number of data points in the dataset in the column "DATA POINTS".

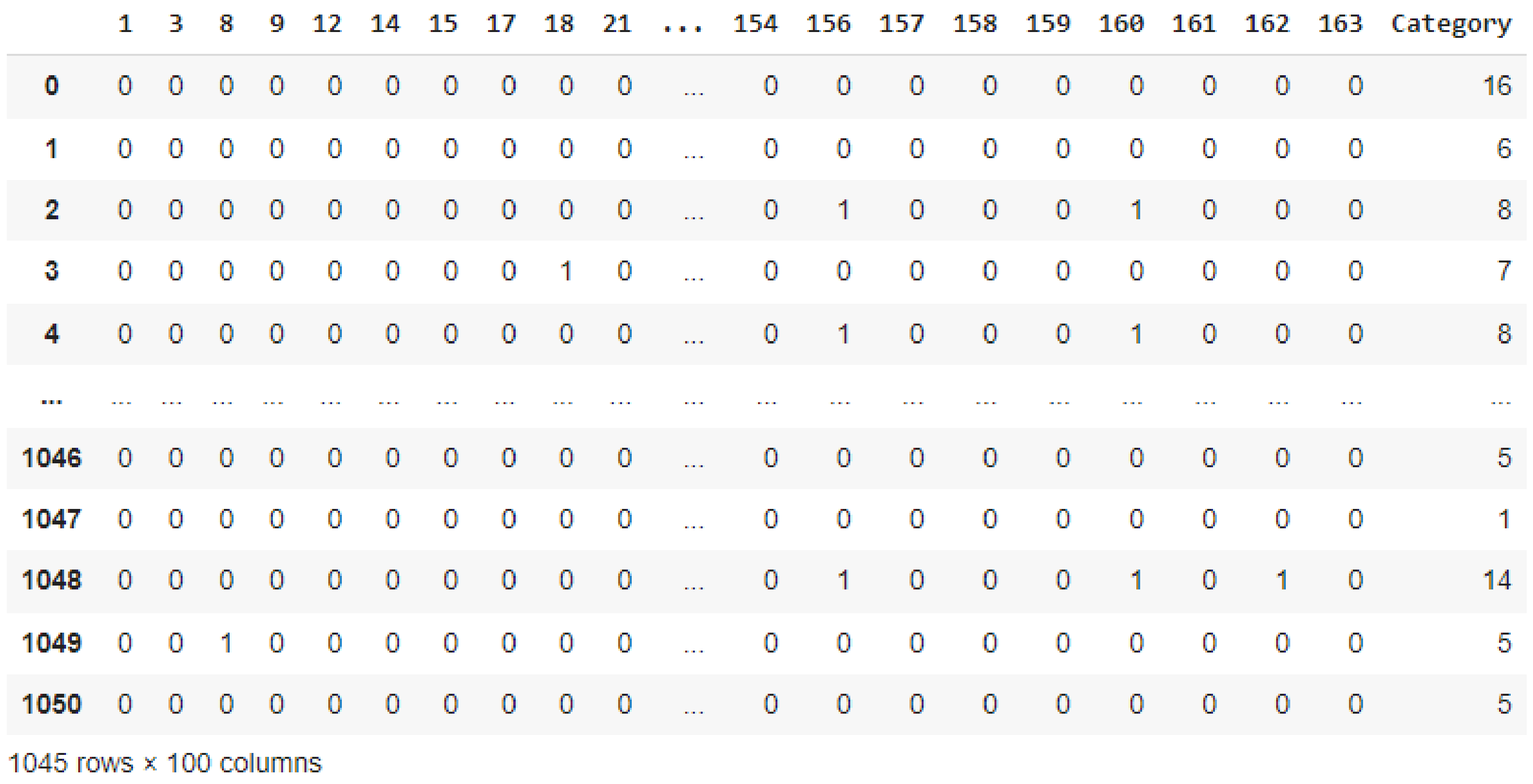
Figure 4.2 shows the resulting dataset after the preprocessing phase: it consists of 1045 rows (i.e., 1045 different failure events) each characterized by the following information, distributed across 100 columns of the dataset[2]:

- 99 alarms raised on the network components used in the failed link. For each alarm, a value of 0 (alarm not triggered) or 1 (alarms triggered) is provided. The values of these columns along the 1045 rows are the features "X".
- one label corresponding to the failure category. The values of this column along the

[2]Note that, due to confidentiality reasons, we provide a synthetic, non-detailed description of the used alarms.

1045 rows represent the ground-truth label "y".

The goal of this thesis is to develop ML models able to correctly classify a hardware failure in microwave networks in one of these 17 categories (see Table 4.3). As can be observed from the column "DATA POINTS" of this table, the dataset is affected by data scarcity, since most categories have very few data points compared to the classes with highest numbers of data points.



	1	3	8	9	12	14	15	17	18	21	...	154	156	157	158	159	160	161	162	163	Category
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	16
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	6
2	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	1	0	0	0	8
3	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	0	0	0	7
4	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	1	0	0	0	8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1046	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	5
1047	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
1048	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	1	0	1	0	14
1049	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	5
1050	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	5

1045 rows × 100 columns

Figure 4.2: Portion of the processed dataset. All the sensitive information like alarms name have been removed due to confidentiality reasons..

4.3. Baseline Machine Learning Models

In this section we present the baseline ML models used in this thesis, and described in Chapter 3.2.2, i.e., *eXtreme Gradient Boosting* (XGBoost), *Support Vector Classifier* (SVC) and *Artificial Neural Network* (ANN). These models are used to determine the baseline performance, i.e., Accuracy, Precision, Recall and F1-score, that we aim to enhance using the methodologies presented in Chapter 4.4. Figure 4.3 shows the pipeline we used to develop these models, including the following main building blocks to perform failure-cause identification:

1. **BLK-1**: we split the dataset (Figure 4.2) in 5 folds using *Stratified k-fold Cross-Validation* to validate the aforementioned performance metrics.
2. **BLK-2**: we used the train set for the hyperparameters optimization; each hyperparameter set chosen is cross-validated using *Stratified k-fold Cross-Validation* to optimize the chosen performance metric that is the accuracy.
3. **BLK-3**: we tested the output models (referred to as *best models* in the figure), i.e, the ones trained on the train set with the best set of hyperparameter found, on the test set. We then stored the performance metrics obtained on this split.
4. **BLK-4**: at the end of the five iterations, we computed the average performance metrics for each model.

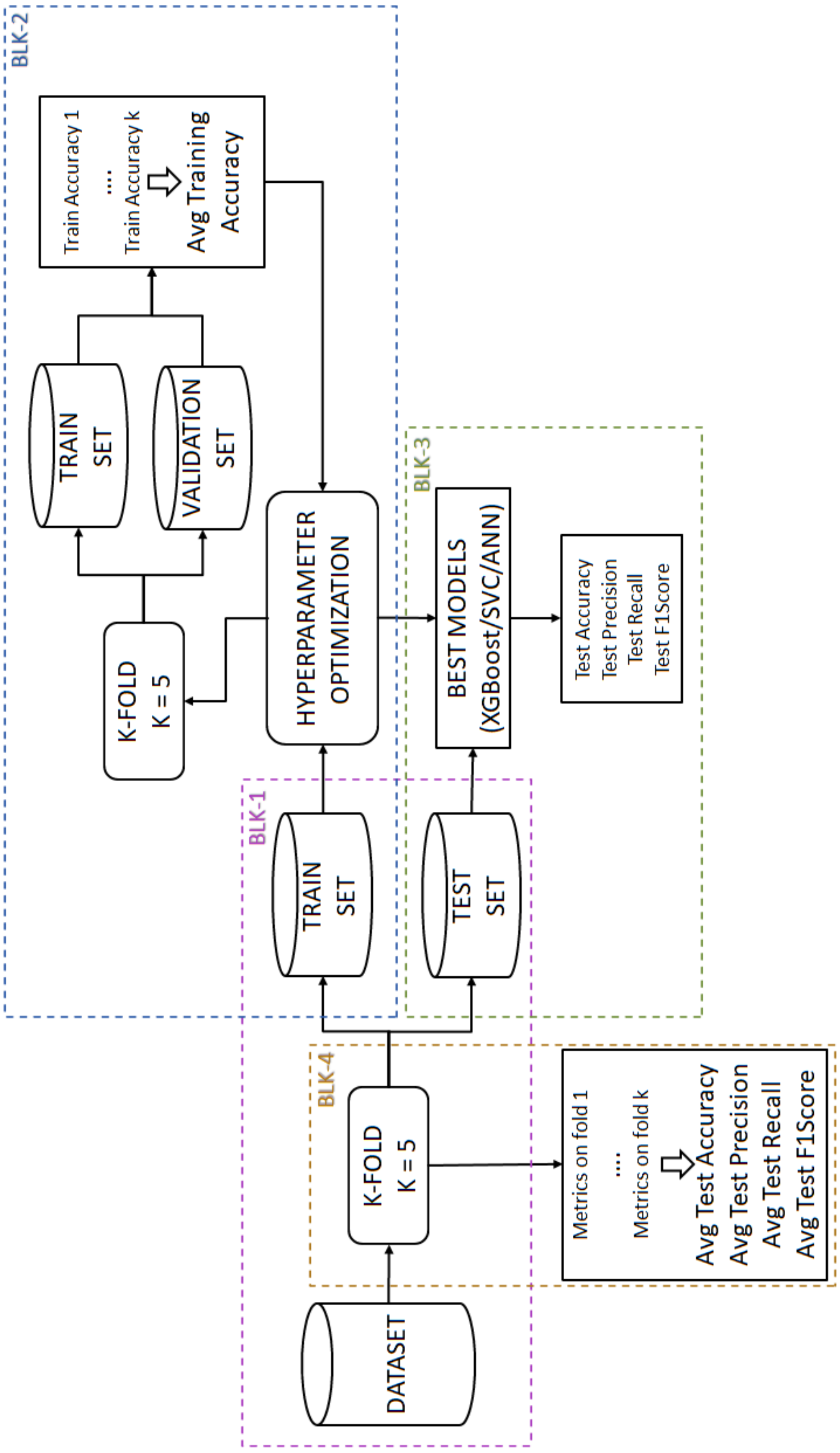


Figure 4.3: Baseline Models pipeline with hyperparameters optimization.

For each model, an hyperparameter space has been defined as presented in Chapter 3.3. We used the described pipeline in Figure 4.3 to train the baseline models, obtaining for each of the five training set splits an optimal set of hyperparameters, as shown in Table 4.4 for XGBoost, in Table 4.5 for SVC, and in Table 4.6 for ANN.

HYPER-PARAMETER	SEARCH SPACE	SET 1	SET 2	SET 3	SET 4	SET 5
learning_rate	[1, 0.9, 0.7, 0.5, 0.3, 0.1, 0.05]	0.05	1	0.1	**0.5**	0.9
max_depth	[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]	**16**	4	6	**16**	8
n_estimators	[50, 100, 150, 200, 250]	**200**	150	**200**	150	250
subsample	[0.1, 0.3, 0.5, 0.7, 0.9, 1]	0.9	0.7	**1**	**1**	**1**
gamma	[0, 0.01, 0.03, 0.05]	0.03	**0**	**0**	**0**	0.03
Accuracy		0.93	0.92	0.93	0.93	0.94

Table 4.4: Five sets of hyperparameters obtained for XGBoost classifier on the 5 training splits.

HYPER-PARAMETER	SEARCH SPACE	SET 1	SET 2	SET 3	SET 4	SET 5
kernel	['linear', 'poly', 'sigmoid', 'rbf']	**sigmoid**	linear	linear	rbf	**sigmoid**
gamma	['auto', 'scale']	**auto**	scale	scale	**auto**	**auto**
C	[1, 10, 1000, 10000]	10000	1	10000	**1000**	**1000**
Accuracy		0.93	0.93	0.93	0.93	0.95

Table 4.5: Five sets of hyperparameters obtained for SVC classifier on the 5 training splits.

HYPER-PARAMETER	SEARCH SPACE	SET 1	SET 2	SET 3	SET 4	SET 5
activation	['leaky_relu', 'elu', 'gelu', 'relu']	elu	**relu**	gelu	**relu**	leaky_relu
layers	[1, 2, 3]	**1**	2	**1**	3	**1**
ratio_input	[0.25, 0.5, 1, 1.25]	0.25	**1.25**	0.25	1	**1.25**
epochs	[10, 25, 40, 55, 70, 95]	55	**40**	**40**	70	95
batch	[8, 16, 32, 64]	64	16	8	**32**	**32**
learning_rate	[0.1, 0.05, 0.01, 0.001]	0.5	**0.01**	0.001	0.001	**0.01**
dropout_rate	[0.1, 0.15, 0.20]	**0.1**	**0.1**	0.2	0.15	**0.1**
l2_value	[1e-4, 1e-3]	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**
Accuracy		0.85	0.86	0.86	0.85	0.87

Table 4.6: Five sets of hyperparameters obtained for ANN classifier on the 5 training splits.

The average results that we obtain considering for each baseline model the best sets of hyperparameters on the 5 training splits are presented in Chapter 5.1.

For each baseline model we analyzed the obtained five sets of hyperparameters to extract from them the best set of hyperparameters that will be used as optimized hyperparameters for the models in *SMOTE Analysis* presented in Chapter 5.4. The criteria used to select the value of each hyperparameter to include in the best set is the following:

- if one of the values tested is predominant, i.e., it is present in the most of the sets, that is the value we selected. An example is the max_depth value or subsample value chosen for XGBoost, as shown in Table 4.4
- if we have two predominant value pairs, we select the one that is associated with a higher accuracy value. An example is the n_estimators value chosen for XGBoost, as shown in Table 4.4.
- if no predominant value is present, we selected the median value. An example is the learning_rate value chosen for XGBoost, as shown in Table 4.4. In case of

categorical hyperparameters, e.g., activation functions, we select the one that is associated with a higher accuracy value.

Following this criteria, the resulting best baseline models are the following:

HYPER-PARAMETER	DEFAULT VALUE	OPTIMIZED VALUE
learning_rate	0.3	0.5
max_depth	6	16
n_estimators	100	200
subsample	1	1
gamma	0	0

Table 4.7: Best hyperparameters selected for XGBoost model.

HYPER-PARAMETER	DEFAULT VALUE	OPTIMIZED VALUE
kernel	rbf	sigmoid
gamma	scale	auto
C	1	1000

Table 4.8: Best hyperparameters selected for SVC model.

HYPER-PARAMETER	DEFAULT VALUE	OPTIMIZED VALUE
activation	-	relu
layers	-	1
ratio_input	-	1.25
epochs	-	40
batch	32	32
learning_rate	0.001	0.01
dropout_rate	-	0.1
l2_value	0.01	0.001

Table 4.9: Best hyperparameters selected for ANN model.

4.4. Machine Learning methodologies to address Data Scarcity

In this section we show the methodologies used to address data scarcity. We will present *Synthetic Minority Over-sampling Technique*, *Transfer Learning*, *Auxiliary-Task Learning*, and *Denoising Autoencoders*.

4.4.1. Synthetic Minority Over-sampling TEchnique

In this section we apply *Synthetic Minority Over-sampling Technique*, in shorten *SMOTE*, that is the data augmentation methodology presented in Chapter 3.4.1, to our classification problem. With this methodology our goal is to address the data scarcity problem enriching the preprocessed dataset with synthetic data generated by SMOTE. We used this enriched dataset to train the *Baseline Models*, i.e., XGBoost, SVC and ANN.

Figure 4.4 illustrates the proposed pipeline for this methodology:

1. **BLK-1**: we split the dataset (Figure 4.2) in 5 folds using *Stratified k-fold Cross-Validation* to validate the aforementioned performance metrics.
2. **BLK-2**: on the train set we applied SMOTE, obtaining a new train set composed by real data and synthetic data.
3. **BLK-3**: we used the synthetic train set for the hyperparameters optimization; each hyperparameter set chosen is cross-validated using *Stratified k-fold Cross-Validation* to optimize the chosen performance metric that is the accuracy.
4. **BLK-4**: we tested the *Best SMOTE Models*, i.e., XGBoost, SVC and ANN trained on the *Synthetic Train Set* with best set of hyperparameter, on the test set. We stored the performance metrics obtained on this split.
5. **BLK-5**: at the end of the five iterations, we computed the average performance metrics for each model.

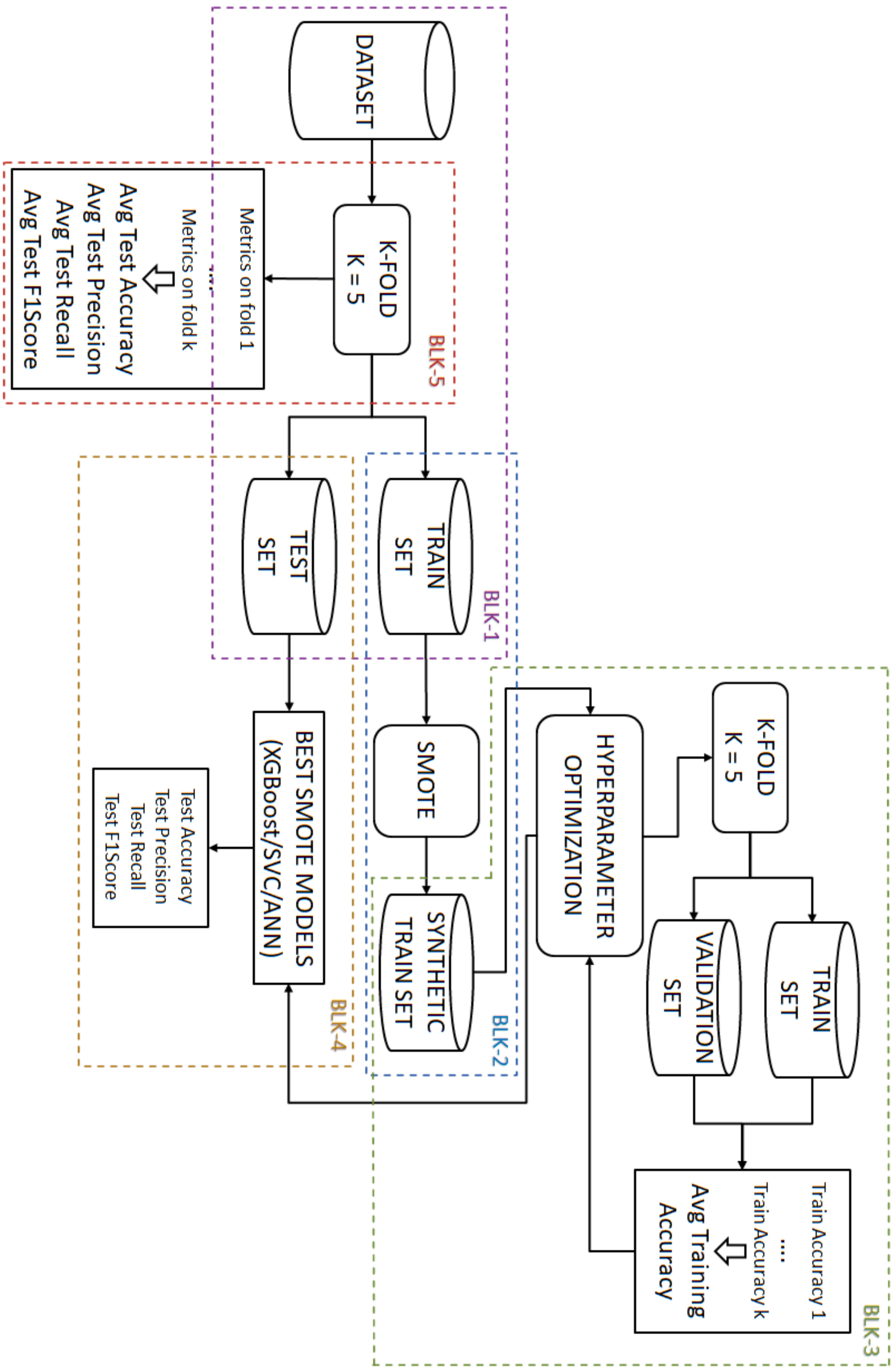


Figure 4.4: SMOTE pipeline.

4.4.2. Transfer Learning

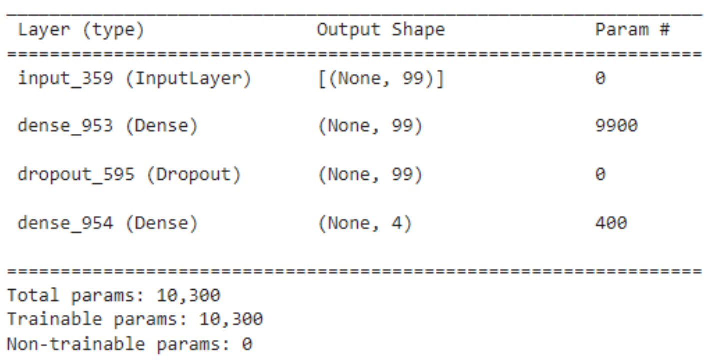
In this section we apply the *Transfer Learning* methodology presented in Chapter 3.4.2 to our classification problem. The goal of applying Transfer Learning is to address data scarcity by first training an ANN model to classify macro-categories and then re-training a TL model derived from it, able to classify the 17 defined categories of hardware failures. We expect that the previous knowledge on the macro-categories classification, will help the TL model in the 17 defined categories classification.

