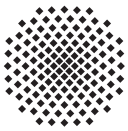


Universität Stuttgart

Institute of Computer Architecture and Computer Engineering

University of Stuttgart
Pfaffenwaldring 47
D–70569 Stuttgart

Master's Thesis Nr. 3580

Machine Learning Methods for Fault Classification

Siddharth Sunil Gosavi

Course of Study:	InfoTECH
Examiner:	Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich
Supervisors:	Dipl.-Inf. Laura Rodríguez Gómez M.Sc. Alejandro Cook
Commenced:	October 23, 2013
Completed:	April 24, 2014
CR-Classification:	B8.1, I2.1, I2.6

Acknowledgments

First and the foremost, I would like to express my sincere gratitude towards Prof. Dr. Hans-Joachim Wunderlich for providing me with the opportunity to contribute towards the research in fault classification techniques, with this thesis. His guidance and suggestions helped me throughout the research and writing of this thesis. I would like to thank my thesis supervisors - Ms. Laura Rodríguez Gómez and Mr. Alejandro Cook, for their guidance, patience and the motivation they provided all along, to meet the goals of the thesis. I cannot imagine better supervisors for my thesis. Without them, going through the thesis would not have been possible.

I am also thankful to all of the staff members of the institute for making my thesis at ITI an enjoyable experience.

My sincere thanks goes to all of my friends, in Germany, back home and abroad, for their constant encouragement and love.

Finally, I would like to thank my parents, Sou. Sheela Gosavi and Shri. Sunil Gosavi, and the rest of my family for their love and support not just during my thesis work, but also throughout my life.

Abstract

With the constant evolution and ever-increasing transistor densities in semiconductor technology, error rates are on the rise. Errors that occur on semiconductor chips can be attributed to permanent, transient or intermittent faults.

Out of these errors, once permanent errors appear, they do not go away and once intermittent faults appear on chips, the probability that they will occur again is high, making these two types of faults critical. Transient faults occur very rarely, making them non-critical. Incorrect classification during manufacturing tests in case of critical faults, may result in failure of the chip during operational lifetime or decrease in product quality, whereas discarding chips with non-critical faults may result in unnecessary yield loss.

Existing mechanisms to distinguish between the fault types are mostly rule-based, and as fault types start manifesting similarly as we move to lower technology nodes, these rules become obsolete over time. Hence, rules need to be updated every time the technology is changed.

Machine learning approaches have shown that the uncertainty can be compensated with previous experience. In our case, the ambiguity of classification rules can be compensated by storing past classification decisions and *learn* from those for accurate classification.

This thesis presents an effective solution to the problem of fault classification in VLSI chips using Support Vector Machine (SVM) based machine learning techniques.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Scaling of the CMOS technology has progressed significantly, defying barriers in manufacturing processes. It has benefited mainly in terms of significant performance improvements and increased integration in the final product. However the reduction in supply voltages and an increase in operating frequencies, which have also accompanied scaling, have resulted in higher error rates [SABR04, Con07a, AHX+05].

Errors that occur on semiconductor chips can be attributed to permanent, transient or intermittent faults. Permanent faults are caused primarily by irreversible changes, like an open or a short link. Transient faults are caused by temporary environment conditions, e.g. cosmic rays. Intermittent faults are caused by an unstable or marginal state of the hardware [Con07a]. Intermittent failures can go away with time or may manifest later into a permanent breakdown, depending on the underlying defect mechanism. The only way to correct failures caused by permanent faults is to replace the offending component. Transient faults occur rarely during the device lifetime, and manifest themselves as soft errors. Some of these failures, specially failures due to transient faults can be avoided by using fault tolerance mechanisms [BS04, MZS+08]. While fault tolerant circuits can significantly reduce failure rates, these circuits themselves can be affected by one of the faults. Hence, these circuits also need to be tested for faults mentioned.

In safety critical systems all faults should be treated equally and circuits which have any of these faults should be rejected. However, a large portion of semiconductor applications are not safety critical. For those applications, permanent faults and those intermittent faults which can severely degrade functionally, can be assumed as *critical*, while all other types of faults can be assumed as *non-critical*.

During the post-production testing of these chips, faults are categorized as either faulty or working [Agr00]. No further attempt is made to check if circuits had any non-critical faults.

This approach leads to yield loss, as some of the healthy chips, which had only transient faults, are also rejected. An alternative approach, that can be explored is shown in figure 1.1. It is proposed that if the chips can be separated as working and partially working, the partially working ones can still be included in the final yield of the product, without degradation of the quality and still reducing yield loss.

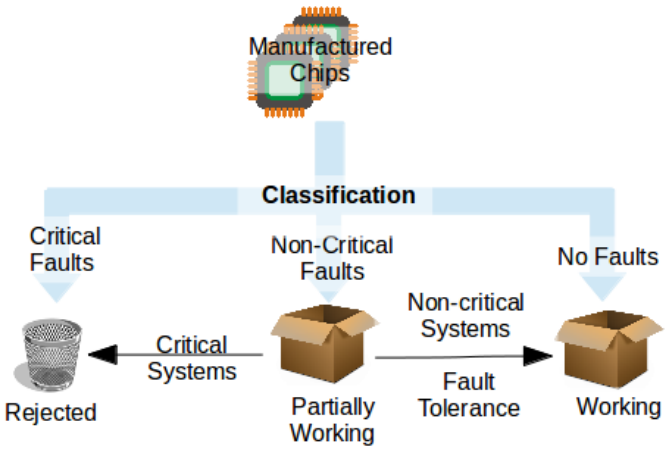


Figure 1.1: Proposed classification flow for VLSI Chips

With scaling it is expected that, non-critical failures (*i.e.* failures due to transient faults) will cause majority of the failures [Con03]. Hence significant yield loss can be avoided by using the method proposed above. This can be achieved if we are able to classify faults as permanent, transient and intermittent, then reject circuits with permanent and intermittent faults, while retaining chips affected only by transient faults. Such classification can also help designers with the statistics about failures, which can be useful to find the underlying defect mechanism, resulting in further improvement of the product quality.

Classification of faults into such categories is difficult as intermittent faults and transient faults manifest similarly. This is even more apparent, as systems move to lower technology nodes. It then becomes difficult to classify faults with traditional approaches, as criteria used for classification are not conclusive. This ambiguity, and the fact that the problem at hand is changing as technology continues to evolve, calls for an alternative approach, which is adaptive and where the system can classify faults accurately.

Machine learning techniques are shown to be useful in cases, where it is difficult to express knowledge in terms of a fixed set of rules. Machine learning approaches have shown that the the lack of knowledge can be compensated with data [Alp04]. In our case, the ambiguity of the classification rules can be compensated by storing past classification decisions and “learn” from those for accurate classification. The primary focus of the thesis is to explore the possibility of fault classification by using machine learning approaches.

1.2 Thesis Organization

This thesis report is organized as follows:

Chapter 2 – Faults in VLSI Systems: The first part of this chapter defines the fault taxonomy used for classification. It covers the definition, sources and fault models used for each of the fault types. It also notes characteristics for fault types used to derive features in later chapters. The second part of the chapter covers related work done to classify faults in computing systems and on VLSI chips. The third part presents a brief account of fault diagnosis and diagnostic parameters.

Chapter 3 – Machine Learning For Classification: This chapter covers some basic terms and definitions used in machine learning. It explains supervised and unsupervised approaches in machine learning. Also, a brief survey of some of the popular machine learning algorithms is covered in this chapter, along with their training procedures.

Chapter 4 – Problem Definition and Feature Selection: A detailed account of the problem definition is covered in the first part of this chapter. The second part covers feature selection for classification and their extraction methods.

Chapter 5 – Experimental Setup: This chapter starts with assumptions and the experimental setup to generate sample population and test data for learning. Rest of the chapter focuses on the procedure to train the classifier and using it for fault classification.

Chapter 6 – Evaluation of results: The results obtained using techniques described in the chapter 4 and 6 are presented in this chapter.

Chapter 7 – Conclusion and Future Work: This chapter concludes the thesis and summarizes possible applications of method suggested in this thesis and the future work.

Chapter 2

Faults in VLSI Systems

The reliability is always a cause of concern during chip manufacturing. A manufactured chip needs to function correctly not just during post-manufacturing tests but during the complete lifespan of the final product. The typical lifespan for a chip designed for commercial purpose is defined as 11.4 years or 100,000 hours [KP09]. The failure rate of ICs with respect to time is shown in figure 2.1, typically known as the *bathtub curve*.

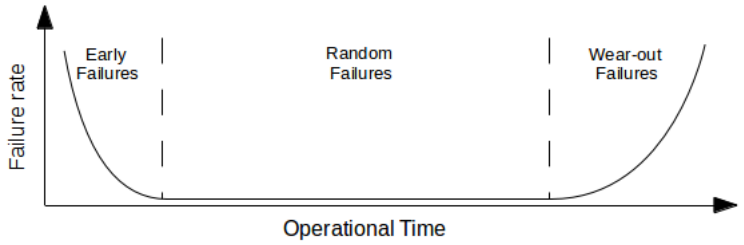


Figure 2.1: Bathtub Curve

The first region of the graph is called early failures or *infant mortality region*. The second region is the lifetime of the device when random failures occur. The error rate in this region is low and constant. The third region of the graph is wear-out and is caused by failures at the end of the useful life [KP09]. It can be expected that the ICs will not enter this region due to technology advances and obsolescence. This makes the first region important from the view of product quality. Early failures of chips can be remedied to a great extent by testing them immediately after manufacturing [Agr00] and thus it is important to detect faults at the manufacturing level.

Failure of a semiconductor chip can be described using a fault, defect or an error. A *defect* in an electronic system is an unintended difference between implemented hardware and its design [Agr00]. Defects can occur either during the manufacturing process or lifetime of the device. An *error* is said to have occurred when an unintended signal is generated by the system.

An error may result in failure at primary outputs of the system. An error is essentially the manifestation of a defect. For the purpose of analysis, a defect is modeled as a *fault*, which is simply representation of defect at the abstracted function level.

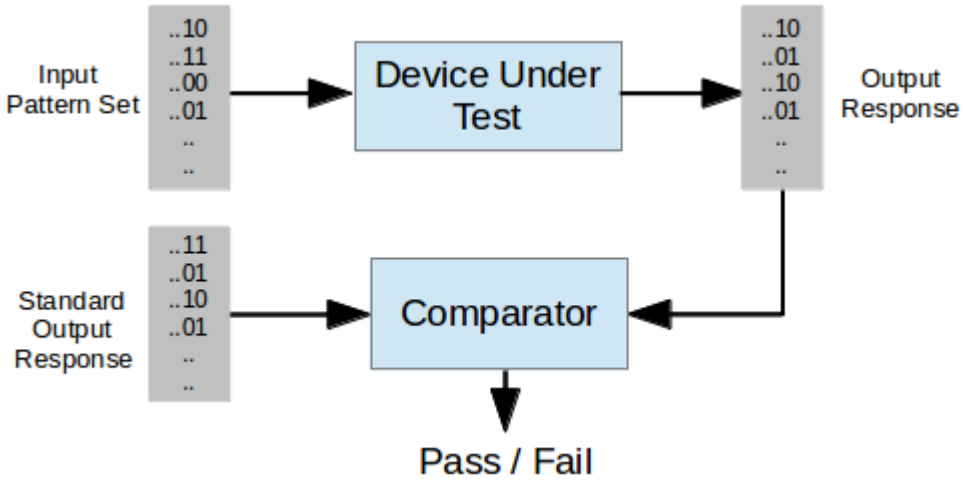


Figure 2.2: Typical test flow for VLSI Chips

The figure 2.2 shows how faulty chips are identified. To decide whether the Device Under Test (DUT) is working properly, a set of input stimuli called *test pattern set* is applied to the DUT. The *output response* is observed and is then compared to the standard output. If these outputs do not match then the chip is said to be faulty.

The sources of the fault can be either internal or external. When healthy chips fail due external mechanisms like α-particle strikes, they are thrown away in a typical test, which contributes to yield loss. Thus to maintain product quality and to reduce yield loss, it is important classify the faults according their criticality. Section 2 of this chapter describes a taxonomy for such classification according to their sources and characteristics. It also focuses on various fault models that can be used to analyze these faults. The existing techniques for such fault classification are explained in section 3. The last section of this chapter explains diagnostic techniques and how they can used to classify faults.

2.1 Fault Taxonomy

According to the source and behavior of faults, they can be classified into three types: *permanent*, *intermittent*, and *transient* [Con03]. Permanent faults reflect irreversible physical changes. Intermittent faults occur because of an unstable or a marginal hardware, and can be activated by environmental changes, like higher or lower temperatures and voltage. Transients occur

because of temporary environmental conditions [Con03]. The likelihood of these faults is expected to increase with increase in transistor densities on semiconductor chips [Con07a].

2.1.1 Permanent Faults

Permanent faults are those which occur due to physical defects on the chip. The source of these faults generally lies in the issues of manufacturing process. However, they can also occur during operational lifetime of the circuit, especially when circuit is old and starts to wear-out.

The common sources of permanent faults are described below:

Manufacturing process: Sso-called *spot defects* can occur during manufacturing of a VLSI chip, and take form of either missing or extra material. Such a defect can cause an unwanted short or open between nodes or make an unintended multi-terminal transistor,leading to changed circuit topology. These defects mainly arise from some contamination, usually in form of dust particles or liquid droplets deposited on the wafer surface during some fabrication step [KM96]. Also missing or excess metal may cause unwanted capacitance and resistance respectively resulting in delay lines [WK95].

Wear-out: *Electromigration* (EM) is defined as mass transport of metal atoms created by collision of electrons [Gha82]. This movement of material will result in voids or hillock growth as in figure 2.3, which can result in an open circuit or a short between adjacent tracks [Ana00]. With lower technology nodes, the wire widths are also getting smaller making EM a serious problem.

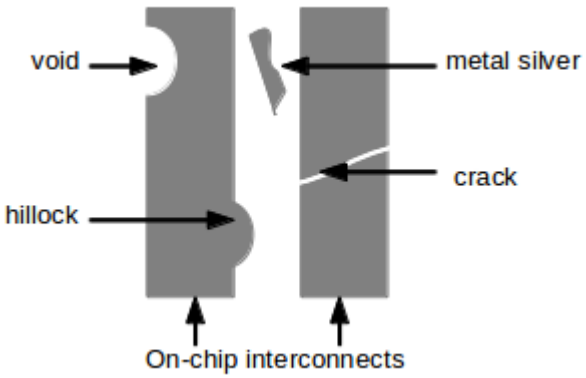


Figure 2.3: Manufacturing defects as sources for intermittent and permanent faults [L+09].

Once a permanent fault appears in the system, it does not go away, unless the offending component is replaced. They are localized on the chip, and hence will affect the same

set of primary outputs (POs). Permanent faults are reproducible and provide a predictable output response. Permanent faults only go away once the offending component is replaced [Con07a].

Some of the ways to model permanent faults are noted below, these are the models used to generate experimental data in this work.

Stuck-at fault model: Stuck-at fault models are the most simplest fault models. Due to its simplicity it is a widely used fault model[Lar06]. It assumes that the fault location has a fixed logical value, either stuck-at 0 or stuck-at 1. These can be seen as short to ground or short to power supply respectively. When it is assumed that there is only one fault in the circuit at a time then *single stuck-at* (SSA) model is used, otherwise in case of multiple defects *multiple stuck-at* model is used.

