Data Learning Based Diagnosis*

Li-C. Wang
University of California, Santa Barbara

Abstract— Traditional diagnosis of defects is based on an assumed fault model. A failing chip is diagnosed to find the subset of faults that can best explain the failure. This paper illustrates a link between this traditional perspective of diagnosis and a new perspective where diagnosis is seen as a form of data learning. We explain that both defect diagnosis and data learning are solving so-called ill-posed problems and the technique for solving such a problem is called regularization. We illustrate a diagnosis framework that employs various data learning techniques to implement two diagnosis approaches: feature ranking and rule extraction. This diagnosis framework is designed to uncover design-related issues that cause systematic uncertainties or any unexpected behavior in silicon. We review the work that has been accomplished for implementing this framework and further discuss issues with its practical application.

I. Design for Reality

As manufacturing technologies continue to scale, diverse trends can be observed in today's IC design industry: (1) For a high-volume high-performance design company, the growing concern has been on the issue of low manufacturing yield. Such a company collects millions of failing parts each quarter and urgently needs a framework that can efficiently analyze vast amount of silicon test data. (2) For a high-performance microprocessor company, there has been increasing burden on the post-silicon debug and diagnosis. Such a company traditionally relies on multiple *silicon steppings* [1] for performance yield optimization after the first tape-out. They demand a new framework that can complement traditional debug and diagnosis and improve the effectiveness of each silicon stepping. (3) For a typical ASIC design house where tens of designs are produced each year, the issue of low yield is reflected in the increase of design margins. While the advancement into the next technology node can bring benefits on power and die size, managing design margins to achieve high enough yield has become very challenging. The framework required is for avoiding over-design, which in turn translates into better chip performance, lower design cost, and earlier time-to-market.

Low yield can be seen as a result of the increased variability (systematic variations) and uncertainty (random variations) from manufacturing process and design-related sources [2, 3, 4, 5, 6, 7]. Variability and uncertainties degrade predictive effectiveness of models and simulation tools on actual silicon behavior. The three trends above, although presented differently, can all be seen as results of reduced predictability. This motivates the development of new tools and methodologies that can help achieve *design for realty*.

An intuitive approach to achieve design for realty is to improve predictability of key models and/or simulation design tools used in today's design flows. Figure 1 illustrates the challenge of adopting this approach in practice. When low yield is observed, there can be an enormous number of potential reasons from different sources, or combinations of sources at different stages of a design process [8]. For a particular design, the most relevant reasons can be quite diverse and highly depend on the design and/or design methodology. Usually, a *point solution* is developed by improving a particular model (for example, cell model [9]) or a particular design tool (for example, SSTA [10, 11]). However, without knowing what the most important contributors to low yield are, it is difficult for a company to prioritize their design resources and adopt the most effective point solutions for yield optimization.

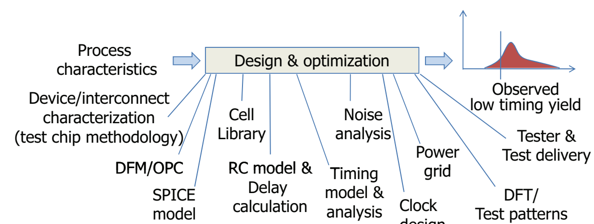

Fig. 1. **Too many factors potentially affect yield**

To prioritize design resources, one desires to know the few most relevant reasons contributing low yield. This information typically comes from the post-silicon debug/diagnosis/yield learning stage. Traditional debug/diagnosis techniques, however, are optimized for analyzing manufacturing defects on individual failing parts [12, 13]. The results are mostly fed back to the process. When applying these techniques to find design-related issues, their effectiveness to analyze vast amount of failing data and to search in an enormous hypothesis space consisting of factors from multiple design stages can be limited.

Given a large database (design and test data), searching for hypotheses in an enormous space to best explain the data, is commonly formulated as a data learning problem. This motivates us to approach the diagnosis problem from a data learning perspective.

*This work is supported in part by SRC 2007-TJ-1585 and by National Science Foundation NSF project CCF-0915259

978-1-4244-5767-0/10/$26.00 2010 IEEE

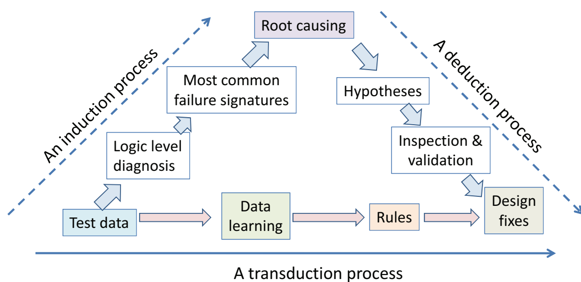

Fig. 2. **Data learning to achieve transduction diagnosis**

Figure 2 illustrates a comparison between the traditional diagnosis approach and the proposed data learning based diagnosis. A traditional approach consists of two main steps in the diagnosis process: induction and deduction. From a mass amount of test data, on every failing chip logic diagnosis is first applied to narrow down to the circuit elements (gates and/or nets) that may potentially cause the failure. This information is aggregated to identify the most common failure signatures. For example, a particular net is the suspect for failing a large number of chips. The common signatures and selected representative failing chips are sent for root cause analysis, such as physical debug to identify the root causes. Once these root causes (hypotheses) are found, they are sent for manual inspection and/or simulation for validation. Once they are confirmed, design fixes are generated and applied.