Figure 4.7 illustrates the proposed pipeline for this methodology:

1. **BLK-1**: to train the TL model, first we needed to train an ANN model for the macro classification and then from it we derived the TL one for the micro classification. This required adding the original column "Macro Category" to the dataset presented in Figure 4.2. We split this dataset in 5 folds using *Stratified k-fold Cross-Validation* to validate the chosen performance metrics for the models, that are: accuracy, precision, recall and F1-score.

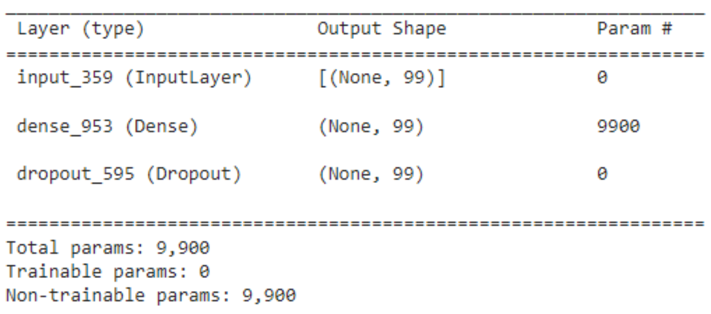
2. **BLK-2**: we used the train set with the macro-categories labels for the hyperparameters optimization of the model for the macro-categories classification. The hyperparameter space is the one we defined for ANN in Chapter 3.3, and each hyperparameter set chosen is cross-validated using *Stratified k-fold Cross-Validation* to optimize the chosen performance metric that is the accuracy.

3. **BLK-3**: we gave to the *create extractor* function the output model called *Best Macro Model* (Figure 4.5a) i.e, the one trained on the train set with the best set of hyperparameter found. The function removed its output layer, i.e., the one that gives in output one of the four macro categories, and outputted a model called *Extractor* (Figure 4.5b).



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_359 (InputLayer)      [(None, 99)]              0

 dense_953 (Dense)           (None, 99)                9900

 dropout_595 (Dropout)       (None, 99)                0

 dense_954 (Dense)           (None, 4)                 400

=================================================================
Total params: 10,300
Trainable params: 10,300
Non-trainable params: 0
```

(a) Example of Best Macro Model.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_359 (InputLayer)      [(None, 99)]              0

 dense_953 (Dense)           (None, 99)                9900

 dropout_595 (Dropout)       (None, 99)                0

=================================================================
Total params: 9,900
Trainable params: 0
Non-trainable params: 9,900
```

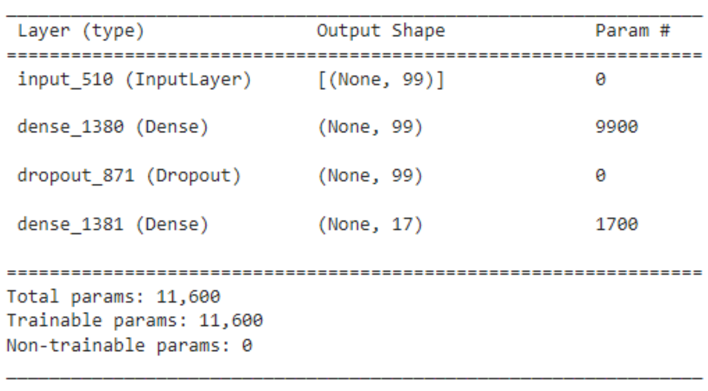
(b) Example of Extractor.

Figure 4.5: On top is shown an example of ANN used to classify macro-categories, while on bottom the extractor derived from it, i.e., the same ANN model without the output layer.

We gave the input train and test features to the *Extractor* that predicted a new features space, i.e. a new train set and test set.

4. **BLK-4**: we used the new train set with the micro-categories label for the hyper-parameters optimization of the model for the micro-categories classification. The hyperparameter space is the one we defined for ANN, and each hyperparameter set is cross-validated using *Stratified k-fold Cross-Validation* to optimize the chosen performance metric that is the accuracy.

5. **BLK-5**: we tested the output model called *Best TL Model* (Figure 4.6), i.e., the one trained on the train set with best set of hyperparameter, on the new test set. We stored the performance metrics obtained on this split.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_510 (InputLayer)      [(None, 99)]              0

 dense_1380 (Dense)          (None, 99)                9900

 dropout_871 (Dropout)       (None, 99)                0

 dense_1381 (Dense)          (None, 17)                1700

=================================================================
Total params: 11,600
Trainable params: 11,600
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.6: Example of Best Transfer Learning Model

6. **BLK-6**: at the end of the five iterations, we computed the average performance metrics for the model.

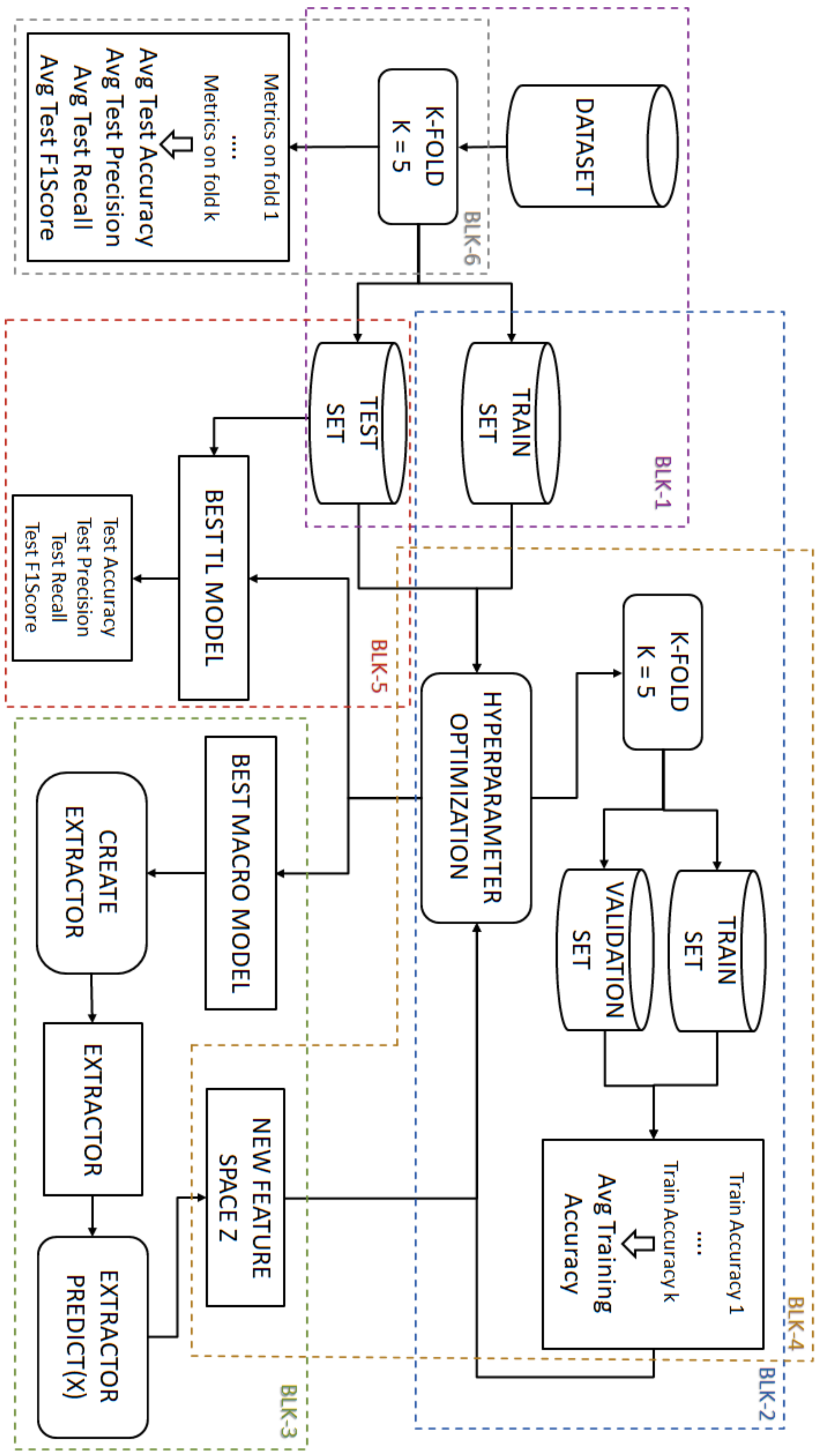


Figure 4.7: Transfer Learning pipeline with hyperparameters optimization.

4.4.3. Auxiliary-Task Learning

In this section we apply the *Auxiliary Learning* methodology presented in Chapter 3.4.3 to our classification problem. With this model our goal is to deal data scarcity by training an ANN model composed of two branches:

- *auxiliary task branch*: is the one able to classify the macro-categories.
- *primary task branch*: is the one able to classify the micro-categories.

Once these two branches are trained, their loss functions are combined, as in equation (4.1):

$$result\_loss = primary\_loss + \lambda * auxiliary\_loss \tag{4.1}$$

where the *primary_loss* is the output loss of the primary task (micro-categories classification), the *auxiliary_loss* is the output loss of the auxiliary task (macro-categories classification), and λ is a new hyperparameter of the model that establishes the weight to be attributed to the computed *auxiliary_loss*. We expect that this defined loss function will benefit from the auxiliary loss to enhance the classification on the 17 categories of hardware failures.

Figure 4.8 shows an example of Auxiliary Task model created using this methodology: it shows an ANN that has an input layer with 99 neurons, a dense layer with 123 neurons, a dropout layer with 123 neurons, and then the network is split into two branches. The left branch is used for the primary task, i.e. micro-category classification, and is made of a dense layer with 62 neurons, a dropout layer with 62 neurons, a dense layer with 41 neurons and the final output layer with 17 neurons. The right branch is used for the auxiliary task, i.e. macro-category classification, and is made of a dense layer with 62 neurons, a dropout layer with 62 neurons, and the final output layer with 4 neurons.

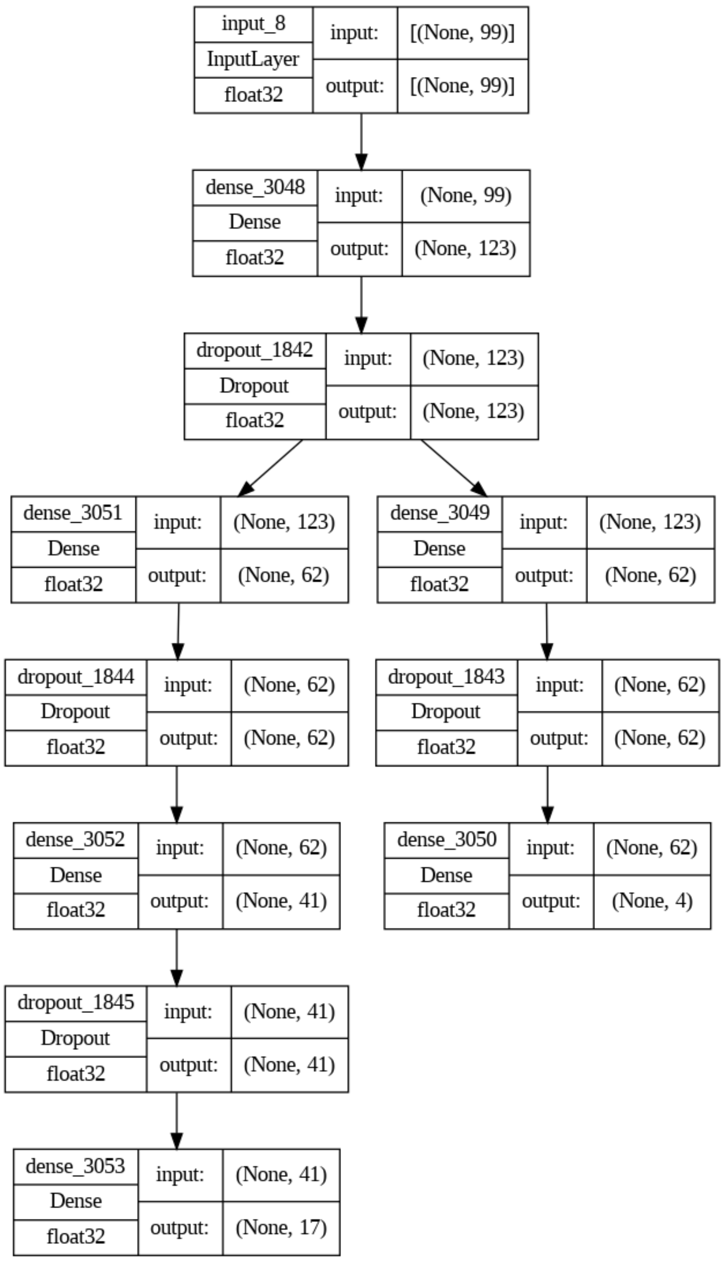


Figure 4.8: Example of Auxiliary-Task model, the left branch is used for the primary task, i.e. micro-category classification, while the right branch is used for the auxiliary task, i.e. macro-category classification.

To realize this type of model, it is necessary to create a subclass of the Keras Model [19] redefining the *init* function, the *call* function, the *train_ step* function and the *test_ step* function.

Figure 4.9 illustrates the proposed pipeline for this methodology:

1. **BLK-1**: to train the network branch for the macro-category classification, we

needed to add the original column “Macro Category” to the dataset presented in Figure 4.2. We split this dataset is split in 5 folds using *Stratified k-fold Cross-Validation* to validate the chosen performance metrics for the models, i.e., accuracy, precision, recall, and F1-score.

2. **BLK-2**: we used the train set for the hyperparameters optimization of the model. The hyperparameter space is the one defined for ANN in the section above, plus the hyperparameter lambda which values tested for this thesis work are: [0.3, 0.5, 0.7, 0.9, 1.0, 1.2, 1.5]. Each hyperparameter set is cross-validated using *Stratified k-fold Cross-Validation* to optimize the chosen performance metric that is the accuracy.

3. **BLK-3**: we tested the output *Best Auxiliary-Task Model*, i.e., the one trained on the train set with best set of hyperparameter, on the test set. We stored the performance metrics obtained on this split.

4. **BLK-4**: at the end of the five iterations, we computed the average performance metrics for the model.

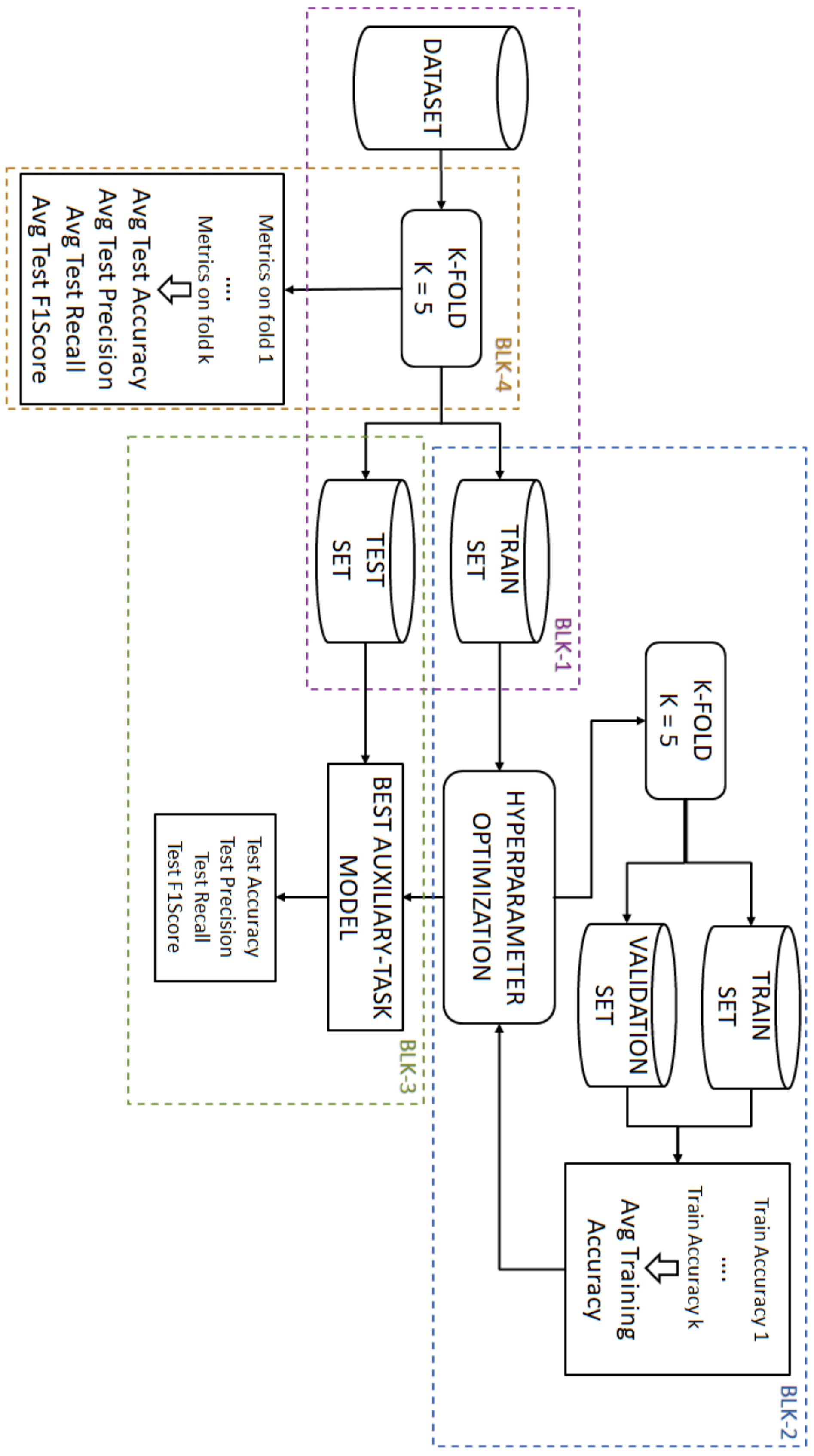


Figure 4.9: Auxiliary-Task Learning pipeline with hyperparameters optimization.

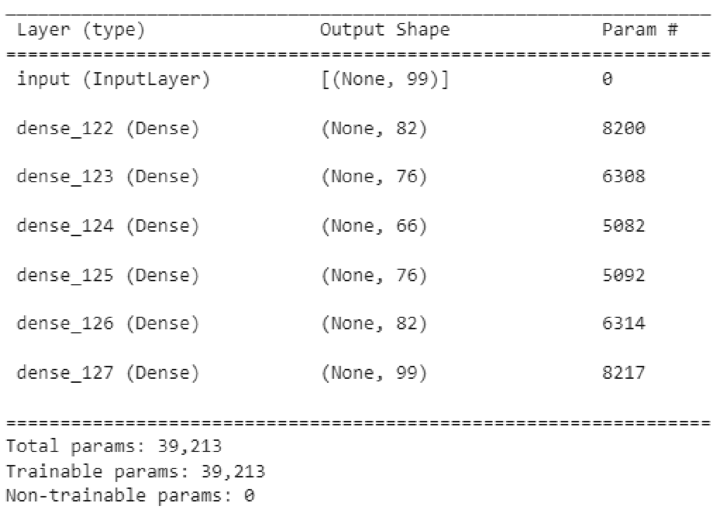
4.4.4. Denoising Autoencoders

In this section we apply the *Denoising Autoencoders* methodology presented in Chapter 3.4.4 to our classification problem. Our goal is to address data scarcity by training a Denoising Autoencoders able to learn useful features representations without the need for labels. The latent space of this self-supervised method, receiving corrupted features as input, must learn not only a compressed representation of the features but also the structural relationships within them. On our dataset, we expect that this model will be able to learn the relationship between the alarms, understanding relationships like:

- if alarm “a” or set of alarms “A” are on, also alarm “b” or set of alarms “B” are on.
- if alarm “a” or set of alarms “A” are off, also alarm “b” or set of alarms “B” are off.
- if alarm “a” or set of alarms “A” are on, alarm “b” or set of alarms “B” are off.
- if alarm “a” or set of alarms “A” are off, alarm “b” or set of alarms “B” are on.

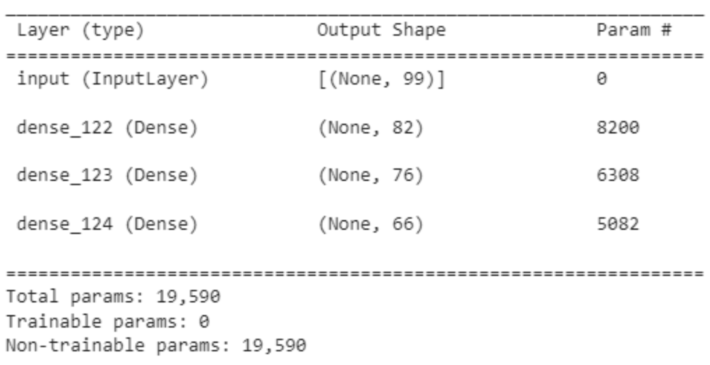
Figure 4.12 illustrates the proposed pipeline for this methodology:

1. **BLK-1**: we split the dataset (Figure 4.2) in 5 folds using *Stratified k-fold Cross-Validation* to validate the chosen performance metrics for the models, that are: accuracy, precision, recall and F1-score.
2. **BLK-2**: we gave the feature in the train set to the *add_ noise* function that outputs a corrupted train set, i.e., for each data point in the train set flipping from 0 to 1 and vice versa a random 30% of features (in our case 30 random features over 99).
3. **BLK-3**: we gave to the function *create_ extractor* a Denoising Autoencoder (DAE) model (Figure 4.10a) trained to reconstruct the original train set from the corrupted one. The function removed its decoding layers, i.e., the ones used to reconstruct the original train set, and outputted a model called *Extractor* (Figure 4.10b).



