UC Santa Barbara
UC Santa Barbara Electronic Theses and Dissertations

Title
Domain Specific Machine Learning - A No-Free-Lunch Perspective

Permalink
https://escholarship.org/uc/item/2qk3n5gp

Author
Nero, Matthew

Publication Date
2022

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California
Santa Barbara

Domain Specific Machine Learning - A No-Free-Lunch Perspective

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Matthew R. Nero

Committee in charge:

Professor Li-C. Wang, Chair
Professor Margaret Marek-Sadowska
Professor Yuan Xie
Doctor Magdy Abadir

March 2022

The Dissertation of Matthew R. Nero is approved.

Professor Margaret Marek-Sadowska

Professor Yuan Xie

Doctor Magdy Abadir

Professor Li-C. Wang, Committee Chair

March 2022

Domain Specific Machine Learning - A No-Free-Lunch Perspective

Copyright © 2022

by

Matthew R. Nero

To my Oma, Opa, Grandmother, Grandfather and to my Parents.

I have followed you this far with further more to go.

Acknowledgements

We would like to thank National Science Foundation (NSF) for their support of our research. This work was supported in part by NSF Grant No. 2006739 and NSF Grant No. 1618118. We would also like to thank Nimbis Services, Inc./AFRL for their support of our research. Our research would not be possible without these kind supports.

Additionally I would like to thank Greg Creech at GLC Technologies, Inc. for his support and guidance over the last several years. To all those who have acted as counselors to me over the years: Nik Sumikawa, Phil Nigh, Jay Shans and KK Hsieh; thank you so much. Much thanks to my advisor, Li-C Wang, without whom not a word would be written. His influence in this work is felt on every page.

Curriculum Vitæ

Matthew R. Nero

Education

2022 Ph.D. in Electrical and Computer Engineering, University of California, Santa Barbara.

2016 M.S. in Electrical and Computer Engineering, University of California, Santa Barbara.

2014 B.S. in Electrical Engineering, University of California, Santa Barbara.

Publications

- **M. Nero**, C. Shan, L. Wang and N. Sumikawa, "Concept Recognition in Production Yield Data Analytics," *IEEE International Test Conference (ITC)*, 2018, pp. 1-10, doi: 10.1109/TEST.2018.8624714.
- N. Sumikawa, **M. Nero** and L. Wang, "Kernel based clustering for quality improvement and excursion detection," *IEEE International Test Conference (ITC)*, 2017, pp. 1-10, doi: 10.1109/TEST.2017.8242071.
- L. Wang, S. Siatkowski, C. Shan, **M. Nero**, N. Sumikawa and L. Winemberg, "Some considerations on choosing an outlier method for automotive product lines," *IEEE International Test Conference (ITC)*, 2017, pp. 1-10, doi: 10.1109/TEST.2017.8242047.

Awards

- Honorable Mention Paper Award - "Concept Recognition in Production Yield Data Analytics," IEEE International Test Conference (ITC) 2018
- Distinguished Paper - "Kernel based clustering for quality improvement and excursion detection," IEEE International Test Conference (ITC) 2017
- Distinguished Paper - "Some considerations on choosing an outlier method for automotive product lines," IEEE International Test Conference (ITC) 2017
- Outstanding TA Award - 2015, 2016, 2017, 2018

Abstract

Domain Specific Machine Learning - A No-Free-Lunch Perspective

by

Matthew R. Nero

This work explores Domain Specific Machine Learning (DSML) problems that arise from a workflow in a semiconductor company. The focus is on understanding what characteristics make them different from a typical machine learning problem. We first discuss a general setup for a DSML problem and provide two postulates to describe the scope of the DSML considered in this research. We examine the theories of machine learning, including four different schools of thinking for articulation of what machine learning is. We review the No Free Lunch Theory for machine learning and discuss the incompleteness of the four theories in view of the No Free Lunch.

Based on the theoretical understanding, we then point out the essence of DSML. Specifically, a DSML problem considered in this work follows an iterative process. In each iteration, the learning goal is to attain a "surprise" for the next iteration. Essentially, the unpredictability in the data appears in the next iteration represents the value of the learning. From this point of view, we argue that the underlying problem in DSML is local No Free Lunch.

An important consequence of this view is that machine learning will no longer be used in making a decision for the user, as that traditionally used. Instead, it is used to help a data summarization process to facilitate user to make a decision. In this thesis, we investigate two types of applications, outlier screening in production testing and wafer map pattern recognition for yield monitoring and improvement. We use these two applications to illustrate our view of DSML as well as various considerations associated with the view.

Based on experience of implementing a machine learning software solution in a company, we explain the lessons learned for deploying a successful machine learning solution. If we consider traditional machine learning as largely focusing on the learning algorithms, we conclude that DSML is largely focusing on all other aspects trying to leverage the capability of a learning algorithm (if possible). From this angle, DSML is solving a problem complementary to machine learning (Co-ML). In other words, DSML shall not be taken as another direct extension of ML. Instead, it would be more appropriate to view DSML (in the semiconductor industry) as Co-ML.

Contents

Curriculum Vitae vi

Abstract vii

List of Figures xii

List of Tables xvi

1 Introduction 1

1.1 Domain specific machine learning (DSML) 2
1.2 Examples from the semiconductor domain 4
1.2.1 Functional verification coverage improvement 6
1.2.2 Speed-limiting path analysis 9
1.2.3 Layout hot-spot detection 11
1.3 DSML postulates 13
1.3.1 Two postulates 15

2 Learning in View of No Learning 17

2.1 The four schools of machine learning 18
2.1.1 The three components of ML 18
2.1.2 Computation learning theory 19
2.1.3 Statistical learning theory 24
2.1.4 Statistical mechanics of learning 29
2.1.5 Bayesian learning 31
2.1.6 No free lunch theorem 34
2.1.7 Summary 36

3 Under-specification 38

3.1 A brief review of test process 41
3.2 Outlier analysis 44

4 Quest for Certainty **48**
4.1 Taking a step back 49

5 Applicability Checking **52**
5.1 A case study revealing a local NFL situation 54
5.1.1 No Universally Best Method Across CQIs 56
5.1.2 Poor Generalization from YL_{min} to YL'_{min} 57
5.2 Applicability of an outlier method 59
5.2.1 Two properties to check for applicability 59
5.2.2 The basic assumption of an outlier 60
5.2.3 Density estimation vs. outlier transform 60
5.2.4 Outlier methods in test are outlier transforms 61
5.2.5 Concern of lack of consistency across wafers 63
5.2.6 Assessing consistency with variance 63
5.2.7 Assessing justifiability with bias 64
5.3 **Calculating Variance and Bias** 65
5.3.1 Result visualization 65
5.3.2 Best method for a particular wafer 67
5.3.3 Results across tests 68
5.4 **Examples to illustrate Variance and Bias** 69
5.4.1 Simulated examples to illustrate Bias 70
5.5 **Applicability results** 72
5.5.1 Usages of the applicability measure 73
5.6 **Re-visiting results in Section 5.1** 73
5.6.1 Improvement on total yield loss YL'_{min} 74
5.6.2 Improvement on yield loss difference 75
5.6.3 Ideal YL'_{min} for each given YL_{min} 75
5.6.4 Consistency without justifiability 77
5.7 Summary 78

6 A "What-If" Analysis **79**
6.1 Consistent threshold selection 83
6.2 A minimum consistent threshold 85
6.3 Summary 86

7 Wafer Map Pattern Recognition **87**
7.1 Wafer map patterns in yield excursions 90
7.2 Cluster patterns 92
7.2.1 Cluster pattern results 94
7.3 Sample-based pattern detection 95
7.4 General wafer map pattern recognition 96
7.4.1 Generative Adversarial Networks 97
7.4.2 The CNN architectures 98

7.4.3 An example recognizer for a wafer map pattern 100
7.4.4 Generality of a recognizer 102
7.4.5 Developing a set of recognizers 103
7.4.6 Developing a sequence of recognizers 104
7.4.7 Six remaining novel patterns 106
7.4.8 Detecting an issue with a production tool 106
7.4.9 Non-interesting wafer maps missed by the recognizers 107
7.4.10 Generality of the recognizers 107
7.5 Summary 108

8 Experience From Developing A Machine Learning Solution 110
8.1 Manufacturing process 111
8.1.1 Manufacturing stages 112
8.1.2 Manufacturing tools and data 114
8.1.3 Defect data 116
8.2 Lot disposition 117
8.3 Disposition methodology 119
8.4 The disposition software system 124
8.4.1 Summary 129

9 Data Summarization In DSML 131
9.1 Facilitating decision making 132
9.2 A possible DSML solution 133
9.3 A summary window design 134
9.3.1 Notifications 137
9.3.2 Issues 142
9.4 Summary 145

10 Conclusion 147
10.1 A Philosophical Remark About This Work 150

Bibliography 152

List of Figures

1.1 A typical dataset for machine learning. 5
1.2 Simulation environment for functional verification. 6
1.3 The iterative process of applying ML in functional verification. 7
1.4 350 critical paths not predicted by STA. 9
1.5 Raster scan to extract snippets. 11
1.6 Learning to model a simulator. 12
1.7 The iterative process of domain specific machine learning. 13
1.8 DSML: Learning to enhance the dataset. 14

2.1 A Machine Learning framework: The data generator G creates the Data, which the machine learning algorithm ML uses to search the Hypothesis Space and output the learned model h. 18
2.2 Number of samples required for PAC learning as a function of the problem size. 23
2.3 Linear classifier (a) shattering 3 points (b) failing to shatter 4 points. . . 25
2.4 Support vector machine binary classifier using the support vectors, circled in red to define a boundary between the classes. 27
2.5 Generalization error is related to the angle between a weight vector $\vec{W}$ and the true weight vector $\vec{V}$. 30
2.6 Posterior likelihood over μ and σ after seeing (a) the first distribution of probabilities and (b) the second distribution of probabilities. 32

3.1 Stages in the evolution of machine learning (Deep Learning image taken from AlexNet [1]). 39
3.2 The processes to deliver silicon chips to a customer. 41
3.3 Outlier analysis setup. 45
3.4 Perspective and Outliers: (a) The chips (one is a CQI) are outlying (b) The two parts in view of subsequent wafer data no longer seems as outlying. 46
3.5 10PPM yield-loss threshold with (a) Dies from a single wafer (b) 1 Million dies. 47

5.1 Conventional approach to evaluate outlier methods. 53

5.2 YL_{min} for each CQI - product VP. 55
5.3 The corresponding YL'_{min} - product VP. 56
5.4 Core Ideas (a) constrain method A to where it is best (b) No method need be universally best 58
5.5 Two approaches to develop an outlier method. 61
5.6 Consistency with an outlier method $\phi()$. 63
5.7 Variance and Bias in outlier transform. 64
5.8 Visualizing Variance and Bias for one test. 66
5.9 % of wafers a method is best on. 67
5.10 % of tests a method is best with - product VP. 68
5.11 % of tests a method is best with - all products. 68
5.12 Original test value distributions $\mathcal{D}_1, \mathcal{D}_2$. 69
5.13 Effect of DPAT, $KS(DPAT(\mathcal{D}_1), DPAT(\mathcal{D}_2))$. 69
5.14 Effect of LA, $KS(LA(\mathcal{D}_1), LA(\mathcal{D}_2))$. 69
5.15 3-wafer examples, all Normal. 70
5.16 3-wafer example that favors AEC. 71
5.17 3-wafer example that favors LA. 71
5.18 Summary results with applicability thresholds. 72
5.19 Improvement on total yield loss YL'_{min}. 74
5.20 Improvement on yield loss difference $YL'_{min} - YL_{min}$. 75
5.21 Less improvement on total yield loss YL'_{min}. 77

6.1 Results for Test 1. Image taken from [2] 81
6.2 Results for Test 2. Image taken from [2] 81
6.3 Min/Max value plot for Test 1. Image taken from [2] 82
6.4 Min/Max value plot for Test 2. Image taken from [2] 82
6.5 Consistent threshold framework. 84

7.1 Wafer map patterns are used to define the scope of a yield excursion. (a) A wafer map showing the excursion's characteristic wafer map pattern. (b) The occurrence of the pattern over time 91
7.2 (a) Failing dies on a wafer. (b) Kernel density estimate of the failing dies. (c) Cluster formed by thresholding. 93
7.3 (a) Polar coordinates of failure clusters. (b) Grouping failure clusters in the coordinate space. 93
7.4 ML cluster pattern detection results. 94
7.5 Two wafer failure patterns from production data. 95
7.6 Illustration of a GAN and it's training. 97
7.7 The two CNN networks in the GAN: (a) The Discriminator. (b) The Generator. 99
7.8 Five training samples. 100
7.9 Validation samples. 101
7.10 Five wafers among the 25 recognized wafers. 101

7.11 Wafer maps produced by the generator. 101
7.12 Recognized maps from the 2nd and 3rd product lines. 102
7.13 Training, validation, recognized samples for the 1st recognizer. 104
7.14 Training, validation, recognized samples for the 2nd recognizer. 105
7.15 Training, validation, recognized samples for the 3rd recognizer. 105
7.16 Training, validation, recognized samples for the 4th recognizer. 106
7.17 Six novel classes of wafer maps discovered. 106
7.18 Non-interesting wafer maps missed by the recognizers. 107
7.19 Novel patterns discovered on the 3rd product line. 108

8.1 Manufacturing steps in building metal layer 2 (M2) on top of metal layer 1 (M1). Shown in Cross-section. 112
8.2 A manufacturing tool with a central chamber and six side chambers where the actual process takes place. Each chamber will have several sensors for monitoring purposes. 115
8.3 Visual defects identified on a wafer. (a) Particle defect (b) Circuit deformation (c) Defect location on a wafer. 116
8.4 Overview of the disposition methodology. 120
8.5 Wafer failure patterns for a lot. Green dies are passing. (a) Shown in a 5x5 grid. (b) Shown as a stacked wafer plot. 122
8.6 Correlating defect data with failure patterns. 123
8.7 Main interface of the disposition system (sanitized) 125
8.8 (a) Stacked wafer plot for the hotspot element. (b) 5x5 wafer plot for the cluster element. 127
8.9 (a) Reports from the WD system. (b) Disposition history. 128

9.1 Summary window as a dashboard display. The dashboard work-space is shown with a blue border, the cards are shown with a green border and the card menu is shown with a red border. 134
9.2 Data selection for wafer map card. A user can select a time range, the device (product) id and the specific lots. 135
9.3 A Notification interface. The notification list shown with a red border contains all notifications. Clicking on a notification will display the notification report in the main body shown with a blue border. 138
9.4 The Systematic View of the notification report. The graph shows the relatedness of the wafer map patterns. A user can click-and-drag the graph nodes so that the graph can be more easily understood. 139
9.5 A graph layout algorithm can be used to produce graph layouts automatically, but they can be difficult to interpret. Compare this graph laid out by and algorithm to the same graph in Figure 9.4 laid out by a user. Green circles show two areas that are difficult to interpret. 140
9.6 Searching for related wafer map patterns. 141

9.7 The Data View shows all the wafer maps that the monitor was examining for this notification. 142
9.8 An issue manager interface. This interface allows a user to add or remove wafer map patterns and can provide a place for discussing the patterns contained in the issue. 143
9.9 The issue manager notification view. All wafer map patterns for the issue can be seen and 145

List of Tables

2.1 Partially Defined Boolean Function 35
2.2 Permutation Closure of a small function 36

5.1 Data used for the study 55
5.2 YL_{min}, YL'_{min}, Ranking for CQI #10 and CQI #29 57
5.3 # of CQIs a method is best for, based on YL'_{min} 58
5.4 # Variance and Bias from different methods 67
5.5 Applicability result across all CQIs 74
5.6 Examples of ideal YL'_{min} based on YL_{min} 76

Chapter 1

Introduction

"If you would be a real seeker after truth, it is necessary that at least once in your life you doubt, as far as possible, all things"

— René Descartes

The last 10 years has been revolutionary for Machine Learning (ML). It has made in-roads into daily life, to the point it is taken for granted. Captioning videos, on websites such as YouTube, used to requiring a human to transcribe the video manually. Today, speech-to-text has become quite practical, the videos are captioned automatically. Virtual assistants, such as Alexa and Siri are in millions of homes. They turn off and on lights, schedule appointments and answer questions; all with the call 'Alexa ...'. Astoundingly, self-driving cars are a thing. They have gone from sci-fi television shows to pilot programs to the highways. It is difficult to travel any significant distance without seeing a Tesla with it's autopilot feature.

These commonplace examples are possible through the dramatic progress in ML Research. The ImageNet challenge [3] is one such example that pioneered the research in

deep learning. The challenge is to identify a thousand different object classes over a set of 1.4 million images. Each algorithmic competitor is graded on it's ability to correctly identify the objects. In 2010, when the challenge started, the best competitor achieved just under a 30% error rate. Five years later that error rate dropped to under 4%; a rate better than humans.

For the first time ever AlphaGo [4] beat champion Go player Lee Sedol. Learning from example games, it gained knowledge on how to play. This knowledge was further refined through self-play, where AlphaGo played against itself. The game-play shown by AlphaGo was so good Lee Sedol remarked, "Surely, AlphaGo is creative" The next year it was surpassed by AlphaGo Zero [5], which mastered the game only through self-play.

More recently we've seen text generation rise to a level that is nearly indistinguishable from humans. Input a small amount of text into GPT-3 [6] and it can complete it. It will write poetry, it will write Python code, it will write news articles; all of them with high level of proficiency. When human evaluators were asked to determine if a news article came from a human or from GPT-3, they had a difficult time. The evaluators correctly identified the GPT-3 article 48% of the time. It is important to note that GPT-3 is cross domain. It is not that GPT-3 can be trained on poetry to write poetry and then trained on Python code to write Python code. Rather, GPT-3 is trained and then with a poetry input prompt it will write poetry and with a Python code input prompt it will write Python code. It will adapt, on-the-fly, to what is asked of it. Within it's bag of tricks GPT-3 can also answer questions, perform arithmetic and translate languages.

1.1 Domain specific machine learning (DSML)

In light of these successes, in this thesis work we look at some challenges of applying ML to "domain specific" problems and how it is different from a typical problem solved by

today's ML mainstream. We use the term *domain specific ML* (or DSML) to differentiate our subject matter from common ML.

First of all, DSML involves a domain expert (or user) who possesses the domain knowledge for a particular application. The value of a result produced by ML is ultimately judged by the expert. In such a context, the data to be learned on, is domain specific, i.e. specific to the application. Naively, one can think of DSML as simply as applying ML to analyze domain specific data.

Taking such a naive view can lead a domain expert to practice DSML as the following:

1. Collect the data (from the application context)
2. Formulate a ML problem, e.g. a classification problem
3. Select a ML tool
4. Run the tool on the data
5. Evaluate the result

Step 2, formulating the problem dictates what tools can be used in step 3. For example, the expert might have a concern on the sufficiency of labeled data samples. To mitigate the issue, the expert decides to formulate the problem as a *semi-supervised* learning problem rather than a typical *supervised* learning problem. Doing so means that in step 3, the expert will be looking for a semi-supervised learning tool.

Label sample scarcity is a common issue in DSML. However, by formulating the problem as a semi-supervised problem, an implicit assumption for it to work effectively is that most of the unlabeled samples would have behaved statistically similar to those labeled ones. On the contrary, if most of the unlabeled samples behave entirely differently from those labeled samples, then result from semi-supervised learning can be misleading.

This point can be illustrated with a rather simple example. Consider in the labeled dataset, we have a sample $(X; y) = ([x_1, x_2, \ldots, x_n]; y) = ([1, 1, \ldots, 1, 0]; 1)$, where n is the number of *features* to encode the sample, X is the feature vector, and y is the label. Suppose we have an unlabeled sample $[1, 1, \ldots, 1, 1]$ which differs from X only at the last feature value. It is reasonable that a learning algorithm would infer the unlabeled sample also has the label 1, especially if n is large such that there is no information to tell it otherwise. However, for the unlabeled sample in the application context it can well be the case that feature f_n is a deciding factor and when its value switches from 0 to 1, the label switches from 1 to 0.

For example, consider an application context where a hardware system is tested. The features to encode a sample are some features in the system. When a test excites the feature, its feature value is recorded in the sample. The output label is an indicator for pass (such as label 1) or fail (such as label 0). Suppose feature value $f_n = 1$ actually implies a system fail. However, in the labeled data there is no sample with $f_n = 1$ and this information is unknown to the domain expert. Then, there is no reason for any learning technique to infer correctly the label of feature vector $[1, 1, \ldots, 1, 1]$.

The naive view toward DSML reduces DSML to be a tool-oriented question. However, a ML tool has an implicit assumption for it to work. This implicit assumption often is difficult to articulate and hence, obscure from the user. Consequently, taking the naive view can make DSML very much look like a trial-and-error search process, without much a scientific principle to guide the search.

1.2 Examples from the semiconductor domain

In this section, we provide several DSML examples whose application contexts are originated from semiconductor chip design process. Before describing those examples, we

first use a simple matrix dataset view to recap a learning setup.

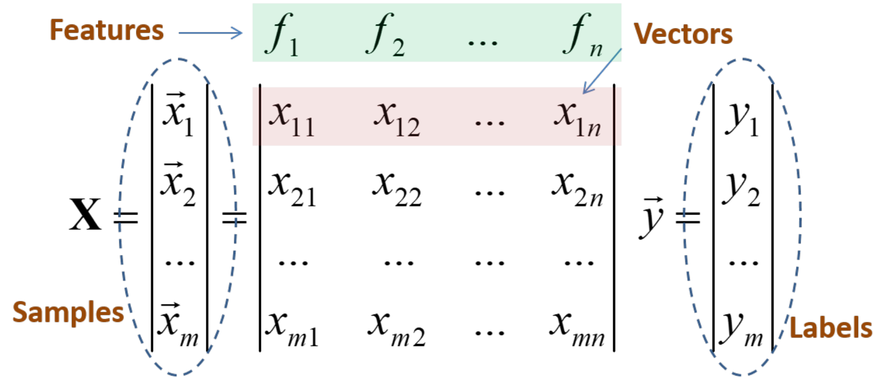

Figure 1.1: A typical dataset for machine learning.

A dataset in this matrix view is shown in Figure 1.1. In supervised learning, the labels are provided. In *unsupervised* learning, the labels are not provided. In semi-supervised learning, labels for a subset of the samples are provided (usually a much smaller subset relative to the number m).

For example, a popular type of analytics is based on two classes of samples. A set of k positive samples with a label of $+1$, these are the samples we're interested in. Then, the rest of $m - k$ samples are negative samples with a label of -1 (or label 0, depending on the context). To learn, a set of n features $f_1, \dots, f_n$ are available to describe each sample. In traditional classification, the goal is to obtain a model based on those features to separate positive samples from the negative ones. In a domain specific application, getting to such a model often is not the end goal. Instead, the goal might be to understand the differences between the positive samples and the negative samples, i.e. what features are relevant to explain their differences. Note that this problem is not necessary the same as the traditional feature importance problem though. This is because different positive samples can be due to different reasons and in many applications, understanding that differences is also important.

In this matrix view, preparing a dataset involves 1) Define samples 2) Assign sample

labels (if needed) 3) Define features, and 4) Calculate feature vectors. Note that this is a *traditional view* of ML. In deep learning, though, the features are supposed to be learned with a neural network from the data.

1.2.1 Functional verification coverage improvement

In the development of a semiconductor design one of the primary concerns is correctness, particularly before manufacturing begins. To insure correctness of a design model (e.g. an RTL model), the task is called *Functional Verification*. Applying ML to improve functional verification was pioneered by the works in [7][8][9][10].

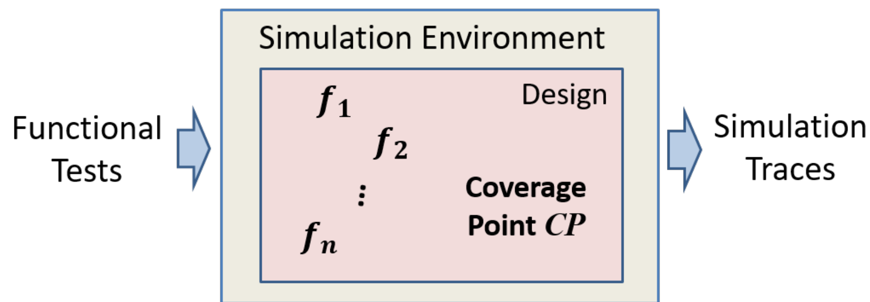

Figure 1.2: Simulation environment for functional verification.

Typically an RTL design model will be written in a Hardware Description Language (HDL) such as Verilog or VHDL. This design model is verified with a simulation environment. The input stimuli are *functional tests*. For a design such as a processor, a System on Chip (SoC) or a Micro-controller, these tests can be just some software programs written in a language like C. The simulation environment includes the support to convert such a C program into input stimuli accepted by the design model. In simulation, *traces* are recorded. These traces are based on selected signals of the design model. For each signal, its simulation waveform (e.g. with a discrete time stamp) is recorded. From traces, *coverage* is calculated. For example, a coverage can be calculated based on a selected set of *coverage points* where each point is represented by a signal name. Figure

1.2 illustrates the overall setup of a simulation-based functional verification environment.

One of the primary applications of ML in the context of functional verification is learning to improve coverage [9]. The thinking is simple: Can we learn from the vast amounts of simulation data and help provide a guidance to modify the functional tests resulting in improved coverage?

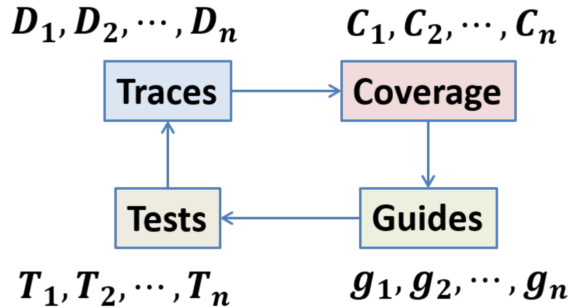

Figure 1.3: The iterative process of applying ML in functional verification.

Note that the process of applying ML to improve functional coverage is iterative, as illustrated in Figure 1.3. In the figure, the test set T_i is simulated to obtain the traces D_i (the "simulation data"). From D_i, the coverage C_i is calculated, which is used to indicate places in the design model for further improvement. Then, the simulation data are analyzed (i.e. "learned") to provide *guides* g_i to improve coverage on those places. Then, g_i is used to create a new test set T_{i+1}. Note that this creation does not necessarily mean modification from T_i only. The overall goal of the process is to maximize the total coverage $C = C_1 \cup C_2 \cup \cdots \cup C_n$. At iteration i, of course all traces accumulated up to that point can be included in the learning. This is why intuitively we say the simulation data is "vast" because the data can be the result of simulation over a long period of time already, e.g. many months.

Now consider in the iterative process what would be considered as an added value from the learning. Without loss of generality, we can say that from C_i to C_{i+1}, the added value is often reflected in a coverage point CP such that $CP \in C_{i+1}$ and $CP \notin$

$C_i \cup C_{i-1} \cup \cdots \cup C_1$. Refer back to Figure 1.2. Suppose covering CP depending on setting some combination of values on a set of signals (features) $f_1, \ldots, f_n$ over some time window. This setting has never occurred in the previous simulation runs and hence, CP is never covered. Our goal is to learn from the simulation data to tell what the setting should be.

Suppose in iteration $i+1$, we succeed in covering CP. Then, we know that the data D_{i+1} contains one piece of information that is not in the simulation data previously, i.e. the setting to cover CP. Suppose we itemize the "information" of each D_i as the set of signal settings where each setting is for covering a coverage point. Then, we can say that the goal of learning at iteration i is learning based on the accumulated data $D_1 \cup \cdots \cup D_i$ is to *add new information to the data.*

The problem of learning to cover CP is similar to the situation discussed in Section 1.1 above: If we have not seen how to cover CP, there is no reason to think that the provided data contains any information that can help. However, this is the case only if we are not allowed to analyze the design model.

Suppose by analyzing the design model we know that to cover CP, we need to set $s1 = 1$ and $s2 = 1$ for two signals $s1, s2$ in the model. Suppose in the data, we do have information how to set $s1 = 1$ and how to set $s2 = 1$, individually. Then, we can focus the learning to extract the two guides. Then, we provide the two guides together as the guide to cover CP.

In practice, it is often the situation that the event $s1 = 1$ and the event $s2 = 1$ rarely occurs in the simulation and hence, the two events never overlap. Suppose each event happens a few times. Then, to learn a guide to cover each event, the dataset would contain only a few positive samples. And because there are many situations where the event is not covered, the dataset would have many negative samples. This means that the dataset would be extremely imbalanced. For example, one could have a dataset

without only 1 positive sample and tens of thousands negative samples. This is no longer a traditional classification problem. Rather, this becomes more like an *outlier analysis* problem (considered in this thesis) where the 1 positive sample is an *outlier* and our goal of learning is to understand why the outlier occurs.

1.2.2 Speed-limiting path analysis

Next, we will look at a different application context. This context often takes place after the design is sent for fabrication to obtain silicon chips. The goal is to analyze those chips to further improve the performance. Typically, the analysis is based on those timing-critical paths observed on the silicon chips [11][12][13][14]. Because they are observed critical paths, they sometime are called *speed-limiting paths* to differentiate them from timing-critical paths determined in the pre-silicon stage by the timing analysis tool such as Static Timing Analyzer (STA).

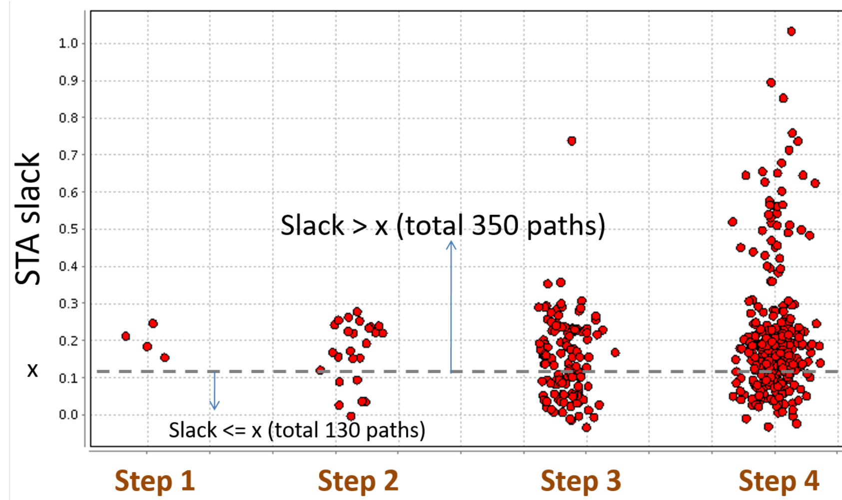

Figure 1.4: 350 critical paths not predicted by STA.

For example, figure 1.4 shows some speed-limiting paths collected from a silicon experiment [11] and their predicted timing slack from a STA. For a path, its timing is

measured on a set of process cores. The measurement is carried out by *frequency stepping* where step 1 has the lowest frequency. Each dot shown in the plot represents a path. For example, at step 1 there are four paths.

In STA, the assumption is that if the path slack is less than a selected value x, then the path is reported as a STA-critical path. In the plot, the value of x is shown, where 350 out of the 480 paths shown in the plot are not reported as STA-critical paths. In fact, based on x, the STA reports 21,589 STA-critical paths and only 130 paths show up in this plot [11].

The learning problem starts with the goal to understand the top 4 paths. Note that these top 4 paths may each be due to a different reason. Hence, we may separate the analysis of 4 paths into four learning problems. For each problem, we have only 1 positive sample. The negative samples can come from those STA-critical paths which do not show up as a speed-limiting paths in the four frequency steps, i.e. there are $21589 - 130 = 21459$ negative samples. To enable the learning, one needs to create a set of features to encode paths. The work in [11] discusses how those features can be created. Note that this feature creation is manual.

Note that the improvement process is iterative. For example, after improving the design to speed up those top speed-limiting paths, other paths will show up as top speed-limiting paths. Then, in the next iteration those new top paths will be the focus. Each iteration is called a *re-spin*. The process can go through a number of re-spins before it is decided a closure has been reached. Because re-spin is expensive, typically the number of re-spins cannot be too large.

From one re-spin to the next, the goal of learning based on the current data is to add new information to the data. This is similar to that in the context of functional verification. In functional verification, we learn to modify the tests to generate data that contains new information. Here, we learn to modify the design to generate data that

contains new information, i.e. new speed-limiting paths. The underlying challenge faced in the learning is also similar, i.e. having a dataset with very few or one positive samples and with many negative samples. As mentioned above, this is more toward an outlier analysis problem than a traditional classification problem.

1.2.3 Layout hot-spot detection

For the above two application contexts, one might think that the underlying issue is in data generation. In functional verification, generating desired data depends on having the corresponding tests. In speed-limiting path analysis, the data depends on the available measurements (which also depends on the available tests). In both contexts, it is not so easy to obtain data samples as desired.

In this section, however, we review another application context to illustrate that the challenge can go beyond data generation. The context is in building a model to replace a simulator. For example, the work in [15] pioneers the study to build a ML model to replace a lithography simulator for the purpose of layout hot-spot detection. In hot-spot detection, the layout is evaluated through a lithography simulator. Hot spots are identified to indicate places with high variability, i.e. places that are more likely to cause an issue in manufacturability.

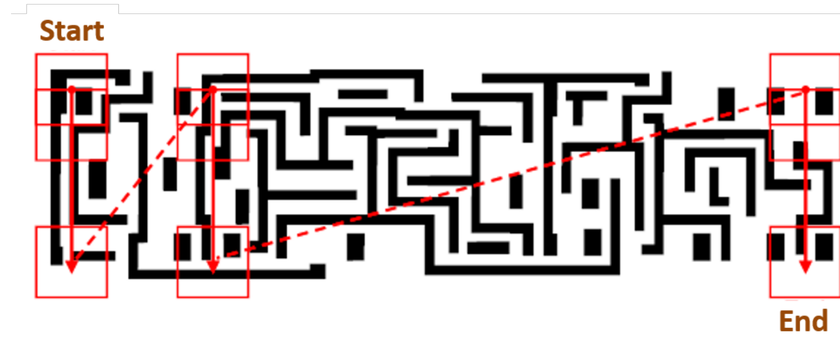

Figure 1.5: Raster scan to extract snippets.

For example, the layout file can be given in the GDSII format. In the evaluation, a raster scan is applied to extract *snippets* from the layout. Figure 1.5 illustrates this raster scan process. Each snippet is a rectangle region of layout. While the figure only shows the layout in 2D, in general a snippet contains a 3D piece of layout. A snippet is evaluated by the simulator to obtain a (variability) score.

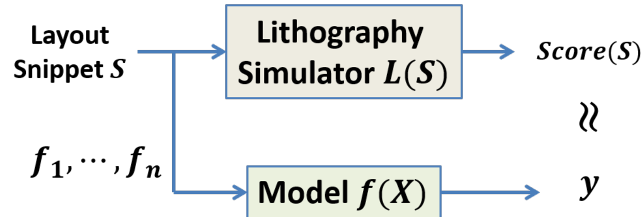

Figure 1.6: Learning to model a simulator.

The learning problem is illustrated in Figure 1.6. To learn a set of features $f_1, \ldots, f_n$ are derived. Using the features, a snippet is encoded into a feature vector. The training data contains feature vectors from many snippets and their variability scores. The learning is to build a model $f()$ such that given a snippet vector X, $f(X)$ is a good approximation of its true variability score. This can be seen as a typical *regression* problem.

In this context, training samples can be obtained by invoking the simulator. Hence, how many samples are available, depends on the simulation time. Of course, it also depends on the available layout examples. Nevertheless, obtaining data samples is not as complicated as the previous two application contexts. Hence, we might think this problem is relatively easier.

In reality, it is not necessary the case. This is because even though data generation is less an issue, obtaining a complete set of features to enable the modeling remains challenging. For example, suppose feature f_{n+1} is necessary to determine the variability for a certain type of snippet and is not included. Then, it is not reasonable to expect that

the learned model will predict well on the type of snippet. In other words, the underlying challenge for the learning problem is not in building a regression model, but in searching for a complete feature set to enable modeling for all types of snippets.

Note that the feature search problem is also inherent in the previous two application contexts. In functional verification, if a required signal to cover CP is not included in the feature set $f_1, \ldots, f_n$, then one cannot expect to completely learn how to cover CP. Also, in speed-limiting path analysis, if a required design feature is missing, then one cannot expect to fully explain a speed-limiting path. Overall, in all three application contexts, the learning problem includes the search for a complete feature set.