From test data to root cause analysis is an induction process where the inference is frequency based, i.e. if the same thing occurs many times, it is more important and likely to be the truth. From root causes to design fixes is a deduction process where design fixes are deduced from the premises embodied in the hypotheses. The bottleneck in such a diagnosis approach is getting to the root causes. It is well known that root causing can be tedious and expensive. As a result, only the most important (and most frequent) information in the test data would eventually translate into design fixes. Other information is lost in the diagnosis process.

If our ultimate goal is to find fixes, it may be desirable to find fixes directly from the data without going through the root causing step (and hence avoiding the most expensive step in the induction-deduction diagnosis paradigm). We call such an approach *transduction diagnosis* [14] (The word ”transduction” was used in [14] to characterize the modern non-parametric learning paradigm similar to the concept illustrated in Figure 2). The idea of transduction diagnosis is ”direct inference” by learning these fixes directly from the data without knowing the physical reasons behind the fixes. In other words, such fixes are entirely ”data driven.” By skipping the root causing step, transduction diagnosis can be more effectively applied in practice and demands less resources.

The rest of the paper is divided into four sections. Section II draws the link between traditional logic diagnosis and data learning through the notion of the *ill-posed problem* which is closely related to the concept of diagnosis resolution well studied in traditional logic diagnosis. Section III explains how data learning can be employed to implement a diagnosis framework consisting of two diagnosis approaches: *feature ranking* and *rule extraction* that can be achieved with various learning techniques. Section IV discusses our experience with various learning algorithms in applying the diagnosis framework. Section V describes a few scenarios where the proposed diagnosis framework has been applied and discusses practical considerations encountered. Finally, section VI concludes.

II. Diagnosis and Learning

In diagnosis, we are given with two set of data, one dataset P describing predicted behavior and the other dataset M describing measured behavior where $P \neq M$. Typically, the dataset P is obtained by simulating a pattern set T. The dataset M is obtained by applying T on the chip(s) to be diagnosed. The job is to find the best explanation for why $P \neq M$. In logic diagnosis, P and M can each be seen as an $n \times m$-bit vector of 0/1 values, where n denote the number of patterns and m denote the number of scan flip-flops.

To explain the mismatch, a traditional approach starts with a fault model F consisting of n possible faults $\{f_1, \ldots, f_n\}$. Let $\mathcal{F} = 2^F - \phi$ be the set of all subsets of F, excluding the empty set ϕ. Each subset $f \in \mathcal{F}$ is a possible explanation for the outcome. Hence, there can be in total $2^n - 1$ different explanations.

Let A denote the simulation of the design. We use $A \circ f$ to denote the resulting dataset (vector) of fault simulation based on the subset f. Hence, $A \circ \phi = P$, denotes the result of good design simulation. Then, we can write the diagnosis problem as finding the explanation f to ”best” solve the following equation (or equivalently, we can express it as $A \circ f = P - M$ if we view A as the simulation to compute the difference between the good design and the faulty design):

$$A \circ f = M, \text{for} f \in \mathcal{F} \tag{1}$$

To determine which explanation is the best, we need a way to evaluate the *fitness* of an explanation. We can define a loss (or error) function. For example, a loss function can be written as the following:

$$R(f) = \| A \circ f - M \|^2 \tag{2}$$

In logic diagnosis, $A \circ f$ and M are bit vectors. $R(f)$ therefore measures the square distance between the explained vector $A \circ f$ and the measured vector M. When each bit has only two possible values, 0 and 1, $R(f)$ essentially counts the number of bits that cannot be explained by f, i.e. with fault f, cannot explain the difference between P and M. With such a loss function, the best f can be the one that minimizes $R(f)$.

A. Diagnosis resolution

Once a loss function is defined, finding f to minimize $R(f)$ is an intuitive strategy. Such a f is typically called a “best-fit” answer to the dataset. However, there is a fundamental issue with this strategy: The best-fit answer to a dataset may not

always be the correct answer. This is because the dataset may not be "complete."

In traditional logic diagnosis, the completeness of a dataset is captured in the notion of *diagnosis resolution* [15]. With a logic fault model such as a single stuck-at fault model, diagnosis resolution of a dataset (or of a test pattern set) can be well defined [15]. In that sense, one can decide if a given dataset allows diagnosis to pin-point to a particular fault. If the diagnosis resolution is not high enough, one may only be able to diagnose to a subset of faults and among these faults, no information is available to further distinguish them.