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 99)]              0

 dense_122 (Dense)           (None, 82)                8200

 dense_123 (Dense)           (None, 76)                6308

 dense_124 (Dense)           (None, 66)                5082

 dense_125 (Dense)           (None, 76)                5092

 dense_126 (Dense)           (None, 82)                6314

 dense_127 (Dense)           (None, 99)                8217

=================================================================
Total params: 39,213
Trainable params: 39,213
Non-trainable params: 0
```

(a) Example of Denoising Autoencoders (DAE).



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 99)]              0

 dense_122 (Dense)           (None, 82)                8200

 dense_123 (Dense)           (None, 76)                6308

 dense_124 (Dense)           (None, 66)                5082

=================================================================
Total params: 19,590
Trainable params: 0
Non-trainable params: 19,590
```

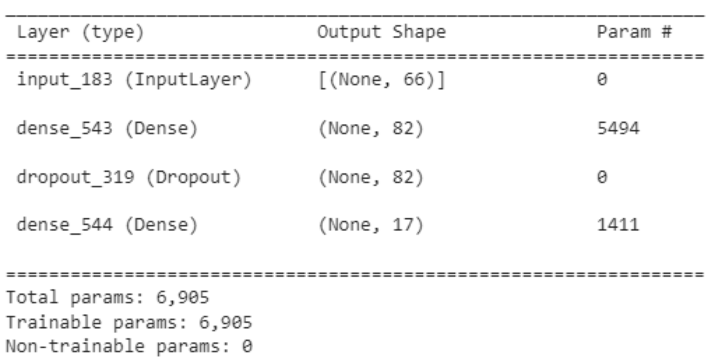
(b) Example of Extractor.

Figure 4.10: On top is shown an example of DAE model used to reconstruct the original train set from the corrupted one, while on bottom the extractor derived from it, i.e., the same DEA model without the decoding layers.

We gave the input train and test features to the *Extractor*, that predicted a new features space, i.e. a new train set and test set.

4. **BLK-4**: we used the new train set for the hyperparameters optimization of the model. The hyperparameter space is the one defined for ANN, and each hyperparameter set is cross-validated using *Stratified k-fold Cross-Validation* to optimize the chosen performance metric that is the accuracy.

   BLK-5: we tested the output model called *Best DAE Model* (Figure 4.11), i.e., the one trained on the train set with best set of hyperparameter, on the test set. We stored the performance metrics obtained on this split.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_183 (InputLayer)      [(None, 66)]              0

