UNIVERSITÀ DEGLI STUDI DI PADOVA

MASTER THESIS

A Machine Learning-based Test Program Quality Tool for Automotive Microcontrollers

Author:
Asma KHEDRI

Supervisors:
Prof. Gian Antonio Susto
Ing. Angelo De Poli
Dr.Ing. Giambattista Carnevale

A thesis submitted in fulfillment of the requirements for the degree of Master

in ICT for Internet and Multimedia

Infineon Technologies
Department of Information Engineering

November 26, 2019

"In god we trust, all others bring data"

— W. Edwards Deming

UNIVERSITÀ DEGLI STUDI DI PADOVA

Abstract

Department of Information Engineering

A Machine Learning-based Test Program Quality Tool for Automotive Microcontrollers

by Asma KHEDRI

In the semiconductor industry where large amounts of data are generated, data driven quality control technologies are gaining increasing importance. In Infineon, production testing is an important aspect in the automotive microcontroller manufacturing, during which thousands of data are stored, the purpose of this thesis is to make use of these data to build a quality gate tool based on machine learning techniques in order to improve testing quality and facilitate better usage of test information for yield improving. In fact, tests in the production flow involves a large number of sequential steps, mainly two important phases, the front-end testing, i.e before packaging, and the back end-testing, after packaging. In this thesis, we study the possibility of predicting the final state of the packaged chips based on the tests done before packaging.

Keywords: Automotive, Semiconductors, Machine-Learning (ML), Production Test, Yield, etc.

Acknowledgements

I would like to take this opportunity to express my heartfelt gratitude to a number of people whose I was blessed to know and work with.

First, to Professor Gian Antonio Susto, my thesis advisor, the one I turn to whenever I run into a trouble spot, you were an incredible mentor to me, your motivational words and insightful suggestions were the reasons behind the success of this thesis.

To Giambattista, Pierre and Andrea, the best team someone could ever work with, thank you for the continuous support, patience, motivation, and immense knowledge. To Pierre, thank you for being optimistic when I first thought my results are not good enough, you always made it easier for me and taught me not to give up. To Andrea, thank you for your enthusiasm for Machine Learning, your brilliance inspired me. To Giambattista, thank you for trusting my skills, for being there every time I needed help, you boosted my confidence and encouraged me to move forward. I have learnt a lot from you all during these months, and for that I am deeply grateful.

To Angelo DePoli, thank you for welcoming me in the MC team in Padua, my experience in Infineon Technologies was a milestone in my career, thank you for offering me the chance to be part of such an interesting thesis work.

To Ms. Roberta Pellizzaro, without you this graduation wouldn't be happening today, thank you.

To my cousin Nadia, words are not enough to thank you, you supported me in every step since I set a foot in Padua.

To my family, far from the eyes but close to the heart, thank you for being patient and strong, for supporting my choices, for being there for me in all my achievements, you taught me to be the person I am today, thank you.

To Amen, the one I always annoyed when I felt anxious and stressed, the one who was there through my ups and downs, thank you.

To my dear friend Mouna, the reason I applied for the University of Padova in the first place, thank you for being my partner in this rest journey.

To anyone who ever offered me a smile and a word of motivation while I was writing this thesis, it meant a lot to me, thank you.

Asma Khedri

Contents

List of Figures

List of Tables

List of Abbreviations

MC	**M**icro **C**ontroller
ML	**M**achine **L**earning
FE	**F**ront **E**nd
BE	**B**ack **E**nd
NVM	**N**on **V**olatile **M**emory
PdM	**Pred**ictive **M**aintenance
VM	**V**irtual **M**etrology
FDC	**F**ault **D**etection
NN	**N**eural **N**etwork
SVM	**S**upport **V**ector **M**achine
IC	**I**ntegrated **C**ircuit
ATE	**A**utomatic **T**est **E**quipment
DUT	**D**evice **U**nder **T**est
HBIN	**H**ard **B**in
SBIN	**S**oft **B**in
TN	**T**est **N**umber
KDE	**K**ernel **D**ensity **E**stimation
KS	**K**olmogorov **S**mirnov
PCA	**P**rincipal **C**omponent **A**nalysis
BRF	**B**alanced **R**andom **F**orest
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
TN	**T**rue **N**egative
FN	**F**alse **N**egative
TPR	**T**rue **P**ositive **R**ate
FPR	**F**alse **P**ositive **R**ate
ROC	**R**eceiver **O**perating **C**haracteristic
CV	**C**ross **V**alidation
QG	**Q**uality **G**ate

For those who answer the call in the middle of the day or night. For those who answer the call from near and from far. For those who answer the call for help with no expectation of personal gain.
This work is dedicated to you all . . .

Chapter 1

Introduction

1.1 Introduction

This thesis was entirely realized during an internship at the Padua Development Center of the company Infineon Technologies, working as a member of the Microcontroller team. The aim of this chapter is to introduce the framework this thesis was conceived in: the company and its philosophy as well as the team itself. It then explains the motivation behind starting this thesis, as well as a review of some related work. Finally, it discusses the overall goal and structure of the thesis.

1.2 Infineon Technologies

Infineon Technologies is a leading innovator in the international semiconductor industry founded in April 1999. It is head quartered in Munich, Germany but, it has 17 Production sites and 35 Research and Development ones scattered all over Europe, the Americas and the Pacific Regions, and more than 40k employees worldwide.

Today the company is working on 4 business areas. Here follows the official presentation given by the company for these 4 areas [38]:

- **Automotive (ATV)**: In the ATV segment, Infineon develops products and solutions for conventional drivetrains while also actively shaping

FIGURE 1.1: Infineon's logo

the keystone trends that define the industry. Demand for our power semiconductors is on an upward path, fueled by the rising number of electronic applications in cars –a trend further accentuated by the growing popularity of electromobility. We are the undisputed market leader in silicon-based IGBTs and IGBTmodules. Our expertise in silicon carbide is also increasingly relevant for automotive power semiconductors. We are paving the way for self-driving cars with our radar sensors and microcontrollers. Positioned as number two in the radar sensor market, we are already noting strong momentum from the proliferation of driver assistance systems. In the long term, radar systems will be fused with other sensor technologies. We are laying the ground work for this by developing products such as LIDAR solutions. With our $AURIX^{TM}$ family, we are also benefiting from the trend towards increased automation. Our products here control electronic systems such as steering and braking, also acting as host controllers to provide functional safety and data security for central computing platforms.

- **Industrial Power Control (IPC)**: The IPC segment specializes in the efficient conversion of electric energy along the entire supply chain – from generation and transmission right through to consumption. Applications here include wind turbines, high-voltage DC transmission systems, energy storage systems, charging infrastructures for electric vehicles, and household appliances. Infineon is the world leader in IGBT-based discrete power semiconductors and power semiconductor modules. To further strengthen this core IPC business, we are aiming for technology leadership in silicon carbide. Complementary product areas are also becoming increasingly important for us, in particular Intelligent Power Modules (IPMs) integrating controllers, drivers and switches to enable digital control capabilities.

- **Power Management and Multimarket (PMM)**: Our PMM segment focuses on power semiconductors for energy management as well as components for wireless infrastructures and mobile devices. PMM also specializes inultrareliable components for applications in industries such as aerospace. Infineon is the clear leader in the global MOSFET market. Our $CoolMOS^{TM}$ and $OptiMOS^{TM}$ families deliver excellent levels of

energy efficiency. We also offer leading-edge solutions based on gallium nitride. In parallel to this product group, we are continuing to expand our portfolio of complementary drivers and controllers. Battery-operated devices are one of the fastest-growing applications for power semiconductors. In the high-frequency and sensor space, we have established a strong technology footprint with MEMS microphones (silicon in particular),time-of-flight sensors for 3D cameras and radar applications. We have already established very successful positions in the respective markets. At the same time, we can apply our expertise in these areas to more and more use cases that are set to gain momentum over the coming years. Key examples here include human-machine interaction (HMI) and facial recognition.

- **Digital and Security Solutioms (DSS)**: The Digital Security Solutions (DSS) segment has over thirty years' experience delivering some of the world's most challenging and large-scale digital security projects. Our success here is built on our wealth of expertise in conventional smart card applications. We are transferring our core skills in payment cards and government documents to the fast-growing field of embedded security applications. As digitalization shapes more and more areas of everyday life, security is becoming a key success factor for applications across industries as diverse as computing, automotive, Industry 4.0 and smart homes. Parallel to its role as an independent business segment, DSS acts as a competence center for our other three segments, supporting their efforts to hardwire security functionality into the irrespective system solutions.

1.2.1 Automotive MC team in Padua

Located in the Development Center of Padova, the Microcontroller (MC) team is part of the product and testing engineering organization of ATV, focusing on embedded Flash of automotive and industrial microcontrollers. The team has different responsibilities:

- To contribute to Non-Volatile Memory (NVM) testing concept, analyzability and manufacturability, validation and analysis planning.
- To provide NVM analysis tools to improve automation.
- To provide embedded firmware and test patterns for productive testing on NVM.

- To perform NVM test-chip analysis for design and technology learning.
- To execute NVM validation and characterization.
- To perform enhanced In-System tests on application conditions.
- To provide Design Validation Reports for Customer presentations.
- To support NVM qualification (product and technology).
- To review test program contents of test package releases (production, qualification, characterization).

To better respond to these tasks, Padua MC team is divided into two subteams:

- **Test Engineering**: this group develops embedded software for analysis and testing. In particular this group is responsible for development of the Firmware and of test program used for performing Flash memory tests.
- **Product Engineering**: this group characterizes Embedded Flash and it validates robustness of non volatile memories. Its purpose is ensuring the quality of memory by validating the requirements of customers, also considering statistical aspect, extending validation to many samples. It does the analysis, it searches the causes that generate the problem to solve, and therefore it offers solutions for the designers and technology experts. The characterization activity tries to push parameters, like temperature, power and system frequencies, to the limits.

This thesis was entirely realized during an internship with the MC team, working as member of the Test engineering team.

1.3 Motivation and Related Work

Semiconductor manufacturing is a complex and lengthy process, during which, voluminous data are generated and collected. This data has received considerable attention from researchers to transfer this complex engineering data into valuable information and knowledge for process improvement and yield enhancement [8]. With the rapid progress in artificial intelligence, machine learning and statistical techniques are widely applied in predicting the outcomes of manufacturing process and measurement tests. In literature, the

large amounts of data generated by electrical tests, measurement of physical parameters, and collection of equipment and process parameters were used for different purposes such as:

- Virtual Metrology (VM) systems
- Predictive Maintenance (PdM) systems
- Fault Detection (FDC) systems

Virtual Metrology

A VM system consists of a mathematical model that estimates a process results based on previous metrology measurements, instead of measuring it practically. In other words, the purpose is to be able to predict these "costly to measure" quantities from readily available fabrication parameters or sensor data that can be used without further costs [37]. There are many advantages to virtual metrology including [3]:

- **Reducing wafer scraps:** process inspection can be performed through VM for every wafer to sustain yield performance
- **Tighter process control:** VM provides a basis to overcome the metrology delay problem for run-to-run control
- **Increasing throughput:** wafer handling from process tool to metrology tool can be reduced and, thus, production cycle time can be shorted.

VM has been approached by using different techniques, both Linear, such as Ordinary Least Square (OLS) and Partial Least Squares (PLS) [19], [20], and Non-Linear, such as Artificial Neural Networks (NNs) [46].

Predictive Maintenance

PdM is a new approach to maintenance management where actions are performed only when necessary, given the fact that PdM systems can statistically assess the health status of a piece of equipment allowing advance detection of pending failures, enabling timely pre-failure interventions, thanks generally to prediction tools based on historical data and statistical inference methods [36].

Different machine learning techniques were applied in literature for PdM systems, such as NNs in [45], the Kalman predictor in [34] and Support Vector Machines in [1] [36].

Fault Detection

FDC system does not predict the future behavior of the tool/process, but aims to detects and classifies different faults in it. FDC systems identify the root cause of the abnormal behavior. This is of particular interest in the everyday work of a semiconductor plant: the root causes of faults in a complex process may be dozens, sometimes hundreds, and even expert process engineers have difficulty understanding the pathology and, therefore, how to properly cope with the faulty process/tool [37]. The FDC systems employ classification techniques, for example K-NearestNeighbour (kNN) in [16], Principal Component-based kNN in [17] and Support Vector Machines (SVMs) in [28]

In following, we describe how ML techniques are used in this thesis, the goal we want to reach and the challenges faced.

1.4 Problem Statement

The semiconductor manufacturing process, from the first stage up to final product shipping, is interleaved with the testing of the product itself. There are mainly two phases:

- **The front-end process (FE)**: in this step the chips (IC) are patterned and projected on the silicon wafer. In this stage, chips are referred to as silicon chips or dies, that will be later inside a final package/chip. However, before the wafer is sent to chip preparation, it goes through **FE testing**, where every single die on the wafer is tested, and is accessed through specialized prober machines (Automatic Test Equipment (ATE)).
- **The back-end process (BE)**: is mainly cutting and packaging of the wafer to single protected chips. Once packaged the chip goes through other tests before final shipping, which we refer to as **BE testing**, in this stage the chip is accessed trough its standard I/O lines (pads or pins), the same ones which will be used during its working life.

Throughout the entire process, a microcontroller goes from being a circuit integrated on a wafer, i.e. die, to a packaged chip, as seen in Figure 1.2. Each chip has specific (X,Y) coordinates on a wafer, and every wafer has a unique ID. Usually wafers are organized in groups of 25 or 50, called lots, and each

FIGURE 1.2: From wafer to chip: The process start with a wafer with hundreds of ICs, that goes through FE testing, the dies who pass the FE are packaged and sent to BE testing, the passing packaged chips are ready to be shipped

lot has a unique ID as well.

In the automotive MC team in Padua, test engineers are responsible of developing the Test Program that Infineon's chips have to go through. Thus, the quality of the testing algorithms is crucial in order to detect all possible faulty dies. The aim of this thesis is to build a quality gate tool to our Test Program, using machine learning techniques as explained in the following section.

1.5 Thesis Overview

As seen aforementioned, the chip goes through FE then BE testing phases during which all tests results are stored. Dies that passed the FE testing are sent to assembly and BE testing. However, some faulty dies may be undetected during the FE tests, yet they will ultimately fail in BE tests. The first task of our tool is to take into consideration all dies that passed the FE, the results of each test, and predict whether these dies will pass or fail the BE test phase. This is done by first extracting chips who passed FE, their FE tests results, and their corresponding labels (i.e. Fail or Pass) after BE testing, data extraction is described in Section 4.2. In the context of machine learning, the test results are the features, and the BE labels present the target. The second step is to study whom of these features are the ones that are more contributing in the failure of the chip, also known in this thesis as killer parameters. We refer to this step as data processing which is detailed in both Section 3.2 and Section 4.3. Once the dataset is ready, it is used to train a ML model. The model will then, given the FE tests results of new chips, predict the final state of the chip after the BE. It is important to note that in this step, the dataset is

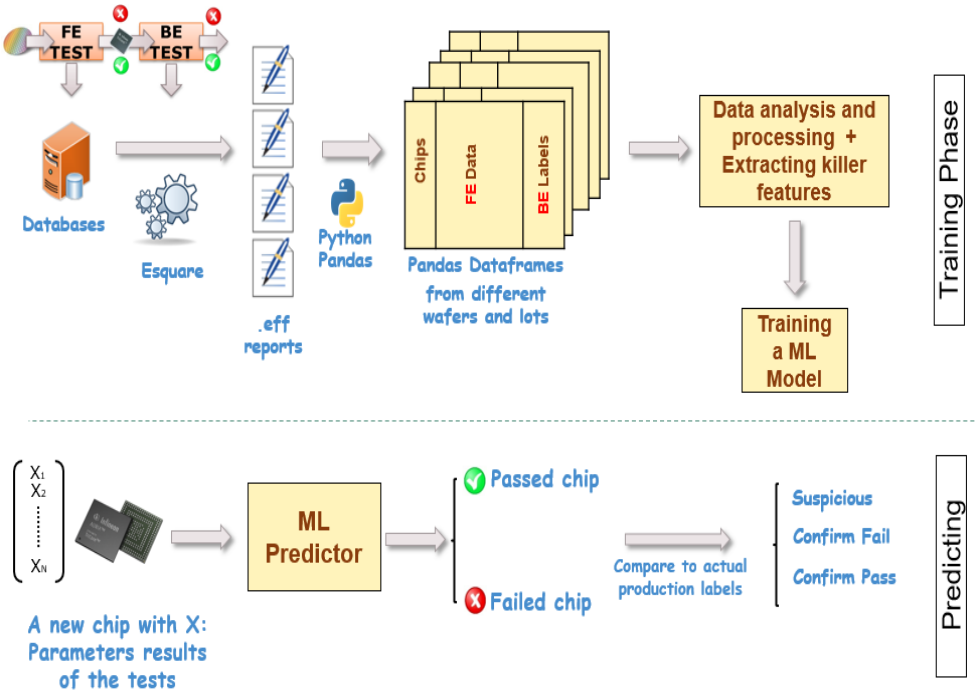

FIGURE 1.3: Qaulity Gate tool big picture

imbalanced, indeed the number of BE failing chips is usually very low compared to the passing ones, which highlight the need of carefully training and choosing the model for correct classification, details regarding this challenge is mentioned in Section 3.4.1. The second task of the Quality Gate tool, is to compare these predictions, to the real labels given to these chips after going through BE. In case of an accurate model, we expect the labels assigned after BE testing to be similar to the ones predicted, thus we confirm the reliability of our test program in detecting failures. The QG tool will either confirm the results of the BE testing or raise a warning in case of suspicious behavior, that is a chip is predicted to fail when it actually passed and vice-versa. Details about the selected ML model and the output of the quality gate tool are presented in Chapter 5.

Fig 1.3 is a simplified representation of the big picture of the entire thesis, starting from data extraction to final comparison.

Chapter 2

Production Test Flows for Non-Volatile-Memory

2.1 Introduction

Infineon Technologies AG is a leading player and pioneer in automotive electronics. Thanks to its testing quality, Infineon's microcontrollers are well-suited for safety-critical applications to support clean, autonomous and connected cars. This chapter is dedicated to explain the entire testing process within Infineon, starting by an overview on the tested devices, then a description of the testflow, and finally the test results and binning. In the end of this chapter, we describe the thesis outlook in the context of the testing process explained.

2.2 Device Under Test (DUT)

The production data analyzed during this thesis are related to the $AURIX^{TM}$ 2G microcontroller family produced by Infineon, more specifically the product line TC39x. In fact, $AURIX^{TM}$, standing for Automotive Realtime Integrated NeXt Generation Architecture, is Infineon's current family of microcontrollers that serve the precise needs of the automotive industry in terms of performance and safety. With the second generation being a further enormous increase in performance, $AURIX^{TM}$ TC39x were specifically designed for electric and/or autonomous vehicles. They are equipped with [39] :

- up to 16 Mbytes of embedded Flash memory
- more than 6 Mbytes of RAM
- up to six 32-bit $TriCore^{TM}$ processor cores.

$AURIX^{TM}$ TC39x characteristics are summed up in Figure 2.1. Each of these modules in the microcontroller is tested in order to deliver reliable chips. However, as mentioned in Section 1.2.1 ,Flash memory testing is MC Padua team's core business. In fact, embedded Flash (eFlash) memories represent a large percentage of the area of modern automotive microcontrollers, thus significantly contribute to the overall product quality and yield. In this context, our main focus in this thesis will be regarding the tests done on Flash, and in predicting failure caused by flash related tests.

Feature Set		9x Series eXtension (16MB)
TriCore 1.6	# Cores / Checker	6/4
	Frequency	300MHz
Accelerator	Signal processing Unit (SPU)	2xSPU
Flash	Program Flash	16MB
	Data Flash (physical/logical)	1024kB
SRAM	Total (DMI , PMI, LMU, AMU)	6912KB
DMA	Channels	128
ADC	Modules Primary / Sec / FC / DS	8/4/8/14
	Channels Primary / Sec / FC /DS	64/64/8/14
Timer	GTM TIM / (A)TOM / MCS	64 / 192 / 10
	CCU / GPT modules / bit streaming	2/1/1
Interfaces	FlexRay (#/ch.)	2 /4
	CAN-FD / TT	12/1
	QSPI / ASCLIN / I2C	6 /12/2
	SENT / PSI5 / PSI5S	25/4/1
	HSSL / MSC / EBU	2/4/1
	Ethernet 100Mbps/1Gbps	1/1
	eMMC/SDIO	1/1
	Radar /ext. ADC IF (RIF)	12x400Mbps LVDS
	Camera IF (CIF)	-
Security	HSM	HSM+ECC256
Safety	SIL Level	ASIL D
Power	EVR	Yes (3.3V/5V)
	Standby Control Unit	yes

FIGURE 2.1: AURIX TC39x Feature Table [39]

2.2.1 Flash Memory

The embedded flash memory, is a non volatile memory, that comprises the following components:

- **Flash Standard Interface (FSI)** : a programmable finite state machine that handles sequences to perform erase, program and verify (see following section) operations on all Flash memories.