The concept of diagnosis resolution is intuitive in logic fault diagnosis. First, the fault model is discrete and the set of all possible answers is enumerable. Under the single fault assumption, suppose we have n faults $\{f_1,\ldots,f_n\}$ and m test patterns $\{t_1,\ldots,t_m\}$. One can build a *fault dictionary* as that shown in Figure 3. Each entry s_{ij} corresponds to the outcome (or signature) of fault simulation using test pattern t_j assuming that f_i occurs. Note that two signatures can be the same even though the test patterns and/or faults are different.

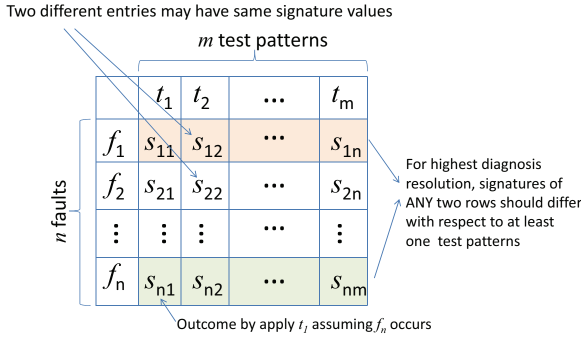

Fig. 3. **Illustration of a fault dictionary**

The diagnosis resolution can be understood using such a fault dictionary. For every pair of faults f_i, f_j, if the signatures across the two rows, row i and row j are different at least with respect to one test pattern, all faults (potential answers) are distinguishable and hence, the the resolution is the highest.

On a given failing silicon chip, the test patterns result in outcomes $\{o_1,\ldots,o_m\}$. Suppose the underlying defect is really a single stuck-at fault. Then, each o_k matches to one or more entries in the column below t_k, which corresponds to a subset of faults. The intersection of all these subsets becomes the answer. If the resolution allows all faults to be distinguishable, then the answer contains only one fault. Otherwise, the answer may contain two or more faults.

If we remove the single fault assumption and employ the same fault model, then we need to consider all possible fault combinations to be the potential answers. Essentially, this expands the fault dictionary from n rows to $2^n - 1$ rows. Suppose the computational cost is not an issue, with such an expanded dictionary, diagnosis resolution can still be defined.

The notion of diagnosis resolution become not so clear when the fault model is not enumerable or contains infinite number of faults. For example, in timing defect diagnosis (or more specifically in statistical timing defect diagnosis), it becomes challenging to apply the diagnosis resolution concept [16]. This is because delay is a continuous value. One may bound this value but it becomes difficult to diagnose to the exact delay value. Conceptually, faults in such a fault model are not enumerable. In this case, one can think that the fault dictionary has infinite number of rows.

The problem becomes even more challenging if we try to diagnose not defects but any potential issues that may impact timing, for example modeling errors, layout issues, issues associated with a timing sign-off flow, and so on. In such a scenario, one may not even have a clear idea of how to define a structure of the fault model to begin with.

B. Diagnosis is an ill-posed problem

The diagnosis resolution defined in traditional logic diagnosis captures a notion of *completeness* in a given dataset, i.e. whether the dataset has enough information to pin-point an exact answer. With a logic fault model and single fault assumption, it is possible to achieve a complete dataset because the number of possible answers is limited. In general (and in practice), however, the dataset is incomplete.

With an incomplete dataset, the problem of finding an answer $f \in \mathcal{F}$ to fit the equation $A \circ f = M$ becomes *ill-posed*. This means that the best-fit answer f, for example by minimizing the $R(f)$ in equation (2) can change if M is replace with a new dataset M'. In other words, there is an inherent instability in the problem solving process, i.e. the best answer can heavily depend on the dataset.

Ill-posed problems arise when one tries to uncover unknown causes from known consequences. The mathematical notion of ill-posed problem was first formulated in the early 1900s by Hadamard when solving the operator equation $Af = F$, $f \in \mathcal{F}$ where f and F are continuous functions and A is a mapping (operator) from functions to functions [14].

C. Solving an ill-posed problem - Regularization

The main issue of solving an ill-posed problem is that we have only limited information in the data. Hence, in performing inference, we should not over-fit the data. This is to avoid having an answer that overly depends on the data. This means that during the inference process, somehow we need to *generalize* the information in the dataset [14]. One way to avoid over-fitting is to relax from finding the "best" answer fitting the dataset, i.e. the one that minimizes the loss function $R(f)$. This relaxation strategy is called *regularization* that was proposed in 1960s for solving ill-posed problems. With regularization, instead of minimizing a loss function, one minimizes a regularized function $R^*(f)$:

$$R^*(f) = R(f) + \gamma\Omega(f) \quad (3)$$

where γ is a constant representing the tradeoff between minimizing the loss $R(f)$ and minimizing the regularization function $\Omega(f)$.

A regularization function is data independent. It characterizes some property regarding the solution f itself. For example, one common way to define $\Omega()$ is to measure the complexity of f. With such a regularization function, one is basically

saying that a simpler answer will be preferred over a complex answer. For example, if $R(f_a) = R(f_b)$ and f_a is more complex than f_b, then f_b is a better answer even though both have the same loss value.