 dense_543 (Dense)           (None, 82)                5494

 dropout_319 (Dropout)       (None, 82)                0

 dense_544 (Dense)           (None, 17)                1411

=================================================================
Total params: 6,905
Trainable params: 6,905
Non-trainable params: 0
```

Figure 4.11: Example of Denoising Autoencoder Model.

5. **BLK-6**: at the end of the five iterations, we computed the average performance metrics for the model.

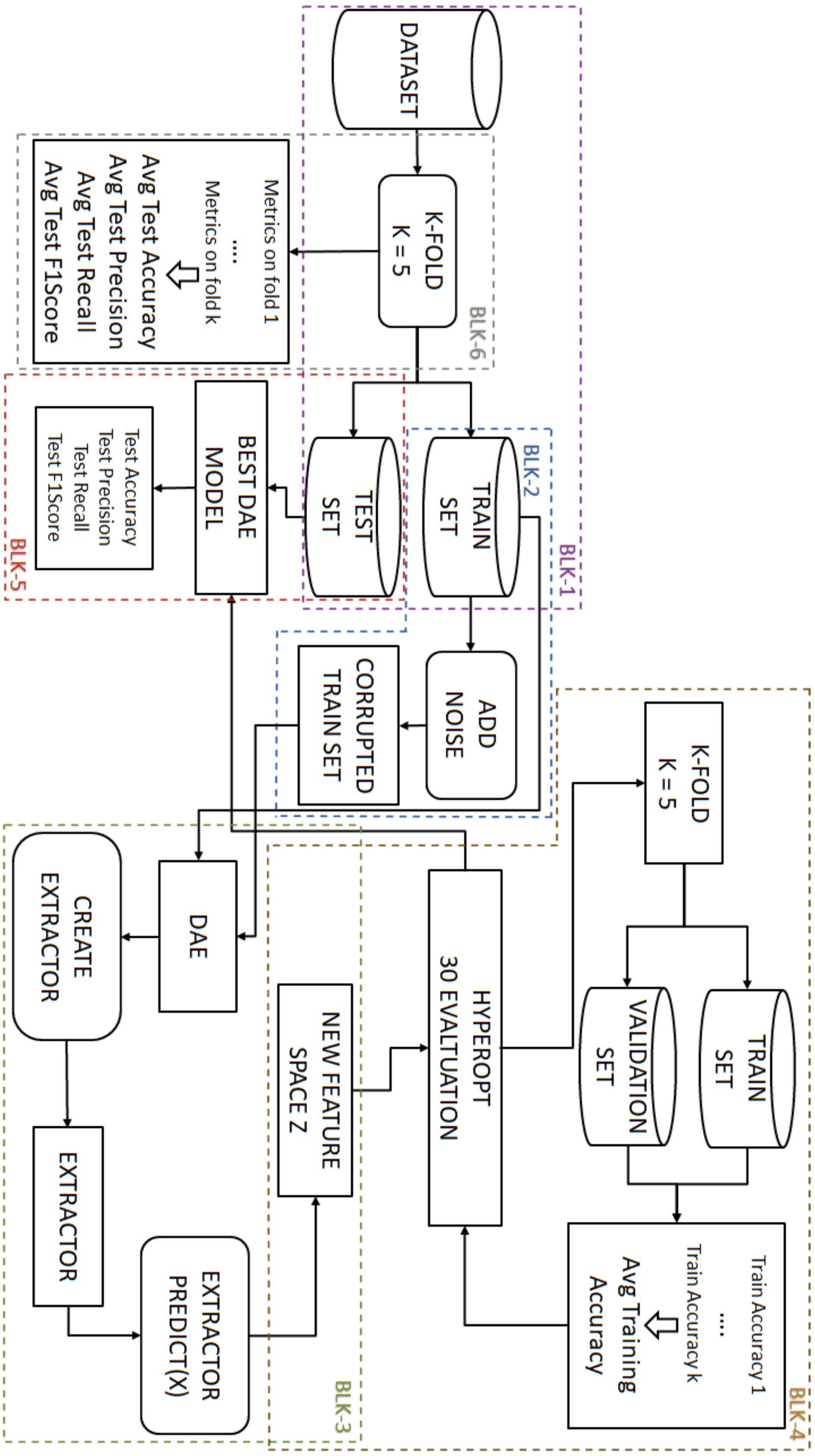


Figure 4.12: DAE pipeline with hyperparameters optimization.

5 | Numerical Results

In this chapter we discuss illustrative numerical results of the methodologies discussed in Chapter 4. We show numerical results when utilizing the baseline models, i.e., *XGBoost*, *SVC* and *ANN*, and when utilizing methodologies adopted to address data scarcity, i.e. *Synthetic Minority Over-sampling Technique (SMOTE)*, *Transfer Learning*, *Auxiliary-Task Learning* and *Denoising Autoencoders.*
Chapter-5 is organized as follows: in Section 5.1 we evaluate the performance of the baseline models, while in Section 5.2 we evaluate the impact of the methodologies adopted to address data scarcity, and in Section 5.3 we select which is the best best methodologies to address data scarcity. Finally, in Section 5.4 we provide a detailed analysis on SMOTE approach to evaluate its impact on the specific failure classes considered in this study, in order to assess in which conditions it is more convenient for a certain class to perform data augmentation by generating synthetic data.

5.1. Baseline Models

In this section we present results obtained for each baseline models described in Chapter 4.3, i.e., XGBoost, SVC and ANN, using the pipeline in Figure 4.3.

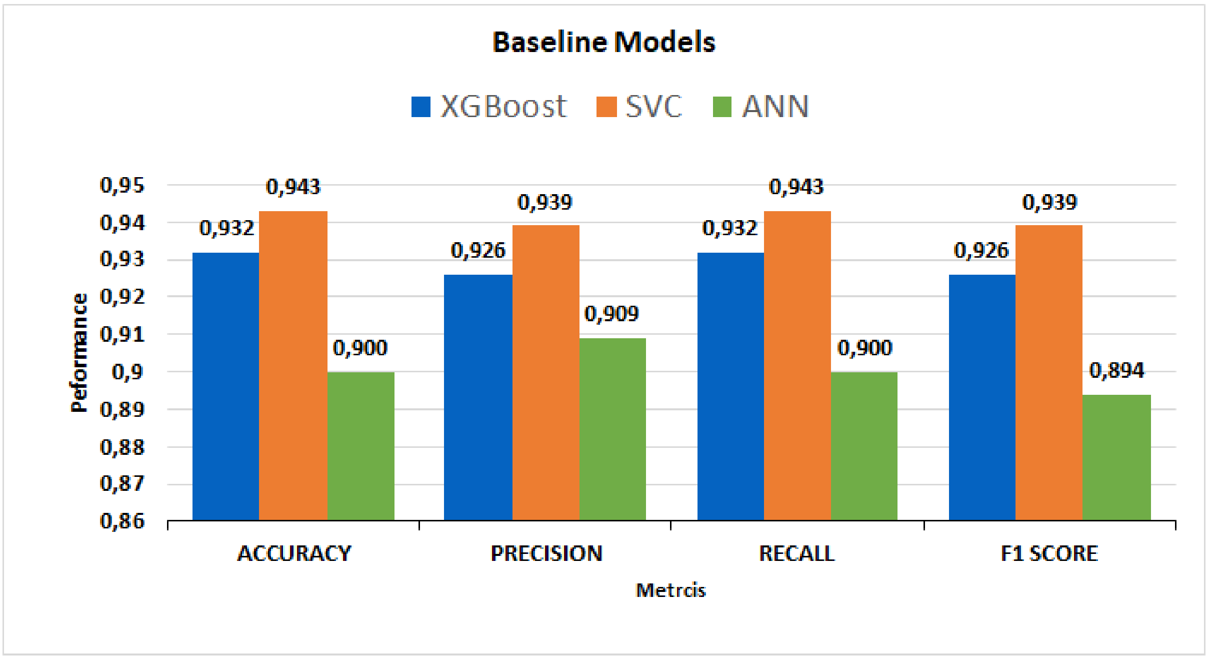


Figure 5.1: Accuracy, Precision, Recall and F1-score for the baseline models, i.e., XG-Boost, SVC and ANN, optimized on each of the 5 dataset splits.

Figure 5.1 shows performance results in terms of *Accuracy*, *Precision*, *Recall* and *F1-score* for XGBoost, SVC, and ANN. We can observe that SVC outperforms the other classifiers on all the considered metrics, and XGBoost in turn performs better than ANN. As expected, SVC and XGBoost perform better than ANN, because the latter usually requires large amounts of data for effective training.

These are the baseline performance on Accuracy, Precision, Recall, and F1-score metrics that we aim to enhance adopting the methodologies presented in Chapter 4.4.

5.2. Methodologies to address Data Scarcity

In this section we present the results obtained for each methodology adopted to address data scarcity as described in Chapter 4.4, i.e., SMOTE, Transfer Learning, Auxiliary-Task Learning and Denoising Autoencoders. The results that are presented are the ones obtained using the pipeline in Figure 4.4 for SMOTE, in Figure 4.7 for TL, in Figure 4.9 for Auxiliary-Task Learning, and in Figure 4.12 for Denoising Autoencoders.

5.2.1. SMOTE

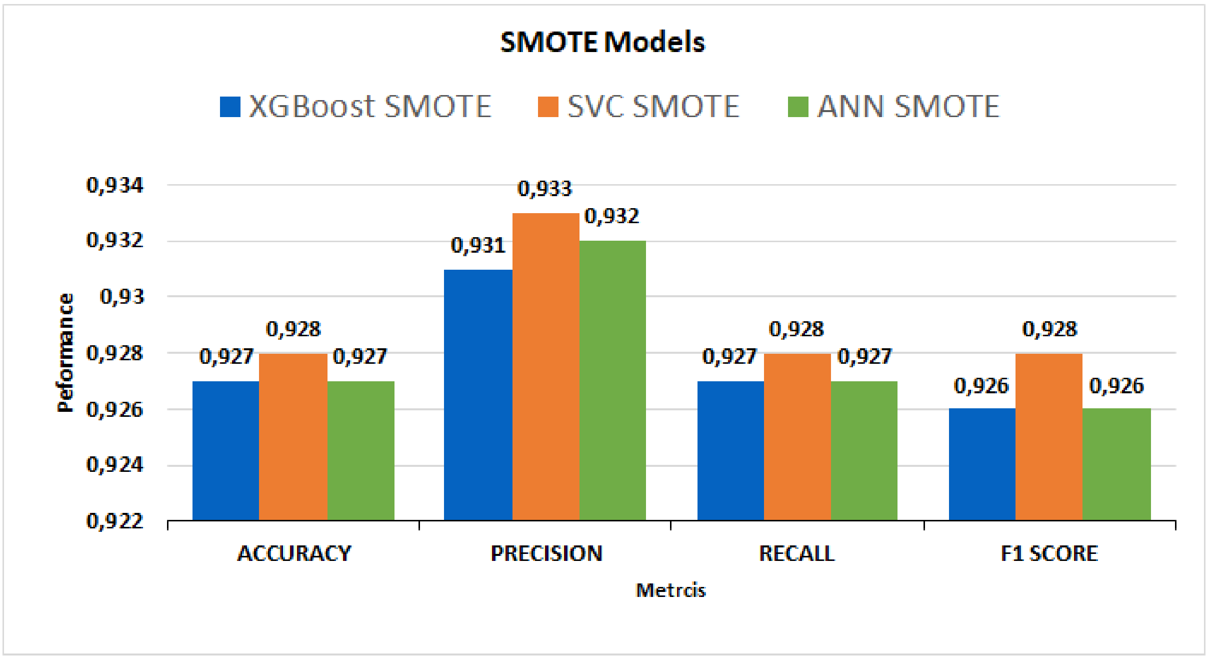


Figure 5.2: Accuracy, Precision, Recall and F1-score for SMOTE models, i.e., XGBoost, SVC and ANN, optimized on each of the 5 dataset splits.

Figure 5.2 shows performance results in terms of *Accuracy*, *Precision*, *Recall* and *F1-score* for XGBoost, SVC, and ANN trained with synthetics data generated by SMOTE. We can observe that the use of synthetics data has a very significant impact on the ANN, which improves in all the considered performance metrics with respect to the baseline performance. The ANN performance are now comparable with the one obtained using XGBoost. We can also note that the highest metric obtained is the *precision*, i.e., the number of true positive predictions divided by the total number of positive predictions, which is improved for XGBoost and ANN, while is almost the same for SVC. This result is due to the fact that SMOTE oversampling the minority class can help the classifier better learn the characteristics of the minority class and reduce the number of false positive predictions made by the classifier. Instead, we can observe that for XGBoost and SVC we have a significant drop in *recall* performance score, i.e., the number of true positive predictions divided by the total number of actual positive samples in the dataset. This is due to the fact that SMOTE oversampling the minority class can introduce synthetic samples that may not accurately represent the true distribution of the minority class. As result, the classifier may learn to overfit to the synthetic samples and not generalize well to new, unseen samples from the minority class. This can lead to a higher number of false negative predictions, which in turn can decrease the recall of the classifier. So, while SMOTE leads generally to an increase in the precision, similarly it leads to a decrease in the recall. The increase in accuracy and decrease in recall affects the F1-score: for ANN we have a significant increase in F1-score since there is an increase in both precision and recall, while for XGBoost there is an increase in precision but a comparable decrease in recall so we have the same F1-score, while for SVC, since it decreases both in precision and recall we consequently have a decrease in F1-score.

5.2.2. Transfer Learning

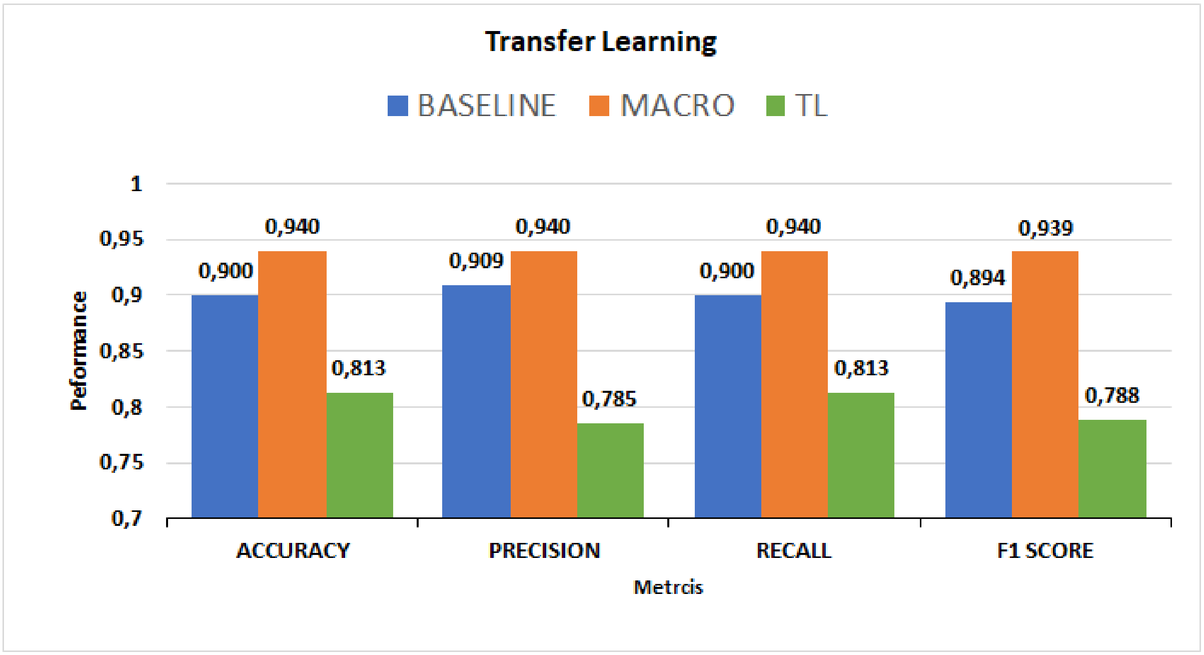


Figure 5.3: Accuracy, Precision, Recall and F1-score obtained using a ML model built on Transfer Learning ("TL") methodology, optimized on each of the 5 dataset splits. Its performance are compared with the ones got for the ANN Baseline model ("BASELINE") and with the ones got trying to classify the macro categories before doing the transfer learning task ("MACRO").

Figure 5.3 shows performance results in terms of *Accuracy*, *Precision*, *Recall* and *F1-score* for the ML model built using the *Transfer Learning* methodology presented in Chapter 4.4.2. As shown in the pipeline in Figure 4.7, to train the TL model firstly we trained an ANN model able to classify the macro-categories. The results we obtained with this model are the ones presented as "MACRO" in Figure 5.3. As expected, we can observe that this model performs better than the baseline one, because this classification task is easier than the ones on the micro-categories, due to the higher number of data points for each macro-category (see Table 4.1). Then, with the procedure explained in Chapter 4.4.2, we obtained the results for the micro-categories classification with the TL model, that are presented as "TL" in Figure 5.3. As we can see, these results are worst than the baseline ones and this can due to the task complexity or to the overfitting over the macro-classification.

5.2.3. Auxiliary-Task Learning

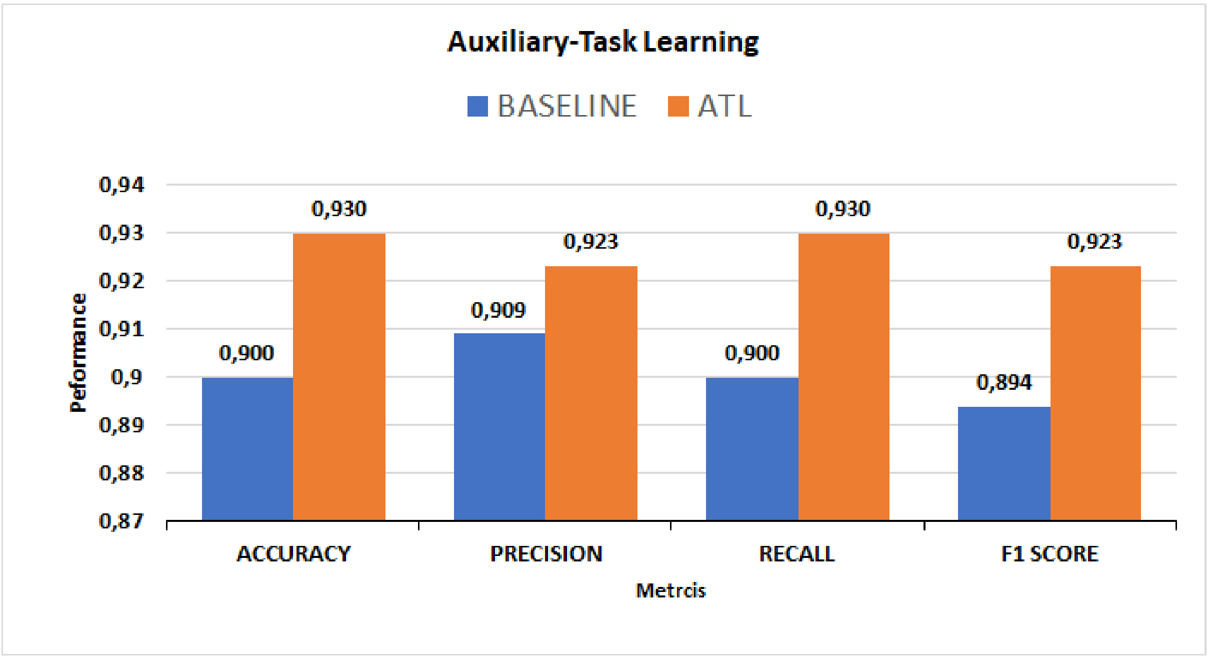


Figure 5.4: Accuracy, Precision, Recall and F1-score obtained using a ML model built on Auxiliary Task Learning ("ATL") methodology, optimized on each of the 5 dataset splits. Its performance are compared with the ones got for the ANN baseline model ("BASELINE").

Figure 5.4 shows performance results in terms of *Accuracy*, *Precision*, *Recall* and *F1-score* for the ML model built using the *Auxiliary-Task Learning* methodology presented in Chapter 4.4.3. As shown this methodology improved the model performance in all the considered metrics, this is because of the contribution that the auxiliary task, i.e., the classification on the macro-categories, is making on the loss function for the classification on the micro-categories.

5.2.4. Denoising Autoencoders

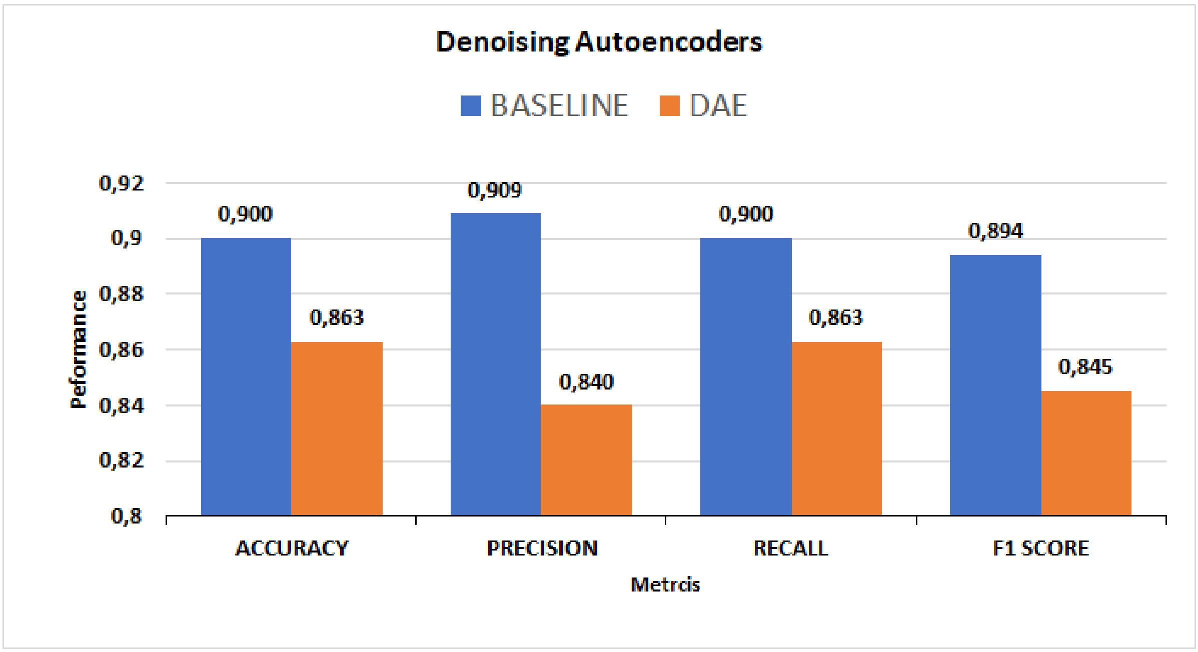


Figure 5.5: Accuracy, Precision, Recall and F1-score obtained using a ML model built on Denoising Autoencorders ("DAE") methodology, optimized on each of the 5 dataset splits. Its performance are compared with the ones got for the ANN Baseline model ("BASELINE").

Figure 5.5 shows performance results in terms of *Accuracy*, *Precision*, *Recall* and *F1-score* for the ML model built using the *Denoising Autoencoders* methodology presented in Chapter 4.4.4. The showed results are the one obtained using on each dataset split the best hyperparameters found through *HyperOpt*. As we can see this methodology brings no performance improvements on all the considered metrics. The reason may be due to overfit problem to the training data, because of model complexity or the training data is too small.

5.3. Best Methodology to address Data Scarcity

In this section we identify which is the best methodology to address data scarcity among those tested. As observed from the results in the previous section, TL and DAE turn out to be inefficient for this purpose. Instead, both SMOTE and Auxiliary-Task Learning (ATL) methodologies have led to better results than the ones obtained using the baseline models (Figure 5.6).

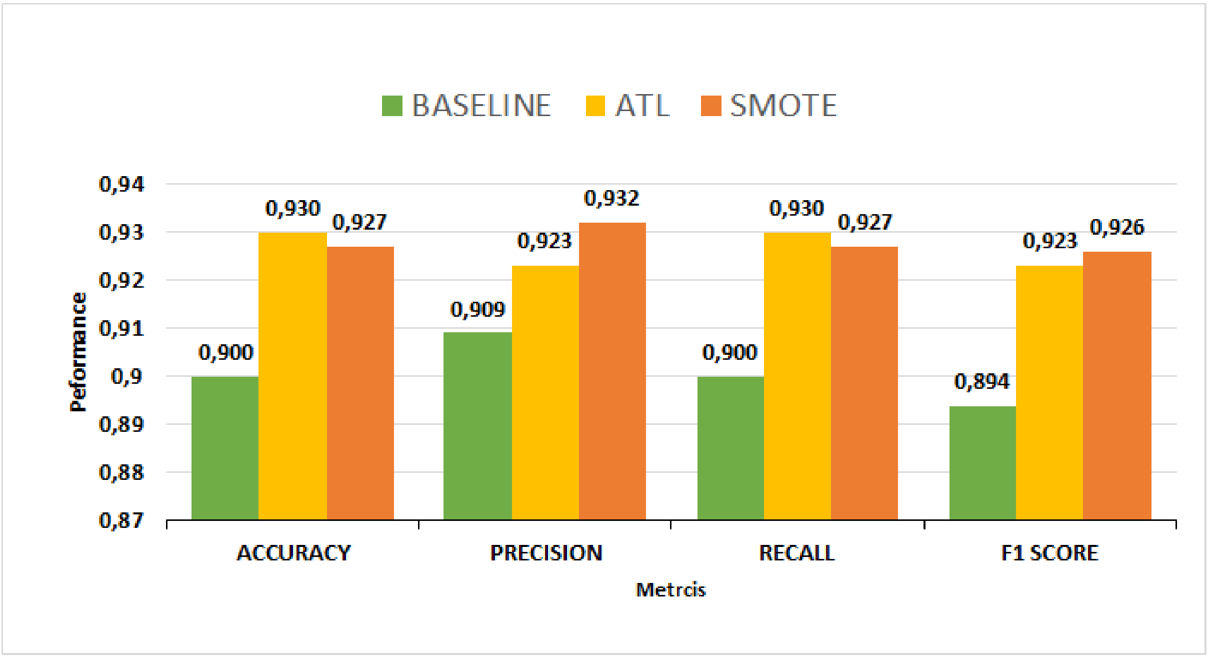
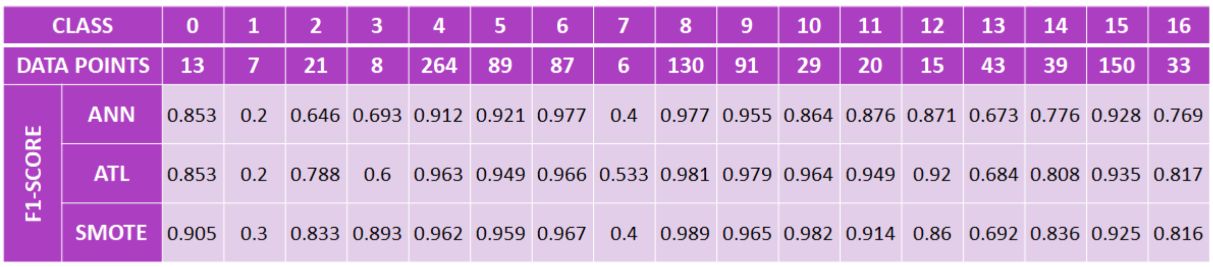


Figure 5.6: Comparison between Accuracy, Precision, Recall and F1-score obtained using ANN baseline model ("BASELINE"), a ML model built on Auxiliary-Task Learning ("ATL") methodology, and a ML model built on "SMOTE" methodology.

At this stage, to figure out which is the best one, we will use the per-class F1-score metric. We decided to choose F1-score as per class metric because we have an unbalanced class distribution, which makes accuracy unreliable, preferring metrics as precision and recall. The reason why we chose F1-score is because it is a trade-off between precision and recall, providing a single value that summarizes the overall performance of the model.



CLASS		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DATA POINTS		13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
F1-SCORE	ANN	0.853	0.2	0.646	0.693	0.912	0.921	0.977	0.4	0.977	0.955	0.864	0.876	0.871	0.673	0.776	0.928	0.769
	ATL	0.853	0.2	0.788	0.6	0.963	0.949	0.966	0.533	0.981	0.979	0.964	0.949	0.92	0.684	0.808	0.935	0.817
	SMOTE	0.905	0.3	0.833	0.893	0.962	0.959	0.967	0.4	0.989	0.965	0.982	0.914	0.86	0.692	0.836	0.925	0.816

Figure 5.7: Comparison between per-class F1-score values obtained using ANN baseline model ("ANN"), a ML model built on Auxiliary-Task Learning ("ATL") methodology and a ML model built on SMOTE methodology.

Figure 5.7 shows the comparison between the F1-score values of the ANN baseline model and the ones obtained applying ATL and SMOTE methodologies. We can observe that on the classes that are characterized by a low number of data points in general the highest per-class F1-score is obtained using SMOTE. This is true for class 0, 1, 2, 3, 10, 13, and 14, while only for class 7, 11, and 12 the opposite is true.

From these results we get that SMOTE is the most suitable technique to deal with data scarcity, so in the next section we propose an in-depth analysis of this methodology.

5.4. SMOTE Analysis

In the previous section we showed that SMOTE methodology was the best one to address data scarcity. In this section we provide an in-depth analysis of this methodology to evaluate its impact on the specific failure classes considered in this study. Our goal is to assess in which conditions it is more convenient for a certain class to perform data augmentation by generating synthetic data. For this purpose we used the per-class F1-score given by the *Best Baseline Models*, i.e., XGBoost, SVC and ANN, initialized with the optimized hyper-parameters found in the previous section (Table 4.7, Table 4.8, Table 4.9), to understand how the data scarcity problem is impacting this performance parameter. We decided to adopt the *Best Baseline Models*, because they are already optimized models and this will allow us to not provide for the optimization of hyperparameters during this analysis.

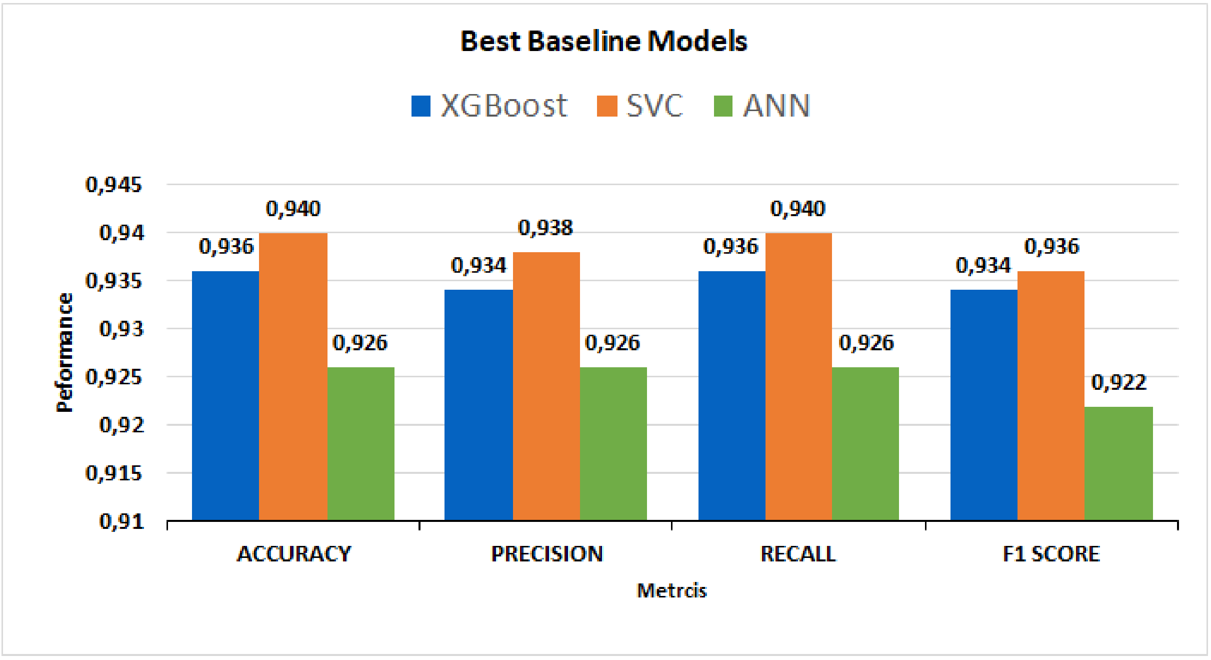
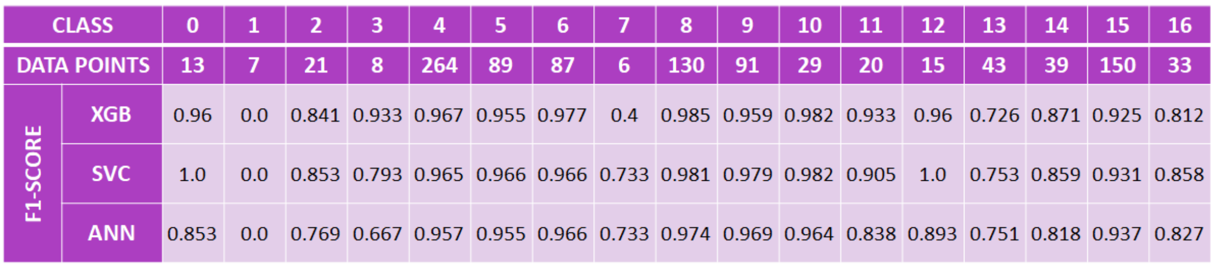


Figure 5.8: Performance results in terms of Accuracy, Precision, Recall and F1-score for the Best Baseline models, i.e., XGBoost, SVC and ANN with the best hyperparameter configurations.

Figure 5.8 shows the results obtained with *Best Baseline Models* on the considered metrics, i.e., accuracy, precision, recall, and F1-score. As we can see, all the *Best Baseline Models* exploiting the best hyperparameter configurations perform almost the same as or even better than the *Baseline Models*, i.e., the ones where we do hyperparameters search on each split. This shows the validity of the criterion explained in Chapter 4.3 for choosing the best hyperparameters. As we got for the baseline models, also with the Best Baseline Models the results obtained with SVC are better than the ones obtained with XGBoost that in turn are better than the ones obtained with ANN.