- **ProgramFlash(PFLASH**): Divided into one or more banks [1] each connected to a CPU. It is used by the application to store program code and data constants. In addition to the Flash arrays [2], it also contains an analog block with pumps and regulators.

- **Data Flash (DFLASH):** The Data Flash Module is divided in banks and used to emulate an EEPROM (Electrically Erasable Programmable Read-Only Memory [3]) to store data for user and security applications. DFLASH read accesses are relatively slow compared to PFLASH accesses. Data Flash Module also contains regions to store configuration data in User Configuration Blocks (UCBs), and Configuration Sector (CFS). This last region is not accessible by user and stores system set-up data needed for the correct working configuration of the chip. In addition to the Flash Arrays, there is an analog block containing pumps and regulators.

An Aurix^{TM} TC39x contains a total of 16 MB of Program Flash memory. Each Program Flash memory consists of 5 memory banks with a size of 3 MB and 1 with the size of 1 MB. Each bank can be subdivided into physical sectors of 1 MB, so every Program Flash bank consists of 3 physical sectors, with the exception of the 1 MB size one, that has only one physical sector [24].

In this thesis, all tests done on all Flash components, banks and sectors are considered. In fact, for flash testing, an approach, based on a software solution for embedded memories testing called Flash Software Implemented Self-Test (FSIST), is followed. This technique consists of using the computing resources of the Device Under Test (DUT) to test its eFlash memory itself. The FSISTs are portions of software executed by the DUT's CPU, and could perform the following tests [10]

- **Erase Command:** performs erase operation on the flash array and measures the erase time, pump voltage used, number of erased sectors, etc.

[1] a Flash module (e.g PFLASH) is divided in separate banks. The banks support concurrent operations with some limitations due to common logic

[2] Flash array is the physical memory used to store information.

[3] EEPROM is a type of non-volatile memory, used in electronic devices to store small amounts of data that must be maintained when power is off.

According to the module design specification, it is possible to erase different cluster of cells.

- **Program Command:** performs a program operation with some particular patterns, and measure program time and pump load used. Possible patterns could be solid patterns (Program all zeros, Program all ones) and checkerboard patterns (alternated zeros and ones).
- **Verify Command:** the verify test reads all the Flash memory and compares the read content (*actual content*) with the one previously injected during the programming/erasing step (*expected content*). it is possible to verify all kind of patterns, verify all zeros, verify all ones, verify checkerboard patterns.
- **Disturb and Stress Command:** executes disturb and stress on the flash array to stimulate the appearance of faults. Different patterns of disturb and stress are performed, such as Gate Disturb, Drain disturb, side wall stress, Burn-In all, etc.
- **Redundancy Command:** This command handle the repair of the flash by computing how to allocate the redundancy resources in order to repair the failed Flash cells detected during the verify.
- **Analogue Command:** performs analogue measurements such as current, voltage and frequencies, in order to perform analogue tuning.
- **Trimming Command:** is used to trigger tuning tests on eFlash. every time an erase and verify are performed, trimming is performed to adjust the parameters to get the minimum bit erros. Trimming is done on the following parameters: Voltages done chip to chip (aka CTCT Chip To Chip Trimming), and Flash Oscillator trimming.
- **Cycling Command:** cycling is a combination of erase and program operations done to disturb the flash.
- **Functional Command:** contains several test objects that cannot be linked to any other FSIST command but belong to functional testing.

Each of these commands have a set of failing criteria, i.e. specifications, if a test does not meet these specifications it fails and the chip is binned.
Indeed, during FE testing some chips are failing during the aforementioned flash tests if specifications aren't met. The following section describe in details the entire test flow, and the binning of chips.

2.3 Non-Volatile-Memory Production Test Flow

2.3.1 Front End Insertions

The FE test stages can be divided into sub-phases which differ in the testing temperature (from $-40°$ to $125°$ Celsius), and usually referred to respectively as S1 and S2. Each test phase is composed of a multiple single tests which are identified by a unique number called test number (TN). For each insertion, we define a test suite, which is a sequence of test routines that are usually composed of the following steps: Stress, Disturb, Program/Erase, Verify, Repair, Flash scan, Analogue measurement and trimming. Note that, this sequence is not strictly followed, it may happen that after a single Erase, several kinds of verify are executed and not all of them are followed by a repair. However, it is possible to differentiate these single tests by their unique TN.

As mentioned before in Section 2.2.1, when describing FSIST commands, each of these tests produce a set of measurements, that according to certain specifications, it is decided whether a die should pass or fail that test. For example, given an erase command, a test can be declared **FAIL** based on the following criteria: Erase time too long or Pump load limits exceeded. For a verify command, a test will fail if the number of SBER detected exceed SBER_LIMIT, i.e. accepted limit on Single Bit Error found, on overall selected banks.

Now for each test, if a die failed a test not meeting the criteria, it is filtered out and it does not undergo the rest of the test-flow. By the end of the FE test, bad dies are separated from good dies when cutting the wafer, and only dies that passed all the S1 and S2 insertions remain and will be packaged.

2.3.2 Assembly and Packaging

As a first step, the wafer is cut into singular dies. These dies are actually functional but are impossible to use without an external protective package, indeed any scratch would impact the reliability of the chip, and any shock would cause its failure. Therefore, as a second step, the individual chips are placed in a package and terminals are attached. The result is a finished semiconductor device, that should go through several other tests, i.e. BE testing, in order to be shipped to the customer in the highest quality.

2.3.3 Back End

Similar to FE testing, Back End tests can also be divided into sub-phases with different testing temperatures. However, the first step in BE testing is IBIS, IBIS testing is the crucial part of BE testing, as it is the step where most stress is applied on packaged chips, the following insertions usually do not bring any additional test coverage [4] with respect to NVM operations. For that reason, in this thesis we consider the labels assigned right after IBIS testing as a final label of the packaged chip, yet we will refer to it as BE. Figure 2.2 is a representation of the entire test process.

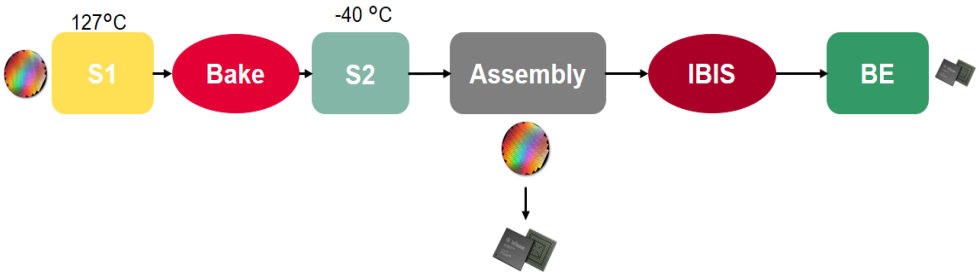

FIGURE 2.2: TestFlow

2.3.4 Test Binning

When a chip fail the testing it is discarded into specific bins according to the type of the failing part of the chip, or the failing test. There are two types of BINS: Hard-Bins and Soft-Bins.

- **HBIN**: a HBIN is defined by a single-digit number indicating the container index for failing devices depending on the filing module. For example, the HBIN1 is the bin that contains all the passing devices. HBIN8 is the BIN that contains all the chips that have failed because of flash related reasons. As we are interested in this thesis in the Flash binning, only these failing chips of this bin are considered.

- **SBIN**: a soft bin could be seen as a *sub-bin* of the HBIN where the failing devices are binned based on particular module tests, e.g. for flash testing, a chip exceeded the number of SBER tolerated in a verify test is binned in HB8 and a certain SBIN. SBIN is defined by a three-digit number which more precisely identifies the faults causing the rejection.

[4] Test Coverage: the ability of the test to detect a given set of faults that may occur on the DUT. It is the ratio of detected failures over the total failures

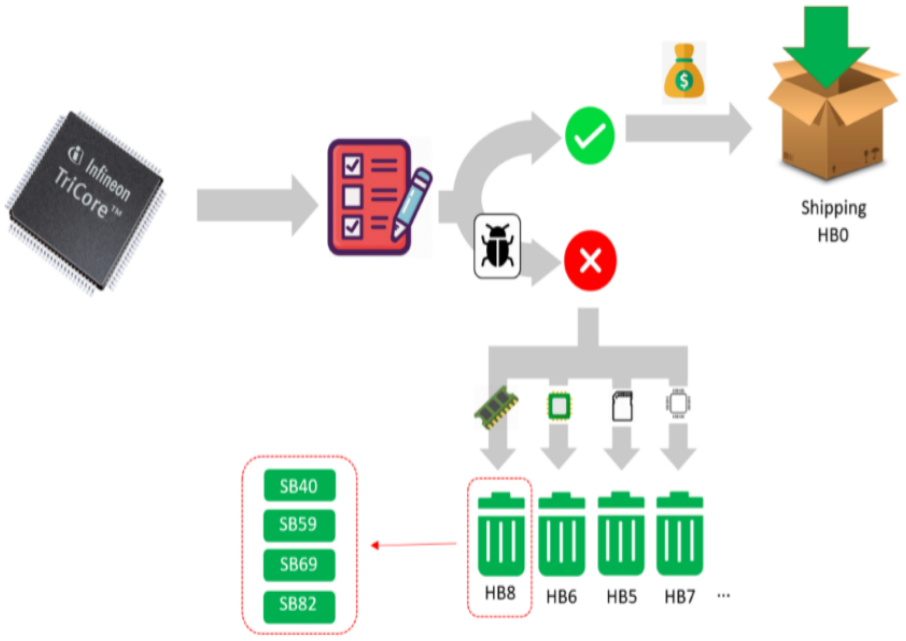

FIGURE 2.3: Passing and failing chips distributed into Bins [32]

In our case, we will consider only the chips that passed the FE testing, so chips in HBIN1 after FE tests, among these chips we consider only the ones that either failed or passed IBIS testing for flash related issues, therefore we are considering passing chips, and chips binned in HBIN8 after IBIS tests. Further details about the dataset are provided in Chapter 4.

2.4 Test Results

All the test algorithms produce some results, which contain the results of the test and some other additional information. According to the specific test algorithm, we have two kind of results:

- **Bitmaps**: are special output of a verify and a repair test. As its name indicate, a bitmap is a topographical map of bit error position. This kind of results is not used in this thesis, and therefore is not detailed in this section and it will not be again referenced in the following chapters.
- **Parametric results**: are the results returned by FSIST tests and that can be expressed in numeric format via two parametric tables , the *result table* and the *debug table*. These data are stored into reserved RAM portions during the tests execution and are then downloaded by the ATE which save them in an Infineon's shared database for data analysis purpose.

Result Table

The result table is the main output of a test algorithm, it comprises up to 6 parametric results. Each FSIST fills the result table with different information, depending on the relevance it has for that specific test. In table 2.1 the result table of the Program, Erase and Verify algorithms is reported [10] .

Parametrs	Program	Erase	Verify
PARAM1	Program Time [s]	Erase time [s]	Number of SBER
PARAM2	Pump Max Load [%]	Pump Max Load [%]	Number of MBER
PARAM3	Pump Avg Load [%]	Pump Avg Load [%]	Number of failing zeros
PARAM4	Pump Sigma Load [%]		Number of failing ones
PARAM5			Number of RED BL (i.e Redundancy bit line) used. Number of RED MNSEC (i.e Redundancy word line) used.
PARAM6			Number of new RED BL used. Number of new RED MNSEC used.

TABLE 2.1: Example of Result tables of Program, Erase, and verify commands

Debug Table

The debug table is a much larger results table. It collects all the other information that may be useful to know about the executed test algorithm. It is optional and it is actually dumped only when expressively required for debug purposes. For example the program test reports in the debug table: Number of programmed pages, Mean page program time, Max page program time, etc.

2.5 Data Source For ML

In this thesis, as explained in chapter 1, we aim to predict the final state i.e. Pass/Fail of the packaged chip based on the passed FE tests that it went

through, that is given a new chip that passed FE, depending on the results of these test, we predict whether it will fail or pass the BE. However, as explained aforementioned, each testing algorithm produce two tables of 4 to 6 parameters. Therefore, for each chip, each parameter of each test in FE is a feature in our ML problem, lot and wafer ID are excluded, and the final label assigned after IBIS, will be our target. The final label assigned is either HBIN1 indicating pass or HBIN8 indicating flash related fail. However, in HBIN8, we have different possible failures indicated by soft bins as explained previously, some of these soft bins only have less than 10 instances, some occurs more than others in the dataset, and some are easier differentiated it from pass than others. For that reason, we will focus on this thesis, as a first step, on predicting one of the most occurring SBIN, SBIN555 (analysis regarding the choice of SBIN are provided in appendix). Therefore, the dataset is re-labeled as following, only samples with the selected SBIN are designated by a binary "0" indicating fail and all others (even samples that had a different fail mode, i.e different SBIN) are designated by a binary "1" indicating pass. This model thus uses a binary classifier, however in next step, other SBINs could be considered, or as well as a multi-classifier predicting different SBINs. Further details regarding the data are explained in Chapter 4

Chapter 3

Methodological background

3.1 Introduction

This chapter is dedicated to explain the theory behind the techniques used in this thesis work. According to usual workflow of Machine Learning [22] [9], the work is divided in four steps: data extraction, data processing, model selection and performance evaluation. The following chapter is structured as following, we first describe the statistics done on data and feature engineering [1] techniques used, we then introduce the challenge of data imbalance and present the proposed solutions, finally we define the possible classifiers we are using, the data used to train and test them, and how to evaluate them.

3.2 Feature Engineering

This section is dedicated to explain the theory behind the techniques adopted for feature filtering and processing.

3.2.1 Density Estimation

When the distribution of the data is unknown, it is rather useful to be estimated in order to understand the behavior of the data. There are two main density estimation techniques we use in order to visualize our data:

- **Kernel Density Estimation (KDE)** : Kernel density estimation (KDE) is a non-parametric way to model the probability distribution that generated a dataset. A basic example is the histogram, an histogram divides the data into discrete bins, counts the number of points that fall in each bin, and then visualizes the results in an intuitive manner. Kernel

[1] feature engineering field contains a variety of issues and tasks. The most representative issues and tasks are feature transformation, feature generation and extraction, feature selection, automatic feature engineering, and feature analysis and evaluation

density estimates are closely related to histograms, but can be enriched with properties such as smoothness or continuity by using a suitable kernel [30]. In this thesis, we use the python function *kdeplot*, with a typical kernel that is the Gaussian kernel, to plot the density distribution of our data [31].

- **Probability barplot**: Another way to visualize the distribution of data, is to measure the probability of occurrence of each possible values in the data. This is usually done when dealing with data with discrete values.

These techniques provide an accessible way to see and understand trends, outliers, and patterns in data. Indeed these techniques were used in this thesis to analyze the features. In fact, in case of features with continuous values, KDE was used to present the distribution of the measurements, meanwhile for features with discrete values, it was more reasonable to use histograms and barplots.

3.2.2 Kolmogorov–Smirnov Test

The Kolmogorov–Smirnov **K-S** test [21], is a non-parametric test that compare continuous or discontinuous probability distributions. It could be used to compare a representative sample of data with a reference probability distribution, which is usually referred to as **one-sample K–S test**, or to compare two samples, usually referred to as **two-sample K–S test**. In this thesis, we are interested in the two-sample K–S test.

The K-S test is actually constructed as a statistical hypothesis test. We determine a null hypothesis, that the two samples we are testing come from the same distribution. Then we search for evidence that this hypothesis should be rejected. This is done by computing the K-S statistic $\mathcal{D}$, if $\mathcal{D}$ exceeds a certain confidence level, we reject the null hypothesis that the samples are from the same distribution. On the other hand if the K-S statistic $\mathcal{D}$ is very small we cannot reject the hypothesis that the distributions of the two samples are the same.

The Kolmogorov–Smirnov statistic is defined as:

$$\mathcal{D}_{n,m} = sup_x |(F_{1,n}(x) - F_{2,m}(x)| \tag{3.1}$$

Where where $F_{1,n}$ and $F_{2,m}$ are the empirical distribution functions of the first and the second sample respectively, sup the supremum function, and

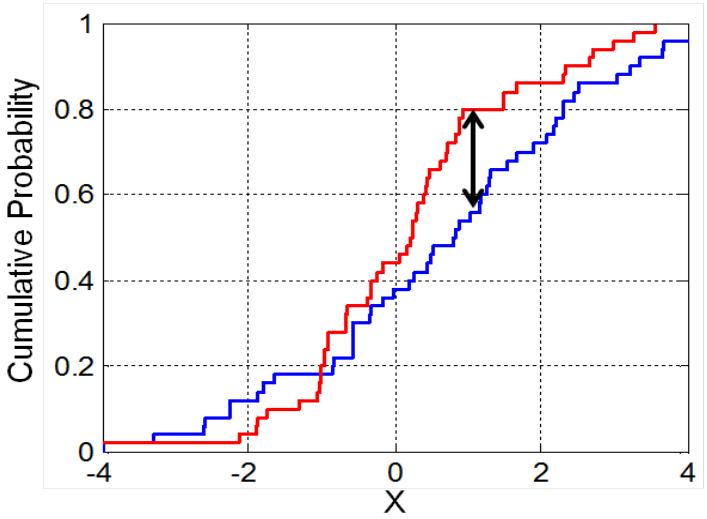

FIGURE 3.1: Illustration of the two-sample Kolmogorov–Smirnov statistic. Red and blue lines each correspond to an empirical distribution function, and the black arrow is the two-sample KS statistic [2]

n, m are the sizes of first and second sample respectively. It could also be seen as the maximum vertical distance between the empirical cumulative distribution functions of the two samples [41], as in figure 3.1.

During this thesis, K-S test was a very efficient way to study the significance of the features. In fact, K-S test was used to determine if for a certain feature, the failing instances and the passing ones were different from each other. In other words, if we take one feature, and consider a sample of only passing chips and a sample of only failing chips, we can make use of K-S statistics to know whether these two samples are different or similar, in case of a very small K-S statistic the failing chips sample is similar to the passing chips sample and this feature is not actually bringing any information on how to separate these two classes. These results are better described in Chapter 4.

3.2.3 Correlation Analysis

One of the most common approaches to dimensionality reduction is correlation analysis [35] [7], the purpose of this analysis is to identify the correlations between each pair of features, or between a feature and the target, to understand which features are the most relevant to the model.
The correlation between two variables is defined as:

$$\rho_{x,y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{3.2}$$

where X and Y are two vectors, with mean values μ_X and μ_Y and standard deviations σ_X and σ_Y. $E[.]$ is the expected value operator, and cov denotes covariance. The correlation coefficient $\rho_{X,Y}$ cannot exceed 1 in absolute value, and is a measure of the degree of linear relationship between two random variables.
The closer the correlation coefficient in absolute value to 1 the more closely the two variables are related. In case of correlation between a feature and the target, a high correlation shows that the feature is strongly related to the target, and therefore an interesting feature to keep. However, in case of two features, a high correlation means these two features are related, therefore bringing the same information to the model, and it is rather more reasonable to keep only one of them. Note that, $\rho_{X,Y}$ is a measure of linear correlation, thus it does not take into consideration non linear correlations.

In order to exploit linear relationships between all pair of features, we compute the correlation matrix, given the matrix $X = [X_1, \ldots, X_n]$:

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} & \ldots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \ldots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \ldots & x_{dn} \end{pmatrix}$$

Where $X_1, X_2, \ldots, X_n$ vectors are the n features of our dataset, d is the size of instances in our dataset. The correlation matrix R_X could be defined as following:

$$R_X = \begin{bmatrix} \rho_{X_1,X_1} & \rho_{X_1,X_2} & \rho_{X_1,X_3} & \cdots & \rho_{X_1,X_n} \\ \rho_{X_2,X_1} & \rho_{X_2,X_2} & \rho_{X_2,X_3} & \cdots & \rho_{X_2,X_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{X_n,X_1} & \rho_{X_n,X_2} & \rho_{X_n,X_3} & \cdots & \rho_{X_n,X_n} \end{bmatrix}$$

R_X is a symmetric matrix with the correlation measures, ρ_{X_i,X_j} is the correlation coefficient between the i^{th} and j^{th} features, where $i, j = \{1, \ldots, n\}$, and n is the number of features. The correlation matrix for our features is computed and analyzed in Chapter 4.

3.2.4 Data Normalization

To better describe the data normalization, some definitions of the related statistics are firstly given:

- **Mean:** the statistical mean refers to the mean or average that is used to derive the central value of a discrete set of numbers, it is determined by

adding all the data points in a variable X and then dividing the total by the number of points:

$$\mu_X = \frac{\sum_i x_i}{n} \tag{3.3}$$

Where μ_X is the mean value of the variable X, n is the size of X, and x_i is the i^{th} point of X.