Wired AND/OR fault model: Unlike stuck-at, *bridge fault* models a short between signal lines. *Wired AND/OR* fault model is a type of the bridging fault model. These models are used to describe the logic behavior of two nodes that are shorted in the circuit. Wired AND model assumes that the faulty node of the bridge always has value 0, whereas wired OR model assumes faulty node has value 1.

Delay fault model: *Delay fault* model is used to model timing related faults. Delay testing is required for modern VLSI systems running at high frequencies, as even minor timing violations can lead to system performing out of specifications [Lar06]. There are two was to realize delay viz. *Gate delay* and *Path delay* models. Gate delay model assumes that the delay is only between input and output of individual logical gates on chip. In contrast, path delay models assume that the delay is spread over complete path from input to output.

2.1.2 Intermittent faults

Intermittent faults are those caused by a marginal or an unstable hardware and are activated when certain conditions like voltage, temperature or frequency are met [Con03, L+09]. Intermittent faults often precede occurrence of permanent failures [L+09].

Some of the common sources for intermittent faults are described below:

Manufacturing defects: As illustrated in figure 2.3 *metal silvers* are stray pieces of metal on die due to some process imperfections. In certain conditions like increase in temperature, the metal may expand and touch the interconnects creating a short. In some cases the short might cause a current surge, damaging the circuit and can manifest into a permanent fault [HKS03]. Similarly cracks, as shown in same figure can continue to work normally at design temperature but at low temperatures can cause open circuits.

Technology scaling: With the technology scaling, the reduced thickness of oxide layers may result in current leakage, with a mechanism known as soft breakdown (SBD) [Sta01]. In such a breakdown, the current fluctuates creating intermittent fault, without causing a thermal damage [Sta01, Con07b, Con07a].

Power droop: With the technology scaling, supply voltages are also lowered down, this results in degraded tolerance to power supply noise resulting in a *low frequency power droop*[PCKB07]. A *high frequency power droop* occurs when multiple cells on a chip connected to the same power grid segment switch in the same direction, increasing their current demand causing a power starvation in some other part of chip [PCKB07].

Once intermittent fault appears in the system, its probability of recurrence increases with time[BCDGG00]. These faults are localized on the chip. They will affect the same set of primary outputs (POs) on recurrence. Intermittent faults have tendency to occur in bursts [Con07a, Con03]. Intermittent faults are not reproducible every time for same set of test patterns. Intermittent faults only go away once the offending component is replaced [Con07a].

Intermittent faults can be modeled as conditional stuck-at faults, activated by trigger condition. The activation condition can be expressed as a Boolean function and can depend on timing or environmental conditions [HW09].

High frequency power droop: This type of fault occurs when specific set of input causes power starvation in some other part of the chip. Hence, this fault is modeled as a set of aggressor lines $a_1, a_2, ...$ and a victim line v [PCKB07]. The fault occurs on victim line, due to presence of the aggressor lines.

2.1.3 Transient faults

Transient faults are deviations of normal circuit function caused by some environmental factors or some external phenomenon. They are called soft errors as they do not do any permanent damage to the chip. A *single-event upset* (SEU), which is change in value of single bit, is the most common manifestation of transient faults.

Some of common sources for transient faults are noted below:

ESD and EMI: *Electromagnetic interference* caused by sources emitting high energy signal may interfere with the working chip to bring about SEUs. An *electrostatic discharge* due to users releasing static charge can also affect chips to cause transient faults.

Particle strikes: When an α-particle, a proton or a neutron passes through a semiconductor material and starts to loose energy, it frees electron-hole pairs along its path [DM03]. If this material happens to be a reversed biased p-n junction, it can result in significant transient currents to bring about an SEU. Hence with scaling to lower technology nodes, it is very likely that probability of such SEUs will increase.

Transient faults are non-deterministic faults. These faults are not localized hence can affect any of the POs. Transient faults are not reproducible. Replacing the offending component may not make transient faults to go away [Con07a]. Transient faults are isolated incidences of error occurrence. They usually do not occur in bursts like intermittent faults [Con07a, Con03]. Once the condition triggering the transient fault disappears, circuit return to normal functioning.

One of the ways to simulate a transient fault is to implement a conditional stuck-at fault, at multiple fault location and to use a deterministic function to trigger the fault. This work uses a bit-flip as a fault model to model transient faults.

bit-flip A bit at random location is flipped using a trigger condition. This corrupted in value is then propagated through circuit [GBGG01].

2.2 Fault Classification

Test flows like the one shown in figure 2.2 are able to distinguish between faulty and healthy chips. The traditional manufacturing tests still work the way as shown in figure 2.4 [WE85]. Chips are tested for faulty behavior, and all chips showing single fault in a test run are thrown away.

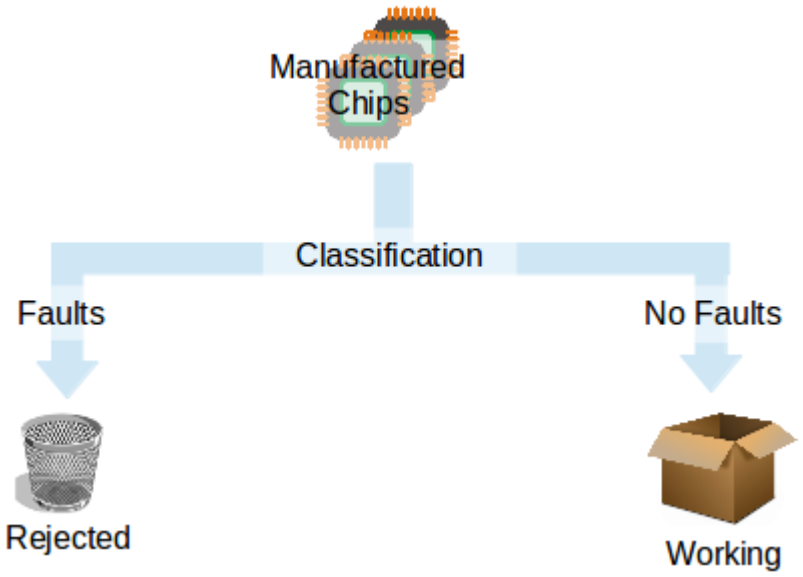


Figure 2.4: Traditional flow for fault classification

With the advances in manufacturing processes, the number of permanent faults is increasing [KP09]. However, on the other hand the impact of soft-errors and other non critical failures is

increasing at much higher rate[Con03]. Some studies have indicated that up to 80% failures can be attributed to SEUs [IR86, DKCP94, KP09], which suggest that these cases will result in an unnecessary yield loss. This section describes a few techniques to discriminate between different error categories described earlier in the chapter. As the fault taxonomy we used is similar in characteristics to those observed on PCs or workstations, the approaches used for their classification also provide a few pointers towards fault classification in VLSI systems.

In [LS90] a mechanism to classify between transient and intermittent faults is explained for error log analysis. In a technique called *Dispersion Frame Technique* (DFT), the inter-arrival time in between successive error events of same error types is used to determine type of the fault in the system. Heuristics are applied to determine their closeness in time and affected area, which are then considered as parameters to decide whether the error is of the same type.

Authors in [IYI90] use a similar technique to identify persistent failures in the system. Here they have used error rates to build up correlation using simple probabilistic techniques between error records, leading to a set of symptoms which may suggest a common cause (permanent errors).

A probabilistic approach is considered in [PSBDG98], which updates the probability of module being affected by permanent fault. It then weighs the consequences of actions performed by a faulty module against a fault-free module. It uses Bayesian inference to discriminate between permanent and transient errors

Historically a lot of work has been done to analyze impact of different types of faults on VLSI systems [Con03, Con07a, DM03] and to classify them [Sav80, EARG13, BCDGG00, DK09].

The most popular techniques to classify transients from other types of faults are grouped under a family called α-count techniques [BCDGG00]. In a scheme called *single threshold α-count techniques*, a single threshold is established and if error count exceeds this threshold then the fault is classified as permanent or intermittent, whereas a smaller non-zero value indicates presence of transient faults. In an other variant of the same called *dual threshold α-count techniques*, two thresholds are established. If the error count exceeds first threshold then that component is assigned a restricted functionality and when it exceeds the second threshold it is taken out of service, like the single threshold technique. However, a component which is in-between thresholds can be taken in full service once its error count is lowered than first threshold.

Another approach using diagnostic probabilities in [DK09] is able to distinguish permanent faults from faults with non-deterministic behavior.

2.3 Fault Diagnosis

Diagnosis is the process of locating faults in a physical chip at the various levels down to real defects. In the traditional fault-dictionary based diagnosis, we are given two sets of data, a *predicted output* P which is a set of outputs when fault a particular fault is active in the system, a *measured set* M, which is the observed fault behavior and corresponding fault $f_i \in \{f_1, f_2, ..., f_n\}$. When the two sets match i.e. $P = M$ the corresponding fault is diagnosed to be active . When $P \neq M$ then logic diagnosis tries to find the best fitting explanation. However it is practically infeasible to construct such fault dictionaries for modern circuits, as the fault dictionary should consist of all possible faults and their combinations [Wan10].

An adaptive approach which does not use fault dictionaries called *Partially Overlapping Impact couNTER* (POINTER) for diagnosis is described in [HW09]. This approach uses test pattern sets with the *Single Location At a Time* (SLAT) property [BSHH01] to diagnose faults present in the circuit. The authors in [HW09] define a *(*Fault Machine) (FM), *i.e.* a reference circuit with stuck-at faults injected. As shown in figure 2.5, a tuple of parameters called *evidence* is defined as,

$$e(f, T) = \{\sigma_T, \iota_T, \tau_T, \gamma_T\}$$

.

σ_T is the sum of number of failing output where the Device Under Diagnosis (DUD) and the FM match. It is calculated as sum of all σ_t values by injecting one stuck-at fault at a time in the FM.

ι_T is the sum of number of output pins which fail in the FM but are correct in the DUD. It is calculated as sum of all ι_t values by injecting one stuck-at fault at a time in the FM.

τ_T is the sum of number of output pins which fail in the DUD but are correct in Fthe M. It is calculated as sum of all τ_t values by injecting one stuck-at fault at a time in the FM.

γ_T is the sum of maximum of corresponding values of ι_t and τ_t for every test pattern.

The values of ι and τ are high and σ is low or zero, if the fault present in DUT is not explainable using any of the stuck-at faults injected in FM. Hence γ is an indication about the ability to explain of the fault. For a permanent fault, the value of gamma would be low (zero).

After diagnosis, in a process called *ranking* obtained values are sorted as evidence with lowest value of γ first. The evidences with the equal values of γ are then sorted with the highest value of σ first. For evidences with equal values of both σ and γ, the ranking algorithm puts evidences with lowest value of ι first [HW09]. This process brings the most explainable faults first. The result of diagnosis is then returned as the evidence with the highest rank.

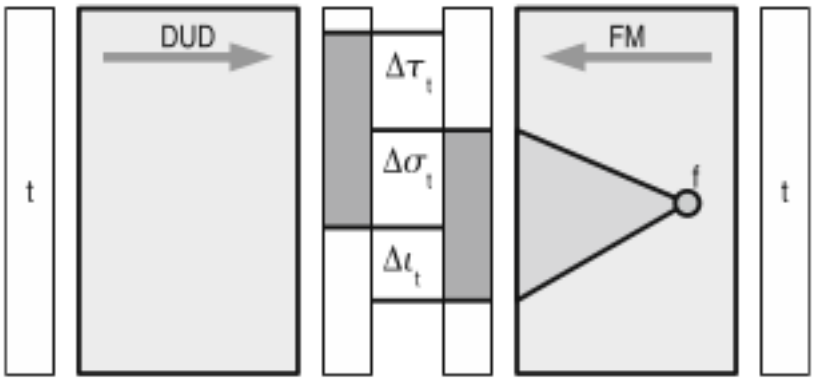


Figure 2.5: Evidence generated during diagnosis [HW09]

These parameters vary depending on the type of fault present in the circuit. For permanent faults, the value of evidence parameters remains same, when same test pattern is applied at the input. The evidence would vary for each test run, as intermittent and transient faults are non-deterministic.

Chapter 3

Machine Learning For Classification

An *algorithm* is a set of instructions used to convert input values to an output, based on certain rules. Consider an example where we need to find all even numbers from a dataset. Here, we can set up a *rule* that if a number is completely divisible by two then it should be included in the output dataset, otherwise not. Naturally, as there can be more than one way to solve a problem, there can be more than one algorithm to solve it. However, there are certain examples where formation of set of rule is practically infeasible. For an example, consider a handwriting recognition software used to scan handwritten forms. Figure 3.1 illustrates the problem at hand, where a simple character can be written in a number of ways. It is interesting to note that humans are able to read this data without a trouble, but it is really difficult to infer a set of rules which would result in an accurate recognition with help of an algorithm. Machine learning is employed in such cases. Specifically *Machine Learning* (ML) is programming computers to optimize a performance criterion (e.g. character recognition) using example data or a past experience [Alp04].

Figure 3.1: Example of Machine Learning: Character recognition

In case of a handwriting recognition system, an “example data”, in the form of images of handwritten characters with their *labels*, *i.e.* a number or an alphabet, which each image represents, is collectively referred to as a *training set*, and is used to teach machine learning how a character with the given label would look like, so that ML can recognize when it

encounters similar data in the future. Machine learning can be applied in a wide range of applications, where it is not possible to the express human expertise, but a large amount of sample data is available. Typical applications of machine learning include computer vision, pattern recognition, spam filtering, search result optimization etc.

3.1 Types of learning algorithms

Based on whether we know labels for the data, ML algorithms can be can be classified in two major categories - supervised learning and unsupervised learning.

Supervised learning algorithms are used when labels of the data to be are known. A spam filter is a good example where supervised learning can be used for *classification*. Here we know an email received is either "spam" or "not-spam", these categories can be used as labels for the sample population and learning algorithm can classify within these two type. One more application of supervised learning is to predict a numerical value in *regression*. Consider a problem to predict value of a used property, the input parameters in this case are the initial value, year of the construction, size of the property, the locality and so on, whereas the output is the current resale value. one can construct a training set of known resale values and receptive values of input parameters and train the leaning algorithm to predict other inputs. To generalize, aim in supervised learning is to learn mapping from input to output whose correct vales are provided by supervisor [GCM08].

Unsupervised learning is used in classification problems where the labels for the data are not known. An example of such problem is data clustering [JD88]. One of applications of this is to cluster news reports which belong to the same category like sport, science, art and so on. The number and the labels of categories in this case are not defined, and the machine learning application needs to cluster articles based on some common words, and provide the supervisor data, which he may use to label the clustered groups. The aim in supervised learning it to find out regularities or correlation the input data, without explicit need of a supervisor [MF08]

In case of fault classification, a classification taxonomy has been discussed in earlier chapter. The sample population, which we would generate in our case will contain labels. This makes our case as supervised classification problem. In following subsection, we define the basic terms as applied to case of supervised learning.