Once the complexity measure $\Omega()$ is chosen, there can be two strategies to minimize $R^*(f)$ [14]:

Fixing $R(f) \approx 0$, minimizing $\Omega(f)$ One can fix a target for $R(f)$, say $R(f) = 0$ and then try to minimize $\Omega(f)$. In diagnosis, this translates to finding the *simplest* explanation to almost perfectly fit the dataset, i.e. able to explain almost every bit difference in the dataset $P - M$ (finding the lowest-complexity f such that $A \circ f \approx M$). For example, if one can find an answer with one fault to perfectly explain the dataset, even though there is another answer with two faults that can also perfectly explain the dataset, the two-fault answer is not preferred (assuming one takes the view that a two-fault answer is "more complex" than an one-fault answer).

Fixing $\Omega(f)$, minimizing $R(f)$ One can fix the complexity of the answer and then try to minimize $R(f)$, i.e. finding the best-fit answer from all possible answers within a given complexity level. For example, in diagnosis one take the assumption of single fault occurrence based on a fault model. This assumption corresponds to limiting the complexity of all possible answers first. In this case, $\Omega(f)$ is a constant because every answer contains a single fault (assuming that complexity is measured by the number of faults). Then, the diagnosis problem becomes finding the best single fault that minimizes $R(f)$.

With a fault model, the second strategy is more intuitive. However, by limiting the complexity of all potential answers, one may not find an answer that explains the dataset well. In other words, with a pre-defined fault model, we often do not find a f to achieve $R(f) \approx 0$. In practice, the cause(s) behind an observed dataset M can be much more complex than what a fault model is capable of explaining. Hence, by fixing the complexity measure $\Omega(f)$, although it avoids the over-fitting issue, one may not find an answer f that is satisfactory.

In diagnosing a large number of failing chips, the second strategy may result in significant information loss. When many chips are not explainable by the strategy, people tend to discard them and focus on those that can be explained.

The first strategy, although not intuitive, is more general for diagnosis without the limitation of a fault model. Because of that, it is more suitable for application in situations where a large number of chips are used in diagnosing timing modeling errors, tool issues, layout issues, and so on. In this work, we therefore follow the first strategy. To implement such a strategy, we take the perspective that diagnosis can be seen as a form of data learning.

D. Link to data learning

To draw the link between diagnosis and learning, we take a typical regression problem formulation as an example.

In regression, we are given a dataset $(\mathbf{X}, \vec{y})$ as shown below where each y_i is the measured result of an unknown function $F(\vec{x}_i)$ corrupted by some random noise, i.e. $y_i = F(\vec{x}_i) + \xi_i$ where ξ_i is a random variable that models the noise.

$$\mathbf{X} = \begin{vmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \dots \\ \vec{x}_m \end{vmatrix} = \begin{vmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{vmatrix} \quad \vec{y} = \begin{vmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{vmatrix}$$

Fig. 4. **Illustration of a regression dataset**

Regression estimates a regression function $f(\vec{x}) = p(y|\vec{x})$ which is a conditional probability function such that $\int p(y|\vec{x})dy = F(\vec{x})$, i.e. the expected value of y given an $\vec{x}$ is $F(\vec{x})$. By letting A be the integral opeation "$\int$", we can re-write the problem as finding $f(\vec{x})$ to solve the equation $Af(\vec{x}) = F(\vec{x})$, or simply $Af = F$ that is in the same form as shown in equation (1) before. Again, the problem is ill-posed. In particular, the dataset $(\mathbf{X}, \vec{y})$ is only a snap-shot of the behavior given by F.

Let $\mathcal{F} = \{f_1, f_2, \dots\}$ be the set (infinite sequence) of regression functions that we can choose from (In the context of diagnosis, this is to say that a model contains an infinite sequence of potential causes). The reason of having infinite number of functions, is that we are working in a continuous domain. For a given $f \in \mathcal{F}$, a typical loss function can be defined as:

$$R(f) = \sum_{i=1}^{m} ||Af(\vec{x}_i) - y_i||^2 \qquad (4)$$

Similarly, the regularized function $R^*(f)$ now becomes:

$$R^*(f) = R(f) + \gamma\Omega(f) \qquad (5)$$

where $\Omega(f)$ measures the complexity of function f. Vapnik in his celebrated statistical learning theory work [14] explains how the complexity of a function can be measured. Once such a measure $\Omega()$ is defined, one can search for the lowest-complexity function that fit the data, i.e. minimizing $\Omega(f)$ constrained by $R(f) \approx 0$ (the first strategy discussed above).

To map the regression problem to the diagnosis problem, consider that $\vec{y}$ is a result derived from the measurement data. Each y_i corresponds to a silicon measurement. $\vec{x}_i = (x_{i1}, \dots, x_{in})$ consists of values on a set of n "factors" $(d_1, \dots, d_n)$ that can potentially impact the measurement. Each d_i is called a *design feature*. For example, each design feature may correspond to some design property. Then, the regression problem becomes trying to find the best function f that maps a set of design properties to the measurement result.