- **Standard deviation:** often represented by σ, is a statistical parameter that measures the dispersion of a dataset relative to its mean. It is calculated as the square root of variance by determining the variation between each data point relative to the mean, for a variable $\mathbf{X}= [x_1, \ldots, x_n]$, σ is calculated as following:

$$\sigma = \sqrt{\frac{\sum_i (x_i - \mu_X)}{n}} \tag{3.4}$$

  If the data points are further from the mean, there is a higher deviation within the data set; thus, the more spread out the data, the higher the standard deviation. A low standard deviation means that most of the numbers are close to the average. Note that, this statistic is also useful to get an idea about the range of values of our features in this work.

In the context of machine learning, while training the model, when features have different range of values, data normalization is rather important to change the values of features to a common scale without changing its behavior or nature, thus no feature has more importance than another due to having larger values. The common feature normalization technique we adopting in this thesis is **Z Normalization (Standardization)** [43] , that is computed as following:

$$X_{i,scaled} = \frac{X_i - \mu_i}{\sigma_X} \tag{3.5}$$

Note that in case of variable where all observations have the exact same value, σ_X will be equal to zero, and hence data normalization cannot be computed for such variable. However, a feature with $\sigma = 0$ is actually not affecting the final target, as it is completely uncorrelated to the class, so it can be dropped anyway.

The importance of data normalization actually depends on the choice of the ML model. Some models exploit "Distance" or "Similarities" between data samples, such as SVM, in this case, if the data is not scaled, some features may be given higher priority than others. On the other hand, other graph-based

models, such as Random forest and Naive Bayes, are invariant to data scaling. Some other models like Neural Network may not necessarily need data scaling, but that could help speed up the learning and lead to faster convergence. Further details about the ML algorithms are presented in Section 3.4.2.

3.3 Dimensionality Reduction

As we are dealing with a lot of possible features, it is important to consider dimensionality reduction techniques. In this thesis we give priority to filtering techniques, that is, choosing features based on their correlations and properties as explained above, however, feature selection and reduction techniques were studied. In literature there are two ways to reduce the dimensions and granularity of features:

- **Feature Selection**: is selecting the most informative features and excluding the least informative ones without changing them. Example of feature selection techniques is to consider random forest algorithm to rank features by importance, and take the top ones. [27]
- **Feature Reduction**: is transforming features into a lower dimension, creating new features that are a combination of the original features. The most known and effective technique for feature reduction is PCA (**P**rincipal **C**omponent **A**nalysis) [44].

However, these two techniques have their pros and cons, Fig 3.2 is a summary of the difference between these two techniques. In fact, reduction technique are rather efficient yet such methods do not select a set of features present in the original dataset, and thus cannot be used to know the killer features in the testflow, in other words, we will not be able to know which feature, i.e FE test is influencing the chip failure in IBIS. On the other hand, feature reduction techniques are not robust to feature set variation. In fact, test program may slightly change in terms of test time, number of tests and flow tests, hence a variation in the features set. Therefore we want to keep track of the exact name of the killer test, to extend the same ML model to other test programs in the future.

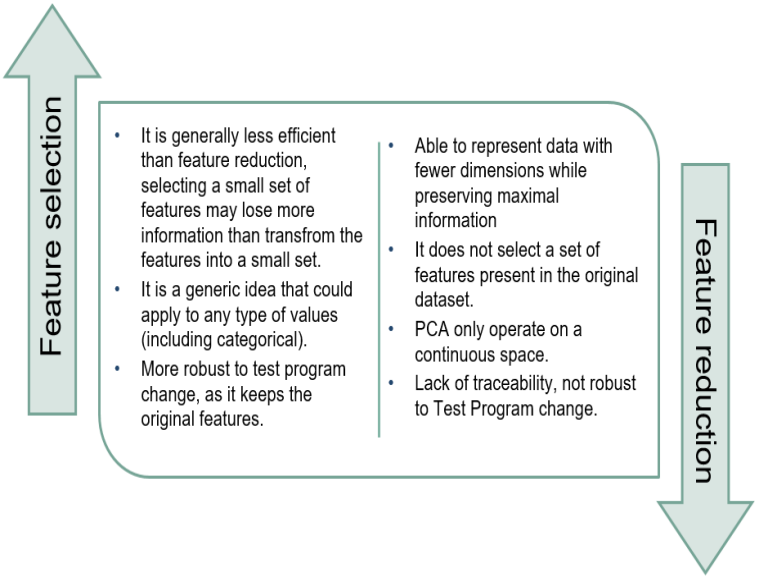

FIGURE 3.2: Feature selection Vs feature reduction

3.4 Machine Learning Techniques

3.4.1 Imbalanced Classification

The imbalanced data is characterized by having many more instances of certain classes than others [33]. In our case, fail instances are rare compared to pass instances, therefore respectively the minority and majority class. The fundamental issue with the imbalanced learning problem is the ability of imbalanced data to significantly compromise the performance of most standard learning algorithms [15]. In fact, most learning techniques are designed to maximize the expected accuracy by empirical risk minimization. That is to say, looking for the method that minimizes the loss as follows :

$$\min_{f} \sum_{i} L(f(x_i), y_i) \tag{3.6}$$

where x is the input, $f(x)$ is the predicted output, and $y \in 1, 0$ is the real label, as also seen in Fig. 3.3. However, when the data is imbalanced, as in our case, the majority class is Pass instances, labeled as $y = 1$, the minimum value of (3.6) may be achieved by setting f to be as simple as $f(x) \equiv 1, \forall x$. In other words, a "*high accuracy*" classifier may be achieved simply by declaring or classifying all samples as being in the majority class [6]. A model who always predict pass is actually useless, for that, different techniques were suggested in literature to deal with the imbalance challenge, whether techniques to re-balance the dataset, or techniques to bias the model in order to

take into consideration the imbalance issue of dataset. In the following section we summarize the techniques we tried in this thesis:

- Data level:
  - undersampling
  - oversampling
- Algorithm level:
  - Cost-Sensitive Learning (CSL)
  - Balanced Random Forest

However various other techniques are available based on different ML problems, the interested reader may refer to [15].

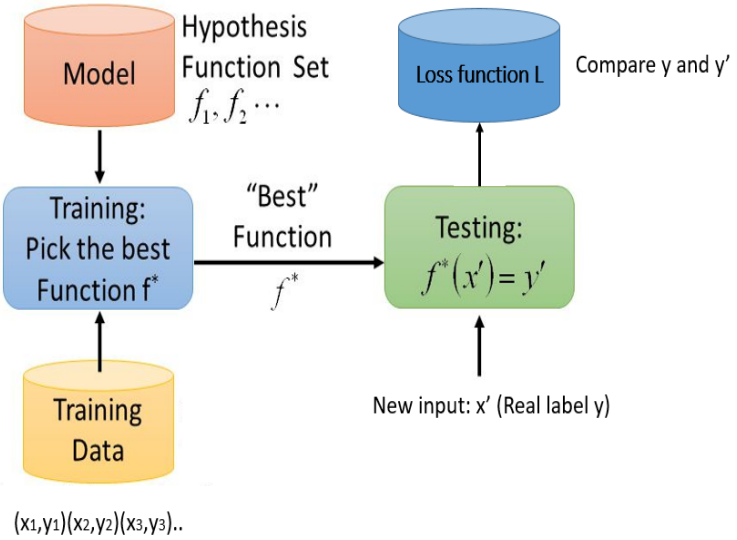

FIGURE 3.3: Machine learning workflow

Balancing Data:

Without any changes in the learning algorithm, we need to modify the dataset itself to provide a balanced training set for the classifier. There are two ways to balance the dataset, augmenting the minority class to be equal to the majority or reducing the number of majority class instances to be equal to the minority class, these techniques are called respectively **Oversampling** and **Undersampling**.

- **Oversampling:** consist of augmenting the minority class, by adding replicates of failing samples. Although this solves the imbalance issue, it may lead to an increase in computation time and to overfit due to a large number of identical minority training instances. However, another recommended oversampling technique is to create new synthetic instances of the minority class, instead of only replications, this technique is known as SMOTE oversampling (Synthetic Minority Oversampling) and it creates entries that are interpolations of the minority class [4], which is one of the techniques adopted in this work, the results are presented in Chapter 5.

- **Undersampling:** consist of randomly picking samples from the majority class, thus removes data from the original data set. Although this may reduce the number of the training instances, but will not append any new data to the original dataset. Results of this technique are also presented in Chapter 5.

ML Algorithms For Imbalanced Data:

Instead of creating a balanced data set for the model to train, other approach consist of changing the ML algorithm to handle the imbalance of data. Two approaches are suggested in this thesis:

- **Cost-Sensitive algorithms** : consists in using models which take into consideration the unbalancing of the data by weighting the training samples differently, depending on the class they belong to [13]. Intuitively, we want to give higher weight to minority class and lower weight to majority class. A possible way to set weights is to consider the ratio between the minority and majority class.

- **Balanced Random Forest (BRF)** : inspired by Random Forest to ensemble trees induced from balanced down-sampled data. The BRF algorithm does the following [5]:

  - For each iteration in random forest, draw a bootstrap sample from the minority class. Randomly draw the same number of cases, with replacement, from the majority class.
  - Induce a classification tree from the data to maximum size, without pruning. The tree is induced with the CART algorithm (Classification And Regression Tree algorithm) [29], with the following

modification: at each node, instead of searching through all features for the optimal split, only search through a set of randomly selected features.

– Repeat the two steps above for the number of times desired. Aggregate the predictions of the ensemble and make the final prediction.

3.4.2 ML Classifiers

Several machine learning classifiers could be used to predict whether a chip is among the Pass class or the Fail class based on data measured on FE test, in this section we provide an overview of the classifiers studied in this thesis work. We use three different approaches:

- **Support Vector Machine (SVM)**: as a techniques based on similarity between data samples.
- **Random Forest (RF)**: as graph-based technique.
- **Neural network (NN)**: as a deep learning approach.

Support Vector Machine:

In a nutshell, SVM is a supervised machine learning algorithm that is based on the idea of finding an hyperplane that best divides a dataset into two classes, as shown in the Figure 3.4. In fact, in case of linearly separated, We suppose that a training dataset S is available

$$S = \{x_i \in \mathbb{R}^{1\times d}, y_i \in \{0,1\}\}, i = 1, \dots n. \tag{3.7}$$

the values -1,1 are assigned to the two classes which the data belong to, could be referred to as class A and class B, for example $y_i = -1$ if the sample $\in$ class A, and $y_i = 1$ if the sample $\in$ class B.

The hyperplane F_0 in the $\mathbb{R}^d$ space is defined as:

$$F_0 = \{x | f(x) = x\beta + \beta_0 = 0\} \tag{3.8}$$

where $\beta \in \mathbb{R}^d$ with $\|\beta\| . = 1$. The classification is then based on the choice of $f(x)$ (and consequently of F_0):

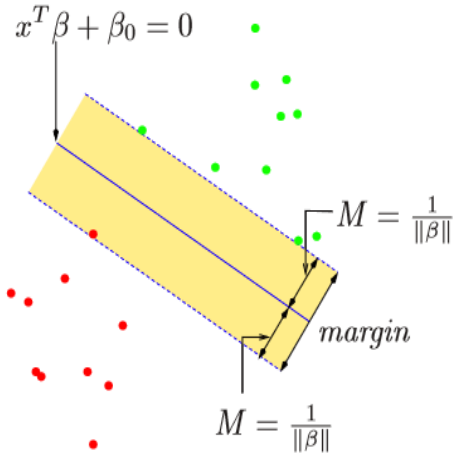

FIGURE 3.4: linearly separable SVM classification problem [36]

for a new sample $x^{new} \notin S$, we classify

$$\begin{cases} 1 & y^{new} = -1 \text{ if } f(x^{new}) > 0 \\ 2 & y^{new} = 1 \text{ if } f(x^{new}) < 0 \end{cases} \tag{3.9}$$

Since, by assumption, the two classes are separable, then it is possible to find a function $f(x)$ s.t.

$$y_i f(x) > 0 \ \forall i \tag{3.10}$$

We choose the hyperplane yielding the largest margin M between the two classes as seen in Fig 3.4.

This can be rephrased in terms of the maximization problem:

$$\max_{\beta,\beta_0,\|\beta\|.} M \ \text{ subject to } y_i f(x) > 1, i = 1, \ldots, n \tag{3.11}$$

that can be further translated into a more convenient minimization problem:

$$\min_{\beta,\beta_0} \|\beta\| . \ \text{ subject to } y_i f(x) > 1, i = 1, \ldots, n \tag{3.12}$$

The best hyperplane is found by optimum β that maximizes the margin. The boundaries of the defined margin, are called the support vectors [36].

However, in case of non linearly separable data, SVM is employed in combination with Kernel Methods. To put it briefly, the idea of kernel methods is to map the non-linear separable dataset into a higher dimensional space where it is possible to find an hyperplane that can separate the samples. The kernel function defines inner product of the mapping function [36].

Popular choices for the kernel function K are:

- **Polynomial:** $K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^d$
- **Radial Basis (RBF):** $K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|}{\sigma^2})$
- **Neural network:** $K(x_i, x_j) = tanh(\langle x_i, x_j \rangle + b)$

Radial Basis kernels are the most widely used and the ones considered in this thesis.

Random Forest:

Decision Tree Model: Decision tree models are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features [29]. In a decision tree, each internal node represents a *"question"* on an attribute (e.g. whether the number of SBER of a verify test is above a certain value or no, see Section 2.3.1 and Section 2.4), each branch represents the outcome of the question, and each leaf node represents a class label. Each branch can generate other branches until a a final predicted labels are reached, which correspond to the leaves of the tree. Figure 3.5 , is an example of a simple decision tree, using *Gini inpurity* as splitting quality metric, and a maximum depth set to 3 [29]

Random Forest Random forest classifiers are among the widely known and the most robust classifiers [12]. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set, see Fig. 3.6. In addition, when splitting a node during the construction of the tree, the chosen split is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases(with respect to the bias of a single non-random tree) but,due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model, less prone to overfitting.

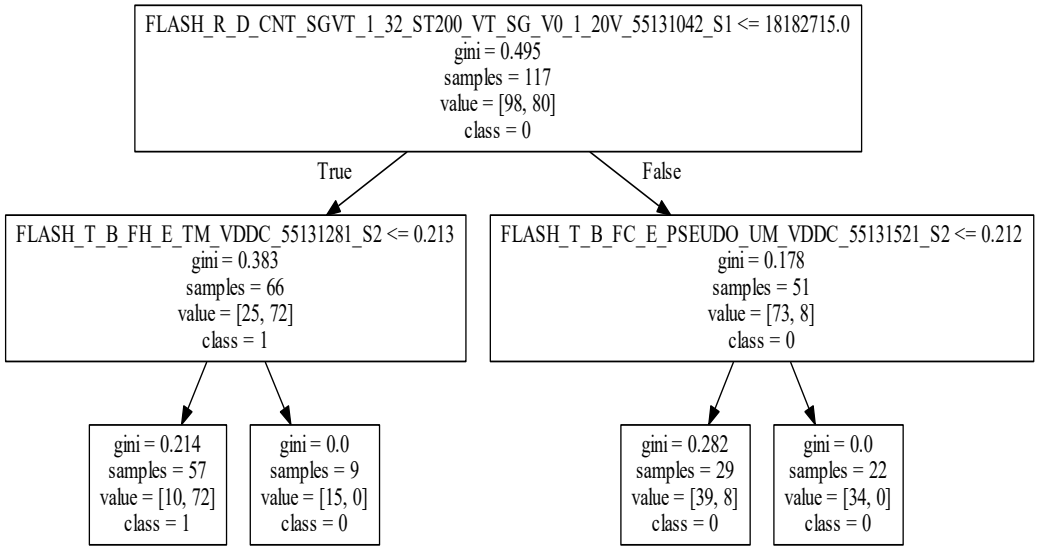

FIGURE 3.5: Decision Tree Example

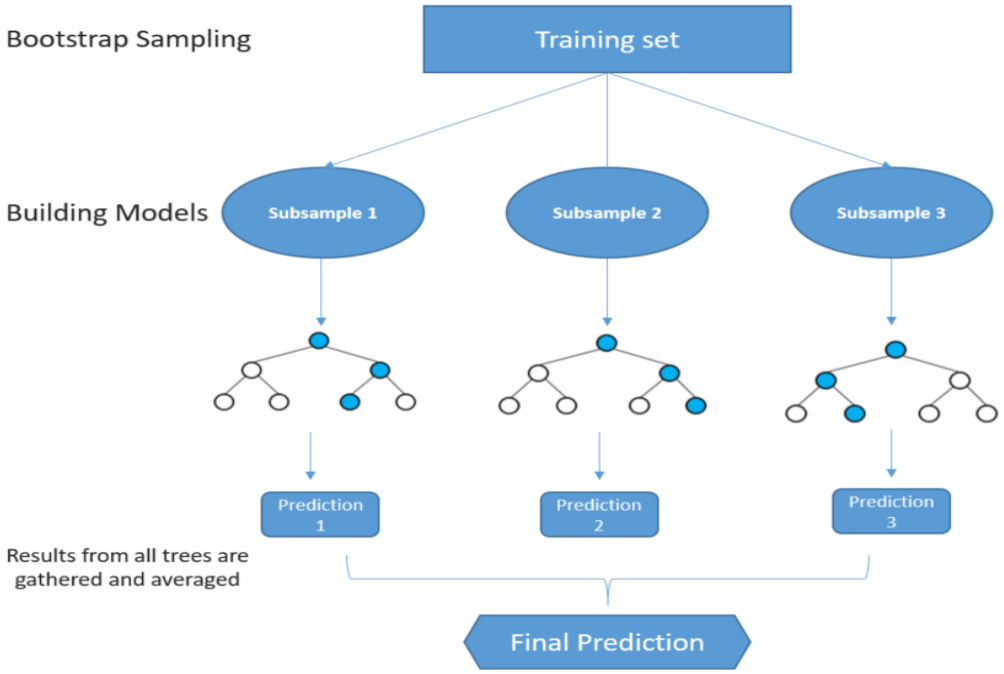

FIGURE 3.6: Random Forest overview

Neural network

In this thesis, we are using a feedforward artificial neural network, sometimes referred to as multi-layer perceptron. It consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. It utilizes a supervised learning technique called backpropagation for training and is able to distinguish data that is not linearly separable [14]. The model uses a ReLU activation function for the hidden layers:

$$ReLU : f(x) = max(0, x) \tag{3.13}$$

and a sigmoid function for its output single neuron

$$sigmoid : f(x) = \frac{1}{1 + \exp^{-x}} \tag{3.14}$$

The usage of a sigmoid activation function in the output node enables to predict a probability, thus the the predictor does not produce a label, but rather a probability of belonging to one or the other class. The predicted label is then the one corresponding to the most likely class. In this thesis, the implemented neural network is constituted of[2]

- 3 hidden layers.
- 512 nodes with 0.5 dropout
- optimizer: *Adamax*
- L2 Regularizer [3] of the neurons weights of 10^{-3}
- batch size: 64
- epochs: 40

Note that the results obtained by the NN were not as promising as the other models, thus we will not present its results in this thesis. However, further study of these model will be among the next steps of this thesis.

These three models are able to predict a probability, i.e. to predict how likely it is for a new instance to belong to a certain class. Given a new chip,

[2]These hyperparameters were chosen based on a grid search for the model's optimization

[3]The regularization is a technique to discourage the increasing of the complexity of a model during its training.

the model predict a measure of probability to be among the fail class, if that probability surpass a certain threshold the chip is classified as fail. The default threshold is usually set to 0.5. These predicted probabilities allow us to to better evaluate our ML model as seen in Section 3.5 and Chapter 5

3.5 Performance Metrics

Once a model is chosen and trained, it is rather important to find out how effective is this model and its performance on new test datasets. However, the choice of the evaluation metrics is very important, in fact, different performance metrics are used to evaluate different Machine Learning problems. In this thesis, as we are facing a class imbalance challenge, we pay a lot of attention to how we evaluate the goodness of our predictions. In fact, some of the known and usually used metrics is the *accuracy*, i.e. the percentage of correctly classified input points with respect to the total amount of them, this metric can be very misleading in our case, where the fail class is the minority class, if a model always predict Pass, the accuracy will be very high, as the only miss-classified instances will be the true fails that are anyway very few. Although this accuracy is high, the model is indeed useless.
In such cases, it is more suitable to deal with each class separately, and to be able to visualize what happens when a prediction is performed on a test set. This could be done by considering the *confusion matrix* and some metrics that could be easily derived from it.