1.3 DSML postulates

The three DSML application contexts can be viewed as an iterative search process. Figure 1.7 illustrates such a process. In each iteration, Machine Learning (ML) is applied to bring existing data D to new data D'. Generating the new data D' may involve some action (Act) from the domain expert.

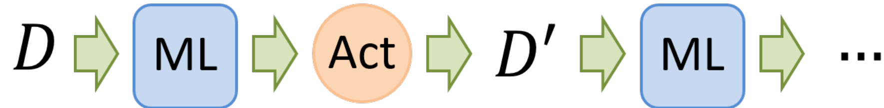

Figure 1.7: The iterative process of domain specific machine learning.

Figure 1.8 then illustrates that the data can be enhanced through two directions, by generating a new sample s_{m+1} and by including a new feature f_{n+1}.

At first glance, getting to a new sample s_{m+1} is not that special. After all, s_{m+1} is an *unseen* sample and the purpose of ML is to predict those unseen samples. However, notice that in our DSML, we are interested in not only predicting the label of an unseen sample, but also being able to generate a desired one. Hence, our DSML naturally includes solving

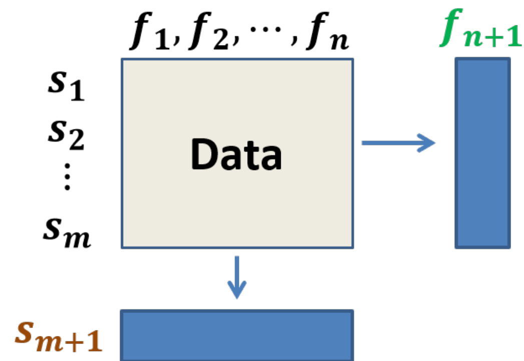

Figure 1.8: DSML: Learning to enhance the dataset.

an *unsupervised* learning problem, in particular a data sample generation problem.

Data sample generation is not a new problem in ML. However, in ML to build a generator model for a class of samples, we need to have training samples of the class to train the generator. In our DSML, as discussed above, the desired s_{m+1} sample can be entirely in a new class. To the extreme, this is almost like we are given many images of dogs and asked to learn a generator to produce an image of a cat. Intuitively, our DSML data generation problem can be much more difficult.

On top of that difficulty, getting to a missing feature f_{n+1} is also challenging, if not more challenging. Intuitively, to know that we need a new feature, we have to first determine that the given set of features $f_1, \ldots, f_n$ are not sufficient for the learning. This means that we need a way to enable us to say that, it is not possible to *learn* based on the given dataset, i.e. "No ML". Evaluating a given dataset to claim no ML remains an open problem. In today's ML, one can evaluate the merit of a model, but to say no ML, one has to use a *threshold* to say one way or the other. As far as we know, there is no systematic way implemented in a tool to determine no ML for a given dataset.

1.3.1 Two postulates

To summarize the discussion in this chapter, we present two postulates to capture the DSML considered in this thesis. Postulate 1 is to capture the essence of DSML, i.e. learning to generate new data samples. From Figure 1.8, we consider two directions of generating the new data samples.

Postulate 1: For a domain-specific machine learning problem involving a data generator G, the goal of the learning based on a dataset D given through G is to modify G to obtain a new dataset D' where D' contains new information that is unknown before.

We add a second postulate to emphasize the baseline for considering the added value from applying DSML in an application context. It is important to note that adding a ML piece to an existing flow (design flow, manufacturing flow, or test flow) is to improve the flow. This improvement can be in terms of throughput, cost, quality, and so on. The added value must be measurable to justify the application of DSML. If not, then even though DSML can produce a good model with high accuracy, it is not useful in practice.

Postulate 2: For a domain-specific machine learning problem, there exists a non-machine-learning approach for solving the same learning problem. Consequently, failure to solve the problem with a machine learning approach is not catastrophic.

Postulate 2 states that DSML is not necessary for the current flow to work, i.e. to be able to produce chips that can be shipped to a customer. Because DSML is not necessary, its added value then needs to be justified. For example, if DSML learns a piece of information that is already well known to the domain expert, then this learning

provides no added value. In contrast, if the information presents a surprise, then it is valuable. In this sense, DSML might be seen as *learning to surprise its user*.

Chapter 2

Learning in View of No Learning

"The world systems are spoken of by the Tathagata as no world systems, therefore they are called world systems"

— The Vajra Prajna Paramita Sutra, translation by Buddhist Text Translation Society

The DSML Postulate 1 in view of Figure 1.8 depicts two directions for the data enhancement learning process: (1) generating a new sample, (2) including a new feature. Given a dataset D consisting of m samples $s_1, \dots, s_m$, we desire the new sample s_{m+1} to be somewhat of a "surprise" to the user. In DSML practice, it is often the case that the more unpredictable s_{m+1} is from the dataset D, the more value it brings to the user. For getting to a new feature, as discussed in Section 1.3, the core of the problem is in the ability to claim no ML. Claiming unpredictability is one way to say there is no ML. However, to say no ML, one has to start with a definition what ML is.

In this chapter, we therefore review four schools of thinking for defining what ML is. Contrasting to these four schools of ML, the No Free Lunch theorems [16] [17] claim the opposite, i.e. there is no ML. Through a better understanding of these schools of

thinking toward ML, our goal is to develop a basis to articulate the essence of DSML given by our postulates. This is important because knowing the impossibilities can help us see the possible and in turn, help us develop a methodology that will work in practice.

2.1 The four schools of machine learning

According to the view provided in [18], there are at least four schools of thought trying to describe what ML is. They are the Computational Learning Theory (CLT) [19], Statistical Learning Theory (SLT) [20], Statistical Mechanics of Learning (SML) [21], and Bayesian Learning [22]. Theories from each school try to offer a precise meaning to ML.

2.1.1 The three components of ML

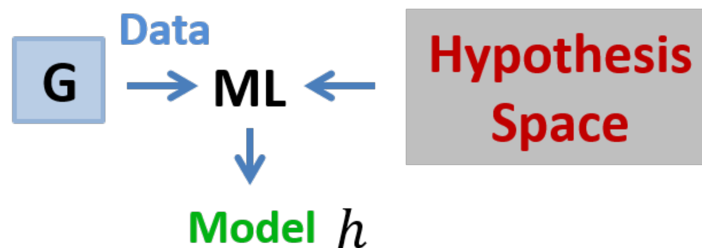

Figure 2.1: A Machine Learning framework: The data generator G creates the Data, which the machine learning algorithm ML uses to search the Hypothesis Space and output the learned model h.

In general there are three basic components associated with ML, as depicted in Figure 2.1. They are: the *data generator* G, the *hypothesis space* H, and the learning algorithm A that builds the model h. For example, the data generator G could be a simulator as we saw with the layout hot-spots context or it could be a semiconductor production line that produces wafer images to be analyzed by a product engineer. A typical assumption for

the data generator is that the generation follows a fixed unknown distribution. Note that in DSML, overall speaking this assumption is not valid because the underlying process generating the data can keep changing.

We use the term *hypothesis space* to mean the set of all possible hypotheses to choose from by an algorithm A to produce the model h. The chosen hypothesis is the model h. Note that this view is not suitable for Bayesian framework for in Bayesian learning, each hypothesis receives a probability. For simplicity of the discussion, we still take the view, with the understanding that when it is used in discussing Bayesian learning, there is no single model output as shown in the figure, unless that model is the *most probable* (MP) model. In ML, the H limits what A can use to fit the data. For example, if H specifies a linear model, A's output will have the form $f(x) = ax + b$. So in this framework an algorithm A will get some data from G and will search the space H for a suitable model h that fits the data.

2.1.2 Computation learning theory

Inspired by computer science's Complexity Theory, Computational Learning Theory (CLT) [19] has a heavy focus on the computational complexity of a learning process. In order to study the complexity, the theory needs a definition for what it means that learning has been accomplished. In other words, it needs a way to characterize the output model so that one can claim the model is good enough. Using that model requirement as the end goal of learning, one can then study the complexity to reach that end goal.

The CLT's definition of learning is based on the Probably Approximately Correct (PAC) learning model [23] which specifies two properties for a model h to satisfy. The first is the *Approximately Correct* property which means that a model must have an error rate bounded by the value ϵ. This is typical in other ML frameworks. The second is the

Probably property which means that the ML algorithm must find that satisfactory model with a probability greater than or equal to $1 - \delta$. Note that the algorithm itself might be deterministic. However, the data sample set produced by the generator is not. Thus this second property intends to reflect that randomness involved in the data.

In CLT, if the representation of the hypothesis space is fixed then it is called *proper*. Given a hypothesis space, if there exists an algorithm to learn a model that satisfies the two properties, then the learning problem is called *PAC learnable*. If there exists an efficient algorithm in view of the complexity, then it is called *efficient PAC learnable*. With these terminologies defined, one can then study a learning problem in view of them. For example, a learning problem can be proven as efficient PAC learnable. For another problem, one can prove that the problem is computationally hard.

In CLT, the focus of study is on learning a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$. This is because otherwise, it would be difficult to connect the study to well established complexity theory, such as the notion of NP-hard. One of the efficient PAC learnable problems is learning a monomial. A monomial has the form $l_1 \wedge \ldots \wedge l_k$ where each l_i is a *literal*, i.e. a Boolean variable or its complement. Given n Boolean variables, the hypothesis space therefore includes all $3^n - 1$ possible monomials. This is because for a monomial it can be the case that a variable appears in it's positive form (x_i), in it's negative form ($\bar{x_i}$), or it does not appear at all.

It is efficiently PAC learnable, as proved with the Algorithm depicted in 1. The algorithm works on a dataset where a positive sample s is the one with $f(s) = 1$ and a negative sample t is the one with $f(t) = 0$. Keep in mind that $f()$ is a Boolean function, and in this case f can be represented by a monomial.

The intuition behind the algorithm is that if a literal is in a positive sample then the inverse of the literal cannot be in the monomial we are looking for. For any two unique positive samples there must be at least one variable that appears as a positive

in one sample and as a negative literal in the other sample. In that case the monomial cannot contain the variable and that variable gets eliminated. Thus, the essence of the algorithm is to use positive samples to filter out those monomials that are not possible to be the answer, i.e. do not fit the data. Note that the negative samples are not used in the algorithm because they have much less filtering power than the positive samples.

Algorithm 2.1: Learning a monomial

```
   Input: m training samples
   Output: Hypothesis h
 1 h ← {x_1, x̄_1, ..., x_n, x̄_n};
 2 foreach positive sample s_i do
 3 |   if s_i contains x_j then
 4 |   |   remove x̄_j
 5 |   end
 6 |   if s_i contains x̄_j then
 7 |   |   remove x_j
 8 |   end
 9 end
```

The essence of the algorithm is therefore in how fast positive samples can filter out monomials that do not fit. The algorithm does not have a full control on the speed of this filtering process because it has no control on the data generation process. To analyze complexity, one either takes an assumption on the generation process or makes the worst-case assumption, i.e. the generation can follow any distribution. The worst-case assumption is usually taken in CLT.

After all the positive samples have been evaluated the learned monomial h can have more literals than the true monomial. Generally speaking h will have the form $a_1 \wedge \ldots \wedge a_l \wedge b_1 \wedge \ldots \wedge b_k$ where the a_i literals are in the true monomial and the b_i literals are not. These b_i's are called bad literals and are the source of the error. The PAC analysis can just focus on these bad literals. The strategy is to first show the condition under which the bad literals will not hurt accuracy beyond ϵ and then second to show the condition

where Algorithm 1 will eliminate all those bad literals that will hurt the accuracy with probability $(1 - \delta)$

To show that the error rate is below ϵ we define the set of events A_i which is the event when a positive sample contains $\bar{b_i}$. The error rate for h is the probability of the union of theses events. We can bound the error rate as in Equation 2.1 for some worst-case probability p.

$$Prob(\cup_{i=1}^{k} A_i) \leq \sum_{i=1}^{k} Prob(A_i) \leq \sum_{i=1}^{k} p \leq kp \leq 2np \tag{2.1}$$

Since this error rate should be less than ϵ we can infer that $p < \frac{\epsilon}{2n}$. This tells us that if the worst case probability of activating a bad literal is below $\frac{\epsilon}{2n}$ then the error rate will be below ϵ.

Next we show that bad literals with activation probability above this limit will be missed with probability less than δ. We define event B_i as the event that literal b_i is not filtered out after m samples. The probability that one of these literals is not filtered out is the probability of the union of these events.

$$\begin{aligned} Prob(\cup_{i=1}^{k} B_i) &\leq \sum_{i=1}^{k} Prob(B_i) \leq \sum_{i=1}^{k} (1 - p_i)^m \\ &\leq \sum_{i=1}^{k} \left(1 - \frac{\epsilon}{2n}\right)^m \leq 2n(e^{-\frac{\epsilon}{2n}})^m = 2ne^{-m\frac{\epsilon}{2n}} \end{aligned} \tag{2.2}$$

Which follows the same basic bounding approach as the previous equation. Setting this bound as less than δ we can derive a minimum for the number of samples.

$$m > \frac{2n}{\epsilon} \left(ln(2n) + ln(1/\delta) \right) \tag{2.3}$$

Which shows that if the number of samples m is above this value then the learned monomial will have an error rate less than ϵ with probability $(1 - \delta)$, proving that this

problem is Efficiently PAC Learnable.

For Equation 2.3 with $\epsilon = 0.001$ and $\delta = 0.01$ Figure 2.2 we show the minimum number of samples m as a function of the number of Boolean variables. Even for moderately size problems the number of samples is in the millions. This illustration shows that the bound might be loose.

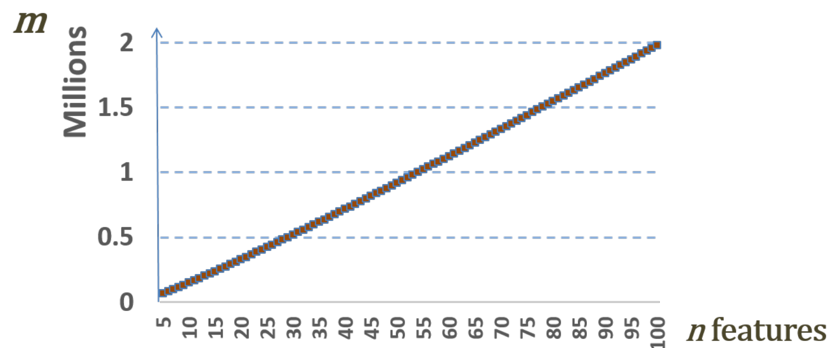

Figure 2.2: Number of samples required for PAC learning as a function of the problem size.

The PAC model is often used to show that a learning problem is computationally hard. For example, learning a 3-term DNF (disjunctive norm form) is NP-hard. A 3-term DNF can be thought of as having three monomials. Additionally, PAC learnable does not promise that the true answer will be learned. This is because the output model only needs to have a low enough error rate.

In CLT, the true function to be learned is assumed to be included in the hypothesis space to begin with. Thus, the assumption is that the hypothesis space contains the true answer. For this reason, its definition of learning is not based on the first principle. PAC learnable only transfers the problem of learning from data to the problem of learning a good hypothesis space. In traditional ML view, this issue is mitigated by assuming a rather complex hypothesis space to ensure that the space most likely will contain the answer. However, doing so exacerbates the *over-fitting* problem.

2.1.3 Statistical learning theory

Of course, one of the downsides for CLT is that the theories mostly deal with Boolean functions. Statistical Learning Theory (SLT) [20] is the counter part of CLT, which deals with real functions. However, once we move from Boolean functions to real functions, it becomes difficult to talk about complexity in view of traditional computational complexity. For this reason, in SLT a new complexity notion is introduced. This notion is based on the famous *VC Dimension* (or *VC Entropy*) idea.

While CLT focuses on computational complexity, SLT is interested in the power of the hypothesis space and the complexity of the resulting learned model. Unlike CLT which makes the assumption that the true model is in the hypothesis space, SLT suggests that the complexity of the hypothesis space should grow according to the complexity seen in the data provided. In that sense, the learning is reached when this growth reaches a convergence point. Finding the conditions to say about this convergence point is therefore in the essence of SLT.

One of the core concepts in SLT is the so-called Vapnik–Chervonenkis (VC) Dimension. It is defined as given a hypothesis space H and a set of m samples, the maximum number of ways to assign labels to those samples by a hypothesis. For example, if H is complex enough, this maximum number will be 2^m assuming labels are binary. The overall assignment process is called “shattering”.

Given a hypothesis space, one can also looks at its complexity from the perspective of fitting a dataset. In that sense, the VC dimension is the maximum complexity of the dataset that the hypothesis space can provide an answer that has a perfect fit to the data, i.e. zero error rate. From this perspective we can see that the VC Dimension is measuring the power of the hypothesis space. However, this measure can be more easily reflected on a given dataset. In practice it is quite difficult, if not impossible, to measure

the VC Dimension directly on a given hypothesis space. Rather, given a dataset, one measures how complex a hypothesis space is needed in order to fit the dataset. This idea is the essence in developing the famous Support Vector Machine (SVM) approach.

To illustrate the VC Dimension, consider a 2D binary classification problem with the restriction that no 3 points can be collinear. We will choose the set of all linear classifiers that take the form $f(x_0, x_1) = sgn(w_0x_0 + w_1x_1 + c)$. It is fairly easy to show that for $m = 1$ and $m = 2$ that a linear classifier can be found. For $m = 3$ as shown in Figure 2.3 (a), there are two cases: 1) all samples have the same label 2) Two samples have the same label. In case 1 it is as trivial as $m = 1$. For case 2 since all 3 points cannot be collinear we can always draw the line between the two samples and pick one parallel to it and between the two sets. When we consider $m = 4$ as Figure 2.3 (b) shows the linear classifier can fail to find a zero-error model. Therefore the VC dimension for this hypothesis space is 3.

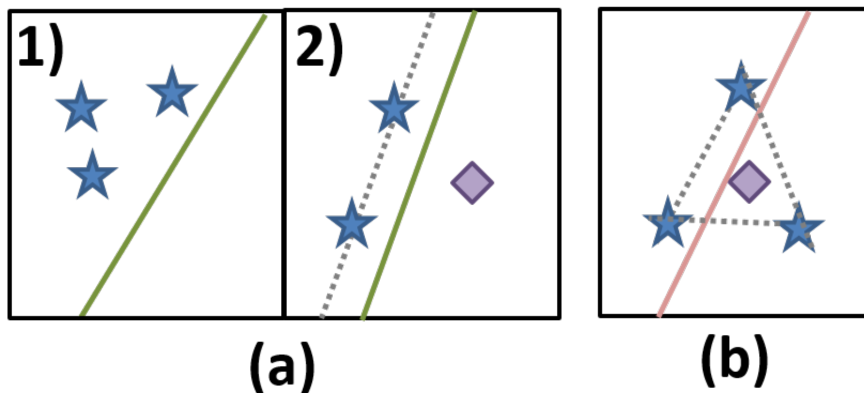

Figure 2.3: Linear classifier (a) shattering 3 points (b) failing to shatter 4 points.

If there are more than 3 samples in a dataset it cannot be guaranteed that the true model is in the 2D linear classifier hypothesis space. To guarantee the true model is in the hypothesis space we should insure the VC Dimension of the hypothesis space is at least as large as our dataset. Since a dataset can be arbitrarily large, usually a hypothesis

space with infinite VC Dimension is selected. With so much power in the hypothesis space it raises the question of consistency, is it possible to converge onto the true model with enough samples. It was shown in [20] by Vapnik that if Equation 2.4 holds, the necessary and sufficient condition for consistency has been met.

$$\lim_{m \to \infty} \frac{VCEntropy(m)}{m} = 0 \tag{2.4}$$

Here the $VCEntropy(m)$ can be interpreted as measuring the average complexity of all sets of m samples. As long as the complexity asymptotically grows slower than the number of samples it is possible to converge onto the true model. $VCEntropy$ is a theoretical concept, used to show consistency is possible. In practice, no method is known to be able to calculate this quantity. Rather, we can see the idea in the quintessential SLT algorithm: the SVM.

For a binary classification problem the SVM model takes the form of

$$M(\vec{x}) = sgn\left(b + \sum_{i=1}^{m} \alpha_i < \Phi(\vec{x}), \Phi(\vec{x_i}) > \right)$$

Where $\vec{x_i}$ are the samples from the ML dataset. The quantity $< \Phi(\vec{x}), \Phi(\vec{x_i}) >$, is called the kernel function and often denoted as $K(\vec{x}, \vec{x_i})$. In this form it is more apparent what the kernel function is doing. First the vectors are mapped with $\Phi : X^n \to X^{n'}$ to a target vector space. This mapping can be linear but is typically non-linear. The dot product $< \cdot, \cdot >$ then measures the similarity of these samples in the target vector space. Intuitively this setup makes sense, the closer a point $\vec{x}$ is to a labeled sample $\vec{x_i}$, the more influence that $\vec{x_i}$ should have on the predicted label of $\vec{x}$.

Training the model is done by minimizing $\mathcal{E} = error + \epsilon \sum \alpha_i$ through quadratic optimization, where ϵ is a regularization constant. Once trained some of the α_i's will have a zero value or near zero, these and their associated x_i vectors will have little affect

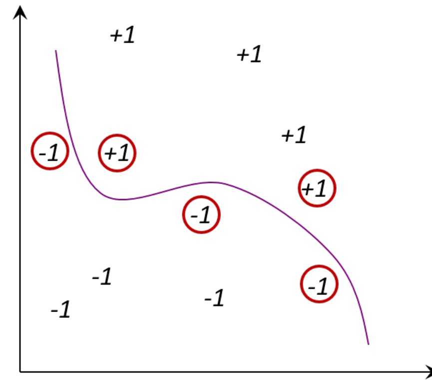

Figure 2.4: Support vector machine binary classifier using the support vectors, circled in red to define a boundary between the classes.

on the model's prediction and can be removed from the model. The remaining α_i's are used in the model and their associated x_i's are called the support vectors. The support vectors, the red circles in Figure 2.4, are the closest vectors to the boundary between the $+1$ and -1 classes, they sit like sentinels keeping the boundary out of their territory. They are also an approximate measure of model complexity. The more complicated the boundary, the more support vectors are needed.

It can be interpreted that minimizing $\mathcal{E}$ is following the Occam's razor principle: 'All else being equal, the simplest explanation is to be preferred'. Finding the minimal complexity model to fit the data is the principle of Structural Risk Minimization (SRM) and minimizing $\mathcal{E}$ reflects that principle by considering the noises in the data. By insuring the $\sum \alpha_i$ term is minimized the model will be 'simpler'. However, minimizing this value will also allow some of the samples to be labeled incorrectly by this simpler model. The value of the regularization constant controls the trade-off between simplicity and training accuracy. If the constant is large, then a simpler model with a higher error rate will be generated and vice versa. The practical justification for allowing some error in favor of a simpler model is that the ML dataset has some noise and the training errors are a result of the noise.

Another interpretation that is applied to the support vectors is that they are related

to the $VCEntropy$ and consistency. First, only the support vectors are needed to train the SVM model. If every sample from the dataset except the support vectors were removed the learned model would be the same. The addition of the other samples does not increase the model complexity. Now consider the case when new training samples are introduced and a new SVM model is learned. If the number of support vectors stays the same, the complexity of the data has stayed the same. If the number of support vectors goes up, new behavior in the data has been identified and the complexity of the data has gone up. In parallel reasoning to Equation 2.4, if the value of $\frac{\#SV}{m}$ goes down as the number of samples increases, it is a good sign telling that the training is converging.

It should be noted that using support vectors is just one way to measure the complexity of a dataset. VC Dimension, on the other hand, is a more general concept. The example provided above is only to help understand what VC Dimension concept intends to capture. The example is based on a linear model in the input space. In general, SLT also does not define learning from the first principle because it leaves room for how the VC Dimension is measured.

For example, given a n-input Boolean function, the maximum hypothesis space is the one with all possible functions, i.e. with size 2^{2^n}. When a dataset of m unique samples are given, it is easy to see that there are 2^m ways to shatter the dataset and with the maximum hypothesis space, all 2^m possible ways are possible (if $m \leq 2^n$). On the other hand, if $m > 2^n$, we know for sure that the dataset contains some repeated sample(s). Thus, with a Boolean space we know that as the number of samples continues to grow, eventually we will be able to "learn" the true function. However, this naive learning is nothing but "seeing" the true function because we have seen all possible samples.

While the intuition is easily seen with a Boolean space, it becomes difficult to see in a real functional space. That is why a central idea to prove the VC theory is to involve this notion called ϵ-net [20] which in a sense, is to "discretize" the continuous

space so that intuitions found in the Boolean space can be carried over to the real space. However, assuming such an ϵ-net also means that the learning depends on the assumption. Even though SLT does not ultimately prove that learning exists, its idea of growing the complexity of the hypothesis space according to the complexity seen in the data remains fundamental to the practical implementation of ML.

2.1.4 Statistical mechanics of learning

Statistical Mechanics of Learning (SML) [21] is the physics view of learning borrowing concepts used in analyzing thermodynamics. One of the core goals in SML is identifying those *self-averaging* quantities in the learning problem. A quantity is self-averaging if as the degrees of freedom move to infinity the distribution of possible values shrinks in variance to the mean like a Dirac Delta function. This self-averaging characteristic makes the typical value of the distribution the same as the average value. An example is the velocity of gas particles in a closed container. When there are few particles, interactions between particles are rare and the velocities can be dispersed quite far from the mean. As the number of particles (degrees of freedom) increases, collisions will occur more and more frequently and the velocities will become more and more concentrated around the mean. The velocities of each particle can be referred to as the micro-states and the average velocity can be referred to as the macro-state.

If a self-averaging quantity can be identified in a ML problem and related to the fitness of the learned model, then something can be said of the learnability of that ML problem. A Perceptron binary classifier model makes a good example for illustrating the thought process. The Perceptron model can be stated as $f(\vec{x};\vec{W}) = sgn(<\vec{W},\vec{x}>)$, where $\vec{x}$ is a n-dimensional vector to be labeled, $\vec{W}$ is an n-dimensional vector of weights to be learned and $<\cdot,\cdot>$ is the dot product. The goal is to find the true weight vector

$\vec{V}$. The vectors $\vec{W}$ and $\vec{V}$ can be projected into a phase space as illustrated in Figure 2.5 where the generalization error is related to the angle between the two vectors.

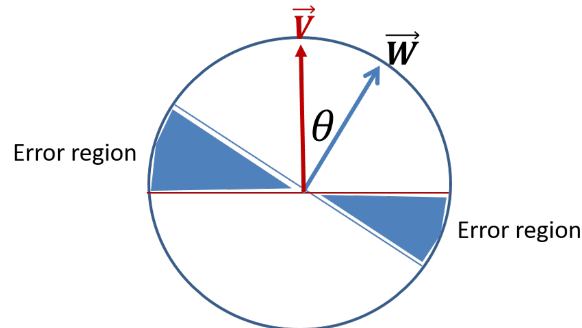

Figure 2.5: Generalization error is related to the angle between a weight vector $\vec{W}$ and the true weight vector $\vec{V}$.

To connect this to learning, the *version space* is defined as the set of all $\vec{W}$'s that correctly label all points in the training dataset. An answer $\vec{W}$ is selected from the version space at random, and this is called the typical model. Its error is called the typical error. This selection from the version space at random is a learning strategy called Gibb's learning. Analysis based on Gibb's learning is focused on the typical performance of a compatible model.

From the SML perspective, the weight values of the $\vec{W}$ vector are the micro-states and the generalization error is the macro-state. This is also based on the asymptotic view that the number of weights is approaching infinity. For a given generalization error there can be many $\vec{W}$'s that will have that particular error, we can define that as the volume $\Omega(\epsilon)$ in the phase space shown in Figure 2.5. We can also define the entropy of that space as $ln\Omega(\epsilon)$. It can be shown [21] that the neither the generalization error nor the volume is self-averaging, but the entropy is. So, as the length n of the weight vector approaches infinity, with an proportional increase in the number of training samples, the entropy of the version space will have the self-averaging property and the typical $\vec{W}$ of the version space will match with the true answer $\vec{V}$.

While the SML provides an alternative view to the learning process, like the previous two schools, it also does not prove learning from the first principle. SML is more like a theory to interpret a known learning process from the thermodynamics perspective. Notice that it makes an additional assumption in the analysis, i.e. the length n of the weight vector approaching infinity. This is similar to growing the complexity of the model in SLT. However, in SLT this complexity only grows as necessary to fit the data. In SML, it begins with the infinite complexity to obtain the self-averaging property.

2.1.5 Bayesian learning

The three schools discussed so far try to claim learning exists by providing a definition of what learning is. However, none provides a theory of learning from the first principle. Bayesian learning [22], on the other hand, does not try to define when learning takes place (i.e. when we know we have learned the "true" answer). Instead, it only tries to describe what a learning process should be. For this reason, Bayesian learning provides a more realistic view to ML.

In Bayesian learning, the learning process is captured in how to update the current knowledge in the face of new evidence. Given data d we are interested in the conditional probability for model h, $P(h|d)$, referred to as the posterior. Before the learning, some knowledge about the choice of h is known and captured as $P(h)$, called the prior. Bayes rule tells us that we need to multiply our prior by the likelihood of the data occurring if the model h was the true model, $P(d|h)$, and then normalize by the likelihood of the data $P(d)$ also called the evidence. This is formalized in Equation 2.5. Notice how there is no indication of a 'correct' model or that a model can be learned within some bound of error. Bayesian learning shows only one step in the learning process journey.

$$P(h|d) = \frac{P(d|h)P(h)}{P(d)} \tag{2.5}$$

Of course, Bayesian learning is more than just Bayes' Rule, it is a set of techniques used to do machine learning in a Bayesian way. Bayes' Rule is a very general description and most of the work is in providing a value for each quantity. For example, we can use Bayesian learning to fit a Probability Density Function (PDF) to a set of data. At first we assume that the data fits a Normal distribution $\mathcal{N}_{\mu,\sigma}$ and we denote this hypothesis space as $\mathcal{N}$. We can rewrite Bayes' Rule to match the problem and our assumptions as $P(\mu, \sigma, \mathcal{N}|d) = \frac{P(d|\mu,\sigma,\mathcal{N})P(\mu,\sigma,\mathcal{N})}{P(d)}$. Initially we will take a uniform distribution for $P(\mu, \sigma, \mathcal{N})$. The likelihood then will be $P(d|\mu, \sigma, \mathcal{N}) = \prod_{x \in d} \mathcal{N}_{\mu,\sigma}(x)$ and the evidence will be the integral $\int_{\mu,\sigma,\mathcal{N}} P(d|\mu, \sigma, \mathcal{N})$.

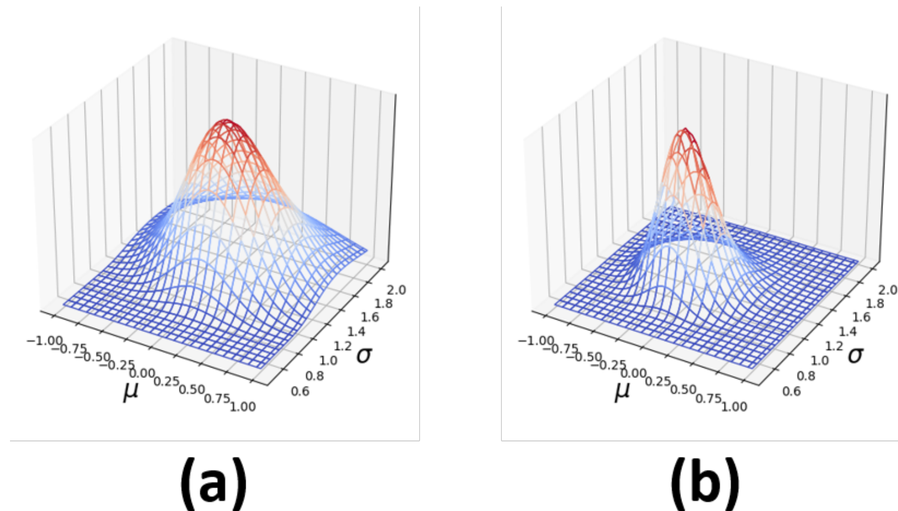

Figure 2.6: Posterior likelihood over μ and σ after seeing (a) the first distribution of probabilities and (b) the second distribution of probabilities.

For an initial set of 10 points generated from a standard normal distribution and assuming a uniform prior, we can calculate the posterior distribution over the set of μ's and σ's as shown in Figure 2.6 (a), remember we are assuming that the probability the model belongs to the set of normal distributions is 1. It is important to understand that

the posterior in Figure 2.6 (a) does not indicate the "true" answer. The peak value is around $\mu = 0.06$ and $\sigma = 1.26$. Instead the posterior is encoding what has been learned from the data so far and indicating what is likely.

If another set of 10 points is generated, then a new update can be performed. Instead of assuming a uniform prior, the values from Figure 2.6 (a) can be used. So the posterior from the first dataset becomes the prior for this new dataset. Figure 2.6 (b) shows the new posterior. In this case the posterior is more concentrated, and the peak value is around $\mu = -0.10$ and $\sigma = 1.08$. While it is nice that the second Bayesian posterior is converging to the right answer, we should focus on the perspective. Bayesian learning is about increasing the understanding across the entire hypothesis space. Whether the posterior likelihood gets more or less concentrated depends on the data and the prior. The posterior represents the best understanding given the available data and that understanding should be updated as new data comes in.

Throughout the previous example it was assumed the hypothesis space was the set of all normal distributions and the posterior likelihood was computed over the parameters μ and σ. This is one level of the analysis, since Bayesian learning is a general framework we can move up a level and compare likelihoods between different hypothesis spaces. For example, if it was proposed that the data might have come from a Cauchy distribution $\mathcal{C}_{x_0,\gamma}$ with hypothesis space $\mathcal{C}$ the values $P(\mathcal{C}|d)$ and $P(\mathcal{N}|d)$ can be compared. For simplicity's sake we can assume that $P(\mathcal{N}) = P(\mathcal{C})$ and then derive the equation:

$$\mathcal{L} = \frac{P(\mathcal{N}|d)}{P(\mathcal{C}|d)} = \frac{P(d|\mathcal{N})}{P(d|\mathcal{C})} = \frac{\int_{\mu,\sigma} P(d|\mu,\sigma,\mathcal{N})}{\int_{x_0,\gamma} P(d|x_0,\gamma,\mathcal{C})}$$

which is just the ratio of likelihood functions integrated over the parameter values. For an $\mathcal{L} > 1$ the evidence favors that a normal distribution generated the data and for a $\mathcal{L} < 1$ the evidence favors a Cauchy distribution. Once again we can see the Bayesian

approach is trying to use all available information to answer the problem at hand.

2.1.6 No free lunch theorem

In 1995 the No Free Lunch (NFL) theorems [16][17] were proposed by David Wolpert and William Macready. They questioned the foundations of the four schools, by indicating that learning, in general cannot happen. The theorems can be summarized by the result [24]:

> For all possible performance measures, no search/learning algorithm is better than another when its performance is averaged over *all* possible discrete functions

The implications of NFL are that our favorite learning algorithm is 'just as good' as our least favorite learning algorithm. The best-fit algorithm is the same as a purely random algorithm. The best-fit algorithm is the same as the worst-fit algorithm. In other words after formulating our problem, picking the algorithm that is least likely to succeed can be the correct choice.

There are generally two reactions to the NFL result: 1) NFL is not a big deal and 2) Domain-specific knowledge needs to be used. On the side of position 1), the set of all possible discrete functions has almost nothing to do with real world, practical problems. This set is infinitely large and is mostly composed of incompressible (random) functions. In other words, the representation of the function is not significantly smaller than enumerating all input-output pairs. For any real world problem, this set is unusable. Position 2) holds that there is no general-purpose algorithm. Instead, the best algorithm needs to be tailored for a specific domain and it should utilize knowledge about that domain.

To gain intuition on the NFL theorem we can consider a partially defined Boolean function such as in Table 2.1 where only 4 of the output values are given and the next four an unknown. There are 2^4 different functions that could complete the table and without some prior assumptions, there is no way to distinguish between them. Even if a new value is given, there are still 2^3 functions that would be compatible, the core problem has not changed. The only way to determine the function is to wait until all the values in the function have been given, at which point the problem can no longer offer anything to learn. Nothing about the first four values can reveal anything about the next four values.

