Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Matemáticas y Física

Maestría en Ciencia de Datos

Machine Learning Techniques For Electrical Validation Enhancement Processes

---

TESIS que para obtener el **GRADO** de
MAESTRO EN CIENCIA DE DATOS

Presenta: **CÉSAR AUGUSTO SÁNCHEZ MARTÍNEZ**

Asesora: **ROCÍO CARRASCO NAVARRO**

Tlaquepaque, Jalisco. Diciembre de 2020.

Abstract

Post-Silicon system margin validation consumes a significant amount of time and resources. To overcome this, a reduced validation plan for derivative products has previously been used. However, a certain amount of validation is still needed to avoid escapes, which is prone to subjective bias by the validation engineer comparing a reduced set of derivative validation data against the base product data. Machine Learning techniques allow, to perform automatic decisions and predictions based on already available historical data. In this work, we present an efficient methodology implemented with Machine Learning to make an automatic risk assessment decision and eye margin estimation measurements for derivative products, considering a large set of parameters obtained from the base product. The proposed methodology yields a high performance on the risk assessment decision and the estimation by regression, which translates into a significant reduction in time, effort, and resources.

Resumen

La validación de márgenes del sistema en pós-silicio consume un tiempo significativo de tiempo y recursos. Para superar esto, se ha utilizado previamente un plan de validación reducido para productos derivativos. Sin embargo, cierta proporción de validación aun es necesaria para evitar algún escape, aún así puede ser propenso a una decisión subjetiva dada por el ingeniero de validación cuando compare resultados de un conjunto reducido de datos de validación de un derivativo contra los datos del producto base. Técnicas de Aprendizaje Automático (Machine Learning) permiten ejecutar decisiones automáticas y predicciones basados en datos históricos existentes. En este trabajo, se presenta una eficiente metodología implementada con Aprendizaje Automático, para realizar una valoración de riesgos para una decisión automática y para una estimación de mediciones de márgenes de ojos para productos derivativos, considerando un conjunto grande de parámetros obtenidos del producto base. La metodología propuesta produce un alto rendimiento en la valoración de riesgos para tomar decisiones y en la estimación por regresión, que se traduce en una reducción significativa de tiempo, esfuerzo y recursos.

Dedications

I want to thank the different aspects and circumstances that happened in my life that drove me to start studying this master's and areas of data analytics, which now has become one of my passions in life.

To my family: my Father Cesar, my Mother Guadalupe, my brother Ivan, that you have always supported me in the decisions I have taken, also I want to include my aunt Carmen, her daughter Mela, her son Paco and her son in law Fernando. All of you observed me studying and doing homework, I thank all of you, the comprehension when I missed some family meetings and during some others that I went to other room to work on master, it was very valuable you were there asking how things were going and showing your support to me. I also thank my little niece Ana Paula that multiple times ask me about my school and told me to effort myself studying.

All my friends, that is a big list, who were there pending of my life and my studies, listening to me when I needed, sharing with me some time for a break and relax and especially for giving me best wishes and to keep big effort until I succeed. From my heart to all of you, thanks!

I thank Rocio as my teacher, thesis advisor, and mentor on this journey of learning data science and completing the thesis work, I will always appreciate your knowledge sharing, contribution, and lessons given to me. I extend this to the rest of my teachers: Fernando, Santiago, Irving, Juan Diego, Riemann, Pablo, Alonso, and Esteban, each of you contributed to my learning, improving, and growing my skills and knowledge, I take all your lessons as teachers and as persons to keep this path of learning and improving. I also thank Juan Carlos, as coordinator of the master's program, for inspiring me to take the master's degree, his support in asking for feedback, and knowing how we were feeling to keep a good environment for students.

My schoolmates, Dani, Jessi, Vicky, Omar, Arjona, Mario, Pablo, Emanuel, and Claudia that we all share this process, studying, doing homework, doing projects, collaborating, giving suggestions to each other or sharing knowledge, all many hours we had together doing this journey.

Thanks to my co-workers that share ideas, knowledge, expertise, or support to complete this work. I extent especially for my friends and co-workers Andres and Paulo, for sharing their experience and knowledge that support this work that also derivates to a paper published in the IEEE.

Dari thank you for your support on the competition of the master, your advice, listening to me, support, and also your contribution with your expertise to define and achieve some of the projects.

Dedications to the Institute

I want to thank the Instituto Tecnologico y de Estudios Superiores de Occidente (ITESO) to grant a scholarship for being the first generation on the Data Science Master's degree.

Preface

List of Figures

List of Tables

Glossary

A

ANN
Artificial Neural Network, 19, 38, 39, 41, 62, 63, 64, 65

B

BER
Bit Error Ratio, 13, 15, 16
BP
Base Product, 12, 17, 34, 57, 58, 60, 63, 65

D

DP
Derivative Product, 12, 17, 18, 19, 69, 72, 74
DUT
Device Under Test, 14

E

e.g.
exempli gratia, meaning for example, 11, 12, 17, 30, 53
EH
Eye Height, 14, 15, 17, 18, 41, 57
EQ
Equalizer, 14, 15, 34, 57
Ethernet
Computer Networking Technology, 12
EV
Electrical Validation, 12, 14, 15, 17, 18, 19, 35, 72
EW
Eye Width, 14, 15, 17, 18, 41, 57

H

HMME
How Much Margin is Enough, 16, 17, 58, 60, 63, 65, 70
HSIO
HighSpeed Serial Input/Output, 12, 13, 34, 57

J

jitter, 76

K

k-fold
cross-validation resampling for evaluating machine learning models with limited data sample, k is for the groups to split the data, 35, 36, 37, 39, 58

L

Logit
Logistic Regression, 19, 35, 36, 40

M

ML
Machine Learning, 11, 12, 17, 18, 34, 35, 36, 37, 57, 58, 59, 69, 70, 71, 72
MSE
Mean Squared Error, 58, 65, 67

O

OLS
Ordinary Linear Regression, 41, 58, 59, 62, 65, 66, 67
optimization, 76

P

PCH
Port Control Hub, 12, 34, 57
PCIe
Peripheral Component Interconnect Express, 12
post-silicon, 76
A prototype of silicon manufactured, typically used for testing before product release, 11, 12, 34, 57, 76
PRQ
Product Release Qualification, 11, 12, 17, 69, 72

R

R^2
coefficient of determination, 58, 67
R_x
Receiver, 12, 13, 14, 15, 16, 17, 34, 58

S

SATA
Serial Advanced Technology Attachment, 12, 34, 57, 76
SMV, 76
System Margin Validation, 12, 13, 14, 15, 16, 17, 36, 59, 69, 70
SVC

Support Vector Classification, 19, 36, 37
SVR
Support Vector Regression, 41, 59, 60, 61, 62, 67

T

T_x
Transmitter, 12, 14

U

UFS
Universal Flash Storage, 12
UPM
Units Per Million, 16, 17, 18, 19, 34, 57, 70
USB
Universal Serial BUS, 12, 14, 76

V

VDC
Volume Data Collection, 16, 17, 34, 57, 71, 72

1 Introduction

1.1 State of art

In the industry of CPU, microchips, microprocessor, and integrated circuits in general, the validation process is a very important aspect since it supports and gives the guarantee that the product that customers or the end-users are going to receive accomplish the standards and quality, it can be compared like insurance that the company uses to guarantee that the functionality of the products after been manufactured is approximated to a real-world scenario where the silicon is going to be implemented, *e.g.* laptops, tablets, servers, desktop computers, etc. To mention some, knowing that the validation of units should cover a series of tests that can be destructive, extreme, or stressful conditions, where the silicon should work and provide a certain level of performance while working on the limits of the specifications or even above them. Those tests should be sufficient to avoid any scape and minimize most the possibilities that the customers find an issue or a bug, those tests can be extensive and for long periods, that depends on the complexity of the technology, that is continuously scaling.

Silicon validation can be generalized as two main areas pre-silicon and post-silicon, where pre-silicon validation refers to the process to test virtually using a simulation of the silicon circuitry and with verification tools on emulation environment. However, this does not cover the functionality on the system-level, here is where post-silicon validation is performed when the first silicon is ready, it is tested on the system environment looking for bugs missed on the pre-silicon validation [1]. The objective of post-silicon validation is to ensure the proper functionality of the design tested pre-silicon emulation, on a system that approximates to the real environment where the silicon would be a product when it is released in the market. The post-silicon validation of a microprocessor is covered by various disciplines, that qualifies the product over multiple stress and operation conditions, the end deliverable of post-silicon validation is the Product Release Qualification (PRQ), that is a decision made based on the results to make a risk assessment of how many systems may fail the specifications and standards of quality [1].

Post-silicon validation aims to understand with physical units the behavior of a design by verifying the proper operation, identify bugs, and for large scale manufacturing estimation of the robustness across the process variation [2], to avoid failures and achieve the quality standards on most of the silicon units fabricated. On the other hand, ML algorithms build statistical models based on datasets to make predictions [2]; these models can take the large volume of data generated during validation to predict the post-silicon behavior.

With the above knowledge of the process of silicon validation, it can be upgraded smartly using Data Science, to apply “smart validation” with tools that can optimize the time required for some tests with the usage of all the data generated during the validation. It has been observed in areas like manufacture or simulation some examples: Using training patterns to learn the behavior of silicon measurements for post-silicon and with simulations on pre-silicon [3] proposed ML algorithm to predict the behavior of a clock signal labeling Pass or Fail based on m-dimensional sample points as inputs. A regression model with electrical measurements is used in [4] build training datasets to predict the stability of fabrication silicon wafers. ML algorithms have also been used to improve efficiency on silicon fabrication by monitoring the degradation and, when necessary, provide maintenance on manufacturing tools and achieve a reduction in operating costs [5].

Currently, the complexity that the silicon industry brings with the integration of multiple functionalities on the same die, sophisticated packaging, and dense integration [6] present difficulties to observe the internal functionality of the circuits with limited points to monitor some physical phenomenon on internal circuits and the output signals [6]. ML algorithms can help to model the behavior of circuits [7] to predict performance using the historical data collected [8], in this case of study for the Physical Layer (PHY) used in High-Speed Serial Input/Output circuitry. The proposal is to build a dataset, with Receivers (R_x) calibration settings, R_x eye margin measurements, also a labeled variable for Pass/Fail criteria based on tuning knob settings. The dataset would therefore include measurements during recipe adjustment as well as PRQ-worthy volume data [2] from the "first design", now then called Base Product (BP), to train ML models that posteriorly, with the collection of data on from the re-usage of PHY on Derivative Products (DP), will predict and compare if BP and DP have a comparable behavior, avoiding the tedious process to evaluate a DP with a significant reduction of time on the risk assessment.

1.2 Problem Statement

One of the disciplines for post-silicon validation is the: Electrical Validation (EV) [9][10][11] that focuses on validating the electrical parameters and behaviors of the circuitry, those can include: Input/Output (I/O) links (*e.g.* Memory channels, Serial interfaces, parallel interfaces, etc.), Phased-Locked Loop (PLL), power networks, clock signals, analog/mixed-signals, R_x, Transmitters (T_x) to mention some of them. The labor of EV engineer testing in a real system is to determine and decide the optimal system parameters, and correct electrical behavior of I/O links on a silicon unit (*e.g.* Central Processor Unit (CPU), Platform Control Hub (PCH), System On a Chip (SOC), or any other silicon unit with an embedded I/O), to perform a risk assessment decision based on evidence that comes from available parameters and measurements.

This work mainly focuses on an EV testing methodology called System Margin Validation (SMV) [11][12] for the physical layer (PHY) contained in Highspeed Serial I/O (HSIO) which can cover interfaces links like Peripheral Component Interconnect Express (PCIe) [13], Serial Advanced Technology Attachment (SATA) [14], Universal Serial BUS (USB) [15], Ethernet [16] and Universal Flash Storage UFS [17] with Mobile Industry Processor Interface (MIPI) [18], those are integrated on-die package and EV analyses the performance and marginality [11], that is the electrical interaction of the interfaces with the platform, connectors, and devices, similar to a final product, when the system is subject to multiple stress using commercial devices, applications and standard signal patterns (defined by specifications) and based on multiple measurements on the circuitry statistically predict the behavior in a real system scenario over different conditions and test cases, as listed below:

- Voltage supply rail variations
- Temperature ranges (extreme conditions and nominal)
- Multiple microprocessors (to add manufacture process variation)
- Channels with different insertion loss (dB losses)
- Different topologies and board connectors
- Board variations

- Different commercial devices

The SMV risk assessment for HSIO interfaces has two of the main circuits tested, those are the T_x and the R_x, being this last where most of the measurements are taken to make the risk assessment, based on the observation of the eye diagram (represented in Figure 1.1) measurements, due to the frequencies handled by the HSIO links, where a signal is sent by R_x pass through the "propagation channel" and captured by the R_x, all the way that the signal pass through the transmission line is subject to different physical phenomena [19], *e.g.* Jitter, Signal Reflections, Crosstalk, Inter-symbol Interference (ISI) and Losses. Figure 2 shows a representation of how the signal could be affected by the different phenomena, Figure based on [19].

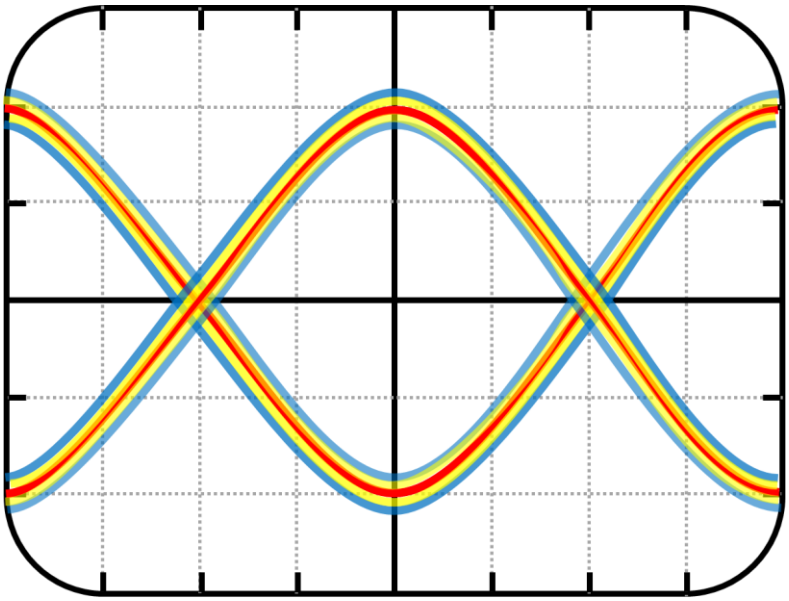

Figure 1.1. Eye Diagram representation for a clean differential signal on HSIO.

The effects mentioned above influence the performance of the HSIO link, they can be measured by the Bit Error Ratio (BER) [20], the fewer errors measured the better link performance. BER is typically used to characterize R_x to determine compliance with industry specifications and standards.

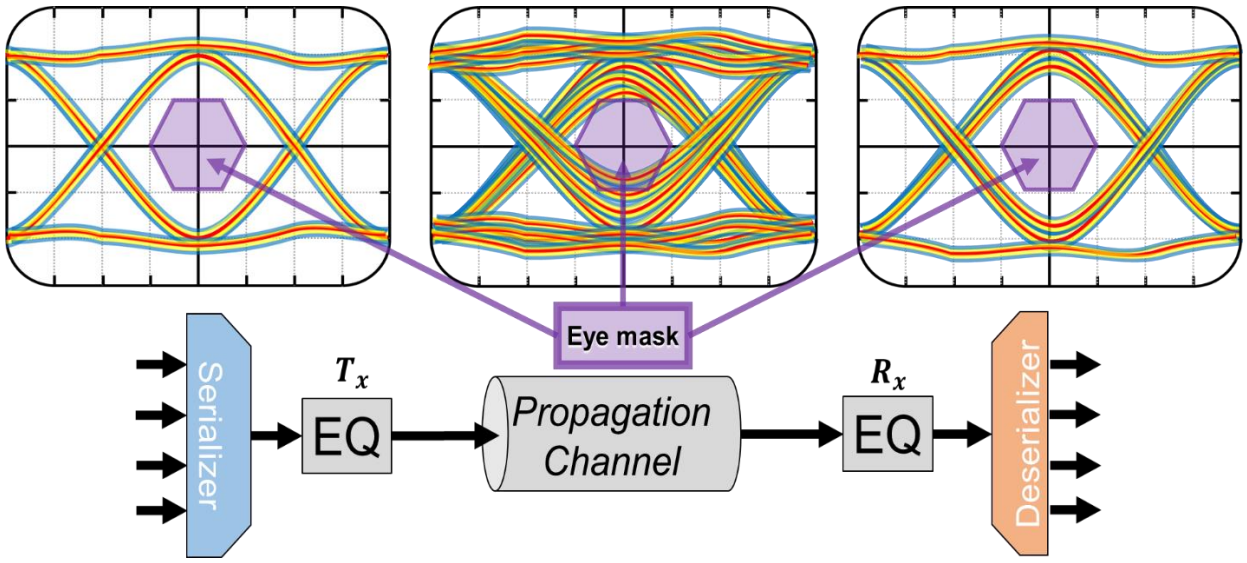

Figure 1.2. Effects on the eye signal after is transmitted: T_x Equalizer (EQ) to propagation channel and then R_x EQ clean the signal.

SMV is one of the most important methods on EV because it measures the R_x eye margins in a real system, that typically constitutes of Device Under Test (DUT), test Platform, connectors (or topology), cable (if topology requires), and the endpoint (i.e. add-in cards, Solid State Drive, USB pen-drive, etc.). Figure 3 shows a visual diagram of the set up to perform SMV tests.

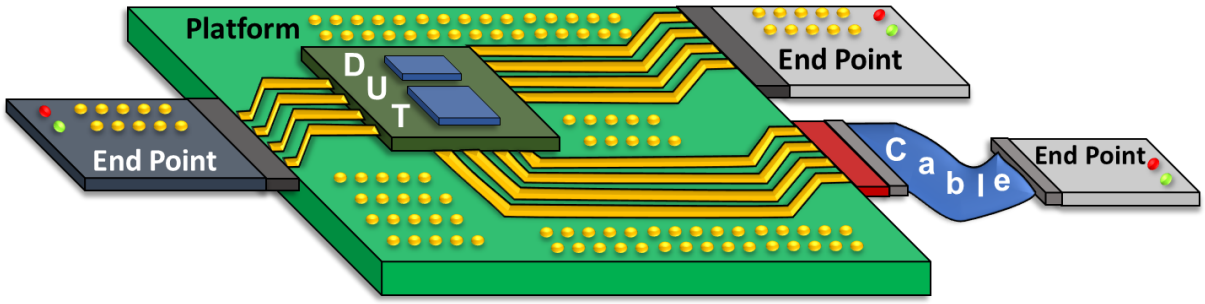

Figure 1.3. Diagram representing a real system set up to test the DUT (silicon unit).

The R_x Eye Margin is considered as the region where the eye diagram is still valid for the sampling circuitry to determine if the capture of the serial signal is 0 or 1, the peak points in the eye margining measurement are the Eye Height (EH) that is the maximum voltage amplitude and the Eye Width (EW) that is the maximum amplitude on timing.

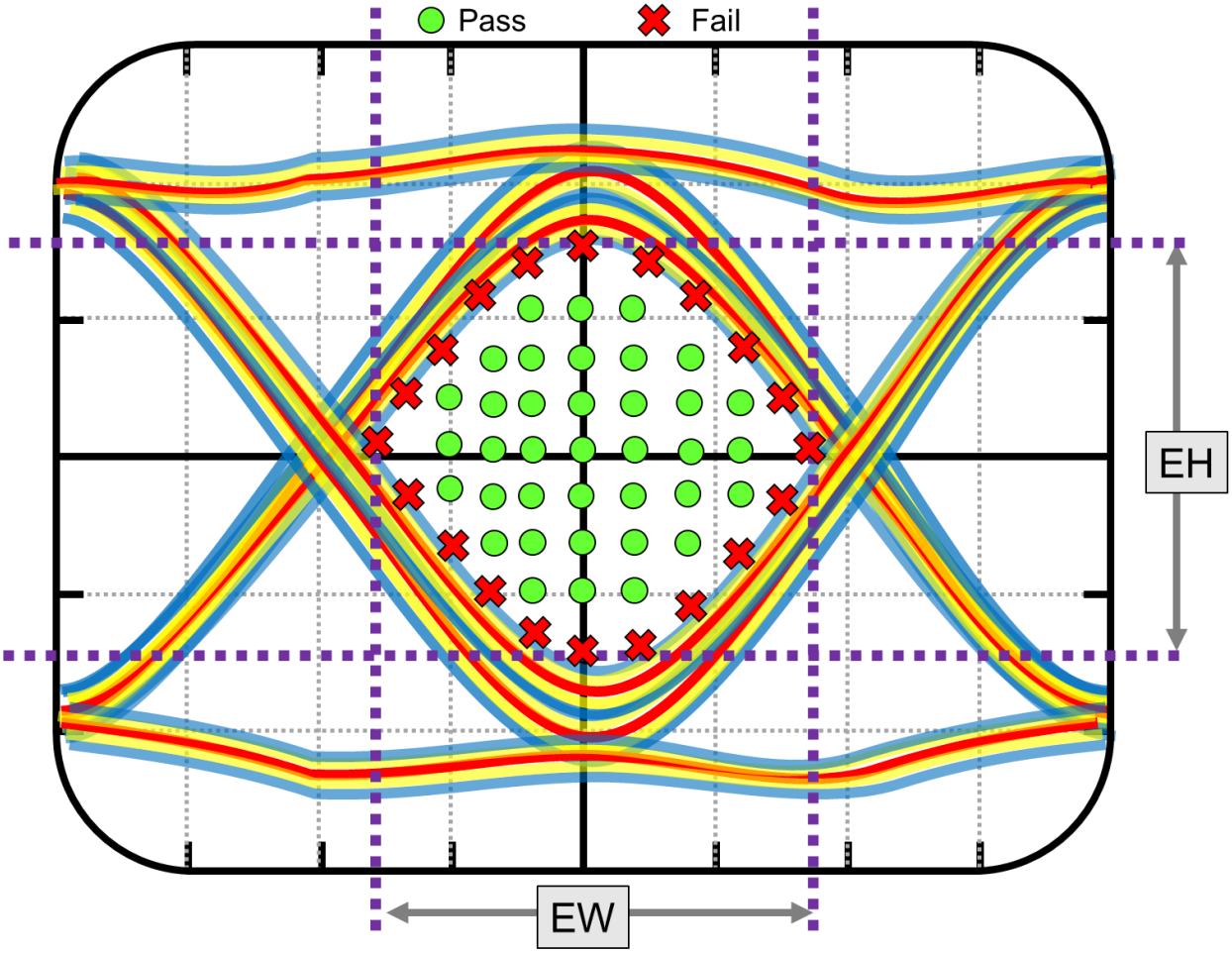

Figure 1.4. Visual representation of how the margin is evaluated in SMV, determining the EH and the EW.

The R_x can clean the signal from noise applying firstly analog and then digital treatments with the EQ block (see Figure 1.2), the equalizations can be grouped in:

- Bandpass filters
- Gain amplifiers
- Samplers
- Clock data recovery

The treatment circuits or EQ reduce the effects of the propagation channel with tunable elements or "calibrations", which are divided into adaptative and static, the tuning process for EV is one of the first stages of the SMV activities, the equalization circuits are adjusted with the usage of "knobs" for each tuning element, the treatment on the signal should guarantee compliance with the specification of the BER [20], once the tuning settings are adjusted and approved by EV engineers, these values are programmed to boot system of the platform, those settings are called "the recipe".