Confusion Matrix

The Confusion matrix is one of the most intuitive and easiest performance metric in classification problems. Indeed, calculating a confusion matrix can give a better idea of what the model is getting right and what types of errors it is making. In binary classification, the confusion matrix is a 2×2 matrix, where each column of the matrix represents a possible predicted condition (output of the classifier), while each row of the matrix represents a possible true condition (ground truth), and is shown in the following graph 3.7

The Confusion matrix in itself is not a performance measure, however, most of the metrics could be derived from the numbers inside it. Taking into consideration that we define the **Fail** class as the **positive** class and the **Pass** class as the **negative** class, we can describe the terms associated to the confusion matrix as following:

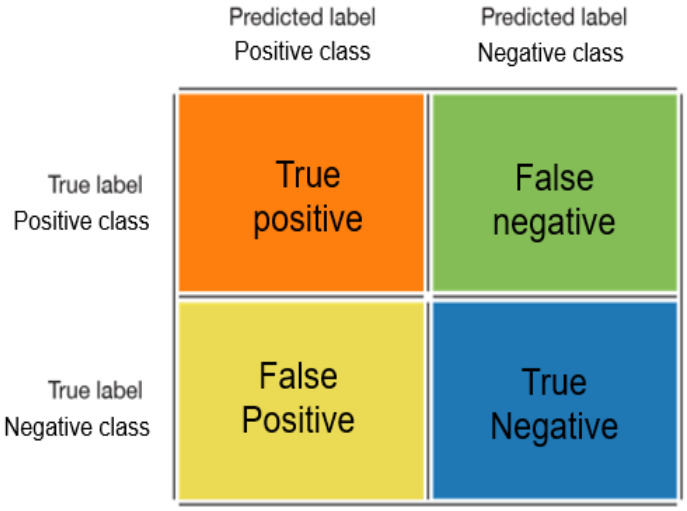

FIGURE 3.7: Confusion matrix and terminology for a binary classification problem

- True Positive (**TP**): The positives instances that were correctly classified, i.e. the fail chips that were indeed predicted as fail.
- False Positive (**FP**): The negative instances that were misclassified, i.e. the pass chips that were predicted as fail.
- False Negative (**FN**): The Positive instances that were classified as negative, i.e. the fail chips that were predicted as pass
- True Negative (**TN**): The negative instances that was correctly classified, i.e. the pass chips that were indeed predicted as pass.

Based on the confusion matrix, the following derivations are developed:

True Positive Rate TPR:

A measure of how well the model is able to detect the positive class. It is also referred to as Recall, or **Sensitivity** and it is defined as following:

$$TPR = \frac{TP}{TP + FN} \tag{3.15}$$

The higher is the TPR, the better is the model.

False Positive Rate FPR:

The FPR is the ratio of negative instances that were misclassified as Positive to the total number of negative instances. It could be seen as 1- TNR, i.e **T**rue

Negative **R**ate, also referred to as **Specificity**, thus the lowest is the FPR, the higher is the specificity, the better is the model in detecting the negative class. It is defined as follows:

$$FPR = \frac{FP}{FP + TN} = 1 - \frac{TN}{FP + TN} \tag{3.16}$$

The lower is the FPR, the better is the model.

In the literature of machine learning, some other metrics are often used. For example, the precision of a classifier is defined as:

- **Precision of the Fail class:** $Precision_{Fail} = \frac{TP}{TP+FP}$
- **Precision of Pass class:** $Precision_{Pass} = \frac{TN}{TN+FN}$

Precision is known as a measure of exactness, in other words when predicting a class how many points of the other class the model is including. For example when predicting fail, how many of these predictions is actually Pass instances and not fail. The less these instances, the higher is the precision of the model in predicting failures. However this measure may not be a good candidate in case of imbalanced dataset since the pass instances are too many compared to the fail ones. However this measure is related to the FPR, indeed a low FPR means a small number of FP, the lower is the FP the better is the precision.

Hence, in this thesis, we will focus on the TPR and FPR, as the two more adequate metrics to evaluate the model.

Receiver Operating Characteristic (ROC) Curve

The Receiver Operating Characteristic (ROC) curve is a graphic illustration of the performance of a binary classifier based on TPR and FPR. The ROC curve is plotted with TPR against the FPR at different classification thresholds, where TPR is on y-axis and FPR is on the x-axis.
An ideal ROC curve goes through TPR = 1, FPR = 0, which means that under some threshold, we can achieve 100% accuracy classification. On the other hand, a classifier based purely on a random guess is expected to give points lying along the diagonal such that FPR = TPR. A practical classifier is somewhere in between. As seen in Figure the more the curve is closer to the ideal case, the more accurate is our classifier.

The area under the ROC curve (AUC) is a useful performance measurement for classification problem at various thresholds settings. Although it could be seen from the ROC plot, it is useful to also have a numerical value

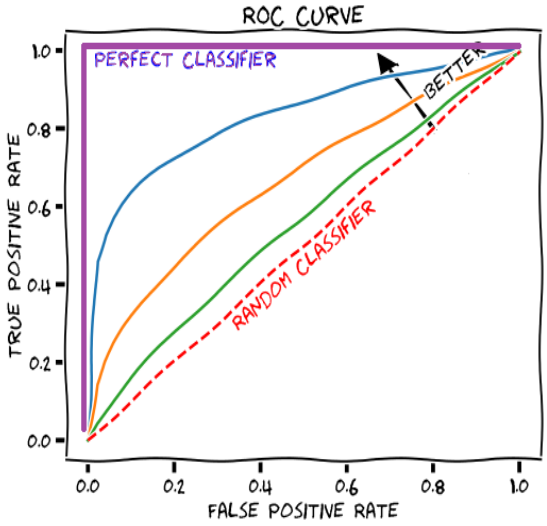

FIGURE 3.8: ROC curves [23]

that summarizes the entire ROC curve. In fact, for the ideal classifier, $AUC = 1$, for the random guess, $AUC = 0.5$, and for a practical classifier, $0.5 < AUC < 1$. In fact, an AUC near 1 means the model has good measure of separability whereas AUC near 0 means it has worst measure of separability. In order to visualize the separation of the two classes, it is recommended to plot the distribution of predicted probabilities (explained in Section 3.4.2). In Figures 3.9, 3.10 and 3.11, the red curve presents the positive class (i.e Fail chips), the green curve presents negative class (i.e Pass chips), and the x-axis indicate the predicted probability to be among the positive class (i.e. probability to fail). In an ideal situation, such as Figure 3.9, the two curves do not overlap at all, which means the model has an ideal measure of separability, failing chips have high predicted probabilities to fail (i.e. above 0.5) thus it is perfectly possible to distinguish between positive class and negative class.

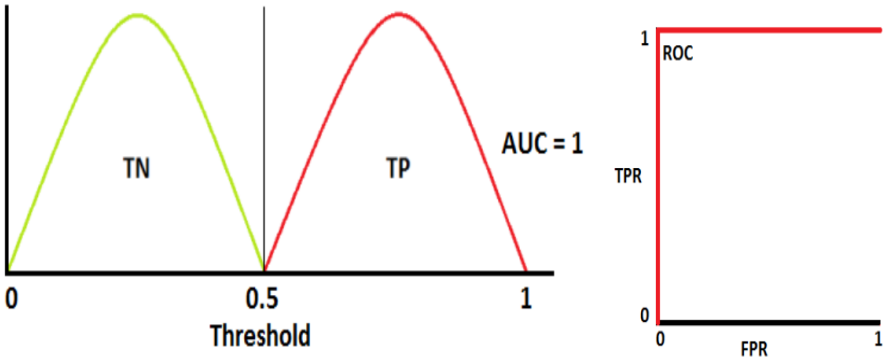

FIGURE 3.9: distribution of predicted probabilities to fail in ideal case [25]

On the other hand, a model is not ideal, and it may missclassify some instances, i.e certain fail instances could have a predicted probability to fail less than the threshold and therefore classified as pass, these instances are the FN in Figure 3.10. The model could also predict a high probability to fail for the pass instances (higher than threshold 0.5), these instances will thus be classified among the fail class, and are the FP presented in the same figure. The number of miss-classifications, FP and FN could be maximized or minimized by changing the threshold. The threshold could be tuned according to our requirements, further details are provided in Chapter 5.

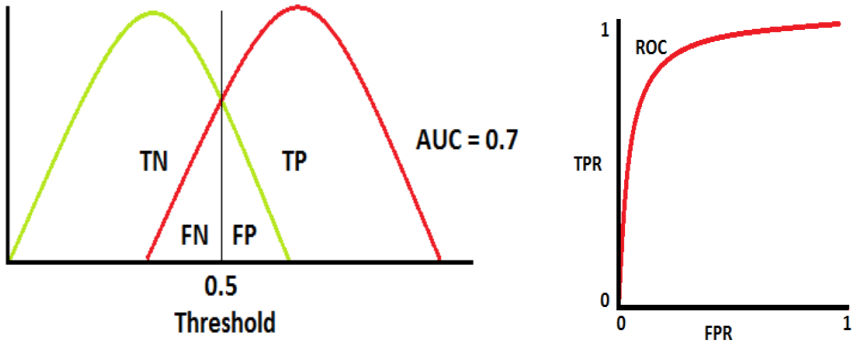

FIGURE 3.10: distribution of predicted probabilities to fail in practical case [25]

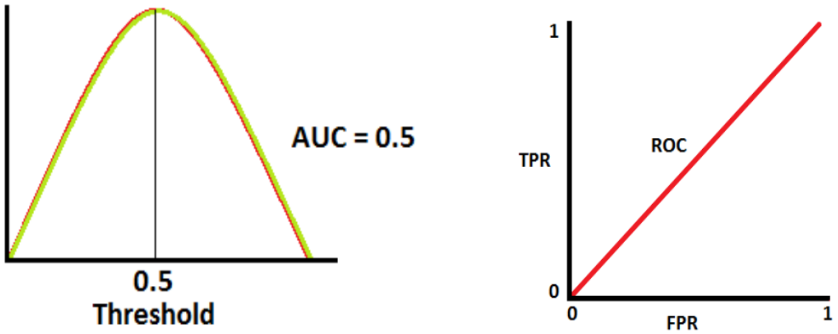

FIGURE 3.11: distribution of predicted probabilities to fail in worst case [25]

In the worst situation, the AUC is approximately 0.5, which mean the model has no discrimination capacity to distinguish between positive class and negative class, as seen in Figure 3.11.

3.6 Data Split

In some real cases, it is not always possible to get new data to test the ML model, thus it is always easier to split the entire available dataset into two datasets:

- **Training set**: to train the model, tune the hyperparameters, and any other activity concerning the development of the model.
- **Test set**: to test the model, and estimate an unbiased assessment of the model's performance. It is crucial not to use these data when training and selecting the model but only to evaluate it.

As obvious as it may look, splitting an imbalanced data is not an easy task. It should rather be handled very carefully in order to guarantee a trustful good performance of the model. In the following,we shed the light on three important data partitioning tips :

- Balancing techniques when splitting data
- Stratification
- Cross validation

Balancing Techniques when Splitting Data:

Since we are using balancing techniques to overcome the imbalance issue, it is necessary to note that the data splitting should always be done before balancing, thus only the training set should be balanced, because in real life cases, new data will be imbalanced, and the model should be able to predict correctly the classes of these data. On the other hand, balancing the data using oversampling technique before splitting may lead to overfitting problem. In fact, if we oversample the data by replicating the instances of minority class, then randomly split the data to train and test, we may have in our test data replications that are also in the train data. Take for example that we start with the original dataset where we have a minority class with two samples. We duplicate those samples, and then split the data, at this point there might be a split , such as the one showed in Figure 3.12, where the training and test set contain the same sample, resulting in overfitting and misleading results.

Although this case may not occur when *undersampling* the data, it is always advisable to split the data before sampling.

FIGURE 3.12: overfitting explained in case of oversampling [11]

Stratification

Stratification is the process of rearranging the data as to ensure each fold is a good representative of the whole [26]. In fact, when splitting data, we usually pick randomly samples from the original data to be in the training or the test set. However in case of imbalanced data, this technique may lead to have a test set with no instances from minority class, or as well with excessive instances from it, whereas the test data should represent the same distribution of instances of whole real data. In other words, it should have approximately the same percentage of samples of each target class as the complete set. Therefore when we sample from the original set to create the training set and the test set, we have to make sure that it is a stratified sampling and not random.

Cross Validation

Although, it makes sense to train the model, evaluate it on new test data, and assume the work is done and the model is good, it might be that the model is only performing well on that particular split. In these cases, it is

always recommended to consider cross-validation to evaluate the model. In this thesis, we consider a **K-Fold Cross-validation**, after splitting the data to train and test sets, we keep the test set for a final evaluation, and we use the train data with cross-validation where the dataset **D** is randomly split into **k** mutually exclusive subsets (the folds) $D_1, D_2, ..., D_k$ of approximately equal size. The model is trained and tested k times, each time $t \in 1, 2, ..., k$,it is trained on $D \setminus D_t$ and tested on D_t. Now, when evaluating the different metrics, with **k** different split of data, we can drive conclusions regarding our model that's not based on only one particular split. Note that also these **k** splits are stratified as explained previously.

Chapter 4

Data preparation

4.1 Introduction

The base of any machine learning problem is the data on which the model will be build, incorrect or inconsistent data leads to false conclusions. And so, how well we clean and understand the data has a high impact on the quality of the results. Indeed the quality of data beats the fanciness of algorithms. For that, in this thesis, a huge importance was given to the data. This chapter is dedicated to explain how the dataset used was extracted, cleaned, and processed to be ready to feed to a ML model.

4.2 Data Extraction

4.2.1 Esquare Tool

As mentioned in Chapter 2, during the testing process, all measurements are stored within Infineon's databases (**DB**). Many extraction tools are available to access these data, the main tool considered in this thesis is Esquare [40]

Esquare is a tool that can interface Infineon DB and can be instructed on which data to extract by specifying some criteria on column and row of DB, this task is done via the so-called Jobs.

Job Definition

A job is an executor of the script that will extract the specified data, thus some information must be set at the definition of the job and that will be used to select the data such as:

- **lot/wafer definition:** The purpose of the lot/wafer definition is to tell to the job which lots or lot/wafers to use for the extraction of the final

data. The lot/wafer definition determines the rows of the extraction result. Such definition consists of a series of conditions which have to be met by the lots in order to be used for the data extraction. The final outcome is a lot/wafer list that meet the criteria set, and will constitute the rows of our data. Example: Use all the lots for which an "S1" measurement has been performed, or use all wafers that belong to a certain lot ID.

- **Data column definition:** The data column definition specifies the columns for the lot/wafer definition in the output data. It is possible first to select the data sources, say data from FE insertions, then further filters are applied according to the data requirements. Example: Columns could be tests names from FE insertions, SBIN, HBIN.

- **Extraction level:** It is where the aggregation level of the extracted data is specified. In other words, after specifying what devices should be considered, based on lot/wafer list, and which data is extracted based on selected columns, we indicate whether this data should be extracted for each lot, for each wafer or for each chip.

These information are mandatory to execute a job, once they are indicated the job can be launched. As soon as it retrieves the required data, the job generates an EFF file with the output data, more details on EFF files are presented in a following section.

4.2.2 Extraction Specifications

In this thesis, we are interested in extracting the tests results of the FE insertions, and the final state of the chip after going through the BE testing. Therefore, two different jobs are launched, with the same lot/wafer definition and extraction level but different data columns. The lots are chosen such as they all went through both FE and BE testing. The requirements are summed up in Table 4.1.

4.2.3 EFF File Structure

The output data obtained from esquare are EFF (**E**DA **F**lat table **F**ormat) files, that are simply files with extension " .eff" and containing semicolon separated values.

Lot/wafer Definition	The lot/wafer definition list is filtered based on certain lots ID. Lots chosen are: 19841006 19848003 19848005 19848006 19848007 19848009 19848010
Data column definition	• **Job1 for FE data:** Select FE testing, S1 and S2 insertions, values and names of each test. • **Job2 for BE data** Select BE testing, BA insertion, IsPass label, HBIN and SBIN.
Extraction level	lot, wafer, XY CHIP

TABLE 4.1: Extraction criteria

These values when visualised via Excel usually have the following structure:

- **Messages** : include general information, such as the number of columns and rows of the file, starting time of extraction, and some information about the EBS from which the data where extracted.
- **EFF header** : includes the specifications we requested when extracting the data from esquare, such as Lot/wafer/X/Y indicating the extraction level, the parameter name, which indicate the names of the test parameters, i.e our data columns names, etc.
- **DATA**: contains the exact values of the corresponding data column, i.e test or label, for all the observations, i.e chips.

From EFF to Pandas Dataframe:

Since in this thesis, we are using *python programming language* for data analysis and ML, it is necessary to convert the EFF files into Pandas dataframes [1]. In the dataframe we only take into consideration the columns names and

[1] a dataframe is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns), provided by Pandas open-source python library

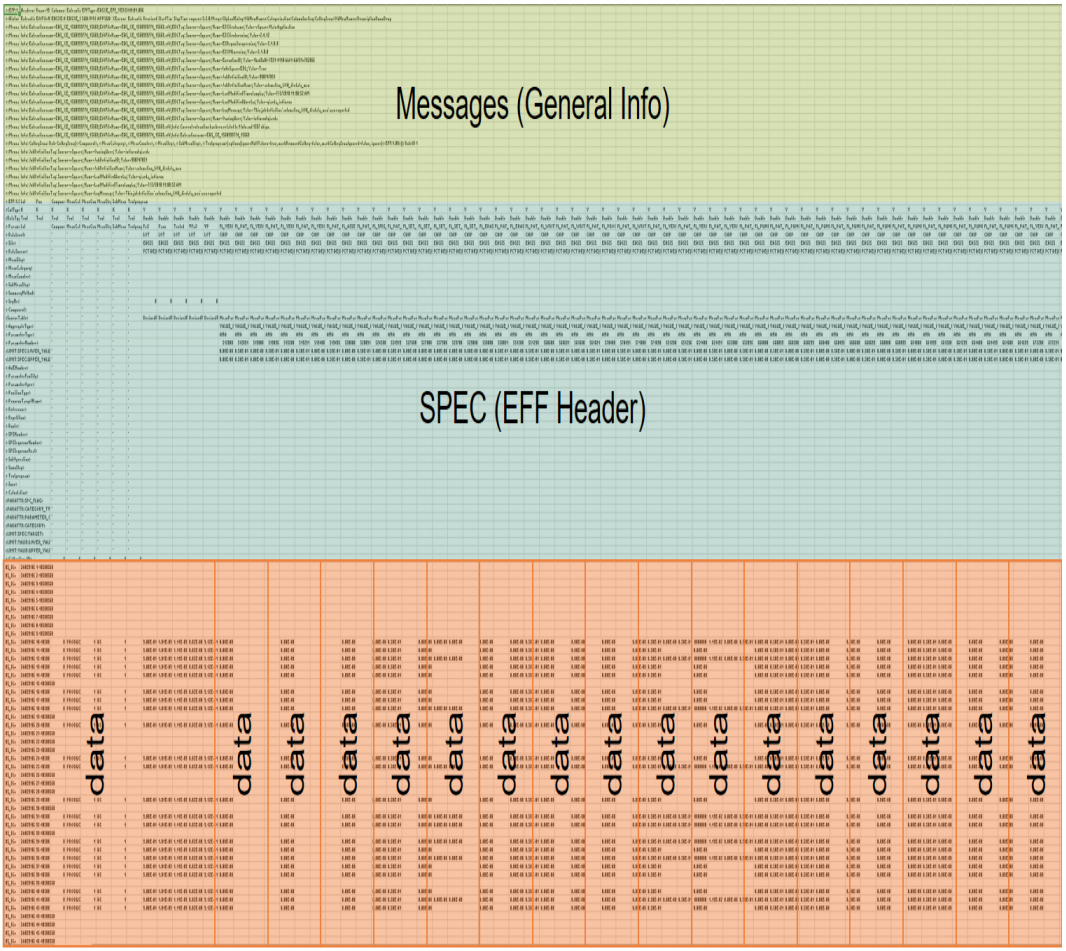

FIGURE 4.1: EFF file structure

the data values, dropping the non useful parts of the EFF file. In fact, these information can be found in **EFF Header** and **DATA**. Figure 4.2 is a detailed *snapshot* of the header, where the useful information for our case are highlighted.

- **ParameterName:** Refers to the columns names, i.e the names of the test parameters.
  Example:
  FLASH_T_B_FP_V0S_M3_RBLNEWUSEDFAIL_RWLNEWUSEDFAIL
  Where the first part refers to the test FLASH_T_B_FP_V0S_M3 , a verify command (Verify ALL 0s), second part refers to the parameter: RBLNEWUSEDFAIL_RWLNEWUSEDFAIL , i.e. Number of New RED BL used + failed, Number of New RED WL/SEC used + failed.
  ParameterName also refer to the IsPass, SBIN, and HBIN labels.

- **MeasStep:** the measurement step, indicating whether it is **S1** or **S2** insertion.

- **ParameterNumber:** refers to test number, which is a unique number for each single test.

<+ParameterName>	Lot	Wafer	X	Y	FLASH_T_B_[	FLASH_T_[	FLASH_T_[	FLASH_T_[	FLASH_T_[	FLASH_T_[	FLASH_T_[	FLASH_T_[	FLASH_T_[	FLASH_T_[
<+PositionType>			CHIP:N	CHIP:N										
<SourceTable>			MeasPartF	MeasPartF	MeasPartVal:	MeasPart\	MeasPart\	MeasPart\	MeasPart\	MeasPart\	MeasPart\	MeasPart\	MeasPart\	MeasPart\
<+Site>					EBS31	EBS31	EBS31	EBS31	EBS31	EBS31	EBS31	EBS31	EBS31	EBS31
<+AggregateType>					VALUE_F	VALUE_F	VALUE_F	VALUE_F	VALUE_F	VALUE_F	VALUE_F	VALUE_F	VALUE_F	VALUE_F
<+DataLevel>					CHIP	CHIP	CHIP	CHIP	CHIP	CHIP	CHIP	CHIP	CHIP	CHIP
<+DataSource>					FCT FE	FCT FE	FCT FE	FCT FE	FCT FE	FCT FE	FCT FE	FCT FE	FCT FE	FCT FE
<+MeasStep>					S1	S1	S1	S1	S1	S1	S1	S1	S1	S1
<+MeasCategory>					PRODUCTIVE	PRODUCTI	PRODUCTI	PRODUCTI	PRODUCTI	PRODUCTI	PRODUCTI	PRODUCTI	PRODUCTI	PRODUCTI
<+MeasCounter>					MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
<+SummaryMethod>														
<+SubMeasStep>					MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
<+ParameterType>					ANA	ANA	ANA	ANA	ANA	ANA	ANA	ANA	ANA	ANA
<+ParameterNumber>					49130361	49130362	49130363	49130364	49130763	49130764	49130765	49130766	49130767	49130768

FIGURE 4.2: EFF header

For our final data, we need the chip coordinates, the results of the tests, and the final label, these values are available in **DATA**. However, test results could be referred to either by their test name or their test number. Since we are considering all repetitive single tests done in both S1 and S2, it was rather better to keep these information for the column name, therefore each variable in our dataset will be named as following: TestName_TestNumber_S1 or TestName_TestNumber_S2. Figure 4.3 explain the overall representation of data, a snapshot of our final data is presented in appendix.

Lot	Wafer	X	Y	TestName_TestNumber_S1_1	TestName_TestNumber_S1_2	....	TestName_TestNumber_S2_1	....	Label
19842000	1	10	2	..	..	..	..	..	Fail
19842000	1	17	2	..	..	..	..	..	Pass
19842000	1	18	2	..	..	..	..	..	Pass
..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..
				..	..	..	..	..	..

FIGURE 4.3: Dataframe with values of FE tests for each extracted chip. In columns we have chip position, FE tests and label.

4.3 Data Cleaning and Pre-processing

As explained in Chapter 2, a chip goes through hundreds of tests, for each test, we have multiple parameters, therefore taking into consideration all these tests when extracting the data, we end up with large number of features, from which we don't know the ones that are the most affecting the final status of our chip, and the ones that are rather useless. Indeed the dimensions of the extracted raw data are:

- **Features:** 3400 test parameter results.
- **Observations:** 38259 chips belonging to 7 different lots

In order to be able to make decisions regarding the features, and the feasibility of building a model based on these features, we went through studying the features one by one. In fact, since each feature is a test that is executed by an FSIST command (see Section 2.2.1), we split our data to several dataframe based on FSIST commands, i.e. dataframe with columns of Erase command tests, dataframe with columns of Verify command tests, etc. This make it easier to understand the features as the dataframes sizes was more reasonable, therefore instead of having to deal with a dataframe with thousands of features, we have multiple dataframes with couple of hundred features that's rather simpler to go through.
Different statistics were done on features, in order to take filtering decision, and determine which features are rather inuseful for the predicition and which are bringing us some information. The following sections are dedicated to go through the details of these statistics, the results we obtained, and the corresponding interpretations, organized as following:

- **Data Description**
- **Data correlation analysis**
- **Density distribution**
- **Filtering decisions**

4.3.1 Data Description

Going through the different parametric results we obtain in each test command, we can see that we are dealing with an heterogeneous dataset, i.e. the features of the dataset does not have the same units, nor the same types, as some data are continuous and others are discrete values. In fact, Figure 4.4 is a summary of the possible data types in our dataset, e.g current measures, number of SBER, etc. These features are more likely to have different range of values, which actually influence the choice of the machine learning model, as explained in Chapter 3. In order to get a flavor of the possible values a feature can have, we computed for each feature the following measurements, described in Chapter 3:

- **Mean and Standard deviation**
- **First quantile, median and third quantile**
- **Max and Min**

In case of data normalization, these measures were used as explained in Chapter 3. However, some of these statistics are reported in appendix.

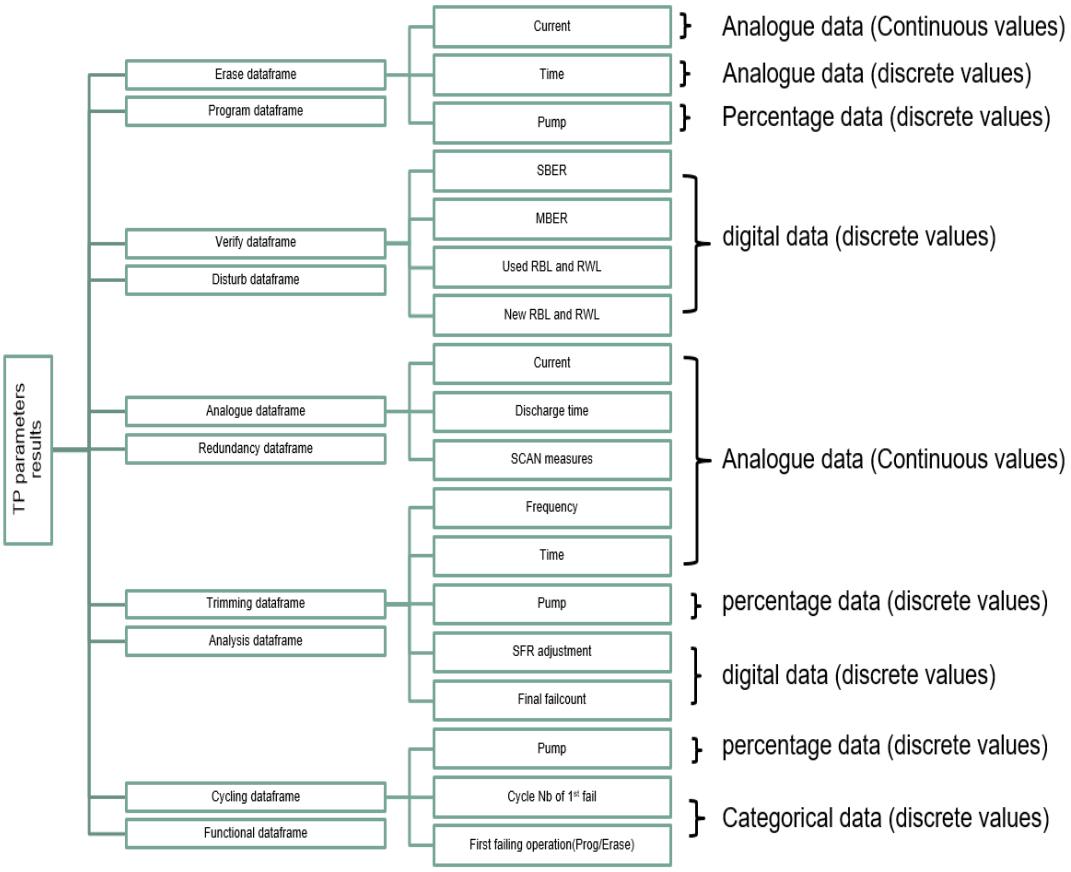

FIGURE 4.4: Possible types of test results

Data Correlation Analysis

Since the test-flow is composed of multiple repeated single tests, it is more likely that some of these tests are bringing the same information. Therefore, a correlation analysis is performed to omit variables bringing little information to the dataset. For every couple of features a Pearson correlation is computed as explained in section Chapter 3, yet the best way to visualize it is to plot a heatmap of the confusion matrix. However, since we have a large number of features, the correlations were computed subset by subset. The following matrix, Figure 4.5, is a correlation matrix of a subset of features. We can see that the feature:
Flash_T_B_FP_GD_VDDC_55131161_S1, which is a **G**ate **D**isturb test done on **P**rogram **F**lash (FP) using VDDC a power supply of $1.25V$ during S1 insertion, is highly correlated with the feature:
Flash_T_B_FX_GD_VDDC_55131881_S1, which is also a **G**ate **D**isturb test,

using VDDC, during S1 insertion, but on Data Flash (FX), indeed the correlation is almost equal to 1 in this case.
The feature Flash_T_B_FP_GD_VDDC_55131161_S1 is also correlated with correlation equal to 0.91 with the feature:
Flash_T_B_FP_DDS_ACC_VDDC_55131321_S1 which is another kind of disturb test (Drain Disturb Selected) done also on Program Flash with the same power supply. These correlations shows that the same test with the same conditions when done on different part of the flash will behave more or less the same. In addition, different tests although must be from the same FSIST command (e.g. in this case disturb and stress command), with the same conditions and during the same insertions, when done on the same part of flash (i.e in the aforementioned feature Program Flash) tend to be highly related. However, other features are shown very weak correlation, taking the same feature:
Flash_T_B_FP_GD_VDDC_55131161_S1 it is weakly correlated with:
Flash_T_B_FP_GD_VDD33_55131882_S1 which is the same test also done during S1 insertion but with different power supply , i.e $3.3V$. It also have a weak correlation with the feature:
Flash_T_B_FP_GD_VDDC_55130561_S2 which is the same test done with the same conditions but during S2 insertion. To conclude, tests are mostly related when done with the same conditions and during the same insertion.

Density Distribution

Another way to properly visualize and analyze the data was density plots, i.e. a representation of the density distribution of all possible values of a certain feature. The density was estimated using KDE (Kernel Density Estimation) technique explained in Chapter 3. In fact, to better interpret the features, we considered separately the values of the failing chips and the values of the passing chips, the density was estimated for both and plotted on the same graph. Figures 4.6 and 4.7 are two examples among the features we have, where the blue curve is the distribution of values of the feature for the chips that failed BE, and the orange curve is the distribution of values of that feature for the chips that passed BE. Looking into these two features, we can already conclude that some FE test parameters does indeed influence the BE final label, such as **Feature1** in Figure 4.6, whereas some others are not correlated in any way to the final label, such as **Feature2** in Figure 4.7. Similar plots were done for all the features in the dataset, these plots were analysed

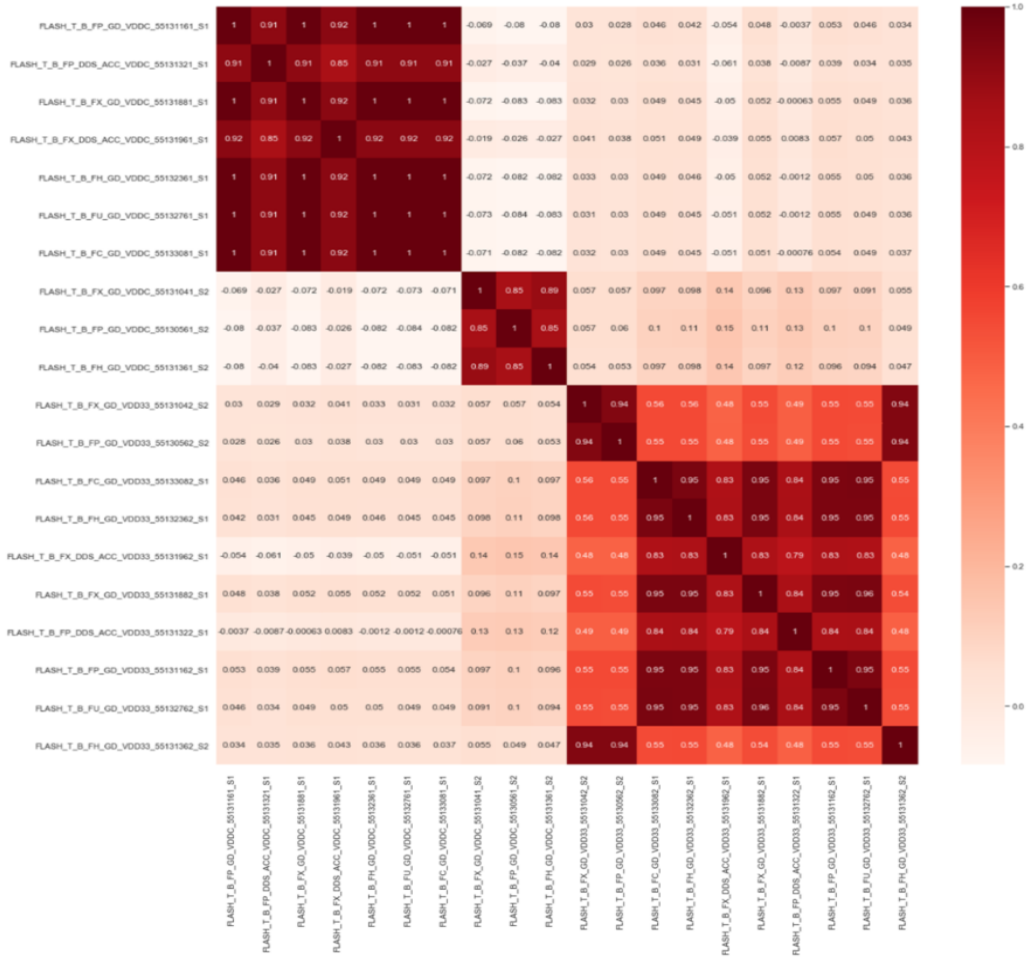

FIGURE 4.5: correlation matrix of subset of features

together by test engineers who rather confirmed certain features behaviour. In other words, from a domain knowledge, it is expected for a feature like **Feature1** that higher values may lead to the failure of the chip. Given this information, it is rather worthwhile considering some of the FE tests as features to predict the final BE label of a chip. However, in case of features with discrete values, we used barplots, as described in Chapter 3, to visualize the possible values of a feature and its probability of occurrence. Figure 4.8 is an example of discrete values feature, where a blue bar indicate the probability of occurrence of a certain value for Fail instances, whereas an orange bar indicate the probability of occurrence of a certain value for Pass instances, some values occur in both failing and passing instances and therefore the overlapped bars, the overlapped part is in purple.

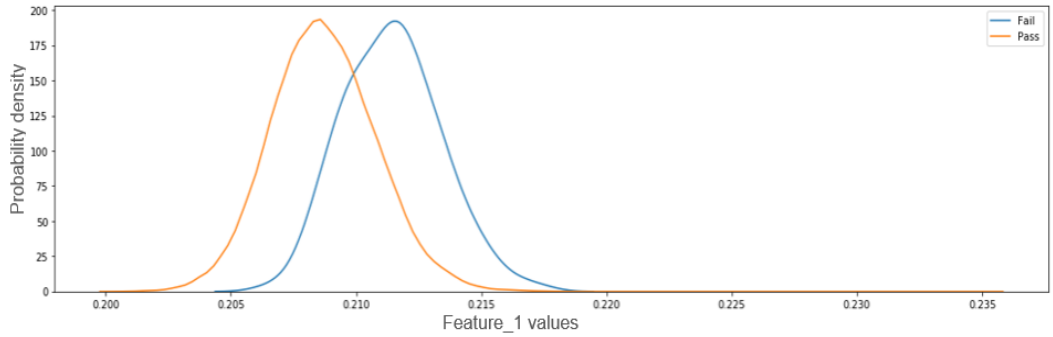

FIGURE 4.6: Density distribution of feature1 values

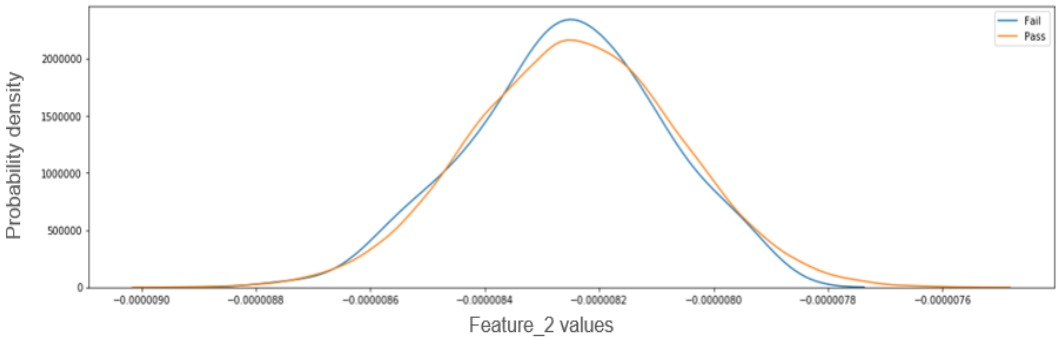

FIGURE 4.7: Density distribution of feature2 values

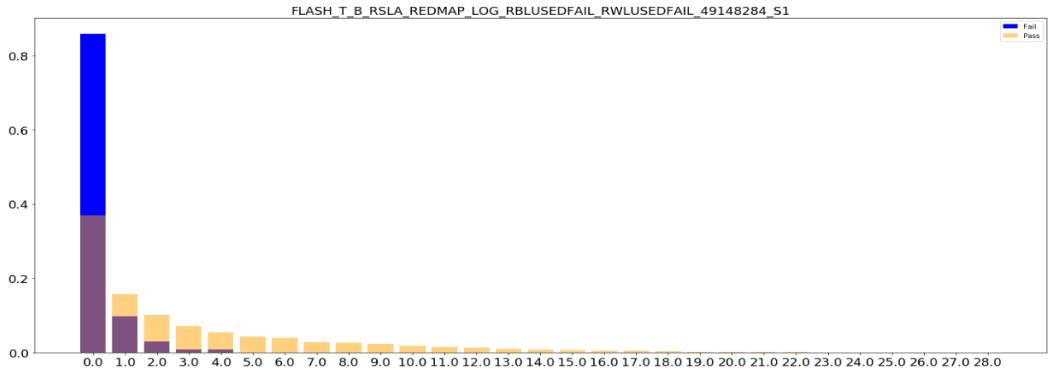

FIGURE 4.8: probability distribution of discrete values feature

4.3.2 Filtering decisions

According the the aforementioned statistics, FE test results do bring some information regarding the final label, yet many of them:

- are highly correlated
- have the same distribution of values for failing and passing chips
- have a standard deviation $\sigma = 0$

Such features are meaningless to keep. In fact, regarding the features that are highly correlated, these features may or may not bring information to the

model, however the fact that they are related, so whether we keep all of them or only one, the performance of the model will remain the same. On the other hand, the features that have the same distribution of values for failing and passing chips are actually not bringing any information on how to detect a class from another, because both classes are behaving the same way for that feature and therefore it is not possible to separate them based on it. Finally, the features with $\sigma = 0$ means that these features have the same value for all instances, so it is rather invariant to the model, indeed if instances have all the same value, there is no way to differentiate them.

Since there is no point in including features that does not contribute to predictions, it is better for storage and speed concerns to remove it. For that, we filtered our features based on the following decisions:

- Drop features that have some missing values for certain chips.
- Drop features with $\sigma = 0$
- Drop features where the distribution of Fail and pass have a KS statistic value below a certain threshold. See section, Chapter 3
- Drop features that have a correlation above 0.99, keeping only one.

4.4 Final dataset

After going through filtering, many useless features were dropped, and from thousands of features, we ended up with 275 feature that are actually bringing information to the model. The final dimensions of the dataset are:

- **Features:** 275 test parameter results.
- **Observations:** 38259 chips from 7 lots, where 134 are failing instances.

Chapter 5

Tool Development and Results

5.1 Introduction

This chapter is dedicated to present and analyze the results of the ML models, focusing on the performance of the selected model. The chapter is structured as follows:

- Comparing Models
- Metrics results of the selected model
- Final Feature set
- Quality Gate results

5.2 Model Selection

The metrics computed in this chapter in order to compare models and evaluate the selected one are obtained by using the following datasets explained in Chapter 3, Section 3.6

- **Trainset:** 26081 instances with 100 Fail and 25981 Pass
- **Testset:** 8694 instances with 34 Fail and 8660 Pass

In the following section, we analyze the results obtained by using different balancing techniques mentioned in Chapter 3, in terms of **TPR** and **FPR**. We first start with Random forest models, where Fig 5.1 presents blue and red boxplots which reports respectively the TPR, and FPR of the different balancing techniques used with RF over a a 10-fold stratified cross-validation (see Section 3.6) on the train set.
As explained in Chapter 3, the higher is the TPR the better is the model in detecting the Fail class, on the other hand the lower is the FPR the better is

the model in correctly classifying the Pass instances, a good model should have both a high TPR and low FPR.

First, we observe the performance in terms of TPR. In fact, looking into the TPR results in Fig 5.1, we can see that both BRF and RF with undersampling techniques are giving higher TPR compared to RF model with the SMOTE sampling technique. Indeed, when analyzing the three boxplots, we can see that BRF has values of TPR above 0.95, the RF model with undersampling techniques has TPR values around 0.9, whereas the RF model with oversampling technique is giving very poor TPR, where most TPR values are less than 0.15.

On the other hand, in terms of FPR, looking to the red boxplots in Fig 5.1,BRF has FPR values below 0.14, RF with undersampling has values of FPR below 0.17, and RF with SMOTE sampling technique has all values equal to zero, presented by the red line in the Fig 5.1.