Table 2.1: Partially Defined Boolean Function

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

For this problem, we may wish to use some assumptions. For example if we prefer to use "simplicity" in choosing the best model, then $f = 1$ (a tautology) is a simplest model with a perfect fit to the data. We could also propose that the function must be a monomial and find that $f = \bar{x}$. In some cases these assumptions might be justifiable. In others, they might not. The NFL result is telling us that with no assumptions about the learning, there is no ML algorithm that can learn.

The objection that NFL can be safely ignored because it only applies to an infinite set had some merit. However, the NFL theorem was "sharpened" in 2001 [25] to show that NFL can apply to finite sets as well: if the set of functions is closed under permutation,

then NFL applies.

To illustrate the sharpened NFL, consider a simple example [25]: a set of bijective functions $f : \{1, 2, 3\} \rightarrow \{A, B, C\}$. We can describe a single function as $< A, B, C >$ where $f(1) = A$ etc. Permuting this function until no new functions can be generated yields Table 2.2 which is the permutation closure of our first function. If we are given two samples from the function such as $< B, C, ? >$ we automatically know this is function 4, since we are only considering a set of bijective functions. However, if we are given only one sample such as $< A, ?, ? >$ there are 2 possible functions. This is regardless of which samples we are given ($f(1)$, $f(2)$ or $f(3)$), the permutation closure guarantees that no matter which sample is given there will always be two functions that fit that sample.

Table 2.2: Permutation Closure of a small function

1)	$< A, B, C >$
2)	$< A, C, B >$
3)	$< B, A, C >$
4)	$< B, C, A >$
5)	$< C, A, B >$
6)	$< C, B, A >$

On the other hand if we omit function 6 from the problem then the functions are not closed under permutation and there is a way to do better. A function where $f(1) = C$ must be function 5 and $f(2) = B$ must be function 1 and $f(3) = A$ must be function 4. Any algorithm that can take advantage of this information could do better than one that does not.

2.1.7 Summary

Among the four schools reviewed in this chapter, we consider Bayesian learning to be the most realistic. As the other three schools all try to define when learning happens, their weaknesses in doing so are revealed in view of the NFL theorems. Bayesian learning,

on the other hand, only tries to describe the learning process. It does not try to tell us when the learning should end. It tells us what can be done in each step of a learning process and how our assumption affects the outcome.

In the case of problems like that shown in Table 2.1, the situation is more toward the impossibility to learn. In essence, the situation is that for an unseen sample x, if you have not seen its value y, you simply do not know the value. We call such a situation a *Local NFL* (L-NFL) because the no free lunch situation is local to the specific situation represented by the given samples. In a DSML context, L-NFL can happen when we extract a dataset to be analyzed.

Recall that earlier we discussed the point that in DSML, the value of a new sample is proportional to its unpredictability from the existing samples. In other words, this is almost saying that the value of doing DSML is to show there is no ML on a given dataset. From the surface, it seems that ultimately we will be running into a dead end, and essentially there is no DSML, because DSML is based on no ML.

However, this pessimistic view might be true only if we are trying to solve a DSML problem entirely. For example in Table 2.1, if our goal is to provide a learning tool to solve a learning problem like that, no matter what learning tool we provide, the tool will not be good enough because of the L-NFL. On the other hand, if we take a step back and only try to provide a tool that evaluates the outcome of an assumption provided by a user, then it becomes more feasible. The former is like following the first three schools of thinking which aim to define what learning is. The later is like following the Bayesian thinking which only tries to evaluate the outcome based on an assumption and the data. Such a tool can provide "what-if" analytics to the data. It is not a ML tool in the traditional sense. It is a tool that assists the user to do learning themselves. From another perspective, it is a tool that helps user understand what the data can provide, i.e. more like a data inspection tool than a data learning tool.

Chapter 3

Under-specification

"There's no sense in being precise when you don't even know what you're talking about."

— John von Neumann

As stated before, we consider a DSML process that iteratively expands the dataset with "surprise" samples, moving from dataset D to dataset D'. These "surprise" samples are the "value" aimed at by the DSML process. The process continues until a convergence point is reached, for example when no additional surprise can be discovered over a period of time with effort spent. In each iteration of a DSML process, one can view the learning problem as a problem with an *under-specified* dataset. However, the underlying question is less about finding the best fitting model for the given dataset. Rather, the question is more about learning from the dataset to help a user to see that the given dataset is indeed under-specified. In this chapter, we will take this under-specification view to further illustrate a DSML process. In particular, we focus on an important problem in test data analytics, namely the outlier analysis problem to illustrate our points. Outlier analysis perhaps is among the simplest problems considered in our DSML paradigm.

However, studying it carefully does reveal important considerations for DSML.

The context we will focus on in the remainder of this thesis is semiconductor chip production. In chip production, we can consider two separate processes, the manufacturing process that produces the chips and the testing process that screens out defective chips. Tremendous amounts of data are collected in these processes. Applying machine learning on those data has been a hot topic of research for years [26][27]. Machine learning can roughly be divided into two stages, the early stage where learning is based on a given *feature space* and the more recent stage of deep learning [1][28] where features are automatically learned. We call the early stage *traditional ML*. Figure 3.1 illustrates these two stages.

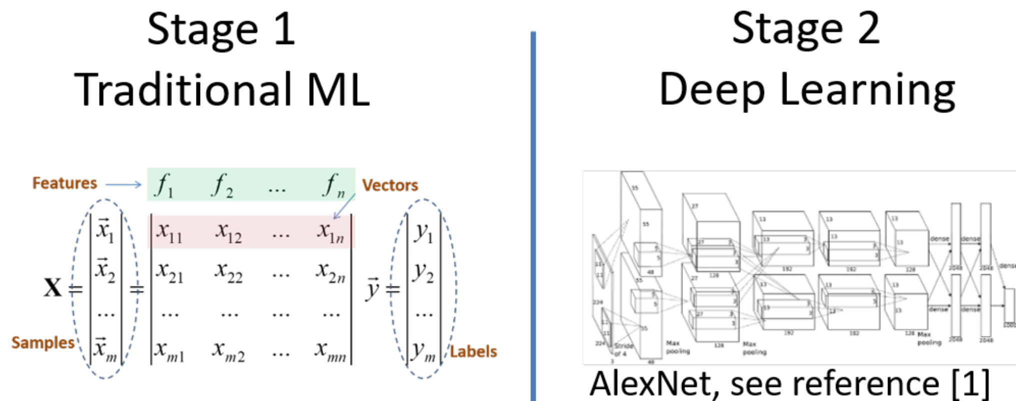

Figure 3.1: Stages in the evolution of machine learning (Deep Learning image taken from AlexNet [1]).

In view of traditional ML, our work focuses on the outlier analysis problem. The problem can be seen as representing an unsupervised learning problem. For example, in Figure 3.1 each sample is a chip. Each feature is a test. Each feature value is a test value based on the particular test. There is no label vector $\vec{y}$. The problem is learning a model to decide which samples are outliers.

An outlier model can be used directly to screen out potentially defective chips, i.e.

an outlier is considered potentially defective. This is the most common use of outlier analysis in testing. In addition, there are other applications where outlier analysis is involved. For example, in *customer return analysis* [29], a customer return is given. A customer return is a chip that passes all tests but fails at customer site. One common approach in the analysis of customer return is to search for an outlier model (which is not used previously as a defect screening model) that screens the customer return as an outlier [29]. In this context, the customer return is a "surprise" and the goal is to use the existing data to explain the surprise.

In *burn-in* reduction [30], outlier models are used to model chips that fail in the burn-in test stage. The intention is to apply those models before burn-in so that only outliers identified by the models go through the burn-in. Those inliers do not need to go through the burn-in. The goal is to save the burn-in cost which is expensive in terms of both equipment cost and test time. In this context, outlier models are used as a predictive models to predict the subset of chips will not fail by the burn-in testing, i.e. those "strong" chips. Due to its wide applicability in test data analytics, outlier analysis is therefore carefully studied in our work. In our study, we also use it as an example to illustrate various important considerations in a DSML process.

In view of deep learning in Figure 3.1 above, we consider Wafer Map Pattern Recognition (WMPR) [31] as our focus of application. The problem can be stated as the following: A sequence of wafer maps $w_1, \ldots, w_n$ are coming in for analysis. Our goal is to identify groups of wafer maps that show some abnormal patterns. Wafer maps with similar patterns are put into the same group. The goal of this classification is to help identify places (e.g. which tools or which steps) in the manufacturing process that might have an issue. We will discuss WMPR in more detail in Chapter 7.

The reason we focus on WMPR is because it is similar to a pattern recognition problem which had observed tremendous successes with deep learning neural networks.

WMPR is also an important problem considered in both manufacturing process and test process. From the manufacturing process perspective, people want to use it to improve the process. From the test process perspective, people want to identify the problems so that they can convey the problems back to the manufacturing organization. Note that WMPR can be seen from an outlier analysis viewpoint as well. For example, in each week a batch of wafer maps are provided and one of the underlying questions is: Do we see any new pattern which we had not seen before? In other words, people are interested in knowing if there is a new issue from the manufacturing process. In this view, what we are looking for is an outlier wafer map.

3.1 A brief review of test process

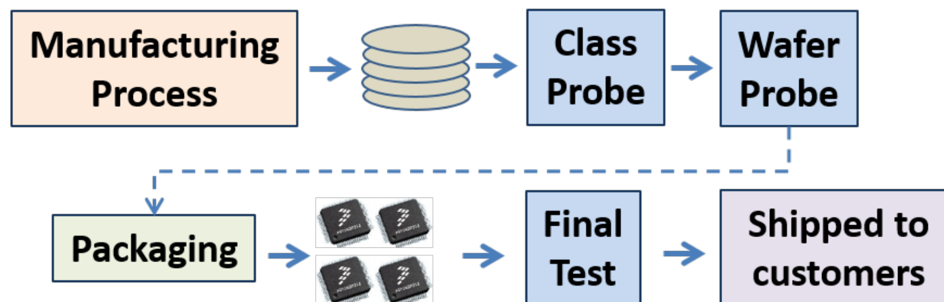

Figure 3.2: The processes to deliver silicon chips to a customer.

We show in Figure 3.2 the major testing stages in silicon chip production from manufacturing to being shipped to the customer. The first box represents the complex manufacturing process. In Chapter 8, we will discuss in more detail what types of data can be collected in the process. In this process, blank silicon wafers enter and upon exiting they will have complete silicon designs built. Wafers are processed in groups of 25 called *lots*. Each wafer will have multiple copies of a single design. Depending on the size of the design a wafer can have as little as 20 or as many as 1000. Within silicon production each

copy of the design is referred to as a *die*, which can be understood as an un-packaged, un-cut chip on a wafer.

After manufacturing, the fist testing stage is called *Class Probe*, or in the industry it can also be commonly referred to as *E-Test*, *Scribe-Line Test*, or *Scribe Test*. The stage measures (or in test terminology "probe") some parameters of basic electronic circuits inserted between dies, or the scribe lines. The circuits can be single transistors, or wire segments and the class-probe tests are usually measuring resistance, capacitance, etc. Class Probe testing is primarily designed by the manufacturing team. It provides a set of tests to measure the health of the manufacturing process.

After Class Probe, the next stage is Wafer Probe. It is here that actual testing of the silicon chips starts. There are usually at least several hundred tests performed at Wafer Probe. Wafer probe is the primary stage used to screen out defective dies due to manufacturing defects. After this stage, good dies are sent for packaging. Note that if there were a burn-in stage, it would take place after the packaging (not shown in the figure). After packaging, packaged chips are tested again in Final Test stage. Then, chips that pass Final Test will be shipped to the customer.

Over the production lifetime of a design from the first wafer produced to discontinuation it will pass through three phases: 1) First Silicon, 2) Ramp-up and 3) Mass Production. First Silicon is the very beginning of production only a few lots will be manufactured in this phase. None of the dies from First Silicon will make it into a commercial part, they are primarily for assessing the manufacturability of the design. At this point the design may exhibit issues requiring design modifications, new masks and a re-spin. There is a strong focus on configuring the manufacturing process to produce dies of sufficient quality that they could be shipped to the customer. Additionally the manufacturing team will need to hit a *yield* target where the yield is the number of good dies as a percent of dies manufactured. During this phase the Wafer Probe and Final Test

test sets are created and over the next 2 phases will be refined. The First Silicon phase is the most volatile of the production phases. The design, the manufacturing process and the testing are constantly undergoing significant changes.

Once usable parts can be produced in small quantities and product begins to be shipped to customers, the design has entered the Ramp-up phase where the manufacturing volume is increased gradually. The Ramp-up phase is focused on increasing the yield. As the manufacturing volume is increased more data becomes available so some of the more difficult to identify yield limiters can be found and the rarer manufacturing defects identified. At this phase the design is fixed and all the changes are being made to the manufacturing process and the test sets.

After the desired production volume is reached and the manufacturing process and test sets have stabilized the design has reached the final phase: Mass Production. Here changes will continue to be made in the manufacturing process and also in the test sets, but the pace is much slower. Mass Production is the least volatile of the three phases. Changes in this phase are usually related to small increases in yield.

A design must complete each phase before moving to the next. We cannot begin Mass Production without first ramping up the production. Likewise, we cannot ramp up production without first having verified the design can be manufactured with reasonable yield. Decisions need to be made early with less data and more volatility. Decisions need to be made when the problem is very much under-specified and we have not seen all the "surprise" samples. It is in the earlier phases where DSML can be of most benefit allowing an engineer to search rapidly through the data and learn as much knowledge as possible. However, care must be taken to prevent these early decisions from causing harm in the future.

3.2 Outlier analysis

One of the simplest DSML problems to study is the outlier analysis. It is simple to understand while displaying the essential considerations for a DSML problem. An outlier analysis model can be a screening model used in both wafer probe and final test stages. Such a model is built on *parametric tests* where their results are measurement values (rather than pass or fail). A simplest form of an outlier model is based on setting a *test limit*. For example, a test applied to a chip returns a value v. The model is a simple rule, saying that if $v > t$ then the chip is bad. The t is called a test limit. Outlier models are commonly employed in production lines serving the automotive market [32][33][34][35][2], for example chips that will go into systems in a car. It should be noted that while quality is a critical concern, over-use of outlier screening can also cause a significant yield loss. Hence, in practice one has to strike a balance between quality and yield. It is also important to note that an outlier model is based on some test(s) applied in the test stage. Hence, the screening capability of an outlier model is limited by the effectiveness of the test(s) involved. In other words, one may create a better test to simplify the task of outlier analysis. In analog, this is like creating a better feature to simplify the task in traditional ML.

Figure 3.3 illustrates the setup of outlier analysis. First, a set of m samples are provided for the analysis. An *outlier method* is selected to calculate a score for every sample. Typical industrial practices include several choices of such methods [32]. Once the scores $s_1, s_2, \cdots, s_m$ are obtained, they can be used to rank the samples. Once ranked, a *threshold* is selected to decide which should be called outliers and which should be called inliers. This threshold decision can be data-driven, i.e. using known defective samples, or can be based on knowledge from design specification. In short, an outlier analysis includes three major components: the set of samples, the choice of outlier score

calculation method, and the threshold. These three components affect the outcome of outlier analysis.

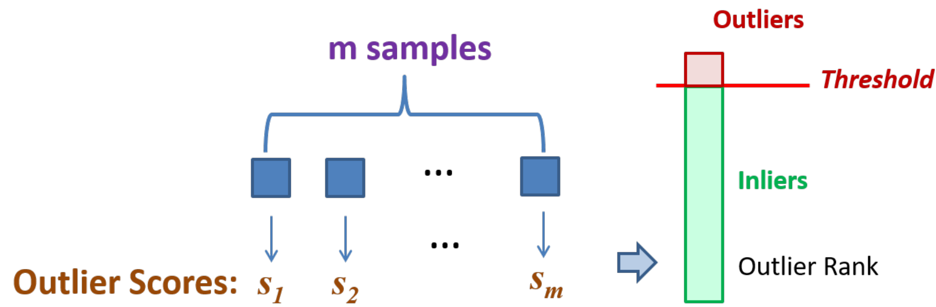

Figure 3.3: Outlier analysis setup.

For example, using a wafer of chips as the set of samples is different from using a wafer lot (25 wafers) of chips as the set of samples. Different score calculation methods can result in different rankings of samples. The threshold selection can be largely subjective.

An important point to make here is regarding the outlier score calculation method, which can be referred to as the outlier algorithm. Traditionally, "improving outlier analysis" largely focused on improving the algorithm [36], [37]. Subjectivity introduced in other parts of the outlier analysis setup is largely overlooked. Our work is the first to look at the problem by taking an entire view shown in Figure 3.3. In later chapters, we will argue that the underlying challenge of outlier analysis is less about the algorithm, but more about the selected set of samples and the selection of the threshold.

Before we discuss those points in detail, in the reminder of this chapter we will show two examples to illustrate the under-specification nature of the outlier analysis problem in test data analytics.

Figure 3.4 shows an example where a customer return (marked as a Customer Quality Incident of CQI) was given. Our task was to find an outlier model. Based on a particular test, the left plot shows that the CQI seems to be an outlier. The x-axis of the plot is indexed by the wafer number ranked according to their production time. The y-axis

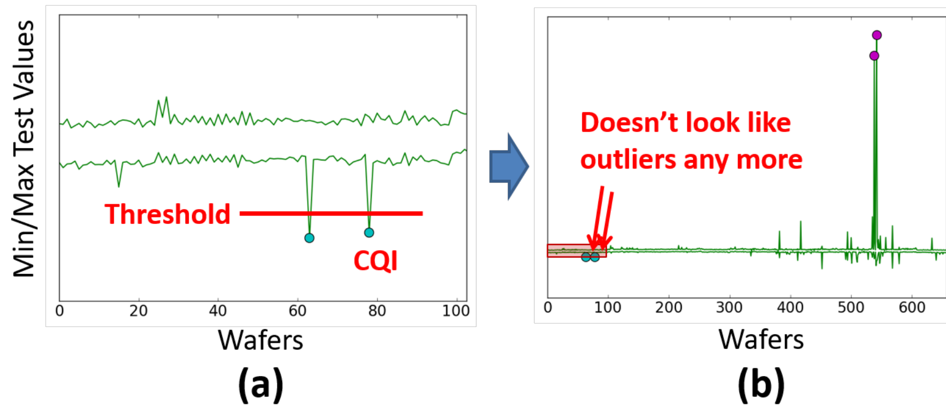

Figure 3.4: Perspective and Outliers: (a) The chips (one is a CQI) are outlying (b) The two parts in view of subsequent wafer data no longer seems as outlying.

shows the min and max values of the test across all dies on a given wafer. The test value of the CQI is the smallest on that respective wafer. There is another die with a similar behavior as the CQI. From plot (a), one may think we should pick a threshold to screen out these two dies, one of them confirmed to be a defective chip.

Plot (b) shows a similar plot but using many more wafers produced subsequently. In this plot, the two outlying parts in plot (a) no longer look like an outlier. If we were use a threshold determined from plot (a), we would have screened out many dies on future wafers. From plot (b), we could see that the screening (or in other words, the incurred yield loss by the threshold) might not be very justifiable.

It is important to note that the test limits, the upper limit and lower limit are both beyond the scales shown in the plots. In other words, all the values shown are still within the design specification. Also note that, to avoid unnecessary yield loss, it is possible that these (hard) limits are set to be not too tight to begin with.

The underlying question is: When we see the first 100 wafers in plot (a) and see that the CQI is like an outlier, how can we guard ourselves from a situation like that shown in plot (b)? Obviously, the data shown in plot (a) is under-specified, i.e. it has not shown us completely what is going to happen in future data.

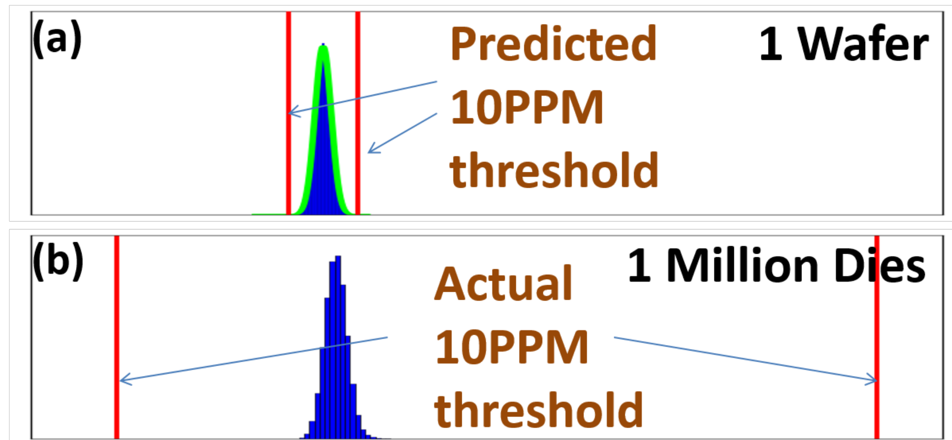

Figure 3.5: 10PPM yield-loss threshold with (a) Dies from a single wafer (b) 1 Million dies.

Figure 3.5 presents another example to show that the selected set of samples and the threshold are two inter-related aspects. Because yield loss is an important concern, one may think that we can start outlier analysis by setting a yield loss goal, such as screening out the top 10 outliers per million (10 PPM, Parts Per Million). Plot (a) shows an estimated threshold based on one wafer of parts. This is shown for a particular test. The test values are plotted as a distribution. One can model this distribution, for example using a Gaussian model. Then, based on the model one can calculate the thresholds (upper and lower) to approximate the 10 PPM result. Plot (a) shows where these thresholds locate.

Plot (b) then collects all parts after the wafer, to accumulate 1M parts and actually plot the thresholds that would screen out the top 10 parts. So plot (b) represents the golden answer to the two thresholds. As we can see, the true thresholds are very far from the estimated thresholds. And from plot (a), one cannot predict the true thresholds because there is no data points to tell us about them. This is clearly another example of under-specification. Again, this example shows that in an under-specification situation we cannot fully trust our best decision, and if we have to make a decision we also need to find a way to guard the decision against future surprise.

Chapter 4

Quest for Certainty

"Doubt is not a pleasant condition, but certainty is absurd."

— Voltaire

When one tries to apply machine learning in a domain-specific application context, it is natural to follow the process commonly seen in the traditional machine learning paradigm: collect the data, pick a ML algorithm, learn a model, and use the model for a task defined in the context. For example, in outlier analysis, the task is to screen out potentially defective parts. Hence, it is natural to ask for an outlier model that can identify parts as outliers which have a high chance to be defective. In building such a model, the last step is to decide on the threshold.

A threshold separates parts into two classes. An outlier model makes the decision how to separate parts into two classes. The two under-specification situations depicted in the previous chapters can be seen as making the wrong decision or selecting the wrong threshold, in retrospect.

By following the traditional ML paradigm, the under-specification problem seems essentially unsolvable, since future data can always surprise us. This is what we call

a local NFL situation in Chapter 2. However, if we look more closely, notice that the traditional ML paradigm is trying to solve a problem completely and convey certainty in the solution. In a sense, it is this quest for certainty that is source of the problem because without sufficient data, in view of traditional machine learning we simply cannot have attained the certainty we desire. And because this certainty is reflected in the threshold we use, when future data breaks our belief for the certainty, the problem becomes how to resolve the situation. If adjusting the threshold can remedy the problem, this might be easier. The harder part is to know if the outlier model should be thrown away, e.g. the test we use is not appropriate to begin with.

4.1 Taking a step back

In machine learning, it is common that we try to build a model M for deciding if a given sample should belong to a particular class. The model M can be quite complex, not as simple as an outlier model used in test data analytcis. However, regardless how complex the M is, ultimately to decide if a sample is within a class, the model calculates a score for the sample with respect to that class. Then, this score is used to reach the decision, meaning that a threshold must be used. Overall, because we want the model to make a decision for us, the quest for certainty is part of the model learning process.

Note that the above simple view is applicable to both common supervised learning such as classification, and unsupervised learning such as novelty detection (similar to outlier analysis) and clustering. These are common types of problems people try to solve in test data analytics.

However, due to under-specification-induced local NFL situation, we know that no algorithm can learn a model M whose decisions we can trust completely. The consequence is that if we still insist on our quest for certainty, then we can run into a situation where

the model fails us. And when a model failure is observed, we are left with the problem to find a fix. Finding this fix can be quite complicated. For example, we might not understand M to start with because it is complex and treated as a black box. This is a difficulty commonly encountered by ML practitioners. When a model fails, they might not even have a good idea to start the debugging process. In the outlier analysis, the process might be simpler. In general, due to the use of a complex M, it would not be an easy task to decide if the source of the problem is in the data, e.g. the new sample is quite different from those seen before, is in the algorithm, e.g. there is another algorithm more suitable for the context, or is in the class boundary, e.g. the threshold decision. Or perhaps we need a new feature to deal with the new sample.

If our quest for certainty is the source that can lead us to all those complicated situations, a logical solution to all the problems we have discussed so far can be to abandon the quest. In other words, we should consider *not* using a ML model to make a decision. Practically, our view for applying machine learning can change accordingly. For example, it can change from building a model to make a decision to utilizing machine learning techniques in extracting information to help users understand the data and make a decision themselves. In other words, the final decision is left for the user to make, and we apply machine learning techniques, if necessary, to help the user make as informative decision as possible.

In the context of outlier analysis, for example, this could lead to two thoughts: (1) when a user chooses to invoke an outlier analysis method, we can provide a way to inform the user how *applicable* the method is to the data; (2) when a user chooses a particular way to define the set of samples for outlier analysis, we can provide a way to evaluate the impact if this set is altered. In the next two chapters, we will describe two approaches based on these two thoughts, respectively.

In particular, in Chapter 5 we will focus on developing a method that can perform

an *applicability check* for a given outlier analysis method. In Chapter 6, we will discuss the method called *consistency check* to evaluate how an outlier threshold decision can be affected by the underlying sample set in use.

In Chapter 7, we will then turn our attention to the WMPR problem, followed by two more chapters discussing our experience from implementing and deploying a ML tool in practice. It is in those last two chapters where we will use our practical experience to illustrate further our main point made above, that ML should be used to help users understand the data rather than to make a decision for them. In other words, the quest for certainty should be left to the user, not the tool.

Chapter 5

Applicability Checking

"The first principle is that you must not fool yourself and you are the easiest person to fool."

— Richard Feynman

In the previous chapter, we use the phrase "quest for certainty" to illustrate how applying machine learning is commonly perceived by practitioners, and point out that it is that perception, in view of the NFL, can cause complication in DSML. When we desire a model that can make a decision, e.g. a classification decision, and we search for the best learning model for making that decision, that is when we might run into a situation where we are directly against the NFL. This can happen more likely in a DSML context, than a traditional ML context that is driven by big data.

In a DSML context, consequently, the focus is less about which algorithm is the best, but more about what assumption is taken by an algorithm. In this chapter, we will use outlier analysis as our target application example and illustrate a method for performing an *applicability check* whose purpose is to check if a given set of samples satisfies the assumption for using a particular outlier score calculation method. With such a check,

the underlying question is no longer which algorithm is the best. Instead, the question becomes, which algorithm is most suitable for calculating outlier scores on the given set of samples, or if a given algorithm should be applied on the given set of samples at all.

We have previously depicted the setup of outlier analysis with Figure 3.3. This setup requires some subjective choices to be made. The first is the choice of sample set, it can be a wafer, a single lot, or multiple lots. The second is the choice in method. Well-known methods include the various Part Average Testing (PAT) methods such as Static PAT (SPAT), Dynamic PAT (DPAT), Automotive Electronic Council PAT (AEC DPAT), and Robust DPAT (RDPAT), as well as Nearest Neighbor Residuals (NNR) [38][36] and Location Average (LA) [38][37]. The last of course is in the choice of threshold.

For outlier based screening, a question frequently asked by people in the industry is: Which method is the best for screening defective parts? Practically, this "best" can be understood as having the maximum chance to screen out a defective part while having a minimal yield loss. This questions can be asked in different contexts, for example, best for a family of product lines, best for a particular product line, or best for a specific test. Regardless in what context the question is asked, asking the question suggests that there exists a best method for the context.

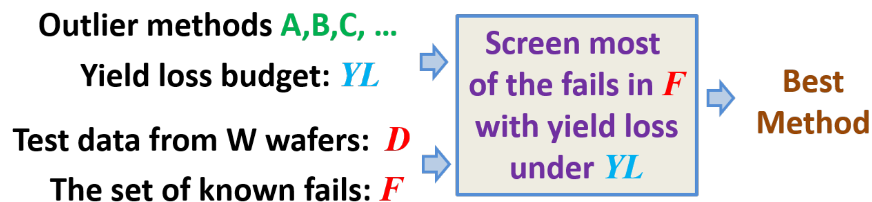

Figure 5.1: Conventional approach to evaluate outlier methods.

A common "benchmark-driven" approach to evaluate a method can be like that depicted in Figure 5.1 where each of the methods $A, B, C, \ldots$ are evaluated on an initial set of wafers D. In the context of testing, the evaluation is based on a method's ability

to screen out known defective dies F with a yield budget YL. Suppose method ϕ is determined to be the best based on some initial data D and the set of known defective parts F. Let D' represent data produced in the future and F' represent all screenable defective parts from D'. The fact that a method is the best based on the data (D, F) does not mean that the method is the best in view of the data (D', F').

5.1 A case study revealing a local NFL situation

In Chapter 2, we use the term *local NFL* to describe a learning setup where the available data does not allow us to predict how the future data might behave. A local NFL situation is specific to the data provided in a particular step in a DSML iterative process. It is different from the NFL which concerns the overall learning problem assuming that the number of samples can grow to infinity. In essence, local NFL can happen because of under-specification for the moment.

With that understanding, in this section we summarize a case study that reveals a local NFL situation. Using the notations above, essentially we evaluate five outlier methods based on some benchmark dataset (D, F) to draw some conclusion about the methods. Then, we will see how the conclusion can be entirely wrong when we move on future dataset (D', F').

The methods included in the study are popular outlier methods: SPAT, DPAT, AEC DPAT, NNR and LA. For 5 product lines shown Table 5.1 we gathered data and obtained a set of Customer Quality Incidents (CQI's) (i.e. customer returns) which we treated as our known defective dies.

In this experiment we try to determine the "best" outlier method and the corresponding model for each CQI. With each method we search for the test that results in the least yield loss based on D which is the set of 25 wafers that make up that CQI's lot (In this

Table 5.1: Data used for the study

Product code	# of wafers	# of tests	# of CQIs
VP	51100	249	32
KM	9675	350	23
A2M	400	45	11
MPC	4888	620	10
ALP	6996	123	77

sense, our sample set definition is the set of 25 wafers of dies). We consider a method and it's model better if it results in less yield loss. In this way we can compare all models and the one with the minimal yield loss YL_{min} will be the "best" (across all tests for a CQI). We take these "best" models (plural, because one model for each CQI) and apply them to the entire dataset D'. The resultant yield loss we denote as YL'_{min}.

First, we check if any of the methods produce a universally best model. We can also check if the performance of a model on (D, F) is predictive of that model on (D', F'). If neither of these results hold then they are evidences telling us that we might have a L-NFL problem. In essence, D is not representative for D' in view of the task we perform.

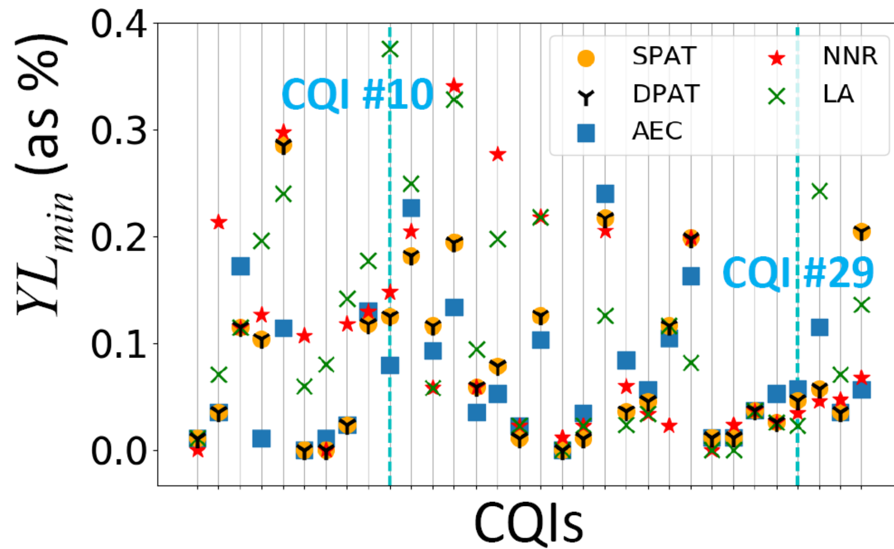

Figure 5.2: YL_{min} for each CQI - product VP.

5.1.1 No Universally Best Method Across CQIs

We plot the YL_{min} for each CQI in Figure 5.2 for all five methods. Each column in the plot represents a single CQI. The "best" model for each method is plotted in for each CQI, the height indicates that model's yield loss. The lower a model appears in the column the lower the yield loss and the better the model (and the method). From this plot we can see that there is no best method universally across all CQIs. Scanning across each CQI column the models with the lowest yield loss represents a diversity of methods. For example the best example for CQI#10 is AEC DPAT and the best method for CQI#29 is LA (Location Averaging).

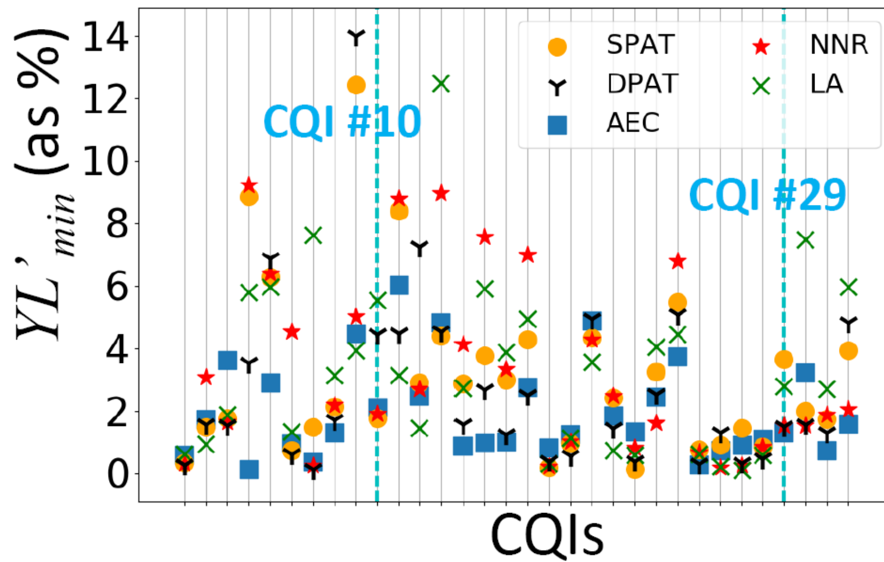

Figure 5.3: The corresponding YL'_{min} - product VP.

For comparison, the results of running each "best" model on the entire dataset D' are plotted in Figure 5.3. The first thing to notice is that these yield loss numbers are substantially larger than the YL_{min} values. Additionally we can see that the model with the lowest yield loss on D is not necessarily the model with the lowest yield loss on D'. For example, AEC DPAT had the model with the lowest yield loss for CQI#10 on the initial dataset, but with the full dataset the lowest yield loss model is SPAT. We can also

see this with CQI#29 where the lowest yield loss model went from LA to AEC DPAT.

5.1.2 Poor Generalization from YL_{min} to YL'_{min}

Figure 5.3 then shows the corresponding YL'_{min} number for each YL_{min}. The YL'_{min} results are based on all data, i.e. the 51100 wafers as shown in Table 5.1. Observe that for most CQIs, the YL'_{min} numbers are substantially larger than those YL_{min} numbers (the scales on y-axis are different), i.e. YL_{min} is a poor predictor for YL'_{min}. Furthermore, the best method determined using YL_{min} is not necessarily the best method using YL'_{min}.

Table 5.2: YL_{min}, YL'_{min}, Ranking for CQI #10 and CQI #29

CQI		
CQI #10	YL_{min}(%):	$0.08 < 0.125 < 0.125 < 0.148 < 0.385$
	Ranking	AEC < SPAT < DPAT < NNR < LA
	YL'_{min}(%):	$1.691 < 1.840 < 2.082 < 4.451 < 5.908$
	Ranking	SPAT < NNR < AEC < DPAT < LA
CQI #29	YL_{min}(%):	$0.023 < 0.035 < 0.046 < 0.046 < 0.058$
	Ranking	LA < NNR < SPAT < DPAT < AEC
	YL'_{min}(%):	$1.309 < 1.405 < 1.466 < 2.532 < 3.184$
	Ranking	AEC < NNR < DPAT < LA < SPAT

From Figure 5.2, we select two examples and show their details in Table 5.2. For example, for CQI #10, based on YL_{min}, the best method is AEC with $YL_{min} = 0.08\%$. The worst method is LA with $YL_{min} = 0.385\%$. However, when we move to YL'_{min}, the best method becomes SPAT with $YL'_{min} = 1.691\%$, and the worst is still LA.