Once the recipe is obtained and chosen for the rest of the validation process, the next stage of SMV is to test the robustness of the R_x across multiple silicon units, due to the nature of the manufacturing process, every silicon unit will face different effects for the: silicon doping, thickness or thinness of the metals, skew, etc. This effect called “part to part variation” and also to other external factors like different platforms, topologies, cables (if applicable), and endpoint devices, and the silicon should work over extreme conditions or corners cases of voltage and temperature, that are defined based on ranges that the design specification of the circuitry indicates. All those variables could make that the R_x has different behavior and SMV should evaluate the robustness of the recipe chosen across multiple silicon units, this stage is known as Volume Data Collection (VDC), Then to determine the How Much Margin is Enough (HMME) some experiments are performed like: Silicon aging, Temperature, Humidity, Voltage, BER adjustment, endpoint variation, to mention some [11][21], then the R_x eye margins gathered in a bunch of DUTs with the VDC would obtain statistically the eye margin distribution [11][21], the VDC is used as a sample to projected over one million units, then with the HMME data, the sample it is forecasted how many units will fall into the allowed standard deviation from eye margin distribution, this analysis is called Units Per Million (UPM), which looks to approximate to zero units under standard deviation considered as fail region, however, exist a maximum UPM allowed, and with that number is the risk assessment decision is driven for SMV. The flow diagram of SMV is shown in Figure 1.5 [2].

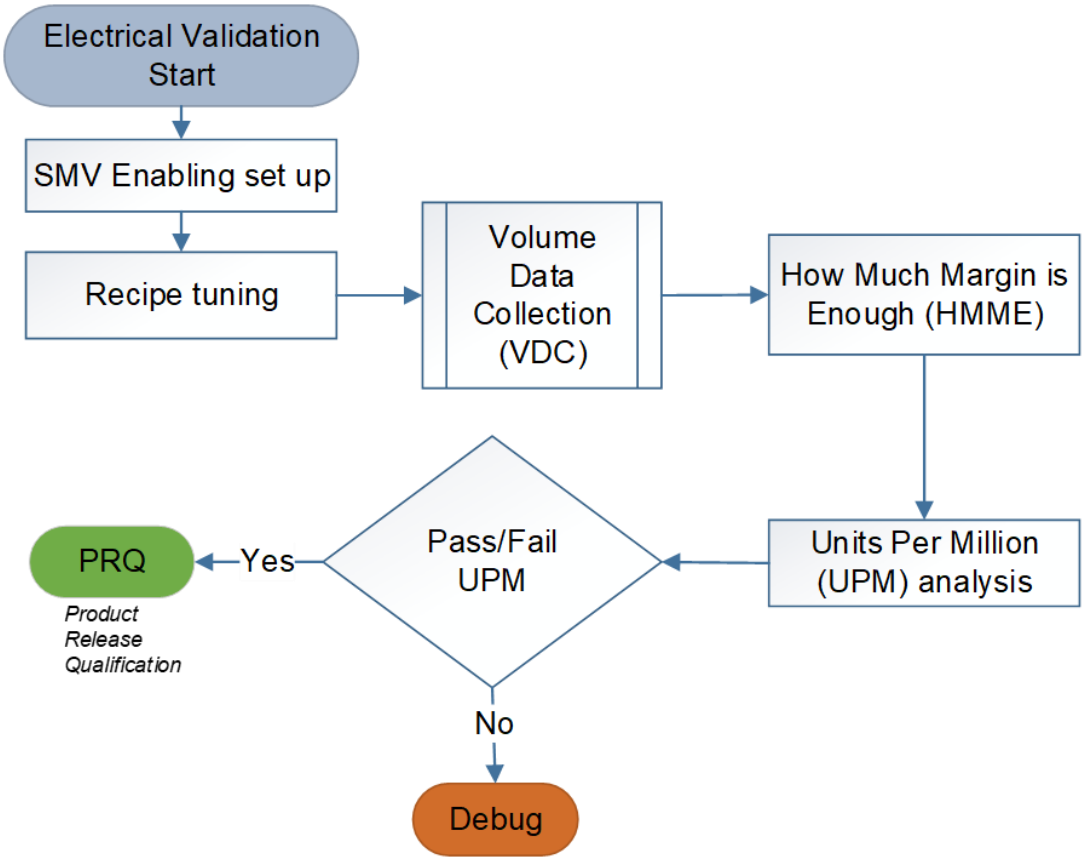

Figure 1.5. SMV validation flow for risk assessment decision on HSIO.

The SMV risk assessment is performed on every product, and it can be divided into two groups:

- Silicon Base Products (also known as BP): that comprises a new architecture, or the first instance of a previous design on a new technology node.
- Silicon Derivative Products (also known as DP): the re-use of PHY from a previous design on the same process node.

When the silicon BP validation is completed and achieved the PRQ decision, the VDC from the BP is used as the baseline for future products that are derived from it, i.e. silicon DP. The final process consists of the SMV risk assessment, where an experienced electrical validation engineer is in charge of emitting a technically evidenced judgment by comparing the BP baseline statistical margin distribution data against the DP data and assess the health of the DP; if the margin distributions between BP vs DP are comparable, then the health of the DP should be the same or similar to the BP. This is the standard methodology.

Another practical solution is the $Nx\ N$ [12], where N is a short amount of silicon units, typically $N \geq 5$ but it is left under consideration of the EV engineer, and this consists of comparing silicon BP vs DP, by collecting the EW and EH margins of the DP on the N silicon units, over various test platforms, considering different topologies, and over different voltage and temperature corner cases. If the margins obtained are above the HMME allowed in the BP, then the DP margin distribution is considered to be equal or better than the BP margin distribution; this is considered to guarantee that the DP is healthy.

Now let's mention some of the disadvantages associated with the previous solutions described.

- In the standard methodology: the analysis requires the VDC equivalent of a BP, and then the assessment of the DP is provided by the experienced validation engineer to decide if the BP vs DP results are comparable. This activity is tedious and more importantly, very time-consuming, *e.g.* 8 to 12 weeks of work.
- The $Nx\ N$ methodology requires less time than the standard methodology, *e.g.* 6 weeks, but it still requires the same amount of resources: engineers, lab space, equipment, platforms, etc. Furthermore, the analysis needs to be more thorough since EV engineers are looking to draw conclusions from a very small number of silicon samples and make a comparison against the BP full volume distribution. In this approach, subjective bias can have a significant effect on the final result.

1.3 Objectives

1.3.1 General Objectives

With the above considerations, the risk assessment performed by the EV engineers tends to be time-consuming, tedious, and prone to subjective bias. This work proposes methodologies based on Machine Learning (ML) to make an SMV risk assessment by analyzing data that include eye margins form R_x from the tuning stage and the VDC stage (evaluated with the UPM analysis), both build a dataset

that includes: the recipe (with static and adaptive calibrations), and the EW and EH measurements which aid the ML system to perform the correct decision.

1.3.2 Particular Objectives

This work proposes to use ML algorithms for two strategies:

- **Classification ML algorithm**:

  That would determine if new samples of a DP would pass or fail. This would include similar criteria that the EV engineers use for UPM analysis, which consist of a sanity check of the DP.

- **Regression ML algorithm**:

  This would generate synthetic Eye margins estimation based on real calibrations data which would reduce considerably the validation time since the capture of EH and EW across multiple silicon units is a very time-consuming activity and the read of the calibration values is very quick to perform.

1.4 Thesis Structure

1. Chapter one: contains the introduction to electrical validation and the problem statement that wants to be solved, also the state of the art in this area.
2. Chapter two: refers to the mathematical background for the ML learning models used across this work for classification proposes.
3. Chapter three: Classification Models to determine the Pass/Fail of Eye Margins.
4. Chapter four: refers to the mathematical background for the ML learning models used across this work for regression proposes.
5. Chapter five: Regression Models to determine an artificial Eye Margin estimation based on R_x calibration settings.
6. Chapter 6: Conclusions and future work.

2 Mathematical Preliminaries For Classification Models

The classification algorithms would determine if new samples of a DP would Pass or Fail. This would include similar criteria that the EV engineers use for UPM analysis, which consist of a sanity check of the DP.

2.1 Models used

For this part four machine learning algorithms were selected:

1. Logistic Regression (Logit).
2. Support Vector Machine for Classification (SVC).
3. Artificial Neural Networks (ANN).
4. Decision Tree with Random Forest methodology.

2.1.1 Logit

The Logistic Regression (Logit) was originally developed for survival analysis [22], *e.g.* a patient "survives" an illness, it consists in the transformation of the linear regression into a probability function, the case for one independent variable:

$$1 \rightarrow P(y = 1|x) = f(\beta_0 + \beta_1 x) \tag{2.1}$$

$$0 \rightarrow P(y = 0|x) = 1 - f(\beta_0 + \beta_1 x) \tag{2.2}$$

The relationship for classification models consists of the response of the $f(x)$, as observed in Figure 2.1. The sigmoid function that transitions asymptotically from zero to one, that never fall below zero or above one, due to the usage of sigmoid function to represent the interval where Logit can predict the expected value of $f(x)$ or $\hat{y}$. The term 'sigmoid' means S-shaped. This type of function is sometimes also called a 'squashing function' because it maps the whole real axis into a finite interval [23]. The logistic sigmoid plays an important role in many classification algorithms. It satisfies the following symmetry property: $-\infty < \beta_0 + \beta_1 x < +\infty$ and it would be: $f(-\infty) = 0$ and $f(+\infty) = 1$.

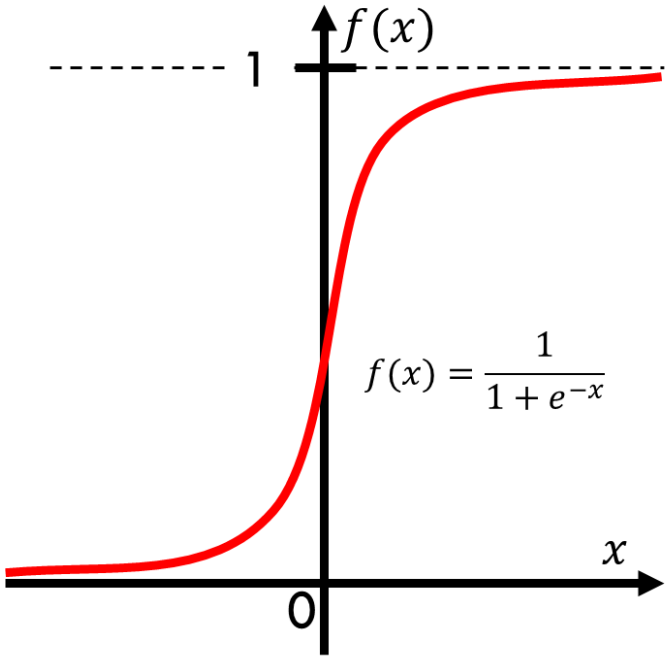

Figure 2.1. The sigmoid function that transitions asymptotically from zero to one.

Where: $f = \Delta(\beta_0 + \beta_1 x)$ is the change rate over the sigmoid function that would provide the probability of the classification to be: 1 or 0. The probability function for one observation is calculated by:

$$P(y = y_i|x) = [f(\beta_0 + \beta_1 x)]^{y_i}[1 - f(\beta_0 + \beta_1 x)]^{1-y_i} \quad (2.3)$$

If it is expanded for n quantity of observation and applying the maximum likelihood [23][24]:

$$L(\beta_0, \beta_1, x_i, y_i) = \prod_{i=1}^{n}[f(\beta_0 + \beta_1 x_i)]^{y_i}[1 - f(\beta_0 + \beta_1 x_i)]^{1-y_i} \quad (2.4)$$

Then to simplify it is transformed with a logarithm into a summation:

$$ln\big(L(\beta_0, \beta_1, x_i, y_i)\big) = \sum_{i=1}^{n} y_i ln[f(\beta_0 + \beta_1 x_i)] + (1 - y_i) ln[1 - f(\beta_0 + \beta_1 x_i)] \quad (2.5)$$

Then to maximize the function it would be required to gather the coefficients with partial derivatives:

$$\frac{\partial ln(L)}{\partial \beta_0}, \frac{\partial ln(L)}{\partial \beta_1} = 0 \quad (2.6)$$

The objective here is to find the β_0 and β_1 coefficients that would support the calculation of the probability $P(y = 1|x)$ or $P(y = 0|x)$ from (2.1) and (2.2) This can be expanded to the general probability function for multiple variables, with a vector x_i and coefficients β_i, where $i = 1,2, \dots, n$, that would be used to calculate the $P(y = 1|X_i)$ [22]:

$$P(y = 1|X_i) = \frac{e^{(\beta_0 + \sum_{i=1}^{n} \beta_i x_i)}}{1 + e^{(\beta_0 + \sum_{i=1}^{n} \beta_i x_i)}} \tag{2.7}$$

2.1.2 Support Vector Classification

The principle used in this classification algorithm is the usage of a separation hyperplane, in a $p - dimensional$ space, a hyperplane is a flat affine subspace, in an instance, a two-dimension hyperplane is a flat dimensional subspace (a line), in three dimensions is the hyperplane would be a plane [24], as observed in Figure 2.2, and with dimensions greater than three it is hard to visualize but the notion is that the subspace for the hyperplane would be $p - 1$, the hyperplane equation is given as a sum w_p of coefficients and each multiplied by the variable x_p, where w_0 or b is the intercept of the equation [24]:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_{p-1} x_{p-1} = w_0 + W^T X = W^T X + b \tag{2.8}$$

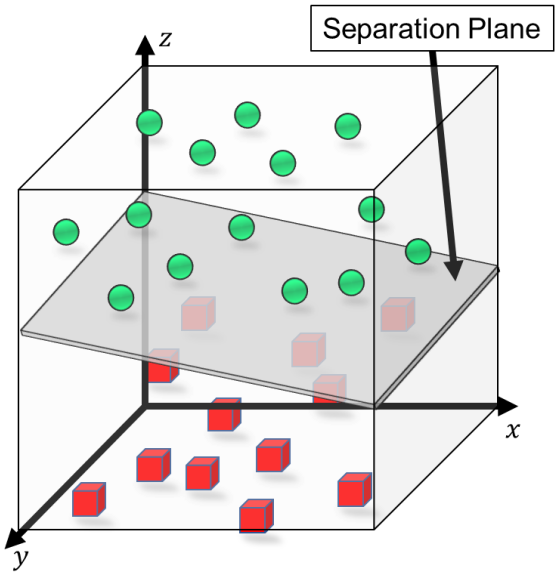

Figure 2.2. Representation of sub-space (plane) on a 3-dimensional space.

As observed in Figure 2.2, it would be required to add a separation between the different data points to start the description, let's use the linear model:

$$f(x) = W^T \varphi(x) + b \tag{2.9}$$

Where $\varphi(x)$ denotes a fixed featured space transformation and *b* is the bias parameter, assuming that exists one choice of W and *b* that satisfice the linear separation of the data set, as observed in *Figure 2.3*, the training dataset comprises N input vectors $x_1 + x_2 + \cdots + x_N$ with a corresponding target values $y_1 + y_2 + \cdots + y_N$ where $y_n \epsilon \{-1,1\}$ and new points of x are classified according to the sign of $f(x)$.

There may are many solutions that could separate the classes exactly, but that solution depends on the arbitrary initial values chosen for W and *b* as well as the order in which the points of the data set are presented, if exist many possible solutions there should be found the one that gives the smallest generalization error, the SVM approach that problem through the concept of "margin" denoted by γ, which shall be the smallest distance between the decision boundary and any of the data samples, as can be observed in Figure 2.3, it can be defined as the perpendicular distance between the decision boundary and the closest of the data points. For the SVM the boundary is chosen to be the one for which the margin is maximized [23][24].

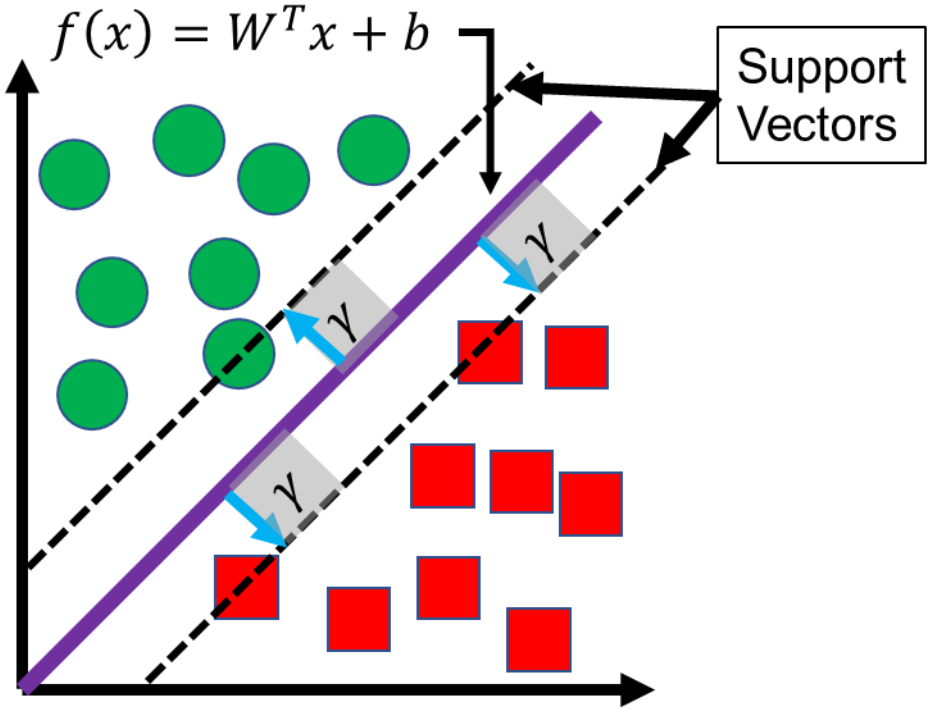

Figure 2.3. Representation of the hyperplane, separation margins, and support vectors.

To define the best separation hyperplane there should be minimized the probability of error relative to the learned on the model (2.10).

$$\gamma_i = \frac{W^T}{\|W\|} x_i, \gamma = \min \gamma_i \Rightarrow for\ i = 1,2, \dots, n \tag{2.10}$$

The optimal hyperplane is the one having the maximum margin γ and it is desired to optimize the parameters W and b to maximize the distance, so the maximum margin solution is found by solving:

$$\underset{W,b}{max}\ (\gamma) = \underset{W,b}{max} \left(\frac{1}{\|W\|} \left[\min_i W^T \varphi(x_i) + b \right] \right) \Rightarrow for\ i = 1,2, \dots, n \tag{2.11}$$

Where the factor $1/\|W\|$ is outside of the optimization over n because W does not depend on n, and it shall be converted into an equivalent problem that is easier to solve [23]. Scaling W and b by a factor and this would give freedom to set:

$$y_n(W^T \varphi(x_n) + b) = 1 \tag{2.12}$$

For the point, that is closest to the surface, and also all data points will satisfy the constraints:

$$y_i(W^T \varphi(x_i) + b) \geq 1 \Rightarrow for\ i = 1,2, \dots, n \tag{2.13}$$

This is known as the canonical representation of the decision hyperplane, by definition, there will always be at least one active constraint because there will be always at least one closest point, also observed in *Figure 2.3*, and once the margin is maximized there will be at least two active constrains, then the optimization problem would require that $\|W\|^{-1}$ is maximized, that is equivalent to minimize $\|W\|^2$, so the optimization problem now converts into [23][24]:

$$\underset{W,b}{min} \left[\frac{1}{2} \|W\|^2 \right] \tag{2.14}$$

Subject to the constraint of (2.13) and the factor $1/2$ is added for convenience, to solve the constrained optimization problem, it would be used Lagrange multipliers $\lambda_i \geq 0$ with one multiplier for each λ_i constraints, it would drive to the Lagrangian function:

$$L(W, b, \lambda) = \frac{1}{2} \|W\|^2 - \sum_{i=1}^{n} \lambda_i \{ y_i(W^T \varphi(x_i) + b) - 1 \} \Rightarrow for\ i = (\lambda_1, \lambda_2, \dots, \lambda_i)^T \tag{2.15}$$

The minus sign in front of the Lagrange multiplier term is because it is required to minimize respect to W and b and maximize respect to λ, then it would be necessary to set partial derivatives of $L(W, b, \lambda)$ respect $W, b,$ and λ equal to zero, then the following conditions are obtained:

$$\frac{\partial L}{\partial W} = 0 \Rightarrow \sum_{i=1}^{n} \lambda_i y_i \varphi(x_i) = W \tag{2.16}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{2.17}$$

Eliminating W and b from $L(W, b, \lambda)$ using those conditions then gives the dual representation of the maximum margin problem where it has been maximized [23][24]:

$$\tilde{L}(\lambda) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \tag{2.18}$$

For λ subject to constraints:

$$\lambda_i \geq 0, \qquad for\ i = 1,2, \dots, n, \qquad \sum_{i=1}^{n} \lambda_i y_i = 0 \tag{2.19}$$

The kernel function K and it is defined as $K(x, x') = \varphi(x)^T \varphi(x')$, this kernel formulation makes the role of the constraint that the function $K(x, x')$ be positive definite, because this ensures that the Lagrangian function $\tilde{L}(\lambda)$ is bounded below, and rising to a defined optimization problem.

To classify new data with the trained model, the sign of $y(x)$ is defined by (2.9). This can be expressed in terms of $\{\lambda_i\}$ and the kernel function by substituting W from (2.16) to give:

$$y(x) = \sum_{i=1}^{n} \lambda_i y_i K(x, x_i) + b \tag{2.20}$$

A data point which $\lambda_i = 0$ will not appear on the sum of (2.20) and hence plays no role in making predictions for new data points. The remaining data points are called support vectors because they satisfy $y_n y(x_n) = 1$, they correspond to the points that lie on the maximum margin hyperplanes in feature space, as can be illustrated in *Figure 2.3*. This property is central to the practical applicability of support vector machines. Once the model is trained, a significant proportion of the data points can be discarded and only the support vectors are retained.

Having the value for λ, then can be determined the value of the threshold parameter b by any given vector x_n that satisfies $y_n y(x_n) = 1$ using (2.20), so it would be:

$$y_n\left(\sum_{m\epsilon\mathcal{S}}\lambda_m y_m K(x_n,x_m)+b\right)=1 \tag{2.21}$$

Where $\mathcal{S}$ denotes the indices of the support vectors. This (2.21) can be solved for b using an arbitrary chosen support vector x_n, a numerically more stable solution is obtained by first multiplying thought by y_n, with the use of $y_n^2=1$, and then averaging these equations, the overall support vectors and solving for b to give:

$$b=\frac{1}{N_S}\sum_{m\epsilon\mathcal{S}}\left(y_n-\sum_{m\epsilon\mathcal{S}}\lambda_m y_m K(x_n,x_m)\right) \tag{2.22}$$

Where N_S is the total number of support vectors [23][24]. Then the classification can be proposed as solving (2.14) subject to (2.13), where $W=\sum_{i=1}^{n}\lambda_i y_i \varphi(x_i)$ (2.16) this would result in next equation and represented in the Figure 2.4 [23][24].

$$f(x)=\sum_{i=1}^{N_S}\lambda_i y_i K(x,x_i) \tag{2.23}$$

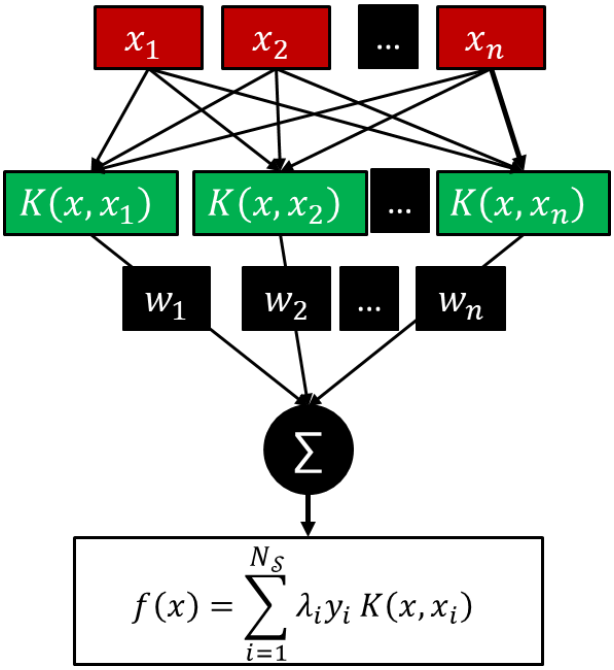

Figure 2.4. Flow diagram representing the SVM for classification, with input features, kernel function, and the weights.