Based on these measures, BRF and RF with undersampling techniques have both high TPR and low FPR, which makes them good models whereas the RF with SMOTE sampling technique has a very low TPR, and FPR around zero because it is actually just assuming all instances are Pass, thus no false positives which explains the FPR measure, yet a lot of false negatives with explains the low TPR.

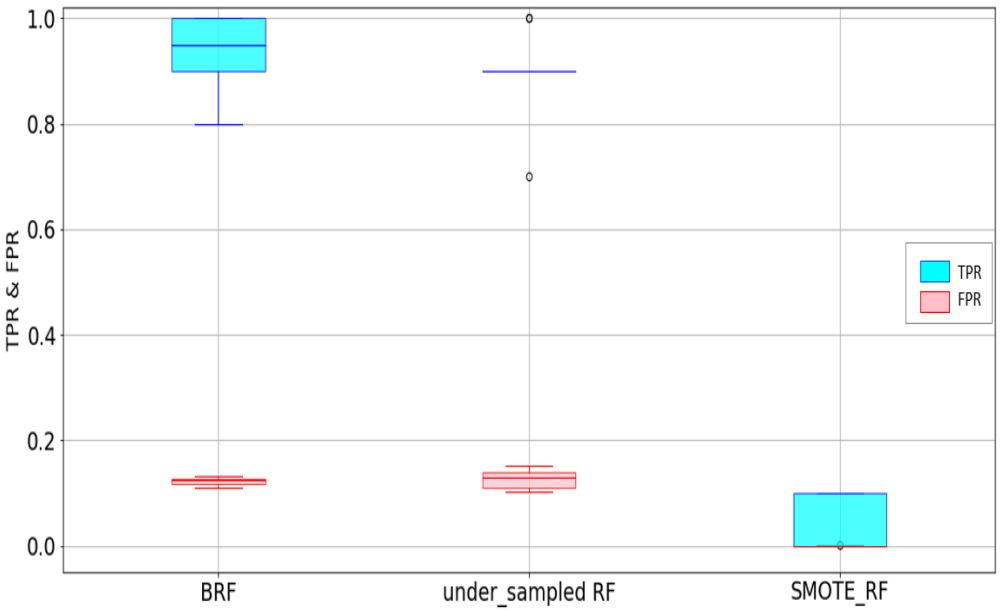

FIGURE 5.1: TPR and FPR of RF models evaluated over cross-validation

The same experiments were done using SVM model with undersampling technique and with oversampling technique, and are presented in Fig 5.2. In case of SVM as well, the oversampling technique isn't performing well, but it is rather biased to the Pass class, indeed FPR values are low yet TPR values are also very low. Regardless of the algorithm, RF or SVM, the oversampling techniques have mediocre performance. This could be explained by the fact that when oversampling, we are appending the original dataset, with either replications or interpolations of the fail instances, which risk to add noise to the data. Indeed, even in literature, recent research showed that undersampling seems to have an edge over over-sampling [5].

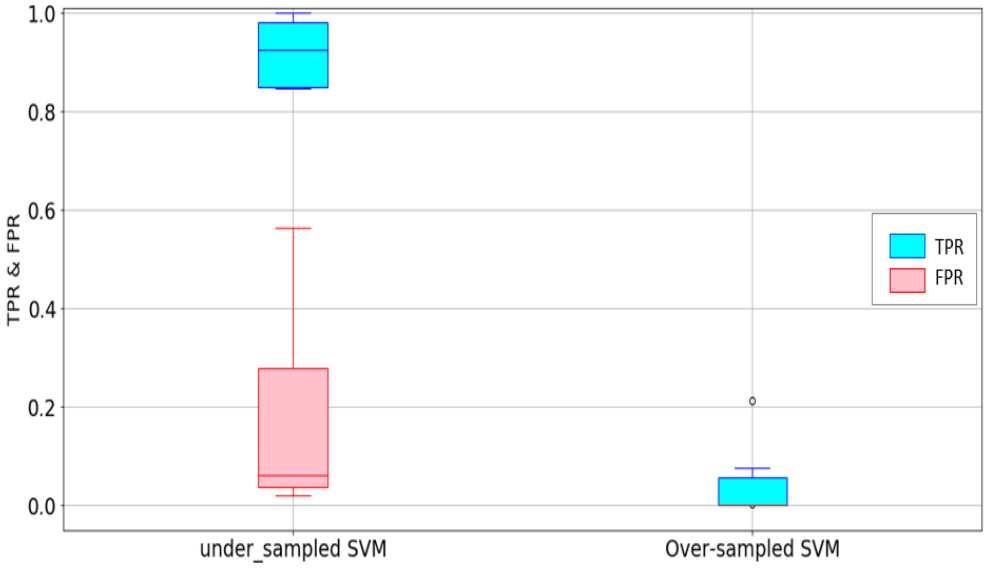

FIGURE 5.2: TPR and FPR of SVM models evaluated over cross-validation

Excluding the models with oversampling technique, the other models seem to have a good performance. In fact, combining the interesting results in Fig.5.1 and Fig 5.2 into one Figure presented in Fig.5.3, we observe that the BRF seems to perform slightly better than the others. In addition, BRF uses the entire dataset and does not reduce the number of instances as done in under-sampling techniques. Furthermore, the computational complexity of SVM is much higher than for RF [42], which make BRF the selected model for building our quality gate tool.

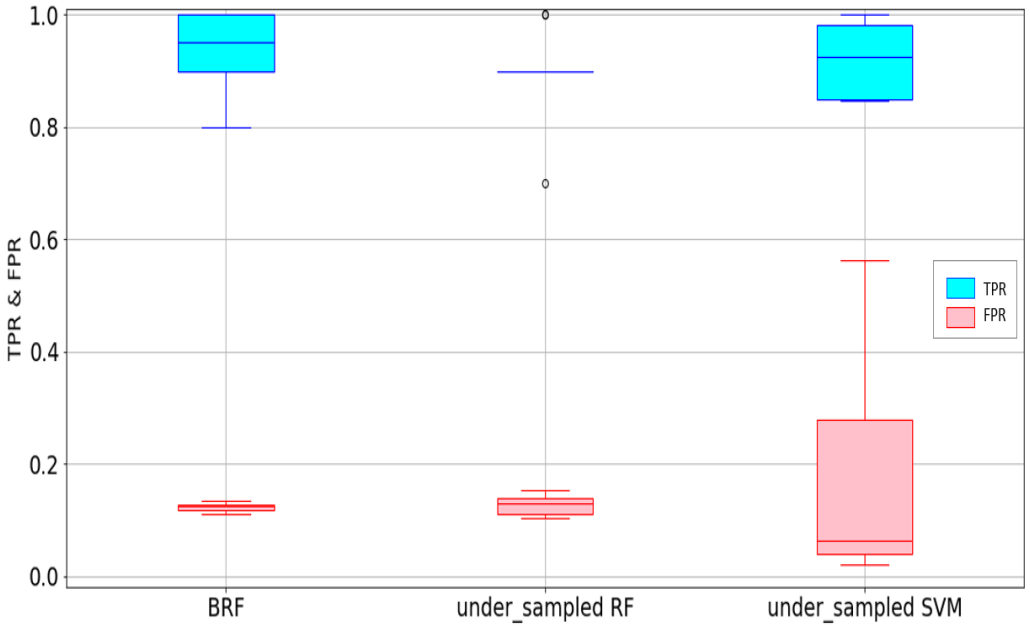

FIGURE 5.3: TPR and FPR of ML models evaluated over cross-validation

5.3 Balanced Random Forest Results

Based on previous comparison, and for computational time reasons [42], BRF was selected, and further evaluated to show that the prediction of BE failing chips based on FE tests using machine learning techniques is feasible. The following performance metrics will therefore focus only on the BRF (see Section 3.4.2 and Section 3.4.1), although they were also computed for the other models and are provided in Appendix A.

5.3.1 Performance metrics

Figure 5.4 displays the ROC curves obtained over 10-fold stratified cross validation (see Section 3.6), we can see that the curve increases quickly from 0 to 1, i.e we can reach a high TPR, around 0.8, while keeping the FPR very close to 0. The curve shows that, the model is able to well detect the fail class without the need to sacrifice too much Pass instances. On the other hand, the AUC is very high (i.e 0.95) indicating that the BRF is indeed a good model to detect the state of a chip.

In order to better visualize the separability of the predicted classes, we plot the distributions of their predicted probabilities to fail as explained in Chapter 3. In Fig 5.5, we can see that some real pass chips have a probability

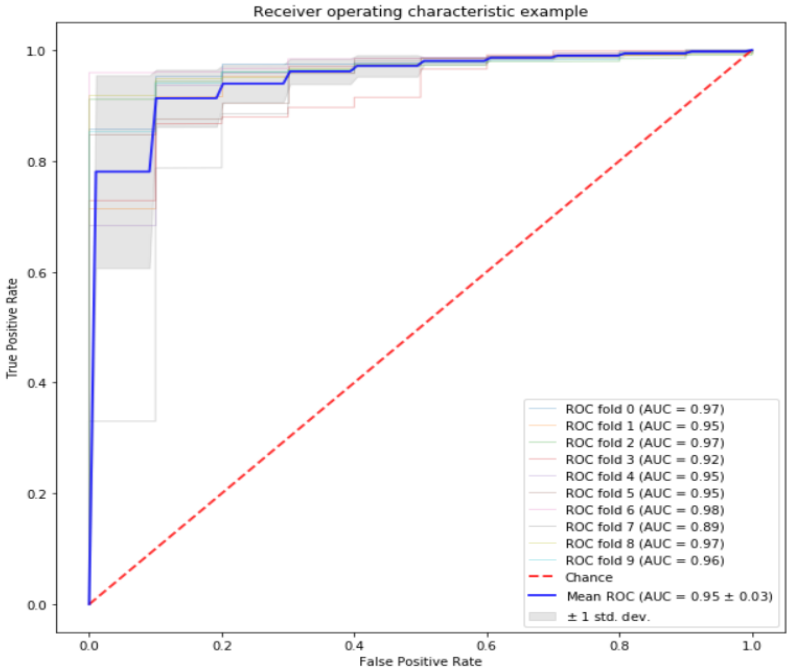

FIGURE 5.4: BRF with cross-validated ROC curve

to fail above 0.5, if the decision threshold is set to 0.5 these instances will be missclassified as failing chips, and are noted in the figure as FP. On the other hand, some fail instances have a predicted probability to fail below 0.5 and therefore in this case are classified as Pass, and noted in the figure as FN. However, the two distributions aren't too much overlapped, and the missclassified instances are actually few, indeed the exact number of FP and FN could be seen in the confusion matrix in Table 5.1. Given the new test set of 8694 instances, BRF was able to identify 32 out of 34 fail instances, i.e the model can detect 94% of the fail chips. The results of the confusion matrix are determined based on a decision threshold equal to 0.5.

	Predicted as Fail	Predicted as Pass	
True Fail	32 (TP)	2 (FN)	**TPR**=0.94
True Pass	1101 (FP)	7559 (TN)	**FPR**=0.127

TABLE 5.1: Confusion matrix using BRF with threshold 0.5

5.3.2 Decision Threshold Adjustment

When a BRF returns 0.9 probability to fail for a particular chip, it means that chip is very likely to fail. Conversely, another chip with a predicted probability to fail of 0.001 is very likely to pass. However, some chips has probabilities

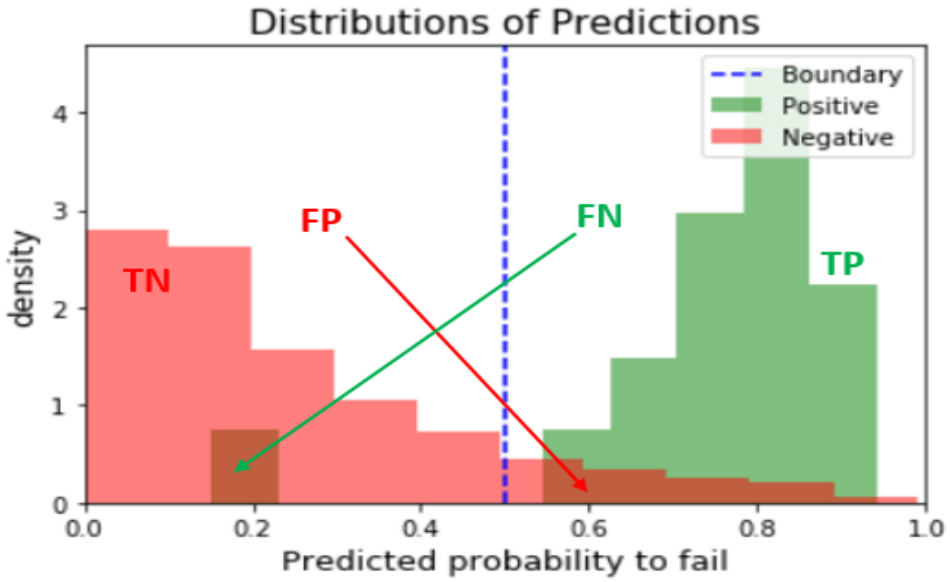

FIGURE 5.5: distribution of predicted probabilities to fail with BRF

around 0.6, those chips will be classified as "Fail" if the decision threshold is set to 0.5, yet they do not have a very high probability to fail compared to the ones around 0.8 and 0.9. Nevertheless, this threshold could be adjusted to meet our needs, i.e we can adjust the threshold to decide when a chip should or should not be considered as fail depending on its predicted probability to fail. In fact, part of choosing a threshold is assessing how much we'll suffer for making a mistake, for example there might be a chip that is a true Fail and has a predicted probability to fail equal to 0.6, it will be correctly classified if the threshold is set to 0.5 however if we vary the threshold to 0.7 it will be missclassified as Pass, thus when choosing the threshold we have to choose which is worse, mistakenly labeling a fail chip as pass or labeling a pass chip as fail.

In Fig. 5.5, we see that most fail instances has a probability to fail above 0.7, as well as most pass chips has a low probability to fail, except for some pass instances that was predicted to have a higher than 0.5 probability to fail. Hence can adjust the threshold to higher than 0.5 to reduce the number of FP while only slightly sacrificing the TP. In the following tables, 5.2,5.3 ,5.4, 5.5 ,the confusion matrices obtained by varying the threshold are presented.

According to these results, and as also seen in the ROC curve in 5.4, we

	Predicted as Fail	Predicted as Pass	
True Fail	30	4	**TPR**=0.88
True Pass	773	7887	**FPR**=0.09

TABLE 5.2: Confusion matrix using BRF with threshold 0.6

	Predicted as Fail	Predicted as Pass	
True Fail	28	6	**TPR**=0.82
True Pass	608	8052	**FPR**=0.07

TABLE 5.3: Confusion matrix using BRF with threshold 0.65

	Predicted as Fail	Predicted as Pass	
True Fail	25	9	**TPR**=0.74
True Pass	435	8225	**FPR**=0.05

TABLE 5.4: Confusion matrix using BRF with threshold 0.7

	Predicted as Fail	Predicted as Pass	
True Fail	18	16	**TPR**=0.53
True Pass	210	8450	**FPR**=0.02

TABLE 5.5: Confusion matrix using BRF with threshold 0.8

can reach very low values of FPR without degrading too much the TPR. Indeed, with a 0.65 threshold we can reduce the number of missclassified pass chips from 1101 to 608 while the model can still identify around 82% of the fails. The threshold decided depends on the requirements of the QG tool, and will be discussed in Section 5.5.

5.4 Feature Selection

Although $\sim$ 200 informative features seems to be a reasonable number to train a ML model, it will be simpler to have a reduced number of features because it is easier and faster for the test engineers to interpret this set of killer features . While all these features are somehow contributing in the final state of the chip, some of them are more important than other and can be enough to predict the Fail or Pass of the chip. In fact, when exploiting the random forest *feature_importances* attribute (see Section 3.3, [27]), we can rank

our features based on their importance score.
In order to see the influence of these features, we tested our model, using:

- The 5 worst ranked features
- The 20 worst ranked features
- all features
- The 20 best ranked features
- The 10 best ranked features
- The 5 best ranked features

The boxplots of the TPR and FPR respectively blue and red boxplots obtained by BRF over 10-fold cross-validation (see Section 3.6), are presented in Fig. 5.6. Looking to the boxplots, we can see that the performance of the model using only the 5 worst features decreases significantly as most TPR values over the 10 splits are above 0.6 and most FPR values are above 0.25. The same goes for using the model with the 20 worst features, although it is better than using only the 5 worst since we are adding 15 other more informative features, the performance is still mediocre compared to using all features.

On the other hand, looking into the boxplots in case of using the model with the top best features, it seems that the performance is more or less the same. In fact, over 75% of TPR values over 10 splits are between 0.8 and 1 for the three models, i.e with all features, with top 20 best features, with top 10 best features and with top 5 best features. Moreover, analyzing the red boxplots of the FPR values, we see that from using all features to only using the top 20, the FPR values are quite close, i.e values around 0.14, yet the FPR values are higher using only top 10 or 5 features, i.e around 0.2 . These results can lead us to think that using only the top important features, the model is still able to identify the fail chips the same way, but the less features, the more the model can miss-classify the pass instances as fails too, thus the model is slightly less precise. In addition, the fact that results are hardly changing whether the model is with all features or only top 20 features could be explained by the high correlation between the features, as seen in Section 4.3.1 in Fig. 4.5, many features have a correlation around 0.8, whereas we only dropped the ones with correlation above 0.99, these features could be as well bringing more or less the same information, which could explains why performance is insignificantly changing when reducing the number of

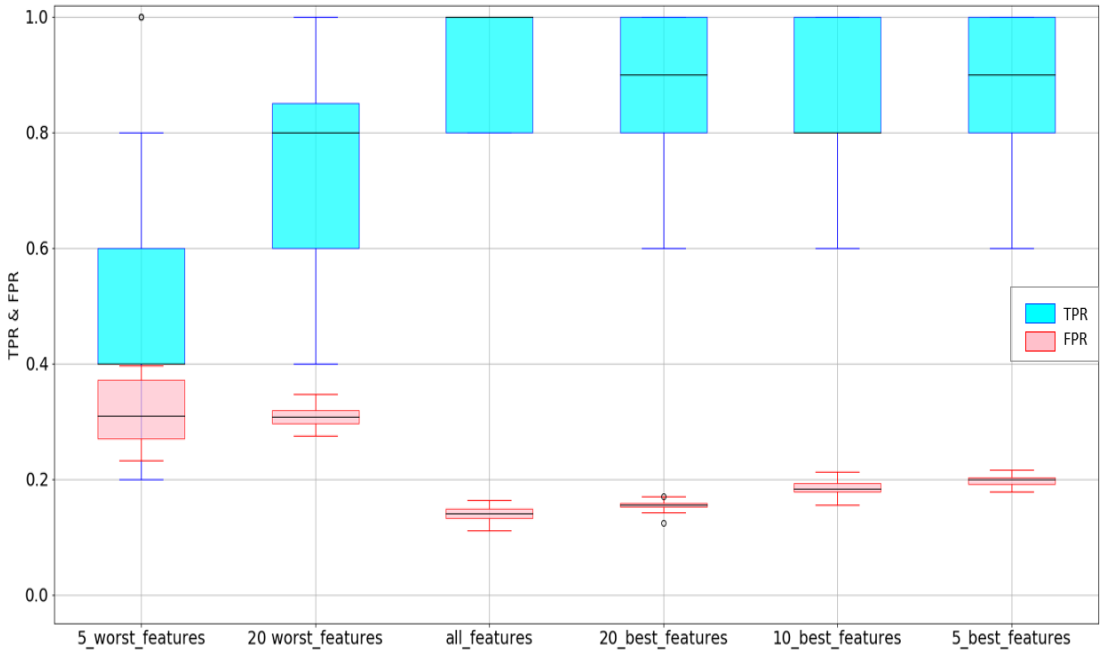

FIGURE 5.6: TPR & FPR with different number of features by BRF over 10-fold cross validation

features to 20. In this thesis, we build our Quality Gate tool using BRF with only the top 20 best features, since it is performing as well as the model with all features, and much faster to train and predict, however we believe that further studies regarding this topic are needed.

5.5 Quality gate tool

The goal of this thesis, is to build a quality gate tool for Infineon's test program. Indeed, as explained in Chapter 2, dies go through FE testing then BE testing which completes NVM test coverage. Thus the QG tool will predict the final state of the chip based on FE testing and compare these predictions to final state assigned by BE testing. When the final label given by the production data and the one predicted by the QG tool are different, this rises a doubt regarding the test coverage of the test program, i.e the ability of the test to detect a given set of faults that may occur on the DUT. The more conformity we get between predictions and real labels, the more we trust our testing.

The QG tool should take in input new chips, predict their final state and

compare it to the real label given to it after BE testing, and assign it a tag accordingly.

- If a chip is actually **Pass** but the QG tool predict it as **Fail**, it will mark it as *Suspicious*.
- If a chip is actually **Pass** and the QG tool predict it as **Pass**, it will mark it as *Certainly Pass*
- If a chip is actually **Fail** and the QG tool predict it as **Fail**, it will mark it as *Certainly Fail*
- If a chip is actually **Fail** and the QG tool predict it as **Pass**, it will mark it as *Undetected Fail*

A way to display the results of the quality gate tool is to consider a wafer level. The tool can take as input the test results of chips belonging to certain wafer, their corresponding real labels, a threshold to decide whether a chip is considered as fail or pass as explained in Section 5.3.2. The tool will therefore compute a graphical representation of the wafer with the state of each chip as seen in Fig. 5.7. In this example we use the *wafer* 6 from the *lot* 19841006 with a threshold equal to 0.6. The green chips corresponds to the ones that are *Certainly Pass*, the red chips correspond to the ones that are *Certainly Fail*, the blues are the ones that are *Suspicious*, the black are the *Undetected Fail* and the ones in grey are the chips that failed during the FE testing and therefore thrown away and did not undergo BE testing.

Looking into Fig. 5.7 we notice that failing chips tend to cluster on the wafer, indeed they are more concentrated in the center and the boarders of the wafer. In fact, chips that are in the boarder are already failing during the FE testing. On the other hand, chips close to the center are either failing during FE, or failing during BE, such as the red chips in the figure. Passing chips are more concentrated in the middle, nor in the boarders nor close to the center, this lead us to think that the final state of the chip is affected by its position in the wafer. For that, in the following section we discuss the possibility and necessity of adding the distance from the center of the chip as an additional feature to predict the final state.

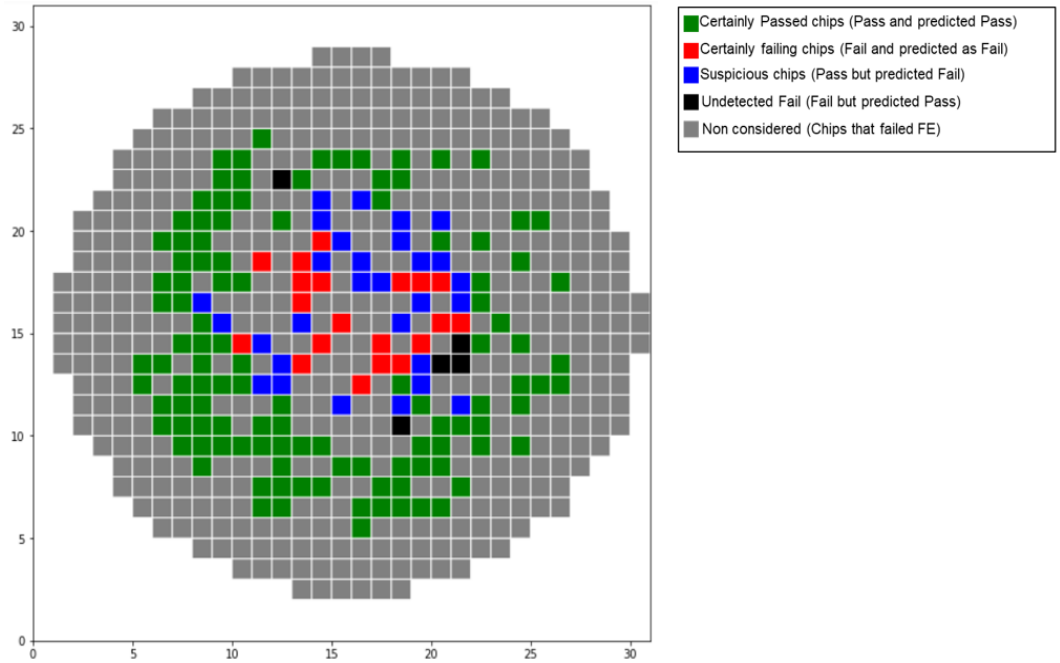

FIGURE 5.7: Quality gate tool on a certain wafer

5.6 Distance of Chip from The Center of Wafer

In this section, we briefly study the contribution of distance from the center as feature to the overall predictions. FE failing chips are usually the ones in the boarders of the wafer as previously seen in Fig 5.7 and confirmed by the test engineers in Infineon. However, the following analysis will focus on the influence of the position of chip on failure of the chip in BE testing, thus passing chips and FE failing ones are not considered.

Fig 5.8 is a representation of the number of BE failing chips in each XY position. Among 134 failing chips in our dataset, we computed the number of total failing chips in a specific XY position. Chips that passed or failed in FE are not considered. We can see that most dies that failed the BE are closer to the center.

In fact, the same could be seen by plotting the density distribution of distances of BE failing chips and BE passing chips as in Fig 5.9. The blue curve of the chip Failing BE indicates that the closer the chip is from the center the likelier it is to fail the BE test.

These results motivated us enough to add to our set of features the distance from the center of the wafer, and analyze the performance of the BRF model in predicting the state of chips. The following Fig 5.10 shows the results in terms of TPR and FPR of the model with and without adding the

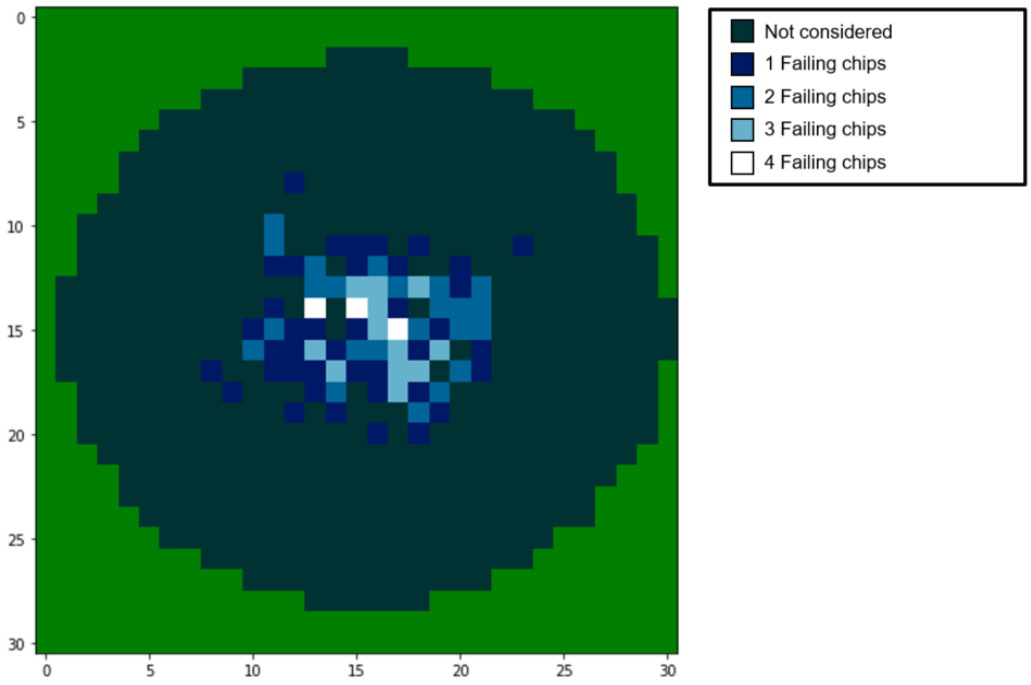

FIGURE 5.8: Number of total failing chips in a specific XY position among 134 failing chips

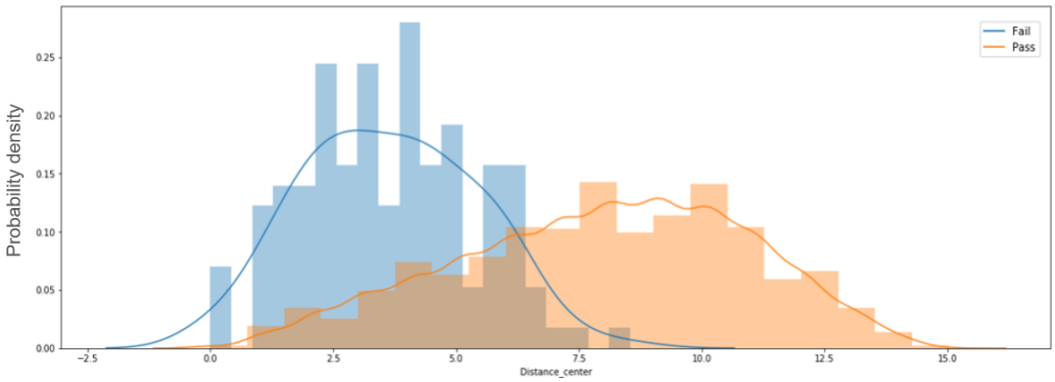

FIGURE 5.9: Density distribution of distances of FE passing chips from center

Distance as feature. The performance of the BRF is hardly changing whether we add or not the Distance feature, indeed TPR values are the same for both cases, however FPR values are slightly lower in case of adding the Distance feature. Although Distance seemed to influence the failure of the chip according to the previous analysis, adding it to the other features does not enhance the performance of the model.

This could be explained by the fact that Distance feature could be correlated with the rest of the features, and thus the same results are obtained with

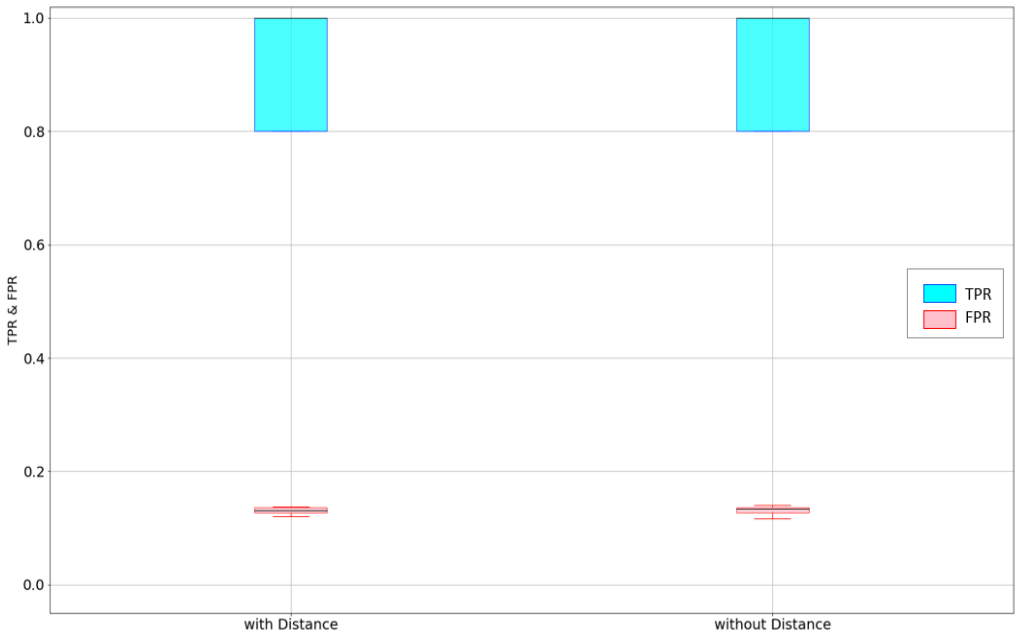

FIGURE 5.10: TPR & FPR with BRF with and without using the feature *Distance* over 10-fold cross-validation

or without it. In fact, looking into Fig 5.7, these results here are obtained without considering the Distance among features, however chips tend to have a certain pattern on the wafer, i.e concentrated in the center, in other words our model is predicting that chips in the center are going to fail without actually having any information regarding their position but only based on the FE tests as features, which explains the metrics values in Fig 5.10. Nevertheless, this is encouraging to further study a possible combination of this spatial feature with the tests results. As a matter of fact, other works [18] considered only chip position related features to predict its state, conversely in our work we only considered test results, however it is interesting for future work to study a combination of both as features.

Chapter 6

Conclusion and Future Work

In this chapter we first summarize results and contributions of this thesis. We then propose the next steps to finalize the Quality Gate tool.

6.1 Summary and Contributions

In this thesis, we have applied machine learning techniques to Infineon testing production data in order to predict the final state of a packaged chip, i.e Fail/Pass after BE testing based on tests results obtained in the FE testing, and we want to use these predictions to build a Quality Gate tool for the Test Program.

The main challenges faced in this work were class imbalance, feature analysis and selection, and model selection. In Chapter 3, we proposed different techniques to deal with the issue of imbalanced data as well as the suitable performance metrics for this case. In Chapter 4, we explained how data was extracted from Infineon's database, processed and analyzed. The different analysis done on features were a crucial part of this thesis work, as it showed the valuable information provided by FE testing to BE testing, in other words the influence of FE tests on BE, thus the feasibility of the QG tool. Chapter 5 included the results of the different ML models discussed in Chapter 3, where the BRF showed the best performance. The implementation and the output of the QG tool were also presented in chapter 5.

In conclusion, the main contributions of this thesis can be summed up as following:

- Preparing a complete script for data extraction.
- Set-up an entire data analysis framework for the FE tests, to examine the density distribution of test results for failing and passing chips, also to observe and understand the correlations between the different tests. (Chapter 4)

- Studying and isolating a set of FE tests that influence the Fail/Pass state of the chip in BE, which we refer to as killer parameters.
- Proposing and building a ML trained model that is able to predict the Fail/Pass state of a chip, with specificity and sensitivity around 0.9
- Enabling a QG tool realization and feasibility that can take in input the data of a new wafer and output a graphical representation of it with the state of each chip. (Section 5.5)

6.2 Future Work

The next step of the QG tool is to integrate it within YETI website (**Y**ield **E**xtractor and **T**est program **I**dentifier), where data should be extracted in real time, and the output of the QG tool should be updated based on the data available. In this thesis, the set of killer features among all FE tests results are identified, therefore it will be easier for next step to directly take into consideration these features. Next step will mainly focus on:

- Improving the ML model performance by combining FE test results and spatial features as mentioned in Section 5.6.
- Continue with further investigations regarding the neural networks already initiated, as described in Section 3.4.2.
- studying the feasibility of considering unsupervised learning.
- Set the data extraction script to extract periodically and automatically the dataset required..
- Integrate the results of the QG tool in YETI website

Appendix A

Appendix

A.1 SBIN analysis

Depends on the root of failure, a chip may be assigned different SBIN. In Fig A.2, the blue curve presents the distribution of values of the feature for the chips that failed with SBIN555, the orange curve presents the distribution of values of the feature for the chips that passed, the green, purple and red curves indicate respectively the distribution of values of the feature for chips that failed with SBIN588, SBIN521 ,SBIN566. In fact chips with SBIN555 seems to have more different behaviour from the pass chips than the other SBINs. This could be confirmed by the boxplots presented in Fig A.2, where the TPR is higher when applying the BRF to detect SBIN555 failures from Pass.

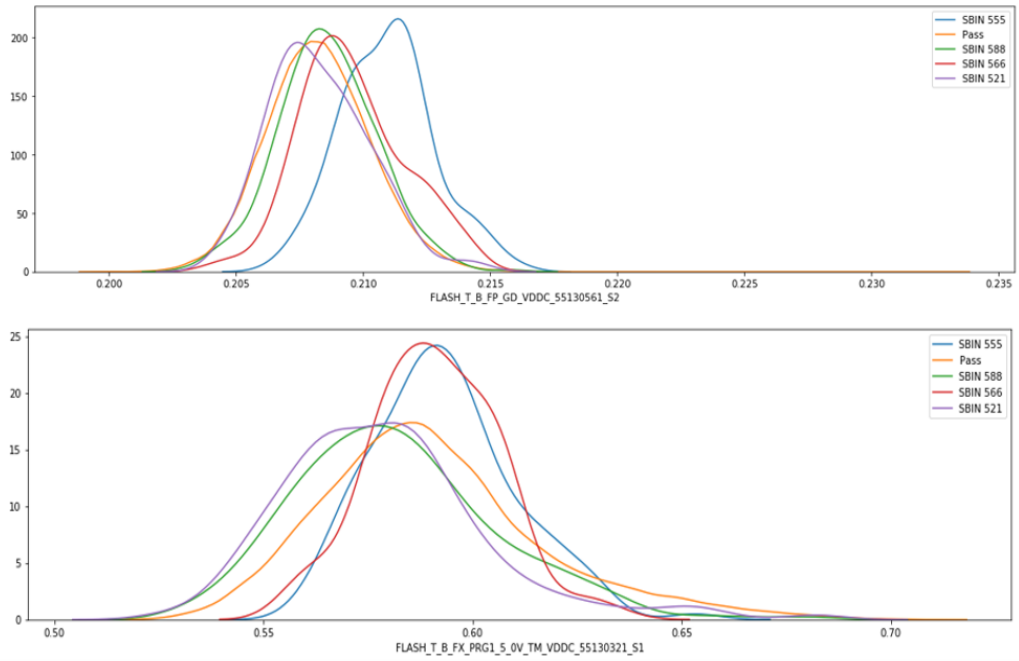

FIGURE A.1: Density distribution of two features with different SBINs

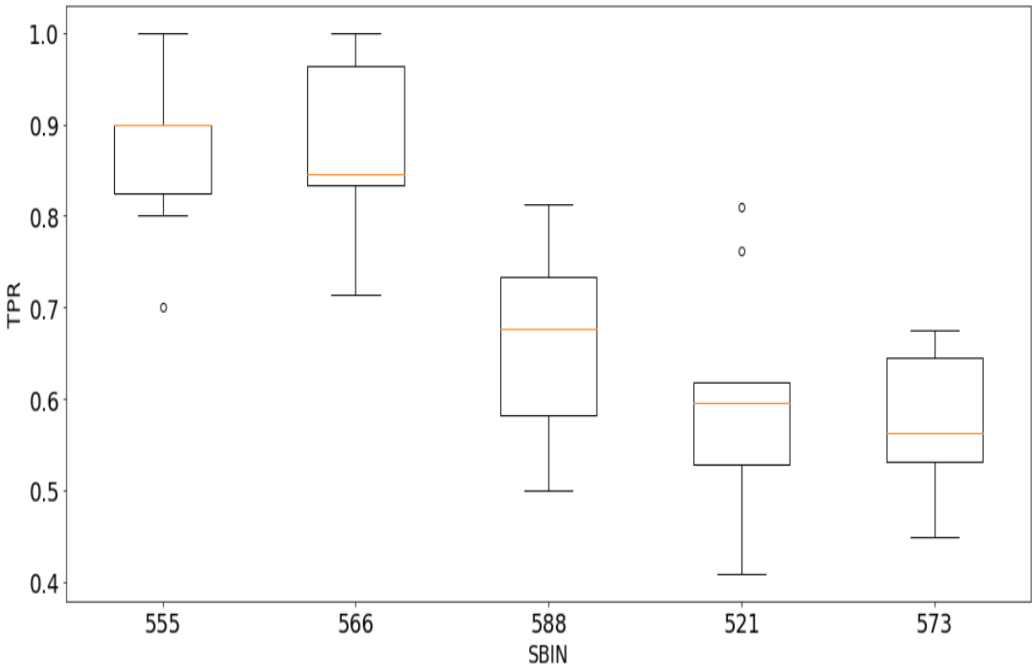

FIGURE A.2: TPR of BRF when detecting different SBINS from Pass

A.2 Data Statistics

Different statistics were computed for all features, such as standard deviation, Max, Min and Median (aka 50% quantile) the following Fig A.3 presents the different measures for a subset of features.

	count_zeros	Mean	Std	Max	Min	25% quantile	50% quantile	75% quantile
FLASH_T_B_REDMAP_LD_MAP_RBLUSEDFAIL_RWLUSEDFAIL_52531763_S1	15222	5.041467e+00	9.791286e+00	1.150000e+02	0.000000e+00	0.000000e+00	1.000000e+00	5.000000e+00
FLASH_T_B_RSLA_REDMAP_LOG_RBLUSEDFAIL_RWLUSEDFAIL_49148284_S1	12886	2.977886e+00	4.010153e+00	2.800000e+01	0.000000e+00	0.000000e+00	1.000000e+00	4.000000e+00
FLASH_R_D_CNT_SGVT_1_32_ST200_VT_SG_V0_1_60V_55130444_S2	0	1.864138e+06	1.192819e+06	1.106233e+07	2.725000e+04	1.024358e+06	1.586848e+06	2.367064e+06
FLASH_T_B_FX_E_TM_VDDC_55130801_S2	0	2.087332e-01	2.104776e-03	2.350427e-01	2.005495e-01	2.072650e-01	2.086386e-01	2.101648e-01
FLASH_T_B_FU_E_PSEUDO_UM_VDDC_55131961_S2	0	2.103412e-01	2.118782e-03	2.339744e-01	2.019231e-01	2.089438e-01	2.103175e-01	2.116911e-01
FLASH_T_B_FP_GD_VDDC_55130561_S2	0	2.082985e-01	2.052089e-03	2.330586e-01	1.999390e-01	2.069597e-01	2.081807e-01	2.095543e-01
FLASH_R_D_CNT_SGVT_1_32_ST200_VT_SG_V0_1_20V_55131042_S1	0	1.805767e+07	2.119502e+05	1.821885e+07	1.210956e+07	1.802082e+07	1.812015e+07	1.817306e+07
FLASH_T_B_FH_E_TM_VDDC_55131281_S2	0	2.085310e-01	2.102193e-03	2.350427e-01	2.003968e-01	2.071123e-01	2.084860e-01	2.098596e-01
FLASH_T_B_FC_E_PSEUDO_UM_VDDC_55131521_S2	0	2.103364e-01	2.113516e-03	2.327534e-01	2.022283e-01	2.089438e-01	2.103175e-01	2.116911e-01
FLASH_T_B_FH_GD_VDDC_55131361_S2	0	2.101745e-01	2.110783e-03	2.368742e-01	2.020757e-01	2.087912e-01	2.101648e-01	2.115385e-01
FLASH_T_B_FX_GD_VDDC_55131041_S2	0	2.085274e-01	2.098165e-03	2.335165e-01	2.002442e-01	2.071123e-01	2.084860e-01	2.098596e-01

FIGURE A.3: Table of different statistics for a subset of features

A.3 Performance metrics of SVM with under-sampled balanced data

SVM with under-sampled data showed more or less good results, yet not as good as BRF. In following, we present the different performance metrics we computed for this model.