CLASS		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DATA POINTS		13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
F1-SCORE	XGB	0.96	0.0	0.841	0.933	0.967	0.955	0.977	0.4	0.985	0.959	0.982	0.933	0.96	0.726	0.871	0.925	0.812
	SVC	1.0	0.0	0.853	0.793	0.965	0.966	0.966	0.733	0.981	0.979	0.982	0.905	1.0	0.753	0.859	0.931	0.858
	ANN	0.853	0.0	0.769	0.667	0.957	0.955	0.966	0.733	0.974	0.969	0.964	0.838	0.893	0.751	0.818	0.937	0.827

Figure 5.9: Per-class F1-score values for the Best Baseline Models, i.e., XGBoost, SVC and ANN with the best hyperparameter configurations.

Figure 5.9 shows the per-class F1-score given by the *Best Baseline Models* that will be used to evaluate the SMOTE performance to address data scarcity during this analysis. We can observe that classes with low F1-score performance are also those with fewer data points, e.g. class 1, 2, 3, 7, 13, 14 and 16. This shows us how the problem of data scarcity manifests itself on models' performance.

The first step of this analysis is to identify a metric that, based on the per-class F1-score, defines the criticality of the single class, i.e. how poor is the performance score on that class. The chosen metric is the first and second quantile score computed on the per-class F1-score given by the three *Best Baseline Models*. In particular, we assigned a color to each class according to its F1-score value and the quantiles, following this criteria:

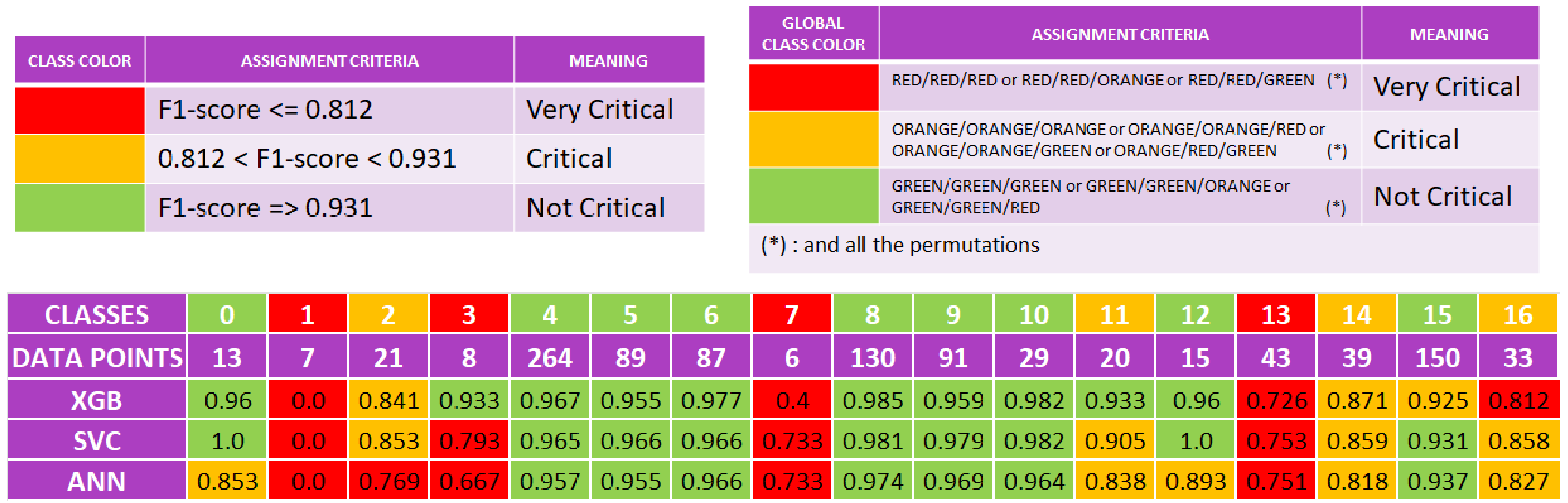
- RED: if F1-score is below than or equal to first quantile (<=25%). This means that the class is considered as very critical.
- ORANGE: if F1-score is below than the second quantile (<50%). This means that the class is considered as critical.
- GREEN: if the F1-score is greater than or equal to the second quantile (=>50%). This means that the class is considered as not critical.

Once we had for each class the colors given by the considered classifiers, we assigned a global class color according to the following criteria:

- If the tree classifiers give a unique class color, that is the global color we assigned to that class.
- If the three classifiers give two different class colors, we assigned as global class color to that class the predominant one, i.e., the one given by largest number of classifiers.
- If the tree classifiers give a different class color, we assigned as global class color to that class the orange color.

Since we had 3 *Best Baseline Models* and 17 classes of hardware failures, we got 51 F1-score values overall, as shown in Figure 5.9. On these values we computed the first and second quantile, which turned out to be 0.812 and 0.931, respectively. Following the approach explained, we gave a global color to each class, as shown in Figure 5.10, i.e.:

- **RED**: classes 1, 3, 7, and 13.
- **ORANGE**: classes 2, 11, 14, and 16.
- **GREEN**: classes 0, 4, 5, 6, 8, 9, 10, 12, and 15.



CLASS COLOR	ASSIGNMENT CRITERIA	MEANING
🟥	F1-score <= 0.812	Very Critical
🟧	0.812 < F1-score < 0.931	Critical
🟩	F1-score => 0.931	Not Critical

GLOBAL CLASS COLOR	ASSIGNMENT CRITERIA	MEANING
🟥	RED/RED/RED or RED/RED/ORANGE or RED/RED/GREEN (*)	Very Critical
🟧	ORANGE/ORANGE/ORANGE or ORANGE/ORANGE/RED or ORANGE/ORANGE/GREEN or ORANGE/RED/GREEN (*)	Critical
🟩	GREEN/GREEN/GREEN or GREEN/GREEN/ORANGE or GREEN/GREEN/RED (*)	Not Critical

(*) : and all the permutations

CLASSES	🟩 0	🟥 1	🟧 2	🟥 3	🟩 4	🟩 5	🟩 6	🟥 7	🟩 8	🟩 9	🟩 10	🟧 11	🟩 12	🟥 13	🟧 14	🟩 15	🟧 16
DATA POINTS	13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
XGB	🟩 0.96	🟥 0.0	🟧 0.841	🟩 0.933	🟩 0.967	🟩 0.955	🟩 0.977	🟥 0.4	🟩 0.985	🟩 0.959	🟩 0.982	🟩 0.933	🟩 0.96	🟥 0.726	🟧 0.871	🟧 0.925	🟥 0.812
SVC	🟩 1.0	🟥 0.0	🟧 0.853	🟥 0.793	🟩 0.965	🟩 0.966	🟩 0.966	🟥 0.733	🟩 0.981	🟩 0.979	🟩 0.982	🟧 0.905	🟩 1.0	🟥 0.753	🟧 0.859	🟩 0.931	🟧 0.858
ANN	🟧 0.853	🟥 0.0	🟥 0.769	🟥 0.667	🟩 0.957	🟩 0.955	🟩 0.966	🟥 0.733	🟩 0.974	🟩 0.969	🟩 0.964	🟧 0.838	🟧 0.893	🟥 0.751	🟧 0.818	🟩 0.937	🟧 0.827

Figure 5.10: F1-score value for each class of hardware failure and each Best Baseline Model, i.e., XGBoost, SVC and ANN with the best hyperparameter configurations. Each class has a color assigned as explained in the legends.

Comparing the global color of the classes with their number of data points, we notice that:

- All classes marked as green, except for class 0, 10 and 12, have a relatively high number of data points.
- All classes marked as red, except for class 13, have a relatively low number of data points.
- All the classes marked as orange are the ones with an intermediate number of data points.

As a main takeaway from this analysis, we observe that, in most cases, there is a correlation between data scarcity and classification performance.

We used this class classification to define two "sampling strategies" by which SMOTE will be applied:

- **Critical-classes**: which considers only the critical classes, i.e., the ones that have RED or ORANGE as global class color.
- **All-classes**: which considers all the classes independently from the global class color assigned.

Given the global colors for each class and these two strategies, we find that:

- Classes 1, 2, 3, 7, 11, 13, 14 and 16 are the ones considered critical by the "Critical-classes" strategy.
- Classes 0, 4, 5, 6, 8, 9, 10, 12 and 15 are the ones considered not critical by "Critical-classes" strategy.

With the purpose of identifying the amount of synthetic data to be generated on the classes, we established four "percentages" of synthetic data generation, i.e, "12%", "35%", "50%" and "65%", where each percentage represents the number of data points, i.e., real data points plus synthetic data points, in the train set of a specific class. We computed these percentages taking as reference the class with the highest number of data points, i.e. class 4 with 264 samples (Table 4.3). In particular, each percentage has the following meaning:

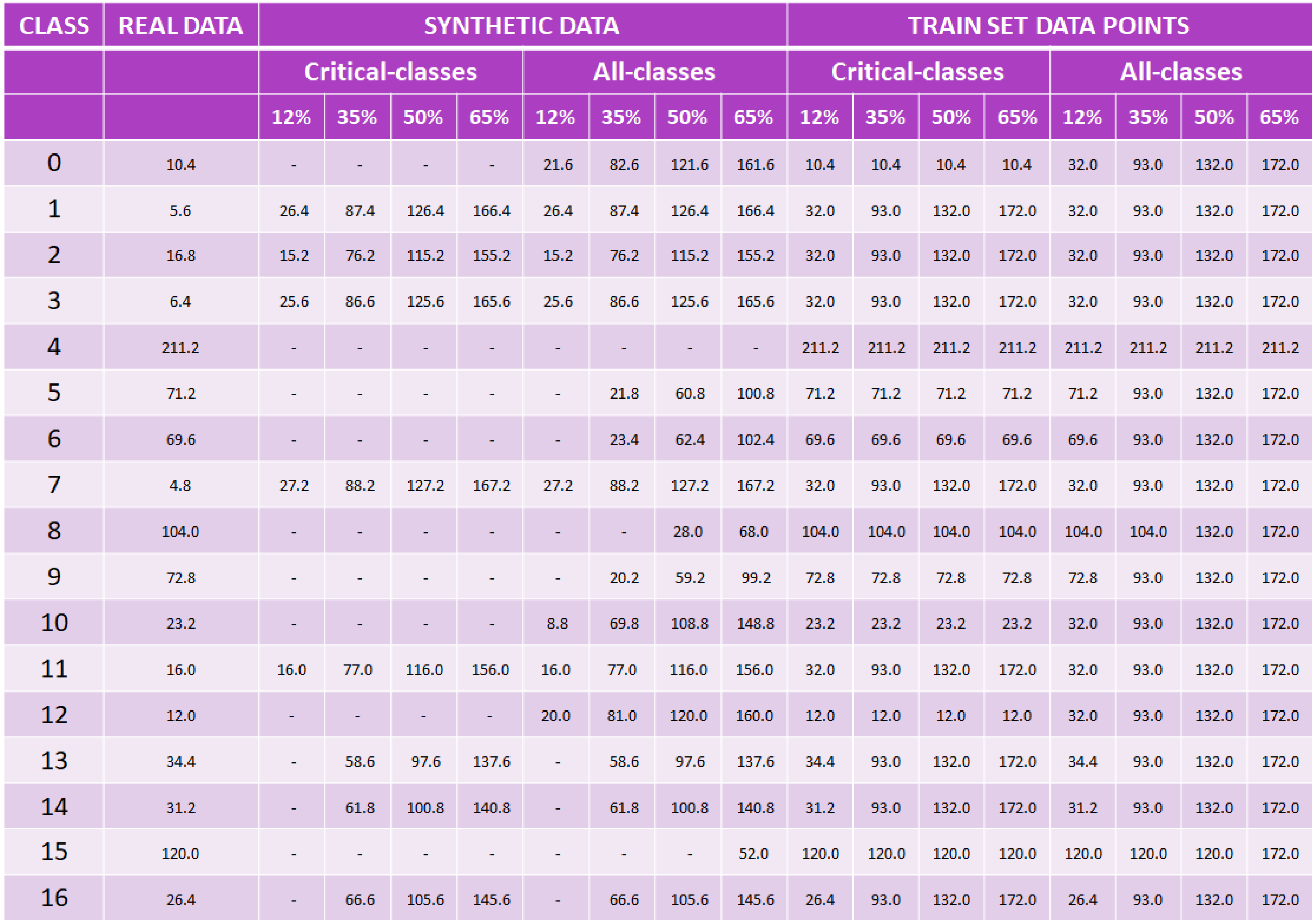
- 12% corresponds to 32 data points in the train set.
- 35% corresponds to 93 data points in the train set.
- 50% corresponds to 132 data points in the train set.

- 65% corresponds to 172 data points in the train set.

Combining the strategies, i.e., “Critical-classes” and “All-classes”, and the percentages above, we obtain the following “combinations” are obtained:

- **Critical-classes 12%**: classes 1, 2, 3, 7, and 11 now have 32 data points in the train set.
- **Critical-classes 35%**: classes 1, 2, 3, 7, 11, 13, 14, and 16 now have 93 data points in the train set.
- **Critical-classes 50%**: classes 1, 2, 3, 7, 11, 13, 14, and 16 now have 132 data points in the train set.
- **Critical-classes 65%**: classes 1, 2, 3, 7, 11, 13, 14, and 16 now have 172 data points in the train set.
- **All-classes 12%**: classes 0, 1, 2, 3, 7, 10, 11, and 12 now have 32 data points in the train set.
- **All-classes 35%**: classes 0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, and 16 now have 93 data points in the train set.
- **All-classes 50%**: classes 0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 16 now have 132 data points in the train set.
- **All-classes 65%**: classes 0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, and 16 now have 172 data points in the training set.

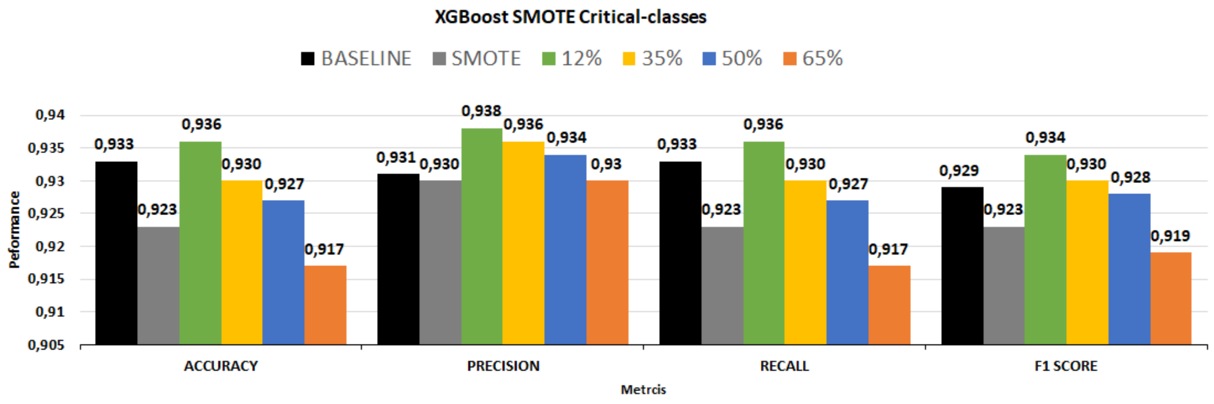
Figure 5.11 shows the number of real data points and synthetic data points in the train set on all the classes for all the combinations described above.



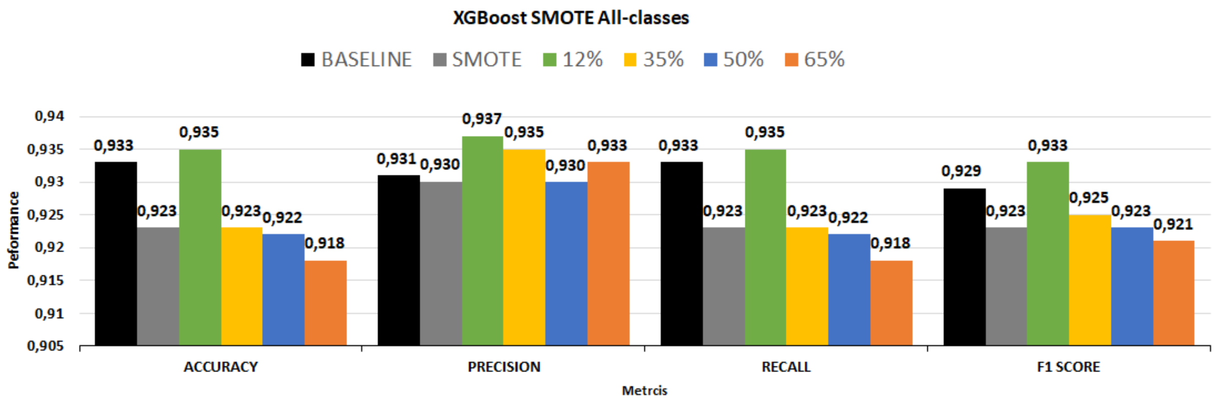
CLASS	REAL DATA	SYNTHETIC DATA								TRAIN SET DATA POINTS							
		Critical-classes				All-classes				Critical-classes				All-classes			
		12%	35%	50%	65%	12%	35%	50%	65%	12%	35%	50%	65%	12%	35%	50%	65%
0	10.4	-	-	-	-	21.6	82.6	121.6	161.6	10.4	10.4	10.4	10.4	32.0	93.0	132.0	172.0
1	5.6	26.4	87.4	126.4	166.4	26.4	87.4	126.4	166.4	32.0	93.0	132.0	172.0	32.0	93.0	132.0	172.0
2	16.8	15.2	76.2	115.2	155.2	15.2	76.2	115.2	155.2	32.0	93.0	132.0	172.0	32.0	93.0	132.0	172.0
3	6.4	25.6	86.6	125.6	165.6	25.6	86.6	125.6	165.6	32.0	93.0	132.0	172.0	32.0	93.0	132.0	172.0
4	211.2	-	-	-	-	-	-	-	-	211.2	211.2	211.2	211.2	211.2	211.2	211.2	211.2
5	71.2	-	-	-	-	-	21.8	60.8	100.8	71.2	71.2	71.2	71.2	71.2	93.0	132.0	172.0
6	69.6	-	-	-	-	-	23.4	62.4	102.4	69.6	69.6	69.6	69.6	69.6	93.0	132.0	172.0
7	4.8	27.2	88.2	127.2	167.2	27.2	88.2	127.2	167.2	32.0	93.0	132.0	172.0	32.0	93.0	132.0	172.0
8	104.0	-	-	-	-	-	-	28.0	68.0	104.0	104.0	104.0	104.0	104.0	104.0	132.0	172.0
9	72.8	-	-	-	-	-	20.2	59.2	99.2	72.8	72.8	72.8	72.8	72.8	93.0	132.0	172.0
10	23.2	-	-	-	-	8.8	69.8	108.8	148.8	23.2	23.2	23.2	23.2	32.0	93.0	132.0	172.0
11	16.0	16.0	77.0	116.0	156.0	16.0	77.0	116.0	156.0	32.0	93.0	132.0	172.0	32.0	93.0	132.0	172.0
12	12.0	-	-	-	-	20.0	81.0	120.0	160.0	12.0	12.0	12.0	12.0	32.0	93.0	132.0	172.0
13	34.4	-	58.6	97.6	137.6	-	58.6	97.6	137.6	34.4	93.0	132.0	172.0	34.4	93.0	132.0	172.0
14	31.2	-	61.8	100.8	140.8	-	61.8	100.8	140.8	31.2	93.0	132.0	172.0	31.2	93.0	132.0	172.0
15	120.0	-	-	-	-	-	-	-	52.0	120.0	120.0	120.0	120.0	120.0	120.0	120.0	172.0
16	26.4	-	66.6	105.6	145.6	-	66.6	105.6	145.6	26.4	93.0	132.0	172.0	26.4	93.0	132.0	172.0

Figure 5.11: Number of real, synthetic and overall data points in the train set on each class for each combination of strategy and percentage.

For each combination, we obtained the following results:

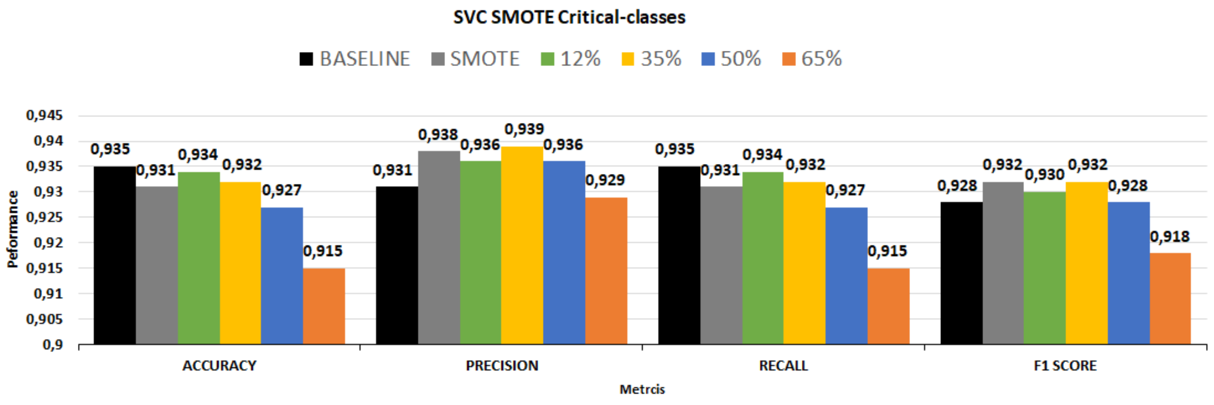


(a) XGBoost results with SMOTE "Critical-classes" strategy.

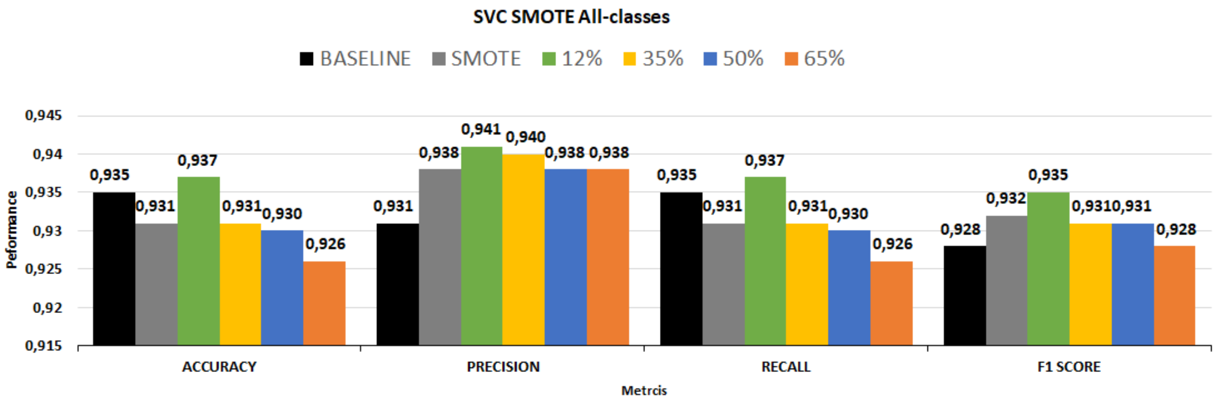


(b) XGBoost results with SMOTE "All-classes" strategy.

Figure 5.12: Accuracy, Precision, Recall and F1-score results obtained training the Best XGBoost Baseline Model, i.e., XGBoost with the best hyperparameter configuration, with synthetic data using the defined strategies ("Critical-classes" in (a) or "All-classes" in (b)) and percentages ("12%", "35%", "50%" or "65%"). Its performance is compared with that of the Best XGBoost Baseline Model ("BASELINE") and that of the "SMOTE" model without the use of strategies and percentages.

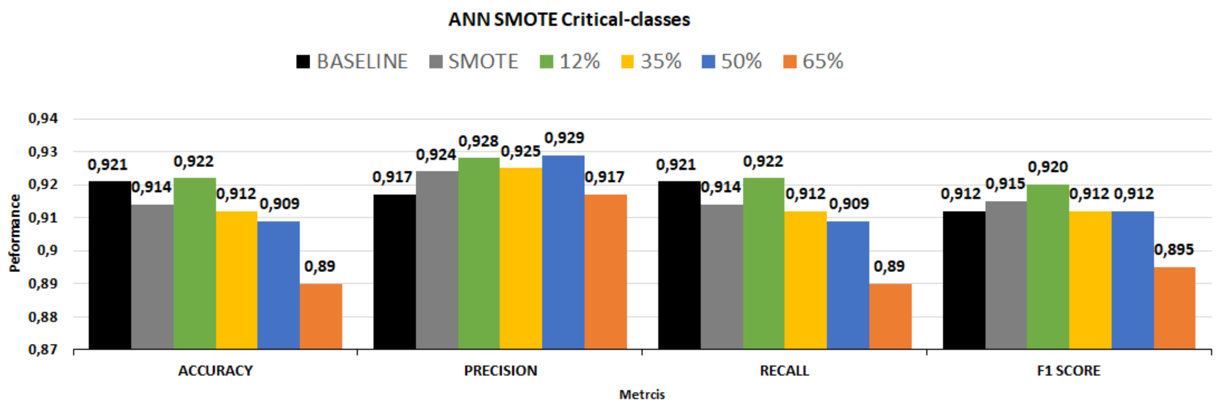


(a) SVC results with SMOTE "Critical-classes" strategy.

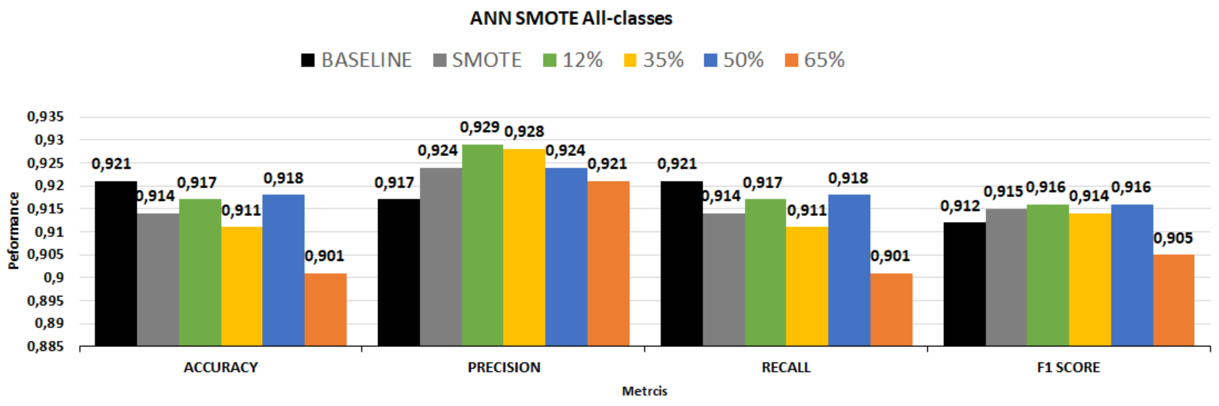


(b) SVC results with SMOTE "All-classes" strategy.

Figure 5.13: Accuracy, Precision, Recall and F1-score results obtained training the Best SVC Baseline Model, i.e., SVC with the best hyperparameter configuration, with synthetic data using the defined strategies ("Critical-classes" in (a) or "All-classes" in (b)) and percentages ("12%", "35%", "50%" or "65%"). Its performance is compared with that of the Best SVC Baseline Model ("BASELINE") and that of the "SMOTE" model without the use of strategies and percentages.



(a) ANN results with SMOTE "Critical-classes" strategy.



(b) ANN results with SMOTE "All-classes" strategy.

Figure 5.14: Accuracy, Precision, Recall and F1-score results obtained training the Best ANN Baseline Model, i.e., ANN with the best hyperparameter configuration, with synthetic data using the defined strategies ("Critical-classes" in (a) or "All-classes" in (b)) and percentages ("12%", "35%", "50%" or "65%"). Its performance is compared with that of the Best ANN Baseline Model ("BASELINE") and that of the "SMOTE" model without the use of strategies and percentages.

Referring to results in Figure 5.12, Figure 5.13 and Figure 5.14, we can see that:

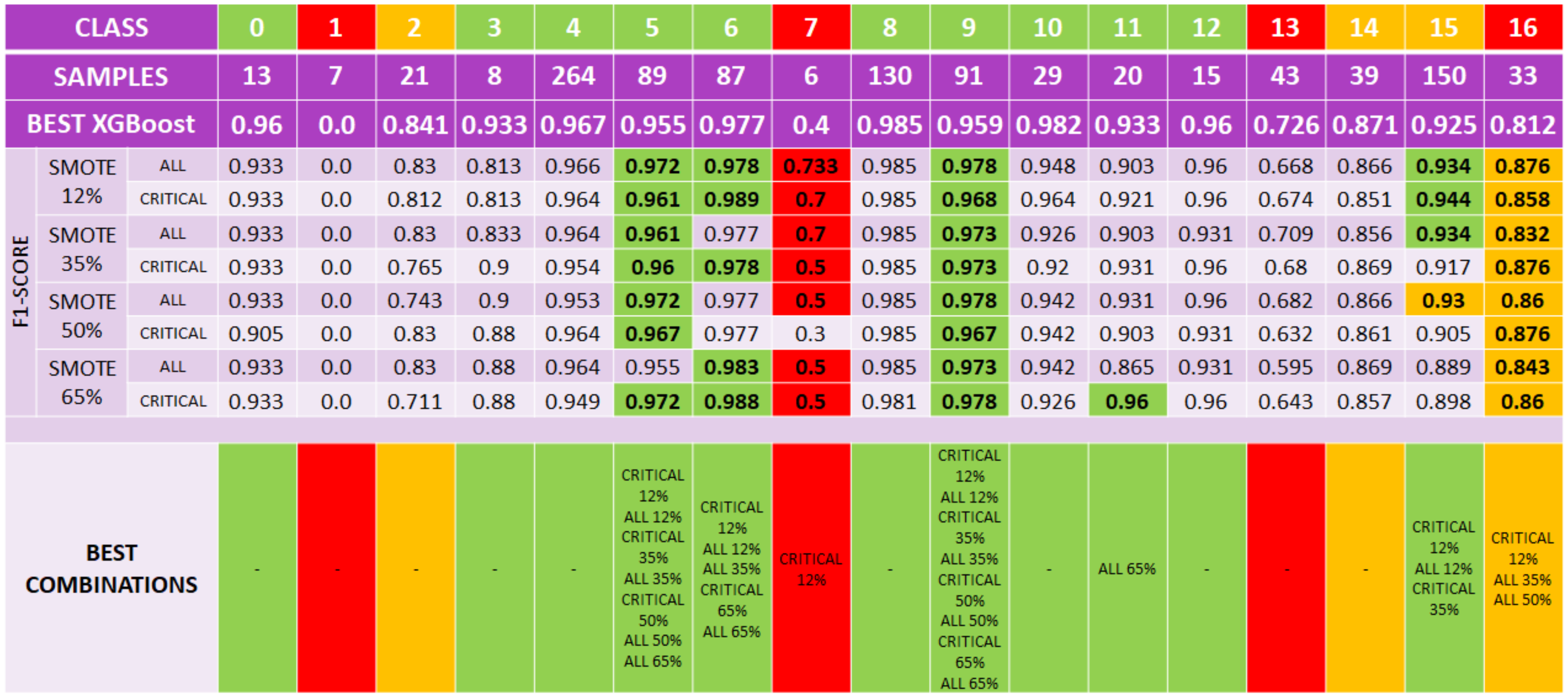
- In most cases synthetic data help to improve the global performance of the baseline models. In particular, we can observe how models trained with SMOTE on the various combinations of strategy and percentages perform better than the model trained with SMOTE without the use of them. A striking example are the results obtained with XGBoost, where we can observe that the performance of SMOTE without the use of combinations is even worse than the performance obtained with XGBoost model trained without synthetic data. Instead, using some of the proposed combinations the results are improved. This allows us to say that in order to achieve better performance with SMOTE, it is necessary to find a criterion in data generation, both in terms of classes on which to generate synthetic data and in terms of percentages of synthetic data to generate.

- Generally, the use of "All-classes" strategy performs better than the "Critical-classes" strategy, and this tells us that in order to improve the overall performance of the classifiers, it is not enough to consider only the critical classes in generating the synthetic data, but also all the others in order not to get the dataset unbalanced again towards the critical classes.