For CQI #29, based on YL_{min}, the best method is LA and the worst is AEC. However, based on YL'_{min}, the best is AEC and the worst is SPAT.

Table 5.3 then summarizes the result of "best method" for all products. Here the best method is from the YL'_{min} point of view. Observe that for each product, each method is the best for a subset of CQIs.

From these results we see that given a method A the performance of A seen on D cannot generalize to the performance of A on D'. To overcome this barrier, our idea is

Table 5.3: # of CQIs a method is best for, based on YL'_{min}

Product	SPAT	DPAT	AEC	NNR	LA	total CQIs
VP	3	7	11	5	6	32
KM	4	2	9	1	7	23
A2M	0	3	4	1	3	11
MPC	0	4	4	2	0	10
ALP	11	20	23	12	11	77

to *constrain* the space for this generalization to take place.

Our goal is to determine a subset of wafers, S_A, on which A is *applicable*. It is important to note that this goal is *not* to determine the exact applicable subset. Instead, an applicable subset serves only as a constraint to restrict the application of A to those wafers in S_A. In other words, if a wafer is in S_A, we have a high confidence that this method is applicable. If it is outside, applicability is simply *undecided*.

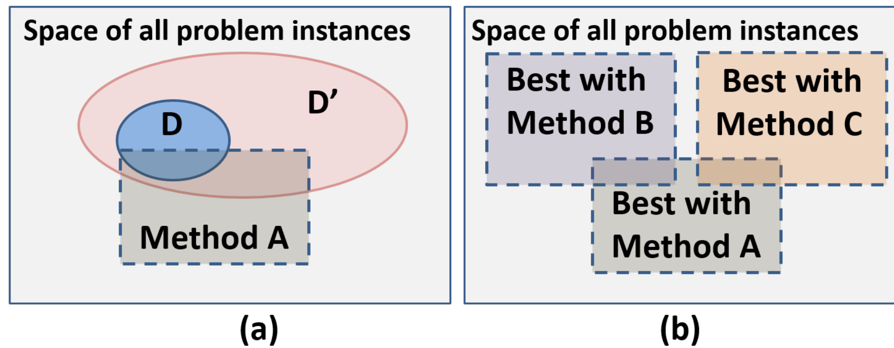

Figure 5.4: Core Ideas (a) constrain method A to where it is best (b) No method need be universally best

Our conjecture is that while generalization is poor from D to D', generalization can be improved if we constrain the scope to be only from $D \cap S_A$ to $D' \cap S_A$. Furthermore, with such an applicable scope concept, we no longer need to decide a universally best method. Each method can be the best on its respective applicable subset of wafers.

5.2 Applicability of an outlier method

Results shown above seem to reflect a local NFL situation: (1) The benchmark is misleading, and (2) In general no one method is better than another.

Given a set of wafers, even if one method is not generally better than another across all the wafers, it is still possible that if we restrict to a subset of wafers, one method is better than others. In other words, each method has its own subset of applicable wafers (e.g. see Figure 5.4 before).

Given a wafer G_i and a method ϕ with a test t_j, our goal is to develop a method $APP(G_i, t_j, \phi)$ that calculates an *applicability* measure for applying ϕ on the wafer data $\mathcal{D}_{ij}$. Let $\phi(\mathcal{D}_{ij})$ be the result of applying ϕ on $\mathcal{D}_{ij}$, our goal is to check some properties of the result $\phi(\mathcal{D}_{ij})$ in order to decide if ϕ is *applicable* or not.

5.2.1 Two properties to check for applicability

We first summarize the two properties in checking for applicability before we discuss why those two properties are at the core of the checking.

Conceptually, let $E(\phi, t_j)$ denote the "expected result" of applying ϕ to a wafer based on t_j. The first property to check is the difference between $\phi(\mathcal{D}_{ij})$ and $E(\phi, t_j)$, i.e. does the result meet the expectation? As it will be explained later, this property is to ensure that outlier decision made by method ϕ is *consistent* across wafers.

Then, the second property is to check on the assumption that each $\phi()$ should follow a Normal distribution. The property is to check on average how $\phi()$ deviates from the Normality assumption. This property is to ensure that outlier decision made by method ϕ is *justifiable*.

In other words, our notion of *applicability* is that the outlier decision made by applying a method on a given set of wafers is both *consistent* and *justifiable*.

To evaluate the first property, we will develop a *Variance* concept. To evaluate the second property, we will develop a *Bias* concept. Below we will explain their meanings with respect to a given set of wafers.

5.2.2 The basic assumption of an outlier

First, note that in outlier analysis there is no golden definition of what a "true" outlier should be. One of the most basic statements that can be said about an outlier is:

> Given a distribution $\mathcal{D}$, a sample is an outlier if its probability of occurrence is so small that it is unlikely the sample is drawn from $\mathcal{D}$.

This definition remains subjective to the probability threshold to define how small is small enough. However, this is the minimal about one must assume. The difficulty to apply this definition directly to find outliers is that one need to know the distribution $\mathcal{D}$.

For example, suppose n samples are drawn from a Normal distribution $\mathcal{N}(\mu, \sigma)$. For each sample value x_i, one can calculate the so-called z-score $z_i = \frac{x_i - \mu}{\sigma}$. A common method to identify outliers is then using the Grubb's test [39] which calculates the probability of the largest z-score given n (Notice that this probability depends on n). For example, for $n = 2000$ and probability threshold 10^{-6}, Grubb's test says if a sample has a z-score > 6.19, it is an outlier.

5.2.3 Density estimation vs. outlier transform

Based on the basic definition above, in practice there can be two approaches to develop an outlier method as illustrated in Figure 5.5.

The first is by applying *density estimation* [40] to estimate the distribution $\mathcal{D}$. Once the distribution is known, one applies a probability-based reasoning similar to the Grubb's test to find a threshold. Density estimation, however, is not very reliable for finding

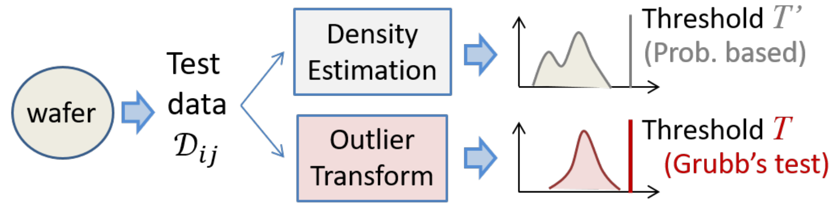

Figure 5.5: Two approaches to develop an outlier method.

outliers because it is very difficult to accurately estimate the probability density on the tails of a distribution [41].

The second approach is by applying an *outlier transform* to convert each test value into an *outlier score*. As a result, each test value distribution becomes an *outlier score distribution*. Then, a threshold is applied to each outlier score distribution to find outliers.

There are two concerns with the second approach. (1) If outlier score distributions on different wafers are different, then applying the same threshold on them have different meanings. In this case, the outlier decision is *inconsistent* across wafers. (2) To justify using the threshold on each outlier score distribution, we also desire each outlier score distribution to follow the assumed distribution where the probability-based reasoning is applied to derive the threshold. Otherwise, the threshold lacks a justification to be used with the outlier score distributions.

These two concerns reflect the two properties of applicability in section 5.2.1 above. In the following, we will explain how our *Variance* concept reflects the first concern and our *Bias* concept reflects the second concern.

5.2.4 Outlier methods in test are outlier transforms

An outlier method in test is basically a transform of a test value into an *outlier score*. For example, given a test value t for a die c, the five methods studied (SPAT, DPAT, AEC DPAT, NNR, LA) can all be thought of as following the abstract transform to obtain an outlier score O_c:

$$O_c = \frac{t - E(c)}{\varsigma} \tag{5.1}$$

where $E(c)$ is the *expected test value* of die c, and ς is the quantity to normalize the score. In other words, different methods differ in how they calculate the expected value and what normalization value should be used.

For example, SPAT uses the sample mean μ_s of test values across the entire wafer for $E(c)$ and uses $\varsigma = 1$, i.e. it does not normalize. DPAT also uses μ_s as $E(c)$ and in addition, uses the sample standard deviation σ_s as the normalization value, i.e. $\varsigma = \sigma_s$.

AEC uses the sample median value M_s across the wafer for $E(c)$. AEC uses "$0.43 \times (P_{99} - M_s)$" or "$0.43 \times (M_s - P_{01})$" as the normalization value, depending on which side of the test value t of c is located with respect to the median, where P_{99} and P_{01} are the 99% and 1% quantile values of the test value distribution, respectively.

NNR also uses the sample median value M_c, but the samples are not from the entire wafer. Rather, a $k \times k$ window (say 7×7) is decided centering on the die c. Only dies located in the window are used to calculated M_c (for 7×7, there are up to 49 dies). In this way, M_c can be different for different dies. Typically, NNR uses $\varsigma = 1$.

LA also uses a window to calculate M_c and typically uses $\varsigma = 1$. The difference is that M_c is not calculated based on all available dies in the window. Rather, it is based on only a percentage (say 50%) of the dies whose test values are the closest to the test value t of c.

Overall, every method makes an assumption of what the expected test value should be. Then, it makes an assumption of what a fair comparison should be by deciding how to normalize the deviation value "$t - E(c)$."

5.2.5 Concern of lack of consistency across wafers

Given a set of dies on a wafer G_i, conceptually their test values based on a given test can be thought of as forming a distribution $\mathcal{D}_i$. Note that the discussion from this point on assumes the test t_j is fixed. Hence, instead of using $\mathcal{D}_{ij}$ to denote the test data as before, we simply use $\mathcal{D}_i$.

An outlier method $\phi()$ transforms each test value into an outlier score. Effectively, the result is another distribution $\phi(\mathcal{D}_i)$. Hence, given W wafers, the result of outlier analysis by a method can be represented as a sequence of outlier score distributions $\phi(\mathcal{D}_1), \ldots, \phi(\mathcal{D}_W)$.

In outlier screening, one decides a threshold T to identify outliers. This T is repeatedly applied to each of the outlier score distributions $\phi(\mathcal{D}_1), \ldots, \phi(\mathcal{D}_W)$. If these score distributions are different (e.g. $\phi(\mathcal{D}_i) \neq \phi(\mathcal{D}_j)$ for $i \neq j$), it means that outlier decision made on different wafers are different, i.e. the decision is not *consistent* across wafers.

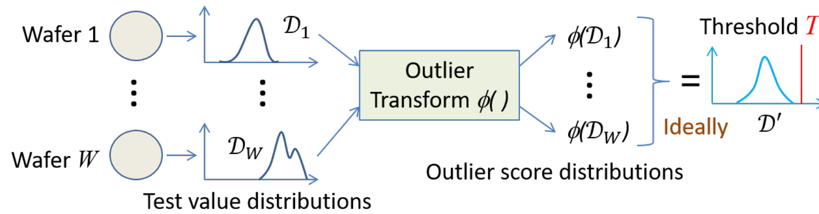

Figure 5.6: Consistency with an outlier method $\phi()$.

Ideally, to be consistent we would like to have $\phi(\mathcal{D}_1) = \cdots = \phi(\mathcal{D}_W) = \mathcal{D}'$ as illustrated in Figure 5.6 where $\mathcal{D}'$ can be the *expected* outlier score distribution $(E(\phi, t_j))$ as mentioned in section 5.2.1 before.

5.2.6 Assessing consistency with variance

When $\phi(\mathcal{D}_i) \neq \mathcal{D}'$ for some of the i, we would like to have a way to estimate the degree of discrepancy.

Suppose we have a distance function $DIST()$ that can measure a *distance* (difference) between two given distributions, i.e. let $d'_i = DIST(\phi(\mathcal{D}_i), \mathcal{D}')$. Then, we can define the *Variance* of an outlier transform across W wafers as:

$$\text{Variance of method } \phi() \text{ on } W \text{ wafers } = \frac{\sum_{i=1}^{W}(d'_i)^2}{W} \tag{5.2}$$

A larger Variance means the method is less consistent across the W wafers.

5.2.7 Assessing justifiability with bias

The expected distribution $\mathcal{D}'$ can be thought of as the *average* distribution across all $\phi(\mathcal{D}_1), \ldots, \phi(\mathcal{D}_W)$. To make the threshold T a probability-justifiable decision, we would like $\mathcal{D}'$ to be close to a known distribution. For example, we assume $\mathcal{D}'$ should be Normal to justify using Grubb's test. Let $\mu_{\mathcal{D}}$ and $\sigma_{\mathcal{D}}$ be the sample mean and sample standard deviation of $\mathcal{D}'$. Let $\mathcal{D}$ be the Normal distribution $\mathcal{N}(\mu_{\mathcal{D}}, \sigma_{\mathcal{D}})$. We therefore can model the degree of *justifiability* as a *Bias*:

$$\text{Bias of method } \phi() \text{ on } W \text{ wafers } = DIST(\mathcal{D}', \mathcal{D}) \tag{5.3}$$

The larger the Bias is, the less justifiable the method is. Figure 5.7 illustrates the Variance and Bias concepts.

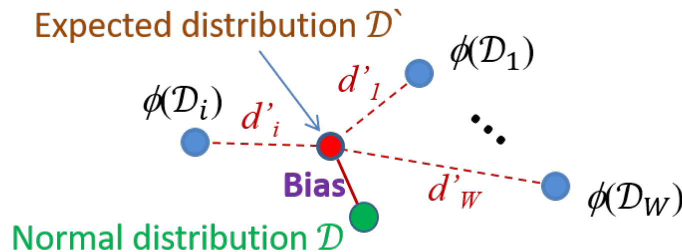

Figure 5.7: Variance and Bias in outlier transform.

It is important to note that a large Bias does not necessarily imply the method is not justifiable *in general*. It only means we cannot justify it based on our Normality

assumption of $\mathcal{D}$. Hence, applicability is constrained by the assumed $\mathcal{D}$. Consequently, lack of applicability should not be interpreted as "not applicable." A more meaningful interpretation is that we do not know if it is applicable or not. As mentioned in at the beginning with Figure 5.4 before, we treat applicability as a constraint to identify wafers where a method can be applied with a high confidence.

5.3 Calculating Variance and Bias

In order to calculate Variance and Bias, we require a way to implement the distance function $DIST()$. Our choice for $DIST()$ is the popular Kolmogorov-Smirnoff (KS) test. The KS test, denoted as $KS(\mathcal{D}_a, \mathcal{D}_b)$, measures a discrepancy between two given distributions $\mathcal{D}_a, \mathcal{D}_b$. The result is a value between 0 and 1, where 0 means the same and 1 means the most different. In practice, a KS tool [42] takes $\mathcal{D}_a$ and $\mathcal{D}_b$ as two vectors of sample values $\vec{v}_a, \vec{v}_b$. Note that the lengths of these two vectors can be different.

Hence, with W outlier score distributions $\phi(\mathcal{D}_1)$, . . ., $\phi(\mathcal{D}_W)$, each $\phi(\mathcal{D}_i)$ is represented as a vector $\vec{v}_i$ of outlier scores. The $KS()$ tool enables us to obtain pairwise measures $d_{ij} = DIST(\phi(\mathcal{D}_i), \phi(\mathcal{D}_j)) = KS(\vec{v}_i, \vec{v}_j)$, for $i \neq j$.

To calculate the Variance and Bias with equations (5.2) and (5.3), we need to calculate the *average* distribution $\mathcal{D}'$. With the KS tool, $\mathcal{D}'$ is simply the vector $\vec{v}'$ comprising the union of outlier scores from $\vec{v}_1, \ldots, \vec{v}_W$, i.e. $\vec{v}' = \vec{v}_1 \cup \cdots \cup \vec{v}_W$. Then, we can obtain each $d'_i = KS(\vec{v}', \vec{v}_i)$.

To calculate a Bias, we simply perform a random sampling based on the assumed Normal distribution $\mathcal{D}$ to obtain a vector of scores $\vec{v}$. Then Bias is calculated as $KS(\vec{v}', \vec{v})$.

5.3.1 Result visualization

In order to visualize a result like Figure 5.7, we also need to project each distribution $\phi(\mathcal{D}_i)$ (i.e. the $\vec{v}_i$) as a point in Euclidean space. The trick is that in this space, the

points should be positioned relative to their pairwise distances $KS(\vec{v}_i, \vec{v}_j)$. This is a typical *multidimensional scaling* (MDS) problem. We utilize an MDS tool from [43] to project each distribution into a 3-dimensional space.

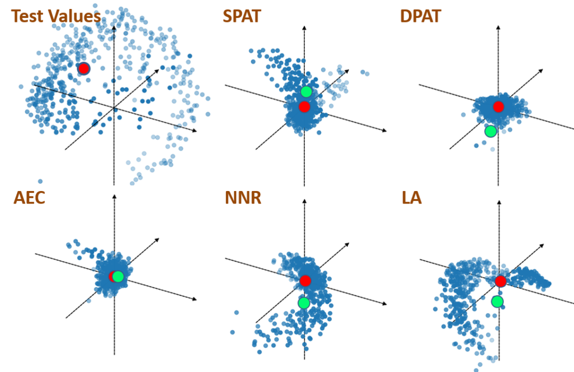

Figure 5.8: Visualizing Variance and Bias for one test.

Figure 5.8 shows plots similar to Figure 5.7 for one particular test from product VP. The results here are based on 500 wafers. The computation is limited to $W = 500$ wafers because to produce each plot, we need to compute W-choose-2 pairwise distances $KS(\vec{v}_i, \vec{v}_j)$.

In Figure 5.8, each blue-gray dot represents a distribution (from a wafer). The first plot "Test Values" shows the original test value distributions $\mathcal{D}_1, \ldots, \mathcal{D}_{500}$. Then, in each method plot, the score distributions $\phi(\mathcal{D}_1), \ldots, \phi(\mathcal{D}_{500})$ are shown. The red dot marks the *expected* distribution $\mathcal{D}'$ which is also used to center each plot. The green dot marks the assumed Normal distribution $\mathcal{D}$ based on $\mathcal{D}'$.

In Figure 5.8, a larger spread means a larger Variance and hence the result is less consistent. A larger distance between the red dot and the green dot means a larger Bias and hence, the result is less justifiable.

Figure 5.8 shows that DPAT and AEC have smaller Variance than others. Table 5.4 shows their Variance and Bias values. First, observe that all methods achieve a *variance reduction*. For example, DPAT brings the variance from its original value 0.223 down

to 0.0094. This means that the original test value distributions across wafers are much more diverse. After an outlier transform, the resulting distributions become more similar, enabling one to make a more consistent outlier decision across wafers than using the original test values. This variance reduction can be thought of as a key objective of an outlier transform.

Table 5.4: # Variance and Bias from different methods

	Original	SPAT	DPAT	AEC	NNR	LA
Variance	0.223	0.046	0.0094	0.0104	0.0543	0.0718
Bias	—	0.163	0.158	0.0633	0.174	0.174

While DPAT has the smallest variance, its Bias is not the smallest. Overall, we see that AEC is better because its Variance value and Bias value are both small.

5.3.2 Best method for a particular wafer

For a given wafer G_i, we can further compare methods based on the distance directly to the Normality assumption $\mathcal{D}$, i.e. $d_i = DIST(\phi(\mathcal{D}_i), \mathcal{D})$. We say that the smaller the d_i is, the more *applicable* the $\phi()$ is for the wafer because d_i can be thought of as measuring the combined effect of both consistency and justifiability together.

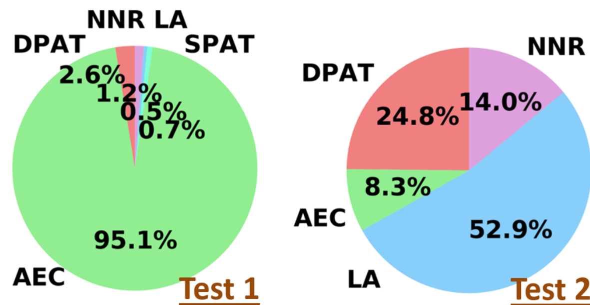

Figure 5.9: % of wafers a method is best on.

Based on d_i, Figure 5.9 shows the percentage of wafers a method is the best on. "Test 1" is the test used to produce Figure 5.8. We see that AEC is the best on 95.1% of the wafers, an expected result based on what we observe in Figure 5.8. We further select

a "Test 2" to show a contrasting result, where LA is the best on 52.9% of the wafers. These examples illustrate that no single method is the best on every wafer and moreover, different tests result in different evaluation results.

5.3.3 Results across tests

For a given test, we can say that a method is the best if its Variance is the smallest or if its Bias is the smallest. These two measures provide different perspectives to examine how overall a method is applicable with the test. Based on the two perspectives, Figure 5.10 shows the percentage of tests a method is best with, across all 249 tests from product VP. DPAT has the largest percentages in both cases.

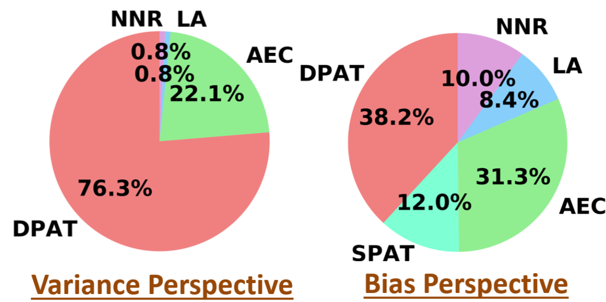

Figure 5.10: % of tests a method is best with - product VP.

Further, if we consider all (1387) tests across the five products, Figure 5.11 shows similar results.

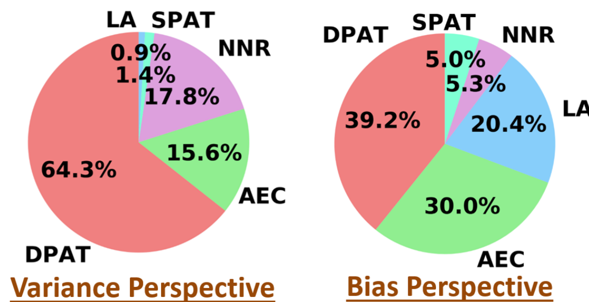

Figure 5.11: % of tests a method is best with - all products.

Overall, even though DPAT has the largest percentages, every method shows its unique values. Thus in a test application, one should not discard any of the methods in

advance.

5.4 Examples to illustrate Variance and Bias

Figure 5.12 shows the test value distributions of two wafers selected from Figure 5.8 presented before. Observe that these two distributions are quite different. In fact, their KS distance is 0.905 (recall that $0 \leq KS() \leq 1$).

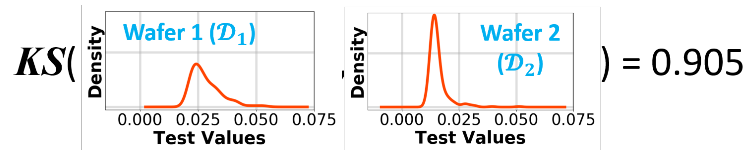

Figure 5.12: Original test value distributions $\mathcal{D}_1, \mathcal{D}_2$.

Figure 5.12 then shows the respective outlier score distributions resulting from DPAT transform. Their KS distance is 0.151, substantially smaller than 0.905. This illustrates the variance reduction effect with DPAT transform.

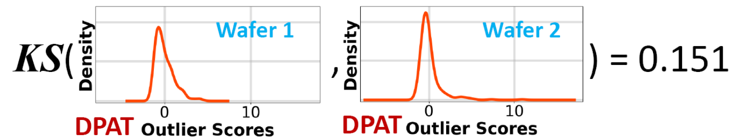

Figure 5.13: Effect of DPAT, $KS(DPAT(\mathcal{D}_1), DPAT(\mathcal{D}_2))$.

In comparison, Figure 5.14 then shows the respective outlier score distributions resulting from LA transform. Their KS distance is larger than the DPAT distance. From this perspective, we can say that LA is not as effective as DPAT.

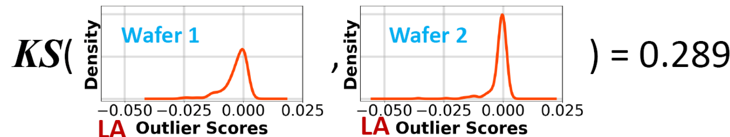

Figure 5.14: Effect of LA, $KS(LA(\mathcal{D}_1), LA(\mathcal{D}_2))$.

The example illustrates how in Figure 5.8 LA has a larger spread (a larger Variance) than DPAT. In fact, the two wafers are selected based on Figure 5.8 where their distances are among the farthest in the LA plot.

5.4.1 Simulated examples to illustrate Bias

In general, there is an implicit assumption with every outlier transform on what properties a test value distribution should have. While this assumption might not have been explicitly or formally characterized with a method, such an assumption should always exist. Consequently, when this assumption is violated, a large Bias can occur.

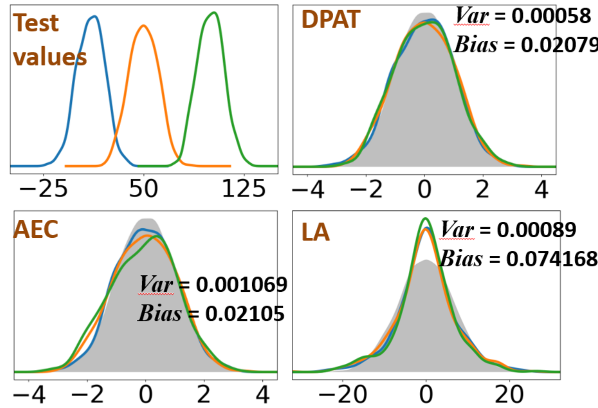

Figure 5.15: 3-wafer examples, all Normal.

For example, Figure 5.15 depicts an example with 3 wafers. Each test value distribution is sampled from a Normal distribution. 400 dies are sampled from $\mathcal{N}(10, 10)$, $\mathcal{N}(50, 10)$, and $\mathcal{N}(100, 10)$, respectively.

Because the test value sample distributions are all Normal, it is expected that DPAT would have a small Bias. As seen in the DPAT plot, the 3 resulting outlier score distributions (colored similarly to the "Test values" plot with blue, orange, and green) are close to the assumed Normal distribution $\mathcal{D}$ (colored as the shaded gray area).

AEC is intended to improve DPAT by taking asymmetric distribution into account. We see that AEC also has a small Bias. LA has a slightly larger Bias because each outlier score is calculated based on a smaller sample size with dies in a given window. A smaller sample size can cause more noise. As seen, the 3 distributions from LA deviate more from the assumed Normal distribution (shaded gray area).

Figure 5.16 then depicts a different 3-wafer example. The test value distributions

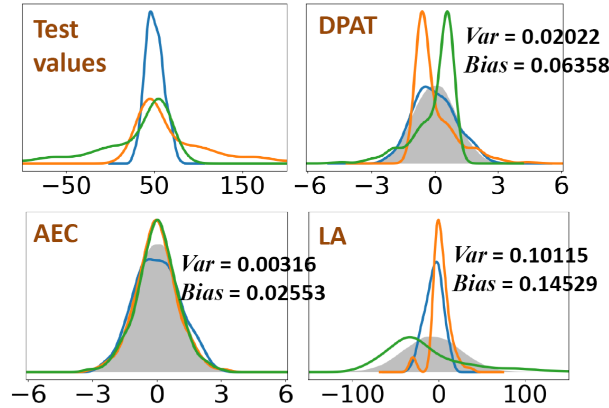

Figure 5.16: 3-wafer example that favors AEC.

are skewed (asymmetric) from a Normal distribution. This asymmetry violates DPAT's assumption but meets the assumption of AEC. As a result, DPAT has a larger Bias. AEC's Bias remains comparable to that seen in Figure 5.15. Since LA does not consider asymmetric distribution of test values, its Bias increases from that in Figure 5.15 as well.

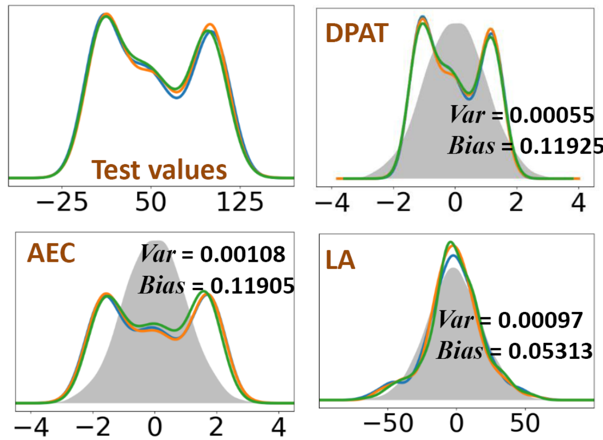

Figure 5.17: 3-wafer example that favors LA.

The assumption of LA is that there is some location-based wafer pattern across the test values. In Figure 5.17, on each wafer, we create a horizontal line pattern where each line is repeatedly sampled from one of the three Normal distributions, $\mathcal{N}(10, 10)$, $\mathcal{N}(50, 10)$, and $\mathcal{N}(100, 10)$. Each wafer is sampled in the same way. We expect LA to have a smaller Bias on such an example.

As seen in Figure 5.17, LA has a small Bias while DPAT and AEC, which do not

consider any wafer pattern, both have a large Bias.

The three examples illustrate that a large Bias can be an indication that the test value distribution is out of the consideration by a method (i.e. violating its assumption of how the distribution should be). When a large Bias occurs, using the resulting outlier scores becomes not justifiable.

5.5 Applicability results

Let $\mathcal{D}_{ij}$ denote the test value data based on test t_j on wafer G_i. Let $\mathcal{D}$ be our assumed Normal distribution. Given a method ϕ, in section 5.3.2 before, we use the distance $d_{ij} = DIST(\phi(\mathcal{D}_{ij}), \mathcal{D})$ to measure the combined effect of consistency and justifiability. Here, we can simply define *applicability* of an outlier method ϕ with t_j on G_i as

$$\text{Applicability: } APP(G_i, t_j, \phi) = 1 - d_{ij} \tag{5.4}$$

Let H_{app} denote an applicability threshold. For example, for $H_{app} = 0.90$ we are interested in seeing how many wafer-test combinations have applicability greater than this threshold. Figure 5.18 summarizes such results.

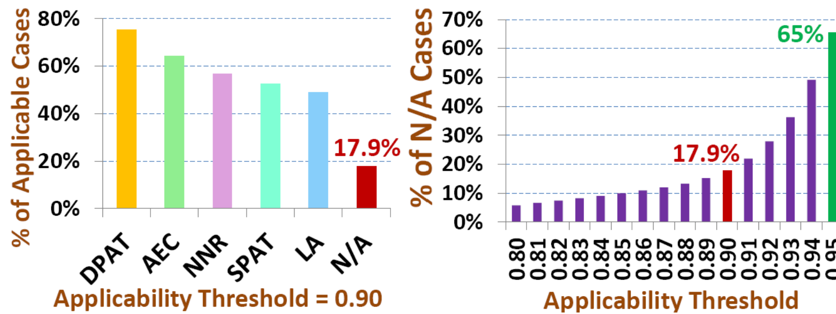

Figure 5.18: Summary results with applicability thresholds.

The left chart shows, for all wafer-test combinations across all products, the percentage of wafer-test combinations for which a method is 0.9-applicable (or 90%-applicable). This percentage is not exclusive, i.e. multiple methods can be applicable for the same

combination. In addition, we include an "N/A" **bar** to indicate the % of combinations where no method has applicability ≥ 0.90.

As one changes the threshold H_{app}, the percentage of N/A combinations would also change. The right plot in Figure 5.18 therefore shows how the "N/A" percentage changes with respect to the change of the threshold H_{app}.

As seen in the right chart, if we set $H_{app} = 0.95$, then about 65% of the wafer-test combinations become N/A. One can think of $H_{app} = 0.95$ as 95% confidence to apply the outlier method to a wafer-test combination. Hence, the plot shows that for 65% of the wafer-test combinations, we do not have 95% confidence to apply any one of the five outlier methods. However, if we lower our confidence to 90%, the percentage of N/A is reduced to 17.9%.

5.5.1 Usages of the applicability measure

Given a collection of outlier methods, Figure 5.18 shows that applicability can be used to identify the subset of wafer-test combinations for which each method is applicable, and the subset of combinations for which no method is applicable. Identifying the applicable subset with each method enables one to choose the best method for each combination with more confidence. Identifying the N/A subset enables one to assess if the collection of methods is sufficient. In the next section, we will re-visit the results presented in the beginning of this chapter and show how those total yield loss YL'_{min} numbers can be reduced.

5.6 Re-visiting results in Section 5.1

In Section 5.1, the "best" outlier model is established for each CQI. For each CQI, we can use applicability to evaluate if the model (based on the particular method with the particular test) is indeed applicable on the CQI wafer. With an applicability threshold

$H_{app} = 0.95$, Table 5.5 shows the # of CQI cases that are not applicable ("N/A") and the # of cases that are applicable. In total, about 38% ($= \frac{59}{153}$) of the CQI cases where their models are applicable.

Table 5.5: Applicability result across all CQIs

Product	VP	KM	A2M	MPC	ALP	Total
# CQIs	32	23	11	10	77	153
# "N/A"	17	22	5	7	43	94
# Applicable	15	1	6	3	34	59

5.6.1 Improvement on total yield loss YL'_{min}

For the 59 applicable CQI cases, Figure 5.19 shows how much the total yield loss YL'_{min} is reduced if the CQI outlier model is applied to only wafers with applicability ≥ 0.95. For each CQI, the "Without applicability" bar denotes the YL'_{min} from the original experiment described in Section 5.1. The "With applicability" marker denotes the new YL'_{min}. For a fair comparison, note that this new yield loss % is based on the total number of parts from only those applicable wafers, not the entire set of wafers as before. In other words, both YL'_{min} numbers are the percentage from the same formula: $\frac{\text{total \# of parts screened out}}{\text{total \# of parts applied on}}$.

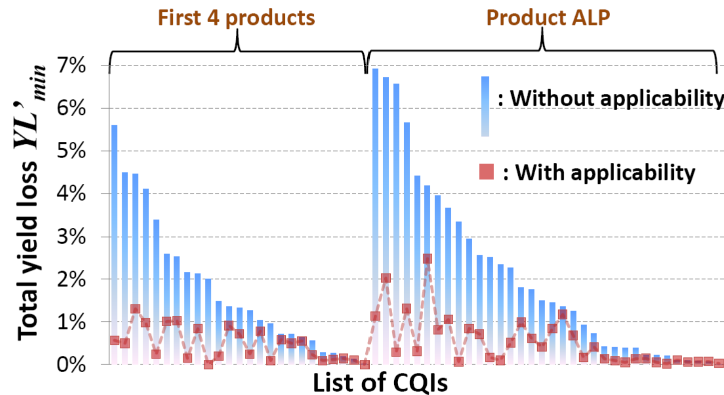

Figure 5.19: Improvement on total yield loss YL'_{min}.

The improvement on the total yield loss can be clearly observed. This is interesting because applicability check reduces the total number of parts applied on. By reducing

the denominator in the YL'_{min} formula above (can be a significant reduction as indicated from Figure 5.18 that many cases (65%) are not applicable with $H_{app} = 0.95$), we observe that the resulting YL'_{min} is also reduced. This implies that the % of yield loss on those not-applicable wafers is greater than the % of yield loss on those applicable wafers. This suggests that the wafers removed by the applicability check indeed contain the excessive yield loss.

5.6.2 Improvement on yield loss difference

In Section 5.1, poor generalization is shown by the significant difference between YL_{min} and YL'_{min}. Based on this difference, "$YL'_{min} - YL_{min}$," Figure 5.20 shows how applicability check improves this generalization.