Suppose a f can be found such that $R(f) \approx 0$. f can be seen as an explanation for the result $\vec{y}$ in terms of the design properties. However, such an explanation may not be useful because it could be hard to look at the function and make some practical sense out of it. Hence, additional information needs to be extracted from f. One way to extract such information is

to rank the features based on f. For example, one can decide the top 3 design properties that impact the measurement result the most. These top 3 properties (or features) become the result of diagnosis which are output to the user. For example, this approach was employed in the methodology proposed in [17].

III. Data Learning in Diagnosis

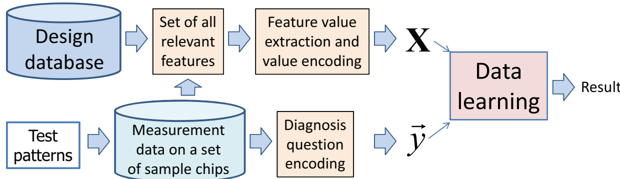

Fig. 5. **Data learning based diagnosis flow**

Figure 5 illustrates the data learning based diagnosis flow discussed above. In this flow, design database and measurement data can be obtained from existing design and test infrastructures. Depending on the application scenario, the design database may consist of such as design netlist, timing model, layout information stored as Lef/Def, timing constraints, RC table, STA timing report, and so on. Measurement data may consist of for example, measured delays for a set of paths across a set of chips. There are four new components to enable the diagnosis flow: extracting relevant features, feature value encoding, diagnosis question encoding, and data learning.

A. Extraction of relevant features

Design databases are complex, containing all information on a design and its design process. Given a set of measurement data, not all design features are relevant. For example, some cells are not used in the circuit to be analyzed. Extracting relevant features is the first step to narrow down the search space for the diagnosis. This step can be based on a manually-defined feature scheme. For example, the work in [18] defines features in terms of cells and wires. The work in [17] gives a more comprehensive list of features dividing into five categories: cell-based, interconnect-based, location-based, dynamic effects, and tool-related effects. Other features can be defined. For example, one can define features to capture the different characteristics between a data path and its clock path, such as the difference between the number of low-vt cells used on the two paths.

In the ideal situation, relevant features should be automatically extracted from the database given the measurement data. However, such a tool is still under development.

B. Feature value encoding

Given n features $\{d_1,\ldots,d_n\}$, for each measurement i, the values of the features, $x_{i1},\ldots,x_{in}$ need to be computed. Each measurement corresponds to a portion of a circuit with some measurement condition. Hence, typically these features describ the characteristics of the circuit and the condition. Depending on the feature, the value can mean different things. For example, a feature may describe the slew rate while another may describe the load. Hence, their values need to be treated differently. For example, the work in [17] discusses the use of different *kernel functions* to interpret different types of features in the learning.

C. Diagnosis question encoding

Given the measurement data, one needs to form a diagnosis question. This is straightforward if there are clearly two classes of measurements, for example the class that contains all measurements within the expected values and the class out of the expected range. In other scenarios, more analysis needs to be done to capture "abnormal behavior".

For example, m measurements $t_1,\ldots,t_m$ can correspond to the measured delays on m paths. Suppose their predicted delays are $p_1,\ldots,p_m$, respectively. We can let $y_i = p_i - t_i$ for $1 \le i \le m$. Then, the diagnosis question should ask why predicted delays are not the same as measured delays. While diagnosis for inaccurate prediction is intuitive, in practice it is not the most interesting application.

A more interesting application can be letting $y_i = 1$ if $p_i - t_i \ge 0$ and $y_i = 0$ if $p_i - t_i < 0$. In this case, the data is divided into an over-estimation class and an under-estimation class where measurements fall into the over-estimation class are considered as normal. Typically, one should see far more measurements falling into the over-estimation class than the under-estimation class. The diagnosis is then asking why under-estimation occurs.

The two-class example may not always be applicable in practice. For the two-class diagnosis question to make sense, the predicted and measured delays have to be based on the same environmental conditions (ex. temperature and voltage, etc.). In practice, measured data may not correspond exactly to any of the timing sign-off corners. In that case, the two-class diagnosis question described above becomes less meaningful.

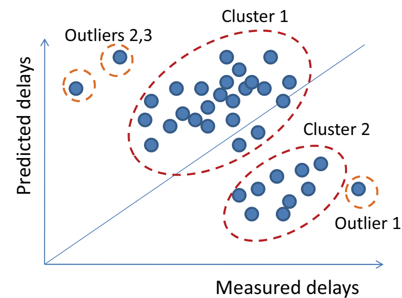

Fig. 6. **An example of measurement-prediction data plot**

Figure 6 illustrates an example where measured delays are plotted against predicted delays. In this plot, all measurements under the 45° line are in the under-estimation class based on the definition above. However, this mismatch may be due to that timing analysis used to generate the predicted delays was run at the nominal corner rather than a worst-case corner. If running at a worst-case corner, many points in the under-estimation class can be moved to the over-estimation class.