- Generally, the highest performance is obtained when considering low percentages of synthetic data, i.e., "12%" and "35%", while moving to higher percentages, i.e., "50%" and "65%", the performance drops. This is due to the fact that using high percentages of synthetic data brings us closer to the default behavior of SMOTE, which, as we saw in the previous section, leads to an overfit on the synthetic samples, and consequently the model do not generalize well to new, unseen samples from the minority class. This can lead to an high number of false negative prediction and so in the decrease in recall and F1-score. This is why these results show that generally, using SMOTE with combinations of strategies and low percentages, allow to reach better performance than using SMOTE without them.

To assess in which conditions it is more convenient for a certain class to perform data augmentation by generating synthetic data and with which percentages of synthetic data generated, we analyzed the per-class F1-score. The second step of this analysis is to compare the per-class F1-scores obtained through the use of the defined combinations with the one given by the *Best Baseline Models*. The results are shown in Figure 5.15 for XGBoost, in Figure 5.16 for SVC, and in Figure 5.17 for ANN. For each class, if the F1-score value is higher than the one obtained using the *Best Baseline Model*, we highlight the value in bold and assign the color to that class and combination, i.e., green, orange or red. The last row named "BEST COMBINATIONS" reports the combination of strategies to add synthetic data and the percentage reflecting the number of synthetic data added, with the new class color given by the color associated to these combination. For each class, the "BEST COMBINATIONS" are assigned considering the highlighted values and their color, following this criteria:

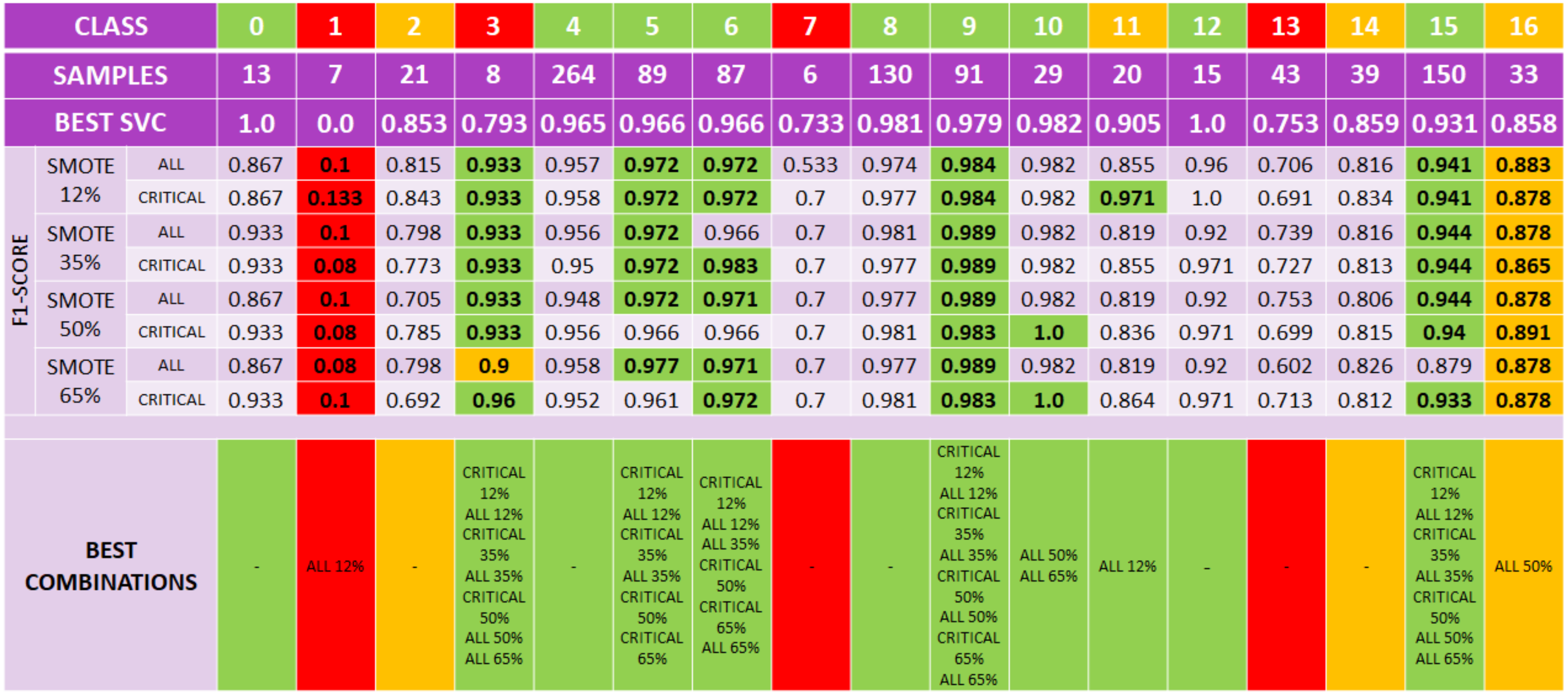
- If a combination enabled a class to improve to a green color, we report those strategies, i.e., 'Critical-classes" or "All-classes", and percentage of added data, in "BEST COMBINATIONS" row.

- If no combination enabled a class to improve to a green color, we report the combinations associated to the highest F1-score values in “BEST COMBINATIONS” row.
- If there are no highlighted values, we let “BEST COMBINATIONS” empty, because none of the combinations of adding synthetic data points, enabled to improve F1-score.



CLASS			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SAMPLES			13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
BEST XGBoost			0.96	0.0	0.841	0.933	0.967	0.955	0.977	0.4	0.985	0.959	0.982	0.933	0.96	0.726	0.871	0.925	0.812
F1-SCORE	SMOTE 12%	ALL	0.933	0.0	0.83	0.813	0.966	**0.972**	**0.978**	**0.733**	0.985	**0.978**	0.948	0.903	0.96	0.668	0.866	**0.934**	**0.876**
		CRITICAL	0.933	0.0	0.812	0.813	0.964	**0.961**	**0.989**	**0.7**	0.985	**0.968**	0.964	0.921	0.96	0.674	0.851	**0.944**	**0.858**
	SMOTE 35%	ALL	0.933	0.0	0.83	0.833	0.964	**0.961**	0.977	**0.7**	0.985	**0.973**	0.926	0.903	0.931	0.709	0.856	**0.934**	**0.832**
		CRITICAL	0.933	0.0	0.765	0.9	0.954	**0.96**	**0.978**	**0.5**	0.985	**0.973**	0.92	0.931	0.96	0.68	0.869	0.917	**0.876**
	SMOTE 50%	ALL	0.933	0.0	0.743	0.9	0.953	**0.972**	0.977	**0.5**	0.985	**0.978**	0.942	0.931	0.96	0.682	0.866	**0.93**	**0.86**
		CRITICAL	0.905	0.0	0.83	0.88	0.964	**0.967**	0.977	0.3	0.985	**0.967**	0.942	0.903	0.931	0.632	0.861	0.905	**0.876**
	SMOTE 65%	ALL	0.933	0.0	0.83	0.88	0.964	0.955	**0.983**	**0.5**	0.985	**0.973**	0.942	0.865	0.931	0.595	0.869	0.889	**0.843**
		CRITICAL	0.933	0.0	0.711	0.88	0.949	**0.972**	**0.988**	**0.5**	0.981	**0.978**	0.926	**0.96**	0.96	0.643	0.857	0.898	**0.86**
BEST COMBINATIONS			-	-	-	-	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	CRITICAL 12% ALL 12% ALL 35% CRITICAL 65% ALL 65%	CRITICAL 12%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% CRITICAL 65% ALL 65%	-	ALL 65%	-	-	-	CRITICAL 12% ALL 12% CRITICAL 35%	CRITICAL 12% ALL 35% ALL 50%

Figure 5.15: Per-class “BEST COMBINATIONS” for SMOTE on XGBoost.



CLASS			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SAMPLES			13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
BEST SVC			1.0	0.0	0.853	0.793	0.965	0.966	0.966	0.733	0.981	0.979	0.982	0.905	1.0	0.753	0.859	0.931	0.858
F1-SCORE	SMOTE 12%	ALL	0.867	**0.1**	0.815	**0.933**	0.957	**0.972**	**0.972**	0.533	0.974	**0.984**	0.982	0.855	0.96	0.706	0.816	**0.941**	**0.883**
		CRITICAL	0.867	**0.133**	0.843	**0.933**	0.958	**0.972**	**0.972**	0.7	0.977	**0.984**	0.982	**0.971**	1.0	0.691	0.834	**0.941**	**0.878**
	SMOTE 35%	ALL	0.933	**0.1**	0.798	**0.933**	0.956	**0.972**	0.966	0.7	0.981	**0.989**	0.982	0.819	0.92	0.739	0.816	**0.944**	**0.878**
		CRITICAL	0.933	**0.08**	0.773	**0.933**	0.95	**0.972**	**0.983**	0.7	0.977	**0.989**	0.982	0.855	0.971	0.727	0.813	**0.944**	**0.865**
	SMOTE 50%	ALL	0.867	**0.1**	0.705	**0.933**	0.948	**0.972**	**0.971**	0.7	0.977	**0.989**	0.982	0.819	0.92	0.753	0.806	**0.944**	**0.878**
		CRITICAL	0.933	**0.08**	0.785	**0.933**	0.956	0.966	0.966	0.7	0.981	**0.983**	**1.0**	0.836	0.971	0.699	0.815	**0.94**	**0.891**
	SMOTE 65%	ALL	0.867	**0.08**	0.798	**0.9**	0.958	**0.977**	**0.971**	0.7	0.977	**0.989**	0.982	0.819	0.92	0.602	0.826	0.879	**0.878**
		CRITICAL	0.933	**0.1**	0.692	**0.96**	0.952	0.961	**0.972**	0.7	0.981	**0.983**	**1.0**	0.864	0.971	0.713	0.812	**0.933**	**0.878**
BEST COMBINATIONS			-	ALL 12%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% CRITICAL 65%	CRITICAL 12% ALL 12% ALL 35% CRITICAL 50% CRITICAL 65% ALL 65%	-	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% CRITICAL 65% ALL 65%	ALL 50% ALL 65%	ALL 12%	-	-	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	ALL 50%

Figure 5.16: Per-class “BEST COMBINATIONS” for SMOTE on SVC.

CLASS		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SAMPLES		13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
BEST ANN		0.853	0.0	0.769	0.667	0.957	0.955	0.966	0.733	0.974	0.969	0.964	0.838	0.893	0.751	0.818	0.937	0.827
F1-SCORE SMOTE 12%	ALL	**0.893**	0.0	**0.807**	**0.833**	0.949	**0.978**	0.96	0.6	**0.981**	**0.979**	0.922	0.81	0.86	0.696	0.804	0.935	**0.839**
	CRITICAL	**0.893**	**0.2**	0.751	**0.767**	0.939	**0.971**	0.961	0.514	**0.985**	0.957	0.932	0.83	0.831	0.67	**0.824**	0.931	**0.886**
SMOTE 35%	ALL	**0.893**	**0.08**	0.726	**0.767**	0.936	**0.96**	0.96	0.7	**0.988**	0.963	0.913	0.771	0.831	0.671	0.809	0.931	**0.871**
	CRITICAL	**0.893**	**0.213**	0.7	**0.8**	0.935	**0.971**	0.966	0.7	**0.984**	0.946	0.92	0.804	**0.9**	0.709	**0.841**	0.929	0.801
SMOTE 50%	ALL	0.767	**0.1**	0.762	**0.733**	0.935	0.951	0.966	0.633	**0.977**	**0.984**	0.938	0.771	**0.92**	0.707	**0.825**	0.916	**0.847**
	CRITICAL	0.853	**0.08**	0.683	**0.767**	0.952	0.955	**0.977**	0.7	**0.977**	**0.973**	0.923	0.762	0.86	0.708	0.792	0.933	**0.837**
SMOTE 65%	ALL	0.827	0.0	0.757	**0.693**	0.925	0.951	**0.971**	0.567	0.973	0.944	0.92	0.771	0.86	0.615	0.776	0.903	0.812
	CRITICAL	**0.893**	0.0	**0.788**	**0.9**	0.934	0.95	**0.971**	0.467	**0.981**	0.966	0.904	0.743	0.86	0.672	**0.832**	0.901	0.782
BEST COMBINATIONS		CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% ALL 65%	ALL 35%	CRITICAL 12%	ALL 65%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35%	ALL 50% CRITICAL 65% ALL 65%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	CRITICAL 12% CRITICAL 50% ALL 50%	-	-	CRITICAL 50%	-	ALL 35%	-	ALL 12%

Figure 5.17: Per-class “BEST COMBINATIONS” for SMOTE on ANN.

From these results we can observe that, generally, adding synthetic data to the train set is improving the per-class F1-score values. In particular, we highlight the following:

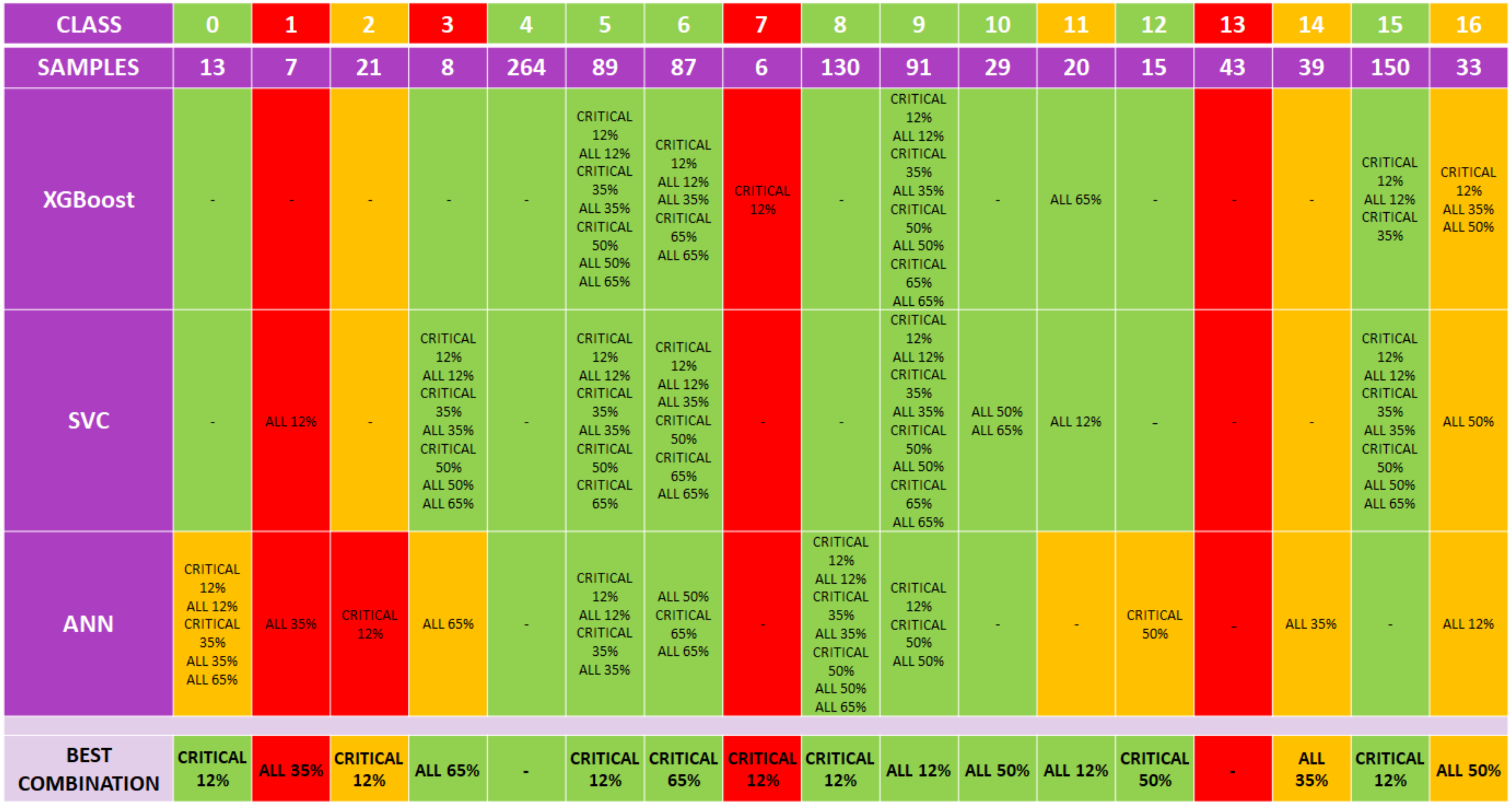
- **XGBoost**: found "BEST COMBINATIONS" for 7 out of 17 classes. Considering critical classes, i.e., the ones in ORANGE and RED in row "CLASS", we improved the F1-score values of:
  - *class 7*: from 0.4 (RED) to 0.733 (RED).
  - *class 15*: from 0.925 (ORANGE) to 0.944 (GREEN).
  - *class 16*: from 0.812 (RED) to 0.876 (ORANGE).

  As a result, the number of critical classes for XGBoost is reduced from 7 to 6.
- **SVC**: found "BEST COMBINATIONS" for 9 out of 17 classes. Considering the critical classes, i.e., the ones in ORANGE and RED in row "CLASS", we improved the F1-score values of:
  - *class 1*: from 0.0 (RED) to 0.133 (RED).
  - *class 3*: from 0.793 (RED) to 0.96 (GREEN).
  - *class 11*: from 0.905 (ORANGE) to 0.971 (GREEN).
  - *class 16*: from 0.858 (ORANGE) to 0.891 (ORANGE).

  As a result, the number of critical classes for SVC is reduced from 8 to 6.
- **ANN**: we found "BEST COMBINATIONS" for 11 out of 17 classes. Considering the critical classes, i.e., the ones in ORANGE and RED in row "CLASS", we improved the F1-score values of:
  - *class 0*: from 0.853 (ORANGE) to 0.893 (ORANGE).
  - *class 1*: from 0.0 (RED) to 0.213 (RED).
  - *class 2*: from 0.769 (RED) to 0.807 (RED).
  - *class 3*: from 0.667 (RED) to 0.9 (ORANGE).
  - *class 12*: from 0.893 (ORANGE) to 0.92 (ORANGE).
  - *class 14*: from 0.818 (ORANGE) to 0.841 (ORANGE).
  - *class 16*: from 0.827 (ORANGE) to 0.886 (ORANGE).

  As a result, the number of critical classes for ANN is not changed, but we have significant improvements on F1-score of critical classes.

This analysis allowed us to looked deeper into SMOTE methodology, understanding that it is a valid methodology to address the data scarcity problem, but to give better and satisfactory results, one must carefully choose the classes on which to generate the synthetic data and also understand the amount of synthetic data to be generated on those classes. The evidence that for each classifier there are combinations of strategies and percentages that allow for better results on the F1-score of critical classes, prompted us to ask whether it is possible to find for each class a combination of strategy and percentage that allows for improvements regardless of the type of classifier used, a.k.a "BEST COMBINATION".



CLASS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
SAMPLES	13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
XGBoost	-	-	-	-	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	CRITICAL 12% ALL 12% ALL 35% CRITICAL 65% ALL 65%	CRITICAL 12%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% CRITICAL 65% ALL 65%	-	ALL 65%	-	-	-	CRITICAL 12% ALL 12% CRITICAL 35%	CRITICAL 12% ALL 35% ALL 50%
SVC	-	ALL 12%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% CRITICAL 65%	CRITICAL 12% ALL 12% ALL 35% CRITICAL 50% CRITICAL 65% ALL 65%	-	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% CRITICAL 65% ALL 65%	ALL 50% ALL 65%	ALL 12%	-	-	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	ALL 50%
ANN	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% ALL 65%	ALL 35%	CRITICAL 12%	ALL 65%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35%	ALL 50% CRITICAL 65% ALL 65%	-	CRITICAL 12% ALL 12% CRITICAL 35% ALL 35% CRITICAL 50% ALL 50% ALL 65%	CRITICAL 12% CRITICAL 50% ALL 50%	-	-	CRITICAL 50%	-	ALL 35%	-	ALL 12%
BEST COMBINATION	**CRITICAL 12%**	**ALL 35%**	**CRITICAL 12%**	**ALL 65%**	**-**	**CRITICAL 12%**	**CRITICAL 65%**	**CRITICAL 12%**	**CRITICAL 12%**	**ALL 12%**	**ALL 50%**	**ALL 12%**	**CRITICAL 50%**	**-**	**ALL 35%**	**CRITICAL 12%**	**ALL 50%**

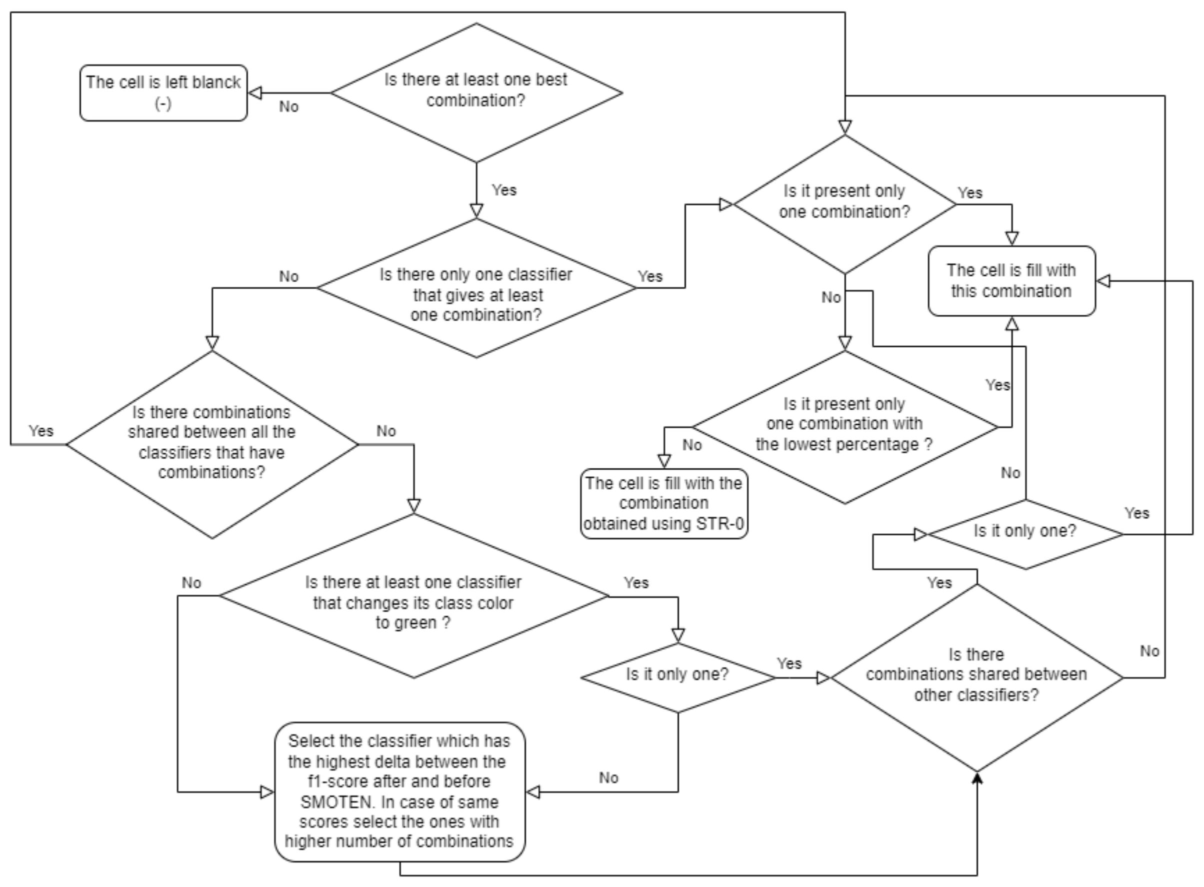
Figure 5.18: SMOTE best combination found for each class and new global class color.

Figure 5.18 summarizes, in the rows “XGBoost”, “SVC” and “ANN”, the “BEST COMBINATIONS” we obtained with the respective color. From the color assigned in row “BEST COMBINATION”, we can notice that with this procedure we were able to reclassify the classes with these new global class colors (previous global class colors are the ones reported in row “CLASS”):

- **RED**: classes 1, 7, and 13.
- **ORANGE**: classes 2, 14, and 16.
- **GREEN**: classes 0, 3, 4, 5, 6, 8, 9, 10, 11, 12, and 15.

Comparing the previous with the new global class colors, we can appreciate that the number of critical classes is decreased from 8 to 6.

In row “BEST COMBINATION” we report the best combinations we found using the criteria in Figure 5.19.



The cell is left blanck (-)
No
Is there at least one best combination?
Yes
No
Is there only one classifier that gives at least one combination?
Yes
Is it present only one combination?
Yes
The cell is fill with this combination
No
Yes
Is there combinations shared between all the classifiers that have combinations?
No
Is it present only one combination with the lowest percentage ?
No
Yes
The cell is fill with the combination obtained using STR-0
No
Is it only one?
Yes
No
Is there at least one classifier that changes its class color to green ?
Yes
Yes
Is it only one?
Yes
Is there combinations shared between other classifiers?
No
Select the classifier which has the highest delta between the f1-score after and before SMOTEN. In case of same scores select the ones with higher number of combinations