3.2 Basic terms in supervised machine learning

A *feature* (x_i) is a result of measurement made on a unit input data. Generally, a set of features $(\boldsymbol{x})$ is needed to characterize a unit of input data and is expressed as,

$$\boldsymbol{x} = [x_1, x_2, \dots x_m]^T$$

Its label l denotes the class $C_i \in \{C_1, C_2 \dots C_k\}$ it belongs to and it is denoted as,

$$l_i = \begin{cases} 1 & \text{if } \boldsymbol{x} \in C_i; \\ 0 & \text{if } \boldsymbol{x} \in C_j, j \neq i \end{cases}$$

The *training set* X is then defined as a set containing N values of such examples,

$$X = \{\boldsymbol{x}^t, \boldsymbol{l}^t\}_{t=1}^N$$

The aim for machine learning algorithm is to learn data and their labels in the training set and then classify the new examples $\boldsymbol{x}$ by estimating the value of $C(\boldsymbol{x})$. To achieve this, the algorithm tries to find out a hypotheses for every class, $h_i, i \in \{1, 2, \dots k\}$ from a set of all possible hypotheses such that,

$$h_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in C_i; \\ 0 & \boldsymbol{x} \in C_j, j \neq i \end{cases}$$

The *empirical error* after training is calculated as,

$$E(\{h\}_{i=1}^k | X) = \sum_{t=1}^{N} \sum_{i=1}^{k} |h_i(\boldsymbol{x}) \neq l_i^t)|$$

Figure 3.2 shows two possible hypotheses h_1 , h_2 and the actual boundary of classification C. For a simple 2-class classification problem, both the hypotheses have the same value of empirical error. If we choose hypothesis h_1 then the examples which lie in region between h_1 and C will get incorrectly classified and this is called as *overfitting*. On the other hand, if we choose h_2 then same will happen for examples in the region between C and h_2, called *underfitting*. To check if overfitting or underfitting is has occurred, typically one more labeled dataset called as *cross-validation set* is picked. The empirical error is then calculated over this set and hypotheses obtained during training and the hypothesis with least value of error is then selected.

In this report, the term *sample population* collectively for the training set and the cross-validation set.

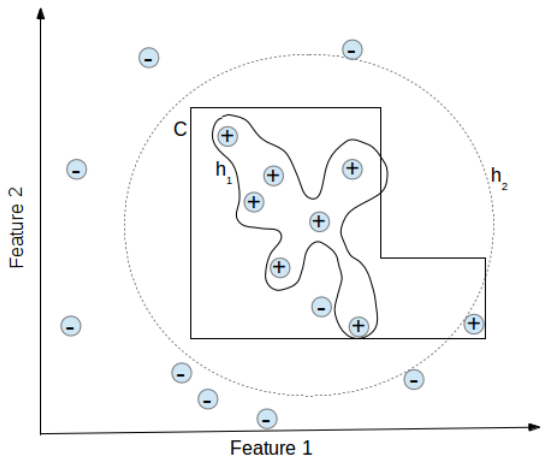


Figure 3.2: Example of overfitting and underfitting

3.3 Machine Learning Algorithms for Classification

This section provides a brief overview of some of the most frequently used machine learning algorithms for the classification problem. It includes advantages and disadvantages for individual algorithms.

3.3.1 Naive Bayes

Naive Bayes is one of the simplest algorithms for learning, more interestingly in some cases it may outperform most of the sophisticated learning algorithms [JL95]. The naive Bayes uses maximum-likelihood estimation to classify new examples. It is based on the Bayes' theorem which states,

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

Where $P(A)$, $P(B)$ being the probabilities of A and B, and $P(A|B)$ and $P(B|A)$ are conditional probabilities of A, given B and B, given A respectively.

During the training of the naive Bayes, the probability of finding an example of each class in the sample population is calculated and stored as the *prior* probability for that class. It also calculates probability for instances $\boldsymbol{x}$ given its class c. Under the assumption that attributes in $\boldsymbol{x}$ are independent, it simply becomes a product of probabilities of each single attribute

[WZA06]. Hence Bayesian theorem, when applied to classification problem using Naive Bayes, becomes

$$P(C_i|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|C_i) \times P(C)}{P(\boldsymbol{x})}$$

The probability $P(C_i|\boldsymbol{x})$ is referred to as *posterior* probability. A class C_i is chosen if $P(C_i|x) = \max_k\{P(C_k|x)\}$, *i.e.* the class with highest posterior probability.

A clear advantage of using the naive Bayes is that, it is fast to train and fast to classify the data. This is because it needs to scan the database to compute probabilities and store it in a table during training and use this table to classify future examples. Also the naive Bayes is inherently more robust against irrelevant features [KLT08], as the likelihood of a class is product of probabilities of each single attribute. On the other hand for prior probabilities to be realistic, the sample population needs to be truly representative of the actual data. Another major disadvantage is that the classifier assumes features to be independent of each other. However in many cases Naive Bayes classifier performs reasonably well even in cases where features are dependent on each other [JL95, WZA06].

3.3.2 Decision Trees

Decision trees are hierarchical models, wherein each step is a simple threshold-test function of nominal value of a feature against a fixed threshold value [Kot13]. The steps in the hierarchy are called as *decision nodes* and a *test* is implemented in the form of a function on features $\boldsymbol{x}$ of an example, with discrete outcomes represented as branches. These nodes apply tests recursively on a feature or a set of features of the example data, until it flows down the tree and hits a *leaf node*, which represents the output (class in case of classification) [Alp04]. A simple decision tree is illustrated in figure 3.3.

In terms of a computer program, this algorithm devices a set of rules which can be interpreted as nested `IF-ELSE` structure. The *decision tree learning* algorithms are used to derive decision trees. ID3, C4.5 are some examples of these algorithms [Mit97]. ID3 [Qui86] is the simplest of these algorithms. In the case of decision trees, training is to choose features which provides the most information about training set. It then constructs tree using top-down approach. Other advanced algorithms like RIPPER [C$^+$95] build upon the same approach and then employ *pruning* to reduce the training error.

Decision trees use a “white-box” approach, wherein the internal decision making and structure of tree is visible to user. This also makes decision trees easy to visualize and interpret [Kot13]. Decision trees also perform a feature screening to put less informative features near leaf nodes, by its construction.A disadvantage of decision trees is that they can create over-complex trees that do not generalize the data well, *i.e.* the overfitting of data. The problem of learning

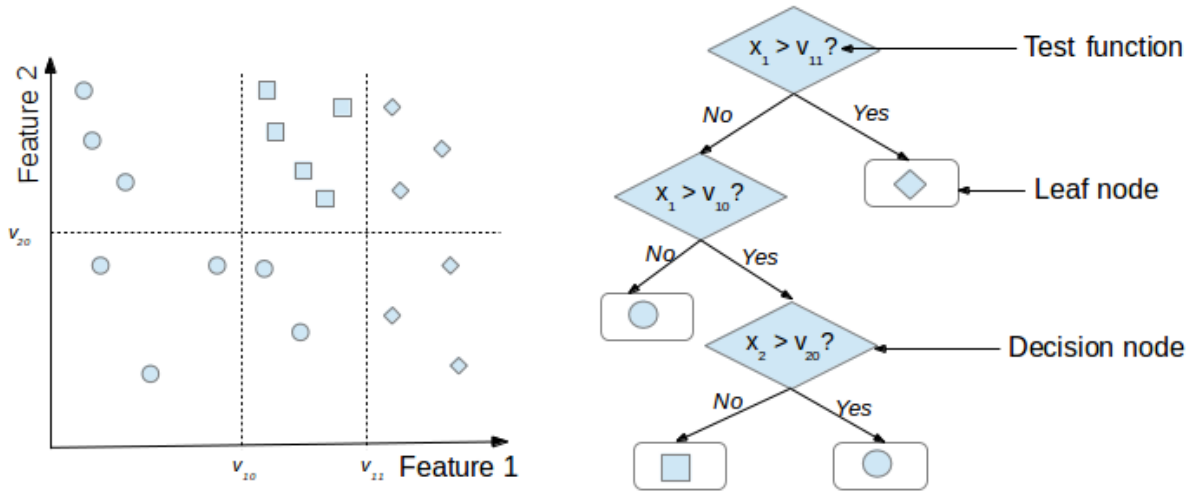


Figure 3.3: Example of decision tree

decision trees is known to be NP-complete hence its worst-case training speed can be slow [HR76].

3.3.3 Multi-Layer Perceptrons

Multi-Layer Perceptrons (MLP) is a type of artificial neural network models and has been in use since the early 80's. In this model, each feature and output are represented as nodes, and feature nodes in each layer are connected to the upper layer using weights or *synapses*. Figure 3.4 is an example of a simple, two layered perceptron. Inputs $x_1, x_2 \ldots x_k$ are features and $x_0 = +1$ is a *bias element*, used to make model more general by allowing user to fine-tune the the output by shifting the output function. $\boldsymbol{a}, \boldsymbol{b}$ are matrices of weights on the synapses in the first to the hidden layer and the hidden layer to the output, respectively.

The output of perceptron in figure 3.4 can be represented mathematically as

$$y = \sum_{j=0}^{n} b_j (\sum_{i=1}^{k} (a_{ij} x_i + a_0))$$

During the training of a perceptron, the training algorithm will try to find the appropriate connecting weights. Multiple layers of perceptrons can be constructed by implementing a hidden layer of nodes between features and output, by doing so one can implement non-linear output functions. The degree of non-linearity depends on the number of hidden layers. Back-propagation algorithm [RHW85] is one of commonly used algorithm for training MLPs. It

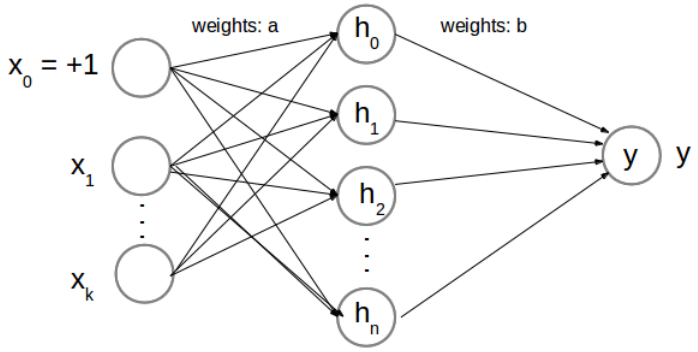


Figure 3.4: Two layered perceptron

works by calculating error correlations at each output and use these to calculate the error terms in previous layers and so on. The error terms are then used to adjust weights of the individual synapses.

The error function in this case is defined as,

$$E(\boldsymbol{a}, \boldsymbol{b}|X) = \frac{1}{2}\sum_{t}(l^t - y^t)^2$$

The gradient of this error function is calculated during back-propagation, and weights are updated once the gradient function reaches a local minimum.

Once trained, MLPs are able to classify data fast. They can implement higher order polynomial functions and are flexible and powerful due to well researched mathematical background and variety of training algorithms available, which can be selected according to the application and the amount of data available. A disadvantage is that the network needs to be completely re-trained when new training data is to be added. Selection of features also has a profound impact on the performance of MLPs [KM02, EK10].

3.3.4 Support Vector Machines

Support Vector Machines (SVMs) have existed for a long time, but the research on these gained particular momentum since Vapnik [Vap95] evaluated these methods in his book on statistical learning theory. SVM belongs to the class of linear classifiers. In higher dimensions, SVM tries to divide the feature space using decision hyperplanes. However, as there can be several planes that divide feature space, SVM selects the plane with maximum distance from support vectors.

Figure 3.5 shows a case where two possible lines divide feature space, h_1 and h_2. SVM will choose h_2 as the *decision boundary* or the *discriminant function* as it provides maximum margin for the classification. The examples with least distance from the decision hyperplane are called *support vectors*.

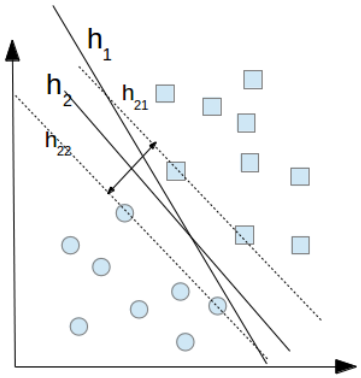


Figure 3.5: SVM with linear decision boundary

The linear discriminant function used in this case can be expressed as,

$$g(x) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + w_0$$

Where w_0 denotes a bias, and the vector $\boldsymbol{w}$, called weight vector, is the distance of respective hyperplanes passing through support vectors from the origin.

Referring to figure 3.5, hyperplane h_2 is actually a result of two hyperplanes, defined by the respective support vectors of two classes. Let h_{21} and h_{22} represent these hyperplanes such that:

$$h_{21} : \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = 1 \quad \text{when label is } +1;$$
$$h_{22} : \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b = -1 \quad \text{when label is -1}$$

However as illustrated in the figure 3.6 the feature space may not be linearly separable at all. In these cases SVM uses *kernel trick* to achieve linearly separable kernel space. The idea behind kernel trick is to apply a function ϕ to transform all points in the feature space to a higher dimension, so that resulting feature space is linearly separable. After transformation, the regular SVM algorithm is used for classification.

This makes the linear discriminant used of the form,

$$\boldsymbol{y} = \boldsymbol{w}^{\boldsymbol{T}}\phi(\boldsymbol{x}) + w_0$$

where ϕ denotes the function used to convert the non-linear feature space to linear one.

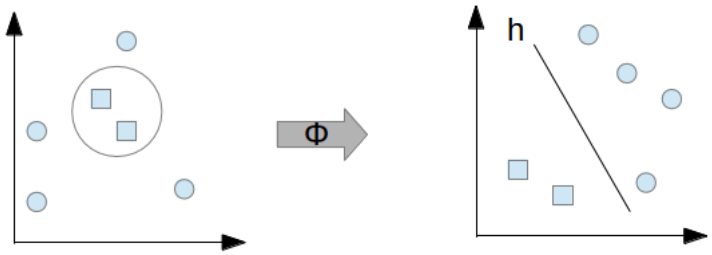


Figure 3.6: Kernel trick for SVM

Let $||\boldsymbol{w}||$ denote the distance between these hyperplanes, C a constant to adjust variance and to minimize the training error ξ . Then the training of SVM is to find a maximum margin hyperplane, which can be viewed as an optimization problem [Vap95, CL11],

$$\begin{aligned} \min_{\boldsymbol{w},\xi,w_0} \quad & \tfrac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & l^t(\boldsymbol{w^T}\phi(\boldsymbol{x}) + w_0) \end{aligned}$$

This QP may be difficult to solve as function $\phi(\boldsymbol{x})$ can be high in dimensions. Making the mapping used computationally expensive [BHW10].

To solve this, suppose $\boldsymbol{w}$ can be expressed as $\boldsymbol{w} = \sum_{i=1}^{n}\alpha_i\boldsymbol{x_i}$ (dual of this problem), and also define a *kernel function*,

$$K(\boldsymbol{x},\boldsymbol{x_i}) = \phi(\boldsymbol{x})\phi^T(\boldsymbol{x_i})$$

Substituting this in minimization problem reduces the dimensionality back to the original.

Most commonly used kernels are:

Linear $K(\boldsymbol{x},\boldsymbol{x_i}) = \boldsymbol{x}^T\boldsymbol{x_i}$

Polynomial $K(\boldsymbol{x},\boldsymbol{x_i}) = \gamma(\boldsymbol{x}^T\boldsymbol{x_i} + r)^d, \gamma > 0$

Radial Basis Function (RBF) $K(\boldsymbol{x},\boldsymbol{x_i}) = \mathrm{e}^{-\gamma||\boldsymbol{x}-\boldsymbol{x_i}+r)||^2}, \gamma > 0$

Sigmoid $K(\boldsymbol{x},\boldsymbol{x_i}) = \tanh(\gamma\boldsymbol{x}^T\boldsymbol{x_i} + r)$

γ, r and d are the kernel parameters.