Given such a plot, one interesting question to ask is not why under-estimation occurs, but for example, why there are two clusters of data. Note that this trend is independent of the notions of under-estimation or over-estimation (the 45° line has no meaning here). Another interesting question can be asked is regarding why the three *outliers* occur. For example, outlier 1 represents the most under-estimation case, relative to all other measurements. One can ask the question what causes the outlier to happen. This can still be seen as a two-class problem where one class contains all points but the outlier and the other class contains only the outlier.

The example in Figure 6 shows that in general, one can try to diagnose the reason behind an unexpected trend (such as two clusters) or the reason behind extremely unexpected behavior (such as outliers). This is possible even when the environmental condition in measuring the data is slightly different from the sign-off condition in the timing analysis flow and hence, extends the applicability of the diagnosis flow.

D. Two types of data learning for diagnosis purpose

After the above three steps, data from design database together with test measurement data are formatted into a dataset $(\mathbf{X}, \vec{y})$ as shown in Figure 5 above. Various learning techniques may be applicable at this point. Typically, one can perform two types of learning on the dataset, depending on the characteristic of the dataset and the application scenario.

Feature ranking As mentioned before, feature ranking is used to identify the top features that are most relevant to the diagnosis question. In feature ranking, the learning problem is either formulated as a regression problem [17] or a classification problem [18]. If it is a classification problem, there should be enough samples in each class for meaningful learning to take place. If one class contains very few samples, for example only one sample, then the rule extraction described below is more appropriate.

Rule extraction Sometimes, knowing the top features may not be enough. One desires to diagnose to a more explicit answer. In other occasions, the size of measurements in the two classes of data differ significantly, producing a very unbalanced dataset. In such application scenarios, rule extraction can be applied.

The main body of a rule is a combination of features. For example, let d_1 be a feature denoting the number of a low-vt AND cell and d_2 be a feature denoting the delay on the clock path calculated by the timing analysis. A rule may look like "If $d_1 \geq 2$ and $d_2 \geq 25ps$ then the path delay is under-estimated with a probability of 98%". Given a dataset, multiple rules are usually extracted because a single rule may not be able to explain all behavior reflected in the dataset, analogous to traditional logic diagnosis using a multi-fault model when no single fault can fully explain the data.

IV. Experience with learning algorithms

In the field of machine learning, many types of algorithms have been proposed and studied. Generally speaking, learning algorithms can be divided into two categories: supervised and unsupervised. In supervised learning, dataset is given as $(\mathbf{X}, \vec{y})$. In unsupervised learning, only $\mathbf{X}$ is given (without $\vec{y}$). For examples, regression and classification are supervised. Clustering and association rule mining are unsupervised.

Popular algorithms for supervised learning include Neural Network and Random Forests [19]. Support Vector Machine (SVM) [20], and more generally kernel-based learning [21] emerged to be popular learning paradigms in the past decade and have been applied widely in many fields. Gaussian Process (GP) [22], which can be thought as a Bayesian version of the SVM approach, gained popularity in recent years as it not only can learn more effectively but also can provide confidence estimate of its learning.

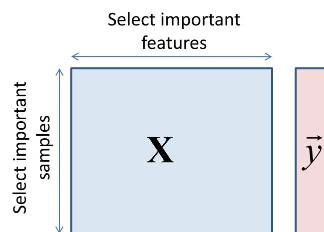

Fig. 7. **Supervised learning can be carried out in two directions**

Figure 7 illustrates the two directions that a supervised learning algorithm may carry out. In the horizontal direction, a supervised learning algorithm may weight the importance of features. One can look at this as finding the minimum set of features for building a model. Typically, by picking up the most important features that influence $\vec{y}$, a better model can be learned. For example, Random Forests and Gaussian Process algorithms both are capable of weighting features.

In the vertical direction, samples are weighted based on their importance. For example, in SVM some samples are weighted zero, meaning that they contain redundant information for building the model. These samples become *non-support vectors*. When analyzing a large number of chips, support vector analysis can be applied to select the most important chips for subsequent analysis.

In general, a supervised learning algorithm carry out optimization in either direction or both. For example, SVM is designed to optimize in the sample direction and usually can handle a large number of samples (say hundreds of thousands or millions) with a large number of features (say thousands or tens of thousands). GP typically is multiple times slower than SVM because it carries out optimization in both directions.

Consider a learning algorithm that takes a dataset as shown in Figure 7 and tries to build an almost perfect model for explaining the dataset, by finding a minimal set of features and a minimal set of samples. This is the strategy discussed in Section II- C before as "fixing $R(f) \approx 0$, minimizing $\Omega(f)$" where the complexity measure $\Omega()$ is measured in terms of number

of features and number of samples. Hence, we see that modern learning algorithms such as SVM and GP both follow this strategy.

A learning algorithm such as SVM and GP build a model to map $\mathbf{X}$ to $\vec{y}$. Typically, this model is used for prediction purpose, not for interpretation purpose. To interpret the result of learning, one can either rank features based on the model or extract some *rules* directly from the data. Moreover, selecting important features and samples can be employed as a preprocessing step to reduce the size of dataset and facilitate the subsequent analysis step such as rule extraction.

In machine learning, two types of rule extraction paradigms exist. When multiple classes are present in the data, *rule induction* is applied. Some best known rule induction algorithms are the CN2 [23] and its extension the CN4 algorithm [24]. For example, in a recent work [25] the authors propose using rule induction for analyzing unexpected silicon behavior divided into classes. When data are given without classification (without $\vec{y}$), *association rule mining* is applied. To handle continuous feature values, the quantitative association rule mining approach [26] is among the first methods proposed. Recently, a special rule extraction technique was proposed [27] to handle the cases where there are two classes of samples, and one class has only one sample and the other class contains the rest of the samples.

In most of the diagnosis scenarios addressed by the proposed work, data are divided into classes. Hence, the CN2/CN4 algorithms or special rule extraction technique [27] can be used. Association rule mining may be applied when only one type of data is available, for example only failing data are available. Note that the quality of rules extracted based on two classes of data is usually higher than those with one class. In other words, it is easier to diagnose by comparing information in two classes of data than by using only one class of data.

Unsupervised learning is applied when analysis is carried out on one class of data. For example, the one-class SVM algorithm [20] has shown great promise in outlier analysis [28]. In diagnosis, outlier analysis might be applied as a preprocessing step to divide samples into classes (outliers and the rest of the population) if the data is given without pre-labeled classes. One-class SVM has been applied in another context that is closely related to diagnosis — to implement *similarity search* [29]. Suppose that the objective of diagnosis is to find fixes. Suppose we are given a number of speed paths as bad examples. The goal is to find other paths to improve based on what we learn from the speed paths. This can be formulated as a similarity search problem. The work in [29] applies one-class SVM to build a model for the speed paths. Then, this model is used to scan all other paths to find similar paths for improvement. Recently, a more sophisticated similarity search approach was proposed and applied to the speed path analysis problem using industrial data [30].

V. Application scenarios and considerations

The proposed diagnosis framework has been applied in two scenarios in practice. The first scenario is speed path analysis and the second scenario is calibration of a timing analysis flow.

In high performance design, speed paths are paths that limit the performance of a chip. Speed paths are typically identified using silicon samples. After these paths are found, one desires to understand what make these paths special. In such a diagnosis scenario, we are given with a small set of speed paths and a large set of non-speed paths. Hence, the learning is by comparing speed paths to non-speed paths.

In the second scenario, a large number of paths are measured with scan test patterns on a number of chips. The path delays are compared to the delays predicted by timing analysis. In each diagnosis task, some paths are selected to be the "abnormal" paths. Note that this abnormality is defined by referencing to the timing analysis result. For example, some paths are much more under-estimated than the rest of the paths. Note that this problem formulation is different from the speed path analysis scenario where timing analysis result is not used in defining the speed paths.

In both scenarios, we are given two classes of paths, one smaller set of paths deemed to be special and the other larger set of paths deemed to be the normal population. The goal is to uncover the reasons behind the special paths.

The first important consideration is to decide if there are commonalities among the special paths. If each special path is unique in its nature, one may want to diagnose each special path individually. In this case, the approach proposed in [27] is more suitable. If one desires to find common reasons behind multiple paths, then it can be treated as a two-class problem and a rule induction approach such as [23][24] can be applied.

The second important consideration is to define the set of features. In practice, this is not a trivial step. In the ideal situation, one would like to throw in as many different features as possible and let the diagnosis framework to figure out what features are relevant and what feature are not. In reality, features may be hierarchical and diagnosis is applied iteratively. For example, one may begin with a rough set of features covering cells and wires. If the diagnosis result indicates that the problem is largely due to wires, then in the next round, more detailed layout features describing the wires may be used. The hierarchical approach allows user to focus diagnosis at the abstraction level of his/her interest. This is particularly useful for interpretation of the diagnosis result in practice.

The third important consideration is in the validation of diagnosis result. Suppose the result is given as a set of rules. Typically, the learning framework is capable of giving a confidence measure on each rule it reports. However, one usually would like to validate such rules before treating them as design fixes. The validation process could be manual, for example by consulting with the designer. The validation could be simulation based, for example by applying the rules in a hypothetical simulation environment to test their impact.

The last important consideration is how to apply the diagnosis result as design fixes. For example, if a rule says that using a particular cell too many times along a clock path could cause a problem, then one needs to decide what action to take to fix the design. From the application point of view, one also needs to decide how specific a rule is, i.e. whether it is general

enough for other designs and/or technology nodes to adopt or it is specific to the design and/or technology node only.

VI. Conclusion

This paper summarizes a data learning based diagnosis paradigm that we call transduction diagnosis for differentiating it from traditional root causing approach. In transduction diagnosis, design fixes are learned directly from data without going through the expensive root causing steps. These fixes can be in the form of a feature rank or feature-based rules, in terms of features characterizing the design databases. We discuss various learning algorithms that can be used to implement various functions in a transduction diagnosis framework and also discuss practical considerations when applying such a framework in different scenarios. The proposed framework is still in its early development phase. More industrial experiments are required to evaluate its effectiveness in practice. Additional tools are under development to facilitate the integration of the framework into existing design, test, and diagnosis flows.