No

Figure 5.19: Schema on how to choose the best combination for each class.

This procedure aimed to identify the best per-class combination. Once they have been identified, for each class we first read the strategy found, to understand if on a class we needed to generate synthetic data or not (Figure 5.11), and than read the percentages of data which in the case should be generated. For example, for class 0 the best combination is “Critical-classes 12%” and this tell us that we do not need to generate synthetics data for class 0 as it is not a critical class and so it is not considered by “Critical-classes” strategy. Instead, for class 1 the best combination found is “All-classes 35%” and since “All-classes” strategy considers class 1, we need to generate synthetic data on that class with a percentage of “35%”, so bringing the number of data points in the train set to 93. Taking this reasoning over all classes we obtained the following results:

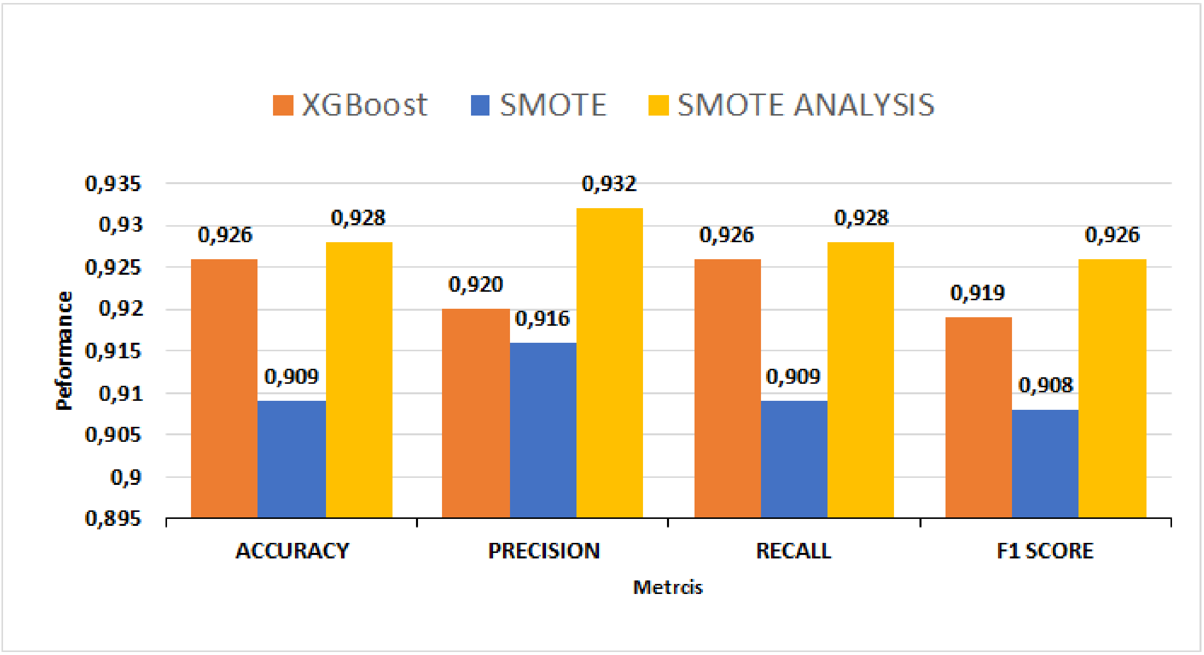


Figure 5.20: Accuracy, Precision, Recall and F1-score results obtained training the Best Baseline “XGBoost” models without synthetic data, with synthetic data using “SMOTE” and with synthetic data using SMOTE with the best combination found “SMOTE ANALYSIS”.

	CLASS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	DATA POINTS	13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
F1-SCORE	XGB	0.833	0.0	0.804	0.6	0.961	0.936	0.972	0.267	0.985	0.968	0.935	0.876	0.96	0.709	0.833	0.917	0.857
	SMOTE	0.833	0.0	0.717	0.8	0.932	0.936	0.977	0.48	0.985	0.978	0.944	0.921	0.971	0.617	0.841	0.897	0.844
	SMOTE ANALYSIS	0.833	0.0	0.822	0.933	0.957	0.942	0.977	0.533	0.985	0.978	0.935	0.921	1.0	0.718	0.859	0.914	0.857

Figure 5.21: Per-class F1-score values obtained training the Best Baseline “XGBoost” models without synthetic data, with synthetic data using “SMOTE” and with synthetic data using SMOTE with the best combination found “SMOTE ANALYSIS”.

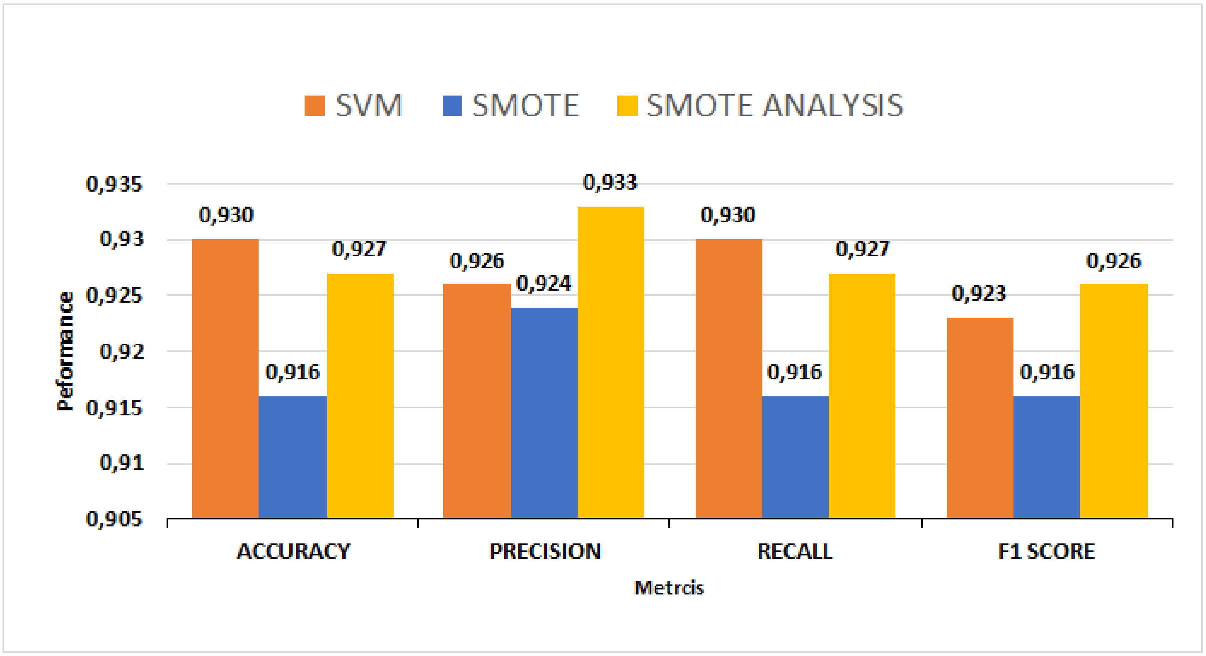
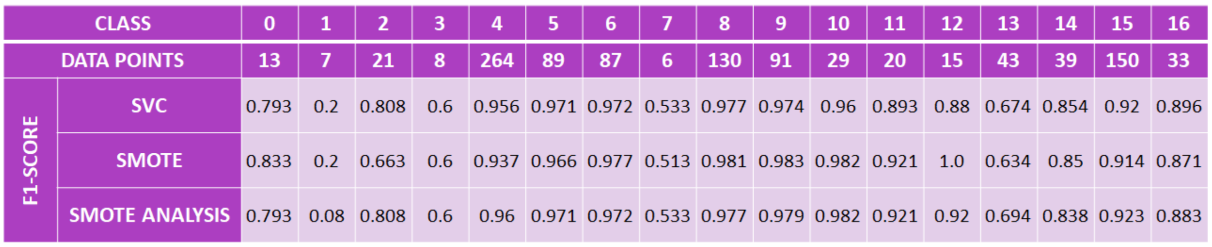


Figure 5.22: Accuracy, Precision, Recall and F1-score results obtained training the Best Baseline “SVC” models without synthetic data, with synthetic data using “SMOTE” and with synthetic data using SMOTE with the best combination found “SMOTE ANALYSIS”.



	CLASS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	DATA POINTS	13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
F1-SCORE	SVC	0.793	0.2	0.808	0.6	0.956	0.971	0.972	0.533	0.977	0.974	0.96	0.893	0.88	0.674	0.854	0.92	0.896
	SMOTE	0.833	0.2	0.663	0.6	0.937	0.966	0.977	0.513	0.981	0.983	0.982	0.921	1.0	0.634	0.85	0.914	0.871
	SMOTE ANALYSIS	0.793	0.08	0.808	0.6	0.96	0.971	0.972	0.533	0.977	0.979	0.982	0.921	0.92	0.694	0.838	0.923	0.883

Figure 5.23: Per-class F1-score values obtained training the Best Baseline "SVC" models without synthetic data, with synthetic data using "SMOTE" and with synthetic data using SMOTE with the best combination found "SMOTE ANALYSIS".

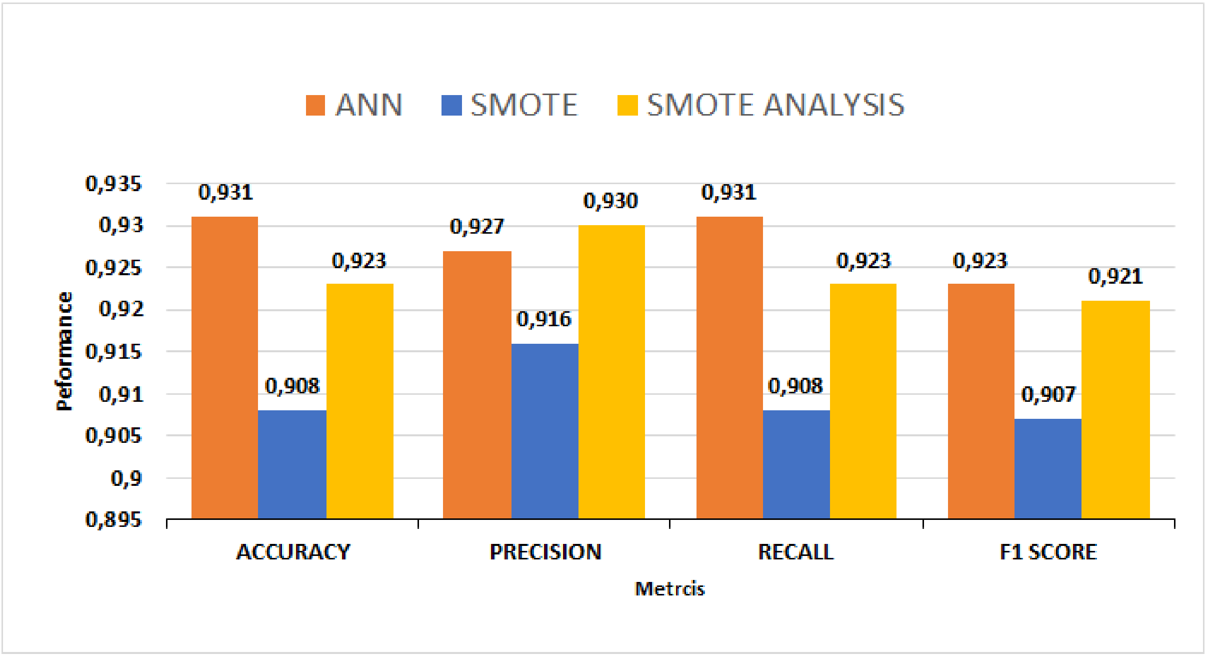
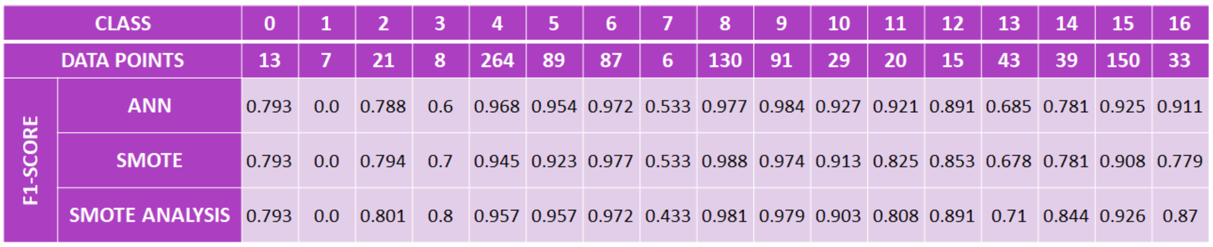


Figure 5.24: Accuracy, Precision, Recall and F1-score results obtained training the Best Baseline "ANN" models without synthetic data, with synthetic data using "SMOTE" and with synthetic data using SMOTE with the best combination found "SMOTE ANALYSIS".



	CLASS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	DATA POINTS	13	7	21	8	264	89	87	6	130	91	29	20	15	43	39	150	33
F1-SCORE	ANN	0.793	0.0	0.788	0.6	0.968	0.954	0.972	0.533	0.977	0.984	0.927	0.921	0.891	0.685	0.781	0.925	0.911
	SMOTE	0.793	0.0	0.794	0.7	0.945	0.923	0.977	0.533	0.988	0.974	0.913	0.825	0.853	0.678	0.781	0.908	0.779
	SMOTE ANALYSIS	0.793	0.0	0.801	0.8	0.957	0.957	0.972	0.433	0.981	0.979	0.903	0.808	0.891	0.71	0.844	0.926	0.87

Figure 5.25: Per-class F1-score values obtained training the Best Baseline "ANN" models without synthetic data, with synthetic data using "SMOTE" and with synthetic data using SMOTE with the best combination found "SMOTE ANALYSIS".

Looking at these final results, we can conclude by stating that generating synthetic data through SMOTE helps to improve classification on classes affected by data scarcity, as long as we understand on which classes and with what percentages to generate the synthetic data. In particular we have the following improvements:

- for XGBoost we obtain these per-class F1-score improvement on the classes we had identified as critical:
  - *2*: from 80.4% to 82.2%, corresponding to an increase of 1.8%.
  - *3*: from 60% to 93.3%, corresponding to an increase of 33.3%.
  - *7*: from 26.7% to 53.3%, corresponding to an increase of 26.6%.
  - *11*: from 87.6% to 92.1%, corresponding to an increase of 4.5%.
  - *13*: from 70.9% to 71.8%, corresponding to an increase of 1.9%.
  - *14*: from 83.3% to 85.9%, corresponding to an increase of 2.6%.
- for SVC we obtain these per-class F1-score improvement on the classes we had identified as critical:
  - *11*: from 89.3% to 92.1%, corresponding to an increase of 2.8%.
  - *13*: from 67.4% to 69.4%, corresponding to an increase of 2%.
- for ANN we obtain these per-class F1-score improvement on the classes we had identified as critical:
  - *2*: from 78.8% to 80.1%, corresponding to an increase of 1.3%.
  - *3*: from 60% to 80.0%, corresponding to an increase of 20.0%.
  - *13*: from 68.5% to 71%, corresponding to an increase of 2.5%.

  - *14*: from 78.1% to 84.4%, corresponding to an increase of 6.3%.

These results show us how our proposed procedure for selecting classes and percentages actually enhances the overall results, improving the F1-score of the classes characterized by a low number of data points and thus of those affected by data scarcity.

6 | Conclusion and Future Developments

In this thesis, we investigated the problem of data scarcity when performing failure management in microwave networks using Machine Learning (ML). More specifically, we considered data from a real microwave network, and performed failure-cause identification, i.e., aim at discriminating different causes of hardware failures given the status of equipment failure in a certain time frame. Motivated by the poor classification performance, obtained with state of art ML classifiers, in particular for some failure classes scarcely represented in the dataset, we aim at improving classification performance by adopting different strategies, such as SMOTE, Transfer Learning (TL), Auxiliary Task Learning (ATL), and Denoising Autoencoders (DAE).

The main takeaways from the use of these methodologies are:

1. The use of TL or DAE does not improve the data scarcity problem. In fact, with TL we obtained an F1-score of 78.8% and with DAE an F1-score of 84.5%, while with the baseline classifier we obtained an F1-score of 89.4%.

2. The use of SMOTE or ATL does improve the data scarcity problem. In fact, with SMOTE we obtained an F1-score of 92.6% and with ATL an F1-score of 92.3%, while with the baseline classifier we obtained an F1-score of 89.4%.

3. Analyzing the per-class F1-score obtained with SMOTE and ATL, we observed that with SMOTE we have higher F1-score improvement on classes characterized by few points than with using ATL.

Overall, the results show that SMOTE outperforms all other methodologies, so as last contribution of this work we have done an in-depth analysis of this methodology. Using per-class F1-score as metric, we proposed a method to identify the critical classes on which to go and generate the synthetic data and then defined two strategies to be applied to SMOTE. The first generates synthetic data only on the classes identified as critical, while the second does not distinguish between critical and noncritical classes. Then, we proposed different synthetic data generation percentages, and by combining the strategies and percentages we obtained new performance results. The main takeaways from this analysis are:

1. Regardless of the ML classifier considered, in most cases synthetic data help to improve the performance of this classifier.
2. There are combinations of strategies and percentages that allow to achieve better results than apply SMOTE without the use these combinations.
3. Generally, generating synthetics data on all the classes performs better than generating it only on critical classes. This tells us that to improve the overall performance of the classifiers, in generating the synthetic data it is not enough to consider only the critical classes but it is necessary to consider all classes, in order not to get the dataset unbalanced again towards the critical classes.
4. Generally, the highest performance is obtained when considering low percentages of synthetic data while moving to higher percentages the performance drops.

Based on these takeaways, we proposed a method for identifying on which classes to generate synthetic data and what amount of synthetic data to generate on them, ragardless the classifiers adopted. From these results we concluded by stating that generating synthetic data through SMOTE helps to improve classification on classes affected by data scarcity, as long as we understand on which classes and with what percentages to generate the synthetic data. Considering the final per-class F1-score results, we found that the model that experiences the most improvement using our strategy is XGBoost, for which there are F1-score improvements on 6 of the 8 identified critical classes.

As possible future work, we plan to investigate the following research directions:

- Evaluate the use of other variants of SMOTE, such as SVMSMOTE, KMeansSMOTE, BorderlineSMOTE, and similar other approaches.
- Combine the different methodologies used in this work, such as SMOTE and Auxiliary-Task Learning to aim improving further the classification performance.
- Evaluate the use of other methodologies to generate synthetics data, such as Generative Adversarial Networks and Variational Autoencoders.
- Make use of XAI (eXplainable AI) frameworks, as SHARP and LIME, to investigate the root-cause of misclassified classes. XAI can be used to develop interpretable models that can help users understand how the model works and how it arrived at its predictions. This can be used to create explanations for the original data, that

can be used to create new examples, through a data augmentation technique, that are similar to the original data.

Bibliography

[1] Ieee standard for definitions of terms for antennas. *IEEE Std 145-2013 (Revision of IEEE Std 145-1993)*, pages 1–50, 2014. doi: 10.1109/IEEESTD.2014.6758443.

[2] H. B. Atya, O. Rajchert, L. Goshen, and M. Freiman. Non parametric data augmentations improve deep-learning based brain tumor segmentation. In *2021 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, pages 357–360, 2021. doi: 10.1109/COMCAS52219.2021.9629083.

[3] O. Ayoub, N. Di Cicco, F. Ezzeddine, F. Bruschetta, R. Rubino, M. Nardecchia, M. Milano, F. Musumeci, C. Passera, and M. Tornatore. Explainable artificial intelligence in communication networks: A use case for failure identification in microwave networks. *Computer Networks*, 219:109466, 2022. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2022.109466. URL `https://www.sciencedirect.com/science/article/pii/S138912862200500X`.

[4] O. Ayoub, F. Musumeci, F. Ezzeddine, C. Passera, and M. Tornatore. On using explainable artificial intelligence for failure identification in microwave networks. In *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pages 48–55, 2022. doi: 10.1109/ICIN53892.2022.9758095.

[5] E. Balevi and R. D. Gitlin. Unsupervised machine learning in 5g networks for low latency communications. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, pages 1–2, 2017. doi: 10.1109/PCCC.2017.8280492.

[6] Y. Bao and S. Yang. Two novel smote methods for solving imbalanced classification problems. *IEEE Access*, 11:5816–5823, 2023. doi: 10.1109/ACCESS.2023.3236794.

[7] I. A. Bartsiokas, P. K. Gkonis, D. I. Kaklamani, and I. S. Venieris. Ml-based radio resource management in 5g and beyond networks: A survey. *IEEE Access*, 10:83507–83528, 2022. doi: 10.1109/ACCESS.2022.3196657.

[8] Y. Bengio. Learning deep architectures for ai. *Foundations*, 2:1–55, 01 2009. doi: 10.1561/2200000006.

[9] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. Algorithms for hyper-parameter optimization.

[10] A. Bouchachia. On the scarcity of labeled data. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 1, pages 402–407, 2005. doi: 10.1109/CIMCA.2005.1631299.

[11] F. Bre, J. M. Gimenez, and V. D. Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. 2017.