The advantage of using SVM is its ability to tackle non-linear data. By using a variety of kernels like linear, polynomial, RBF and so on, the user has flexibility to classify the data with non-linear feature spaces [CL11]. Handling higher dimensional features spaces is also not an issue, because of the theoretical framework [Vap95] supports an n-dimensional space. SVMs also provide a unique solution as the optimality problem is convex, which is an advantage as

compared to neural networks, which can provide solutions at a local minimum [AM08]. SVM, being a maximum margin classifier, avoids underfitting of data. One the other hand, training an SVM is solving an quadratic optimality problem (known to be NP hard), hence it can take a long time to train when datasets are large. Also, a common disadvantage of SVM is that the model it constructs is a “black-box” approach to classify the data, specially when the feature set is large [AM08].

Chapter 4

Problem Definition and Feature Selection

There are a number of methods to separate permanent faults from non-recurring faults as explained in the section 2.2. However available techniques do not separate faults as permanent, transient and intermittent. This is of particular importance from the point of view of reducing yield loss, by including chips which showed only transient faults. Also, if faults can be categorized into permanent transient or intermittent, then it gives some additional information to the designer about the underlying fault mechanism, so that additional optimization of yield can be achieved.

The yield can be improved by taking the rejected chips, and analyzing them further by running tests again multiple ($TestRuns$) times, and decide on what type of fault caused the failure during the test. These chips that showed a non-critical fault can still be included in the final yield. However an incorrect analysis would result in the degradation of product quality. Hence, the classifier should be as accurate as possible.

The criticality of a fault is an abstract concept and it is defined by the application domain of the final product. Sometimes the user is more interested in optimizing the yield for quality, he would then wish to reject all chips with a slightest possibility of intermittent failure. In this case, the classifier should be optimized for classifying intermittent faults more accurately. However, this would happen at an expense of healthy chips, which showed intermittent failures, and will result in a yield loss. On the other hand, if the user wants to optimize for yield, classifier should be tuned to classify transient faults more accurately, which can impact product quality as more intermittents would be classified inaccurately. The classifier to be designed needs to take this fact in consideration and should provide the user with a functionality to find a suitable trade-off between the magnitude of yield and its quality.

The classification approaches explained in section 2.2 are mostly rule or heuristic based (e.g. the threshold value in α-counting techniques). Generally speaking, when an intermittent

fault occurs in a system, its activation rate is higher than the transient fault rate [BCDGG00]. However, as systems are moving to lower technology nodes, transient faults are also on the rise as explained in section 2.1. Hence with traditional techniques, it becomes difficult to separate transients from intermittents, as the fault rates for these two types of faults become close to each other. Hence an elaborate analysis is required to update these rules for every product and technology. Hence, the classifier to be designed should also focus on building a universal model, which can classify faults irrespective of the product and technology.

To summarize, we need an automated and adaptive approach which is independent on the product and the technology, and that can classify faults as intermittent, transient and permanent accurately.

4.1 Machine learning approach for fault classification

Machine learning has been used in a wide variety of classification applications with a reasonable accuracy [PLV02, NA08, Seb02, Kot07]. As explained in chapter 3, machine learning is used when it is not practical to arrive at rules by looking at the data. Machine learning algorithms can be implemented as a "black-box" approach for classification and all the user needs to do is adjust a few parameters, depending on the machine learning algorithm used. Even the parameter searching can be automated and the user can fine-tune them for further improvement in the accuracy [HCL$^+$03, CMP$^+$00]. This makes machine learning a practical and automated approach when large amount of data is available.

Once a feature set is fixed and the learning parameters are decided, machine learning algorithms analyze the data to set up a classification model. When a technology node is updated, one might have to change the database and train the algorithm again, but the training algorithm takes care of the changed feature space and builds a new classification model accordingly, making machine learning approaches adaptive.

Figure 4.1 explains the basic steps for classification using machine learning methods.

First step in designing a machine learning system for classification is to decide on features, which can be efficiently classify the data. Selection of proper features has the most impact in accuracy of any classifier [MSTC94]. This, along with design of methods to extract features from input data is explained in section 4.2 of this chapter. Selection machine learning algorithm is to be used is explained in the last section of this chapter. The next step is to generate a sample population, covered in the next chapter.

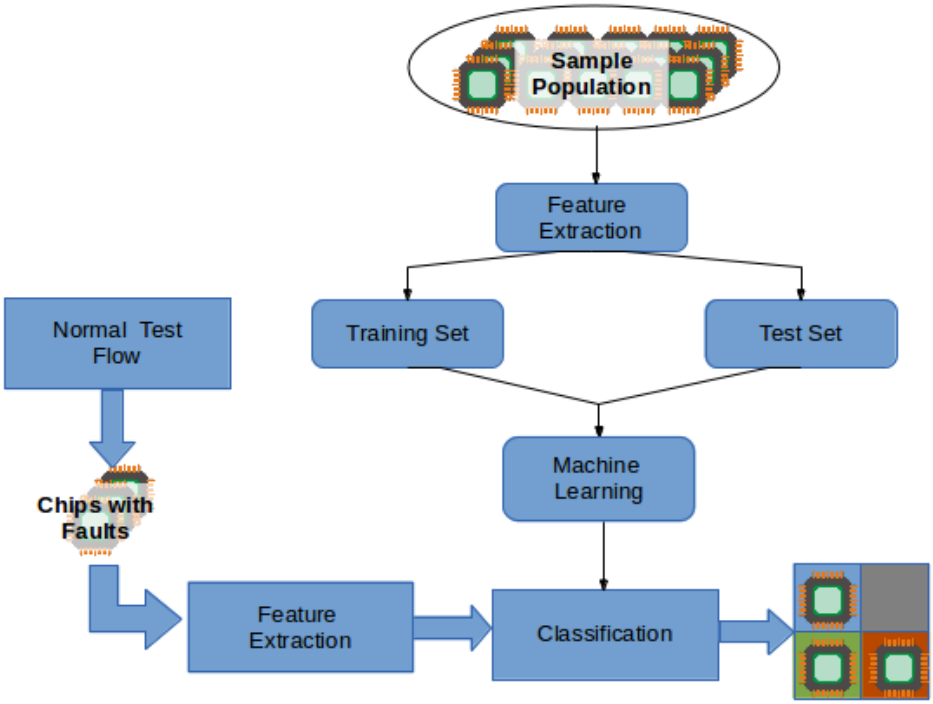


Figure 4.1: Classification of VLSI chips using machine learning

4.2 Feature Selection

To begin with the feature selection, it is important to take a look at all the behavioral characteristics of different types of faults, as covered in chapter 2. Table 4.1 summarizes all important characteristics to be considered for selection of features. Rest of the section describes features that were selected and algorithms for extraction of those features.

Characteristic	Permanent faults	Intermittent faults	Transient faults
Affected outputs	Affects the same set of output pins	Affects the same set of output pins	Affects any of the outputs
Reproducibility	Reproducible for the same test vector	Sometimes reproducible for the same test vector depending upon the fault activation rate	Not reproducible
Location on chip	Fixed to a fault location	Fixed to a fault location	Can affect any location on chip
Fault behavior	Deterministic	Non-deterministic	Non-deterministic

Table 4.1: Characteristics of faults in VLSI systems

4.2.1 Reproducibility of fault

The reproducibility of a fault pattern during multiple test runs is defined as the maximum number of maximum occurrences of same faulty output pattern for a fixed input pattern, and it is denoted by ϵ.

Let the P be the test pattern set, REF_i be the reference output and $OP_{i,j}$ be the output pattern at input i and test run j then,

$$\epsilon = \max_{i \in P} \left\{ \max_{j \in TestRuns} \left\{ \sum_{k \in TestRuns} 1 \cdot e_{ijk} \right\} \right\}$$

where;

$e_{ijk} = 1$ ... when $OP_{ij} = REF_{ik}$ $OP_{ij}, OP_{ik} \neq REF_i$, $j \neq k$, $i \in P$ and $j, k \in TestRuns$

$e_{ijk} = 0$ ... otherwise

Algorithm 1 explains the extraction of ϵ. It takes the expected and the actual output patterns as inputs. It then checks if any of the output patterns was faulty and it calculates maximum occurrences of every faulty pattern for a given input pattern and stores in into an array. The final value of ϵ is maximum value of in this array.

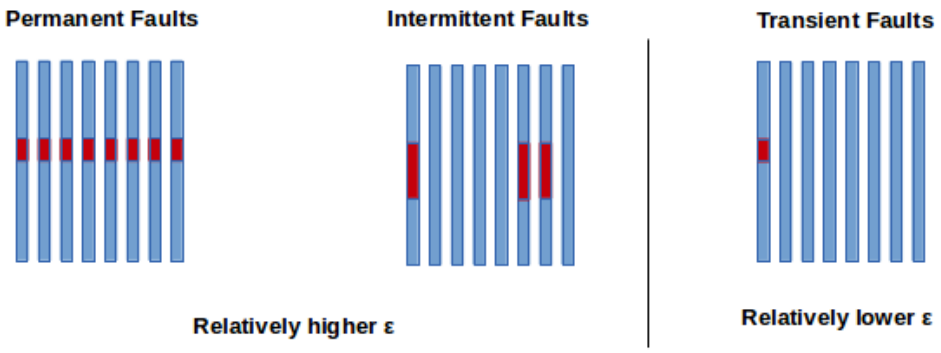


Figure 4.2: Expected behavior of ϵ for different faults

Algorithm 1 Algorithm to evaluate ϵ

```
procedure ComputeEpsilon(Expected output pattern array (EX), Observed output pattern
array for all test runs (OP))
    ε[size(EX)] ← 0
    Index ← 0
    while Index < size(EX) do
        if EX[Index] ≠ any pattern of OP[Index][] then
            ε[Index] ← max(SimilarFaultyPatterns(OP[Index][]))
        else
            ε[Index] ← 0
        end if
        Index++
    end while
    ε ← max(ε[])
    return ε
end procedure
```

Figure 4.2 shows the expected behavior of ϵ for different fault types. Permanent faults are repeatable and the value of ϵ is expected to be equal to the number of test runs for these type of faults. Intermittent faults occur at a higher rate than that of transients for a fixed input pattern, hence they are also expected to have somewhat higher value than transient faults. Figure 4.3 shows actual values of ϵ for a simple circuit (p45k). It should be noted that in this figure, some of the intermittents (marked in red) have the value of $\epsilon = 1$, which is not clearly visible due to resolution constraints and relatively large amount of data. Other circuits used in the experiment also follow a similar trend.

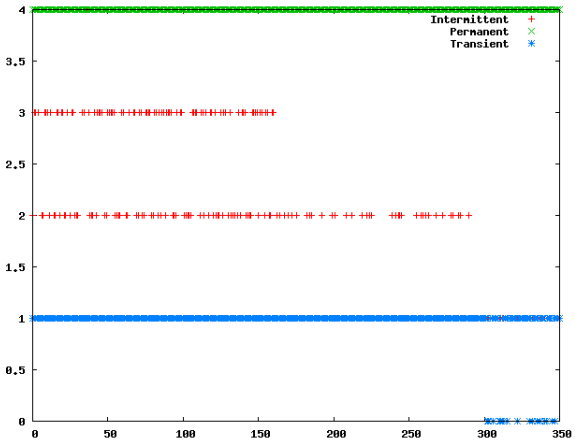


Figure 4.3: Plot of ϵ for p45k

4.2.2 Resemblance of erroneous output patterns

The resemblance of erroneous output patterns is defined in terms of the hamming distance between a set of erroneous output patterns obtained during multiple test runs, for the same input test pattern in a test set. The *Hamming Distance* of a set is evaluated as the maximum of number of positions in which the output patterns differ, pairwise. It is denoted using notation δ_H, subscript H stands for "horizontal", denoting the orientation of calculation of the hamming distance. If all output patterns for an input test pattern are correct then the hamming distance and hence the value of δ_H would be zero.

Let the P be the test pattern set, REF_i be the reference output and $OP_{i,j}$ be the output pattern at input i and test run j then,

$$\delta_H = \max_{i \in P} \{HammingDistance\{OP_{ij} | OP_{ij} \neq REF_i \forall j \in TestRuns\}\}$$

Algorithm 2 explains extraction of δ_H. It takes the expected and the actual output patterns as inputs. It then checks if any of the output patterns is faulty, to save some computational efforts. If any of output patterns is indeed faulty, it calculates the value of δ_H and stores it in an array against the corresponding index of the input pattern. The final value of δ_H is the maximum value of δ_H in this array.

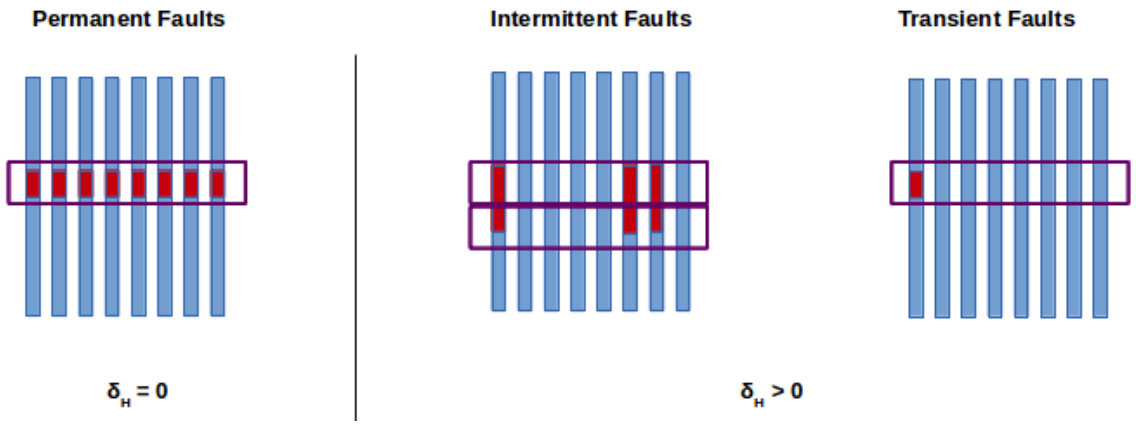


Figure 4.4: Expected behavior of δ_H for different faults

Algorithm 2 Algorithm to evaluate δ_H

```
procedure ComputeDeltaH(Expected output pattern array (EX),Observed output pattern
array for all test runs (OP))
    δH[size(EX)] ← 0
    Index ← 0
    while Index < size(EX) do
        if EX[Index] ≠ any pattern of OP[Index][] then
            δH[Index] ← HammingDistance(OP[Index][])
        else
            δH[Index] ← 0
        end if
        Index++
    end while
    δH ← max(δH[])
    return δH
end procedure
```

Figure 4.4 shows expected behavior of δ_H for different fault types. Permanent faults repeat with same faulty output pattern and value of δ_H is expected to be zero. Intermittent faults, even though fail with same faulty output, are not repeatable and hence are expected to have a δ_H value other than zero. Transient fail randomly at random output locations and hence are expected to have higher δ_H value. Figure 4.5 shows actual values of δ_H for a simple circuit (p45k). Other circuits used in the experiment also follow a similar trend.