The Kernel $K(x', x)$ functions that can be used to transform the data can be summarized in:

Model	Kernel
Linear	$K(x', x) = x' \cdot x$
Polynomial	$K(x', x) = (x' \cdot x + c)^d$
Radial Basis	$K(x', x) = e^{-\frac{\|x - x_c\|^2}{2\sigma^2}}$

Table 2.1 Kernel transformation formulas.

2.1.3 Decision Tree

Various wide used models work by partitioning the input space into cuboid regions, whose edges are aligned with the axes, it can be viewed as a model combination method in which only one model is responsible for predicting at any given point in input space. It can be described by a sequential decision-making process corresponding to the traversal of a binary tree (one that splits into two branches at each node). Figure 2.5 shows an illustration of a recursive partitioning of the input space, along with the corresponding tree structure, in that example, the first split is given when the input space into two regions whether $x_1 \leq \theta_1$ or $x_1 > \theta_1$ where θ_1 is a parameter of the model. That creates two subregions, that also can be subdivided, $x_1 \leq \theta_1$ or $x_1 \geq \theta_1$ gives the regions A and B. The recursive subdivision can be described by the traversal of the binary tree as shown in Figure 2.6. For any new input x, the region is determined depending where if falls starting on the root node and following the path down to a specific leaf node according to the decision criteria at each node.

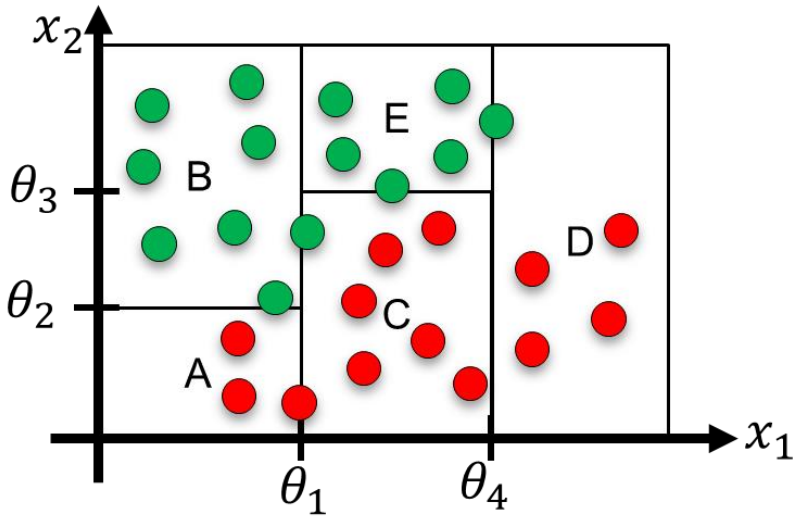

Figure 2.5. Two-dimensional representation for input space partitioned in five regions aligned with axis boundaries [23].

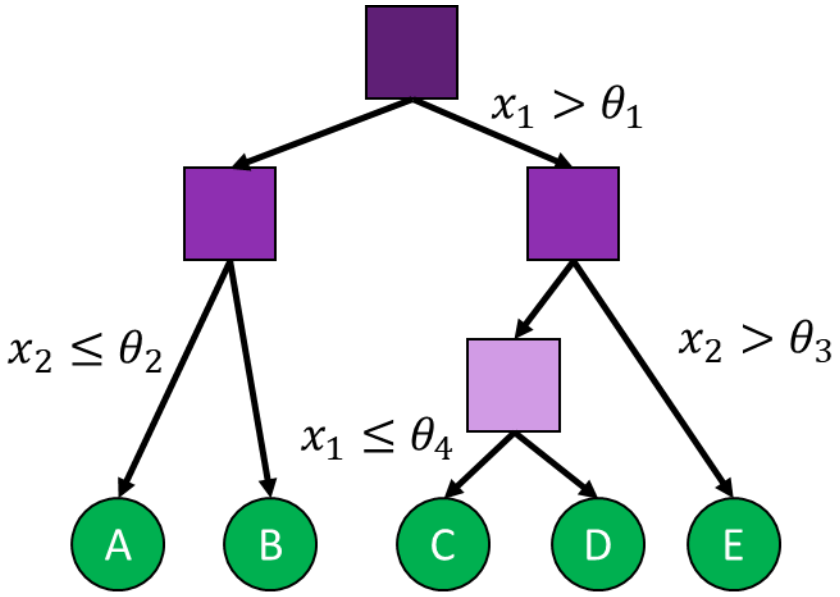

Figure 2.6. Binary tree showing the partitioning of input space as in Figure 2.5 [23].

For classification it might assign each region to a specific class, tree-based models are readily interpretable because they correspond to a sequence of binary decisions applied to individual input variables [23].

To learn such a model from the training set, it is required to determine the structure of the tree, including which input variable is chosen at each node to form the split criterion as well as the value of threshold parameter θ_i for the split and also determine the values of the predictive variable within each region [23].

Consider a regression problem in which the goal is to predict a single target y from a $p - dimensional$ vector $X = (x_1, x_2, \dots, x_p)^T$ of input variables. The training data consists of inputs vectors $\{X_1, X_2, \dots, X_n\}$ along with the corresponding continuous labels $\{y_1, y_2, \dots, y_n\}$. If the partitioning of the input space is given and it is minimized the sum-of-squares error function, then the optimal value of the predictive variable within any given region is just given by the average of the values of y_i for those data points that fall in that region. Now to determine the structure of the decision tree the problem of determining the optimal structure to minimize the sum-of-square error is usually computationally infeasible due to the large combination of a possible solution. Instead, a greedy optimization is done starting with a single root node, corresponding to the whole input space, and then growing the tree adding nodes one at a time. Each step would have some candidate regions in the input space that can be split, corresponding to the addition of a pair of leaf nodes to the existing tree. For each of these, there is a choice of which the p input variables split also the choice of the input variable and threshold, can be done efficiently by exhaustive search, the optimal choice of the predictive variable is given by the local average of the data, previews are repeated for all possible choices of a variable to split, and the one that gives the smallest residual sum-of-squares error is retained [23].

Even with the greedy strategy, there remains the issue of when to stop adding nodes, it is common to practice grow a large tree and stop criteria is based on the number of data points associated

with the leaf nodes and then prune back the resulting tree. The pruning is based on the criteria of balancing residual errors against a measure of model complexity. *E.g.* Defining a T_0 as start tree and pruning nodes from it, then $T \subset T_0$ and it is a subtree of T_0 if it can be obtained by pruning nodes from T_0. Suppose leaf nodes are given by $\tau = 1,2, \dots, |T|$ where leaf node τ denote the total number of leaf nodes, then the optimal prediction for the region $\mathcal{R}_\tau$ is given by [23]:

$$y_\tau = \frac{1}{N_\tau} \sum_{x_n \epsilon \mathcal{R}_\tau} y_n \tag{2.24}$$

For classification, the measure of performance used is defined with $p_{\tau k}$ as the proportion of data points in the region $\mathcal{R}_\tau$ assigned to class k, where $k = 1,2, \dots, m$ then the common choices are the Cross-Entropy:

$$Q_\tau(T) = \sum_{k=1}^{m} p_{\tau k} \ln p_{\tau k} \tag{2.25}$$

And the Gini index:

$$Q_\tau(T) = \sum_{k=1}^{m} p_{\tau k}(1 - p_{\tau k}) \tag{2.26}$$

The pruning criterion is then given by:

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda|T| \tag{2.27}$$

The regularization parameter λ determines the trade-off between the overall residual sum-of-squares error and the complexity of the model measured by the number $|T|$ of leaf nodes, and its value is chosen by cross-validation [22][23].

These both vanish for $p_{\tau k} = 0$ and $p_{\tau k} = 1$ and it has a maximum at $p_{\tau k} = 0.5$. Both encourage the formation of regions in which the proportion of the data of regions in which a high proportion of the data points are assigned to one class. The Cross-entropy and Gini index are better measurements than the misclassification rate for growing the tree because they are more

sensitive to the node probabilities, also unlike the misclassification rate, they are differentiable and hence better suited to gradient-based optimization methods [23].

2.1.4 Artificial Neural Network

The term Neural Network has its origins in the attempts to find a mathematical representation of information processing in the biological system [23][28][25][26][27]. In Figure 2.7 can be observed the representation of the biological neuron and the mathematical model. The usage of ANN as models for efficient pattern recognition.

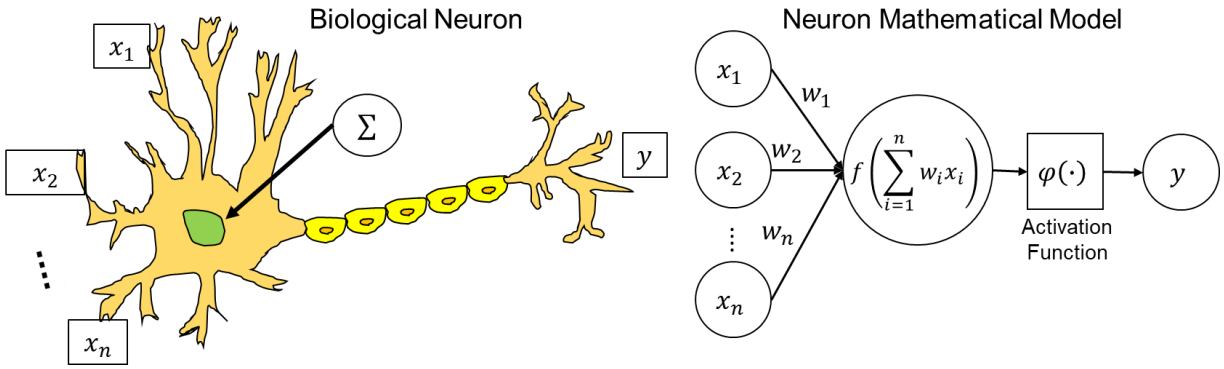

Figure 2.7. Representation of biological neuron with the analogous compare with the artificial neuron of the single perceptron.

ANN is a non-linear model build by the connection of multiple modules of signal processing called *Neurons* it can be observed as an example in Figure 2.8.

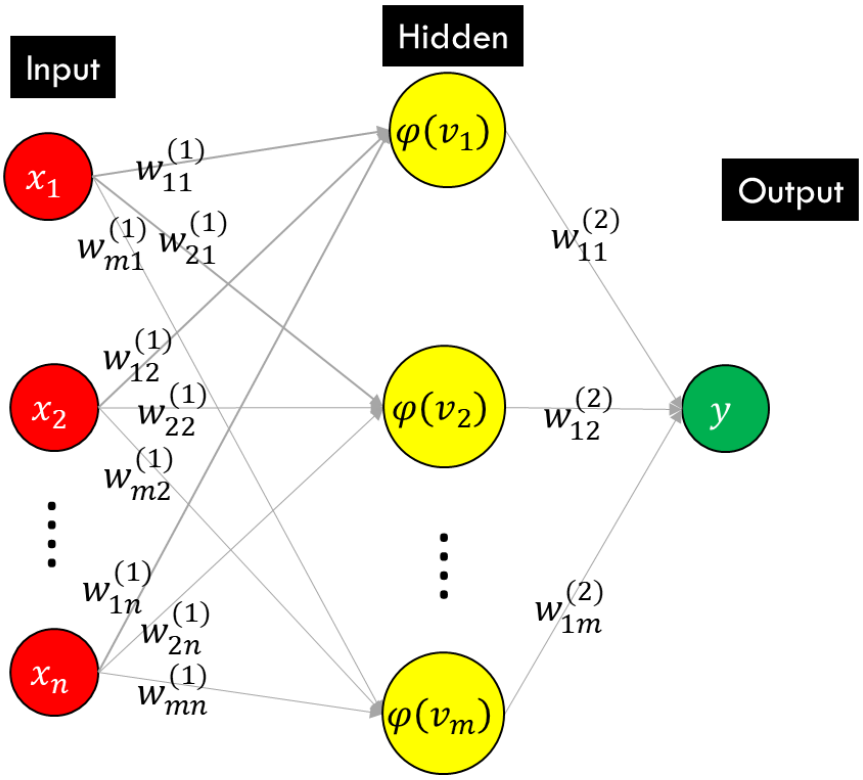

Figure 2.8.Neural Network representation for binary classification.

The models are based on linear combinations of fixed nonlinear functions $\phi_j(X)$ and it would have the form:

$$y(X, W) = \varphi\left(\sum_{j=1}^{m} w_j \phi_j(X)\right) \tag{2.28}$$

Where $\varphi(\cdot)$ is a nonlinear activation function for classification, the goal is to extend this model with the basis function $\varphi_j(X)$ that depends on the parameters adjusted with the coefficients w_j during the training process [23][28]. Then it can be formed the basic ANN model:

$$v_j = \sum_{i=1}^{n} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \Rightarrow for\ j = 1,2,\dots,m \tag{2.29}$$

The superscript (1) indicates the parameters for this case of the first layer of the network and $w_{ji}^{(1)}$ are the weights and $w_{j0}^{(1)}$ the biases. The v_j are then the activations, each one is transformed into a nonlinear activation function $\varphi(\cdot)$ *e.g.* sigmoid or hyperbolic tangent [23]. Those are the outputs of the basic function of (2.28) and for ANN they are known as the hidden units and the linear combination results in:

$$v_k = \sum_{j=1}^{m} w_{kj}^{(2)} \varphi\left(v_j^{(1)}\right) + w_{k0}^{(2)} \Rightarrow for\ k = 1,2,\dots,p \tag{2.30}$$

Where p is the number of outputs of the ANN and again $w_{kj}^{(2)}$ are the weights and $w_{k0}^{(2)}$ the biases, that gives:

$$y_k = \varphi(v_k) \tag{2.31}$$

For binary classification there is only one y then [23]:

$$\varphi(v) = \frac{1}{1+e^{-v}} \tag{2.32}$$

To comprise the training let's consider the input vectors x_i, where $i = 1,2,\dots,n$ with the corresponding target vectors y_i, the simplest approach to determine the network parameters W is given by minimizing the sum-of-squares error function or by the maximum likelihood [23][28].

$$E(W) = \frac{1}{2}\sum_{i=1}^{n} \|y(x_i, W) - y_i\|^2 \tag{2.33}$$

So the next step would be to determine the weight vector W which minimize the chosen function $E(W)$, the change rate for the weight is defined by $W + \delta W$ and change the error function is $\delta E \simeq \delta W^T \nabla E(W)$, where the $\nabla E(W)$ points in the direction of the greatest rate of the increase of the error function and the smallest value will occur at a point in weight space when the gradient function vanishes to $\nabla E(W) = 0$ [23].

To find the solution of $\nabla E(W) = 0$ is very difficult with the analytical solution, so most of the techniques involve to chose an initial value of $W^{(0)}$ for the weight vector and then moving through weight space in a succession of the steps of the form:

$$W^{(\tau+1)} = W^{(\tau)} + \Delta W^{(\tau)} \tag{2.34}$$

Where τ refers to the iteration step and the weight vector update is given by $\Delta W^{(\tau)}$ that would depend on the algorithm, most of them use the gradient information given after each update of $\nabla E(W)$ evaluated every each $W^{(\tau+1)}$.

To comprise a small step in the direction of the negative gradient, the parameter η is added and it is known as the learning rate, where $\eta > 0$ and after each update the gradient is re-evaluated for the new weight vector.

$$W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E\left(W^{(\tau)}\right) \tag{2.35}$$

The error function is defined with respect to the training set, so each step requires that the entire set is processed to evaluate ∇E, techniques as mentioned above, that uses all data set are called batch methods [23]. Another method with error functions based on the maximum likelihood for an independent set of observations, they are known as *on-line* gradient descent (also known as sequential gradient or stochastic gradient descent), and those make the weight vector update based on one data point at a time, the (2.35) would transform [23]:

$$W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E_n\left(W^{(\tau)}\right) \tag{2.36}$$

The update is repeated by cycling n times, through the data in sequence or by randomly selected points.

Now the way to evaluate the best choice for W using the backpropagation [23] for a general network having arbitrary feed-forward topology. Let's consider the problem to evaluate $\nabla E_n(W)$ in terms of error functions using a simple linear model where y_k are linear combination of variables x_i so [23]:

$$y_k = \sum_i w_{ki} x_i \tag{2.37}$$

And then with the error function that, for a particular input pattern n, takes the form [23]:

$$E_i = \frac{1}{2} \sum_k (y_k(x_i, W) - y_{ik})^2 \tag{2.38}$$

The gradient of this error function with respect to a weight w_{ji} is given by:

$$\frac{\partial E_n}{\partial w_{ji}} = \left(\hat{y}_{nj} - y_{nj}\right) x_{ni} \tag{2.39}$$

The error function (2.39) can be extended to the multilayer feedforward networks with the partial derivative form:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial \varphi_j}\frac{\partial \varphi_j}{\partial w_{ji}} \tag{2.40}$$

The term φ denotes the activation function related to each node, and the partial derivative can be defined to:

$$\delta_j \equiv \frac{\partial E_n}{\partial \varphi_j} \tag{2.41}$$

To evaluate the $\delta's$ for hidden units, the chain rule is used for partial derivatives

$$\delta_j \equiv \frac{\partial E_n}{\partial h_j} = \sum_k \frac{\partial E_n}{\partial h_k}\frac{\partial h_k}{\partial h_j} \tag{2.42}$$

Where the sum runs over all units k to which unit j sends connections. This can include other hidden units and/or output units. One fact is that the variations for φ_j give rise to variations on into φ_k, now to define the backpropagation formula [23]:

$$\delta_j = \varphi'(v_j)\sum_k w_{kj}\delta_k \tag{2.43}$$

Where $\delta_k = y_k(X, W) - y_k$, this means that the value of δ for a particular hidden unit can be obtained by propagating the δ's backward from units higher up the network, the summation takes over the first index of w_{kj}, that corresponds to the backward propagation, and the forward propagation is given by the second index. Since the values of the δ's for the output units are known, then (2.43) is applied recursively to evaluate all the δ's for the hidden units in a feed-forward network, regardless of the topology [23].

3 Classification Models for Pass/Fail Eye Margins

3.1 Data Structure

The BP is an Intel post-silicon validation platform that included a CPU and PCH. The HSIO link selected to test the methodology was SATA [14] located within the PCH. The dataset was built based on two different collections [2]:

1. **The data from the tuning stage:** Consist of multiple eye margins from a single silicon unit across several R_x EQ settings.
2. **The data from the VDC stage:** Contains eye margins over several silicon units that have a unique EQ recipe which is optimal on the tuning stage and the data were used for UPM analysis for the risk assessment decision, this is the optimal behavior of the R_x along the population targeted to cover the manufacture process variations.

These collections of data add a variety of eye width and eye height margins, with static and adaptive calibrations that were used to train the ML models with correct and incorrect behavior of the analog circuitry. The combination generated the experimental dataset X that consist of 1275 samples. Where each sample is comprised of 164 feature vectors.

$$X = \vec{x}_i \in \Re^{164} \text{ for } i = 1,2,3, \dots, 1275 \text{ samples} \tag{3.1}$$

From X dataset, a variable shall be removed: “Chip ID”. This item is a code of silicon unit an identifier for informative proposes.

For the classification dataset, the sample vector is defined as:

$$\vec{x}_i = [e_h \quad e_w \quad c_a \quad c_s]^T \tag{3.2}$$

From e_h and e_w refers to the eye height: eye high, eye low, minimum eye height, total eye height, and asymmetry between eye high and low and e_w refers to the eye width: eye left, eye right, minimum eye width, total eye width, and asymmetry between eye left and right, while the c_a and c_s are the vectors for the adaptive and static calibration settings for the Rx circuitry [2].

The ML model for classification is intended to predict the risk that supports the assessment decision with:

$$\vec{C}_i = \{1,0\} \tag{3.3}$$

3.2 Dataset

From the above equations with the features the labels $\{1,0\}$ from (3.3) that represents the *pass* or *fail* criteria for each sample of $\vec{x}_i$ depending on the calibrations of the data and this labeling was built analyzing the X by three EV engineers with different levels of expertise made a consensus to labeled $\vec{C}_i$ for each sample to reach a single set of labels of $Y = \vec{C}_i$ for $i = 1,2,\dots,1275$. This process would reflect the typical risk assessment performed on validation. This would lead to a paired set of parameters and labels for training and testing proposes of classification ML model.

$$\{X, Y\} \tag{3.4}$$

3.3 Results

From the dataset is mapped as $Y = f(X)$ and on (3.1) defines input features for p predictors and (3.3) are the output data where each $\vec{C}_i$ is the desired output. The experiments for different models to approach a good classification.

3.3.1 Logit

The experiment for this ML algorithm was performed using the python library “Logit” from “statsmodels.api” to build the classification with Logit function, the Logit contain multiple optimization functions but the model of this work, presented problems with the calculation of the inverse of the Hessian matrix, due to the number of variables when the second derivative could present a problem to be calculated, the options that offered the best way to avoid that was the usage of the “*Broyden-Fletcher-Goldfarb-Shanno*” as an optimizer, the instruction is set with: *fit(method='bfgs')*

The performance results for the Logit are the mean of the accuracy, precision, and recall (also known as sensitivity) [29], over 5 batches iterations to train the algorithm using the technique of the k-fold [24] to split data. Figure 3.1 shows the results for the Logit.