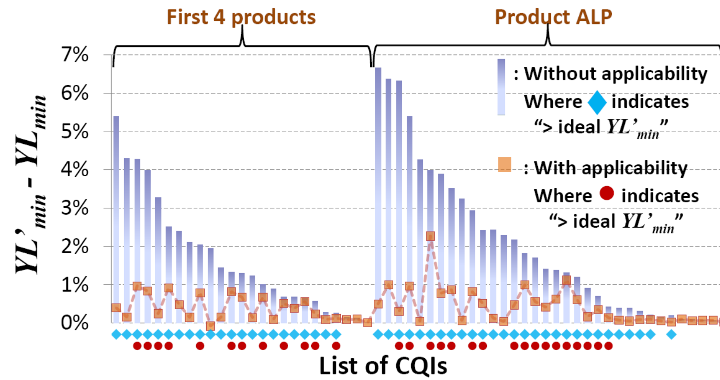

Figure 5.20: Improvement on yield loss difference $YL'_{min} - YL_{min}$.

5.6.3 Ideal YL'_{min} for each given YL_{min}

It is important to note that because YL'_{min} is based on a (much) larger number of parts, it is expected that $YL'_{min} > YL_{min}$. Suppose all wafer outlier score distributions follow our assumption of Normal distribution $\mathcal{D}$. Even with this ideal situation, we still expect to see $YL'_{min} > YL_{min}$.

To see this, for a given YL_{min}, we perform Monte Carlo simulation of the ideal situation based on the assumption $\mathcal{D}$ and using the same number of parts in each case

(e.g. on CQI lot, on all lots, without applicability, with applicability). Through this simulation, we calculate an *ideal* YL'_{min} for each case. Table 5.6 shows some example results based on the first five CQIs shown in the figures above.

Table 5.6: Examples of ideal YL'_{min} based on YL_{min}

CQI	1	2	3	4	5
Results without applicability check					
YL_{min} =	0.197%	0.209%	0.191%	0.124%	0.12%
Ideal YL'_{min} =	0.768%	0.754%	0.649%	0.611%	0.523%
Actual YL'_{min} =	5.597%	4.496%	4.461%	4.107%	3.39%
Results with applicability check					
YL_{min} =	0.172%	0.343%	0.341%	0.154%	0.00%
Ideal YL'_{min} =	0.683%	1.06%	1.124%	0.653%	0.184%
Actual YL'_{min} =	0.567%	0.497%	1.31%	0.984%	0.241%

Note that with the applicability check, the YL_{min} on the CQI lot can change as well, even though we do not change the outlier model. This is because some wafers from the CQI lot can be not-applicable.

As seen in the table, each ideal YL'_{min} is greater than the YL_{min}, but not too much greater. For each case, if the actual YL'_{min} is greater than the ideal YL'_{min}, then this means the total yield loss is not ideal.

Without applicability, we see that all cases have an actual YL'_{min} above the ideal yield loss. And the difference between the ideal and the actual is quite large.

With applicability check, for CQI 1 and CQI 2, the actual YL'_{min} is smaller than the ideal. Even for the other three cases, the actual YL'_{min} is much closer to the ideal than those numbers without the applicability check.

Refer back to Figure 5.20. In the figure we use a diamond marker to note the CQI cases where the actual YL'_{min} is greater than the ideal yield loss in the "Without applicability" experiment. We use a circle marker to note those similar CQI cases in the "With applicability" experiment.

As seen from those makers in Figure 5.20, without applicability many CQI cases have

a total yield loss greater than the ideal. With applicability, many of these cases (without a marker) become ideal or smaller than the ideal yield loss. Note that for most of those CQIs with the circle marker, the differences between the actual YL'_{min} and the ideal yield loss are quite small, similar to those shown in Table 5.6.

5.6.4 Consistency without justifiability

In the above experiment, we use the applicability equation (5.4). As mentioned above, this applicability reflects the combined effect of both consistency and justifiability. Suppose we change this to measure only consistency and discard justifiability. In other words, we use $CON(G_i, t_j, \phi) = 1 - d'_{ij}$ where $d'_{ij} = DIST(\phi(\mathcal{D}_{ij}), \mathcal{D}')$, i.e. distance to the average distribution $\mathcal{D}'$, rather than to the Normal distribution D. By replacing $APP()$ with $CON()$, we re-perform the experiment for the 59 CQI cases by keeping all the other aspects in the experiment the same.

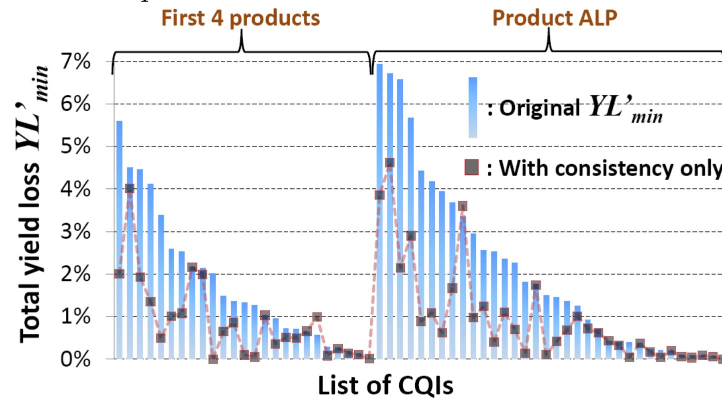

Figure 5.21: Less improvement on total yield loss YL'_{min}.

Figure 5.21 shows a similar chart as that shown in Figure 5.19 above. Notice that, although the total yield loss YL'_{min} is still reduced for many CQIs, the reductions are smaller than those shown in Figure 5.19. For a few CQIs, the new YL'_{min} is the same as before or even worse. Figure 5.21 shows the importance to include justifiability (i.e. Bias) in our applicability measure. Without checking the justifiability, we would not be able to obtain the substantial yield loss improvements.

5.7 Summary

In the semiconductor test industry, practitioners had been searching for the best outlier method for years, due to the importance of outlier screening in quality assurance for products sold to the automotive market or alike. However, in evaluating different methods to choose the best method, the evaluation relies on some initial set of samples. If this set is not representative enough, the best method deemed by the evaluation has no meaning in view of the future samples. This situation is captured in our notion of local NFL in view of an overall DSML setup.

The important first point from our study is that, we need to give up on searching for the best method and accept that it does not exist in general. Then, we should realize that every method has an assumption to begin with. In application of a method, this assumption needs to be evaluated for the application to be justified. Otherwise, comparing methods in terms of their outcome with respective to some performance metric, e.g. yield loss, can become misleading or even meaningless. Focusing on evaluating the assumption helps achieve a deeper understanding about the data and the methods at hand, enabling a practitioner to make a more informative decision.

That is the philosophy behind the work presented in this chapter, that one should not look at a method as a better method or a worse method. One should look at the method and the data *together* and ask, if the method is appropriate for the data. When we are taking that perspective, an evaluation scheme of the assumption with a method is no longer just about the method. It is also a way to assess the data where the result reveals some characteristics about the data. From this angle, those methods can be seen as the vehicles for a user to achieve a better understanding more about the data. This point will be emphasized more in the later chapters when we discuss deployment of a machine learning application software in practice.

Chapter 6

A “What-If” Analysis

A man may imagine things that are false, but he can only understand things that are true, for if the things be false, the apprehension of them is not understanding.

— Sir Isaac Newton

In Chapter 2, we pointed out that it could be futile trying to achieve a provable machine learning outcome because that would be directly against the NFL theorem (or Local NFL in DSML). The Bayesian view of machine learning provides a more feasible alternative. Implementation-wise, the view suggests that it is more practical to implement a tool whose goal is only to evaluate the outcome from an assumption given by the user. In Bayesian view, this is the well-known prior-to-posterior framework. In practice, we can also look at it as implementing a “what-if” analysis approach.

In Chapter 3, we described three components in an outlier analysis setup, which can be affected by user’s subjectivity: the sample set selection, the algorithm selection, and the threshold selection. While the applicability check in the previous chapter tries to mitigate the effect of subjectivity in the choice of algorithm, in this chapter we discuss a way to

mitigate the effect related to the other two components. The method was proposed in [2], which aims to calculate a so-called *consistent threshold* for an outlier screening model. In this chapter, we will interpret the work from the perspective of what-if analysis. We use this example to illustrate how the theoretical view from Chapter 2 can actually echo our practical view in approaching the problem.

It should be noted that the work in [2] was originally developed for a different purpose than trying to perform what-if analysis: The goal was to have an outlier analysis approach that can say "There is no outlier". This need was motivated in the context of customer return analysis (see Chapter 3 for more discussion). In the analysis, a customer return is given and we are asked to search for an outlier model that projects the customer return as an outlier. This problem is not completely specified, because being an "outlier" is subjected to our choices made in the three components: the sample set, the algorithm, and the threshold. The work in [2] tries to find a way to eliminate some of the choices and in turn, reduce the search space for an outlier model. In other words, the original motivation was from pruning-the-search-space point of view.

The work in [2] started with evaluating three common outlier methods (the "algorithm"): SPAT (Static Part-Average-Testing), DPAT (Dynamic PAT) and LA (Location Averaging) and showed an inconsistency in the outlier sets deemed by the three methods. The dataset was from an automotive product line, containing 5000 wafers and totals 3 million dies. As an example, one can select a threshold aiming to screen out X PPM (parts per million) such as 100 PPM, 10 PPM, or 1 PPM. Suppose all the 3M dies are used. Then, at 100 PPM, the threshold would be set to screen out the 300 most outlying dies. Recall that the ranking is decided by the outlier scores calculated by the particular method in use.

The results of two tests were summarized in [2] as Venn diagrams. We re-display the results from Test 1 in Figure 6.1 and Test 2 is Figure 6.2. One thing to observe is that

for Test 1 as the threshold is changed from 100PPM to 1PPM the disagreement between the methods decreases, and all methods consistently identify the same 3 most outlying dies in the 1 PPM case. However for Test 2 we see that this is not the case with SPAT completely disagreeing with LA and DPAT about which 3 dies are the most outlying dies. From this we can understand that depending on our PPM goal, for some tests there is consistency between the models and for other tests, there is not.

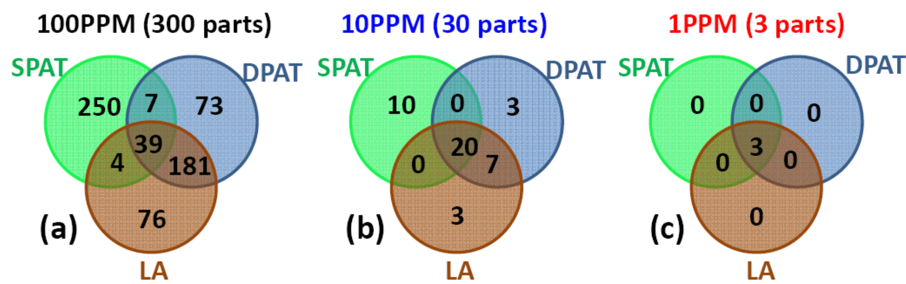

Figure 6.1: Results for Test 1. Image taken from [2]

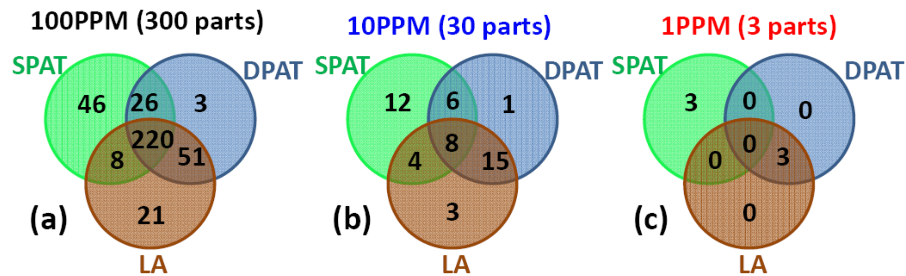

Figure 6.2: Results for Test 2. Image taken from [2]

Looking at the minimum/maximum test values led to a rational for why these two tests cause the models to behave differently. In Test 1, the more consistent test, the min/max test values are shown in Figure 6.3 where the extreme nature of the outliers requires two scales to be presented. There we have highlighted the top 7 outliers in red, the scale of the top plot in Figure 6.3 is so large that the 7 outliers appear as 3 groups. We have provided numbers next to each group to indicate how many outliers are in that group. In contrast to Test 1 and it's two scale presentation, Test 2 can be presented on

a single scale in Figure 6.4. Since 20 of the values in Test 1 are "obvious" outliers it is easy for the 3 different methods to agree that they are outliers. However for Test 2, none of the values are "obvious" outliers; because any outliers can only be *marginal* it makes the outlier models inconsistent with each other. Hence, the idea is to find a way to differentiate between these two situations, i.e. having "obvious" outliers and having no "obvious" outliers. And the method proposed in [2] is called *consistency check*.

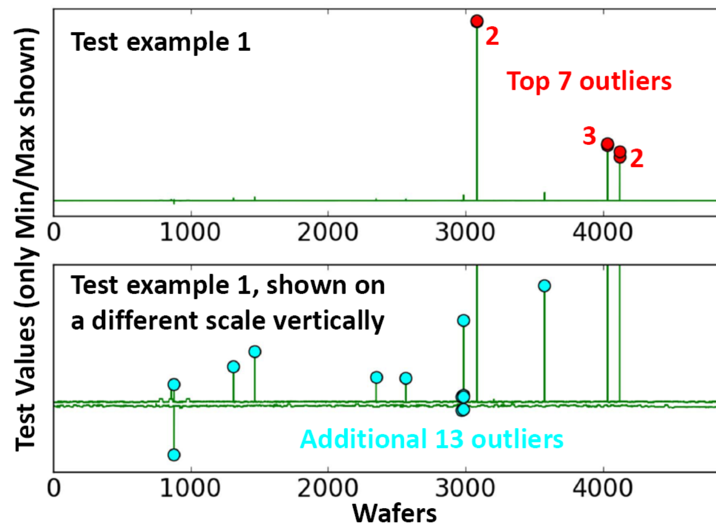

Figure 6.3: Min/Max value plot for Test 1. Image taken from [2]

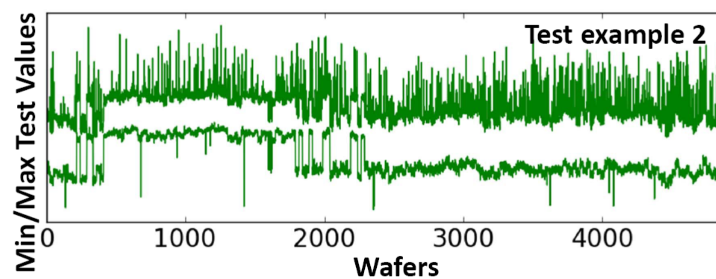

Figure 6.4: Min/Max value plot for Test 2. Image taken from [2]

6.1 Consistent threshold selection

Given a die with test value m_i and outlier score s_i, suppose this score is calculated based on all dies from the same wafer. If the die is an outlying die, it only means that the die is an outlying die with respect to other dies on the same wafer.

If this die is an "obvious" outlier like those in Test 1 (Figure 6.3) then it should also be an outlier if it were put on another wafer. To check this we can take the original test value and compute it's outlier score as if it were on another wafer. If it is an "obvious" outlier then its ranking from this new score would be comparable to the original ranking. On the other hand if the die is a "marginal" outlier like those in Test 2 (Figure 6.4) then moving it to other wafers will likely change it's ranking substantially.

In essence, the idea just described is to perform a what-if analysis of a given outlier. The idea is evaluating what if we see the outlier from other wafers' point of view. This is a simple idea, but an effective one. As shown in [2], it enabled showing no outlier for a large number of tests and hence, could reduce the search space for a customer return significantly.

In the what-if analysis, the more wafers that the outlier can "survive" and still remain an outlier gives more evidence that this die is one of these "obvious" outliers. If moving a die tends to change it's outlier/inlier status, it is called an inconsistent outlier. If a die's status does not tend to change when moved to another wafer it is a consistent outlier. An convenient measure of a die's consistency is as a percent of inlier / outlier labels. So if a die is "moved" to 100 wafers and in 95 wafers it is an outlier we can say it is a 95% outlier.

This procedure of "moving" dies can also provide a estimation of outlier score confidence. The collection of outlier scores gathered by moving a die from wafer to wafer will have some distribution of values. If these values are tightly grouped together then

the score itself is consistent and we can be more certain of the categorization of that die whether inlier or outlier. If the values are widely distributed then our confidence that a die is an inlier/outlier would go down. The distribution of the outlier scores is not due to the die itself since the test value does not change. Rather the distribution is due to wafer-to-wafer variations so in a real sense this confidence is measuring noise in data with respect to a particular die and model.

Following from these ideas when we select a threshold for an outlier model there are three things that it can do: 1) It can identify consistent outliers, 2) minimize inconsistent outliers and 3) It can provide a margin of safety. As we have argued here consistent outliers have evidence that they are "obvious" outliers while the inconsistent outliers are not. For this reason we can justify screening out the consistent outliers but not the inconsistent ones.

$$\begin{bmatrix} \overrightarrow{v_1} \\ \overrightarrow{v_2} \\ \vdots \\ \overrightarrow{v_n} \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1W} \\ s_{21} & s_{22} & \cdots & s_{2W} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nW} \end{bmatrix}$$

Figure 6.5: Consistent threshold framework.

Putting all this together we can create a framework for selecting a threshold. Suppose there are W wafers $w_1, \ldots, w_W$ in our dataset. For a given die with test value m_i we can create a row vector of outlier scores $\vec{v_i} = (s_{i1}, \ldots, s_{iW})$. For a component in this vector s_{ih} it is the outlier score when that test value is moved to wafer W_h. Combining these $\vec{v_i}$ row vectors into the matrix shown in Figure 6.5 allows the quality of any threshold to be measured.

6.2 A minimum consistent threshold

In [2] they chose to only consider 100% consistent outliers, ones that were never identified as an inlier (based on all the wafers available for evaluation). They also only consider thresholds that identify these 100% consistent outliers and no others. This approach greatly simplifies the analysis and gives the highest quality threshold that can be found. Additionally they utilized a *noise band* based on the distribution of outlier scores for a single die. If a consistent threshold was found to be within the noise band of a die it was rejected.

The procedure given by [2] for selecting a threshold was to start with a small threshold value, one that was guaranteed to be inconsistent. This initial threshold value would then be increase it until a consistent threshold could be identified that was not in a noise band. If no consistent threshold could be identified then that model was not applicable.

One of the experimental results they showed was that the consistent outliers obtained from different outlier methods can have a containment relationship. Let C_s, C_d, C_l be the sets of consistent outliers for SPAT, DPAT and LA. Experimentally, it was observed that the relationship $C_d \subseteq C_l \subseteq C_s$ always held. Because there is some ordering to the sets it gives a means of providing a performance measure across the three methods. In particular, in our metric for evaluating the goodness of an outlier method is its resulting yield loss, then this ordering essentially would say DPAT is better than LA which is better than SPAT. Here we do find a free lunch. However, keep in mind that we only find the free lunch by restricting ourselves into looking for consistent outliers. Hence, it is under that constraint of consistency where a free lunch can be found.

6.3 Summary

As stated in Chapter 2, we consider the Bayesian view to be the most practical view to apply machine learning. In that view, the focus is on evaluating the starting assumption rather than on the algorithm to compute the model based on the assumption. Using outlier analysis as the application example, this chapter explains how that idea can be translated into a practical method which can provide great values in various application contexts. The essence of taking a Bayesian view in practice can be seen as implementing a what-if analysis framework where the antecedent with the if-part is an assumption that can be chosen or varied by the user. It is also interesting to note that while outlier analysis is a seemingly local NFL problem where one cannot say one method is universally better than another, under a restricted scope such as finding only consistent outliers, it seems that some free lunch can be found.

Chapter 7

Wafer Map Pattern Recognition

"One cool judgment is worth a thousand hasty counsels. The thing to do is to supply light and not heat."

— Woodrow Wilson

In Chapter 3 we introduced the wafer map pattern recognition (WMPR) problem in the context of the deep learning stage in the evolution of ML. The Wafer Map Pattern Recognition (WMPR) problem is a well researched problem and the research spans both stages in view of the two stages of ML depicted earlier. For example, the authors in [44] developed a comprehensive wafer map dataset called WM-811K. This dataset has 811 thousand wafer maps with 172 thousand labelled. The work in [44] follows a traditional ML approach to building a WMPR and similarity ranking system. They used several types of features, including ones based on Radon transforms and analyzing geometric properties (e.g. failing die counts, region labeling, line detection, etc). A Support Vector Machine (SVM) [45] algorithm was used to build the model. That SVM model had an overall accuracy of 94.63% on the dataset which was better than the deep learning approach which had an 89.64% accuracy.

Several other works had also used the WM-811K dataset. For example, the work in [46] followed a similar approach with the aim of improving multi-pattern detection accuracy by selecting different features. The work in [47] advocated using more discriminant features based on Linear Discriminant Analysis (LDA) to simplify the model building step. With Radon transform based features, the work in [48] proposed a special Decision Tree based ensemble learning method. Decision tree models are generally more interpretable than SVM models.

To help deep learning on WM-811K, the author in [49] proposed using an Autoencoder (AE) as a pre-training step to learn features before learning a classifier. However, the author found that pre-selected features were still needed. This required that feature vectors and not wafer maps images be used as input for the AE.

To apply deep learning, the author in [50] proposed a 2-stage classification process: first to classify between having a pattern and having no pattern (the later is called the "None" class in WM-811K), and if there is a pattern, classify which class it is. Note that both works [49] and [50] acknowledged that the wafer map images were noisy and a de-noising step was required to make the learning work.

WM-811K is a highly imbalanced dataset where some classes have many more samples than others [44]. The work in [51] proposed a special data augmentation method based on GAN (Generative Adversarial Network) [52]. Instead of using GAN, the authors in [53] used AE [54] for data augmentation. Moreover, the authors found that augmenting the samples with rotation could help. In contrast, the authors in [55] used pre-determined methods to augment the dataset and a deeper CNN for training the classifier.

The authors in [56] approached WM-811K dataset from a different angle. They tried to address the concern that a wafer map to be predicted might contain a new pattern not seen in the dataset, or contain a multi-pattern. As a result, the classifier's prediction might not be reliable. The work proposed using Selective Learning to estimate confidence

of a prediction. Then, the deep learning model could include the choice to abstain from making a prediction.

In view of Figure 3.1, the earlier works [44][46][47][48] follow a traditional ML approach. The work [49] makes a transition to a deep learning approach. The later works [50][51] [53][55][56] follow a deep learning approach where data augmentation and sample de-noising are two helpful steps.

We should note that most of the previous works on wafer map classification were done from a manufacturing process perspective. In fact, the problem has been studied in the semiconductor manufacturing field for decades. Most of the previous works appear in publication places in that field. The works reviewed above are those that specifically utilize the WM-811K dataset.

Note that the WMPR problem attracts the interest of a fabless company just in recent years. This is because in the past, ensuring high yield was primarily a responsibility of the foundry and WMPR is an important step in their methodology to improve yield. In recent years, as the technology advances, fabless companies started to share more responsibility of the yield improvement effort. As a result, WMPR becomes also important for a fabless company, enabling them to communicate more effectively with the foundry, e.g. asking more sensible questions.

There is a fundamental difference between how a foundry sees the WMPR problem and how a fabless company sees the problem. From the foundry point of view, for example, providing the WM-811K dataset implies that there is a pre-defined set of pattern classes to begin with. In other words, foundry people have the domain knowledge to enable them to know what patterns they are looking for. Hence, they see WMPR as a multi-class classification problem, as formulated in the WM-811K dataset.

In contrast, from a fabless company point of view, they would not know in advance what patterns to look for. Defining the pattern classes is part of the problem. From this

perspective, we can say that the fabless version of the WMPR problem is not supervised, but unsupervised. This consideration affected how we approached the problem from the start. The novelty of our work is that we are looking at WMPR from a fabless company's perspective. In this Chapter, we will present some of the initial results along the WMPR line of research. Note that there are other results [57][58][59] not included in this thesis, which are discussed in more detail in another thesis [60] parallel to this one.

7.1 Wafer map patterns in yield excursions

In Chapter 3 we discussed the chip production environment shown in Figure 3.2. A chip design will go through a manufacturing process and then, two major stages of testing prior to being shipped to a customer. If everything is proceeding normally, one expects to see yield stabilizing at a certain level without too much fluctuation.

In reality, there will be times when the yield drops precipitously. These drops can be localized to a wafer, a lot or multiple lots. In particularly bad cases it can affect multiple designs that are being produced with the same manufacturing line. These drops in yield are called *yield excursions* and when they happen, people eagerly try to find out the *root causes* in order to correct them.

One of the signature characteristics of a root cause can be the pattern of dies on the wafer that it affects. For some root causes they will be more likely to affect certain locations on a wafer compared to others. For example, in the manufacturing process chemicals are sprayed onto the surface of the wafer with the goal of creating an even symmetric coating across the surface. If the spray nozzle were damaged by being bumped with a wrench, it would have a consistent asymmetric pattern. Any failing dies caused by this asymmetric spray would form a similar pattern on the wafer.

A common procedure is to use a wafer map pattern to define the scope of a yield

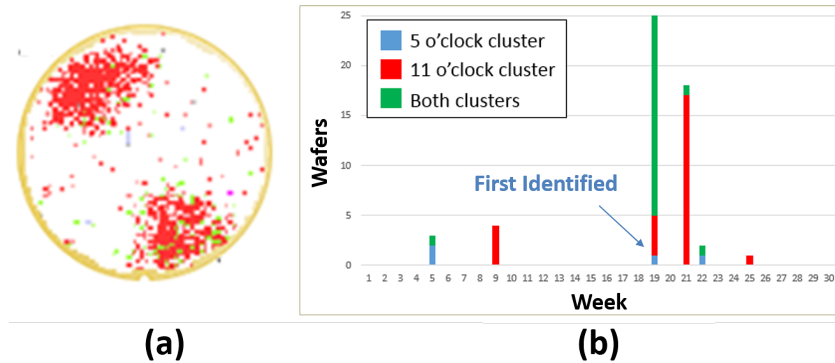

Figure 7.1: Wafer map patterns are used to define the scope of a yield excursion. (a) A wafer map showing the excursion's characteristic wafer map pattern. (b) The occurrence of the pattern over time

excursion. For example, the wafer map in Figure 7.1, plot (a), is one such case. We can see that the wafer has two clusters of failing dies, one at the 11 o'clock position and one at the 5 o'clock position. This wafer map pattern was first reported in week 19 (see plot (b) in Figure 7.1) when all 25 wafers in a single lot showed either the 5 o'clock cluster, the 11 o'clock cluster or both. Once this pattern was reported, the production history of this device was searched and two more occurrences were found in weeks 5 and 9. Later the same pattern was found in weeks 21, 22 and 25.

By defining the scope of the excursion the engineers can know how serious the excursion is (i.e. how much yield loss is occurring) and react accordingly. Additionally a fully defined scope aids in analyzing the root cause of the excursion. Root cause analysis often comes down to what the affected wafers have in common that the unaffected wafers do not. A larger dataset helps rule out potential causes.

In the company we worked with, this pattern search was largely a manual search. An engineer had to load the data, plot it and then look at it; noting which wafers have the pattern. This was a tedious, error prone process and since an engineer had limited time they often tried to reduce the search space.

In this case the engineers looking for the two cluster pattern only searched the pro-

duction history of a single design. Many designs are run through the manufacturing process using the same tools. If one of these tools caused this design's yield excursion, it is possible that another design is also being affected by this same tool. However, this can represent an enormous search space in view of the engineer. In this two cluster pattern case for that 30 week time frame shown in Figure 7.1 (b), limiting the search to 15 potentially affected designs, there are $\tilde{4}$0k wafers that would need to be examined. This is a task that a ML model would be suitable for.

7.2 Cluster patterns

Recognizing the wafer pattern in Figure 7.1 (a), requires that we solve two subproblems: 1) Detecting clusters and 2) Identifying wafers with clusters in similar locations. To detect a cluster such as the one in Figure 7.2 (a), all we have is a list of x, y coordinates where failures occurred and we need to decide if some of these coordinates form a failure cluster. In this context a cluster is not a clearly defined concept. A cluster can be continuous region of failing dies, but there can also be gaps and the edges tend to be diffuse. It does not take a particular shape, it can be circular or ovular but it can also take shapes for which there is no name. The only characteristic a cluster has is that it is a relatively dense region of failures. Therefore, we can try to define a failure cluster by estimating the density of failures.

The field of non-parametric statistics has an analogous problem where they take a list of samples from a random variable and try to estimate the Probability Density Function (PDF) of the random variable. One solution to the PDF estimation problem is the Kernel Density Estimate (KDE) [61]. Each sample value x is replaced with a normal distribution $\mathcal{N}_{x,\sigma}$ centered on the value x. The estimated PDF is the sum of these sample distributions $\tilde{f} = \sum_x \mathcal{N}_{x,\sigma}$. For the KDE solution there is one free parameter σ called the bandwidth

and the user needs to define this value.

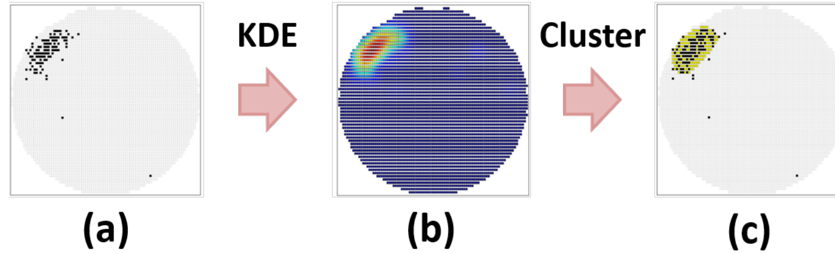

Figure 7.2: (a) Failing dies on a wafer. (b) Kernel density estimate of the failing dies. (c) Cluster formed by thresholding.

The KDE solution can be easily extended to the 2D case needed for wafer failure density. Each failing die's location is replaced with a 2D normal distribution and the total failure density is represented as the summation of all these distributions. We can then calculate the failure density for all the die locations on a wafer as shown in Figure 7.2 (b). We can then define a failure cluster as a continuous region where the failure density is above some value T_c, the cluster threshold. Figure 7.2 (c) shows the cluster as the yellow region in the top left. Note it is important that we require clusters be continuous because each wafer can have multiple failure clusters.

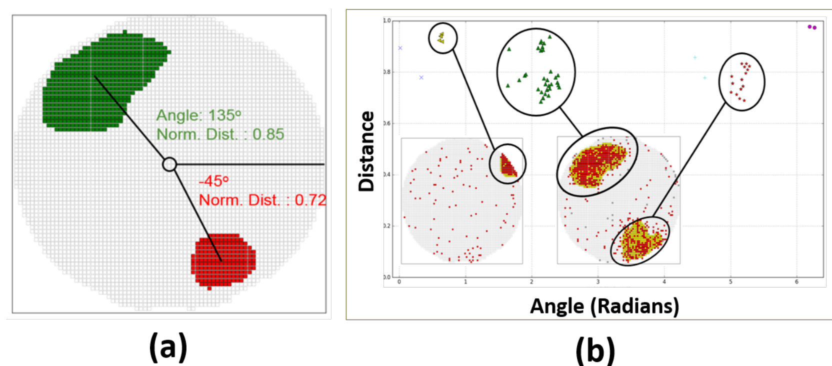

Figure 7.3: (a) Polar coordinates of failure clusters. (b) Grouping failure clusters in the coordinate space.

Just detecting a cluster is the first part of the problem, we also need to find wafers with clusters at similar locations. We calculate the location of each cluster as shown in

Figure 7.3 (a), using a polar coordinate system. A polar coordinate system was chosen because it experimentally worked better. The radial component is measured as the distance from the center of the wafer to the cluster center, where the cluster center is the average location of all dies in the cluster. The angular component is the angle between the x-axis and the line between the cluster center and the wafer center.

With the locations of each failure cluster defined we can use a clustering algorithm to group them together. Each group of failure clusters can then be used to label a new cluster. Plotting the clusters in the polar coordinate space we get Figure 7.3 (b), where we can see how each of the clusters group together.

7.2.1 Cluster pattern results

With these two components defined we can build the model and run it on wafer data to detect this two cluster pattern of the original problem. We selected the 40k wafers from the same time range shown in Figure 7.1 (b). Running the model over that data yields the results shown in Figure 7.4 where the number of detected wafers has increased dramatically. For week 19 the number of wafers almost doubled and for the weeks prior to week 19 the number of wafers went from 5 to 55.

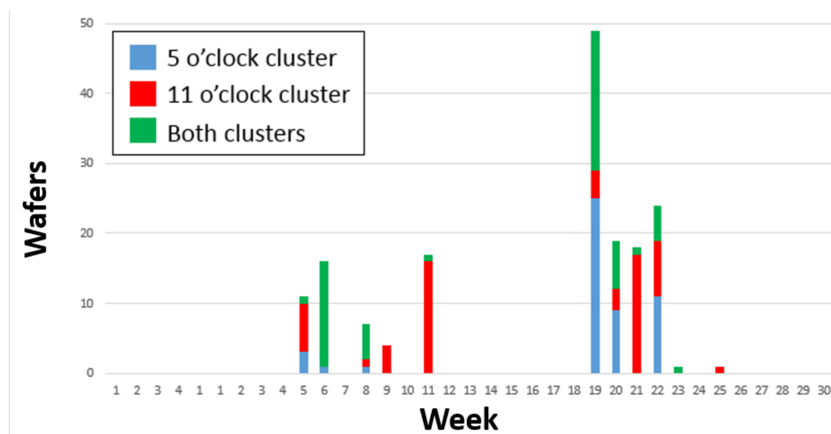

Figure 7.4: ML cluster pattern detection results.

7.3 Sample-based pattern detection

The cluster pattern model is a simple, practical solution that works. However, while it can work well with a cluster type of pattern, it is not a general solution for all patterns. For example, we show two other patterns in Figure 7.5, where the purple dots represent passing dies on the wafer and the yellow dots are the failing dies. The simple cluster pattern model would not work well on capturing those patterns. Also, the cluster pattern detection requires someone to tune the two bandwidths for the 2D kernel density estimation and also the cluster threshold T_c. As the number of dies on a wafer grows very large or very small these parameters will need to be tuned to achieve the best results. This approach also depends on a clustering algorithm, which will also require some tuning.

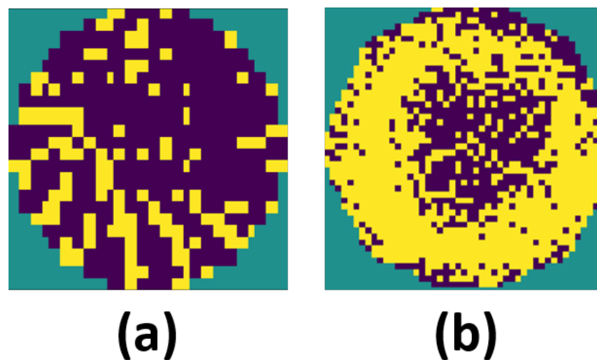

Figure 7.5: Two wafer failure patterns from production data.

In Figure 7.5(a) we see multiple arcing lines characteristic of a Chemical Mechanical Planarization (CMP) root cause. CMP uses an orbital polishing motion to flatten the surface of a wafer layer. Occasionally some debris breaks off from the wafer and is dragged along the surface forming arcing scratches. Knowing the cause of this pattern raises serious questions on the applicability of the cluster pattern model detailed above. First describing each of these arcs as a cluster is dubious. Second the location of these arcs is irrelevant to the pattern. The nature of the CMP process is such, that rotating

this pattern by 90 degrees is also a CMP pattern and should be recognized as such. Yet the cluster pattern detection approach would classify the rotated pattern differently than the original. The pattern in Figure 7.5(b) shows a “ring” shaped pattern. This pattern is interesting because there seems to be three concentrating sub-clusters of passing dies in the center (the reason will be explained later).

What these examples show is that we need a general pattern detection approach, one that makes very few assumptions on what a pattern can be. A domain expert will know that what they see is interesting when they see it, but they will not be able to predict what the pattern will look like or what basic features it would have.

7.4 General wafer map pattern recognition

A more generic approach than the cluster pattern detection algorithm is a Neural Network (NN) or a Convolutional Neural Network (CNN) design. This is the architecture used for modern image recognition ML models such as the designs used in the ImageNet [3] competition. Using an image recognition approach is a natural fit for the wafer pattern detection problem because domain experts use plots to detect interesting patterns and plots can be considered as a special type of image.

One of the difficulties in implementing a CNN is the number of training samples required to produce good results. In the ImageNet dataset there are ∼1,200 images per class in the training dataset [3]. Even reducing that amount by an order of magnitude the required number of samples is impractical for WMPR.

As the first step to approach the WMPR problem, we explored the possibility to use Generative Adversarial Networks (GANs) [52] as one possible solution to this training data scarcity problem. Note that in this initial work, we assume some minimal set of samples are available for training a *recognizer* of the pattern. We do not yet concern the

problem where those minimal set of samples come from. That is a problem completely addressed in [57] (which is discussed in more detail in [60]).

7.4.1 Generative Adversarial Networks

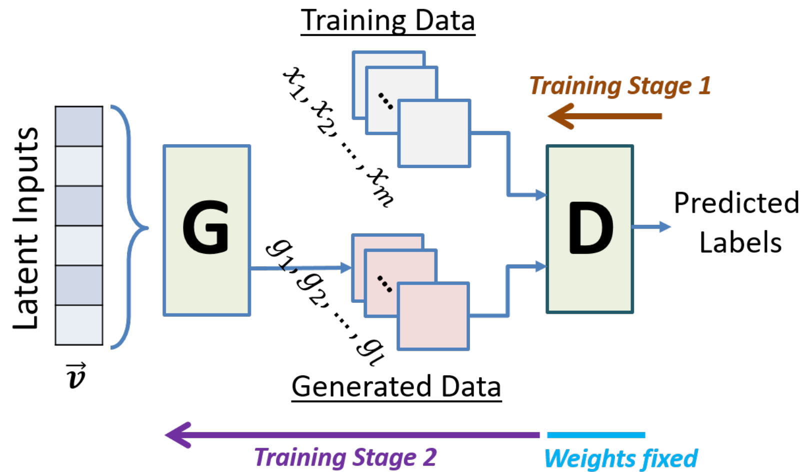

Figure 7.6: Illustration of a GAN and it's training.

Normally a GAN is used to train a generative model. This model will be trained to synthesize new samples similar to it's training data. In the GAN architecture there is the generator network **G** and the discriminator network **D**. Figure 7.6 shows this architecture. These two networks are locked in a competition with each other. The discriminator tries to tell an original sample from a synthesized sample. The generator tries to synthesize a sample the discriminator cannot distinguish from an original sample. While normally the goal of a GAN method is the trained generator, a fully trained discriminator can be used as a recognizer for samples similar to the training data.

To train a recognizer, a few wafer maps are used. Suppose there are m wafer maps and denoted as $x_1, \ldots, x_m$. These are our training data. Without loss of generality, assume each map is a square image. For training, the generator produces some l images, denoted as $g_1, \ldots, g_l$. Each generated image is produced according to a random vector

$\vec{v}$. Each variable of $\vec{v}$ can be thought of as a *latent input*. These variables define a *latent space* where each vector in this space represents an image produced by the generator.

The training process is iterative. Each iteration has two stages and each stage of training can use the common stochastic gradient descent (SGD) approach. In each iteration, two classes of samples $x_1, \ldots, x_m$ and $g_1, \ldots, g_l$ are used. From iteration to iteration, the samples $x_1, \ldots, x_m$ remain the same, but $g_1, \ldots, g_l$ are re-produced by the generator for each iteration based on the weights learned in the previous iteration.

In the first stage of training, the goal is to learn the weights in the **D** network in order to separate $x_1, \ldots, x_m$ from $g_1, \ldots, g_l$ as much as possible. During back propagation, the gradients are computed backward from the output of **D** to its inputs. In the second stage, weights in **D** are fixed. SGD is applied to learning the weights in **G**. The gradients calculated on inputs of **D** are further back propagated to the inputs of **G**. The optimization objective is to have **G** adjust the generated samples such that their output labels by **D** are as close as possible to the output labels of the training samples $x_1, \ldots, x_m$.

The idea of training **D** and **G** can be thought of as playing a game [52] where the **D** network learns to beat the **G** network by discriminating the samples generated by **G** from the training samples $x_1, \ldots, x_m$. On the other hand, the **G** network learns to generate samples to fool the discriminator **D** as much as possible. Over iterations, the generated samples become more like the training samples and it becomes harder for **D** to separate them.

7.4.2 The CNN architectures

In this work, our implementation of GANs is based on two deep CNNs shown in Figure 7.7, following the ideas proposed in [62] which suggests a set of constraints on the architectural topology of Convolutional GANs to make them stable to train.

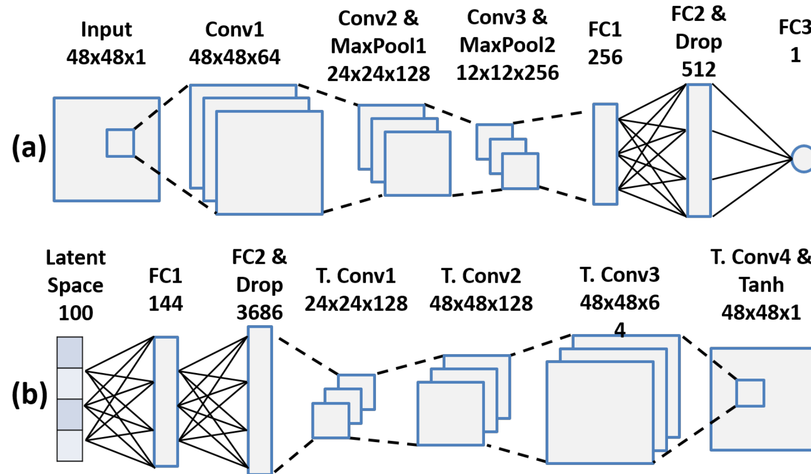

Figure 7.7: The two CNN networks in the GAN: (a) The Discriminator. (b) The Generator.

Figure 7.7 (a) shows our CNN architecture for the discriminator. The leftmost block shows our input assumption. Each input is an image with 48-by-48 pixels. Each pixel has three values: -1, 0, and +1. These three values indicate negative color, no color, and positive color, respectively. Before a wafer map can be used as an input sample to this CNN, pre-processing is required to convert the map into this representation.

The architecture of the discriminator is a fairly standard CNN architecture. There are 3 Convolution layers and 3 fully connected layers. It takes in a 48-by-48 image and then outputs a "synthesized" or "original" label for the image.

The generator's architecture is shown in Figure 7.7 (b). It is the inverse of the discriminator architecture. There are 2 fully connected layers followed by 4 transverse convolution (T.Conv) layers. The transverse convolution layer can be though of as the inverse of the convolution layer, in that it can 'undo' the convolution. The generator takes in a vector of random numbers and then outputs a 48-by-48 image.

7.4.3 An example recognizer for a wafer map pattern

To train our GANs, we need a dataset divided into a training dataset and a validation dataset. Because it is an unsupervised learning, the validation dataset alone cannot fully determine the stopping point. The validation dataset is used to ensure the discriminator does not over-fit the samples in the training dataset, by ensuring that all samples in the validation set are also classified correctly. In our experiments, the stopping point in the training is assisted by inspecting the samples generated by the generator. If these samples show features similar to the training samples, then we stop. If not, the training is resumed for more iterations.

Because our focus is on the discriminator (used as a image recognizer), we concern more about the quality of the discriminator than the quality of the generator. If the latter is our concern, we might need to train with more iterations until the generator is capable of producing images close to the training samples. This in turn might require additional techniques in the implementation to ensure convergence.

What we found is that the GANs usually do not require a large number of samples to train if those samples share some common features. To illustrate this, Figure 7.8 shows five training samples used for training a recognizer for this class of wafer pattern. To enhance the training dataset, each sample is incrementally rotated to produce 12 samples in total. Then, overall we have 60 samples for training.

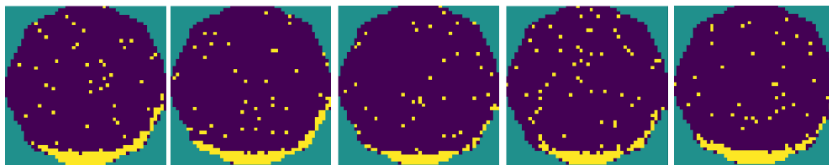

Figure 7.8: Five training samples.

Figure 7.9 shows the five samples used for validation. Similarly, each is rotated to produce 12 samples with a total of 60 validation samples. Note that these samples look

alike because these are wafers from the same lot.

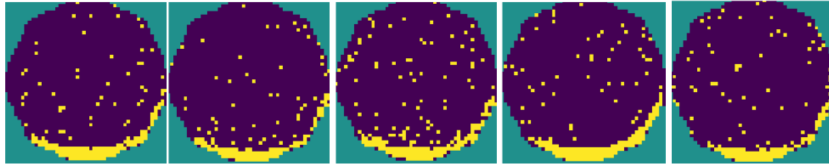

Figure 7.9: Validation samples.

After the training, the discriminator (treated as our pattern recognizer) is used to recognize similar wafer map patterns on 8300 other wafers. The recognizer recognizes 25 wafers and some are shown in Figure 7.10. On these samples, we see that they all show up with a similar edge pattern.

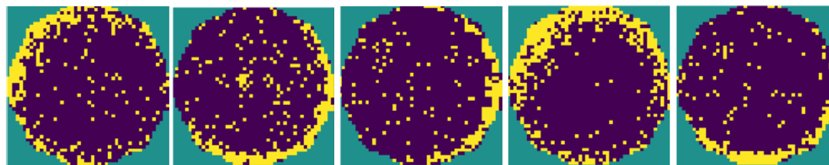

Figure 7.10: Five wafers among the 25 recognized wafers.

Because the samples generated by the generator are inspected to determine the stopping point, it would be interesting to show the wafer maps produced by the generator. Figure 7.11 shows five such wafer maps (by giving the generator five random inputs). It can be seen that the generated images do not look the same as the original images shown in Figure 7.8. However, the feature of having many fails on the edge are also present in these generated images.

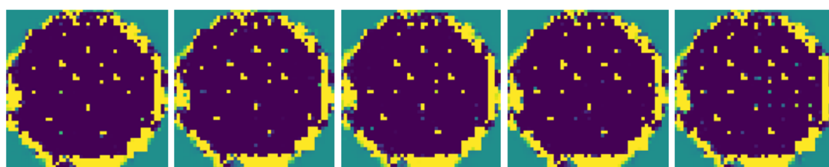

Figure 7.11: Wafer maps produced by the generator.

7.4.4 Generality of a recognizer

The wafers used in the above experiment each has about 2100 dies. Recall from Figure 7.7 that input images are in size of 48×48 pixels. One interesting question to ask would be how the recognizer performs on those wafer maps from other product lines where each wafer map is based on more or has less number of dies.

To answer this question, Figure 7.12 shows the result of applying the edge pattern recognizer on a 2nd product line. The recognizer was applied to scan 2011 wafer maps and found only 1 recognized map as shown in the figure. Each wafer for this product line has about 4500 dies, more than twice as many as that in the first product line used for the experiment above. It is interesting to see that the recognized map also shows a clear edge pattern.

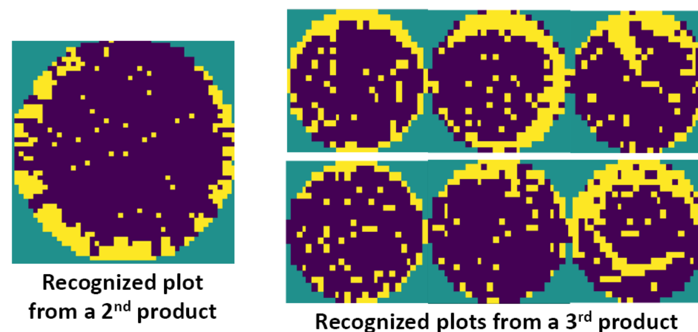

Figure 7.12: Recognized maps from the 2nd and 3rd product lines.

Then, the recognizer was applied to a 3rd product line to scan 7052 wafer maps and found 24 recognized wafer maps. Some examples are also shown in Figure 7.12. For this product line, each wafer has about 440 dies, much less than that in the first product line. However, each recognized wafer maps also show a clear edge pattern.

From the results shown above across three product lines, it is interesting to see that the pattern recognizer, trained with a small set of rather similar edge patterns as shown in Figure 7.8, is able to generalize the learning and recognize other edge patterns even

with some noise in the pattern (Figure 7.12). These results show that one might not need to re-train a recognizer for every product line even though their numbers of dies per wafer are different. These results also indicate that one can train a recognizer with some higher-level “intent.” For example, in the above, our intent is to capture an “edge pattern.”

7.4.5 Developing a set of recognizers

We can develop a set of recognizers that will recognize most of the wafer maps. Recall from Postulate 1 that the domain expert is performing a search to look for new interesting samples. Creating a set of recognizers can aid in this search by filtering out all of the wafer maps for known classes. The remaining unrecognized maps can then be ranked, for example based on their yield loss. Recognized maps can be interesting or non-interesting. For developing the recognizers, we can begin with those non-interesting maps, followed by developing the recognizers for interesting maps, and then followed by applying a ranking.

There are two main reasons for following this ordering. First, there are usually many more non-interesting maps than interesting ones. Hence, to begin with, there are more samples for training a non-interest pattern recognizer. Second, we know that in DSML problems we will not have a concrete idea of what to look for in advance since the problem is under-specified. It would be easier to randomly pick a few non-interesting maps (because there are many) and ask a domain expert to verify their non-interest. After majority of the non-interesting maps are filtered out, the remaining set is much smaller and easier for a domain expert to select those interesting examples for training an interesting pattern recognizer.

7.4.6 Developing a sequence of recognizers

Recognizers discussed in this section are developed based on wafer maps from the product, a medical product with a dataset of 8300 wafers. The training follows the approach discussed in Section 7.4.3 where five wafer maps are used as the training samples and five wafer maps are used as the validation samples.

Figure 7.13 shows four of the five training samples and four of the five validation samples. After the training, the recognizer is used to scan the rest of the 8300 wafer maps. Four example recognized maps are also shown in the figure.

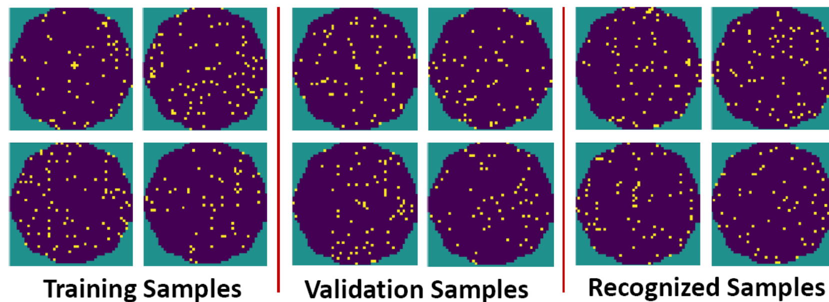

Figure 7.13: Training, validation, recognized samples for the 1st recognizer.

As seen in Figure 7.13, the first recognizer is trained to recognize the pattern that can be called “sparse and random failing” (i.e. “low-density random failing”). Majority of the wafer maps are in this class. Then, samples shown in Figure 7.14 are among those not recognized by the 1st recognizer. The left four are among the five samples used to train the 2nd recognizer. The middle four are the validation samples. The right four are example maps recognized by the 2nd recognizer. More than 88% of the wafer maps are recognized by the first two recognizers. As a result, less than 12% of the maps remain after applying the first two recognizers.

Wafer maps recognized by the 2nd recognizer look similar to those recognized by the 1st recognizer. However, notice that the 2nd class of maps generally contain more failing dies. In theory, the two classes of samples can be combined to train a single recognizer.

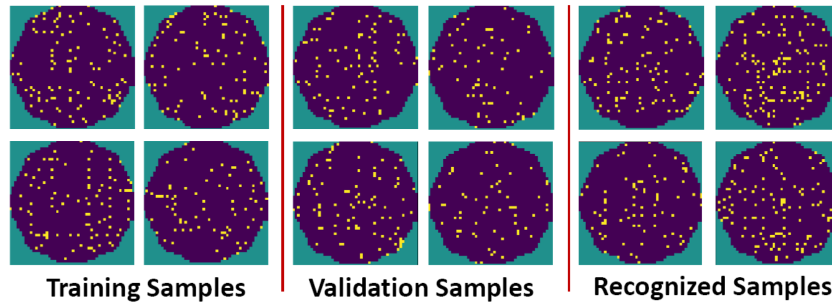

Figure 7.14: Training, validation, recognized samples for the 2nd recognizer.

However, because of the difference in their failing density, it requires more samples and would take longer to converge. We separate them into two recognizers to simplify the training.

Those maps picked up by the first two recognizers are considered non-interesting. The 3rd recognizer is trained to recognize a high-density failing pattern as illustrated in Figure 7.15. This class of wafer maps might or might not be interesting, depending on whether they spread out randomly or concentrate on the same lot.

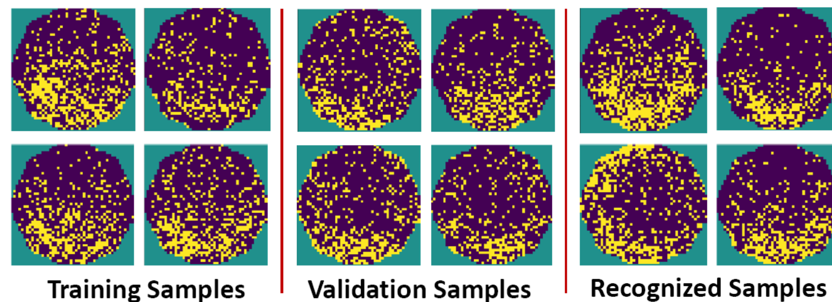

Figure 7.15: Training, validation, recognized samples for the 3rd recognizer.

The 4th recognizer is trained to recognize a grid pattern as shown in Figure 7.16. This class of wafer maps is interesting and may indicate an issue in the test probe. There is also a 5th recognizer which is the edge pattern recognizer already discussed in Section 7.4.3 before.

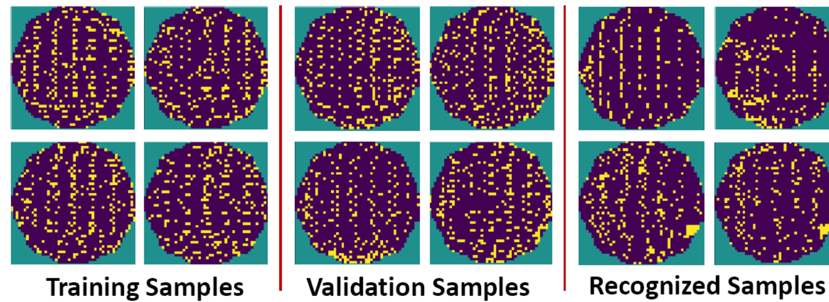

Figure 7.16: Training, validation, recognized samples for the 4th recognizer.

7.4.7 Six remaining novel patterns

After applying the five recognizers, about 5% of the wafer maps remain unrecognized. They include both non-interesting maps and novel maps. About 70% of these can be grouped into one of the six patterns as shown in Figure 7.17. A pattern can contain from at least 10 to over 100 wafer maps and hence, not only the patterns are novel, but also they appear systematically.

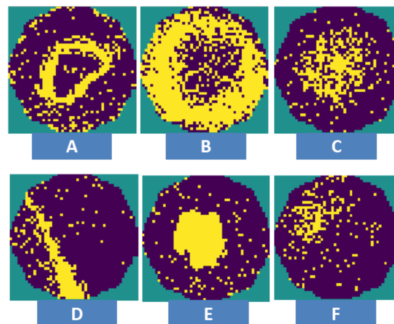

Figure 7.17: Six novel classes of wafer maps discovered.

7.4.8 Detecting an issue with a production tool

As an example, there are a number of wafer maps sharing the same pattern represented by the class B pattern shown in Figure 7.17. In fact, this wafer map is the same as that presented in Figure 7.5(b) before. Further investigation reveals that the issue

was related to the 3 lift bins used by the Gasonics asher tools. Hence, detecting the patterns as those shown in Figure 7.17 proved to be useful in practice. An experienced process engineer could look at a pattern and start forming guided hypotheses to check the related manufacturing equipments. In practice, the recognizers help sort out majority of the wafer maps and bring the attention of an engineer onto those novel wafer maps.

7.4.9 Non-interesting wafer maps missed by the recognizers

Figure 7.18 shows some additional examples missed by the recognizers and not in the six pattern classes. These can be deemed as non-interesting wafer maps. Notice that the failing locations look also quite random as those maps shown in Figure 7.14 but the density of the failing is between those shown in Figure 7.5(b) and those shown in Figure 7.15. These missing maps suggest that a 6th recognizer might be developed to recognized a "medium-density random-failing" map. The decision to develop an additional recognizer mostly depends on the number of maps available for the the training and validation.

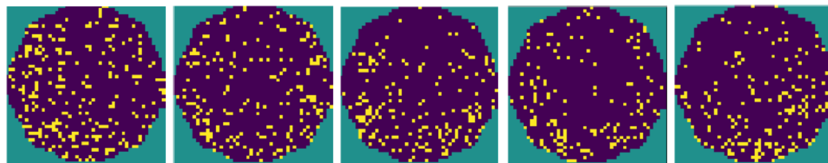

Figure 7.18: Non-interesting wafer maps missed by the recognizers.

7.4.10 Generality of the recognizers

As pointed out in Section 7.4.4, a recognizer developed for a product line can be applied to the wafer maps from another product line, even though their numbers of dies on a wafer are quite different. The five recognizers above are based on 1st product line. Next, we explain the result by applying these recognizers to wafer maps from the 3rd product line which has about one fifth of the dies on each wafer.

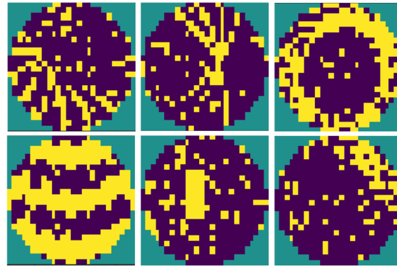

Figure 7.19: Novel patterns discovered on the 3rd product line.

The result is that more than 84% of the wafer maps are recognized by the first two recognizers, comparing to the 88% number mentioned above for the 1st product line. As seen, these two numbers are comparable. After applying the additional three recognizers, similar to the result for the 1st product line, for the 3rd product line also about 5% of the maps remain unrecognized. Figure 7.19 then shows examples of novel maps found on the 3rd product line. As seen, these maps show different novel patterns than those shown in Figure 7.17 above.

7.5 Summary

Identifying a new pattern on a wafer map can be a start for a more comprehensive analysis. In the analysis, one task is to search for all wafer maps with a similar pattern. In this context, a ML model can be handy to facilitate the search. In the study summarized in this Chapter, we assume a minimal set of training wafer maps to begin with. Then, the goal is to learn a recognition model for the pattern exhibited on the small set of wafer maps. We found that as few as five wafer maps are enough to train a good recognizer.

The emphasis in our work is that we do not approach the WMPR problem as those previously done before, solving it as a multi-class classification problem. Our suggested methodology is to develop a sequence of pattern recognizers, iteratively. This iterative approach is more consistent in view of a DSML setup where we try to work with a

limited set of data samples at hand (and at the moment). By enabling training a pattern recognizer with five samples, we can provide a tool that is useful in view of an iterative DSML process. More importantly, because recognizers are built independently, the set of recognizers can be naturally accumulated over iterations. This is one benefit that a multi-class classifier cannot provide. If one encounters a new class in the future, the classifier needs to be re-trained.

Chapter 8

Experience From Developing A Machine Learning Solution

"It takes considerable knowledge just to realize the extent of your own ignorance."

— Thomas Sowell

In the previous chapters, we discuss DSML from a research perspective and review two sets of studies, one in the context of outlier analysis and the other in the context of WMPR, to illustrate various important challenges in view of DSML as well as strategies to mitigate those challenges. In this chapter and the next, we discuss DSML from a practical perspective. We summarize our experience in trying to implement a ML software solution with the end goal to deploy the solution in production, as well as to make a measurable impact in the company, e.g. improving their productivity or saving some cost. The story in this chapter is not a successful one. However, it is through the unsuccessful experience that we had learned some important lessons about DSML from a practical perspective, which could be valuable for a future development of a practical DSML solution.

In 2019 we had already completed an implementation of the cluster pattern algorithm discussed in Section 7.2 as a standalone software tool for the company. We were looking for an opportunity where this tool could be used by engineers. We engaged a Device Engineering team and tried to develop a complete software solution to help them in their day-to-day tasks.

The close interaction we had with the Device Engineering team provided insights into their job, what tasks were hard, what tasks were easy, as well as the methodology they used in solving their problems. Additionally, the team could provide real-time feedback on the software including their thoughts and how they interacted with it.

This work experience with the Device Engineering team enabled us to learn important lessons on how to make a DSML solution successful. In short, we eventually realized that even though one could have a good grasp on the theories and concepts for practicing ML or DSML, there could still be unanticipated concerns and constraints that render the ML aspect unappealing to a user. The consequence is that, while leveraging ML technologies might be the original motivation for develop a DSML solution, because of those concerns and constraints, we could end up with a solution where the ML aspect is irrelevant. In the following, we describe an experience of such.

8.1 Manufacturing process

In order to understand the tasks of a Device Engineer we first need to provide a little more detail about the manufacturing process. The tasks that a Device Engineer are expected to do, are related to events that occur inside a manufacturing process. Overall, their job is to ensure the yield is satisfactory and also ensure the production line is run with as the highest throughput as possible. Both are important metrics that affects the profitability of a production line.

8.1.1 Manufacturing stages

As we have stated before, the manufacturing process builds a design in layers. It starts with the transistors as the bottom layer, then it builds layers and layers of metal interconnects. Each layer is built in *stages* and each of these stages can be classified by the process used at that stage. These processes are: Deposition, Chemical Mechanical Planarization (CMP), Etch, Implant and Cleans.

We can understand these processes by looking at an example of manufacturing a layer of a design. In Figure 8.1 we show the steps needed to build the second layer of metal interconnects, denoted as M2. It is important to note that these figures are shown in cross-section so we are looking at the wafer edge-on as the layers are built upwards. In Figure 8.1 (a), we see metal layer 1, denoted as M1, the wafer material is shown in gray and the metal is shown in a copper color. All of the wires in the same metal layer are parallel to each other and perpendicular to the next layer. From this perspective the M1 wires are left-right oriented on the wafer, the wires in the M2 layer will be in-out oriented.

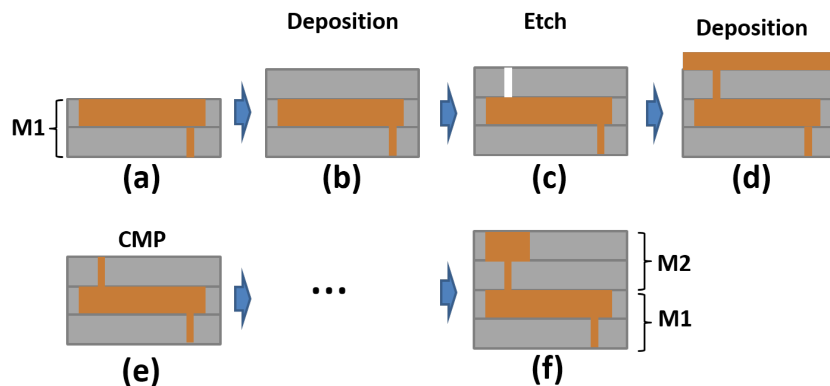

Figure 8.1: Manufacturing steps in building metal layer 2 (M2) on top of metal layer 1 (M1). Shown in Cross-section.

In the top-center of Figure 8.1 (a), we see a single wire traveling from left to right.

On the right end of the wire we see a piece of metal extending downwards to the layer below, this is a *via* and it's purpose is to electrically connect two layers. In this case we can think of the via connecting the right end of this wire to a terminal on a transistor.

The first stage in building the M2 layer is to deposit wafer material on top of the M1 layer. As shown in Figure 8.1 (b), this buries the entire M1 layer, electrically isolating it from the wires we will build in the M2 layer. We do need to connect the M1 and the M2 wires, so within this wafer material we just deposited vias will be built to connect the two layers.

In order to construct these vias we need to bore holes through the wafer material. This is done through the Etch process, Figure 8.1 (c). The exact details of the Etch process are not necessary, we only need to know that we can remove wafer material in very specific locations.

After the via holes are created we can then use metal deposition to fill them. We show in Figure 8.1 (d), the filled via hole and a thin layer of metal coating the wafer. This metal coating results from the coarse nature of deposition and will have to be removed.

The CMP process is used to remove this thin layer of metal. It will both polish off this extra metal and flatten the surface of the wafer. Figure 8.1 (e) shows the completion of the CMP stage. What remains is a flat surface with the vias peaking out ready to connect to the M2 wires.

From just building the vias we have seen all the steps necessary to build a metal layer. These steps shown in Figure 8.1 (b), (c), (d) & (e) used to build the via are repeated to form the M2 wires. The result is shown in Figure 8.1 (f) where we can see the end of an M2 wire (with it's in-out orientation) on top of the via we just built.

Following this simple example shows us the Deposition, Etch and CMP processes, there are two that have not be discussed yet: Cleans and Implant. A Cleans process removes oxidation, debris and chemical contaminants from the wafer. In the metal layer

example shown in Figure 8.1 a cleans stage would sit between each of those stages displayed in that figure.

The Implant process is a specialized process used only for the transistors. A transistor has two *wells*: a P-well and an N-well. These wells are necessary for the transistor to function. The P-well and N-well are essentially the wafer material that has been *doped* with atoms from another element such as Boron or Phosphorus. The Implant process is how these P-wells and N-wells are doped. They ‘implant’ the atoms into the well.

8.1.2 Manufacturing tools and data

Chaining together all these stages we can bit-by-bit create the transistors and then the metal interconnect layers and complete fabrication of a whole design. Each of these stages in the manufacturing process corresponds to a manufacturing tool. There is a tool for the deposition of wafer material, there is a different tool for the deposition of metal, and there is a tool for CMP. These tools are highly automated, when everything is functioning well a technician will load a lot into the tool and then start the process. The tool will begin to process each wafer in-turn and indicate when it has completed the entire lot. The technician can then remove that lot from the tool and transport it to the next tool for the next stage of the manufacturing process.

Manufacturing tools are highly varied taking many shapes and forms, but Figure 8.2 is a useful visualization of a tool. A tool can be built with a central chamber and multiple side chambers. Side chambers allow for multiple wafers to be processed in parallel, increasing throughput. Wafers that are loaded into a tool will be moved with a robot arm through the central chamber to a side chamber. In that side chamber the wafer will be isolated and the process for that stage can begin. A second wafer can be moved into a second side chamber etc. Figure 8.2 shows 6 side chambers and the tool

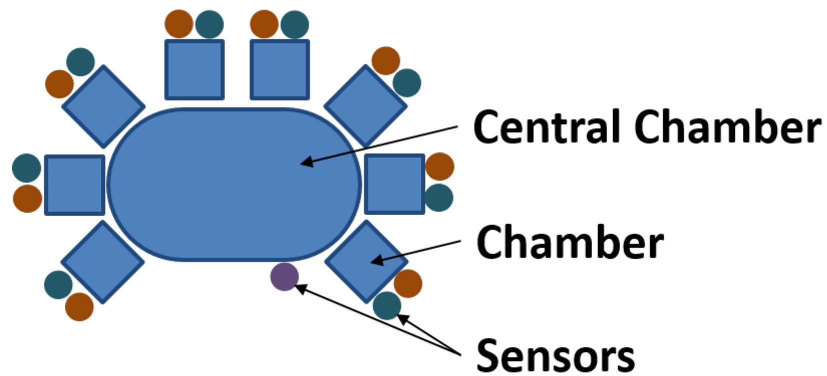

Figure 8.2: A manufacturing tool with a central chamber and six side chambers where the actual process takes place. Each chamber will have several sensors for monitoring purposes.

can process 6 wafers in parallel.

Each of the chambers within a tool will have sensors attached to it for monitoring purposes. What these sensors monitor will depend on what is critical to measure for that process and tool. For example some tools include spraying an inert gas onto a wafer in order to control the temperature of that wafer during the process. For a tool like this there will be several temperature sensors. Engineers monitor the ambient temperature of the chamber and they would also want to measure several points on the surface of the wafer. Since gas is used, an engineer may also want to monitor the flow rate and pressure of that gas for the duration of the process.

The primary purpose of this tool sensor data is to enable real-time monitoring of the chamber. If something goes wrong this data can raise a warning immediately, giving the technicians a chance at fixing the issue and preventing any more wafers or lots from being affected. For example in a tool that uses inert gas to control wafer temperature a warning can be raised if the gas line pressure exceeds known-good thresholds.

8.1.3 Defect data

If we stop a wafer in-between stages and look at it's surface with an electron microscope we will find many defects such as particle contaminants and circuit abnormalities. A particle contaminant can be seen in Figure 8.3 (a), where the particle is sitting on the surface of the wafer above four wire trenches. This picture was taken after an Etch stage but prior to the metal deposition stage. Figure 8.3 (b) shows a contact from the top layer of the wafer and above the contact is a circuit deformation. What this deformation is can be hard to ascertain without further analysis, but we do know that this is undesirable and may cause this die to fail.

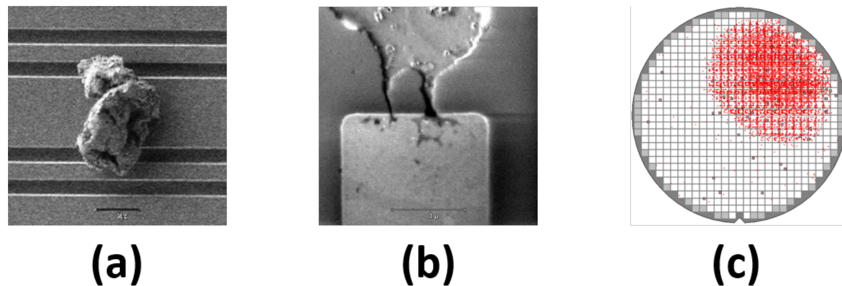

Figure 8.3: Visual defects identified on a wafer. (a) Particle defect (b) Circuit deformation (c) Defect location on a wafer.

To be clear, just looking at a defect on the surface of the wafer does not tell us that the die will fail later during testing or after delivery to the customer. Some of these defects, such as the particle in Figure 8.3 (a) may not have any adverse affects to prevent a functional chip. While it looks like that particle will hinder metal deposition, it is far from certainty. The particle could be dislodged from it's location by being moved to the next stage, or from the next process itself. Additionally modern designs have quite a bit of redundancy built into them. Without knowing how critical those wires are to the function of the design we cannot know the affect on the die. While we may not be able to look at a single defect and know that a die will fail testing, engineers in the

manufacturing process find that the amount of defects correlates to the "health" of the process. More defects for a stage indicate that the process is poorly controlled and more dies are likely to fail.

Gathering this defect data requires that the process be automated. A lot of wafers will be pulled from the manufacturing line and then inserted into an electron microscope. The microscope will then begin to image the surface of a wafer. It will scan across the wafer like reading a book: left-to-right, top-to-bottom.

As the microscope creates an image of the surface, that image will be analyzed immediately. The task is to identify defects and if an image does not have a defect then it is clean data and will be discarded. Various specialty computer vision algorithms will analyze each image and identify if the image contains a defect. These algorithms can also provide a coarse categorization of the defect into so called *defect bins*.

When a defect is identified, the electron microscope will take several images of the defect. Some images will be high resolution of the defect itself, and some images will be zoomed-out showing more of the circuit around the defect. Additionally the location of the defect on the wafer will be recorded. These locations can be plotted in a defect map, such as that shown in Figure 8.3 (c), where each defect location is plotted on a wafer. Similar to wafer failure patterns, these defect maps can be of enormous benefit to understanding issues within the manufacturing process.

8.2 Lot disposition

As we have mentioned before, occasionally a manufacturing tool will raise a warning based on it's sensor data. Technicians will begin remediation, trying to fix the issue and prevent the issue from reoccurring in the future. In parallel to remediation is *Lot Disposition.* Since the tool raised a warning, the lot being processed is immediately a

suspect. It is possible that the wafers have been irreparably damaged and none of the chips from this lot will pass testing. In this case it is not economically viable to continue to process this lot. Lot disposition is where the decision is made to either *release* this lot or to *scrap* all or part of the lot.

Generally, only a lot that has been *flagged* will be dispositioned and lots can be either automatically or manually flagged. We have already seen that a manufacturing tool can automatically flag a lot based on sensor data. A lot can also be automatically flagged after Class Probe or Wafer Probe testing if there are too many test failures. Additionally, at Wafer Probe if there are too many failures of a particular failure mode/bin it can also be flagged for disposition by a Statistical Bin Limits (SBL) system.

Reasons for manually flagging a lot can vary widely but it is generally related to human actions that are not easy to measure or monitor. For example, a technician was transporting a lot from one tool to another tool, moving it in a lot carrier. Picking up the lot carrier, the technician did not get a firm grip and the lot carrier started to slip part-way through transportation. The technician was forced to set the lot onto the floor contaminating it with dust particles. This lot was held back from processing and manually flagged for disposition.

A lot can be flagged anywhere from the first step of the manufacturing process all the way to Wafer Probe testing. However, there are two types of dispositioning depending on if the lot is in the manufacturing process or has completed it. While a lot is in the manufacturing process only the entire lot or a subset of wafers can be scrapped. It is not possible to manufacture half a wafer. After a lot has completed manufacturing and reached Wafer Probe, it is possible to scrap individual dies by *force inking* them. These force inked dies will not be packaged and will not continue to Final Test, which are considered as two expensive steps.

For an engineer working on a lot disposition inside the manufacturing process they

have to work at the granularity of a wafer. They must weigh the costs of scrapping the wafer and losing viable chips against the benefits of saving material and manufacturing time on that wafer. Sometimes there is a reason to be concerned for a wafer, but not enough to justify scrapping the entire wafer. For example, the lot that was set on the floor may yield good chips after the dust particles are cleaned off it. However, we do not know how gently the technician set the lot carrier down. It's entirely possible that instead of gently setting the lot on the floor, they dropped it causing physical damage to the wafers.

If we still remain suspicious of a lot, but there is not sufficient evidence to scrap any of the wafers, then that lot will be provisionally released and flagged for another disposition after Wafer Probe. This means that the lot will continue through the manufacturing process as normal. When the lot has completed the Wafer Probe testing, it will be placed *on hold* for dispositioning. Unless the lot is flagged for some other reason the engineer performing the disposition must wait for the lot to be placed on hold, before beginning.

8.3 Disposition methodology

One of the responsibilities of a Device Engineer is lot disposition. The team we were embedded with primarily focused on dispositioning lots of wafers after they reached Wafer Probe testing. The methodology has three steps: 1) prioritize lots waiting for disposition & select one, 2) Investigate the lot, 3) Release the lot, force inking the "bad" dies.

We show an overview of the lot disposition methodology in Figure 8.4. To complete lot disposition multiple in-house and commercial tools were used to manage the disposition status, complete the investigation and scrap "bad" dies, wafers or lots.

To begin the lot disposition an engineer must navigate to an internal company website

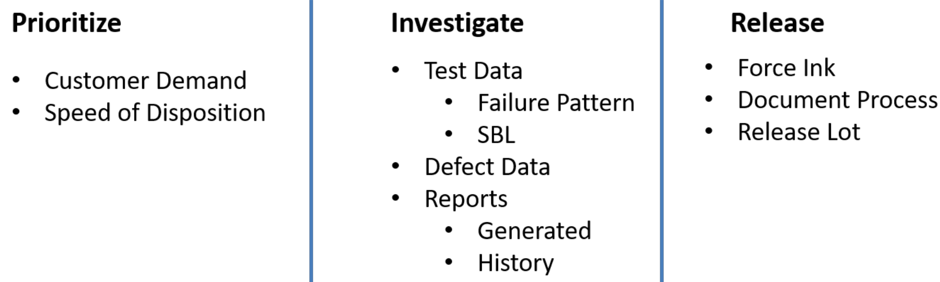

Figure 8.4: Overview of the disposition methodology.

that lists all lots on hold for disposition. The list contains information about the lot such as the lot's id and the design's id. There will also be some where-when-why information on the disposition hold. This information will include: at what step in the manufacturing process the lot is being held, when it was placed on hold, a brief message about why it is being held, and links to any relevant reports including reports from prior dispositions.

Based on this information the engineer can prioritize these held lots and select one to begin disposition. Usually priority is given to lots based on customer demand. For example, if the company is reaching the deadline of an order and needs to meet a quota, lots that contribute to that order will be prioritized over lots that do not. Another reason a lot may be prioritized is based on the expected time to disposition that lot. Quick (turn-around) lots may be prioritized because they fit within the current schedule of the engineer and likewise long (turn-around) lots may be dispositioned when there is enough time available.

Once a lot has been selected for disposition, the investigation can begin. The purpose of the investigation is to uncover evidence that certain dies are likely damaged and will have a shortened lifespan. With this evidence the engineer can justify force inking this dies.

The general strategy of the investigation is to identify areas of the wafer that have been affected more than others. The technique used is to look for patterns on the wafers.

For example, we have seen wafer failure patterns previously. When die failures form a cluster we know that the area where the cluster exists has been detrimentally affected. However, experience has shown that dies near that failure cluster may have barely passed testing and we should be suspicious of their quality. A later example will expand on this with the defect data.

The first step in the investigation is to review the history of the lot as documented in the disposition reports. The content of these reports will guide the investigation and provide the background context in which to interpret all other data. For example, if the report says a sensor warning was raised for a CMP machine at stage X, they will definitely look at the defect data for that stage. The CMP machine also has wafer failure patterns that are associated with it, so the engineer will be actively looking for anything that appears like those patterns.

The second step in the investigation is to look at the wafer probe test data. The wafer failure patterns for all 25 wafers in the lot are plotted in a 5x5 grid. We show an example of this 5x5 plot in Figure 8.5 (a) where the green dies are passing dies and any other color indicates the die failed testing. The disposition engineer can get an idea of failure patterns for each individual wafer and the lot as a whole. They are looking not only for patterns on each individual wafer, but also looking for relationships between wafers. For example we have marked five wafers with a blue *, these wafers show a cluster of failures on the left side of the wafer. We have also marked a wafer with a red *, this wafer has some failures on the left side, but there may not be enough to support calling it a cluster. However, with the context of the other five wafers we may be willing to consider that this wafer has a cluster that is not fully formed.

This type of analysis, looking for patterns across wafers, is called a z-axis analysis. The terminology comes from thinking of wafers in a stack. The x and y axes move across the surface of a wafer and the z-axis moves through the stack of wafers.

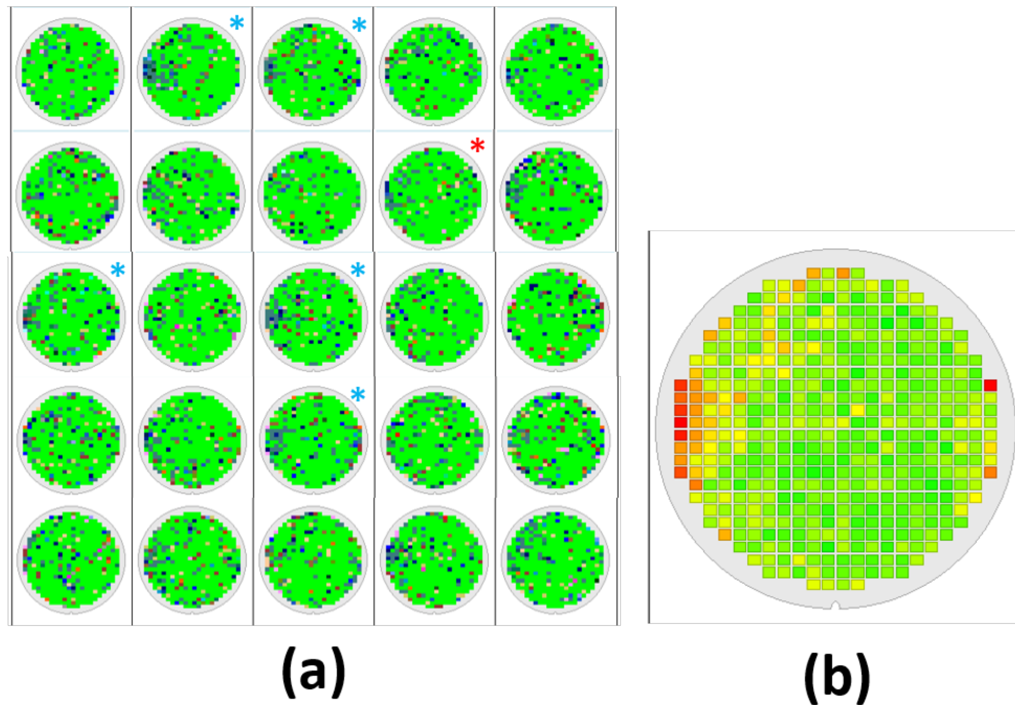

Figure 8.5: Wafer failure patterns for a lot. Green dies are passing. (a) Shown in a 5x5 grid. (b) Shown as a stacked wafer plot.

Z-axis analysis is commonly associated with a stacked wafer plot which we show in Figure 8.5 (b). What a stacked wafer plot indicates is the yield for that x, y location across the wafers in the lot. A red color indicates that almost all dies at that location failed testing and a green color means most passed. Looking at Figure 8.5 (b) we can see what we already knew: this lot has many failures on the left side of the wafer. We can also see something new, two locations on the right side of the wafer have a lot of failures as well. Depending on what else was uncovered in the investigation and the quality demands of the customer an engineer may look at this stacked wafer plot and decide to force ink all dies in those two locations.

The test data will also contain some information of the failure modes, these are denoted as *bins* or *bin numbers*. In Figure 8.5 (a) the failure modes are indicated with color,

a different color indicates a different failure mode. Since they are a coarse categorization of why each die failed, they can be used to evaluate the extent of a failure pattern. A wafer will usually have some amount of random failures in it and the boundaries of a sparse pattern can be blurred by these random failures. Using failure modes can sometimes provide a clearer picture of where the random failures are and where the failure pattern ends.

At this stage in the methodology the disposition engineer may also consider which of the wafers can be scrapped entirely. It is a rare and extreme decision, but sometimes the engineer will conclude that none of the dies from a wafer are worth saving and an entire wafer will be scrapped.

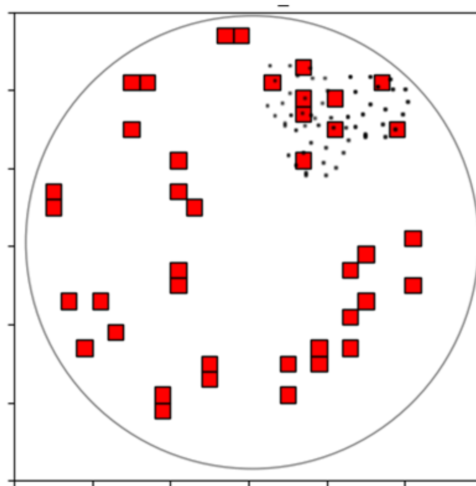

Figure 8.6: Correlating defect data with failure patterns.

The next dataset examined is usually the defect data. Defect data, like the failure mode data is used to identify the extent of a failure pattern. The disposition engineer will look for defects that form a pattern that correlates to the failure pattern. For example, we show in Figure 8.6 a super position of a wafer failure pattern (red squares, result from testing) and defect locations (black dots, result from defectivity analysis during the manufacturing process). In the top right there is a correlation between the defect

locations and a small cluster of 9 failing dies. This correlation indicates that dies in this area around those 9 failures are also likely affected and should be force inked.

Once all this information has been looked at and the engineer has a good idea of the history of the lot and what patterns are appearing on each wafer in the various datasets, they will synthesize the information and begin forcing inking dies. Force inking is a relatively straight-forward procedure. An in-house program is opened and the lot data is selected, the engineer will be able to select a single wafer. At this stage to force in a die the engineer simply needs to click on that die. After they have inked out all the dies a forced ink file can be saved to disk. This file is uploaded into a management system so the tools at the packaging step can know not to package these dies.

After the engineer has confirmed that the force inking had taken effect, they can release the lot via another website and begin documentation. Everything that was done was documented in their own internal reports, saved for future reference. From this documentation they created an executive summary that was required to be submitted to 3 different websites for 3 different teams.

The whole disposition process for a single lot is not a very time consuming process. If everything goes smoothly the entire methodology from the prioritization to writing the reports can be done in about 30 minutes. A typical day may have a Device Engineer dispositioning 2 to 6 lots. Meaning it would not be unusual for the Device Engineer to spend the morning dispositioning lots prior to attending to their other responsibilities.

8.4 The disposition software system

Before engaging with the team and understood the detail of a Device Engineer's task in lot deposition, we knew, on the surface, that wafer map patterns are somehow involved in a decision making process. Naturally, we would think that a software tool

with automatic wafer map pattern recognition could be useful in the application context. Hence, for the team there is an opportunity to develop and deploy a "ML solution" that can have a real positive impact.

Of course, this naive thinking was before our engagement with the team and started building the software system where its various aspects and functionality were guided (or requested) by the engineers. The system we built for the disposition team was a web-based application. Figure 8.7 shows the main interface. The idea was to create an augmented list of lots where all the data needed for the investigation was in one central location. This was an important request. With everything in one location the engineers work load is reduced. They no longer have to load multiple tools or navigate to multiple websites, each of which requires login credentials and a database query before the correct information can be displayed.

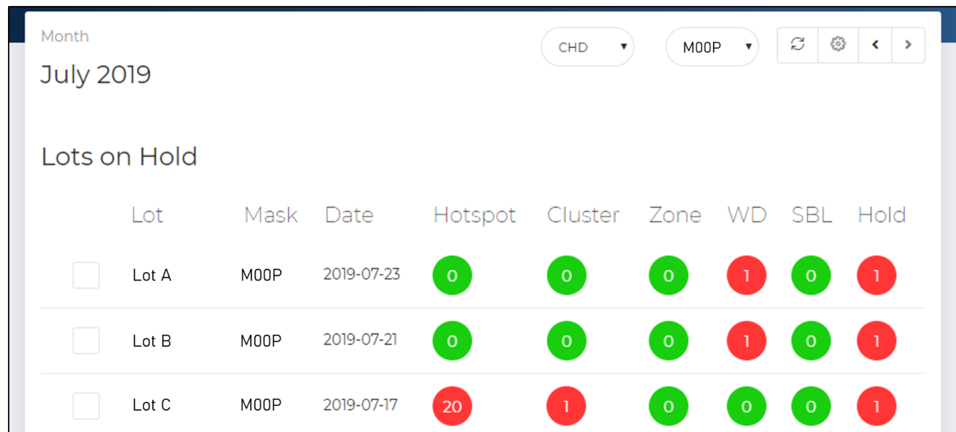

Figure 8.7: Main interface of the disposition system (sanitized)

The list is the central element in body of the interface. Figure 8.7 shows the interface with three lots in the list. This list supports some basic functionality such as sorting and filtering based on date or device.

An item in this list contains 9 elements, contained in 9 columns. The organization of these columns was based on the structure of a disposition report that the Device

Engineers developed themselves. The first three contain basic information about the lot on hold. The middle three are related to wafer failure patterns and contain results related to 3 different algorithms. The last three are automatically/manually generated reports containing information on why a lot was placed on hold. Additionally the first three are static elements just for displaying information, the last 6 elements are clickable.

Clicking on any of the last 6 elements will display additional information for use in lot disposition. If a clickable element is green it indicates that the information is "good to go" and the engineer should not need to inspect it. A red color indicates that there is something that the engineer does need to inspect. The color coding provides a quick overview of the information available for inspection. Additionally there is a number on each clickable element, this is there to indicate a quantity related to that element. For example, the number on the Hotspot element indicates how many hotspots were found for a lot.

The Lot, Mask and Date columns are essential information for the disposition methodology. They contain the lot id, the mask (design) id and the date the lot was placed on hold, respectively. This is the basic information needed to identify the lot on hold and prioritize it's disposition.

The Hotspot column is the first column with a clickable element and it is also the first element to display results from an algorithm. The algorithm is a simple one, designed help in z-axis analysis. It simply calculates the percent of failures in a particular x, y location for that lot. A location is considered a hotspot if the percent of failures rises above a chosen threshold. The hotspot element will be red if any hotspots are found and will also display the number of hotspots found.

Clicking on the element will display a stacked wafer plot, like the one shown in Figure 8.8 (a). It shows the grid of die locations on the stacked wafer plot, each location color coded based on percent of die failures at that location. White means little to no failures,

yellow means the percent of failures is approaching the threshold and red indicates that the percent of failures has exceeded the threshold.

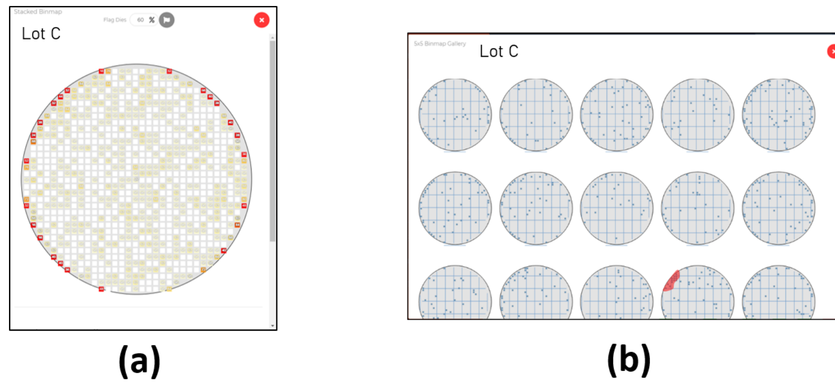

Figure 8.8: (a) Stacked wafer plot for the hotspot element. (b) 5x5 wafer plot for the cluster element.

The Cluster column contains an element that will display information from the same cluster pattern algorithm described in Section 7.2. A red color on this element will indicate that clusters were found and the number of clusters will be displayed on the element. Clicking on this element display a 5x5 wafer plot such as shown in Figure 8.8 (b) only three of the five rows are shown, the other two rows are visible upon scrolling. The detected clusters will be highlighted in red on the failure plots.

The Zone column is the last column that will display information related to wafer failure patterns. It is another simple algorithm, it basically divides the wafer in to many different zones and checks if one zone has a statistically higher number of failures. The zones are usually formed by dividing the wafer into quarters (top-left, top-right, bottom-left, bottom-right) or into rings (center, middle, outer). Clicking on this element will open another 5x5 plot like Figure 8.8 except the zones will be highlighted instead of clusters.

The WD column is the first column related to an automatically generated report from a proprietary system. This system monitors the Class Probe data and can automatically

flag a lot for disposition and generate a report. Clicking this element will display the report page seen in Figure 8.9 (a). The reports themselves are text files with difficult-to-read ASCII tables, so we summarize the report information in a table at the top of the page and provide links to the reports at the bottom of the page.

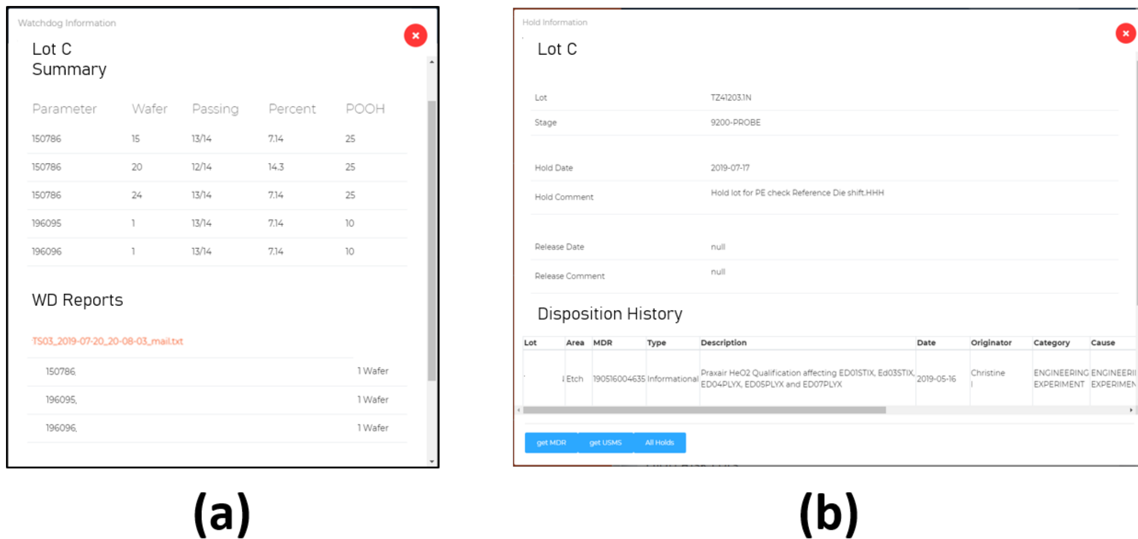

Figure 8.9: (a) Reports from the WD system. (b) Disposition history.

The SBL column contains information about the Statistical Bin Limit violations. At the time these screenshots were taken this element was not operational. We had made an attempt to incorporate this data into a visualization for the engineers to look at, but getting access to the pre-calculated limits proved to be a bureaucratic difficulty.

The last column has the element for displaying the disposition history. A red color and the number '1' indicates that the lot is flagged and being currently held for disposition. Once the lot has been released the color will be green and the number will be changed to '0'. Clicking on this element will bring up the reports page, shown in Figure 8.9 (b), where the entire disposition history is displayed.

8.4.1 Summary

By working closely with the Device Engineers on their disposition methodology, we were able to provide a software system that the team found beneficial. The major benefit provided is that all the information needed to prioritize the lots, and most of the information needed for the investigation is provided in one central location. Before, a Device Engineer needed to manually pull all those information from various places, and displayed or inspected the information separately. Note that we were not able to incorporate the SBL data or the defect data, primarily because the 'owners' of that data did not provide enough access and support to us.

Comparing the disposition methodology and the disposition system we created, a question must be asked: 'What is the ML doing?' The only area of this system that uses ML is in the cluster element. It indicates if a cluster has been detected or not. The function of the ML component is to act as a filter on the methodology (e.g. if there is no cluster do not look here). At best the ML component of this system indicates there is nothing to look at and the engineer should not investigate this data.

However clicking through the cluster element shows the 5x5 plot and that plot can be used to identify other wafer patterns. While watching a Device Engineer using this system, they clicked through the cluster element every time. If nothing was indicated for those elements, that did not mean that there was not something else that the engineer would find useful or interesting in that data. So, they clicked, they looked, they investigated.

Even if we could create a combination of recognizers and algorithms forming a "complete solution", such that if nothing was indicated, then there was nothing to see. This "complete solution" is not a significant benefit to the disposition methodology. Clicking through to see that there is nothing interesting for each of plot elements (Hotspot, Clus-

ter and Zone) takes about 10 seconds. The primary benefit was in bringing all the data they needed into a single place and displaying in a way that is useful to them and their methodology.

Recall that before our system, a Device Engineer would spend roughly 30 minutes to handle a lot deposition. In a half day, they might investigate about 2-6 lots. In retrospect, we realized that most of the 30 minutes were spent on grabbing, preparing, and displaying the relevant data for the investigation. After the plots were prepared, the visual inspection only took them a few seconds to make a decision.

In this process, the real time-consuming element is *data fusion* and *plotting*. The cost of visual inspection is minimal. Because of its minimal cost, even though we included a cluster pattern identification capability in the software, this capability was irrelevant to the engineers. They would still want to look at the wafer maps themselves, perhaps just to make sure.

From the perspective of providing a useful tool to engineers, the software system was a successful one. However, from the perspective of providing a useful ML solution to engineers, the system was an unsuccessful one. This is primarily because the real aspect of the problem was data fusion, not machine learning. In other words, the methodology involves heterogeneous data sources that come from different places to begin with. Data fusion is what they need. The methodology does not involve a large stream of homogeneous image data where visual inspection of those large number of images is the bottleneck. Because that aspect of the problem did not exist, the practical value of including a ML aspect was not very justifiable. For future development of ML, one should consider (1) if the current workflow has a data fusion problem, and (2) if it does, after the data fusion is resolved if ML can be used to help remove a bottleneck in the workflow.

Chapter 9

Data Summarization In DSML

"... when his work is done, his aim fulfilled, they will say: we did it ourselves."

— Lao Tzu

The lot disposition methodology described in the previous chapter represents a typical scenario of data analytics in a semiconductor company. From the surface, it might seem there are abundant data available. However, these data are heterogeneous in nature. They can come from different organizations, such as those associated with the process technology, with the business unit, or with the test technology. Each organization can have their own legacy data infrastructures with tools which have evolved over time. An engineer relies on information across a variety of data sources to make various decisions. These decisions are driven by various considerations, based on the operation knowledge of the company. It would be naive to think that just because there are abundant data, it will be feasible to build a software system to automate the decision making process. It will be even more naive to think that the reason that such automation does not exist today is largely due to the lack of utilization of modern ML technology.

In view of the lot disposition methodology, we see that the real issue is in data fusion. More importantly, what an engineer needs is a way to quickly summarize the relevant information from the data and present the information is an intuitive way for the engineer to consume. Once that summary information is there, making a decision based on the information is straightforward from the engineer's perspective. In other words, the most useful aspect for an engineer is *data summarization*, not decision making by a ML tool. This observation motivates us to think harder again: Is the essence of DSML really in ML or in data summarization?

9.1 Facilitating decision making

In Chapter 4, we argue that ML should be used to help users understand the data rather than to make a decision for them. The argument is based on the impossibility to make a reliable decision by a ML model in view of under-specification induced local NFL in a DSML process. On the other hand, the practical experience described in the previous chapter seems to further strengthen the point, that the primary goal of a practically useful software system should be to help its users understand the data, rather than making a decision for them. Equally important, they want to understand the data from their required perspective that is determined by their internal workflow or methodology. Such a "perspective" can be defined as what information (items, plots) they would like to see in a single summary window.

In that view, when developing a practical software system, one should start by defining those summary windows. Then, one should ask two questions: (1) how much work involved to solve the data fusion problem, and (2) whether a ML technique can be utilized to fulfill an aspect of the data summarization process. The answers to these two questions determine if the system can become a successful DSML software solution or

not. For example, if the answer to the second question is no, then a practical solution can still be built, but it will be a data fusion solution, not a DSML solution.

9.2 A possible DSML solution

Although in the lot disposition methodology, there is no critical role for a WMPR capability to play, that does not mean there is no demand for a DSML solution built upon such a capability. One application scenario that can be benefited by such a solution is related to the need by a fabless company to monitor and understand their yield in a high-volume production. In such a production, a large number of wafers are produced and tested every week. A wafer map can be plotted not just based on the overall pass/fail, but can also be based on certain groups of tests or individual tests, as well as a stack of wafers, etc. There are a large number of wafer maps to inspect. Essentially, there are two questions with the inspection: (1) Is there an important pattern to pay attention this week? (2) Is the pattern new or has it been seen before?

In this context, the input to the software is a stream of homogeneous data, i.e. wafer maps. However, even in this context, the solution should not try to answer the two questions directly. Instead, the solution should provide a summary to help its user to answer the questions. This is because the words in the questions such as "important", "new", and "seen before" are subjective to the user. Hence, the solution's job is to provide a summary window for the user to quickly see the relevant information, so that they can decide an answer for themselves.

9.3 A summary window design

In this section, we describe a summary window design that can be the front-end for a wafer map analytics system. Because information summarization is the focus and user perspective needs to be taken into account, the design needs to provide certain flexibility with respective to its summary window configuration.

A summary window might take the form shown in Figure 9.1, as a *configurable dashboard*. This type of display allows a user to add *cards* (green border) to a *work-space* (blue border). These cards behave similarly to windows in an operating system, they can be moved around the work-space, resized and closed. New cards can be added to the work-space by dragging them from a *card menu* (red border). Using a configurable dashboard allow the user to define what they want to see in their summary window. For example, Figure 9.1 shows the work-space with two cards: one card is used to display wafer maps and the other contains controls for the software system.

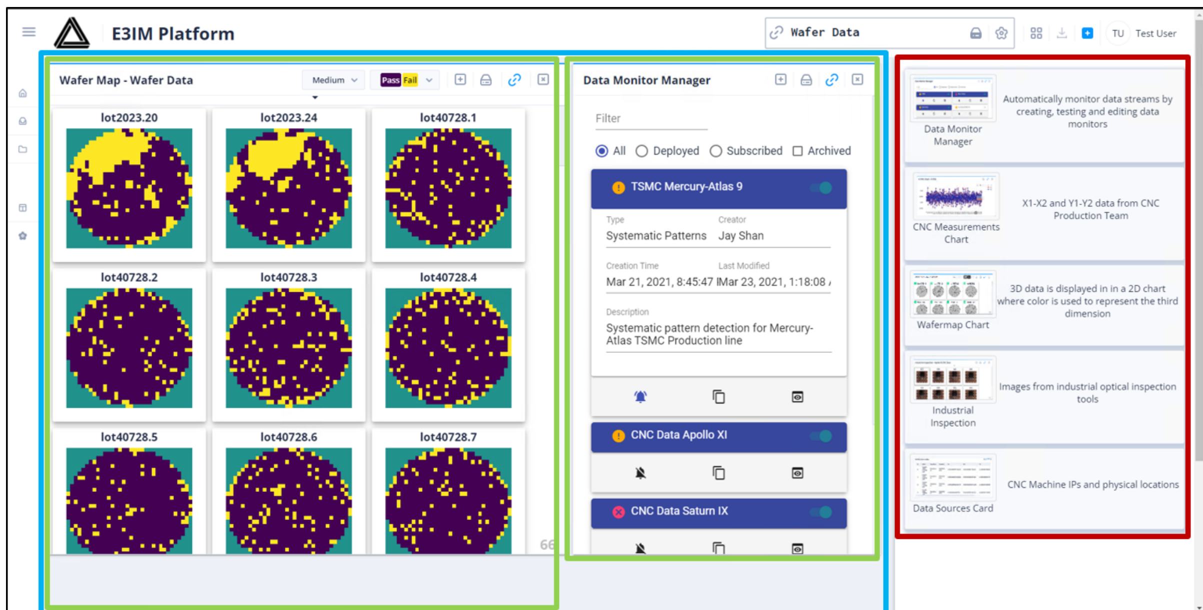

Figure 9.1: Summary window as a dashboard display. The dashboard work-space is shown with a blue border, the cards are shown with a green border and the card menu is shown with a red border.

The wafer map card serves the most basic function that a summary window can provide: it shows the data. One of the controls for the card is for data selection. Clicking on a hard-drive icon will open up the selection dialog shown in Figure 9.2. This allows the user to select data based on the time range, device (Product Id in the dialog) and lot. Whatever data is selected in the dialog will be downloaded and displayed in the wafer map card. It might seem counter-intuitive for a summary window to allow the user to look at unsummarized data. However, this functionality is crucial because an engineer may want to double check certain aspects of the data which are not provided in the other components of the summary window. In other words, the wafer map card is a fail-safe for the user.

Figure 9.2: Data selection for wafer map card. A user can select a time range, the device (product) id and the specific lots.

The wafer map card has controls along the top of the card that can alter the appear-

ance of the wafer maps. Using these controls the user can increase or decrease the size of the wafermaps or they can change the color scheme used in the wafer map. This may seem like mere convenience, but allowing a user to change the appearance of components in the dashboard is something that warrants serious consideration. Controls that change the appearance are an accessibility feature. A user with a disability such as poor vision or color blindness might not be able to understand the data as presented. For example, the wafer maps use color to convey information about which dies are passing and which dies are failing. If the color scheme uses a green color for passing dies and a red color for failing dies a user with red-green color blindness will have a hard time distinguishing passing dies from failing dies. Any time that color is used to convey information, it is necessary to consider how visual disabilities may affect the interpretability of that information. Adding the capacity to modify the appearance of the wafer maps can aid interpretability of information by allowing the user to select the size and color scheme that is easiest for them to understand.

The second card in Figure 9.1 is called the Data Monitor Manager and is used to interact with the ML component of the software system. With the Data Monitor Manager a user can create and deploy ML models called *monitors*. The purpose of a monitor is to aid the user in answering the first question: is there an interesting pattern this week?. As new test data becomes available the monitor examines wafer maps from this new data. If the monitor identifies a wafer map pattern it will generate a notification that the user can receive through a subscription. The types of patterns that the monitor is searching for are systematic patterns. These are related non-random patterns that are found multiple times within a one-week time-frame. We will only cover the user's interactions with the ML component, for details about how that ML component can be implemented please refer to the work in [60].

The Data Monitor Manager card allows multiple monitors to be created. This is to

facilitate workload division. A manufacturing process is producing a large volume of data from multiple different devices. A standard way of managing this large volume of data is to divide it up by the device. For example, an engineer can be assigned to an airbag controller device called AB1. The engineer will only need to look at wafer maps for the AB1 device and not the entire set of wafer maps from the manufacturing process. The Data Monitor Manager card facilitates this division by allowing a monitor to be created for each device. Users can subscribe to notifications from the appropriate monitors and only receive notifications for the devices they were assigned.

All the created monitors are displayed within the Data Monitor Manager card as monitor items. Three of the monitors items can be seen in Figure 9.1. Each monitor item in the Data Monitor Manager displays the monitor's name and along the bottom of the item there are controls for subscribing and unsubscribing to the notifications from the monitor. Additional information about the monitor can be seen by clicking on the name monitor in the top portion of the monitor item. The monitor will expand as shown by the first monitor item in Figure 9.1, displaying who created it, when it was created and a description of the monitor from the creator. This information is there to help the user subscribe to the appropriate monitor.

9.3.1 Notifications

Once a monitor is created it will begin searching new test data for systematic wafer map patterns and generating notifications. Notifications are provided to the user through an interface like the one in Figure 9.3. When a monitor generates a new notification for a user, that notification will be added to the notification list (red border). New notifications will have a blue indicator on the left side indicating to the user that they have not seen this notification yet. For example the first three notifications in Figure 9.3 have not been

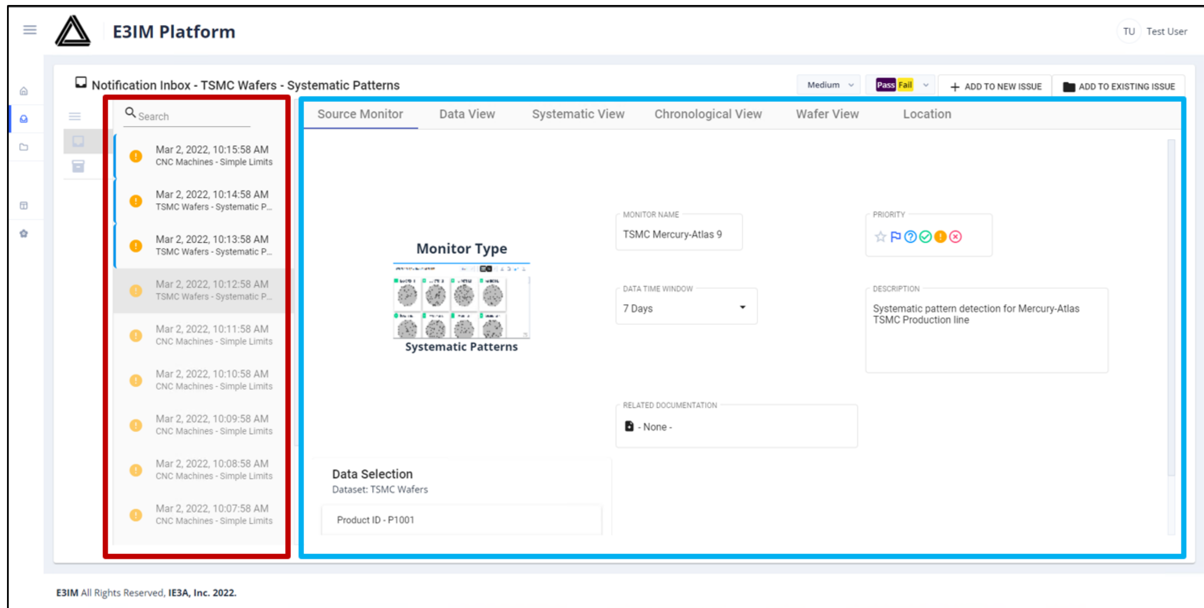

Figure 9.3: A Notification interface. The notification list shown with a red border contains all notifications. Clicking on a notification will display the notification report in the main body shown with a blue border.

seen yet by the user. Clicking on the notification will fill in the main body (blue border) with a notification report. Remember that within our view of DSML the ML is not there to make decisions. The notification report is there to convey information about wafer map patterns that the monitor discovered. The user can then decide if those wafer map patterns are important or not.