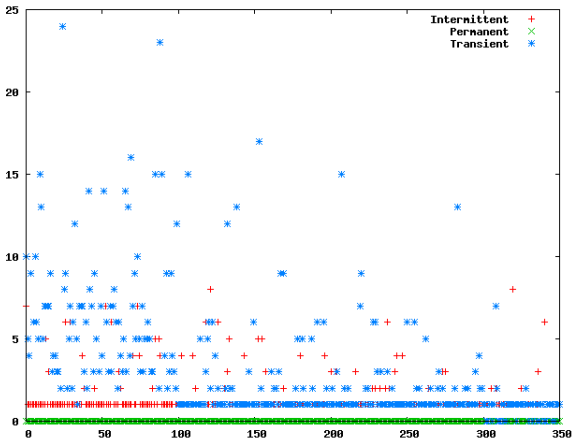


Figure 4.5: Plot of δ_H for p45k

4.2.3 Resemblance of erroneous primary outputs

Resemblance of erroneous primary outputs is defined as hamming distance between primary output locations, that showed a faulty behavior at least once for a respective test run. This quantity is denoted by δ_V, subscript V denoting *vertical* collapsing of all faulty primary outputs for a test run, before evaluating hamming distance.

Let the P be the test pattern set, REF_i be the reference output and $OP_{i,j}$ be the output pattern at input i and test run j, then, the dirty pins are marked as a set M_j for every test run j as,

$$M_j = \{M_{j0}, M_{j1} \dots M_{jn}\}$$

where n is the number of primary Outputs. An element of this set M_{jl}, $0 \leq l \leq n$ has a value 1 if that PO showed a faulty behavior at least once during the test run. This value is 0 otherwise.

δ_V is calculated as,

$$\delta_V = HammingDistance\{M_j\} \quad \forall j \in TestRuns$$

Algorithm 3 explains extraction of δ_V using the method above.

Algorithm 3 Algorithm to evaluate δ_V

```
procedure COMPUTEDELTAV(Expected output pattern array (EX),Observed output pattern
array for all test runs (OP))
    δV[TotalRuns] ← 0
    CurrentRun ← 0
    while CurrentRun < TotalRuns do
        Index ← 0
        while Index < size(EX) do
            ExpectedOutput ← EX[Index]
            ActualOutput ← OP[Index][CurrentRun]
            Iterator ← 0
            while Iterator < length(ExpectedOutput) do
                if ExpectedOutput.CharAt(Iterator) ≠ ActualOutput.CharAt(Iterator) then
                    δV[CurrentRun].CharAt(Iterator) ← 1
                else
                    δV[CurrentRun].CharAt(Iterator) ← 0
                end if
                Iterator++
            end while
            Index++
        end while
        CurrentRun++
    end while
    δV ← HAMMINGDISTANCE(δV[])
    return δV
end procedure
```

It is expected that the value of δ_V is low in the case of permanent and intermittent faults, as these faults manifest into failures at a fixed set of output pins. In contrast, transient faults do not have a fixed set of outputs that it affects,resulting in a high expected value of δ_V. Figure 4.6 shows actual values of δ_V for a simple circuit (p45k). Other circuits used in the experiment also follow a similar trend.

4.2.4 Diagnostic features

Diagnostic features from section 2.3 are also considered as features for learning algorithm. Diagnostic data also provides information about fault present in circuit. A short summery of fault models and observed behavior for diagnostic parameters is summarized in table 4.2 [HW09].

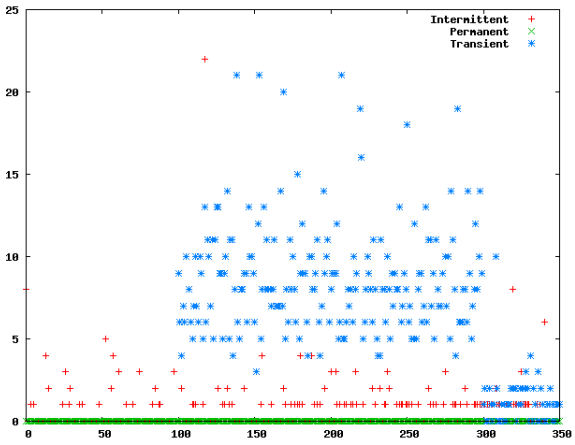


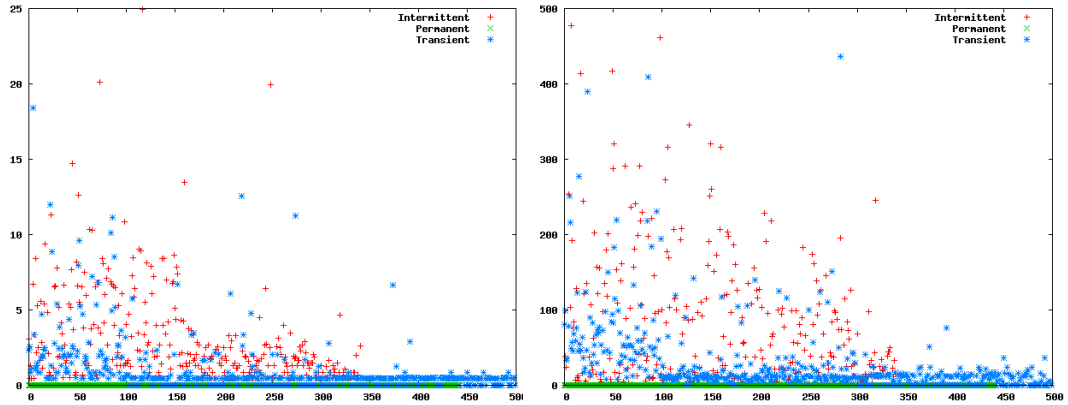
Figure 4.6: Plot of δ_V for p45k

Fault type	σ	ι	τ	γ
Permanent	>0, same sigma values in all of the test runs.	>0, only with transient noise, otherwise 0	>0, only with transient noise, otherwise 0	>0, only for delay fault model, same values in all of the test runs
Intermittent	>0, low as compared to permanent faults	>0, higher as compared to permanent faults	>0, only with transient noise, otherwise 0	0
Transient	>0, varies on every test run	>0, varies on every test run	>0, varies on every test run	>0, varies on every test run

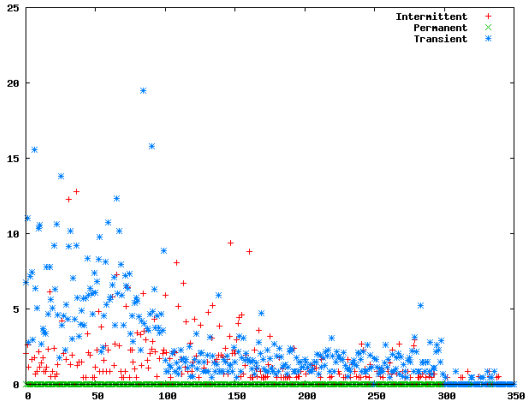
Table 4.2: Expected evidence values for various fault types

In this work, it is assumed that at most only a single intermittent or permanent fault is active in the circuit with or without some transient noise (Refer section 5.1.1). Most of the permanent faults can be modeled as a single stuck-at, single conditional stuck-at or a delay faults. Transient faults can be modeled as conditional stuck-at faults, at multiple fault sites as they do not have a fixed location on chip and some deterministic probability function can be used as a trigging condition. Similarly intermittent faults can be modeled as single conditional stuck-at faults. Hence a combination of values of the parameters can be used to deduce which type of fault might exist on the chip.

Furthermore, the standard deviation of the evidence parameters is expected to be around zero for permanent faults, as once they are detected, then they always can be detected using the same test pattern set, and would result in the same faulty output pattern (except in presence



(a) Sigma (σ) **(b)** Iota (ι)



(c) Tau (τ)

Figure 4.7: Standard deviations for evidence-based parameters for p45k

of transient noise). Hence as standard deviations of evidence based features convey some information about a fault class, they are also considered as features for learning. Figure 4.7 shows actual values of standard deviation of evidence based features for a simple circuit (p45k).

Table 4.3 summarizes the selected features.

4.3 Selection of machine learning algorithm for fault classification

There is a great variety of machine learning algorithms available, and some of the important ones were surveyed in chapter 3. The accuracy of the machine learning classifier depends mainly on the complexity of the feature space and actual training data at hand. Now that

Category	Feature	Symbol
Non-evidence based features	Reproducibility of fault	σ
	Resemblance of erroneous output patterns	δ_H
	Resemblance of erroneous primary outputs	δ_V
Evidence based features	Maximum σ among all test runs	σ
	ι corresponding to maximum σ	ι
	τ corresponding to maximum σ	τ
	γ corresponding to maximum σ	γ
	Standard deviation of σ	SD(σ)
	Standard deviation of ι	SD(ι)
	Standard deviation of τ	SD(τ)
	Standard deviation of γ	SD(γ)

Table 4.3: Summary of selected features

the feature set is known, the selection of the classifier is done with respect to the following factors:

Feature set The feature set is not statistically independent, an important consideration as it violates the prerequisite for Bayes classifier. It can be seen from plots presented in section 4.2, that the feature space has high variance and it is not linearly separable, thus it is not practical to come up with rules to classify faults and hence, the performance of decision trees can be expected to be on the lower side. Also, as the feature space is highly complex, polynomial functions in MLPs might not be sufficient to create an acceptable hypothesis. SVMs, on the other hand can handle a number of different kernel functions and can be expected to create a complex hypothesis, as required in our case.

Sample population Neural networks and decision trees are known to work well with large training sets [DC00, TASF09]. On the other hand, SVMs are shown to be effective with limited and large sized data sets[KH04]. In practice, there is no guarantee that a large training dataset will be available before start of manufacturing.

With this, it becomes clear that SVM can be used as classifier of choice as:

- SVMs can work well with small and medium size data sets, with relatively high accuracy [KH04, Mat14].
- They can handle n-dimensional feature spaces.
- By construction, they can handle complex feature spaces with use of kernel functions.
- Once trained, they are fast to classify data .

Chapter 5

Experimental Setup

With the selection of features and their extraction, and the selection of the machine learning algorithm as SVM, explained in earlier chapter, this chapter explains the experimental setup. This chapter focuses on two topics - the first part being the assumptions for the test setup and the configuration of the sample population as well as the method used to generate the sample population, explained in the section 5.1. The second part, section 5.2, describes the SVM library used for training and classification of faults.

5.1 Generation of sample population

The sample population is required to train and cross-validate the machine learning based classifier (hereon referred to simply as *classifier*). A separate set of data is used to test the accuracy of a classifier.

5.1.1 Assumptions

Following are the assumptions on the sample population and test data:

1. It is assumed that the chips to be classified have displayed the faulty behavior and hence were rejected in earlier test. For further analysis, the test is ran $TestRuns$ times.
2. The chips under consideration are either:
   a) Healthy chips with transient noise or
   b) Affected by a single permanent or intermittent fault, with or without transient noise.

5.1.2 Configuration

For the purpose of running experiments on different circuits, a sample population and a test set for each of circuit types is created with following configuration:

1. A sample population consist of 2500 ($\pm$ 75) of labeled examples. The tolerance of $\pm$ 75 is set as the permanent or intermittent fault instances which did not show any faulty behavior at all at POs, were removed from the sample population.
2. The sample population is equally divided into following five fault categories:
   a) Permanent faults with the label `P`.
   b) Permanent faults along with transient noise (fault rate = 0.001) with the label `P`.
   c) Intermittent faults (fault rate = 0.1, 0.01 0.001) with the label `I`.
   d) Intermittent faults (fault rate = 0.1, 0.01, 0.001) along with transient noise (fault rate = 0.001) with the label `I`.
   e) Transient faults (fault rate = 0.01, 0.001, 0.0001) with the label `T`.
3. Permanent faults are modeled using stuck-at, wired or delay fault models, Intermittents are modeled using the high frequency power droop model, and transient faults are modeled as conditional stuck-at faults at random locations, triggered using a deterministic fault rate.
4. The number of $TestRuns$ to extract features is fixed at 4. This value is set experimentally.
5. A test data has 250 ($\pm$ 15) labeled examples, with the same configuration as that for the sample population.

The evaluation is done on industrial circuits, kindly provided by NXP. The circuits used and the fault coverage test patterns used in the process of generation of sample population are listed in table 5.1.

Circuit	**Fault coverage (%)**
p45k	99.56
p100k	99.56
p141k	98.86
p267k	99.60
p279k	97.89
p295k	99.15

Table 5.1: Fault coverage for circuits used in experiments

5.1.3 Implementation

The sample population is generated as shown in figure 5.1 using an in-house simulation framework called Adaptive Diagnosis of Arbitrary Manifold Artifacts (ADAMA). ADAMA can be used for logic simulation with error injection. Logical representation of a circuit at the gate level and a test pattern set are the required inputs for simulation using ADAMA. A fault description can be provided optionally to inject a fault and analyze its behavior. ADAMA supports all of the fault models that have been considered under configuration in section 5.1.2.

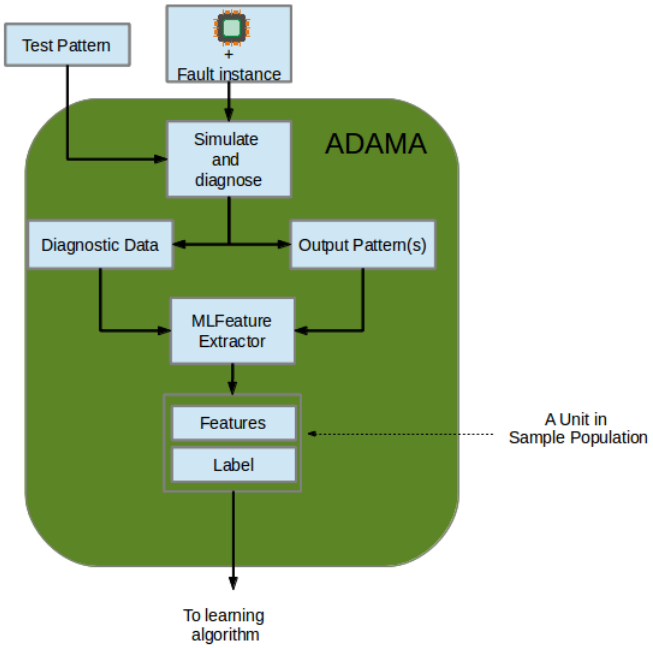


Figure 5.1: Generation of sample population using ADAMA

For the experiments, the ADAMA framework has been extended by adding a task to generate the sample population. The process to generate an example in the sample population is illustrated in figure 5.1.

Simulation The simulator is ran $TestRuns$ times after injecting fault instance in the specified circuit description. Output patterns are stored for further analysis. A reference output pattern, without presence of any fault is also generated and stored.

Diagnosis Diagnosis is ran on each set of output patterns corresponding to each simulation run. The diagnostic data generated is stored for a further analysis.

Feature extraction The non-evidence based features are calculated using the reference output and the set of actual output patterns with the extraction process explained in chapter 4. The evidence based features are extracted using diagnostic data.

The process to generate individual populations of permanent, transient and intermittent faults is described below:

Permanent Faults ADAMA is used to generate fault descriptions for permanent faults. All fault descriptions are put in a file, and then an instance of ADAMA is invoked to generate examples with permanent faults. Permanent faults with transient noise are generated using same file, adding a transient fault instance. After the generation of features is done, examples which did not result in a failure at POs are checked in the output data for permanent faults. Such examples and their corresponding examples in the presence of noise are removed from the sample population.