ROC curve

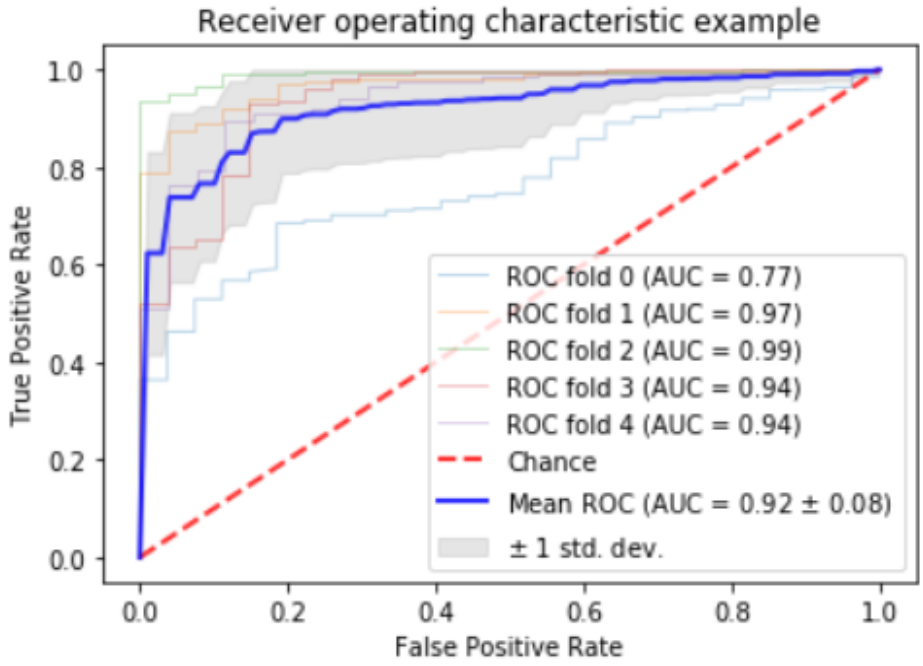

FIGURE A.4: ROC curves over 5-fold cross validation with SVM using under-sampling technique

Confusion Matrix

	Predicted as Fail	Predicted as Pass	
True Fail	32	2	**TPR**=0.94
True Pass	3672	4988	**FPR**=0.42

TABLE A.1: Confusion matrix using SVM with threshold 0.5

We can see in Table A.1, that tested on new data, SVM is able to identify Fail instances as good as BRF but it is much less precise, as it missclassifies

many pass instances as Fail too, e.g 3672 from 8660 Pass instances are miss-classified, thus a high FPR.

Predicted Probabilities Plot

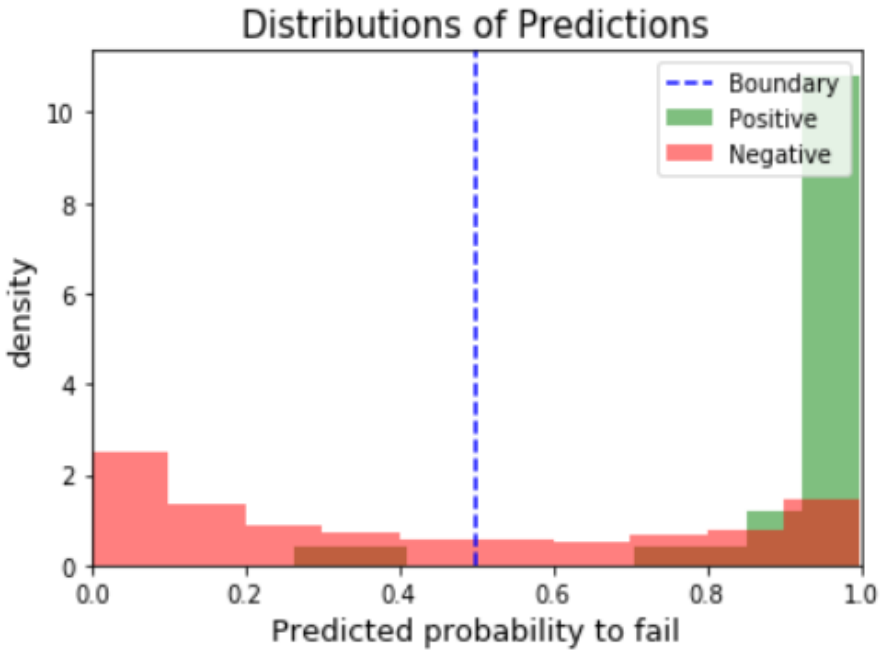

FIGURE A.5: Distribution of predicted probabilities to fail with SVM

A.4 List of Features

The following is the list of the **20 worse ranked features:**

- FLASH_T_B_FP_PRGCHK_5_0V_TM_VPPMAX_VDPMAX_VPNMAX_NU_55131525_S1
- FLASH_T_B_FP_V0P_M2_VM_RBLUSEDFAIL_RWLUSEDFAIL_56400127_S1
- FLASH_T_B_FU_E_PSEUDO_UM_VPPMAX_VPNMAX_NU_NU_55133964_S1
- FLASH_T_B_FC_GD_VPPMAX_VDPMAX_VPNMAX_NU_55133123_S1
- FLASH_R_D_CTCT_DEBUG_DUMP_EXECUTION_TIME_52230281_S1
- FLASH_T_B_FX_DDS_ACC_VPPAVG_VDPAVG_VPNAVG_NU_55132007_S1
- FLASH_T_B_FP_DDS_ACC_VPPAVG_VDPAVG_VPNAVG_NU_55131367_S1
- FLASH_R_D_CNT_MSCAN_8_32_ST2_MSCAN_20UA_55131127_S1

- FLASH_R_D_CNT_DISCTP_DEBUG_PF_DISCHT_COMM1_52033841_S1
- FLASH_R_D_CNT_DISCTD_DEBUG_DF_DISCHT_COMM1_52034521_S1
- FLASH_T_B_FP_E_TM_VPPMAX_VPNMAX_NU_NU_55130804_S1
- FLASH_T_B_FX_PRG1_5_0V_TM_VPPMAX_VDPMAX_VPNMAX_NU_55130684_S1
- FLASH_T_B_FP_E_TM_VPPAVG_VPNAVG_NU_NU_55130128_S1
- FLASH_T_B_REDMAP_LD_MAP_REDDBLUSEDFAIL_REDDWLUSEDFAIL_52532729_S1
- FLASH_T_B_REDMAP_LD_MAP_RBLUSEDFAIL_RWLUSEDFAIL_52531764_S1
- FLASH_R_D_CTCT_DEBUG_DUMP_CTCT_RETURN_CODE_52230362_S1
- FLASH_T_B_FX_V0S_M3_SBER_57730123_S1
- FLASH_A_I_IREF_M_MDACFP1_P02_1_52031080_S2
- FLASH_T_B_FX_CTCT_VPP_SG_CTCT_VPPSG_LUTIDX_52230243_S1
- FLASH_T_B_FC_E_PSEUDO_UM_VPPMAX_VPNMAX_NU_NU_55131564_S2

The following is the list of **20 best ranked features**:

- FLASH-T-B-FX-GD-VDDC-55131041-S2
- FLASH-T-B-FH-GD-VDDC-55131361-S2
- FLASH-T-B-FC-E-PSEUDO-UM-VDDC-55131521-S2
- FLASH-T-B-FH-E-TM-VDDC-55131281-S2
- FLASH-R-D-CNT-SGVT-1-32-ST200-VT-SG-V0-1-20V-55131042-S1
- FLASH-T-B-FP-GD-VDDC-55130561-S2
- FLASH-T-B-FU-E-PSEUDO-UM-VDDC-55131961-S2
- FLASH-T-B-FX-E-TM-VDDC-55130801-S2
- FLASH-R-D-CNT-SGVT-1-32-ST200-VT-SG-V0-1-60V-55130444-S2
- FLASH-T-B-RSLA-REDMAP-LOG-RBLUSEDFAIL-RWLUSEDFAIL-49148284-S1
- FLASH-T-B-REDMAP-LD-MAP-RBLUSEDFAIL-RWLUSEDFAIL-52531763-S1
- FLASH-R-D-CNT-SGVT-1-32-ST200-VT-SG-V0-1-40V-55130443-S2

- DEVCFG-T-B-DUMP-FPC-BINGRADE-APGC-55135003-S1
- FLASH-T-B-RSLA-REDMAP-LOG-RBLUSEDFAIL-RWLUSEDFAIL-49148243-S1
- FLASH-R-D-CNT-SGVT-1-32-ST200-VT-SG-V0-1-20V-55130442-S2
- FLASH-T-B-FPPS00-SGVT-SCAN-SBER-55131003-S1
- FLASH-R-D-CTCT-DEBUG-DUMP-EXECUTION-TIME-52230441-S1
- FLASH-T-B-FP-PRGCHK-5-0V-TM-VDDC-55130641-S2
- FLASH-R-D-CNT-SGVT-1-32-ST200-VT-SG-V0-1-00V-55130921-S2
- FLASH-T-B-FP-E-TM-VDDC-55130321-S2

A.5 Snapshot of Real Data

Lot	Wafer	X	Y	SBIN	FLASH_T_B_RSLA_REDMAP_LOG_RBLUSEDFAIL_RWLUSEDFAIL_49148243_S1	RBLUSED	RBLUSED	ADC_TRIM	RIM_REG	ADC_TRIM	MEAS_AD	MDACFP0
19841006	1	19	13	1	0	0	0	1,173381	7	1,189254	1,20879	-8,4E-06
19841006	1	16	25	1	0	0	0	1,173381	7	1,190475	1,211232	-8,3E-06
19841006	1	8	14	1	0	0	0	1,19658	3	1,214895	1,210011	-8,5E-06
19841006	1	17	24	1	1	0	1	1,173381	7	1,188033	1,20879	-8,6E-06
19841006	1	6	13	1	2	0	2	1,201464	2	1,222221	1,212453	-8,1E-06
19841006	1	15	13	1	0	0	0	1,195359	3	1,189254	1,210011	-8,5E-06
19841006	1	12	26	1	10	0	4	1,175823	7	1,186812	1,213674	-8,3E-06
19841006	1	23	12	1	0	0	0	1,181928	6	1,170939	1,210011	-8,2E-06
19841006	1	10	25	1	4	0	4	1,199022	2	1,206348	1,211232	-8,3E-06
19841006	1	6	14	1	2	0	2	1,205127	1	1,192917	1,20879	-8,5E-06
19841006	1	19	25	1	0	0	0	1,175823	7	1,183149	1,20879	-8,3E-06
19841006	1	10	14	1	0	0	0	1,192917	3	1,203906	1,20879	-8,2E-06
19841006	1	9	11	1	2	0	2	1,189254	4	1,178265	1,210011	-8,3E-06
19841006	1	16	14	1	0	0	0	1,194138	3	1,194138	1,20879	-8,5E-06
19841006	1	20	24	1	0	0	0	1,191696	4	1,197801	1,207569	-8,4E-06
19841006	1	15	26	1	7	0	7	1,177044	7	1,18437	1,20879	-8,5E-06
19841006	1	11	15	1	0	0	0	1,178265	6	1,194138	1,20879	-8,7E-06
19841006	1	22	12	1	4	0	3	1,186812	5	1,190475	1,211232	-8,7E-06
19841006	1	23	14	1	0	0	0	1,197801	2	1,199022	1,20879	-8,4E-06
19841006	1	12	25	1	0	0	0	1,188033	4	1,210011	1,20879	-8,4E-06
19841006	1	20	12	1	0	0	0	1,186812	5	1,206348	1,212453	-7,9E-06
19841006	1	21	14	1	2	0	2	1,174602	7	1,175823	1,211232	-8,2E-06
19841006	1	7	11	1	1	0	1	1,17216	8	1,199022	1,20879	-8,5E-06
19841006	1	13	14	1	0	0	0	1,207569	0	1,189254	1,210011	-8,4E-06
19841006	1	19	11	1	0	0	0	1,175823	7	1,185591	1,210011	-8,2E-06
19841006	1	19	24	1	0	0	0	1,180707	6	1,186812	1,211232	-8E-06
19841006	1	11	14	1	0	0	0	1,192917	3	1,192917	1,207569	-8,4E-06
19841006	1	14	25	1	0	0	0	1,183149	5	1,200243	1,211232	-8,6E-06

FIGURE A.6: Snapshot of the Dataset used in the thesis

Bibliography

[1] Ramy Baly and Hazem Hajj. "Wafer classification using support vector machines". In: *IEEE Transactions on Semiconductor Manufacturing* 25.3 (2012), pp. 373–383.

[2] Bscan. *Kolmogorov–Smirnov test Wikipedia, The Free Encyclopedia*. 2013. URL: https://en.wikipedia.org/wiki/KolmogorovâĂŞSmirnov_test.

[3] Yaw-Jen Chang et al. "Virtual metrology technique for semiconductor manufacturing". In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. IEEE. 2006, pp. 5289–5293.

[4] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[5] Chao Chen, Andy Liaw, Leo Breiman, et al. "Using random forest to learn imbalanced data". In: *University of California, Berkeley* 110.1-12 (2004), p. 24.

[6] Hongge Chen. "Novel Machine Learning Approaches for Modeling Variations in Semiconductor Manufacturing". Master's Thesis. Tsinghua University, 2015.

[7] Fan-Tien Cheng et al. "Evaluating reliance level of a virtual metrology system". In: *IEEE Transactions on Semiconductor Manufacturing* 21.1 (2008), pp. 92–103.

[8] Chen-Fu Chien, Wen-Chih Wang, and Jen-Chieh Cheng. "Data mining for yield enhancement in semiconductor manufacturing and an empirical study". In: *Expert Systems with Applications* 33.1 (2007), pp. 192–198.

[9] François Chollet. *Deep Learning with Python*. 2017.

[10] Matteo Coppetta. *AURIX TC3xx eFlash TestWare, restricted*. 2019. URL: https://envm.sin.infineon.com/envm/FTOS/g3/125761-td.html.

[11] *Dealing with imbalanced data: undersampling, oversampling and proper cross-validation*. URL: https://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation.

[12] Manuel Fernández-Delgado et al. "Do we need hundreds of classifiers to solve real world classification problems?" In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3133–3181.

[13] Vaishali Ganganwar. "An overview of classification algorithms for imbalanced datasets". In: *International Journal of Emerging Technology and Advanced Engineering* 2.4 (2012), pp. 42–47.

[14] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[15] Haibo He and Edwardo A Garcia. "Learning from imbalanced data". In: *IEEE Transactions on knowledge and data engineering* 21.9 (2009), pp. 1263–1284.

[16] Q Peter He and Jin Wang. "Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes". In: *IEEE transactions on semiconductor manufacturing* 20.4 (2007), pp. 345–354.

[17] Qinghua Peter He and Jin Wang. "Large-scale semiconductor process fault detection using a fast pattern recognition-based method". In: *IEEE Transactions on Semiconductor Manufacturing* 23.2 (2010), pp. 194–200.

[18] Seokho Kang et al. "Using wafer map features to better predict die-level failures in final test". In: *IEEE Transactions on Semiconductor Manufacturing* 28.3 (2015), pp. 431–437.

[19] Aftab A Khan, James R Moyne, and Dawn M Tilbury. "An approach for factory-wide control utilizing virtual metrology". In: *IEEE Transactions on semiconductor Manufacturing* 20.4 (2007), pp. 364–375.

[20] Aftab A Khan, James R Moyne, and Dawn M Tilbury. "Virtual metrology and feedback control for semiconductor manufacturing processes using recursive partial least squares". In: *Journal of Process Control* 18.10 (2008), pp. 961–974.

[21] "Kolmogorov–Smirnov Test". In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 283–287. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_214. URL: https://doi.org/10.1007/978-0-387-32833-1_214.

[22] Sergios Theodoridis Konstantinos Koutroumbas. *Pattern Recognition*. 2008.

[23] *Machine Learning: Measuring Performance: AUC (AUROC)*. URL: https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/.

[24] Manzini. "Optimization of CPU-based Software In-Self Test (SIST) for automotive eFLASH modules applying machine learning techniques on bitmap data for suspect fails identification". Master's Thesis. Universita di Roma, 2018.

[25] Sarang Narkhede. *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[26] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-Validation". In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. URL: https://doi.org/10.1007/978-0-387-39940-9_565.

[27] Yvan Saeys, Thomas Abeel, and Yves Van de Peer. "Robust Feature Selection Using Ensemble Feature Selection Techniques". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 313–325. ISBN: 978-3-540-87481-2.

[28] Tomás Sarmiento, Sang J Hong, and Gary S May. "Fault detection in reactive ion etching systems using one-class support vector machines". In: *IEEE/SEMI Conference and Workshop on Advanced Semiconductor Manufacturing 2005.* IEEE. 2005, pp. 139–142.

[29] *Scikit-learn, Decision Trees*. URL: https://scikit-learn.org/stable/modules/tree.html.

[30] David W Scott. "On optimal and data-based histograms". In: *Biometrika* 66.3 (1979), pp. 605–610.

[31] *seaborn- Pyhton library*. URL: https://seaborn.pydata.org/generated/seaborn.kdeplot.html.

[32] Ben Soltane. "Design and development of software application to monitor and analyse the production yield of automotive micro-controllers". Master's Thesis. Higher School Of Communications, Tunis, 2019.

[33] Yanmin Sun, Andrew KC Wong, and Mohamed S Kamel. "Classification of imbalanced data: A review". In: *International Journal of Pattern Recognition and Artificial Intelligence* 23.04 (2009), pp. 687–719.

[34] Gian Antonio Susto, Alessandro Beghi, and Cristina De Luca. "A predictive maintenance system for silicon epitaxial deposition". In: *2011 IEEE International Conference on Automation Science and Engineering*. IEEE. 2011, pp. 262–267.

[35] Gian Antonio Susto, Alessandro Beghi, and Cristina De Luca. "A virtual metrology system for predicting cvd thickness with equipment variables and qualitative clustering". In: *ETFA2011*. IEEE. 2011, pp. 1–4.

[36] Gian Antonio Susto et al. "A predictive maintenance system for integral type faults based on support vector machines: An application to ion implantation". In: *2013 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2013, pp. 195–200.

[37] Gian Antonio Susto et al. "Automatic control and machine learning for semiconductor manufacturing: Review and challenges". In: *Proceedings of the 10th European Workshop on Advanced Control and Diagnosis (ACD 2012)*. 2012.

[38] Infineon Technologies. *About Infineon*. 2018. URL: https://www.infineon.com/cms/en/about-infineon/company/.

[39] Infineon Technologies. *AURIX System Architecture*. 2019. URL: https://www.infineon.com/dgdl/Infineon-AURIX_System_Architecture-TR-v01_00-EN.pdf?fileId=5546d46269bda8df0169ca92d6362599.

[40] Infineon Technologies. *eSquare User Wiki Home, Restricted*. URL: https://confluencewikiprod.intra.infineon.com/display/eSquareCU/eSquare+User+Wiki+Home.

[41] *The Kolmogorov-Smirnov TesT*. URL: https://daithiocrualaoich.github.io/kolmogorov_smirnov/.

[42] The Kernel Trip. *Random forest vs SVM*. 2018. URL: https://www.thekerneltrip.com/statistics/random-forest-vs-svm/.

[43] *Understand Data Normalization*. URL: https://towardsdatascience.com/understand-data-normalization-in-machine-learning-8ff3062101f0.

[44] K Keerthi Vasan and B Surendiran. "Dimensionality reduction using principal component analysis for network intrusion detection". In: *Perspectives in Science* 8 (2016), pp. 510–512.

[45] Sze-jung Wu et al. "A neural network integrated decision support system for condition-based optimal predictive maintenance policy". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 37.2 (2007), pp. 226–236.

[46] Jonathan Chang Yung-Cheng and Fan-Tien Cheng. "Application development of virtual metrology in semiconductor industry". In: *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.* IEEE. 2005, 6–pp.