The notification report uses tabs to switch between different views provided in the report. The tabs can be seen across the top of the main body in Figure 9.3. To switch between views the user can click on that view's tab and that view will be displayed. Each of these views is there to convey some information about the wafer map pattern that the monitor discovered. The first view is the Source Monitor, it is information about the monitor that generated the notification. The Source Monitor view displays the monitors name and description as well as which device data that the monitor is examining. This is important context for the user, knowing the data is related to a particular device allows

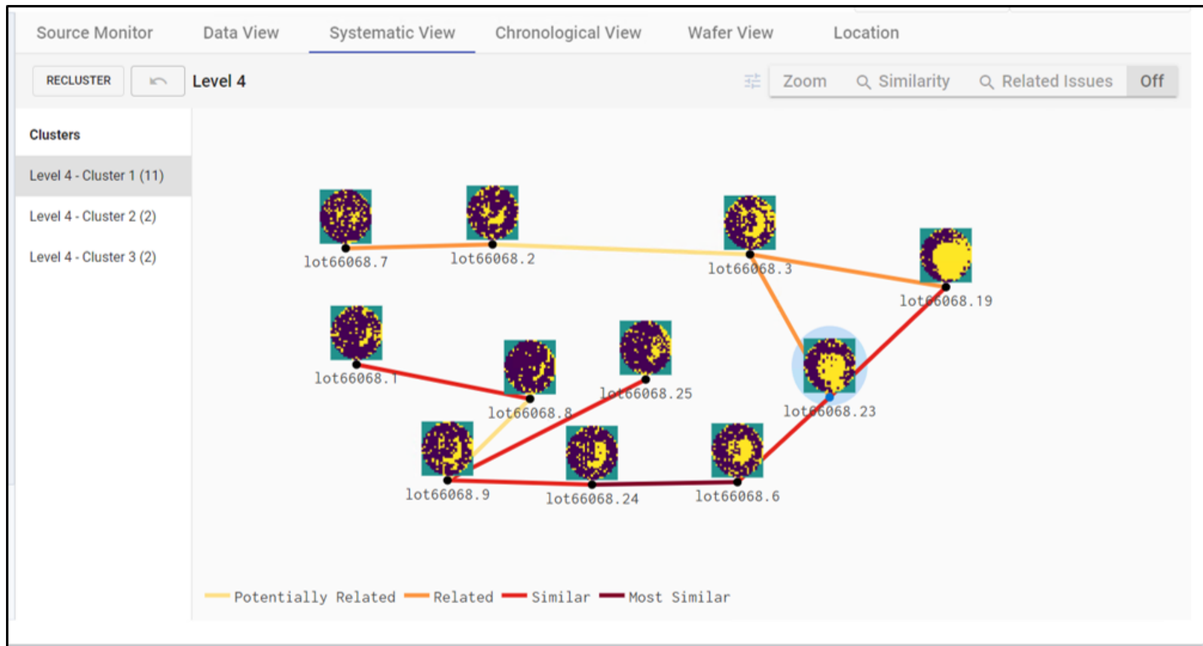

Figure 9.4: The Systematic View of the notification report. The graph shows the relatedness of the wafer map patterns. A user can click-and-drag the graph nodes so that the graph can be more easily understood.

the user to recall operation knowledge related to that device.

The second view we look at is the Systematic View, shown in Figure 9.4. Recall that the monitor is trying to identify related non-random patterns (i.e. systematic patterns). This tab has a graph that shows how the monitor describes the relatedness of the discovered wafer map patterns. Each node in the graph is a single wafer map pattern and the edges of the graph indicate relatedness by color. A dark red color indicates that the monitor would describe these two patterns as strongly related. A light yellow color indicates that the monitor would describe these patterns as potentially related. This graph view of the wafer map patterns allows the user to see the patterns that are occurring in the data and decide for themselves if any of the patterns are important and which patterns are related.

The Systematic View tab is a complicated component and we will not describe all of the features here. However, one feature warrants discussion because it relates to

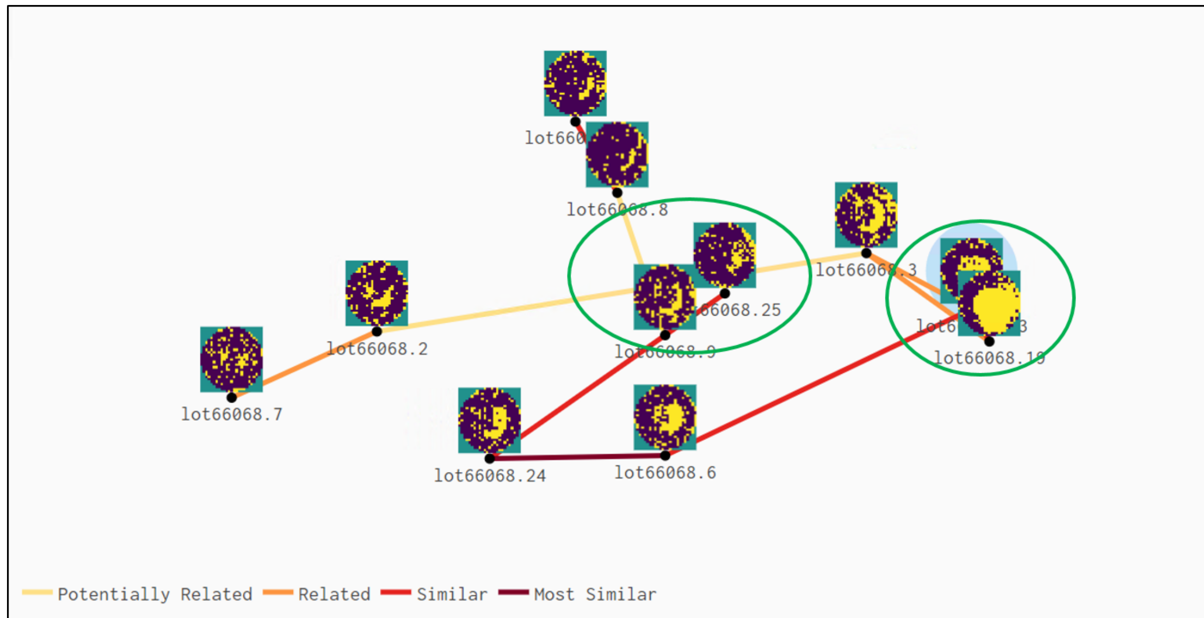

Figure 9.5: A graph layout algorithm can be used to produce graph layouts automatically, but they can be difficult to interpret. Compare this graph laid out by and algorithm to the same graph in Figure 9.4 laid out by a user. Green circles show two areas that are difficult to interpret.

the interpretability of the graph. Computing the graph layout for a graph like the one in Figure 9.4 is a challenging problem. Graph layout is the problem of calculating the positions of the nodes in a graph. While there do exist many algorithms for computing the graph layout, they produce difficult to interpret graphs. For example, compare the graph from Figure 9.4 to the same graph with a different layout in Figure 9.5. In particular look at the two circled areas in Figure 9.5 where the wafer maps obstruct edges making it difficult to determine the structure of the graph. The graph layout in Figure 9.5 was done with a graph layout algorithm and the layout in Figure 9.4 was laid out by a human. To insure that the graph was understandable to a user we added the ability for a user to click-and-drag nodes in the graph. The process is to initially layout the graph with an algorithm and then the user can move the nodes to make the graph understandable to them.

From within the Systematic View tab we can help the user to answer the second

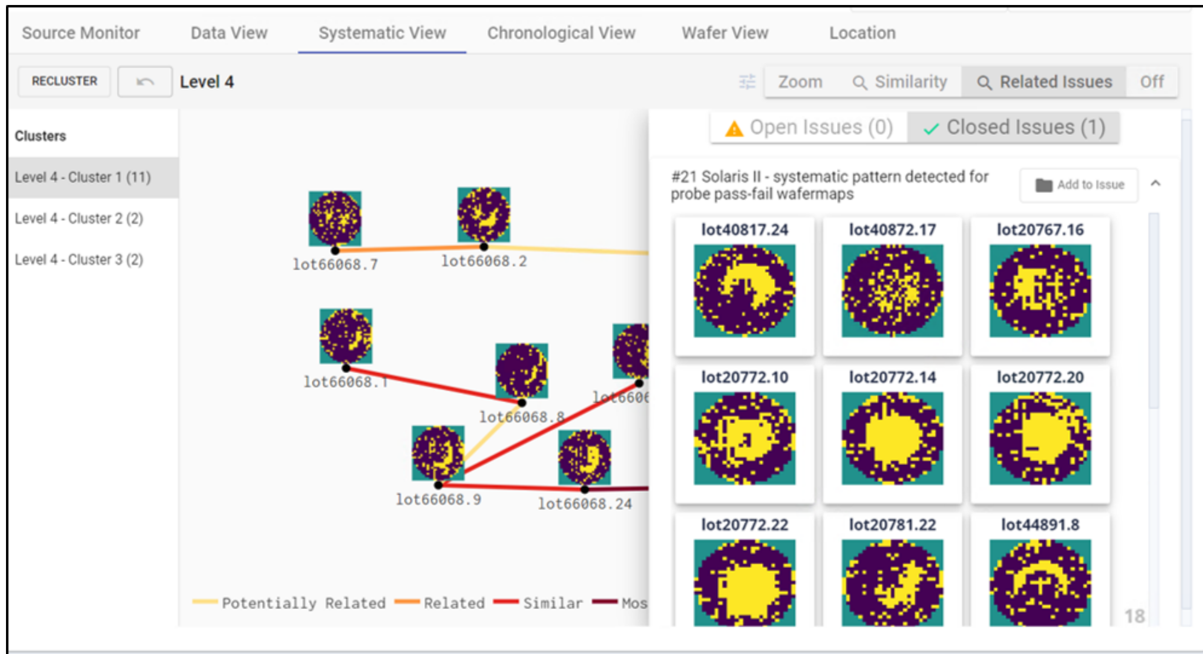

Figure 9.6: Searching for related wafer map patterns.

question: is this pattern new or has it been seen before? We can use ML to search for other related wafer map patterns. The ML model will be used to facilitate the search, but it is up to the user to decide if the the wafer maps really are related. This search only examines wafer maps that are contained within an issue. What an issue is will be discussed later in this chapter, it is enough to know that an issue contains wafer map patterns that someone has already decided are important. In this way the search will only return important wafer map patterns. The user can begin the search by clicking on the 'Related Issues' button in the top right of the Systematic View. When the search has finished a side-panel will open up and all the issues that contain related wafer map patterns will be displayed in it as shown in Figure 9.6. Within that side-panel the user can look at each issue and the wafer map patterns contained in that issue and decide the patterns in the issue really are related to the patterns in the graph.

The core of the notification report is in the Systematic View, it allows the user to

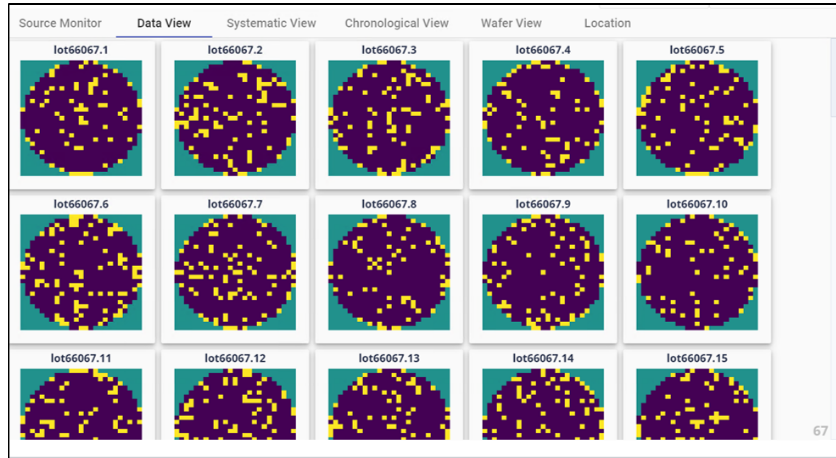

Figure 9.7: The Data View shows all the wafer maps that the monitor was examining for this notification.

answer the two questions from this application scenario. However, we can provide other views with different perspectives that can increase their understanding of the data. One example is the Data View shown in Figure 9.7. This view shows all the wafer maps that the monitor was examining when it generated this notification. This view allows the user to investigate for themselves if the monitor missed something. For example, the monitor may have correctly identified an important pattern, but missed a few wafer maps that were related to that pattern. This view allows a user to overcome any failings in the ML implementation.

9.3.2 Issues

As we have stated before issues contain important wafer map patterns. Within this system the user indicates that a wafer map pattern is important by adding it to an issue. The issue is also how the user indicates that two wafer map patterns are related. For example, if a user receives a notification and decides that there is an important wafer map pattern they can create a new issue for it marking that wafer map pattern as important. Then, in the future the user can receive a second notification with a wafer map pattern

that is related to the first pattern. That related wafer map pattern can be added to the same issue. The fact that these two wafers were added to the same issue indicates that they are related.

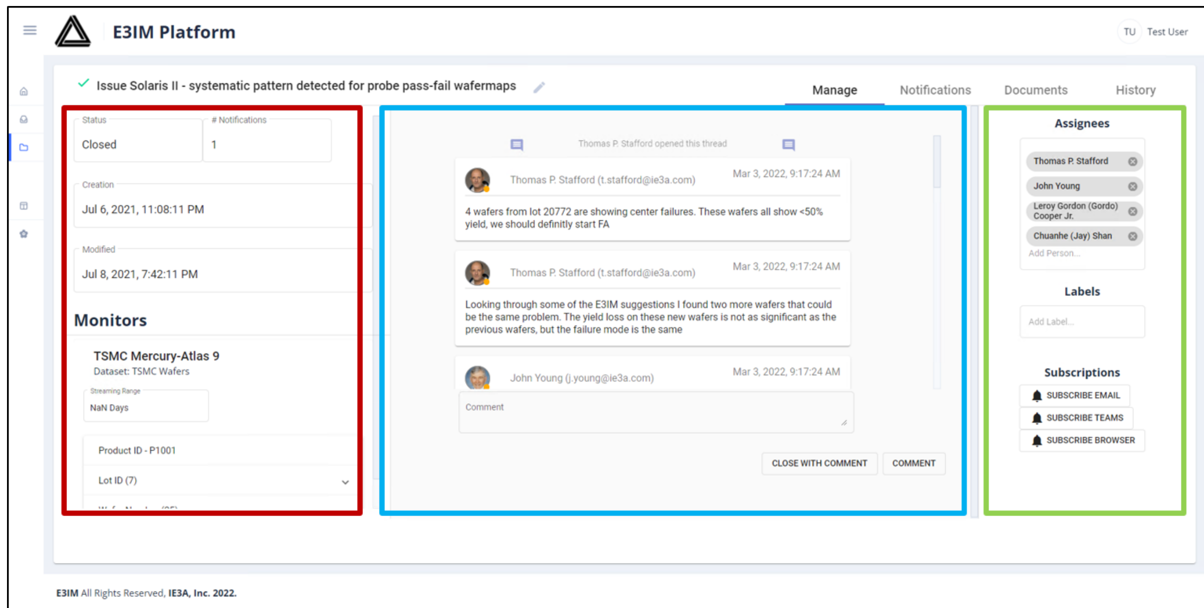

Figure 9.8: An issue manager interface. This interface allows a user to add or remove wafer map patterns and can provide a place for discussing the patterns contained in the issue.

The issues can be managed through an interface like the one shown in Figure 9.8. Within that interface a user can view or remove the wafer map patterns contained within the issue. Additionally, this issue manager interface can provide a place for discussion about the wafer map patterns. The patterns contained in an issue are related for some reason, users can use this discussion section to document what those reasons are. The discussion section can also document any analysis done on these wafer maps and conclusions from that analysis. This type of documentation can be of benefit if similar wafer map patterns are identified in the future. For example a future user could identify an important wafer map pattern and recognize that it is related to an existing issue. By looking at that issue and reading the discussion that user could potentially gain insight

into their current wafer map pattern.

The issue manager interface in Figure 9.8 shows the Manage view of the interface. Like the notification report the issue manager uses tabs to display different views. The tabs can be seen in the top right of the issue manager interface shown in Figure 9.8. The Manage view contains an issue information component (red border), the discussion component (blue border) and assigned users & subscriptions component (green box). The issue information component shows basic information about the issue. This information includes the creation time and last modified time. Additionally there is information on which monitors have been used to identify the wafer map patterns for this issue. The discussion component uses the familiar chat interaction where each user can type a message and have it displayed in the discussion. We can see 3 messages in Figure 9.8 and more can be seen through scrolling down. Below the 3 messages is the input box where a user can type a message. Pressing the 'Enter' key or clicking on the 'Comment' button will add the message to the discussion where anyone can read it. Lastly, the assigned users and subscriptions component is used to document which users are assigned (i.e. responsible) for this issue. Each of the assigned users will receive an email whenever the issue is changed. For example if a new wafer map pattern was added or if a new message is added to the discussion an email will be generated and sent to each assigned user.

The notification view of the issue manager interface is shown in Figure 9.7. It is here that a user can look at all the wafer map patterns for this issue. This is also the location where a user can remove wafer map patterns from the issue. If a user decides that a wafer map pattern no longer belongs in this issue or if a pattern was added mistakenly, then it is in the notification view that the user could remove those patterns. To remove wafer map patterns a user can click to select each the wafer map pattern that should be removed. Once all the patterns are selected, the user can then right-click to display a context-menu. Clicking the 'Remove' menu item will remove the selected wafer map

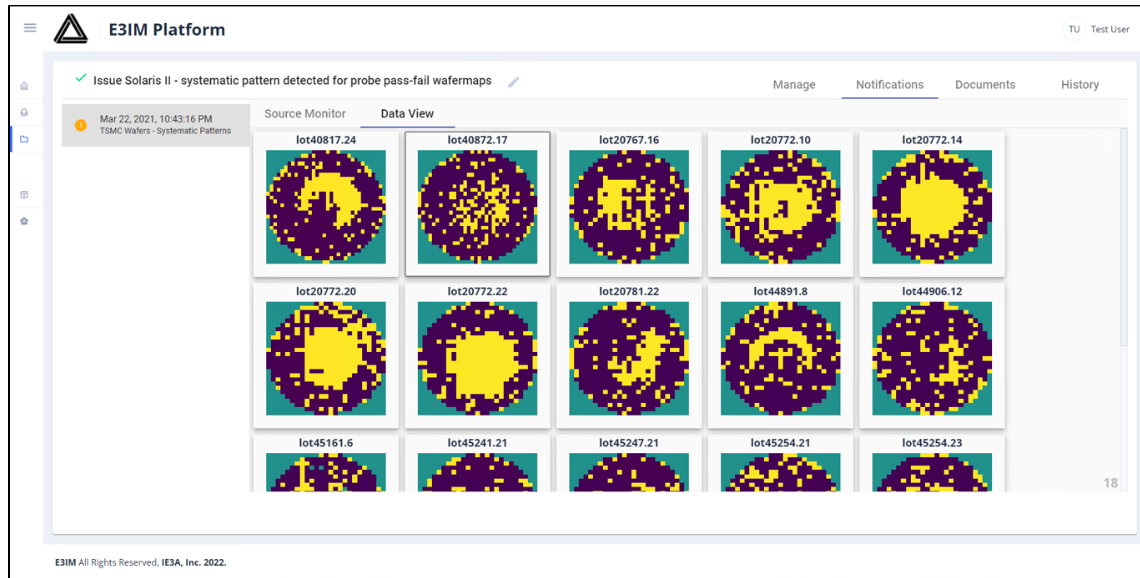

Figure 9.9: The issue manager notification view. All wafer map patterns for the issue can be seen and

patterns from the issue.

9.4 Summary

Our experience in implementing a software system to help engineers with their methodology has led us to conclude that the data summarization is the most useful aspect for an engineer. This chapter uses a WMPR application scenario to demonstrate how a summarization front-end component could be practically implemented and how the user could interact with the software.

We implemented a system where a ML model called a ‘monitor’ that could be utilized to help a user identify important wafer map patterns. We did not describe the details of the ML technology behind such a ML model. Those details are documented in [60]. Instead we focused on the user interactions. This required that we implemented several interfaces for the user. One interface was needed to manage the monitors, allowing the user to create a new monitor and select which data the monitor was examining.

Another interface, the notification interface, was needed for the user to understand what the monitor had uncovered and decide if there were any important wafer map patterns. Finally a third interface was needed so that the user could document what wafer map patterns were found and record any analysis and conclusions about those wafer map patterns.

We also experienced some unexpected challenges in providing interpretable visuals. We had to consider the circumstances of a visually impaired user. A color blind user may not be able to understand the data presented to them if the color scheme is a poor match for that user's disability. We also saw that presenting the relatedness of wafer map patterns as a graph required that we add in interactivity. The standard graph layout algorithms would sometimes produce difficult to interpret graphs. We had to allow the user to modify the graph's layout with a click-and-drag action so that the graph could be understood.

All these interface and the unexpected challenges demonstrate that implementing a data summary system is not a trivial task. This work represents a significant amount of development effort in addition to the efforts spent on implementing the data fusion component and the ML component in the back-end. Our experience shows that any DSML project that hopes to deploy a practical solution should factor in the amount of effort required to implement such a front-end component. After all, DSML is about data summarization and user assistance. Therefore, a user-friendly and flexible front-end design will be critical for its wide adoption.

Chapter 10

Conclusion

"The Buddha said, 'As to Anuttara-samyak-sambodhi, there is not even the slightest dharma which I could attain, therefore it is called Anuttara-samyak-sambodhi.' "

— Chapter 22, The Vajra Prajna Paramita Sutra

This thesis started with a review of the successes of machine learning (ML) such as AlphaGo and GPT-3, followed by a review of the challenges of using ML techniques for certain domain specific ML (DSML) problems. In particular we look at how these DSML problems are different from the typical problems tackled by the ML mainstream.

We examined three DSML problems from the semiconductor domain: functional coverage improvement, speed-limiting path analysis and layout hot-spot detection. From these problems we observed that the DSML application contexts can be viewed as an iterative search process. In each iteration the goal is to augment the existing data with either generating a new sample or including a new feature. We also observed that there always exists a non-machine learning approach to solving these problems. From these two observations we formulated our two DSML postulates.

This goal of either generating a new sample or including a new feature, presents difficulties. To generate a new sample there are data generation algorithms. However, these algorithms require samples of the class to train the generator and the examined DSML problems showed that new samples can be from a new class. Furthermore within DSML problems the benefit of a new sample is correlated to how "surprising" that sample is. On top of the difficulty in generating new samples, including a new feature requires that we first show the current set of features is not sufficient. Meaning we would have to prove that the it is not possible to learn based on the given dataset i.e. "No ML".

In order to say there is no ML we looked to understand what ML was through the lens of four schools of thinking. The four schools were Computational Learning Theory (CLT) with it's probably approximately correct (PAC) model, Statistical Learning Theory (SLT) with it's concepts of VC Dimension & VC Entropy, Statistical Mechanics of Learning (SML) which takes a physics view on learning to identify self-averaging quantities and Bayesian Learning (BL) which describes a learning process using evidence and prior knowledge. The first three schools try to define when learning takes place and the weakness of these schools is revealed by the No Free Lunch (NFL) theorems that describe the conditions under which no ML algorithm is better than random guessing. We consider Bayesian learning to be the most realistic, in that it tells us what can be done in each step of the learning process and how our assumptions affect the outcome.

The iterative DSML process expands the dataset with new "surprise" samples. The process continues until a convergence point is reached, for example when no additional surprise samples can be discovered over a period of time. At each iteration of the process the problem is under-specified. Instead of providing the best fitting model for a given problem we can help the user to understand that the problem is indeed under-specified. We illustrate this under-specified nature of DSML with examples from Outlier Analysis, where a score and a threshold are used to classify samples into two classes: inliers and

outliers. In these examples an outlier model was constructed with an initial dataset, however the model was shown to be wrong when future surprise samples became available.

Following a traditional ML paradigm the under-specification problem seems unsolvable. However the traditional ML paradigm is trying to solve a problem completely and convey certainty in the solution. It is this quest for certainty that is the problem, because we cannot attain the certainty we desire with an under-specified dataset. That is we cannot completely trust the decisions of a ML model. If this quest for certainty is the source of our problem then it is logical that we abandon it. In other words, we should consider not using a ML model to make decisions. Practically our view of ML can change. The final decision is left to the user to make, and we apply machine learning techniques, if necessary, to help the user make as informative a decision as possible.

With this new view of ML we pursued methods in which ML could provide help to the user. In Chapter 5 we focused on developing a method that can perform an applicability check for a given outlier analysis method so a user can know if the assumptions of an outlier model are justifiable given the data the model is used with. In Chapter 6 we show that a "What-if" analysis method can be used to evaluate how an outlier threshold decision could be affected by the underlying sample set. Finally, we looked at the Wafer Map Pattern Recognition (WMPR) problem in Chapter 7. There we did not approach the WMPR problem as others had done before, solving it as a multi-class classification problem. Instead we suggested a method where a sequence of pattern recognizers were developed iteratively. This iterative approach being more consistent in view of a DSML setup where there is a limited set of data samples to use.

Lastly we looked at a DSML from a practical perspective. Describing our experience in trying to implement a ML software solution in production as well as to provide some added value to the company. The software solution was not a successful one in view of providing a ML solution. However, we learned important lessons about DSML from

a practical perspective. We learned that the real benefit of a software system is in data fusion and plotting. More importantly what an engineer needs is a way to quickly summarize the relevant information from the data. Using this experience we then tried to develop a software system in which a user can create a summary window with the help of ML models. This software was not trivial to create and would require a significant portion of the developmental effort needed to deploy DSML.

If ML is about the algorithm for building the best model from the data then our thesis' focus is on studying other aspects. These aspects are: assumptions of the algorithms, characteristics of the data and the practical need & constraints which are critical to enable deployment and wide adoption of DSML application software. From this perspective, we can say that DSML is less about ML and more about the complementary aspects of ML (Co-ML). In short, **DSML is Co-ML rather than ML**.

A large part of the thesis focuses on outlier analysis because the setup of outlier analysis has all the ingredients mimicking a DSML problem setup except an outlier analysis is much simpler than a typical ML algorithm. As pointed out in Chapter 4 even in ML what is calculated is a score and a threshold decision. The decision can be multi-dimensional and the model extremely complicated, but that is not our focus — we focus on everything outside the algorithm. The findings in this thesis can have an impact in the general ML context, because replacing the more complex algorithm with a simple one doesn't affect the Co-ML aspects studied in this work.

10.1 A Philosophical Remark About This Work

The success of Machine Learning (ML) is largely driven by big data. In a DSML context, one cannot rely on big data to attain a success. The fundamental problem faced in DSML is local NFL, a concept emphasized in our work. Big data provides a world

where optimization of a ML algorithm can lead to positive effect. Consequently, much of the ML has been devoted to this optimization. In contrast, DSML lives in a world of local NFL. The essence of the problem is not optimization but understanding.

In a world of NFL, there is no perfect model or perfect answer. There is only answer that a domain user feels helpful or not. Because user's perspective is so important in a DSML context, the goal of learning is not to make a decision for them, but to provide a summarization such that they can achieve an understanding of the data to feel comfortable about making a decision.

When providing a software solution aiming to achieve that summarization goal, it is not necessary that a ML technique must be used. In some application contexts, a user simply desires a software solution that can alleviate their burden from the tedious manual process of data fusion. In such a context, the data fusion part and the summarization interface design can both be very valuable to a user, but insisting on including a ML component in the software solution might not be necessary.

The desire to optimize perceives a world where more optimization leads us closer to a "golden" answer. The assumption is that there is a "real solution" in the world. In a world of NFL, there is no such "real solution" or "golden" answer. All we have is a solution or an answer that seems to be a solution or answer within a scope constrained by our limited view. In such a world, the essence of DSML is to help others reach a solution or answer deemed by their individual view. DSML only tries to alleviate the pain incurred in their search process, and cannot make a final judgement for them. In this regard, perhaps the ultimate goal of DSML is not to tell people how to solve their problem, but to bring comfort and a peace of mind to them. And sometime in doing so, if ML is not required, we need to give up the ML aspect even though we start with the desire to pursue ML. From this angle, we can say that in the world of DSML, while the essence of the problem is understanding, the essence of the practice is compassion.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems (NIPS)*, vol. 25, 01 2012.

[2] S. Siatkowski, C. J. Shan, L.-C. Wang, N. Sumikawat, W. R. Daasch, and J. M. Carulli, "Consistency in wafer based outlier screening," *IEEE VLSI Test Symposium (VTS)*, pp. 1–6, 2016.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, 09 2014.

[4] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[7] O. Guzey and L.-C. Wang, "Functional test content optimization for peak-power validation - an experimental study," *IEEE International Workshop on High Level Design Validation and Test (HLDVT)*, pp. 151–158, 2007.

[8] V. Kamath, W. Chen, N. Sumikawa, and L.-C. Wang, “Functional test content optimization for peak-power validation - an experimental study,” *IEEE International Test Conference (ITC)*, 2012.

[9] W. Chen, L.-C. Wang, and J. Bhadra, “Simulation knowledge extraction and reuse in constrained random processor verification,” *ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2013.

[10] K.-K. H. W. Chen, L.-C. Wang, and J. Bhadra, “Learning to produce direct tests for security verification using constrained process discovery,” *ACM/IEEE Design Automation Conference (DAC)*, 2017.

[11] J. Chen, B. Bolin, L.-C. Wang, J. Zeng, D. G. Drmanac, and M. Mateja, “Mining ac delay measurements for understanding speed-limiting paths,” *IEEE International Test Conference (ITC)*, 2010.

[12] N. Callegari, D. G. Drmanac, L.-C. Wang, and M. Abadir, “Classification rule learning using subgroup discovery of cross-domain attributes responsible for design-silicon mismatch,” *ACM/IEEE Design Automation Conference (DAC)*, pp. 374–379, 2010.

[13] P. Bastani, N. Callegari, L.-C. Wang, and M. Abadir, “Feature-ranking methodology to diagnose design-silicon timing mismatch,” *IEEE Design & Test*, pp. 42–53, 2010.

[14] N. Callegari, P. Bastani, L.-C. Wang, and M. Abadir, “A statistical diagnosis approach analyzing design-silicon timing mismatch,” *IEEE Design & Test*, vol. 28, no. 11, pp. 1728–1741, 2009.

[15] D. G. Drmanac, F. Liu, and L.-C. Wang, “Predicting variability in nanoscale lithography processes,” *ACM/IEEE Design Automation Conference (DAC)*, pp. 545–550, 2009.

[16] D. Wolpert, “The lack of a priori distinctions between learning algorithms,” *Neural Computation*, vol. 8, 03 1996.

[17] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997.

[18] D. Wolpert, “The relationship between pac, the statistical physics framework, the bayesian framework, and the vc framework,” in *The Mathematics of Generalization*, 11 1995.

[19] M. J. Kearns and U. Vazirani, *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

[20] V. Vapnik, *The Nature of Statistical Learning*. Springer, 2000.

[21] A. Engel and C. Van den Broeck, *Statistical Mechanics of Learning*. Cambridge University Press, 2001.

[22] D. McKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[23] L. G. Valiant, "A theory of the learnable," *Communications of ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[24] D. Whitley, "Sharpened and focused no free lunch and complexity theory," in *Search Methodologies*, pp. 451–476, 07 2014.

[25] C. Schumacher, M. D. Vose, and L. D. Whitley, "The no free lunch and problem description length," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 565–570, Morgan Kaufmann, 2001.

[26] L.-C. Wang, "An autonomous system view to apply machine learning," *IEEE International Test Conference (ITC)*, pp. 1–10, 10 2018.

[27] L.-C. Wang, "Experience of data analytics in eda and test - principles, promises, and challenges," *Keynote, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. PP, pp. 1–1, 10 2016.

[28] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[29] N. Sumikawa, J. Tikkanen, L. C. Wang, L. Winemberg, and M. Abadir, "Screening customer returns with multivariate test analysis," *IEEE International Test Conference (ITC)*, pp. 1–10, 11 2012.

[30] N. Sumikawa, L. C. Wang, and M. Abadir, "An experiment of burn-in time reduction based on parametric test analysis," *IEEE International Test Conference (ITC)*, pp. 1–10, 11 2012.

[31] M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," *IEEE Transactions on Semiconductor Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015.

[32] Automotive Electronics Council, *Guidelines for Part Average Testing*, 2011. AEC-Q001 Rev-D.

[33] J. Tikkanen, N. Sumikawa, L.-C. Wang, and M. Abadir, "Multivariate outlier modeling for capturing customer returns — how simple it can be," *Proceedings of the 2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pp. 164–169, 07 2014.

[34] L.-C. Wang, S. Siatkowski, C. Shan, M. Nero, N. Sumikawa, and L. Winemberg, "Some considerations on choosing an outlier method for automotive product lines," *IEEE International Test Conference (ITC)*, pp. 1–10, 10 2017.

[35] S. Siatkowski, C.-L. Chang, L.-C. Wang, N. Sumikawa, L. Winemberg, and R. Daasch, "Generalization of an outlier model into a "global" perspective," *IEEE International Test Conference (ITC)*, pp. 1–10, 10 2015.

[36] R. Daasch, J. Mcnames, D. Bockelman, and K. Cota, "Variance reduction using wafer patterns in iddq data," *IEEE International Test Conference (ITC)*, pp. 189 – 198, 02 2000.

[37] R. Daasch, K. Cota, and J. Mcnames, "Neighbor selection for variance reduction in iddq and other parametric data," *IEEE International Test Conference (ITC)*, pp. 92 – 100, 02 2001.

[38] K. Butler, S. Subramaniam, A. Nahar, J. Carulli, T. Anderson, and R. Daasch, "Successful development and implementation of statistical outlier techniques on 90nm and 65nm process driver devices," *IEEE International Reliability Physics Symposium Proceedings*, pp. 552 – 559, 04 2006.

[39] F. Grubbs, "Procedure for detecting outlying observations in samples," *Technometrics*, vol. 11, p. 53, 04 1974.

[40] B. Silverman, *Density Estimation for Statistics and Data Analysis.* 02 2018.

[41] J. Beirlant and L. Devroye, "On the impossibility of estimating densities in the extreme tail," *Statistics & Probability Letters*, vol. 43, 06 2003.

[42] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, 01 2012.

[44] M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," *IEEE Transactions on Semiconductor Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015.

[45] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* The MIT Press, 2001.

[46] M. Fan, Q. Wang, and B. van der Waal, "Wafer defect patterns recognition based on optics and multi-label classification," *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016.

[47] J. Yu and X. Lu, "Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis," *IEEE Transactions on Semiconductor Manufacturing*, vol. 29, no. 1, pp. 33–43, 2016.

[48] M. Piao, C. H. Jin, J. Y. Lee, and J.-Y. Byun, "Decision tree ensemble-based wafer map failure pattern recognition based on radon transform-based features," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 2, pp. 250–257, 2018.

[49] J. Yu, "Enhanced stacked denoising autoencoder-based feature learning for recognition of wafer map defects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 613–624, 2019.

[50] N. Yu, Q. Xu, and H. Wang, "Wafer defect pattern recognition and analysis based on convolutional neural network," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 566–573, 2019.

[51] J. Wang, Z. Yang, J. Zhang, Q. Zhang, and W.-T. K. Chien, "Adabalgan: An improved generative adversarial network with imbalanced learning for wafer defective pattern recognition," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 3, pp. 310–319, 2019.

[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Advances in Neural Information Processing Systems*, vol. 3, 06 2014.

[53] T.-H. Tsai and Y.-C. Lee, "A light-weight neural network for wafer map classification based on data augmentation," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 4, pp. 663–672, 2020.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[55] M. Saqlain, Q. Abbas, and J. Y. Lee, "A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 3, pp. 436–444, 2020.

[56] M. B. Alawieh, D. Boning, and D. Z. Pan, "Wafer map defect patterns classification using deep selective learning," *ACM/IEEE Design Automation Conference*, 2020.

[57] C. Shan, A. Wahba, L.-C. Wang, and N. Sumikawa, "Deploying a machine learning solution as a surrogate," in *IEEE International Test Conferencel*, pp. 1–10, IEEE, 2019.

[58] Y. J. Zeng, L.-C. Wang, C. J. Shan, and N. Sumikawa, "Learning a wafer feature with one training sample," in *IEEE International Test Conferencel*, pp. 1–10, IEEE, 2020.

[59] Y. J. Zeng, L.-C. Wang, and C. J. Shan, "Miniature interactive offset networks (minions) for wafer map classification," in *IEEE International Test Conferencel*, pp. 1–10, IEEE, 2021.

[60] C. J. Shan, *Domain-Specific Machine Learning - A Human Learning Perspective.* PhD thesis, University of California Santa Barbara, 2022.

[61] M. Rosenblatt, R. Davis, K.-S. Lii, and D. Politis, "Remarks on some nonparametric estimates of a density function," in *Selected Works of Murray Rosenblatt*, pp. 95–100, 03 2011.

[62] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *International Conference on Learning Representations (ICLR)*, 2016.