Intermittent faults First intermittent fault descriptions with random seed values for a location and a specified fault activation rate are generated. Then ADAMA is used to simulate these fault instances without, and then with transient noise. The fault descriptions and corresponding examples in feature files of intermittent faults and intermittent faults with transient noise are removed, where intermittent fault as not active at least for one of the simulation rounds.

Transient faults The circuit description, along with a transient fault instances, with the specified fault rate are generated and then simulated with ADAMA

5.2 Library for SVM based classification - LIBSVM

LIBSVM [CL11] is a popular SVM library, used in a wide variety of applications. It supports linear, polynomial, radial and sigmoid kernel functions and also supports custom user kernels. For experiments, all of the available kernels in the tool have been used to find out accuracy levels for each of them. LIBSVM is coded in C++ and Java and it is available as an open-source software available for free use under a modified BSD license. It supports the multi-class classification using multiple binary classifiers. LIBSVM comes with scripts to train the tool and to find optimal parameters (C and γ) for classification.

5.2.1 Training

The training module in LIBSVM is basically a QP-solver and it tries to solve the minimization problem described in the section 3.3.4 to find out best fitting model for the the SVM classifier. To implement a multi-class class classifier, the trainer uses one-vs-one strategy [KPD90]. Hence to classify data in k-classes, it uses an array of $k(k-1)/2$ classifiers internally [CL11]. Each

classifier uses two classes for the training. The training algorithm algorithm uses v-fold cross-validation (CV), meaning that the data is divided into v subsets and one subset v is tested against the classifier trained with rest of the $v - 1$ subsets, iteratively, until the complete training set is covered [HCL+03]. This results in the training data being tested once completely. The *cross-validation accuracy* is then calculated as a percentage of the data that was correctly classified [HCL+03].

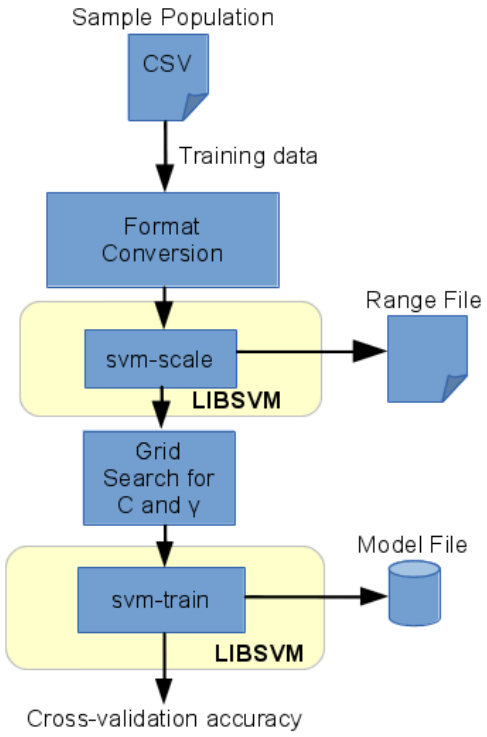


Figure 5.2: Steps to train SVM using LIBSVM

The sequence of steps followed to train the SVM, as shown in figure 5.2 is as follows:

Scaling The scaling module in LIBSVM provides functionality to scale the training data to a user specified input range. The documentation of LIBSVM [HCL+03] suggest the use of range [0,1] for the data containing zero values for some of the features, as in our case. The scaling parameters are stored in a *range file*, which is later used to scale features of future examples or test data.

Training The trainer executable file in LIBSVM is used for training the SVM. The training data and the kernel to be used for training needs to be provided as an input at this stage. It outputs a *model file* to be used for prediction of future examples.

Grid search for C and γ LIBSVM provides a script to find the optimal values of C and γ. The grid search algorithm tries various pairs of C and γ to find a pair with the highest CV accuracy, heuristically. This script internally uses the training module for modeling the classifiers, hence the kernel type for training needs to be specified. A common value of C and γ is searched and applied to all internal classifiers. The *class-weights* can be then applied, which is a simple multiplier to the parameter C of the respective class, to fine-tune the classifier [CL11].

5.2.2 Classification

In the classification of the multi-class data, the classifier uses a voting strategy. Each of the classifiers votes for a given input example. The example is assigned a class with the maximum number of votes. In case that two classes have identical votes, classifier simply choose the class appearing first in the array of storing class names [CL11]. However this is not possible in our case as we use 3 classes, and hence, 3 binary classifiers internally.

Steps to classify data using already trained SVM is shown in figure 5.3. A short description of the these is noted below:

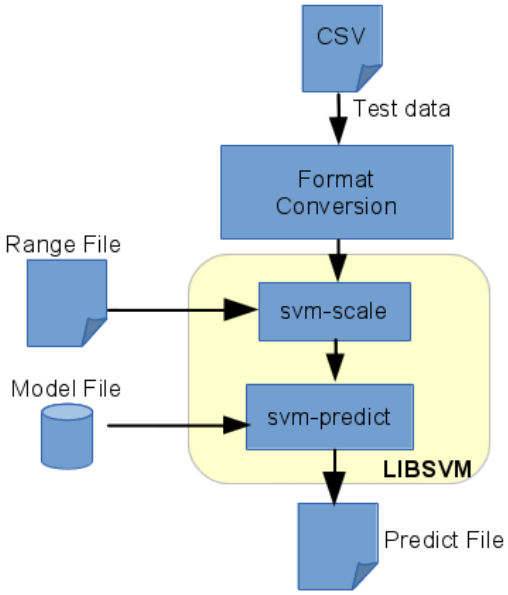


Figure 5.3: Steps for classification of test data with SVM using LIBSVM

Scaling The test data is scaled using same scaling module used for training. The range file generated while scaling the training data is used in this step to scale the test data, instead of manually specifying ranges.

Prediction The prediction tool provided with LIBSVM is used to predict the test data. This tool needs the model file generated during training as an input and it then outputs a *predict file* with predicated labels of the test data. This file is further processed using scripts for the evaluation of accuracy levels of different fault classes.

Chapter 6

Evaluation of results

This chapter presents results of classification using the experimental setup described in chapter 6. The term *classification accuracy* of a class used in this chapter is defined as,

$$\text{classification accuracy}(l_i) = \frac{\text{Samples correctly predicted as } l_i}{\text{Total samples with label } l_i} \times 100 \quad \forall\, l_i \in \{\texttt{P,I,T}\}$$

Where, l is the label of test sample $x = \{\boldsymbol{x}, l\}$.

In the presented evaluation, the cross-validation accuracy defined in the section 5.2.1 is also noted for each set of the experiments, as this is the value of accuracy during the self-validation of much larger training set.

All experiments performed are done after selecting optimal values of C and γ, obtained using grid search. In the first part of this chapter, the classification accuracy for each of the kernel in `LIBSVM` is evaluated, to check their performance for different fault types and in presence of transient noise. The second part tries to explore the possibility of optimizing the classifier for yield or quality of product. This is done with help of experiments carried out by improving accuracy levels of one type of faults at the expense of the others. The third part explores two possibilities - that of using one of the using a single prediction model for predicting all types of circuits, and the second, to use a model trained with a known circuit to classify test data from new, unknown circuits.

6.1 Classification of permanent faults

When a test pattern is able to detect permanent fault in the circuit, running the same pattern again would result in same faulty output every time. Hence it is easy to detect a permanent fault in the circuit. This is also shown to be true in the literature survey presented in the

chapter 2. Hence, in the experimental evaluation the sample population and test data are pre-screened for permanent faults before the classifier is trained.

Consider a the case, where the value of fault reproducibility defined in the section 4.2), $\epsilon = TestRuns$. In our classification problem this can mean,

1. It is a permanent fault.
2. A rare case of highly repetitive intermittent fault, and can be assumed to be "critical".
3. An extremely rare case of a transient fault, as it happened at the same location and for same input test pattern, in all of the test runs.

In the other direction, for permanent faults the value of $\epsilon = TestRuns$ holds true, except a rare possibility that the transient noise affects all of the test patterns, which have resulted in a failure at POs, in at least one of the test runs. This is also an extremely rare possibility. Hence we assume that, a fault is permanent if and only if $\epsilon = TestRuns$.

If we go back to the section 4.2 and observe the plot for ϵ in figure 4.3, we can observe our assumption to be fairly accurate. Hence if we do a slight modification in our experimental setup and already remove faults where $\epsilon = TestRuns$, and classify them as permanent faults, we can achieve up to 100% permanent fault classification, and we would be left with a binary classification problem between transient and intermittent faults.

There are two possibilities to do such pre-screening of permanent faults, one is just to scan all instances for all $\epsilon = TestRuns$ and remove those, or use an ensemble of two SVMs, first one to separate permanents and the other one to classify intermittents and transients. In this evaluation, first option is considered and permanents are removed from sample population, wherever analysis is done without considering permanent faults.

6.2 Classification with different kernels

The first set of experiments consist of evaluation of accuracy levels without assigning class-weights *i.e.* class-weights described in section 5.2.1 are assumed to be {1,1,1} for permanent, intermittent and transient faults respectively. The experiments are repeated for each type of kernel provided by `LIBSVM`.

This set of experiments consist of two rounds each for each kernel, first round considers permanents in the sample population. This is done to show there are indeed kernels which can classify permanent faults with an accuracy up to 100%, in case permanents are to be discarded using an ensemble of 2 different SVM kernels. The second round evaluates classification accuracies, by discarding permanents based on ϵ from the sample population.

6.2.1 Linear Kernel

The accuracy of SVM using a linear kernel is summarized in table 6.1 with considering permanent faults in the sample population, and the same considering permanents are discarded based on ϵ, is summarized in the table 6.2.

Circuit	Accuracy (%)					
	Cross-validation	Permanent		Intermittent		Transient
		w/o noise	with noise	w/o noise	with noise	
p45k	90.14	87.50	87.50	77.50	64.58	93.90
p100k	87.38	100	100	80.95	57.14	95.91
p141k	83.89	98.21	98.21	61.90	57.14	100
p267k	85.12	100	100	55.00	52.50	100
p279k	89.52	98.30	96.61	78.94	52.50	89.79
p295k	89.84	100	100	81.39	65.11	100

Table 6.1: Classification accuracy for linear kernel

In both cases, with and without permanent faults, it can be observed that permanent faults are being relatively accurately classified and the background transient noise seem to have almost no effect on the classification accuracy of permanent faults. Intermittent faults in both cases have relatively low accuracy levels and tend to deteriorate severely in presence of transient noise.

Circuit	Accuracy (%)			
	Cross-validation	Intermittent		Transient
		w/o noise	with noise	
p45k	87.81	72.91	70.08	97.95
p100k	81.43	69.05	59.52	97.96
p141k	84.13	78.57	76.19	100
p267k	77.09	80.00	55.00	100
p279k	84.65	78.94	52.63	91.83
p295k	89.17	90.69	81.39	100

Table 6.2: Classification accuracy for linear kernel, without permanent faults

After removing permanents from the sample population, classification accuracies are observed to improve, with an exception of p100k. However, accuracy levels are observed to increase for intermittents with noise, even in the case of p100k. The classification accuracy of transient faults has also increased. Cross-validation accuracy levels have decreased, as a large chunk of correctly classified data in the earlier case was permanent faults. Comparatively lower accuracy

levels for intermittent faults and almost 100% classification rate for transients suggests that the linear kernel is more biased towards transient faults.

6.2.2 Polynomial Kernel

Circuit	Accuracy (%)					
	Cross-validation	Permanent		Intermittent		Transient
		w/o noise	with noise	w/o noise	with noise	
p45k	90.92	87.50	89.58	68.75	70.83	83.67
p100k	89.70	100	97.82	69.04	66.66	93.87
p141k	87.94	100	100	73.80	71.42	95.00
p267k	87.34	100	96.55	82.50	60.00	85.71
p279k	91.42	98.30	96.55	81.57	73.68	91.83
p295k	90.78	100	98.21	76.09	72.09	95.91

Table 6.3: Classification accuracy for polynomial kernel

The accuracy of SVM using a polynomial kernel is summarized in table 6.3 with considering permanent faults in the sample population, and the same considering permanents are discarded based on ϵ, is summarized in table 6.4.

The accuracy of classification of permanent faults, without noise in case of the polynomial kernel is observed to be even higher than that for linear kernel. For permanents with noise, classification rates are almost similar. Intermittent faults and permanents without noise in first round, are deteriorating in some cases, while improving significantly in others. With noise, however, accuracy of intermittent fault classification is improved. Classification accuracy of transient faults has decreased as compared to the linear kernel.

Circuit	Accuracy (%)			
	Cross-validation	Intermittent		Transient
		w/o noise	with noise	
p45k	87.81	81.25	72.91	87.76
p100k	85.00	78.57	66.66	89.79
p141k	85.57	80.95	76.19	100
p267k	81.38	82.50	72.50	93.87
p279k	85.57	78.94	73.68	93.87
p295k	92.28	93.02	86.04	93.87

Table 6.4: Classification accuracy for polynomial kernel, without permanent faults

In the second round, after removing permanents, a huge improvement in the accuracy of intermittent faults is observed as compared to the first round. The results are also better

than those for the linear kernel. A marginal improvement in the transient fault classification accuracy is also observed as compared to the first round. However, overall figures for transients are lower than those for the linear kernel. These observations suggest that, the polynomial kernel is slightly biased towards intermittents.

6.2.3 RBF Kernel

The accuracy of SVM using a Radial Basis Function (RBF) kernel is summarized in table 6.5 with considering permanent faults in the sample population, and the same considering permanents are discarded based on ϵ, is summarized in table 6.6.

Circuit	Accuracy (%)					
	Cross-validation	Permanent		Intermittent		Transient
		w/o noise	with noise	w/o noise	with noise	
p45k	91.21	100	100	66.66	64.58	91.83
p100k	87.36	100	100	64.28	61.94	93.87
p141k	88.63	100	100	73.80	71.42	93.87
p267k	87.27	100	98.27	80.00	72.50	93.87
p279k	89.52	98.30	96.55	78.92	71.05	93.87
p295k	91.00	100	100	79.06	74.41	95.91

Table 6.5: Classification accuracy for RBF kernel

The classification rates for permanent faults, with and without noise, are higher for RBF kernel as compared to others. This means that RBF can be used to screen permanent faults in pre-screening in an ensemble classifier with 2 kernels. Like in other cases, injection of the transient noise has affected the accuracy of intermittent fault classification significantly. Accuracy figures for both, intermittent faults with and without noise, are lower as compared to previous two kernels. The transient fault classification accuracy shows a marginal improvement over the polynomial kernel, but it is still lower than the linear kernel.

Intermittent fault classification accuracy is higher for smaller circuits, but shows no change for more complex ones. However, for intermittent faults injected with noise, the accuracy is improved as compared to the first round, but is still lower than other kernel types. A slight increase in transient fault classification accuracy is also observed.

A notable observation in case of the RBF kernel is its comparatively higher cross-validation accuracy figures and lower overall classification accuracy when using test data, an indication that the kernel may be overfitting.

Circuit	Accuracy (%)			
	Cross-validation	Intermittent		Transient
		w/o noise	with noise	
p45k	87.56	77.08	70.83	91.83
p100k	83.33	78.57	64.28	89.79
p141k	86.53	83.33	80.95	100
p267k	80.19	82.50	70.00	92.34
p279k	85.64	78.92	76.31	95.91
p295k	85.64	79.06	76.31	95.91

Table 6.6: Classification accuracy for RBF kernel, without permanent faults

6.2.4 Sigmoid Kernel

The accuracy of SVM using a sigmoid kernel is summarized in the table 6.7 with considering permanent faults in the sample population, and the same considering permanents are discarded based on ϵ, is summarized in the table 6.8.