References

[1] K. Killpack, C. Kashyap, E. Chiprout. Silicon Speedpath Measurement and Feedback into EDA flows. *ACM/IEEE Design Automation Conference*, 2007, pp. 390-395

[2] M. Buhler et al. DFM/DFY design for manufacturability and yield - influence of process variations in digital, analog and mixed-signal circuit design. *Design, Automation and Test in Europe*, 2006, pp. 387-392.

[3] H. Ramakrishnan, S. Shedabale, G. Russell, A. Yakovlev. Analysing the effect of process variation to reduce parametric yield loss. IEEE International Conference on integrated Circuit Design and Technology, issue 2-4, 2008, pp. 171-176.

[4] C. Bittlestone, A. Hill, V. Singhal, Arvind N.V. Architecting ASIC Libraries and Flows in the Nanometer Era *ACM/IEEE Design Automation Conference*, 2003, pp. 776 - 781.

[5] P. Zuchowski, P. Habitz, J. Hayes, J. Oppold, Process and Environmental Variation Impacts on ASIC Timing. *ACM/IEEE International Conference on CAD* 2004, pp. 336-342.

[6] N. NS, T. Bonifield, et al, BEOL Variability and Impact on RC Extraction. *ACM/IEEE Design Automation Conference* 2005, pp. 758- 759.

[7] R. Franch et al. On-chip Timing Uncertainty Measurements on IBM Microprocessors. *International Test Confernece*, 2007.

[8] Li-C. Wang and Magdy S. Abadir. Dealing with timing issues for sub-90n designs — from modeling to mass production. Full-day tutorial, *International Test Conference*, 2005-2007.

[9] Noel Menezes, Chandramouli Kashyap, Chirayu S. Amin. A ”true” electrical cell model for timing, noise, and power grid verification. *ACM/IEEE Design Automation Conference*, 2008, pp. 462-467.

[10] F. N. Najm and N. Menezes. Statistical timing analysis based on a timing yield model. *Design Automation Conference*, June 7-11, 2004, pp. 460-465.

[11] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan. First-order incremental block-based statistical timing analysis *ACM/IEEE Design Automation Conference*, June 7-11, 2004, pp. 331-336.

[12] D. Josephson et. al. Debug methodology for the McKinley processor *International Test Conference*, 2001, pp. 451-460.

[13] M. Abramovici, M. Breuer. Fault Diagnosis Based on Effect-Cause Analysis: An Introduction. *ACM/IEEE Design Automation Conference* 1980, pp. 69-76.

[14] Vladimir N. Vapnik. The Nature of Statistical Learning Theory. 2nd edition, Springer 1999

[15] Miron Abramovici, Melvin A. Breuer, Arthur D. Friedman. *Digital Systems Testing and Testable Design* Wiley-IEEE Press, 1994

[16] A. Krstic, et al. Delay Defect Diagnosis Based Upon Statistical Timing Models - The First Step *European Design Automation and Test Conference*, 2003, pp 328-333

[17] Pouria Bastani, et al. Diagnosis of design-silicon timing mismatch with feature encoding and importance ranking - the methodology explained. International Test Conference, 2008

[18] Li-C. Wang, et al. Design-Silicon Timing Correlation — A Data Mining Perspective. *ACM/IEEE Design Automation Conference*, 2007, pp 384-389

[19] Leo Breiman, Random Forests *Machine Learning Journal (45)*, 2001, pp. 5-32.

[20] Bernhard Schlkopf, and Alexander J. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press, 2001.

[21] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis. *Cambridge University Press* 2004.

[22] Carl Edward Rasmussen, and Christopher K. I. Williams. Gaussian Processes for Machine Learning. The MIT Press, 2005.

[23] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning* 3, 1989, pp 261-283

[24] Ivan Bruha and Sylva Kockova. A support for decision-making: Cost-sensitive learning system. *Artificial Intelligence in Medicine*, issue 1, vol 6, 1994, pp 67-82

[25] Nicholas Callegari, et al. Classification Rule Learning using Subgroup Discovery of Cross-Domain Attributes Responsible for Design-Silicon Mismatch. *Manuscript*, Nov 2009

[26] Yonatan Aumann and Yehuda Lindell. A statistical theory for quantitative association rule. *Journal of Intelligent Information Systems*, 20:3, 2003, pp 255-283

[27] Nicholas Callegari, et al. Speedpath Analysis Based on Hypothesis Pruning and Ranking. *ACM/IEEE Design Automation Conference*, 2009.

[28] Sean H. Wu, Dragoljub Drmanac, Li-C. Wang. A Study of Outlier Analysis Techniques for Delay Testing. *International Test Conference*, 2008.

[29] Pouria Bastani, et al. Speedpath Prediction Based on Learning from a Small Set of Examples. *ACM/IEEE Design Automation Conference*, 2008, pp 217-222

[30] Nicholas Callegari, et al. Feature based similarity search with application to speedpath analysis. *International Test Conference*, 2009