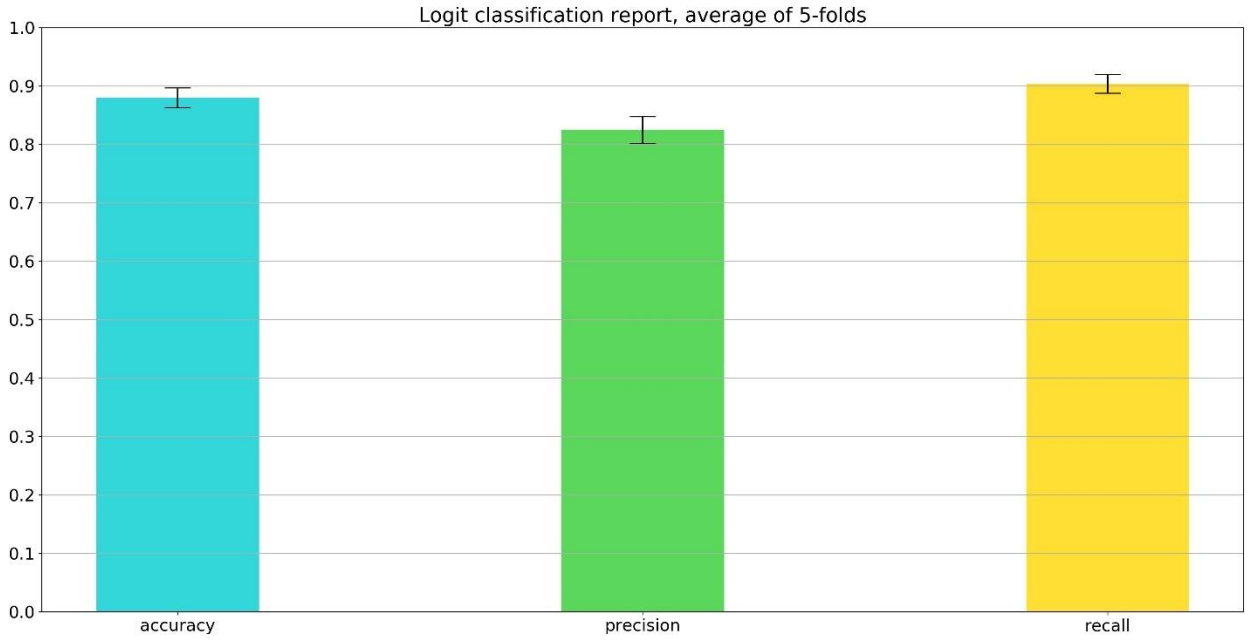

Figure 3.1. Logit mean accuracy, precision, and recall results over 5-folds test with the standard deviation (⊥)

It can be observed that the average of the 5-fold runs is:

- $Accuracy \approx 0.88$
- $Precision \approx 0.82$
- $Recall \approx 0.90$

3.3.2 Support Vector Classification

The experiment for this ML algorithm was performed using the python library "svc" from "sklearn.svm" to build the SMV for classification with the polynomial Kernel function as observed in Table 2.1, where the d degree and c coefficient of the polynomial are defined with values:

- $d = 3$
- $c = 1$

The performance results for the SVC are the mean of the accuracy, precision, and recall [29], over 5 batches iterations to train the algorithm using the technique of the k-fold to split data [24]. Figure 3.2 shows the results for the SVC.

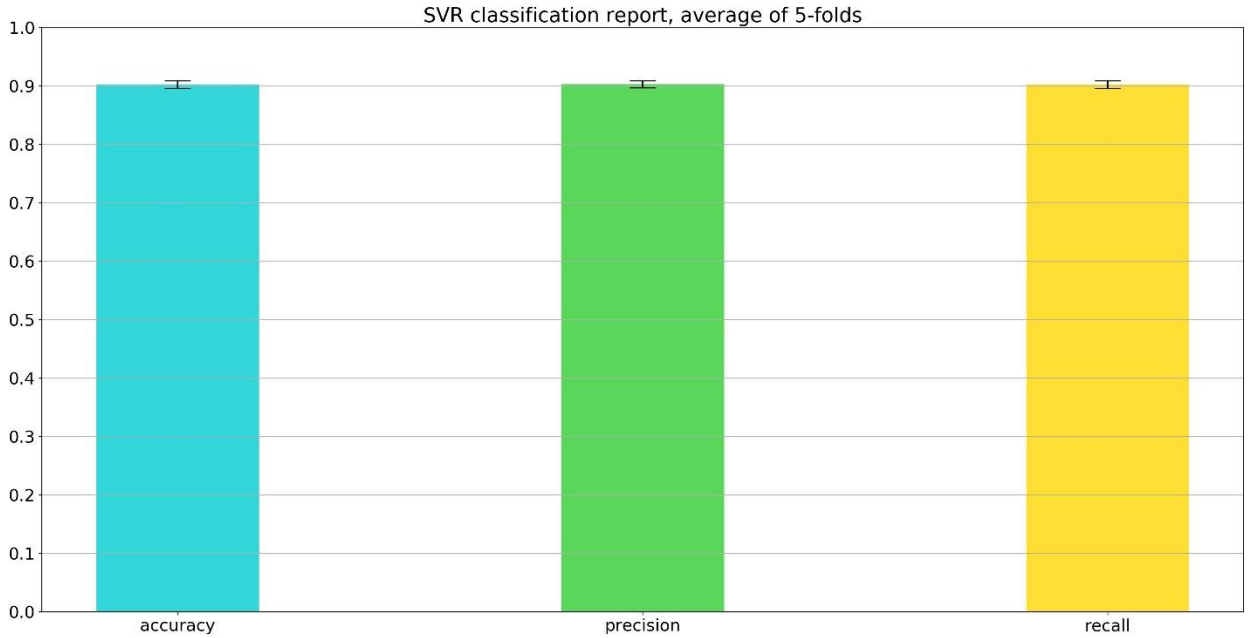

Figure 3.2. SVC mean accuracy, precision, and recall results for a 5-folds test with the standard deviation (⊥).

It can be observed that the average of the 5-fold runs is:

- $Accuracy \approx 0.90$
- $Precision \approx 0.90$
- $Recall \approx 0.90$

3.3.3 Decision Tree

The experiment for this ML algorithm with Random Forest configuration was performed using the python library “RandomForestClassifier” from “sklearn.ensemble” to build the Decision Tree for classification using Random Forest methodology.

The performance results for the Decision Tree are the mean of the accuracy, precision, and recall [29], over 5 batches iterations to train the algorithm using the technique of the k-fold [24] to split data. Figure 3.3 shows the results for the Decision Tree with the Random Forest configuration.

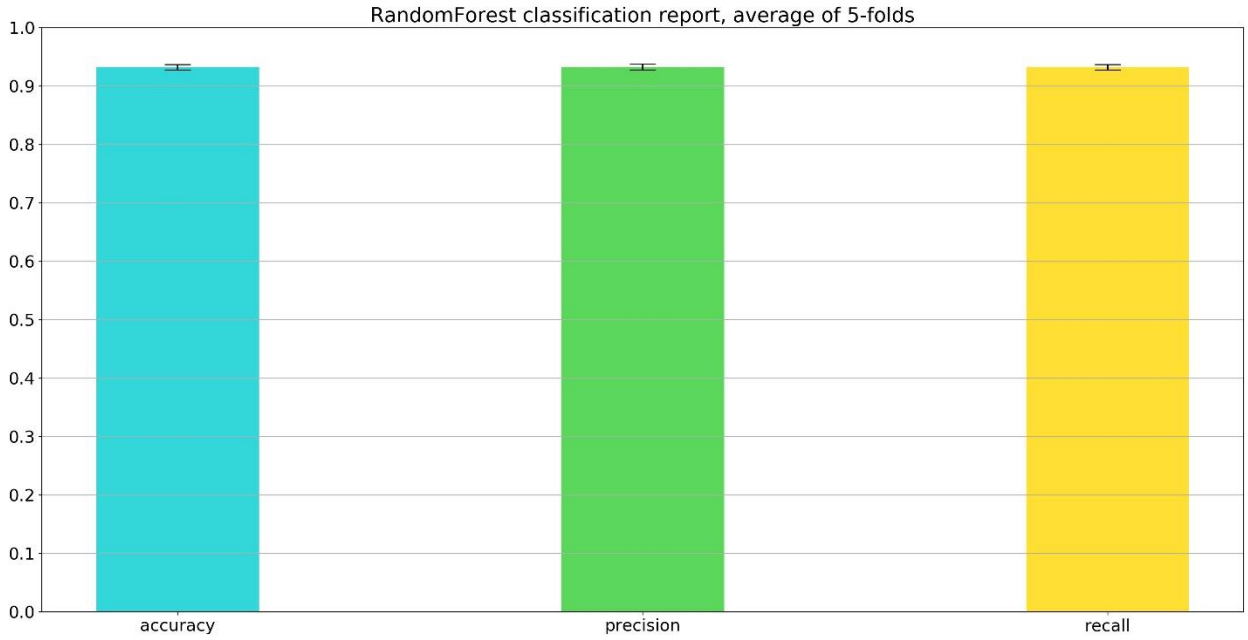

Figure 3.3. Decision Tree "Random Forest" mean accuracy, precision, and recall, results for 5-folds test with the standard deviation (⊥).

It can be observed that the average of the 5-fold runs is:

- $Accuracy \approx 0.93$
- $Precision \approx 0.93$
- $Recall \approx 0.93$

3.3.4 Artificial Neural Network

To build the Artificial NN, it was used the python library "Sequence" from "keras.models" for the frame of the ANN, the library "Dense" from "keras.layers" to add the density of neurons and activation function for each layer of the ANN, and the Stochastic Gradient Descent function with a loss measured with "binary cross-entropy" configuration. The structure of the ANN is:

- Input layer:
  - Activation function: *hyperbolic tangent*
  - Quantity of neurons: 163
- Hidden layer:
  - Activation function: *hyperbolic tangent*
  - Quantity of neurons: 48×48
- Output layer:
  - Activation function: *Sigmoid*
  - Quantity of neurons: 1

The hidden layer quantity of neurons is defined using references from [30] and the ANN was trained when it reached the 100 epochs.

The performance results for the Neural Network are the mean of the accuracy, precision, and recall [29], over 5 batches iterations to train the algorithm using the technique of the k-fold [24] to split data. Figure 3.4 shows the results for the ANN.

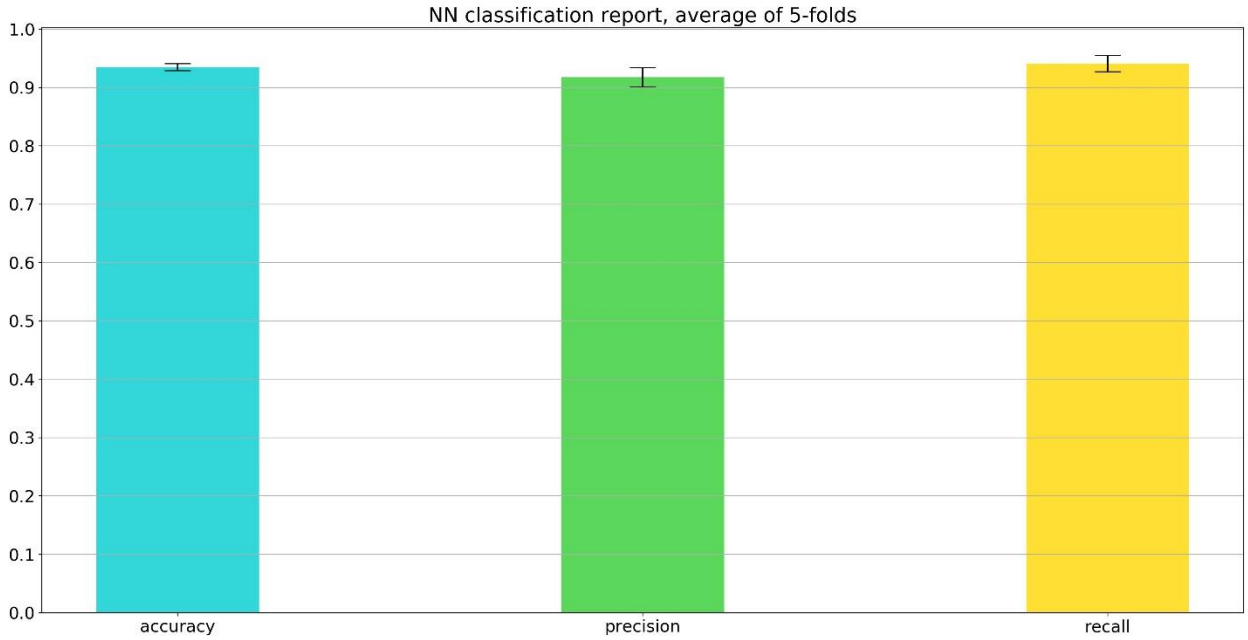

Figure 3.4. Neural Network mean accuracy, precision, and recall, results for 5-folds test with the standard deviation (⊥).

It can be observed that the average of the 5-fold runs is:

- $Accuracy \approx 0.94$
- $Precision \approx 0.91$
- $Recall \approx 0.94$

3.4 Model Comparison

Summary of the results and comparison across models in Table 3.1.

Model	Type	Configuration	Accuracy	Precision	Recall
1	Logit	*Optimizer*: "Broyden-Fletcher-Goldfarb-Shanno"	0.88	0.82	0.90
2	SVC	Polynomial $degree = 3, constant = 1$	0.90	0.90	0.90
3	NN	*Input layer*: Hyperbolic Tangent *Hidden layer*: Hyperbolic Tangent *Output layer*: Linear	0.94	0.91	0.94
4	Decision Tree	Random Forest	0.93	0.93	0.93

Table 3.1. Evaluation of each model based on accuracy, precision, and recall.

3.5 Chapter Conclusions

As observed in Table 3.1 complex algorithms showed a better response to classifying the Pass/Fail criteria for the Eye Margins, as it was observed from each graph where the standard deviation bar is shown as 工.

For the Logit regression model in Figure 3.1, the means and the standard deviation showed some inconsistency on the training of the model over each fold iteration which affects the capacity to classify.

For SVC with a cubic kernel in Figure 3.2, it can be observed a small standard deviation for accuracy, precision, and recall, which means consistency on the proper training of the model over each fold iteration and the capacity to classify.

For Decision Tree with Random Forest in Figure 3.3 it can be observed that each bar has a small standard deviation for accuracy, precision, and recall, which means consistency on the proper training of the model over each fold iteration and the capacity to classify.

For the ANN model in Figure 3.4, it can be observed that accuracy has a small standard deviation (工) and a bit more wide open for precision and recall, compared with Decision Tree and SVC but smaller than the Logit which can be also considered as consistency on the proper training of the model over each fold iteration and the capacity to classify.

4 Mathematical Preliminaries For Regression Models

This would generate synthetic Eye Margins estimation based on real calibrations data which would reduce considerably the validation time since the capture of EH and EW across multiple silicon units is a very time-consuming activity and the read of the calibration values is very quick to perform.

4.1 Models

For this part four artificial intelligence algorithms were selected:

1. Linear Regression with Ordinary Least Square (OLS).
2. Support Vector Machine for Regression (SVR).
3. Artificial Neural Networks (ANN).
4. Decision Tree with Random Forest methodology.

4.1.1 Linear Regression

The regression analysis is a statistical technique to investigate and model the relationship between variables, the simplest form is the relationship of the independent variable x with the dependent variable y, as can be observed in Figure 4.1 the equation of the line represents that relationship [31]:

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{4.1}$$

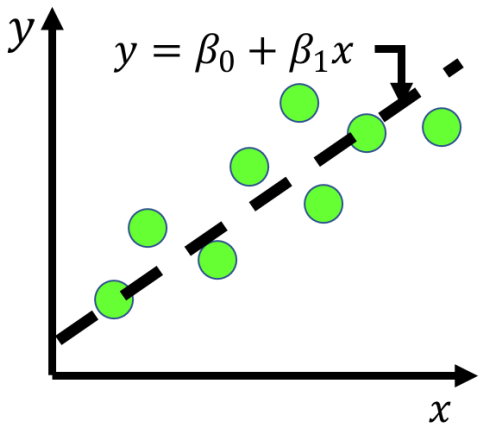

Figure 4.1. Simple linear regression for estimation y given the intercept and coefficient β for any given x point.

Where β_0 is the intercept and β_1 is the slope and ε is the component of the random error. It is assumed that the errors have zero mean and unknown variance and also that the errors are not correlated across them. So it can be said that y is a linear function of x, β_0 and β_1 are the regression coefficients where β_1 is the change rate of the mean distribution produced by y when a change occurs on x. If the interval of the data includes $x = 0$, then the intercept β_0 is the mean of the distribution of the y when $x = 0$.

The parameters β_0 and β_1 are unknown and they shall be estimated with the data from the sample, if exist n quantity of pair of data then exists: $(y_1, x_1), (y_1, x_1), \dots, (y_n, x_n)$, then the estimation is performed with the sum of the square across the differences of the observations of y_i and the fit line when those are minimal, generalizing the equation [31]:

$$y = \beta_0 + \beta_1 x_i + \varepsilon, \quad for\ i = 1,2, \dots, n \tag{4.2}$$

And the criteria for the least-squares is:

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 \tag{4.3}$$

The estimators, for least squares, for β_0 and β_1 that would be designated by $\hat{\beta}_0$ and $\hat{\beta}_1$, they shall satisfy:

$$\left.\frac{\partial S}{\partial \beta_0}\right|_{\beta_0, \beta_1} = -2 \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right) = 0 \tag{4.4}$$

$$\left.\frac{\partial S}{\partial \beta_1}\right|_{\beta_0, \beta_1} = -2 \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right) x_i = 0 \tag{4.5}$$

The solutions for those equations are:

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 x_i \tag{4.6}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} y_i x_i - \frac{(\sum_{i=1}^{n} y_i)(\sum_{i=1}^{n} x_i)}{n}}{\sum_{i=1}^{n} (x_i)^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}} \tag{4.7}$$

The above coefficients and from (4.1) and (4.2) can be extended to multiple variable regression and the form is [31]:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_n x_{in} + \varepsilon \tag{4.8}$$

$$y_i = \beta_0 + \sum_{j=1}^{n} \beta_j x_{ij} + \varepsilon_i, \quad for\ i = 1,2,\ldots,n \tag{4.9}$$

4.1.2 Support Vector Regression

The principle used in this regression algorithm is the usage of a separation hyperplane, in a $p - dimensional$ space, a hyperplane is a flat affine subspace, in an instance, a two-dimension hyperplane is a flat dimensional subspace (a line), in three dimensions is the hyperplane would be a plane [24], as observed in Figure 4.2, and with dimensions greater than three it is hard to visualize but the notion is that the subspace for the hyperplane would be $p - 1$, the hyperplane equation is given as a sum w_p of coefficients and each multiplied by the variable x_p, where w_0 or b is the intercept of the equation [24]:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_{p-1} x_{p-1} = w_0 + W^T X = W^T X + b \tag{4.10}$$

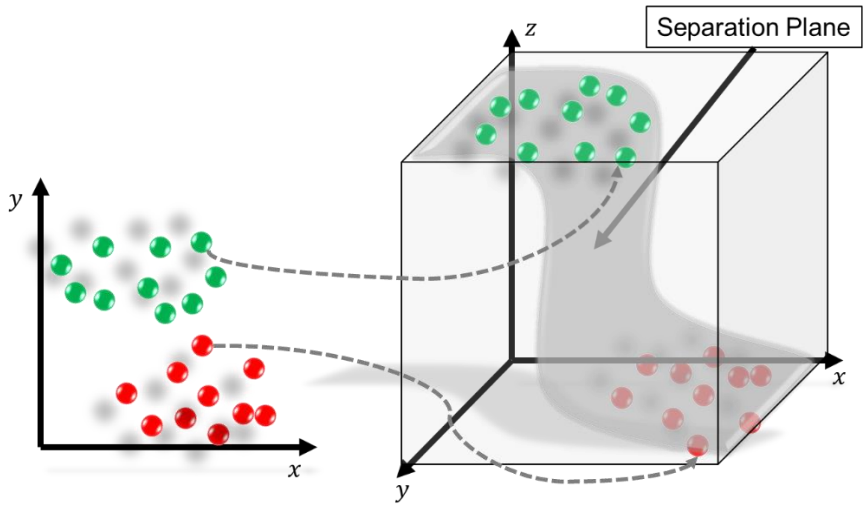

Figure 4.2. Representation of sub-space (plane) on a 3-dimensional space.

As observed in Figure 4.2, it would be required to add a separation between the different data points to start the description, let's use the linear model:

$$f(x) = W^T \varphi(x) + b \tag{4.11}$$

Where $\varphi(x)$ denotes a fixed featured space transformation and *b* is the bias parameter, assuming that exists one choice of W and *b* that satisfice the linear separation of the data set, as observed in Figure 4.3, the training dataset comprises N input vectors $x_1 + x_2 + \cdots + x_N$ with a corresponding target values $y_1 + y_2 + \cdots + y_N$ where $y_n \epsilon \{-1,1\}$ and new points of x are separated according to the sign of $f(x)$.

There may are many solutions that could separate the classes exactly, but that solution depends on the arbitrary initial values chosen for W and *b* as well as the order in which the points of the data set are presented, if exist many possible solutions there should be found the one that gives the smallest generalization error, the SVM approach that problem through the concept of "margin" denoted by γ, which shall be the smallest distance between the decision boundary and any of the data samples, as can be observed in Figure 4.3, it can be defined as the perpendicular distance between the decision boundary and the closest of the data points. For the SVM the boundary is chosen to be the one for which the margin is maximized [23][24].

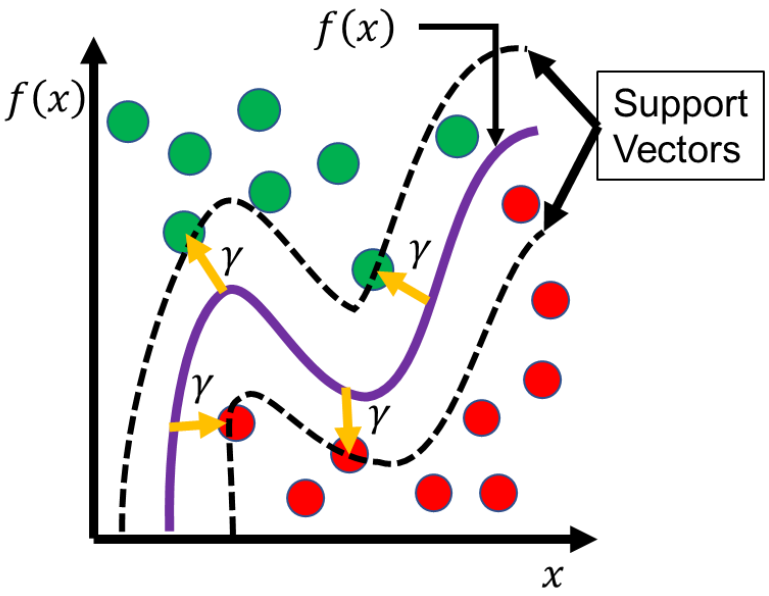

Figure 4.3. Representation of the hyperplane, separation margins, and support vectors.

To define the best separation hyperplane there should be minimized the probability of error relative to the learned on the model (4.10).

$$\gamma_i = \frac{W^T}{\|W\|} x_i, \gamma = \min \gamma_i \Rightarrow for\ i = 1,2,\ldots,n \tag{4.12}$$

The optimal hyperplane is the one having the maximum margin and it is desired to optimize the parameters W and *b* from (4.11) to maximize the distance. The regression for the point that is closest to the surface, and also all data points will satisfy the constraints:

$$g(W) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4.13}$$

And:

$$\begin{cases} 0 & \Rightarrow |y_i - \hat{y}_i| \leq \varepsilon \\ |y_i - \hat{y}_i| & \Rightarrow |y_i - \hat{y}_i| > \varepsilon \end{cases} \tag{4.14}$$

Where the ε-*intensive* error function is added, which should give zero if the absolute difference between the prediction and the target value is less than ε where $\varepsilon > 0$

This is known as the canonical representation of the decision hyperplane, by definition, there will always be at least one active constraint because there will be always at least one closest point, also observed in Figure 4.3, and once the margin is maximized there will be at least two active constrains, then the optimization problem would require that $\|W\|^{-1}$ is maximized, that is equivalent to minimize $\|W\|^2$, so the optimization problem now converts into [23][24]:

$$\underset{W,b}{min} \left[\frac{1}{2}\|W\|^2\right] \tag{4.15}$$

The factor $1/2$ is added for convenience, to solve the constrained optimization problem, for regression it is desired to minimize a regularized error function [23]:

$$\frac{\lambda}{2}\|W\|^2 - \sum_{i=1}^{n} (\hat{y}_i - y_i) \tag{4.16}$$

The ε-*intensive* error function would give a change on the constraints for Lagrange function:

$$E_\varepsilon(W^T X - y) = \begin{cases} 0 & \Rightarrow if\ |\hat{y}_i - y| \leq \varepsilon \\ |\hat{y}_i - y| & \Rightarrow otherwise \end{cases} \tag{4.17}$$

Where $\hat{y}_i - y - \varepsilon \leq 0$ and $\hat{y}_i - y - \varepsilon \leq 0$, a graphical representation can be observed in Figure 4.4.

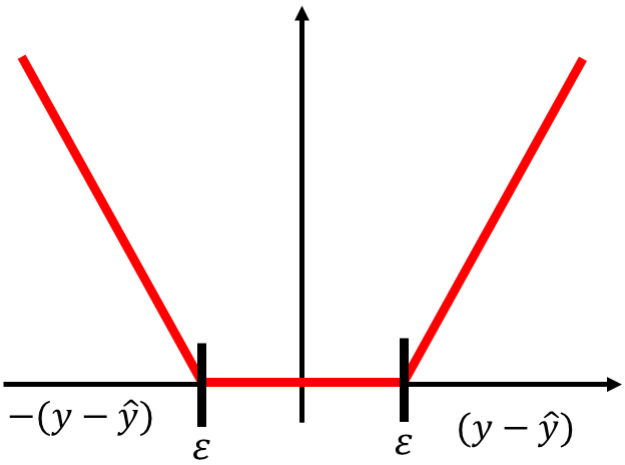

Figure 4.4. ε-intensive error function representation where error increases linearly with distance beyond the intensive region

Therefore the minimize a regularized function given by:

$$\frac{1}{2}\|W\|^2 - C\sum_{i=1}^{n} E_\varepsilon(\hat{y}_i - y) \tag{4.18}$$

Where C is the regularization parameter, and slack variables are added to solve the optimization problem $\xi_i \geq 0$ and $\hat{\xi}_i \geq 0$. The condition of the target point lies inside of ε-tuber that is $y_i - \varepsilon \leq \hat{y}_i \leq y_i + \varepsilon$ where $y_i = \hat{y}_i$. Therefore the slack variables allow points to lie outside of the tube with conditions $y_i \leq \hat{y}_i + \varepsilon + \xi_i$ and $y_i \geq \hat{y}_i - \varepsilon - \hat{\xi}_i$

So the error function can be redefined as:

$$\frac{1}{2}\|W\|^2 - C\sum_{i=1}^{n}(\xi_i + \hat{\xi}_i) \tag{4.19}$$

Which would be minimized subject to constraints $\xi_i \geq 0$ and $\hat{\xi}_i \geq 0$ this can be achieved with Lagrange multipliers $\lambda_i \geq 0$, $\lambda_i^* \geq 0$, $\mu_i \geq 0$ and $\hat{\mu}_i \geq 0$ [23]:

$$\begin{aligned} L(W, \lambda, \lambda^*, \xi, \hat{\xi}) = \frac{1}{2}\|W\|^2 + C\sum_{i=1}^{n}(\xi_i + \hat{\xi}_i) - \sum_{i=1}^{n}(\xi_i\mu_i + \hat{\xi}_i\,\hat{\mu}_i) \\ - \sum_{i=1}^{n}\lambda_i(\varepsilon + \xi_i + \hat{y}_i - y_i) - \sum_{i=1}^{n}\lambda_i^*(\varepsilon + \xi_i + y_i - \hat{y}_i) \end{aligned} \tag{4.20}$$

Now it would be necessary to set partial derivatives of $L(W, \lambda, \lambda^*, \xi, \hat{\xi})$ respect W, b, ξ and $\hat{\xi}$ equal to zero, then the following conditions are obtained by replacing $\hat{y}_i$ with (4.11):

$$\frac{\partial L}{\partial W} = 0 \Rightarrow \sum_{i=1}^{n} (\lambda_i - \lambda_i^*)\varphi(x_i) = W \tag{4.21}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i^* = 0 \tag{4.22}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow \mu_i + \lambda_i = C \tag{4.23}$$

$$\frac{\partial L}{\partial \hat{\xi}_i} = 0 \Rightarrow \hat{\mu}_i + \lambda_i^* = C \tag{4.24}$$

With W it can be built again (4.20) and consider the equivalence of $\|W\|^2 = W^T W$ and eliminating W from L using those conditions then gives the dual representation of the maximum margin problem where it has been maximized [23][24]:

$$\tilde{L}(\lambda, \lambda^*) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i - \lambda_i^*)(\lambda_j - \lambda_j^*)x_i \cdot x_j + \sum_{i=1}^{n}(\lambda_i - \lambda_i^*)y_i - \sum_{i=1}^{n}(\lambda_i^* + \lambda_i)\varepsilon \tag{4.25}$$

With respect λ_i and λ_i^* the kernel function K and it is defined as $K(x, x') = \varphi(x)^T \varphi(x')$, this kernel formulation makes the role of the constraint that the function $K(x, x')$ where $\lambda_i \geq 0$ and $\lambda_i^* \geq 0$ required for Lagrange multipliers where $\mu_i \geq 0$, $\hat{\mu}_i \geq 0$, $\lambda_i \leq C$ and $\lambda_i^* \leq C$.

To perform the regression of new data with the trained model, the sign of $y(x)$ is defined by (4.11). This can be expressed in terms of $\{\lambda_i\}$ and the kernel function by substituting W from (4.21) to give [23]:

$$y(x) = \sum_{i=1}^{n} (\lambda_i^* - \lambda_i) K(x, x_i) + b \tag{4.26}$$

.

The parameter b can be found considering $0 < \lambda_i < C$ for $\xi_i = 0$ and it must be satisfied $\varepsilon + y_i - \hat{y}_i = 0$, then :

$$b = y_i - \varepsilon - \sum_{j=1}^{m} (\lambda_j - \lambda_j^*) K(x_i, x_j) \tag{4.27}$$

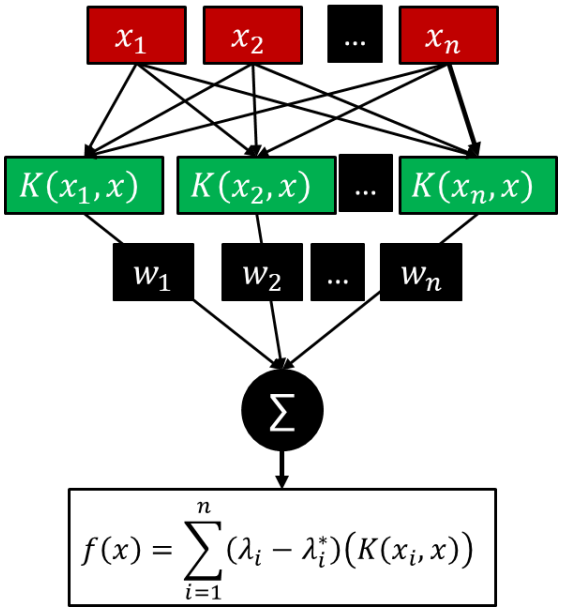

Figure 4.5. Flow diagram representing the SVM for regression, with input features, kernel function, and the weights.

The Kernel $K(x', x)$ functions that can be used to transform the data can be summarized in:

Model	Kernel
Linear	$K(x', x) = x' \cdot x$
Polynomial	$K(x', x) = (x' \cdot x + c)^d$
Radial Basis	$K(x', x) = e^{-\frac{\|x - x_c\|^2}{2\sigma^2}}$

Table 4.1 Kernel transformation formulas.

4.1.3 Decision Tree

Various wide used models work by partitioning the input space into cuboid regions, whose edges are aligned with the axes, it can be viewed as a model combination method in which only one model is responsible for predicting at any given point in input space. It can be described by a sequential decision-making process corresponding to the traversal of a binary tree (one that splits into two branches at each node). Figure 4.6 shows an illustration of a recursive partitioning of the input space, along with the corresponding tree structure, in that example, the first split is given

when the input space into two regions whether $x_1 \leq \theta_1$ or $x_1 > \theta_1$ where θ_1 is a parameter of the model. That creates two subregions, that also can be subdivided, $x_1 \leq \theta_1$ or $x_1 \geq \theta_1$ gives the regions A and B. The recursive subdivision can be described by the traversal of the binary tree as shown in Figure 4.7. For any new input x, the region is determined depending where if falls starting on the root node and following the path down to a specific leaf node according to the decision criteria at each node.

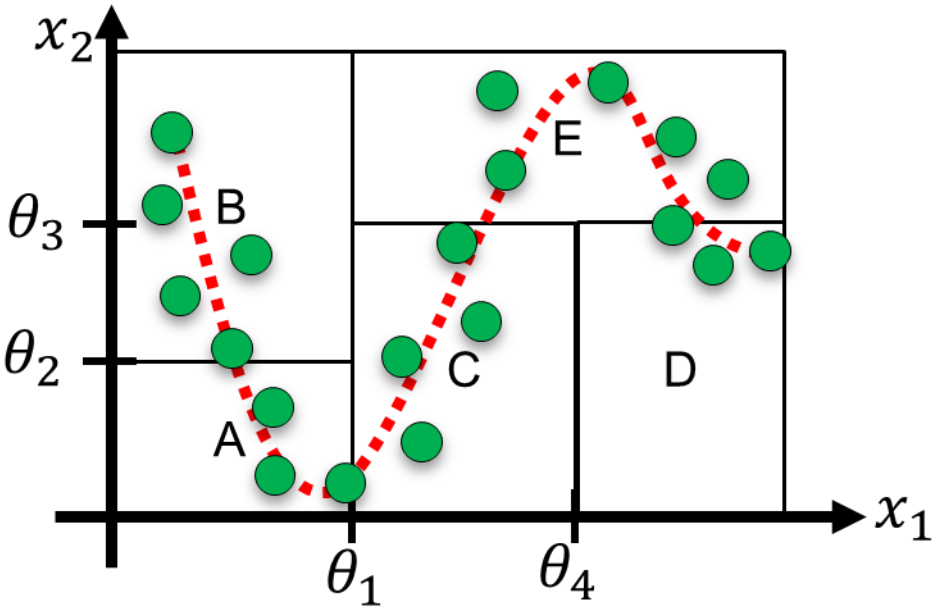

Figure 4.6. Two-dimensional representation for input space partitioned in five regions aligned with axis boundaries [23].

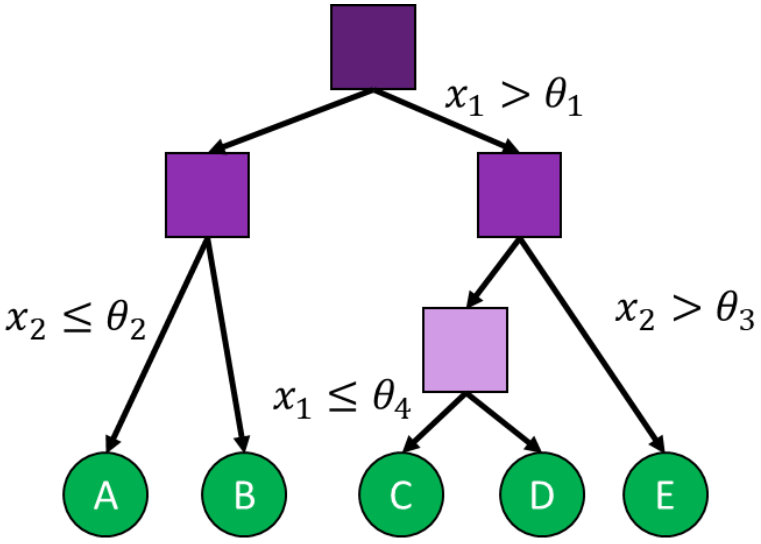

Figure 4.7. Binary tree showing the partitioning of input space as in Figure 4.6 [23].

For regression simply it is desired to predict a target variable, tree-based models are readily interpretable because they correspond to a sequence of binary decisions applied to individual input variables [23].

To learn such a model from the training set, it is required to determine the structure of the tree, including which input variable is chosen at each node to form the split criterion as well as the value of threshold parameter θ_i for the split and also determine the values of the predictive variable within each region [23].

Consider a regression problem in which the goal is to predict a single target y from a $p - dimensional$ vector $X = \left(x_1, x_2, \dots, x_p\right)^T$ of input variables. The training data consists of inputs vectors $\{X_1, X_2, \dots, X_n\}$ along with the corresponding continuous labels $\{y_1, y_2, \dots, y_n\}$. If the partitioning of the input space is given and it is minimized the sum-of-squares error function, then the optimal value of the predictive variable within any given region is just given by the average of the values of y_i for those data points that fall in that region. Now to determine the structure of the decision tree the problem of determining the optimal structure to minimize the sum-of-square error is usually computationally infeasible due to the large combination of the possible solution. Instead, a greedy optimization is done starting with a single root node, corresponding to the whole input space, and then growing the tree adding nodes one at a time. Each step would have some candidate regions in the input space that can be split, corresponding to the addition of a pair of leaf nodes to the existing tree. For each of these, there is a choice of which the p input variables split also the choice of the input variable and threshold, can be done efficiently by exhaustive search, the optimal choice of a predictive variable is given by the local average of the data, previews are repeated for all possible choices of a variable to split, and the one that gives the smallest residual sum-of-squares error is retained [23].

Even with the greedy strategy, there remains the issue of when to stop adding nodes, it is common to practice grow a large tree and stop criteria is based on the number of data points associated with the leaf nodes and then prune back the resulting tree. The pruning is based on the criteria of balancing residual errors against a measure of model complexity. *E.g.* Defining a T_0 as start tree and pruning nodes from it, then $T \subset T_0$ and it is a subtree of T_0 if it can be obtained by pruning nodes from T_0. Suppose leaf nodes are given by $\tau = 1,2, \dots, |T|$ where leaf node τ denote the total number of leaf nodes, then the optimal prediction for the region $\mathcal{R}_\tau$ is given by [23]:

$$y_\tau = \frac{1}{N_\tau} \sum_{x_n \epsilon \mathcal{R}_\tau} y_n \tag{4.28}$$

And the contribution to the residual of sum-of-squares is:

$$Q_\tau(T) = \sum_{x_n \epsilon \mathcal{R}_\tau} \{y_n - y_\tau\}^2 \tag{4.29}$$

The pruning criterion is then given by:

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda |T| \tag{4.30}$$

The regularization parameter λ determines the trade-off between the overall residual sum-of-squares error and the complexity of the model measured by the number $|T|$ of leaf nodes, and its value is chosen by cross-validation [22][23].

4.1.4 Artificial Neural Network

The term Neural Network has its origins in the attempts to find a mathematical representation of information processing in the biological system [23][28][25][26][27], in Figure 4.8 can be observed the representation of the biological neuron and the mathematical model. The usage of ANN as models for efficient pattern recognition.

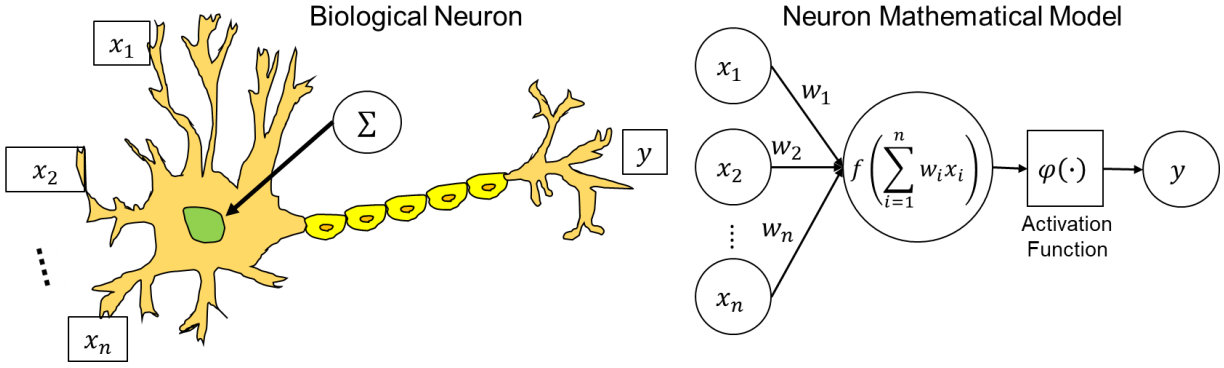

Figure 4.8. Representation of biological neuron with the analogous compare with the artificial neuron of the single perceptron

ANN is a non-linear model build by the connection of multiple modules of signal processing called Neurons it can be observed as an example in Figure 4.9.

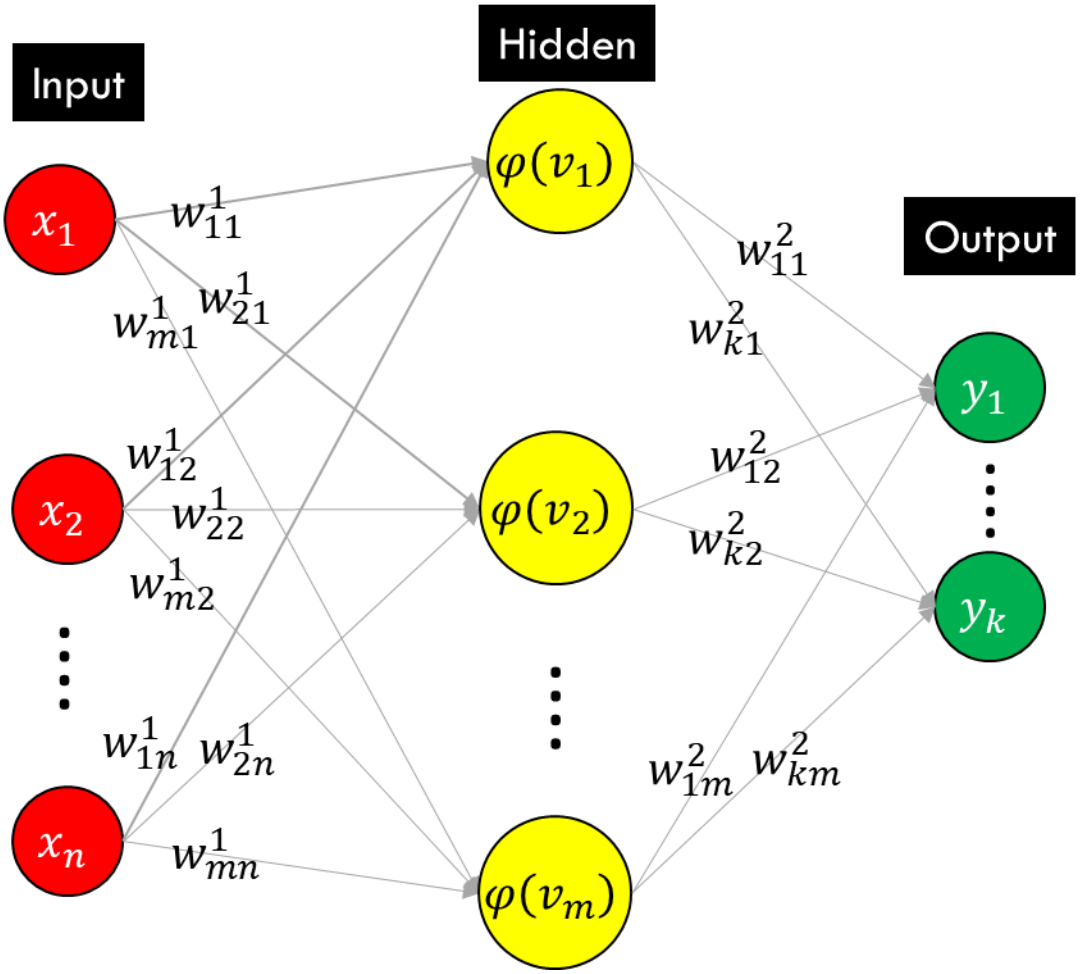

Figure 4.9. Neural Network representation for multiple outputs.

The models are based on linear combinations of fixed nonlinear functions $\phi_j(X)$ and it would have the form:

$$y(X, W) = \varphi\left(\sum_{j=1}^{m} w_j \phi_j(X)\right) \tag{4.31}$$

Where $\varphi(\cdot)$ is a nonlinear activation function for classification, the goal is to extend this model with the basis function $\varphi_j(X)$ that depends on the parameters adjusted with the coefficients w_j during the training process [23][28]. Then it can be formed the basic ANN model:

$$v_j = \sum_{i=1}^{n} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \Rightarrow for\ j = 1,2,\dots,m \tag{4.32}$$

The superscript (1) indicates the parameters for this case of the first layer of the network and $w_{ji}^{(1)}$ are the weights and $w_{j0}^{(1)}$ the biases. The v_j are then the activations, each one is transformed into a nonlinear activation function $\varphi(\cdot)$ *e.g.* sigmoid or hyperbolic tangent [23]. Those are the outputs of the basic function of (4.31) and for ANN they are known as the hidden units and the linear combination results in:

$$v_k = \sum_{j=1}^{m} w_{kj}^{(2)} \varphi\left(v_j^{(1)}\right) + w_{k0}^{(2)} \Rightarrow for\ k = 1,2,\dots,p \tag{4.33}$$

Where p is the number of outputs of the ANN and again $w_{kj}^{(2)}$ are the weights and $w_{k0}^{(2)}$ the biases, that gives [23]:

$$y_k = \varphi(v_k) \tag{4.34}$$

For regression some of the activation functions are observed in Figure 4.10:

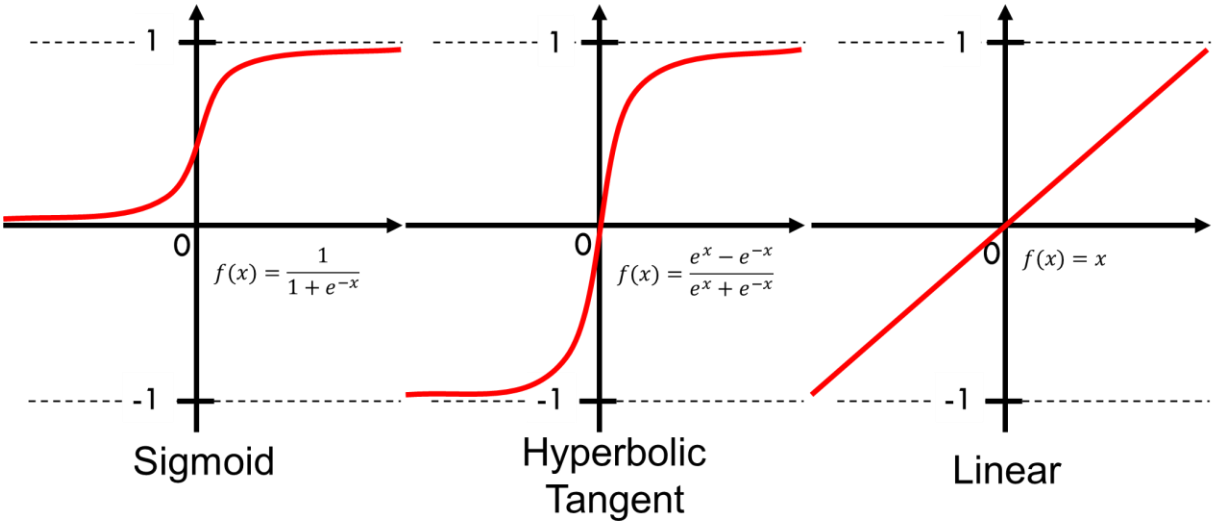

Figure 4.10. Activation functions for neurons. Left: Sigmoid, middle: Hyperbolic Tangent and right: Linear

To comprise the training let's consider the input vectors x_i, where $i = 1,2,\dots,n$ with the corresponding target vectors y_i, the simplest approach to determine the network parameters W is given by minimizing the sum-of-squares error function or by the maximum likelihood [23][28].

$$E(W) = \frac{1}{2}\sum_{i=1}^{n} \|y(x_i, W) - y_i\|^2 \tag{4.35}$$

So the next step would be to determine the weight vector W which minimize the chosen function $E(W)$, the change rate for the weight is defined by $W + \delta W$ and change the error function is $\delta E \simeq \delta W^T \nabla E(W)$, where the $\nabla E(W)$ points in the direction of the greatest rate of the increase of the error function and the smallest value will occur at a point in weight space when the gradient function vanishes to $\nabla E(W) = 0$ [23].