Again, the permanent fault classification rates are is fairly high, which makes the sigmoid kernel another possible candidate for an ensemble classifier using 2 SVM kernels. The intermittent fault classification accuracy is comparatively low (except p100k) despite a high cross validation accuracy. This coupled with the high transient fault classification accuracy indicate that, sigmoid kernel is biased towards transient faults. This has also resulted in a degradation of accuracy levels when intermittents were injected with transient noise.

Circuit	Accuracy (%)					
	Cross-validation	Permanent		Intermittent		Transient
		w/o noise	with noise	w/o noise	with noise	
p45k	89.55	97.50	97.50	77.08	60.41	95.91
p100k	85.08	100	100	95.23	52.38	71.42
p141k	84.17	100	100	57.14	57.14	100
p267k	82.46	100	98.27	42.50	42.50	100
p279k	89.29	98.30	96.55	81.57	52.63	87.76
p295k	85.09	98.21	98.21	69.76	65.11	100

Table 6.7: Classification accuracy for sigmoid kernel

The kernel became more biased towards transients when permanent fault examples were removed from the sample population. The transient fault classification accuracy was observed to increase to near 100%. Also, the intermittent classification accuracy also increased at the same time, with and without the transient noise.

Circuit	Accuracy (%)			
	Cross-validation	Intermittent		Transient
		w/o noise	with noise	
p45k	84.46	68.75	60.41	100
p100k	80.00	81.39	52.38	100
p141k	80.04	78.57	73.80	100
p267k	75.17	75.00	55.00	100
p279k	83.41	81.57	52.63	91.83
p295k	92.28	93.02	86.04	91.83

Table 6.8: Classification accuracy for sigmoid kernel, without permanent faults

This set of experiments show that the relative accuracy levels for the fault classification actually increase if the sample population and the test data is pre-screened for permanents faults. Hence for rest of the evaluation, all experiments are done by considering that the sample population consists of intermittent and transient faults only.

6.2.5 Analysis of intermittent fault classification

The first set of experiments reveal that the classification accuracy for intermittent faults comparatively low as compared to the other fault types. Hence, more detailed analysis is done, which can provide some hints about the improvement of intermittent fault classification accuracy. First, an analysis is done about how fault activation rates might be affecting classification accuracy. Table 6.9 summarizes the result of this analysis and it notes the accuracy results for different intermittent fault activation rates in the test dataset. For this analysis, a subset of circuits were analyzed using all four kernels of `LIBSVM`.

The analysis shows that accuracy levels are fairly high for high fault activation rates of intermittent faults, but go on deteriorating for lower values of fault activation rates. The main reason behind this behavior is that, at lower fault activation rates of 10^{-3}, the behavior of transient faults (activation rates for which are 10^{-2} to 10^{-4}) matches the behavior of intermittents, and the learning algorithm fails to separate them from each other. This suggests that, instances where intermittent failures occurred only once and did not repeat themselves, are not able to convey enough information in their extracted features. Hence no classification algorithm will be able to improve on these figures significantly. To further elaborate this, an accuracy analysis for the same circuits from table 6.9 is done by removing those intermittent faults from the test set, whose reproducibility (ϵ) is less than one. The results of this analysis are presented in table 6.10.

The analysis from table 6.10 shows that, if we remove intermittents with low fault activation rates from the test set, the classification of intermittent faults is accurate up to 100%. The

Circuit	Kernel	Classification accuracy (%)					
		Intermittent faults without noise			Intermittent faults with noise		
		rate = 0.1	rate = 0.01	rate = 0.001	rate = 0.1	rate = 0.01	rate = 0.001
p141k	linear	100	81.25	25.00	100	75.00	25.00
	polynomial	100	81.25	37.5	100	75.00	25.00
	RBF	100	81.25	50.00	100	81.25	37.5
	sigmoid	100	81.25	25.00	100	68.75	25.00
p267k	linear	94.73	82.35	20.00	89.47	35.29	0.00
	polynomial	94.73	88.23	20.00	94.73	64.70	20.00
	RBF	94.73	88.2	20.00	94.73	58.82	20.00
	sigmoid	94.73	70.58	20.00	89.47	35.29	0.00
p295k	linear	100	88.23	77.77	100	88.23	33.33
	polynomial	100	94.11	77.77	100	100	33.33
	RBF	100	94.117	77.77	100	100	33.33
	sigmoid	100	88.23	77.77	100	76.47	22.22

Table 6.9: Accuracy of intermittent fault classification for different fault rates

Circuit	Kernel	Classification accuracy (%)		
		Intermittent		Transient
		w/o noise	with noise	
p141k	linear	100	100	100
	polynomial	100	100	100
	RBF	100	100	100
	sigmoid	100	100	100
p267k	linear	100	100	100
	polynomial	100	100	93.87
	RBF	100	100	95.91
	sigmoid	100	100	100
p295k	linear	100	100	100
	polynomial	100	100	93.87
	RBF	100	100	91.83
	sigmoid	100	100	91.83

Table 6.10: Improvement in accuracy levels after removing intermittents with $\epsilon = 1$

accuracy of transient fault classification remains the same, as the same classifier model and the same test dataset for transient faults is used as earlier experiments.

6.3 Optimization for yield and quality using class weights

Class-weights are used while training the SVM by adding a penalty to samples of particular class, to compensate for the unavailability of sufficient positive examples for that class. `LIBSVM` provides a functionality to adjust individual class weights, to bias classification towards a specified class using `-wi` switch. We can use this functionality to:

- Fine-tune C values for optimal classification. This way we can have a classifier which can separate most of examples in dataset.
- Optimize the classifier to classify one type of faults more accurately than the other.

The first point presents another possibility to improve yield and retaining quality, at the same time. This is done by balancing the training dataset. An unbalanced dataset is a possibility in our case, as we have set a tolerance of $\pm$ 75 on training examples of each class, as explained in the section 5.1.2.

Table 6.11 shows the accuracy values when optimizations mentioned above are performed.

Kernel	Optimization Criterion	Class-weights {I,T}	Classification accuracy (%)		
			Intermittent		Transient
			w/o noise	with noise	
linear	Intermittent	{1.5,1}	100	100	22.44
	Transient	{1,1}	80.00	55.00	100
	All	{1.9,1.8}	72.5	67.5	95.91
polynomial	Intermittent	{1.85,1.05}	97.50	80.00	20.00
	Transient	{1.3,1.95}	82.50	67.50	97.95
	All	{1.9,1.25}	95.00	78.00	77.5
RBF	Intermittent	{1.75,1.15}	95.00	72.50	89.79
	Transient	{1.2,1.3}	72.50	70.00	95.91
	All	{1,1}	82.50	70.00	92.34
sigmoid	Intermittent	{1.7,1.1}	85.00	77.50	65.30
	Transient	{1.9,1.8}	82.5	57.5	100
	All	{1.7,1.75}	82.50	47.00	100

Table 6.11: Accuracy after class-weight optimization for p267k

To find out the set of optimal values of weights in the case of optimization for both intermittent and transient faults at the same time, an exhaustive search of combinations of class-weights for `I` and `T` classes is performed. For experimental purpose, a limited search of region [1,2] with steps of 0.05 for combinations of class-weights is performed. Heat-maps for this search are shown in figure 6.1 for p267k. Using these heat-maps the weight combinations were set up for the analysis presented above.

Optimization for “All” labels is done by considering weights corresponding to the highest cross-validation accuracy. After class-weight optimization, accuracy levels are observed to be improved as compared to the same analysis done in the section 6.1.

6.4 Classification using extrapolation of known training datasets

So far the evaluation of test data is done using a sample population of the same circuit. In this section two other possibilities are considered. First, to use a sample population of a known circuit, to classify the data of an another unknown circuit. Second, to have a single training dataset of all circuits, and use a classifier model built from this to predict example data of a circuit type, whose sample population is already present in this *universal* dataset.

6.4.1 Using single known model for classification

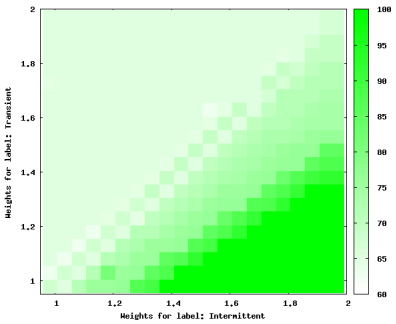
In practical situations, one might not have the necessary training data for a new product beforehand, especially at the start of production. This section explores possibility of using known datasets for unknown circuit types. This set of experiments are done using the training data of p295k to predict test sets of other circuits. p295k is chosen as sample population as it provided with the relatively most accurate results in the first set of experiments. The results obtained are summarized in the table 6.12.

The comparison of the results with those obtained in the section 6.1, shows an increase in accuracy levels. This can be attributed to two factors, a comparatively more balanced dataset of p295k and high cross-validation accuracy levels of this dataset. Hence it is observed that, using a good-quality known dataset, examples from other circuit types can be classified with acceptable accuracy levels.

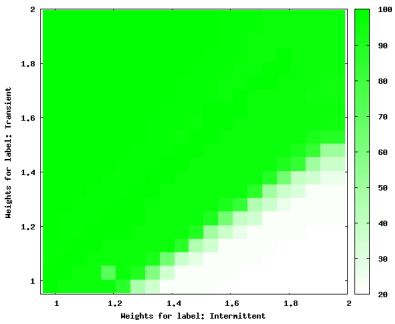
6.4.2 Using a universal training set for classification

In this set of experiments, a single universal sample population is built using sample populations of individual circuits. The aim of this experiment to check a possibility of building an incremental universal dataset, which is able to classify data from any of known circuit types. Table 6.13 summarizes results observed.

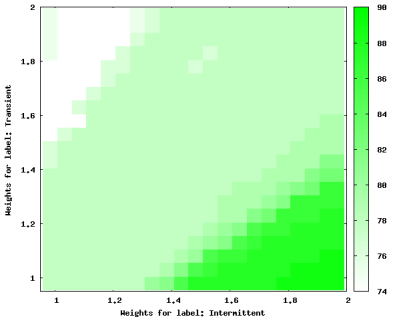
The universal dataset is also able to classify faults fairly accurate. Using polynomial and RBF kernels, accuracy levels of both intermittent and transient fault classification increased considerably, at the same time. This is mainly because of an increase in the sample population providing higher number of positive and negative examples of both classes and eventually resulting in a better fitting of the hypothesis class. A further optimization of accuracy levels, for both or any one of fault types is possible using techniques discussed in section 6.3.



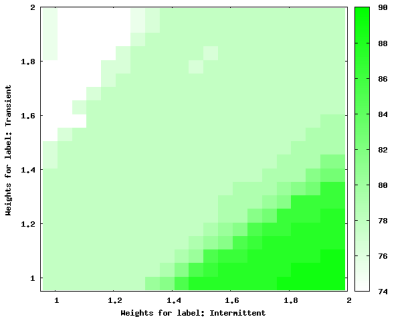
(a) Intermittent fault accuracy, linear kernel



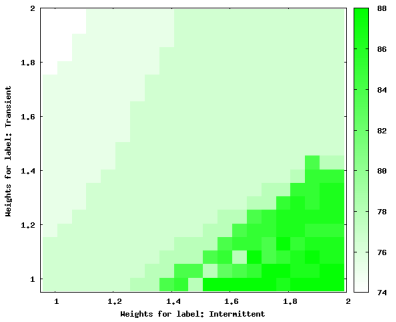
(b) Transient fault accuracy, linear kernel



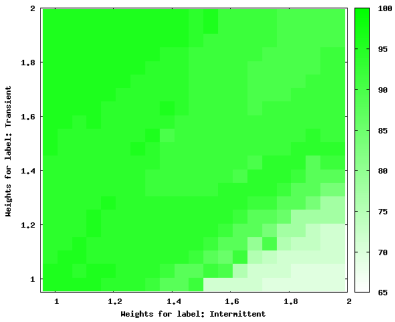
(c) Intermittent fault accuracy, polynomial kernel



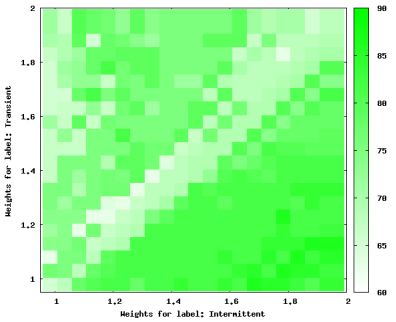
(d) Transient fault accuracy, polynomial kernel



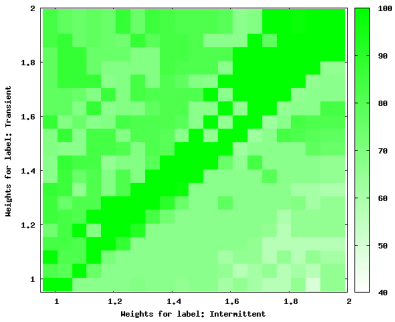
(e) Intermittent fault accuracy, RBF kernel



(f) Transient fault accuracy, RBF kernel



(g) Intermittent fault accuracy, sigmoid kernel



(h) Transient fault accuracy, sigmoid kernel

Figure 6.1: Plots of accuracy by varying class-weights for p256k

Kernel	Cross-validation accuracy (%)	Circuit	Accuracy (%)		
			Intermittent		Transient
			without noise	with noise	
Linear	89.17	p45k	77.08	72.91	95.91
		p100k	71.42	64.28	75.51
		p141k	78.57	73.08	100
		p267k	77.5	62.5	100
		p279k	78.94	57.89	97.95
Polynomial	90.78	p45k	79.16	81.63	75.51
		p100k	78.57	76.19	73.46
		p141k	88.09	80.95	91.83
		p267k	82.5	77.5	87.75
		p279k	78.94	76.31	97.95
RBF	92.28	p45k	79.16	81.25	73.46
		p100k	78.57	73.8	71.42
		p141k	88.09	80.95	97.95
		p267k	82.5	72.5	89.79
		p279k	78.94	71.05	100
Sigmoid	85.64	p45k	79.16	81.25	73.46
		p100k	78.57	73.8	71.42
		p141k	88.09	80.95	97.95
		p267k	82.5	72.5	89.79
		p279k	78.94	71.79	100

Table 6.12: Accuracy by extrapolating training dataset of p295k to test other circuits

Kernel	Cross-validation accuracy (%)	Circuit	Classification accuracy (%)		
			Intermittent		Transient
			without noise	with noise	
Linear	82.84	p45k	66.66	64.58	100
		p100k	66.66	61.9	89.79
		p141k	80.95	76.19	100
		p267k	70	65	95.91
		p279k	71.05	60.52	97.95
		p295k	81.39	79.06	100
Polynomial	86.52	p45k	75	25	95.91
		p100k	98.57	66.66	89.79
		p141k	88.09	78.57	100
		p267k	82.5	67.5	97.95
		p279k	76.31	63.15	100
		p295k	93.02	81.39	95.91
RBF	86.58	p45k	79.16	77.08	89.79
		p100k	78.57	69.04	87.75
		p141k	88.09	78.57	100
		p267k	82.5	70	93.87
		p279k	78.94	68.42	100
		p295k	93.02	86.04	97.95
Sigmoid	83.71	p45k	81.25	64.58	79.59
		p100k	83.33	66.66	83.67
		p141k	88.09	71.42	81.63
		p267k	87.5	72.5	69.38
		p279k	81.57	60.52	79.59
		p295k	97.67	86.04	85.71

Table 6.13: Accuracy using single universal training dataset

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis presented a possibility of using the SVM based machine learning approach for fault classification of semiconductor chips. The chips tested showed moderate to high overall accuracy levels for fault classification. The classification of permanent faults showed no significant issues as they are repeatable and their failures are localized to a set of primary outputs. Separating intermittent faults, when their fault activation rates are in lower range and same as those for transient faults, presents a significant challenge as their fault behavior is similar to each other. However, the results show that intermittent faults, at higher fault activation rates are up to 100% separable from transient faults, even in presence of the background noise.