[12] A. Burkov. *Machine Learning Engineering*. True Positive Inc., 2020.

[13] P. Casas, P. Fiadino, and A. D'Alconzo. Machine-learning based approaches for anomaly detection and classification in cellular networks. 04 2016.

[14] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. 2011. doi: 10.1613/jair.953.

[15] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. 2016.

[16] Commscope. *Microwave communication basics*. Commscope, 2017.

[17] DAEnotes. Digital microwave communication equipment. URL `https://www.daenotes.com/electronics/microwave-radar/digital-microwave-communication-equipment`.

[18] S. Danesh, A. Araghi, M. Khalily, P. Xiao, and R. Tafazolli. Millimeter wave phased array antenna synthesis using a machine learning technique for different 5g applications. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–5, 2020. doi: 10.1109/ISNCC49221.2020.9297196.

[19] fchollet. The functional api, 2019-2020. URL `https://keras.io/guides/functional_api/`.

[20] R. Guo, Z. Lin, M. Li, F. Yang, S. Xu, and A. Abubakar. A nonlinear model compression scheme based on variational autoencoder for microwave data inversion. *IEEE Transactions on Antennas and Propagation*, 70(11):11059–11069, 2022. doi: 10.1109/TAP.2022.3195553.

[21] W. Guo, X. Zha, K. Qian, and T. Chen. Can active learning benefit the smart grid? a perspective on overcoming the data scarcity. In *2019 IEEE 2nd International*

Conference on Electronics and Communication Engineering (ICECE), pages 346–350, 2019. doi: 10.1109/ICECE48499.2019.9058539.

[22] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009. doi: 10.1109/TKDE.2008.239.

[23] S. Hu, Y. Liang, L. Ma, and Y. He. Msmote: Improving classification performance when training data is imbalanced. In *2009 Second International Workshop on Computer Science and Engineering*, volume 2, pages 13–17, 2009. doi: 10.1109/WCSE.2009.756.

[24] T. imbalanced-learn developers. Over-sampling, 2019-2020. URL `https://imbalanced-learn.org/stable/over_sampling.html#smote-adasyn`.

[25] M. A. Jabin, Q. Liu, and M. P. Fok. Generative adversarial network for data augmentation in photonic-based microwave frequency measurement. In *2022 IEEE International Topical Meeting on Microwave Photonics (MWP)*, pages 1–4, 2022. doi: 10.1109/MWP54208.2022.9997615.

[26] P. Kachurka and V. Golovko. Neural network approach to real-time network intrusion detection and recognition. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, volume 1, pages 393–397, 2011. doi: 10.1109/IDAACS.2011.6072781.

[27] Keras. Model training apis, 2023. URL `https://keras.io/api/models/model_training_apis/`.

[28] Keras. Dropout layer, 2023. URL `https://keras.io/api/layers/regularization_layers/dropout/`.

[29] Keras. Layer weight regularizers, 2023. URL `https://keras.io/api/layers/regularizers/`.

[30] Keras. Adam, 2023. URL `https://keras.io/api/optimizers/adam/`.

[31] R. V. Kulkarni, S. H. Patil, and R. Subhashini. An overview of learning in data streams with label scarcity. In *2016 International Conference on Inventive Computation Technologies (ICICT)*, volume 2, pages 1–6, 2016. doi: 10.1109/INVENTIVE.2016.7824874.

[32] S. Lavdas, P. Gkonis, Z. Zinonos, P. Trakadas, and L. Sarakis. Throughput based adaptive beamforming in 5g millimeter wave massive mimo cellular networks via

machine learning. In *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, pages 1–7, 2022. doi: 10.1109/VTC2022-Spring54318.2022.9860566.

[33] S. learn developers. Cross-validation: evaluating estimator performance. URL `https://scikit-learn.org/stable/modules/cross_validation.html`.

[34] L. Liebel and M. Körner. Auxiliary tasks in multi-task learning, 2018.

[35] K. Limthong and T. Tawsook. Network traffic anomaly detection using machine learning approaches. In *2012 IEEE Network Operations and Management Symposium*, pages 542–545, 2012. doi: 10.1109/NOMS.2012.6211951.

[36] F. Lin, J. Chen, G. Ding, Y. Jiao, J. Sun, and H. Wang. Spectrum prediction based on gan and deep transfer learning: A cross-band data augmentation framework. *China Communications*, 18(1):18–32, 2021. doi: 10.23919/JCC.2021.01.002.

[37] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering*, 35(1):857–876, 2023. doi: 10.1109/TKDE.2021.3090866.

[38] S. Microelettronica, 1952. URL `https://www.siaemic.com/index.php`.

[39] F. Musumeci, L. Magni, O. Ayoub, R. Rubino, M. Capacchione, G. Rigamonti, M. Milano, C. Passera, and M. Tornatore. Supervised and semi-supervised learning for failure identification in microwave networks. *IEEE Transactions on Network and Service Management*, 18(2):1934–1945, 2021. doi: 10.1109/TNSM.2020.3039938.

[40] OpenCV. Introduction to support vector machines. URL `https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`.

[41] L. Pan, J. Zhang, P. P. Lee, M. Kalander, J. Ye, and P. Wang. Proactive microwave link anomaly detection in cellular data networks. *Computer Networks*, 167:106969, 2020. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2019.106969. URL `https://www.sciencedirect.com/science/article/pii/S1389128619307662`.

[42] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi: 10.1109/TKDE.2009.191.

[43] K. Peng and F. Xu. Optimization of antenna performance based on vae-bpnn-pca. In *2022 International Conference on Microwave and Millimeter Wave Technology (ICMMT)*, pages 1–3, 2022. doi: 10.1109/ICMMT55580.2022.10023171.

[44] R. Rustogi and A. Prasad. Swift imbalance data classification using smote and

extreme learning machine. In *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, pages 1–6, 2019. doi: 10.1109/ICCIDS.2019.8862112.

[45] W. S. Sarle. Neural networks and statistical models. SAS Users Group International Conference, 1994.

[46] scikit-learn developers. sklearn.svm.svc, 2007 - 2023. URL `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`.

[47] S. Sridhar and S. Sanagavarapu. Handling data imbalance in predictive maintenance for machines using smote-based oversampling. In *2021 13th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 44–49, 2021. doi: 10.1109/CICN51697.2021.9574668.

[48] C. Srinilta and S. Kanharattanachai. Application of natural neighbor-based algorithm on oversampling smote algorithms. In *2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, pages 217–220, 2021. doi: 10.1109/ICEAST52143.2021.9426310.

[49] M. Steer. *Fundamentals of Microwave and RF Design (Third Edition)*. NC State University, 2019. doi: https//doi.org/10.5149/9781469656892.

[50] S. Sönmez, I. Shayea, S. A. Khan, and A. Alhammadi. Handover management for next-generation wireless networks: A brief overview. In *2020 IEEE Microwave Theory and Techniques in Wireless Communications (MTTW)*, volume 1, pages 35–40, 2020. doi: 10.1109/MTTW51045.2020.9245065.

[51] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. A survey on deep transfer learning, 2018.

[52] T. Tandel, O. Ayoub, F. Musumeci, C. Passera, and M. Tornatore. Federated-learning-assisted failure-cause identification in microwave networks. In *2022 12th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 1–7, 2022. doi: 10.1109/RNDM55901.2022.9927592.

[53] K.-H. Thung and C.-Y. Wee. A brief review on multi-task learning. 2018. doi: 10.1007/s11042-018-6463-x.

[54] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. pages 1096–1103, 01 2008. doi: 10.1145/1390156.1390294.

[55] B. Wang, H. Yang, Q. Yao, A. Yu, T. Hong, J. Zhang, M. Kadoch, and M. Cheriet. Hopfield neural network-based fault location in wireless and optical networks for smart city iot. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1696–1701, 2019. doi: 10.1109/IWCMC.2019.8766627.

[56] xgboost developers. Xgboost parameters, 2022. URL `https://xgboost.readthedocs.io/en/stable/parameter.html`.

[57] Y. Xu, X. Cheng, W. Ke, Q.-X. Zhu, Y.-L. He, and Y. Zhang. Smote-based fault diagnosis method for unbalanced samples. In *2022 IEEE 11th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 682–686, 2022. doi: 10.1109/DDCLS55054.2022.9858365.

[58] S. Zhang, X. Hu, Z. Liu, L. Sun, K. Han, W. Wang, and F. M. Ghannouchi. Deep neural network behavioral modeling based on transfer learning for broadband wireless power amplifier. *IEEE Microwave and Wireless Components Letters*, 31(7):917–920, 2021. doi: 10.1109/LMWC.2021.3078459.

[59] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning, 2019.

List of Figures

3.1 The basic components of a microwave link 11
3.2 Split-Mount Microwave Equipment installation 13
3.3 Generative, Contrastive, and Generative-Contrastive Self-supervised learning 16
3.4 Binary Classification 17
3.5 Multiclass Classification 17
3.6 Machine Learning project life cycle 18
3.7 K-Fold Cross-Validation [33]. 20
3.8 Confusion Matrix for Multiclass Classification 22
3.9 Decision tree example. 25
3.10 Decision tree ensambles. 26
3.11 Bagging techniques. 27
3.12 Boosting techniques. 28
3.13 Cart (Classification and Regression trees) [15]. 28
3.14 Linear hyperplane between two classes [40]. 30
3.15 Optimal hyperplane between two classes [40]. 31
3.16 Simple non linear perceptron with logistic activation function [45]. 33
3.17 Artificial Neural Network architecture [11]. 34
3.18 Over-sampling generation [24]. 40
3.19 Different learning processes between (a) traditional machine learning and (b) transfer learning [42]. 41
3.20 Different settings of transfer [42]. 43
3.21 Network-based deep transfer learning schema [51]. 44
3.22 Multi-Task Learning Network structure with 3 main-task [53]. 45
3.23 Auxiliary-Task in Multitask-Learning 46
3.24 Denoising autoencoder model architecture. 47

4.1 Portion of the raw input dataset 53
4.2 Portion of the processed dataset 57
4.3 Baseline Models pipeline with hyperparameters optimization. 59
4.4 SMOTE pipeline. 64

4.5 Best Macro Model and its Extractor 66
4.6 Example of Best Transfer Learning Model 67
4.7 Transfer Learning pipeline with hyperparameters optimization. 68
4.8 Example of Auxiliary-Task model 70
4.9 Auxiliary-Task Learning pipeline with hyperparameters optimization. . . . 72
4.10 Denoising Autoencoders Model and its Extractor 74
4.11 Example of Denoising Autoencoder Model. 75
4.12 DAE pipeline with hyperparameters optimization. 76

5.1 Baseline models results 78
5.2 SMOTE models results 79
5.3 Transfer Learning performance results 80
5.4 Auxiliary-Task Learning performance results 81
5.5 Denoising Autoencorders performance results 82
5.6 Auxiliary-Task Learning vs SMOTE performance results 83
5.7 Auxiliary-Task Learning vs SMOTE per-class F1-score values 83
5.8 Best Baseline models results 85
5.9 Best Baseline models per-class F1-score. 85
5.10 Classes color 87
5.11 SMOTE Combinations 90
5.12 XGBoost global results with SMOTE 91
5.13 SVC global results with SMOTE 92
5.14 ANN global results with SMOTE 93
5.15 Per-class "BEST COMBINATIONS" for SMOTE on XGBoost. 96
5.16 Per-class "BEST COMBINATIONS" for SMOTE on SVC. 97
5.17 Per-class "BEST COMBINATIONS" for SMOTE on ANN. 98
5.18 SMOTE best per-class combination 101
5.19 Schema on how to choose the best combination for each class. 102
5.20 XGBoost final results 103
5.21 XGBoost final f1-score values 104
5.22 SVC final results 104
5.23 SVC final f1-score values 105
5.24 ANN final results 105
5.25 ANN final f1-score values 106

List of Tables

4.1 Macro-categories of hardware failures 50
4.2 Micro-categories of hardware failures 51
4.3 Mapping between Macro-category , Micro-category and Category 55
4.4 XGBoost hyperparameters sets 60
4.5 SVC hyperparameters sets 60
4.6 ANN hyperparameters sets 61
4.7 XGBoost baseline model best hyperparameters. 62
4.8 SVC baseline model best hyperparameters. 62
4.9 ANN baseline model best hyperparameters. 62

Acknowledgements

I would like to thank Professor Francesco Musumeci, who gave me the opportunity to work with him and finish my course of study. A big thank you also to Nicola and Mëm who supported me during the development and writing of the thesis.

I want to thank my family who supported and spurred me to always give my best and focus on my priorities during these years of work and study.

I want to thank my girlfriend Valeria who believes in me and my abilities more than I do. She always has faith in me and with her motivation and fortitude she always encourages me to give my best and never give up.

I want to thank all the university colleagues I have met during this journey, for all the days spent together studying or making projects.

Finally, an important thank you goes to all my former work colleagues who always encouraged and motivated me by appreciating my effort as a student worker. Along with them, a big thank you to my former managers and current managers who have always tried to facilitate the reconciliation of my work and university life.