To find the solution of $\nabla E(W) = 0$ is very difficult with the analytical solution, so most techniques involve choosing an initial value of $W^{(0)}$ for the weight vector and then moving through weight space in a succession of the steps of the form:

$$W^{(\tau+1)} = W^{(\tau)} + \Delta W^{(\tau)} \tag{4.36}$$

Where τ refers to the iteration step and the weight vector update is given by $\Delta W^{(\tau)}$ that would depend on the algorithm, most of them use the gradient information given after each update of $\nabla E(W)$ evaluated every each $W^{(\tau+1)}$.

To comprise a small step in the direction of the negative gradient, the parameter η is added and it is known as the learning rate, where $\eta > 0$ and after each update the gradient is re-evaluated for the new weight vector.

$$W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E\left(W^{(\tau)}\right) \tag{4.37}$$

The error function is defined with respect to the training set, so each step requires that the entire set is processed to evaluate ∇E, techniques as mentioned above, that uses all data set are called batch methods [23]. Another method with error functions based on the maximum likelihood for an independent set of observations, they are known as on-line gradient descent (also known as sequential gradient or stochastic gradient descent), and those make the weight vector update based on one data point at a time, the (2.35) would transform [23]:

$$W^{(\tau+1)} = W^{(\tau)} - \eta \nabla E_n\left(W^{(\tau)}\right) \tag{4.38}$$

The update is repeated by cycling n times, through the data in sequence or by randomly selected points.

Now the way to evaluate the best choice for W using the backpropagation [23] for a general network having arbitrary feed-forward topology. Let's consider the problem to evaluate $\nabla E_n(W)$

in terms of error functions using a simple linear model where y_k are linear combination of variables x_i so [23]:

$$y_k = \sum_i w_{ki} x_i \tag{4.39}$$

And then with the error function that, for a particular input pattern n, takes the form [23]:

$$E_i = \frac{1}{2} \sum_k (y_k(x_i, W) - y_{ik})^2 \tag{4.40}$$

The gradient of this error function with respect to a weight w_{ji} is given by:

$$\frac{\partial E_n}{\partial w_{ji}} = (\hat{y}_{nj} - y_{nj}) x_{ni} \tag{4.41}$$

The error function (4.42) can be extended to the multilayer feedforward networks with the partial derivative form:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial \varphi_j} \frac{\partial \varphi_j}{\partial w_{ji}} \tag{4.42}$$

The term φ denotes the activation function related to each node, and the partial derivative can be defined to:

$$\delta_j \equiv \frac{\partial E_n}{\partial \varphi_j} \tag{4.43}$$

To evaluate the $\delta' s$ for hidden units, the chain rule is used for partial derivatives

$$\delta_j \equiv \frac{\partial E_n}{\partial h_j} = \sum_k \frac{\partial E_n}{\partial h_k} \frac{\partial h_k}{\partial h_j} \tag{4.44}$$

Where the sum runs over all units k to which unit j sends connections. This can include other hidden units and/or output units. One fact is that the variations for φ_j give rise to variations on into φ_k, now to define the backpropagation formula [23]:

$$\delta_j = \varphi'(v_j) \sum_k w_{kj}\delta_k \tag{4.45}$$

Where $\delta_k = y_k(X, W) - y_k$, this means that the value of δ for a particular hidden unit can be obtained by propagating the δ's backward from units higher up the network, the summation takes over the first index of w_{kj}, that corresponds to the backward propagation, and the forward propagation is given by the second index. Since the values of the δ's for the output units are known, then (4.45) is applied recursively to evaluate all the δ's for the hidden units in a feed-forward network, regardless of the topology [23].

5 Regression Models Eye Margins Estimation

5.1 Data Structure

The BP is an Intel post-silicon validation platform that included a CPU and PCH. The HSIO link selected to test the methodology was SATA [14] located within the PCH. The dataset was built based on two different collections [2]:

1. **The data from the tuning stage:** Consist of multiple eye margins from a single silicon unit across several R_x EQ settings.
2. **The data from the VDC stage:** Contains eye margins over several silicon units that have a unique EQ recipe which is optimal on the tuning stage and the data were used for UPM analysis for the risk assessment decision, this is the optimal behavior of the R_x along the population targeted to cover the manufacture process variations.

These collections of data add a variety of eye width and eye height margins, with static and adaptive calibrations that were used to train the ML models with correct and incorrect behavior of the analog circuitry. The combination generated the experimental dataset $\boldsymbol{X}$ that consist of 1275 samples. Where each sample is comprised of 164 feature vectors, as observed on (5.1).

$$X = \vec{x}_i \in \Re^{164} \text{ for } i = 1,2,3,\dots,1275 \text{ samples} \tag{5.1}$$

From $\boldsymbol{X}$ dataset some variables shall be removed, those are:

1. Chip ID.
2. Eye width asymmetry.
3. Eye height asymmetry.
4. Total eye width.
5. Total eye height.
6. Minimum eye width side (left/right).
7. Minimum eye height side (high/low).
8. Pass/Fail criteria.

Item 1 is code of silicon unit to identify each one, the items 2 to 7 are obtained from the Eye Margins so they would have a big correlation with EH and EW that could cause multicollinearity issues and redundancy of information for our estimation, and the item 8 is not required parameter for regression models. This would leave (5.1) like $X = \vec{x}_i \in \Re^{158}$ for $i = 1,2,3,\dots,1275$ samples.

For the regression dataset, the sample vector is defined as:

$$\vec{x}_i = [c_a \quad c_s]^T \tag{5.2}$$

From the c_a and c_s refers to the vectors for the adaptive and the static calibration settings for the Rx circuitry.

The ML algorithm tents to estimate the eye margins given by:

$$\vec{y}_i = \left[e_{high}\ \ e_{low}\ \ e_{left}\ \ e_{right}\right]^T \tag{5.3}$$

Where e_{high}, refers to eye high, e_{low} refers to eye low, e_{left} refers to eye left and e_{right} to eye right, which is the variables of each side of Eye Margin on the cross-hair gathered.

5.1.1 Dataset

The ML model for regression is intended to estimate the eye margins, that are represented with each sample of $\vec{x}_i$ depending on the calibrations of the data on X dataset and for each sample to reach a set of labels of $Y = \vec{y}_i$ for $i = 1,2,\ldots,1275$. This would lead to a paired set of parameters and margins for training and testing proposes of regression ML model.

$$\{X, Y\} \tag{5.4}$$

5.2 Results

As mentioned before the space from the dataset is mapped as $Y = f(X)$ and on (5.2) defines input features for p predictors and (5.3) are the output data where each $\vec{y}_i$ is the desired output for each side of the eye margins: Low, High, Left, and Right.

The performance of the models is measured with the coefficient of determination R^2 and the Mean Squared Error (MSE), the decision of which model was used is based on best-case over 5 iterations of training of the algorithm using the technique of the k-fold to split data into batches [24].

5.2.1 Linear Regression

The experiment for this ML algorithm was performed using the python library “OLS” from “statsmodels.formula.api” to build the OLS for regression. The OLS for regression allows only one regression per model trained, so there were trained four models for regressors $\vec{y}_i$.

Figure 5.1 shows the results for the OLS. The estimation for each side of the eye margin and the real test data as the “fit line”. Also, a red line displays the HMME for that BP, as a reference to

observe if the estimation is above or below the expected margin, those estimated values over y axis, when $x = 0$ are real zero margin values in the test data that the algorithm was not fully capable to estimate and it also can be observed a lot of variation from fit line to the estimation values.

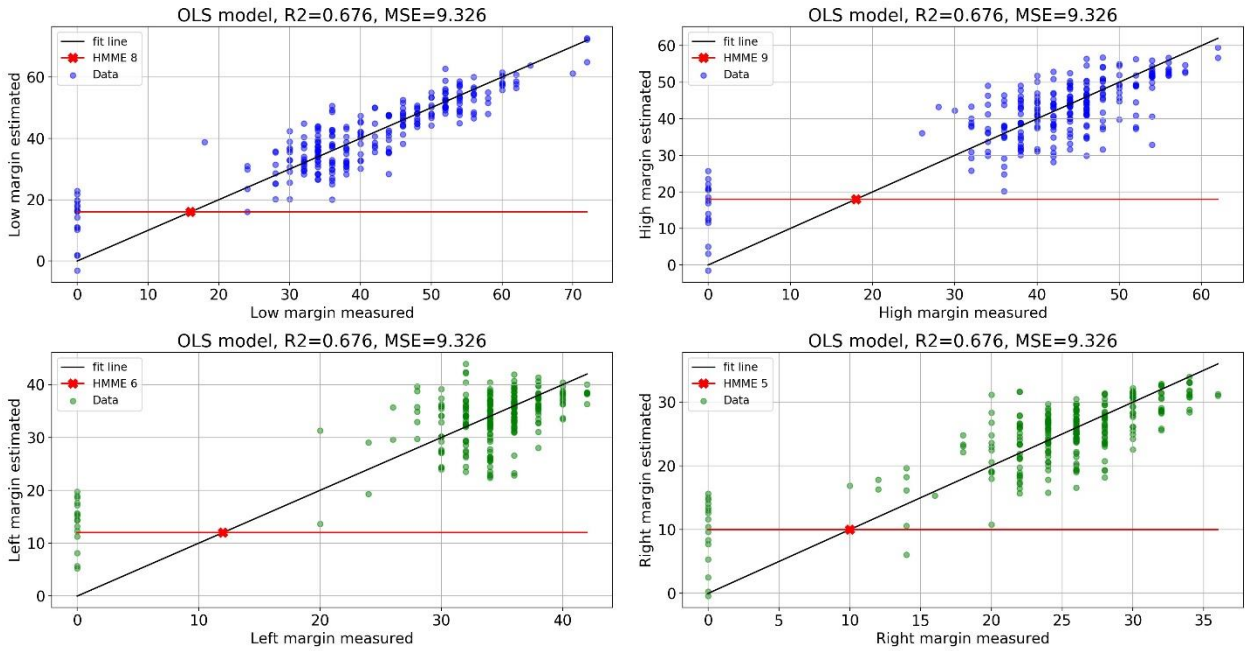

Figure 5.1. OLS estimations for the low, high, left, and right side of eye margins.

The overall summary evaluation of the linear regression with OLS is in Table 5.1.

5.2.2 Support Vector Regression

The experiment for this ML algorithm was performed using the python library “SVR” from “sklearn.svm”, the SVM for regression allows only one regression per model trained, but to solve that, a library called “MultiOutputRegressor” from “sklearn.multioutput” was used to join them in an emulation of a single model with four regressors $\vec{y}_i$.

To build the SMV for regression for different combinations of kernel functions Linear, Radial Basis, and Polynomial, as observed in Table 4.1. Next, it can be observed models for each kernel configuration that were tested:

- Model 2 – Linear Basis
- Model 3 – Radial Basis
- Model 4 – Polynomial
  - $d = 3, c = 1$
- Model 5 – Polynomial

    - $d = 4, c = 2$
- Model 6 – Polynomial
    - $d = 5, c = 3$

Where d is the "degree" and c is the "constant" of the "polynomial" for the Kernel function. Figure 5.2 and Figure 5.3, show the results for the SVR for the best models using this technique, based on Table 5.1 results. The estimation for each side of the eye margin and the real test data as the "fit line". Also, a red line displays the HMME for that BP, as a reference to observe if the estimation is above or below the expected margin, those estimated values over y axis, when $x = 0$ are real zero margin values in the test data that the algorithm was not fully capable to estimate.

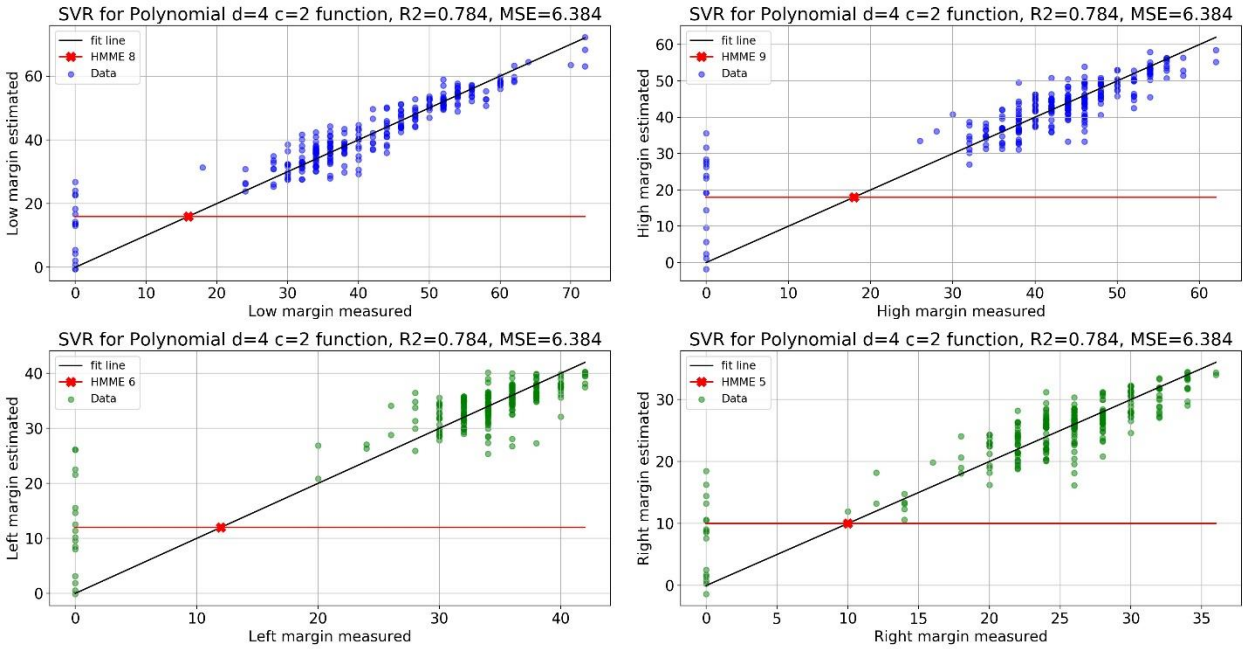

Figure 5.2. SVR estimations for low, high, left and right side of eye margins using the polynomial kernel with $d = 4, c = 2$

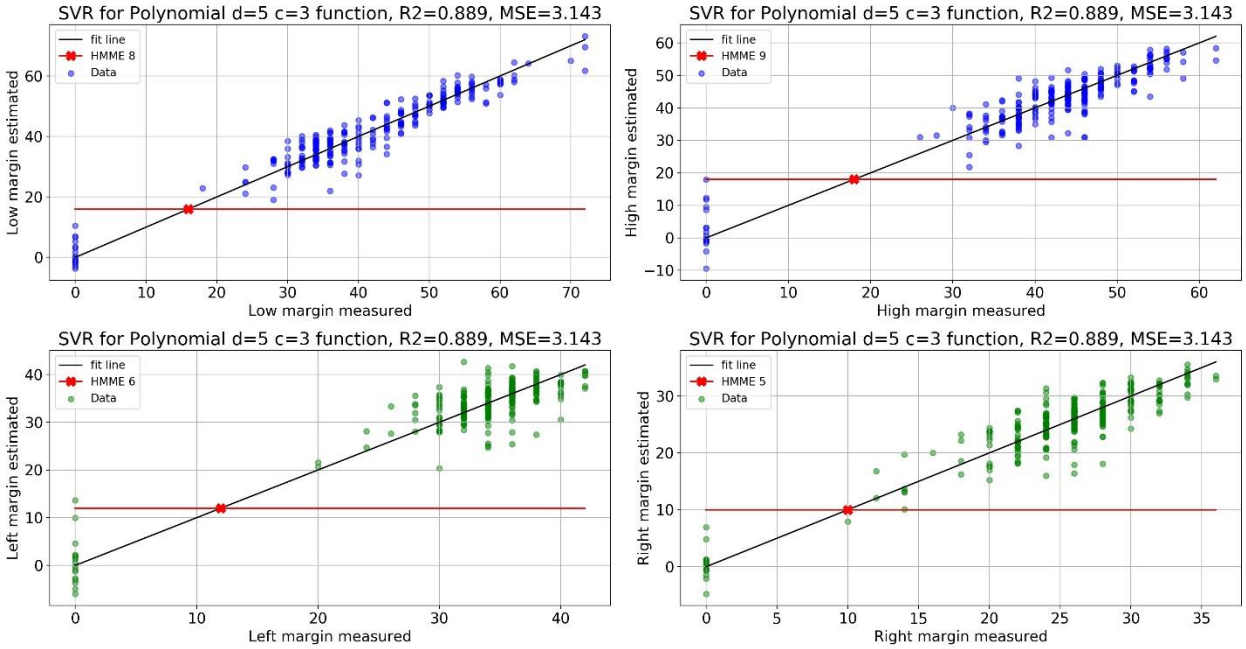

Figure 5.3. SVR estimations for low, high, left and right side of eye margins using the polynomial kernel with $d = 5, c = 3$.

To provide a better view of the capability of the SVR to estimate the eye margins sides, Figure 5.4 and Figure 5.5 show the margin distribution for testing and estimated data, clearly can be observed that the distributions behave similarly.

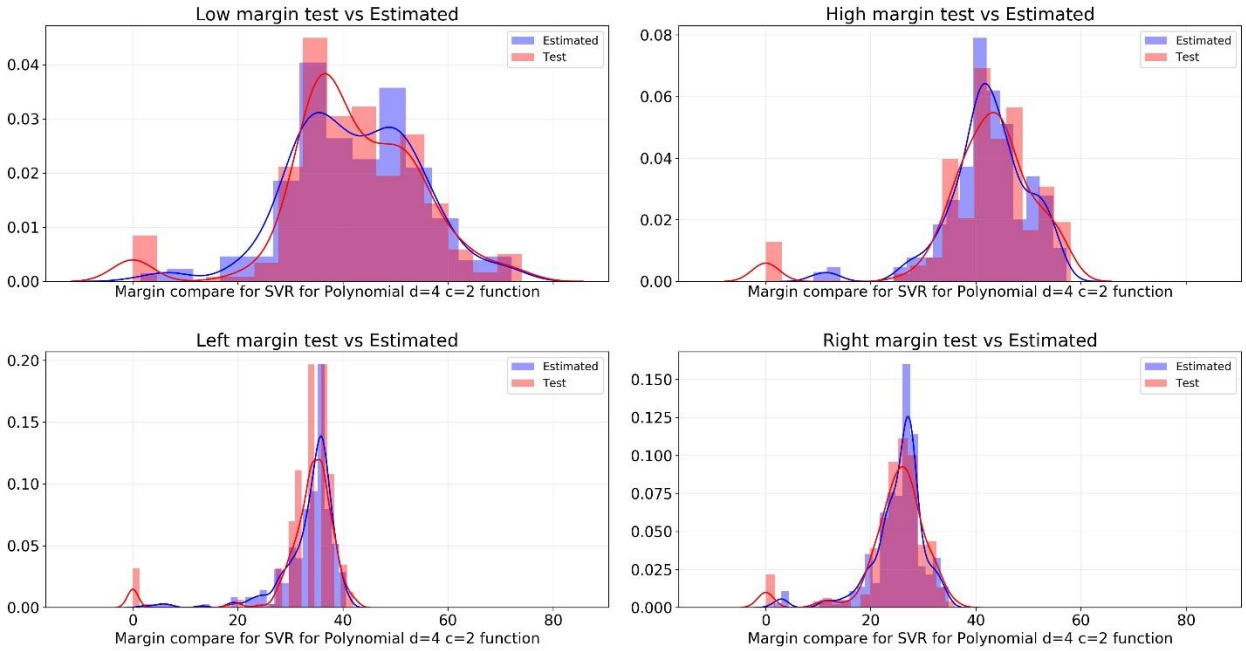

Figure 5.4. SVR kernel polynomial ($d = 4, c = 2$) comparing low, high, left and right side of eye margins for testing data versus estimated data.

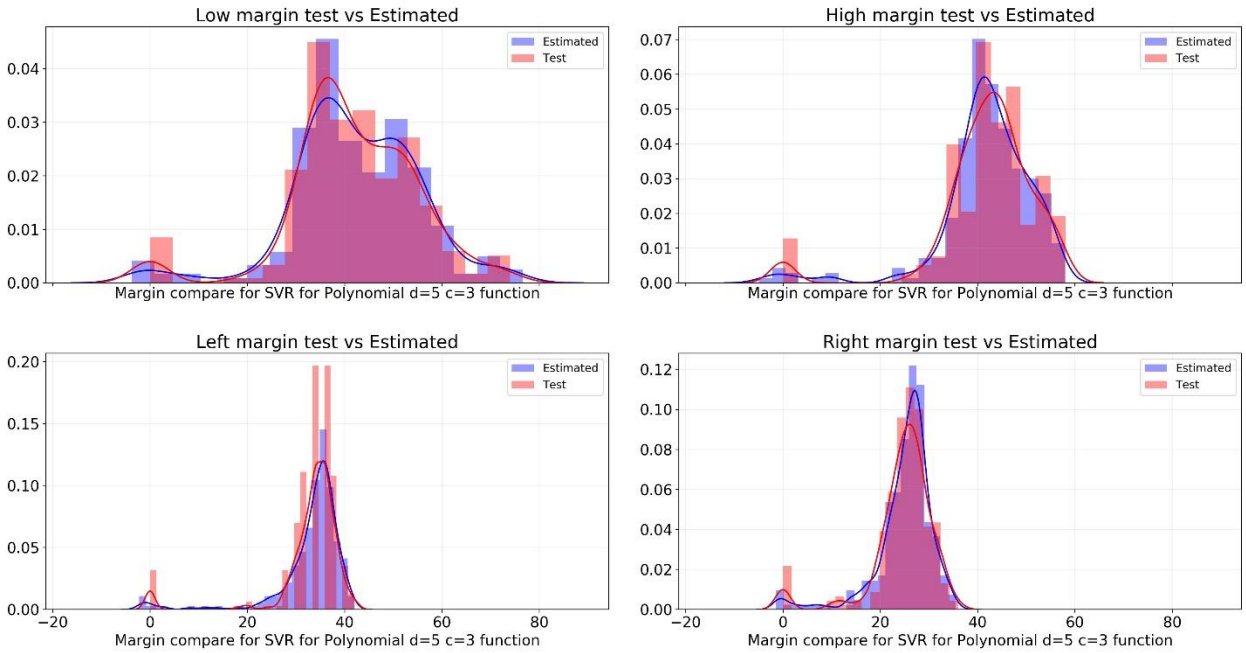

Figure 5.5. SVR kernel polynomial ($d = 5, c = 3$) comparing low, high, left and right side of eye margins for testing data versus estimated data.

The overall summary evaluation of SVR models is in Table 5.1.

5.2.3 Artificial Neural Network

To build the ANN, it was used the python library “Sequence” from “keras.models” for the frame of the ANN, the library “Dense” from “keras.layers” to add the density of neurons and activation function for each layer of the ANN, and the Stochastic Gradient Descent function from “SGD” library from “keras.optimizers” with a loss measured with minimum squared error configuration. The structure of the ANN is:

- Model 7
  - Input layer:
    - Activation function: *hyperbolic tangent*
    - Quantity of neurons: 157
  - Hidden layer:
    - Activation function: *Sigmoid*
    - Quantity of neurons: 48×48
  - Output layer:
    - Activation function: *Linear*
    - Quantity of neurons: 4
- Model 8
  - Input layer:
    - Activation function: *Sigmoid*
    - Quantity of neurons: 157

- Hidden layer:
  - Activation function: *hyperbolic tangent*
  - Quantity of neurons: 48×48
- Output layer:
  - Activation function: *Linear*
  - Quantity of neurons: 4

The hidden layer quantity of neurons is defined using references from [30] and the ANN was trained when it reached the 100 epochs.

Figure 5.6 and Figure 5.7 show the results for the ANN. The estimation for each side of the eye margin and the real test data as the "fit line". Also, a red line displays the HMME for that BP as a reference to observe if the estimation is above or below the expected margin, the estimated values over de y axes, when $x = 0$ are real zero margin values in the test data that the algorithm was not fully capable to estimate.

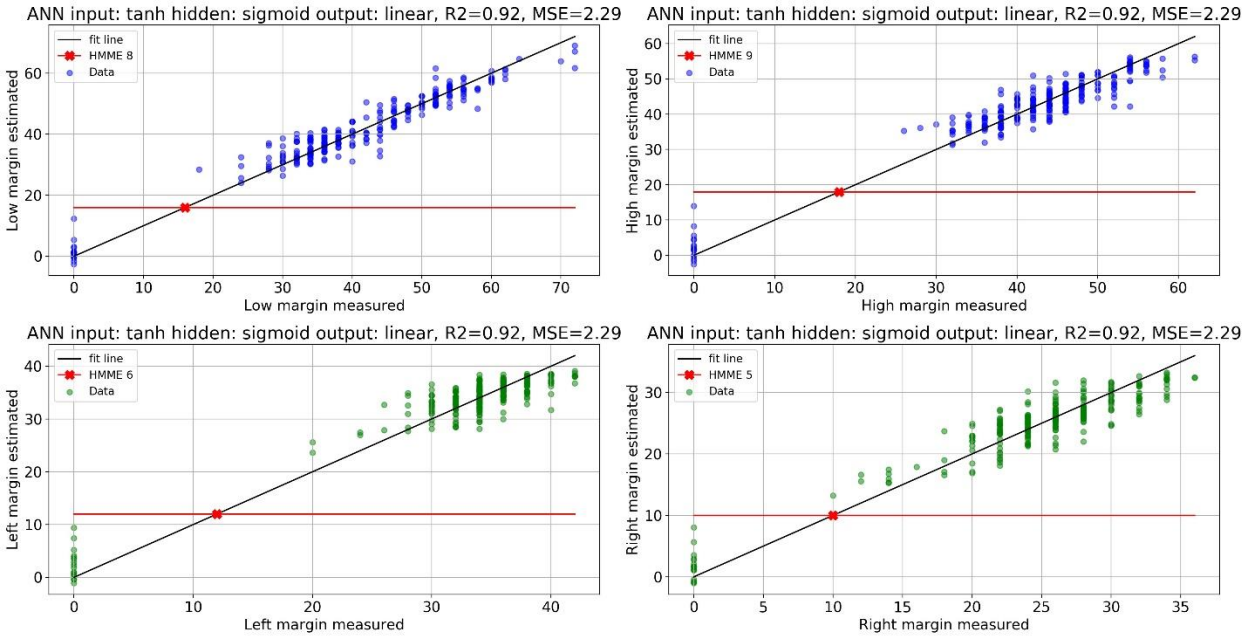

Figure 5.6. ANN estimations for low, high, left, and right side of eye margins for Model 7.

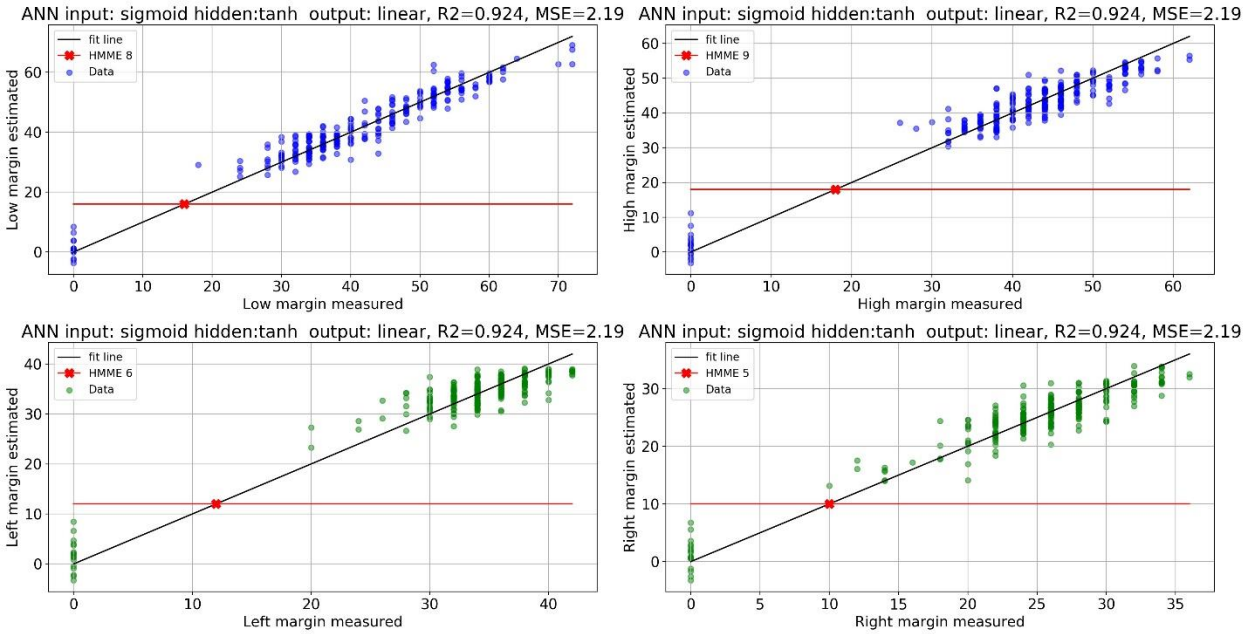

Figure 5.7. ANN estimations for low, high, left, and right side of eye margins for Model 8.

To provide a better view of the capability of the ANN to estimate the eye margins sides, Figure 5.8 and Figure 5.9 show the margin distribution for testing and estimated data, clearly can be observed that the distributions behave similarly, and this is due to the observed performance on Table 5.1 for the both ANN models tested is very similar.

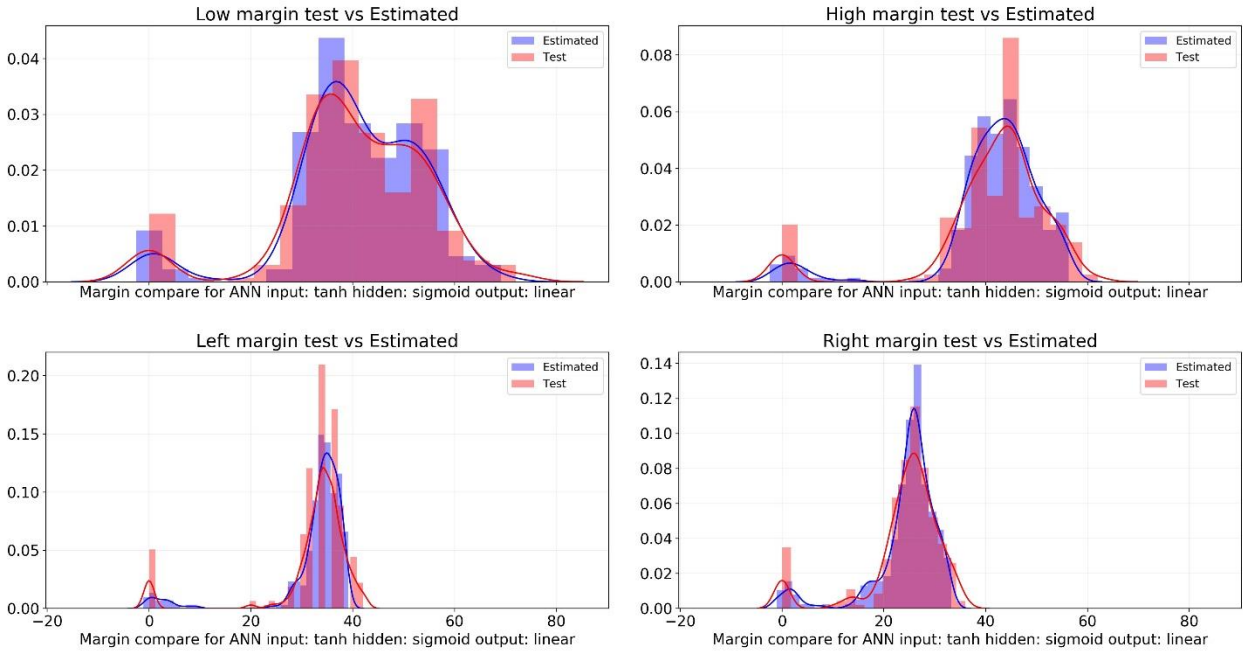

Figure 5.8. ANN comparing low, high, left, and right side of eye margins for testing data versus estimated data for model 7.

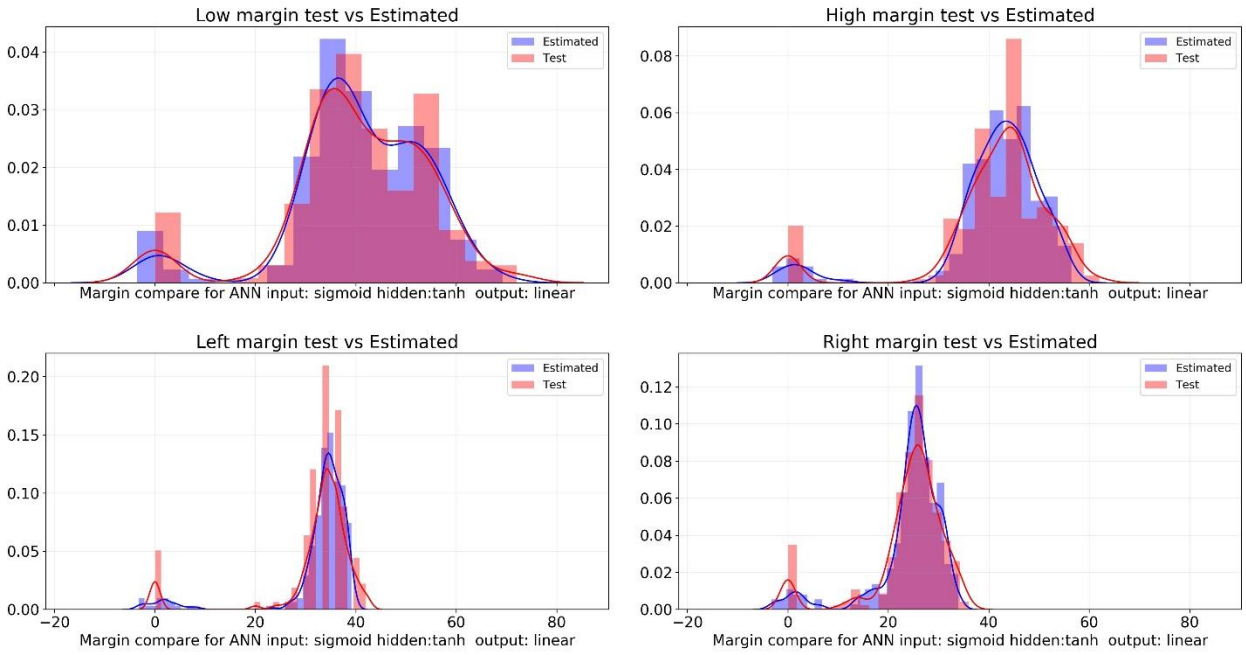

Figure 5.9. ANN comparing low, high, left and right side of eye margins for testing data versus estimated data for model 8.

The overall summary evaluation of ANN models is in Table 5.1.

5.2.4 Decision Tree

To build the Decision Tree using Random Forest configuration, it was used the python library "RandomForestRegressor" from "sklearn.ensemble", the structure of the Decision Tree is:

- Number of estimators (trees): 100
- Evaluation criteria: Mean Square Error (MSE)
- Maximum tree depth: "None" (expand until less minimum sample leaf)
- Minimum sample splits: 2
- Minimum sample leaf: 1
- Maximum features: Total of features
- Bootstrap (select random features per training tree): Enabled

Figure 5.10 and Figure 5.11 show the results for the ANN. The estimation for each side of the eye margin and the real test data as the "fit line". Also, a red line displays the HMME for that BP as a reference to observe if the estimation is above or below the expected margin, the estimated values over y axes, when $x = 0$ are real zero margin values in the test data that the algorithm was not fully capable to estimate.

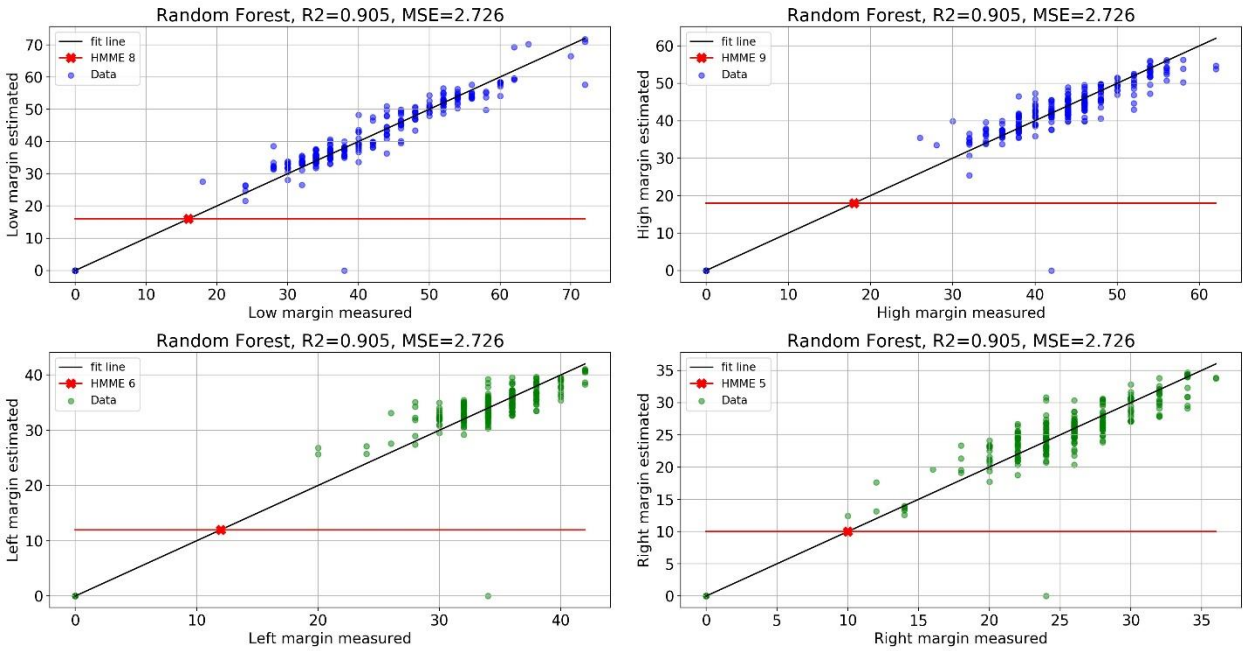

Figure 5.10. Decision Tree – Random Forest estimations for low, high, left, and right side of eye margins.

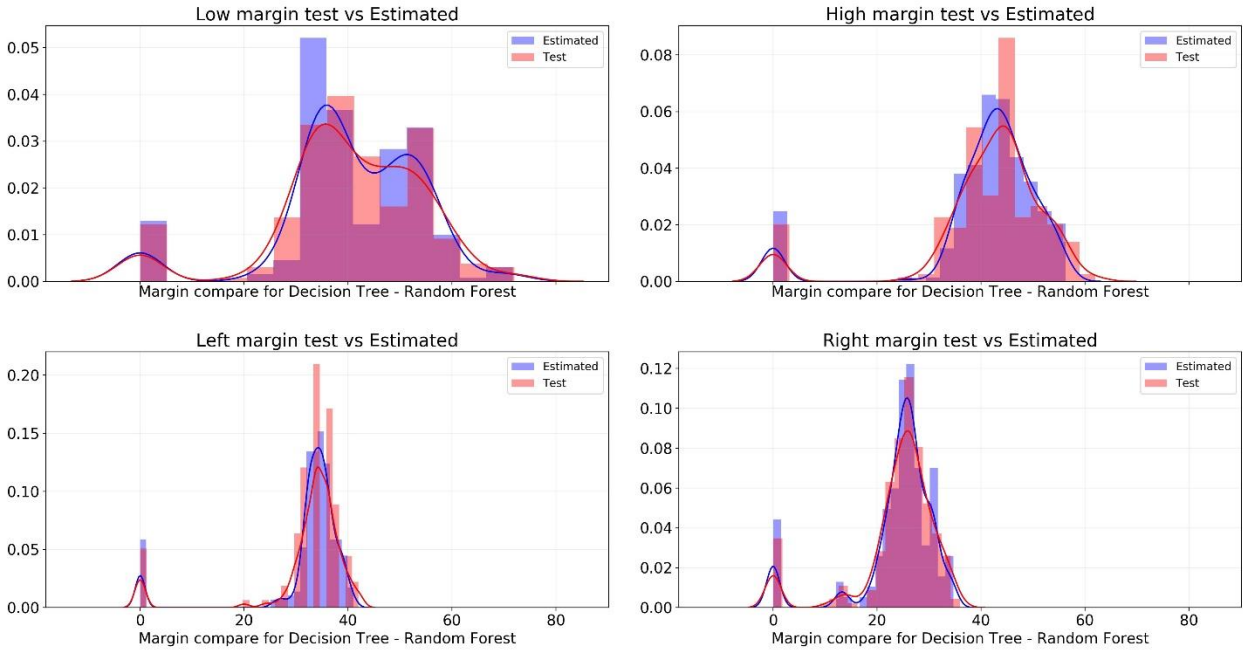

Figure 5.11. Decision Tree with Random Forest comparing low, high, left, and right side of eye margins for testing data versus estimated data.

The overall summary evaluation of the Decision Tree model is in Table 5.1.

5.3 Models comparison

Summary of the results and comparison across models in

Model	Type	Configuration	R^2	MSE
1	OLS	N/A	0.676	9.326
2	SVR	Linear	0.544	12.782
3	SVR	Radial basis	0.47	15.712
4	SVR	Polynomial $degree = 3, constant = 1$	0.548	13.295
5	SVR	Polynomial $degree = 4, constant = 2$	0.784	6.384
6	SVR	Polynomial $degree = 5, constant = 3$	0.889	3.143
7	NN	*Input layer*: Hyperbolic Tangent *Hidden layer*: Sigmoidal *Output layer*: Linear	0.92	2.29
8	NN	*Input layer*: Sigmoidal *Hidden layer*: Hyperbolic Tangent *Output layer*: Linear	0.924	2.19
9	Decision Tree	Random Forest	0.905	2.726

Table 5.1. R^2 and MSE for each regression model.

5.4 Chapter conclusions

As observed in Table 5.1 some algorithms showed a better response to estimate Eye Margins, on the images from chapter results, we can observe the fit line that represents the true data and the dots that represent the prediction or the estimation.

For regression with the OLS model in Figure 5.1 the model can perform an estimation of the eye margins but it does not have a very good performance, since the R^2 is low and the MSE is very high.

For SVR from Table 5.1 that models 2 to 4 were not good to perform Eye Margin estimations, but on the other hand models 5 and 6 had a better performance based on R^2 and MSE and also the data estimation dots around the fit line in Figure 5.2 and Figure 5.3 is better than the linear regression with OLS.

For ANN models from Table 5.1 that models 7 and 8 were good to perform Eye Margin estimations based on their R^2 and MSE and also the data estimation dots around the fit line in Figure 5.6 and Figure 5.7 are better than the linear regression with OLS and the SVRs.

For Decision Tree with Random Forest from Table 5.1, that model 9 was good to perform Eye Margin estimations based on its R^2 and MSE and also the data estimation dots around the fit line in Figure 5.10 is better than the linear regression with OLS and the SVRs but comparable with the ANN.

6 Conclusions and Future Work

The contribution of this work consists in the significant reduction of the burden on validation engineers to make an SMV DP risk assessment decision, which reduces considerably the bulk of resources for execution time, effort, and costs

6.1 Classification models

This part consists in predict the pass/fail of the eye margin measurement supported with calibration. Second to estimate the possible eye margin based on the calibrations measured [2].

The classification ML algorithm in this work looks to replace the risk assessment decision, observed in Figure 6.1.

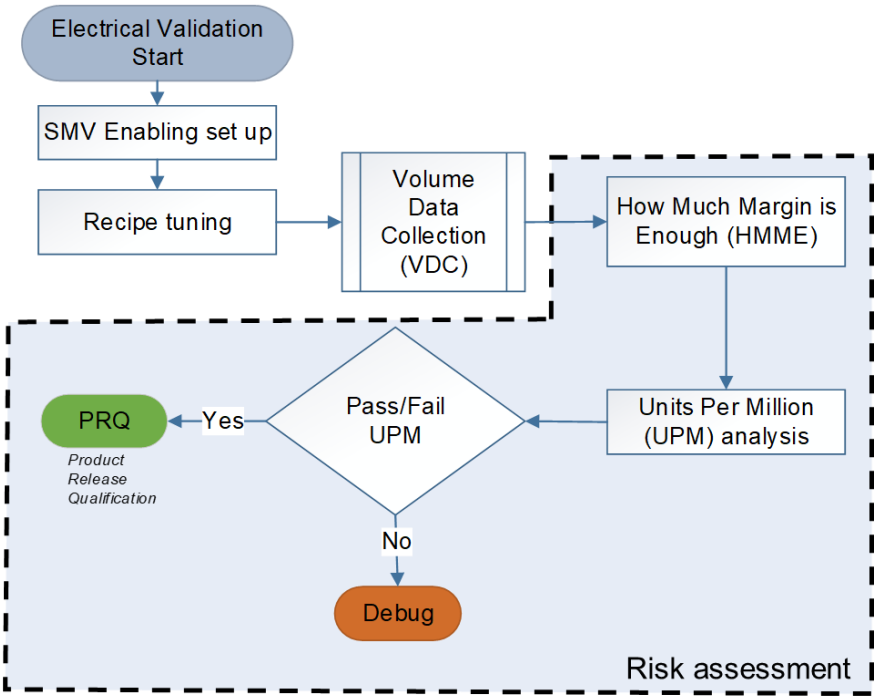

Figure 6.1. The "Risk assessment" inside the dashed line would be substitute by the ML algorithm in a way that it can perform a classification for the risk assessment PRQ decision by determining if measurement pass or fail, see (3.3).

To estimate the gains when classification ML is applied, this activity can be defined as:

$$w_{smv} = w_{en} + w_{tuning} + w_{VDC} + w_{HMME} + w_{UPM} \quad (6.1)$$

Where w_{smv} are the total of weeks required for the SMV validation process, then w_{en} are the weeks required for enabling, w_{tuning} are the weeks required for tuning, w_{VDC} are the weeks required for volume data collection, $w_{min\,margin}$ are the weeks required to perform the experiments for the HMME and w_{UPM} is the number of weeks required for the UPM analysis to provide the decision,

If the classification ML algorithm replaces Inside of the shadowed area observed in Figure 6.1. It would make the w_{HMME} and w_{UPM} approach to virtually zero, this would re-define the (6.1) to:

$$if: w_{HMME} \to 0, w_{UPM} \to 0 \therefore w_{smv} = w_{en} + w_{tuning} + w_{VDC} \quad (6.2)$$

E.g. let's consider SMV test cycle should achieve the *Risk assessment decision*, in eight weeks, the task would be divided in: $w_{en} = 2, w_{tuning} = 2, w_{VDC} = 1, w_{HMME} = 2, w_{UPM} = 1.$

Substitute in (6.1) we would have: $w_{smv} = 8\,weeks$

Substitute in (6.2) the reduction time would be: $w_{smv} = 5\,weeks$

When a risk assessment decision is provided approximately 37% faster than the regular process, This can be translated in cost, resources, and work time-saving for a company, with the benefit that if a potential assessment is considered as "Fail" the debug teams would have more time to root cause and solve the issue with the possibility to deliver the product on time avoiding delays that can cause losses on the company.

6.2 Regression models

This part is to estimate the possible eye margin based on the calibrations measured.

The regression ML algorithm in this work looks to replace the "collect eye margins", shadowed in Figure 6.2, which is a block that reaches the edges of the eye (check Figure 1.5 for reference).

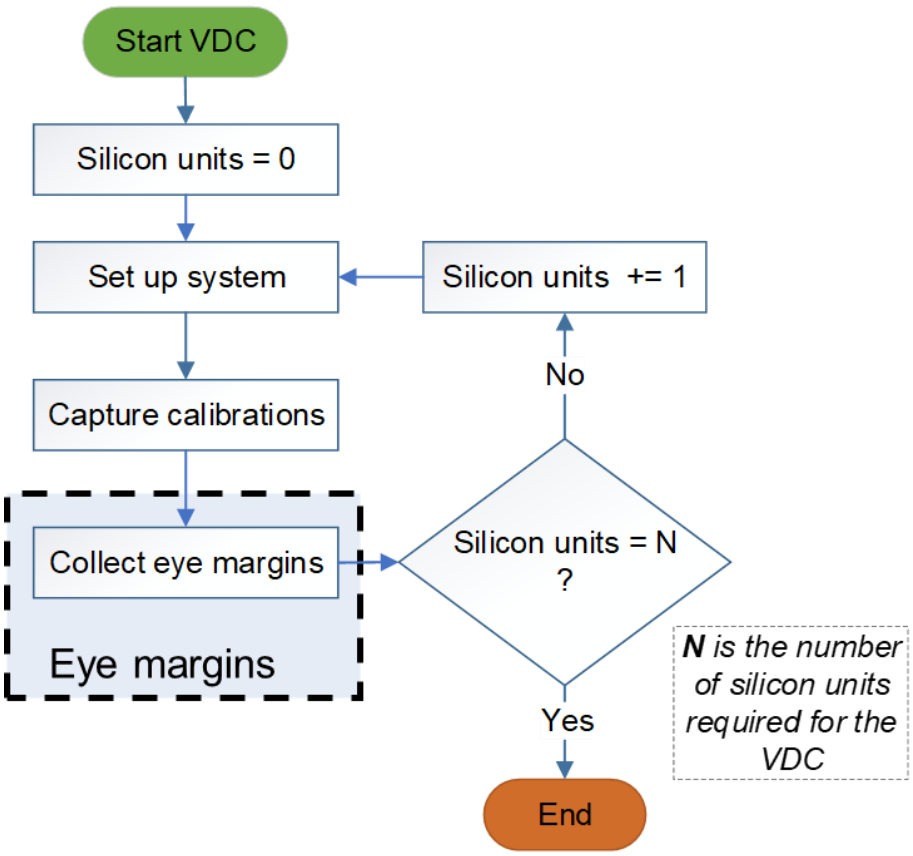

Figure 6.2 Sub-process for the VDC and shadowed the Eye margins *that contains the block that regression ML algorithm would replace by estimating the eye margins.*

The shadowed area (Eye margins) consumes a considerable amount of time to collect the eye margin measurements for each silicon unit, approximately 60% of the time of the VDC flow. (6.3) shows the approximate time consumed for one iteration of the flow in Figure 6.2.

$$m_{eye} = t_{set\ up} + t_{calibrations} + t_{eye\ margins} \tag{6.3}$$

Where m_{eye} is the time in minutes that takes the capture of the eye margining, $t_{set\ up}$, $t_{calibrations}$ and $t_{eye\ margins}$ is the times required per block in Figure 6.2 to collect the measurements from the Eye Margin.

To know the total time required for the volume data collection, the (6.4) defines:

$$t_{VDC} = \frac{(m_{eye} \times r \times v)}{60min} \tag{6.4}$$

Where t_{VDC} is the total time for the VDC, r represent the number repetition of the measurement repetitions per unit (this is to consider variability per measurement), and v as the volume of units considered for testing.

With the regression ML algorithm, the idea is: that $t_{eye\ margins}$ approaches to zero, and since it represents the 60% of the time in (6.3) can be re-defined to:

$$if\ t_{eye\ margins} \to 0 \therefore m_{eye} = t_{set\ up} + t_{calibrations} \quad (6.5)$$

E.g. Consider $t_{set\ up} = 5min, t_{calibrations} = 2min, t_{eye\ margins} = 5min$ all in minutes, $r = 5$ repeats, and $v = 30$ units.

- Substitute variables in (6.3) and (6.4): $m_{eye} \approx 12min \therefore t_{VDC} \approx 30\text{hrs}$
- Substitute variables in (6.5) and (6.4): $m_{eye} \approx 7min \therefore t_{VDC} \approx 17.5\text{hrs}$

Considering the shifts for a worker on average eight hours per day and five times a week this would be transformed to:

- $Standard\ validation\ t_{VDC}\ weeks \approx 1$
- $With\ ML\ algorithm\ t_{VDC}\ weeks\ \approx 0.31$

The reduction of time when the margins are estimated or generated synthetically based on measured calibration values, benefits on the burden efforts that EV engineers should dedicate to this task that would drive into cost reduction and support the risk assessment to deliver with time ahead of the decision for PRQ.

6.3 Future work

6.3.1 Models implementation on for validation

If both models, classification and estimation are combined to be used on a DP, the time to provide a risk assessment decision would be calculated replacing from (6.2) and reducing the w_{VDC} from (6.5). Following the *e.g.* for eight weeks for full validation, the reduction will be:

$$if: w_{VDC} \to 0.31, w_{en} = 2, w_{tuning} = 2\ \therefore\ w_{smv} = w_{en} + w_{tuning} + w_{VDC} \approx 4.31$$

The envisioning would be having ML models on a repository that can be trained once the data has been gathered from the BP and then store those trained models on the repository [2], as observed in Figure 6.3.

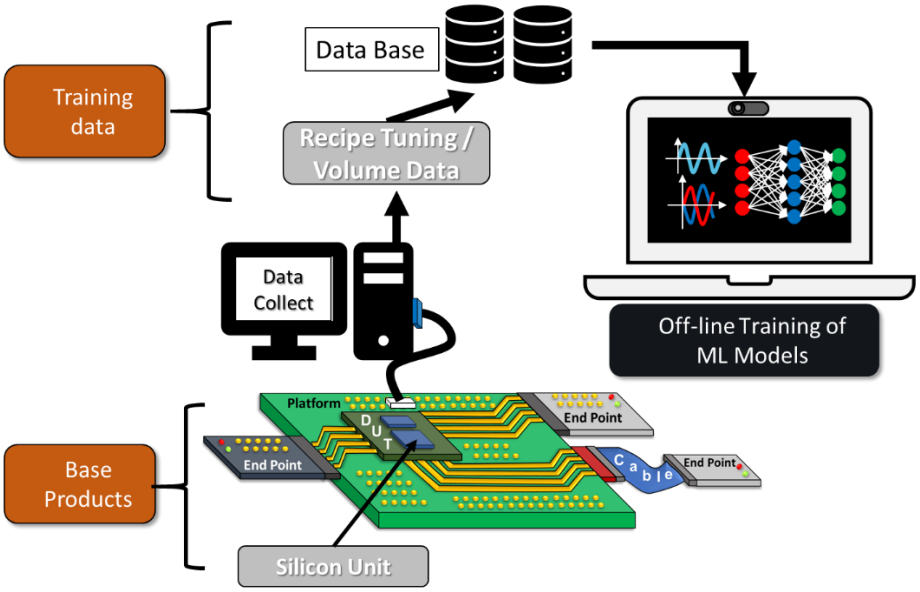

Figure 6.3. Data collection to Train the ML learning model.

Then the engineers that would test the same HSIO interface technology on a DP would use the data collected to feed the ML model trained to compare the data and determine if the product validated that re-use technology is accomplish the same quality standards as its predecessor (BP) [2], as observed in Figure 6.4, in a shorter amount on time.

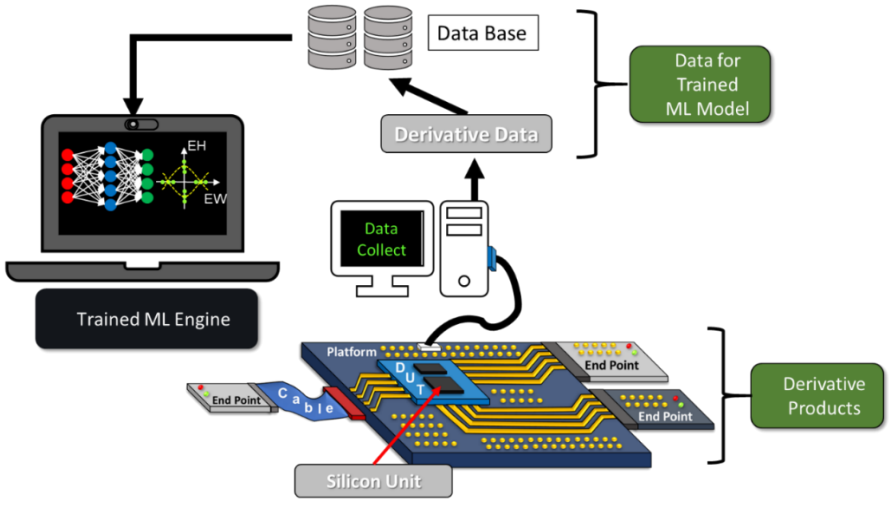

Figure 6.4. Trained Engine to test DP

As it was observed the ML model to train and the trained ML engine would be stored in a repository where offline engineers can consult them to train a new model or to use the current existent to test new DP.

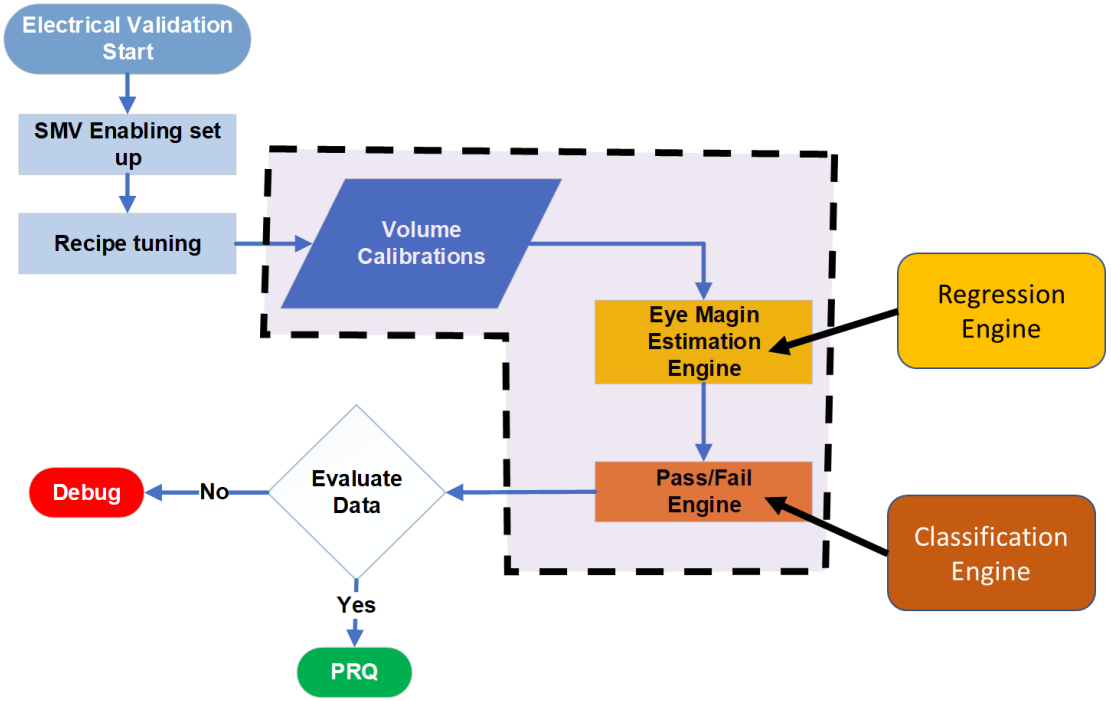

Figure 6.5. Representation of how both engines would be integrated for derivative products validation.

This work can have other additions that would be helpful to increment the automation level and improve the usage of the Classification and Regression models observed in this work. The next two sections would show:

- Automatic calculator of units for DP.
- Auto classifier for Pass/Fail criteria.

6.3.2 Automatic calculator of Derivative Units

Some good support for the Regression and Classification models observed would be adding an algorithm that helps to determine the minimum silicon units required to add into a trained ML engine, in other words, how many units would the algorithm require to fit and determine the sanity check, this would directly impact the time required on de VDC part, since if the quantity decrease, it directly saves time that as it has been observed is cost saving for the company for the validation teams.

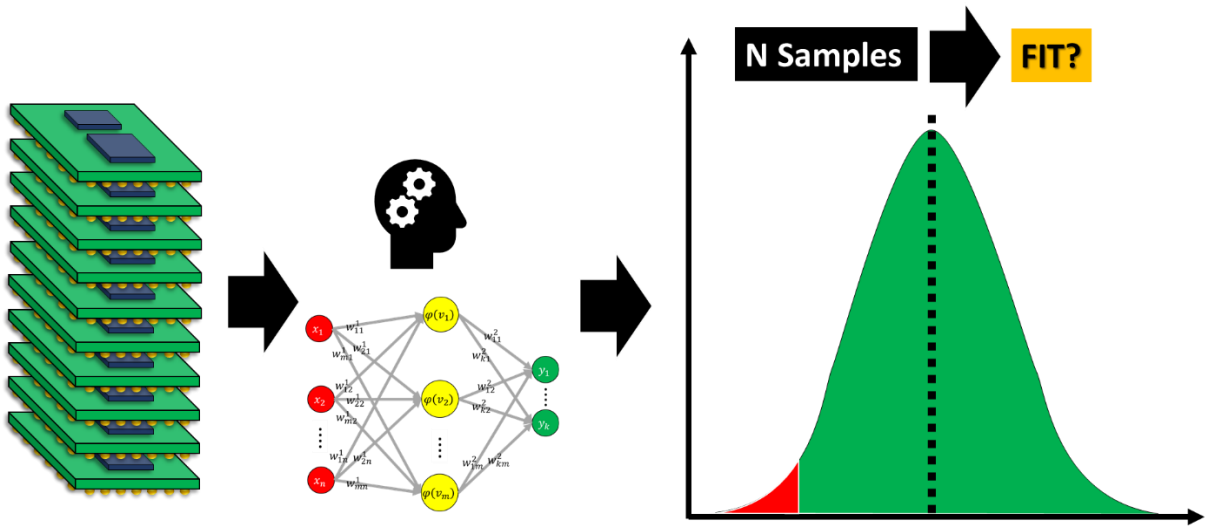

Figure 6.6. A vision of an automatic calculator for the required quantity of derivative units to fit the model.

6.3.3 Auto classifier for Pass/Fail criteria

This is directly for support the ML model for classification since as it was mentioned, the dataset used to train those ML algorithms was classified manually by three engineers adding the Pass/Fail criteria, see (3.4).

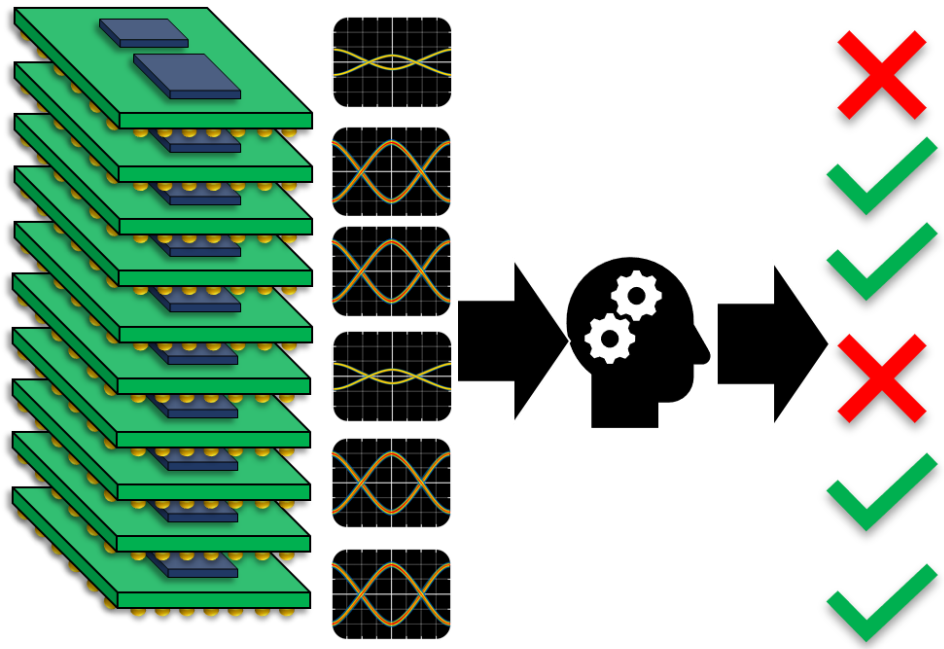

Figure 6.7. A vision of the automatic Pass/Fail margin classifier.

References

[1] H. Rotithor, “Postsilicon validation methodology for microprocessors”, IEEE Design & Test of Computers, vol. 17, Issue 4, 2000, pp. 77–88.

[2] C. A. Sanchez-Martinez, P. López-Meyer, E. Juarez-Hernandez, A. Desiga-Orenday, and A. Viveros-Wacher, “High Speed Serial Links Risk Assessment in Industrial Post-Silicon Validation Exploiting Machine Learning Techniques”, *International Test Conference (ITC), Interconnect Testing & Test Access*, (to be published).

[3] L.-C. Wang, “Regression simulation: applying path-based learning in delay test and post-silicon validation,” in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, Mar. 2004.

[4] C. Zhuo, B. Yu, D. Gao, “Accelerating chip design with machine learning: From pre-silicon to post-silicon,” in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, Munich, Germany, Dec. 2017.

[5] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, A. Beghi, “Machine Learning for Predictive Maintenance: A Multiple Classifier Approach,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 812-820, June 2015.

[6] F. E. Rangel-Patiño, A. Viveros-Wacher, J. E. Rayas-Sánchez, I. Durón-Rosales, E. A. Vega-Ochoa, N. Hakim and E. López-Miralrio, “A holistic formulation for system margining and jitter tolerance optimization in industrial post-silicon validation,” *IEEE Trans. Emerging Topics Computing*, vol. **, no. **, pp. **, *** 2017. (published online: 29 Sep. 2017; DOI: 10.1109/TETC.2017.2757937).

[7] S. Deyati, B. J. Muldrey, A. Banerjee, A. Chatterjee, “Atomic model learning: A machine learning paradigm for post silicon debug of RF/analog circuits,” in *2014 IEEE 32nd VLSI Test Symposium (VTS)*, Napa, USA, May, 2014.

[8] N. Christianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge, UK: Cambridge University Press, 2000.

[9] Intel Corp., “Intel platform and component validation - a commitment to quality, reliability and compatibility”, White Paper, Intel Corporation, 2003.

[10] J. Keshava, N. Hakim, C. Prudvi, “Post-silicon validation challenges: How EDA and academia can help,” Design Automation Conference (DAC), 47th ACM/IEEE, June 2010, pp. 3 –7.

[11] L. Shkolnitsky and E. Einy, “Electrical system-validation methodology for embedded display port,” White Paper, Intel Corporation, June 2010. [Online] Available: https://doc.diytrade.com/docdvr/1140579/32527293/1368382761.pdf

[12] A. Crouch, A. Ley and A. Sguigna. *System Marginality Validation of DDR3|DDR4 Memory and Serial I/O*. ASSET InterTech, Inc, 2014.

[13] PCI SIG Org. (2016), Peripheral Component Interconnect Express 3.1 Specification [Online]. Available: https://pcisig.com/specifications.

[14] SATA Org. (2016), Serial Advanced Technology Attachment 3.2 Specification [Online]. Available: http://www.sata-io.org/

[15] USB Org. (2016). Universal Serial Bus Revision 3.1 Specification [Online]. Available: http://www.usb.org/developers/doc

[16]10GEA Org. (2016), XAUI interface [Online]. Available: http://www.10gea.org/whitepapers/xaui-interface/

[17] UFSA Org. (2019), Universal Flash Storage Association [Online]. Available: https://ufsa.org/

[18] MIPI Org. Available https://www.mipi.org/

[19] F. E. Rangel-Patiño, J. E. Rayas-Sánchez, A. Viveros-Wacher, J. L. Chávez-Hurtado, E. A. Vega-Ochoa, and N. Hakim, “Post-silicon receiver equalization metamodeling by artificial neural networks,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 733-740, April 2019.

[20] D. Hong, K.T. Cheng, “Bit-error rate estimation for bang-bang clock and data recovery circuit,” in *26th IEEE VLSI Test Symp.*, San Diego, CA, April, 2008, pp. 17 – 22.

[21] A. Viveros-Wacher, R. Alejos, L. Alvarez, I. Diaz-Castro, B. Marcial, G. Motola-Acuna, and E. A. Vega-Ochoa, “SMV methodology enhancements for high speed IO links of SoCs,” in *IEEE VLSI Test Symp. (VTS-2014)*, Napa, CA, May. 2014, pp. 1-5.

[22] A. Dobson and A. Barnett, An Introduction To Generalized Linear Models. Boca Raton: CRC Press, Taylor & Francis Group, 2018.

[23] C.M. Bishop, M. Jordan, J. Kleinberg and B. Schölkopf, *Pattern Recognition and Machine Learning*, New York, NY 10013, USA, Springer Science+Business Media, LLC, 2006
[24] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for machine learning*, Cambridge, United Kingdom: Cambridge University Press, 2020.
[25] W.S. McCulloch & W. Pitts, *A logical calculus of the ideas immanent in nervous activity. Bull of Math. Biophysics, 5,* 1943, 115–133.
[26] R. Rosenblatt, *Principles of Neurodynamics*, New York: Spartan Books, 1962.
[27] D.E. Rumelhart, G.E. Hinton, & R.J. Williams, *Learning internal representations by error propagation*. In: D. E. Rumelhart, J.L. McClelland, Eds., Parallel distributed processing: Explorations in the microstructure of cognition, 1: Foundation, Cambridge: MIT Press, 1986, 318–362.
[28] K.-L. Du and M. N. S. Swamy, *Neural Networks and Statistical Learning*. London: Springer London, 2014.
[29] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*, New York, NY: Springer, 2008.
[30] K. Gnana Sheela, S. N. Deepa, "Review on Methods to Fix Number of Hidden Neurons in Neural Network", *Volume 2013, Article ID 425740,* https://doi.org/10.1155/2013/425740, Jun 2013
[31] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. New York: Wiley, 2001.