Different kernels were used for SVM and it can be concluded that, the selection of the kernel primarily depends on the nature of the feature space of the subject sample population. High cross-validation accuracy is a good indicator for accuracy and should be used for selection of the kernel. However, too-high cross-validation accuracy values might indicate that kernel is overfitting, and it should be then tested with a separate test dataset, and this accuracy should also be considered for the kernel selection. Also from the background research for this thesis, it is clear that the selection of features plays a crucial role in efficiency, irrespective of the chosen algorithm and kernel. A well-balanced dataset, containing an equal number of positive and negative examples, is also important in determining the performance of the SVM. The imbalance in the sample population can somewhat be remedied using class-weights.

A set of experiments was also carried out to assess a possibility of using sample population from one known circuit, which showed good accuracy levels, to classify faults on unknown circuits. The experiments show that, this is indeed possible and the accuracy levels were were comparable to, or even better in some cases as compared to those obtained for the same circuits using their own sample population. In a different experiment, a universal training dataset was

also constructed from all known circuits and the result obtained were comparable to those from individual training datasets.

This thesis also proposes a solution were user can optimize the fault classification for yield or for quality. In a practical situation where user wants to focus more the yield of product and not so much on quality, he can make use of class weights to bias the algorithm to reject chips which showed permanent faults and severe intermittent faults. On the other hand, when user wishes to focus more on quality, he can choose to add higher weight for intermittent faults, and algorithm will reject all permanents and most of intermittents.

The motivation of this thesis was to separate critical fault from non-critical ones, to improve yield. The criticality of a fault is an abstract concept, and its definition varies as per the application domain of the product. With use of class weights, user would be able to adjust the yield-quality trade-off.

To conclude, machine learning approach is presented as an effective option for fault classification. This approach does not require the expert knowledge, except during the feature selection phase, as once the feature selection is done, the same features can be used for classification of faults on different circuits, which a certain advantage as compared to rule-based classification approaches, where threshold values need to be adjusted by experts for every individual product and technology. One more benefit is that, once training data is ready, machine learning is automated and can be used as a black-box approach to classification. Moreover, the existing classification model can be made extended by importing sample populations for new circuits, making machine learning adaptive.

7.2 Future work

This works presents a preliminary analysis of using SVM as classifier. One of the reasons for choosing SVM for classification, is its strong mathematical foundations [Vap95, BHW10] and ability to use multiple kernels for classification. A further analysis is possible using different kernels or algorithms for classification. The classification accuracy has further optimization potential, with use of additional features than those suggested in this work.

The experiments which were carried out variety of circuits. Results suggested that using a universal model for classification is possible. It might also be possible that, this approach works for same circuits at different technology node. A more elaborate analysis is required to verify the same.

The experiments were carried out using the number of $TestRuns = 4$. This figure was set experimentally, to facilitate observability of transient and intermittent failures. Work can be done to ensure practicability of this approach, by reducing this number. The sample population was generated by extending the simulation framework ADAMA, using fault models already available in the framework. To make approach more precise, additional fault models can

be added to generate sample population. A more practical solution can be, to use actual manufacturing data from industries, which would be able to replicate failure rates more precisely.

Bibliography

[Agr00] V. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Frontiers in Electronic Testing. Springer, 2000. (Cited on pages 8 and 11)

[AHX+05] M. Agostinelli, J. Hicks, J. Xu, B. Woolery, K. Mistry, K. Zhang, S. Jacobs, J. Jopling, W. Yang, B. Lee, et al. Erratic fluctuations of SRAM cache Vmin at the 90nm process technology node. In *Proceedings of The Electron Devices Meeting, 2005. IEDM Technical Digest.*, pp. 655–658. IEEE, 2005. (Cited on page 8)

[Alp04] E. Alpaydin. *Introduction to machine learning*. MIT press, 2004. (Cited on pages 9, 20 and 24)

[AM08] L. Auria, R. A. Moro. Support vector machines (SVM) as a technique for solvency analysis. Technical report, German Institute for Economic Research, 2008. (Cited on page 29)

[Ana00] Analog Devices Inc. *ADI Reliablity Handbook*, 2000. (Cited on page 13)

[BCDGG00] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Transactions on Computers*, 49(3):230–245, 2000. (Cited on pages 15, 17 and 31)

[BHW10] A. Ben-Hur, J. Weston. A user's guide to support vector machines. In *Data mining techniques for the life sciences*, pp. 223–239. Springer, 2010. (Cited on pages 28 and 63)

[BS04] W. Bartlett, L. Spainhower. Commercial fault tolerance: A tale of two systems. *IEEE Transactions on Dependable and Secure Computing*, 1(1):87–96, 2004. (Cited on page 8)

[BSHH01] T. Bartenstein, D. Sliwinski, D. Heaberlin, L. Huisman. Diagnosing combinational logic designs using the single location at-a-time (SLAT) paradigm. In *Proceedings of The 2013 IEEE International Test Conference (ITC)*, pp. 287–287. 2001. (Cited on page 18)

[C+95] W. Cohen, et al. Fast effective rule induction. In *Proceedings of The 12th International Conference on Machine Learning*, pp. 115–123. 1995. (Cited on page 24)

[CL11] C.-C. Chang, C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. (Cited on pages 28, 45 and 47)

[CMP+00] P. Castillo, J. Merelo, A. Prieto, V. Rivas, G. Romero. G-Prop: Global optimization of multilayer perceptrons using GAs. *Neurocomputing*, 35(1):149–163, 2000. (Cited on page 31)

[Con03] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE micro*, 23(4):14–19, 2003. (Cited on pages 9, 12, 13, 14, 15, 16 and 17)

[Con07a] C. Constantinescu. Impact of intermittent faults on nanocomputing devices. *IEEE/IFIP DSN-2007, Edinburgh, UK*, Supplemental Volume:238–241, 2007. (Cited on pages 8, 13, 14, 15, 16 and 17)

[Con07b] C. Constantinescu. Intermittent faults in VLSI circuits. In *Proceedings of The IEEE Workshop on Silicon Errors in Logic-System Effects*. Citeseer, 2007. (Cited on page 15)

[DC00] R. DeFries, J. C.-W. Chan. Multiple criteria for evaluating machine learning algorithms for land cover classification from satellite data. *Remote Sensing of Environment*, 74(3):503–515, 2000. (Cited on page 41)

[DK09] J. De Kleer. Diagnosing Multiple Persistent and Intermittent Faults. In *IJCAI*, pp. 733–738. 2009. (Cited on page 17)

[DKCP94] A. Dharchoudhury, S.-M. Kang, H. Cha, J. H. Patel. Fast timing simulation of transient faults in digital circuits. In *Proceedings of The 1994 IEEE/ACM international conference on Computer-aided design*, pp. 719–722. IEEE Computer Society Press, 1994. (Cited on page 17)

[DM03] P. E. Dodd, L. W. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on Nuclear Science*, 50(3):583–602, 2003. (Cited on pages 15 and 17)

[EARG13] J. Espinosa, D. Andrés, J.-C. Ruiz, P. Gil. The Challenge of Detection and Diagnosis of Fugacious Hardware Faults in VLSI Designs. In *Dependable Computing*, volume 7869 of *Lecture Notes in Computer Science*, pp. 76–87. Springer Berlin Heidelberg, 2013. (Cited on page 17)

[EK10] K. El-Khatib. Impact of feature reduction on the efficiency of wireless intrusion detection systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(8):1143–1149, 2010. (Cited on page 26)

[GBGG01] J. Gracia, J. C. Baraza, D. Gil, P. Gil. Comparison and application of different VHDL-based fault injection techniques. In *Proceedings of The 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001.*, pp. 233–241. IEEE, 2001. (Cited on page 16)

[GCM08] D. Greene, P. Cunningham, R. Mayer. Unsupervised learning and clustering. In *Machine learning techniques for multimedia*, pp. 51–90. Springer, 2008. (Cited on page 21)

[Gha82] P. Ghate. Electromigration-induced failures in VLSI interconnects. In *Proceedings of The 20th Annual Reliability Physics Symposium, 1982.*, pp. 292–299. IEEE, 1982. (Cited on page 13)

[HCL+03] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003. (Cited on pages 31 and 46)

[HKS03] C. Hawkins, A. Keshavarzi, J. Segura. View from the bottom: nanometer technology AC parametric failures-why, where, and how to detect. In *Proceedings of The 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003*, pp. 267–276. IEEE, 2003. (Cited on page 14)

[HR76] L. Hyafil, R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976. (Cited on page 25)

[HW09] S. Holst, H.-J. Wunderlich. Adaptive debug and diagnosis without fault dictionaries. *Journal of Electronic Testing*, 25(4-5):259–268, 2009. (Cited on pages 5, 15, 18, 19 and 38)

[IR86] R. K. Iyer, D. J. Rossetti. A measurement-based model for workload dependence of CPU errors. *IEEE Transactions on Computers*, 100(6):511–519, 1986. (Cited on page 17)

[IYI90] R. K. Iyer, L. T. Young, P. V. K. Iyer. Automatic recognition of intermittent failures: An experimental study of field data. *IEEE Transactions on Computers*, 39(4):525–537, 1990. (Cited on page 17)

[JD88] A. K. Jain, R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988. (Cited on page 21)

[JL95] G. H. John, P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of The 11th conference on Uncertainty in artificial intelligence*, pp. 338–345. Morgan Kaufmann Publishers Inc., 1995. (Cited on pages 23 and 24)

[KH04] R. Koggalage, S. Halgamuge. Reducing the number of training samples for fast support vector machine classification. *Neural Information Processing-Letters and Reviews*, 2(3):57–65, 2004. (Cited on page 41)

[KLT08] J. Kim, D. X. Le, G. R. Thoma. Naive Bayes Classifier for Extracting Bibliographic Information from Biomedical Online Articles. 2008. (Cited on page 24)

[KM96] J. Khare, W. Maly. *From Contamination to Defects, Faults and Yield Loss: Simulation and Applications*. Frontiers in Electronic Testing. Springer, 1996. (Cited on page 13)

[KM02] T. Kavzoglu, P. Mather. The role of feature selection in artificial neural network applications. *International Journal of Remote Sensing*, 23(15):2919–2937, 2002. (Cited on page 26)

[Kot07] S. B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. In *Proceedings of The 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pp. 3–24. 2007. (Cited on page 31)

[Kot13] S. B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013. (Cited on page 24)

[KP09] K. Kishore, V. Prabhakar. *VLSI Design*. I.K. International Publishing House Pvt. Limited, 2009. (Cited on pages 11, 16 and 17)

[KPD90] S. Knerr, L. Personnaz, G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing*, pp. 41–50. Springer, 1990. (Cited on page 45)

[L+09] T. Lehtonen, et al. *On fault tolerance methods for networks-on-chip*. Ph.D. thesis, Department of Information Technology ,University of Turku, Finland, 2009. (Cited on pages 5, 13 and 14)

[Lar06] E. Larsson. *Introduction to Advanced System-on-Chip Test Design and Optimization*. Frontiers in Electronic Testing. Springer, 2006. (Cited on page 14)

[LS90] T.-T. Y. Lin, D. P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. *IEEE Transactions on Reliability*, 39(4):419–432, 1990. (Cited on page 17)

[Mat14] Supervised Learning (Machine Learning) Workflow and Algorithms From MathWorks Documentation, 2014. URL http://www.mathworks.de/de/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html. (Cited on page 41)

[MF08] S. Marinai, H. Fujisawa. *Machine learning in document analysis and recognition*, volume 90. Springer, 2008. (Cited on page 21)

[Mit97] T. M. Mitchell. *Machine learning*. McGraw Hill, 1997. (Cited on page 24)

[MSTC94] D. Michie, D. J. Spiegelhalter, C. C. Taylor, J. Campbell. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. (Cited on page 31)

[MZS+08] S. Mitra, M. Zhang, N. Seifert, T. Mak, K. S. Kim. Soft error resilient system design through error correction. In *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, pp. 143–156. Springer, 2008. (Cited on page 8)

[NA08] T. T. Nguyen, G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008. (Cited on page 31)

[PCKB07] I. Polian, A. Czutro, S. Kundu, B. Becker. Power droop testing. In *Proceedings of The IEEE International Conference on Computer Design, 2006*, pp. 243–250. IEEE, 2007. (Cited on page 15)

[PLV02] B. Pang, L. Lee, S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of The ACL-02 conference on empirical methods in natural language processing-Volume 10*, pp. 79–86. Association for Computational Linguistics, 2002. (Cited on page 31)

[PSBDG98] M. Pizza, L. Strigini, A. Bondavalli, F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In *Proceedings of The Third IEEE International High-Assurance Systems Engineering Symposium, 1998*, pp. 214–223. IEEE, 1998. (Cited on page 17)

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. (Cited on page 24)

[RHW85] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985. (Cited on page 25)

[SABR04] J. Srinivasan, S. V. Adve, P. Bose, J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of The 2004 International Conference on Dependable Systems and Networks*, pp. 177–186. IEEE, 2004. (Cited on page 8)

[Sav80] J. Savir. Detection of single intermittent faults in sequential circuits. *IEEE Transactions on Computers*, 100(29):673–678, 1980. (Cited on page 17)

[Seb02] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002. (Cited on page 31)

[Sta01] J. H. Stathis. Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits. *IEEE Transactions on Device and Materials Reliability*, 1(1):43–59, 2001. (Cited on page 15)

[TASF09] A. K. Tanwani, J. Afridi, M. Z. Shafiq, M. Farooq. Guidelines to select machine learning scheme for classification of biomedical datasets. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pp. 128–139. Springer, 2009. (Cited on page 41)

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. springer, 1995. (Cited on pages 26, 28 and 63)

[Wan10] L.-C. Wang. Data learning based diagnosis. In *Proceedings of 15th Asia and South Pacific Design Automation Conference (ASP-DAC), 2010*, pp. 247–254. IEEE, 2010. (Cited on page 18)

[WE85] N. H. Weste, K. Eshraghian. Principles of CMOS VLSI design: a systems perspective. *NASA STI/Recon Technical Report A*, 85:47028, 1985. (Cited on page 16)

[WK95] I. A. Wagner, I. Koren. The effect of spot defects on the parametric yield of long interconnection lines. In *Proceedings of 1995 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 1995*, pp. 46–54. IEEE, 1995. (Cited on page 13)

[WZA06] N. Williams, S. Zander, G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16, 2006. (Cited on page 24)

All links were last followed on April 23, 2014.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature