Data Mining in Functional Test Content Optimization

Li-C. Wang
University of California at Santa Barbara

Abstract — This paper reviews the data mining methodologies [1]-[4] proposed for functional test content optimization where tests are sequences of instructions or transactions. Basic machine learning concepts and the key ideas of these methodologies are explained. Challenges for implementing these methodologies in practice are illustrated. Promises are demonstrated through experimental results based on industrial verification settings.

I. Introduction

Data mining is the process of uncovering patterns in data. Many practical data mining tasks involve application of machine learning concepts and algorithms. The term data mining is more recent and less rigorously-defined than the term machine learning where its origin, as described in the statistical learning theory [5], can be traced back to the Perceptron [6].

In his book [5], Vapnik divides the research of machine learning into four periods: (1) The Perceptron (1960s), (2) Construction of the fundamentals of learning theory (1960-1970s), (3) The Neural Networks (1980s), and (4) Construction of alternatives to the Neural Networks (1990s). Perceptron is viewed as the origin of machine learning because it is the first algorithm to demonstrate that a machine can "learn."

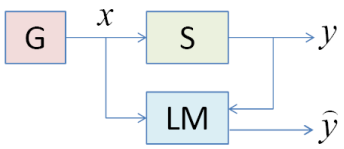

Fig. 1. Setting of a learning problem

In this machine learning view, the word "learning" takes a functional perspective as illustrated in Fig. 1. The learning problem comprises three components: (1) A generator G that draws independent samples x from an unknown but fixed probability density function $p(x)$; (2) A supervisor S that takes x as input and outputs a value y according to a conditional distribution function $F(y|x)$ or a function $y = f(x) + \sigma$ for some random noise σ; (3) A learning machine LM capable of implementing a set of functions $g_\alpha(x) = \hat{y}$ based on a vector of parameters $\alpha \in \Lambda$. In supervised learning, the goal is to learn the function $f()$ based on samples of (x, y). In unsupervised learning, the goal is to learn the density function $p(x)$ based on samples of x. In this case, there is no y.

The performance of the learning machine is usually measured by a loss function, in supervised learning as $L(f, g)$ and in unsupervised learning as $L(p, g)$, which characterizes the error of the learned function g, i.e. the discrepancy between the true answer (f or p) and g. Let L_n be the empirical error calculated based on n samples. The learning theory concerns two fundamental questions: (1) how to guarantee that as $n \to \infty$, the learned function g will converge to the true function, and (2) how to construct a learning algorithm that delivers an optimal converging rate. In other words, the learning theory studies the asymptotic behavior of a learning machine.

For many data mining applications in Electronic Design Automation (EDA) and test, the asymptotic behavior of a learning machine is not the first concern. This is because in these applications [7] one often faces the situation that the data is limited and obtaining large amounts of additional data is cost prohibitive. For these applications a more important concern is: not having a dataset with sufficient information.

To provide the missing information to a learning machine, domain knowledge is required. Applying domain knowledge in learning is not new. In fact, in theory, learning (or generalization from data) would not be possible without any knowledge [8]. For example, in Fig. 1 one assumes that G follows a fixed distribution and the samples are drawn independently. Another example is that traditional statistical analyses such as outlier analysis [9] and linear/quadratic discriminant analysis [10] assume Gaussian densities in the underlying data. The Gaussian assumption enables the analyses to be applied on small-size dataset. Therefore, the question is never whether learning requires domain knowledge or not. The question is how much.

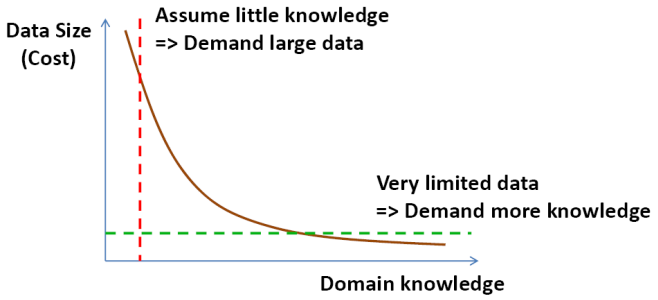

Fig. 2. Tradeoff between knowledge and data

Conceptually, one can think of a tradeoff picture as shown in Fig. 2. On one end, minimal knowledge is employed to make a learning machine as general as possible - almost all information regarding the underlying function to be learned has to come from the data. This can be thought of as the area of focus for traditional machine learning research. On the other end, one has very limited data. To enable learning, substantial domain knowledge is required. In this case, the learning machine becomes specific to the problem domain.

For many data mining applications in EDA and test, one often works with data that is limited. Hence, the learning problems are close to the bottom-right corner of the tradeoff curve. For these problems, the challenges lie in how to obtain and incorporate domain knowledge into the data mining tasks [7].

Because domain knowledge is heavily involved, solving these problems should not be thought of as simply applying some existing machine learning concepts and algorithms. More importantly, it is about developing a methodology that can effectively incorporate domain knowledge into the learning flow. Naturally, such a methodology would be application specific [7].

This paper reviews the data mining methodologies proposed for functional test content optimization (FTCO) [1]-[4]. One of the application contexts is functional verification.

Functional verification starts with a verification plan, specifying the aspects of the design to verify [11]. In addition to manually-written direct tests, tests can be generated by constrained random test generation, which is guided by constraints and biases specified in a test bench (or test template). Verification quality is measured by coverage metrics and coverage results are analyzed to guide the development of more effective direct tests and test templates.

Design is an evolutionary process. From one revision to the next, new features are added and/or bugs are fixed. Functional verification evolves accordingly. In this evolutionary process, assets accumulated through the verification efforts are the important tests and test templates which are collected into a regression test suite. In this context, functional test content optimization can be viewed as the process of optimizing this regression test suite for achieving a desired quality level.

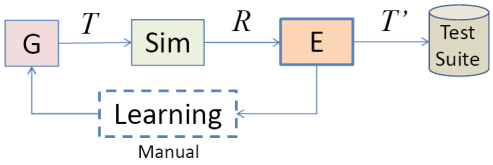

Fig. 3. Existing manual learning for FTCO

In a typical verification flow, learning for FTCO is carried out mostly by manual efforts. Fig. 3 illustrates this process: The generator G produces a set of tests T. The generator can be a person or a constrained random test generator. Each test can be a sequence of instructions/transactions, i.e. an assembly program. The simulator Sim simulates the set of tests and produces the result R that is evaluated through a step E to select the important tests and/or test templates T' to be added into the test suite. For example, an important test could be the one that excites a bug or provides a unique coverage. When the result is not satisfactory, some aspects of the generator (e.g. the test template) are modified to produce new tests. This modification is based on manual learning from the simulation result R and the information collected through step E.

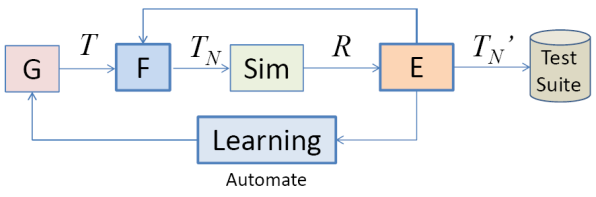

Fig. 4. Two data mining components added in FTCO

Fig. 4 shows two data mining components that can be added into the existing functional test content optimization flow. First, a filter F is added between the generator and the simulator. The objective is to *filter-in* only the novel tests for the simulation and discard the non-novel tests. Here a novel test could be thought of as the one whose characteristics are dissimilar to those tests that have been simulated. The assumption is that a novel test has a much higher chance to be an important test. Another important assumption is that the cost of the simulation is much higher than the cost of the test generation.

Suppose the number of tests to be simulated is limited at N. Without the filtering, N tests would be generated and simulated. With the filtering, $k * N$ tests could be generated and N tests would be simulated, assuming one novel test among every k tests. The idea is that there should be more important tests in the N novel tests than those in the N original tests. Consequently, a more effective test suite can be obtained.

The second component intends to automate the learning process for improving the test generation G. The improvements can be twofold. For a coverage point that has been covered by only a few tests, the learning can be used to obtain more tests to increase the coverage frequency. For a coverage point that is not yet covered, the learning can be used to increase the chance of generating tests to cover the point.

In Fig. 4, if the simulator is replaced with silicon application then the same ideas could be applicable in the context of post-silicon validation. With respect to the two data mining components, works [1][2] proposed methodologies to implement the filtering component. Works [3][4] developed a methodology for the learning and feedback component.

The rest of the paper is the following. Section II summarizes the basic machine learning concepts and highlights the key challenge for implementing the proposed data mining methodologies in practice. Section III explains the proposed data mining components in terms of their learning problem settings. Section IV discusses practical implementation of the filtering component with example results to demonstrate its effectiveness. Section V discusses practical implementation of the learning and feedback component also with example results to illustrate its effectiveness. Section VI makes a final remark.

II. Basic Concepts and Key Challenge

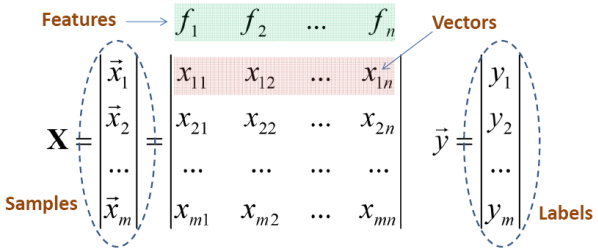

Fig. 5. Typical dataset seen by a learning algorithm [7]

Figure 5 illustrates a typical dataset seen by a machine learning algorithm (For more discussion, see e.g. [7]). When $\vec{y}$ is present and there is a label for every sample, it is called *supervised* learning. In supervised learning, if each y_i is a categorized value, it is a *classification* problem. If each y_i is modeled as a continuous value, it becomes a *regression* problem.

When $\vec{y}$ is not present and only $\mathbf{X}$ is present, it is called *unsupervised* learning. When some (usually much fewer) samples are with labels and others have no label, the learning is then called *semi-supervised* [12].

Each sample is encoded with n features $f_1, \ldots, f_n$. Hence, the characteristics of each sample are described as a vector $\vec{x_i}$. In the context of functional test content optimization as shown in Fig. 4, each sample is a test that is a sequence of instructions/transactions, i.e. an assembly program. The fundamental challenge is therefore to encode an assembly program into an acceptable input format for a learning machine.

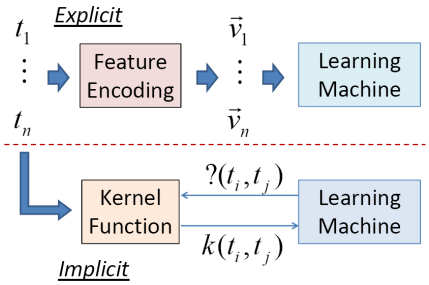

Fig. 6. Two approaches to analyze assembly programs

Fig. 6 shows that there can be two approaches to enable a learning machine to learn from a set of assembly programs. In the explicit approach, a set of *features* is selected. Each test t_i is converted into a sample vector $\vec{v_i}$. In the implicit approach, one develops a *kernel* function that measures the similarity between any given two assembly programs directly.

Many modern learning algorithms follow the paradigm of so-called kernel-based learning where the optimization engine for building the learning model and the definition of the space for the learning are separated (see, e.g. [13][14]). As illustrated in Fig. 6, for a kernel-based machine to learn, it does not need to access the samples directly. Instead, all it needs is the similarity information between pairs of samples. In other words, during the learning process the learning machine queries the kernel function for pairs of tests (t_i, t_j) and the kernel function returns their similarity measure values $k(t_i, t_j)$. Hence, to enable the learning, one can develop a kernel function.

A. Importance of the learning space

A learning machine operates on samples projected into some learning space. This projection can be explicit based on a set of features $f_1, \ldots, f_n$ such as that shown in Fig. 5, or can be implicit based on a kernel function. It is intuitive to observe that the definition of the learning space can substantially dictate the complexity of the learning. Fig. 7 uses a simple example with 20 samples to illustrate this observation.

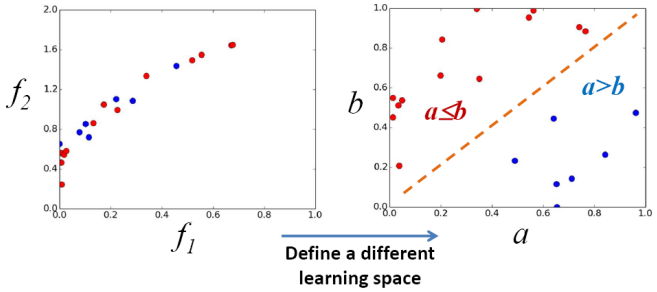

Fig. 7. Simple example to illustrate the importance of the learning space

The example consists of 7 positive samples and 13 negative samples. Hence, it can be seen as a binary classification problem. Suppose these 20 samples are encoded with two features f_1 and f_2, and projected onto the 2-dimensional space as shown on the left plot. In this plot, observe that the two classes of samples are mixed with each other and are hard to be separated. For example, there is no linear model to separate the two classes.

Suppose with domain knowledge, one knows that feature f_1 is generated by two sources a and b as $f_1 = a * b$. Similarly, f_2 is generated by $a + b$. The 20 samples are re-encoded with two source features a and b, and projected onto the new 2-dimensional space as shown on the right plot. As it is observed, the two classes can now be separated easily by the line $a = b$. The positive samples have the property $a > b$ and the negative samples have the property $a \leq b$.

The simple example illustrates that domain knowledge can make a learning problem much easier. A complex problem needs more samples to learn and the resulting model can be hard to interpret. For example, the left plot requires a nonlinear function to separate the two classes. It needs more samples to learn and the resulting model is harder to interpret. In contrast, the right plot is easier to learn and the model is easier to interpret.

The simple example shows another interesting point: If one is looking for an interpretable model, then the learning problem becomes a problem of searching for the learning space where the resulting model is simple and interpretable. In this case, the importance of the learning algorithm can be diminishing [7].

III. Setting of the Learning Problem

A. Problem formulation

To formulate a problem as a learning problem, one needs to consider two sets of questions. The first set concerns the input and output to the learning machine. For example:

- What is a sample?
- Where does the input data come from?
- How is a sample represented?
- How is a learning model represented?
- How will the learning model be applied?

In setting up a learning problem, perhaps the most important aspect is to give a definition of the sample. For example, in earlier discussion, we assume that a sample is a given assembly program. However, depending on the need, one might also consider every k (e.g. $k = 3$) consecutive instructions as a sample, i.e. an assembly program would be broken into multiple samples. It is obvious that learning depends on the definition of what a sample is.

The learning component would not be standalone. It receives input from another component (or components) in an overall flow. It is important to consider the data source(s) and consequently the availability of the data and its characteristics. Sample representation is another important aspect. As discussed above, if a sample is not represented in vector form, additional data processing is required.

Different learning algorithms produce learning models in different forms. The representation of a learning model can be based on rule, tree, equation, a collection of the samples, etc. This leads to the next question as for how will a learning

model be applied. For example, if the model is first to be interpreted by a person, then a complex model would not be of much use. In this case, a rule or a tree model might be more suitable (see e.g. [15][16]). Moreover, some applications may demand models with high accuracy and some may not. The accuracy requirement also dictates how a learning problem should be formulated.

The second set of the questions concerns the choice of the learning approach and the algorithms. For example:

- Should the learning problem be solved by a supervised, unsupervised, or semi-supervised approach?
- Which algorithms are more suitable than others?

Suppose the data comprises n positive samples and m negative samples for $n \ll m$. This situation often arises in learning problems formulated for EDA and test applications [7]. Such a problem could be approached by binary classification with the consideration of handling an imbalanced dataset [17]. This problem could also be seen as finding a space to project the positive samples as outliers (see e.g. [18]) - outlier analysis is a form of unsupervised learning. Or the problem could be approached by techniques designed specifically for semi-supervised learning [12] if most of the negative samples actually mean "don't care" samples. One of the key considerations is the ratio between n and m. For example, if m is on the order of $10n$, binary classification might still be applicable. If m is on the order of $10^6 n$ and n is small (e.g. $n < 20$), it might be more suitable to treat it as an unsupervised learning problem.

Even with a learning approach decided, there can be many algorithms to choose from. For example, a machine learning tool library may consist of more than ten types of algorithms for each approach [19]. Each algorithm may have variations and/or open parameters to be decided on. Moreover, a preprocessing component may be added to a learning machine. For example, dimension reduction or feature extraction [19] may be used to transform the dataset to facilitate the learning.

B. Model validation

Model validation is another crucial aspect of the learning problem setting. In theory, as discussed in the Introduction, one would like to learn the underlying function $f()$ in supervised learning or the underlying distribution $p()$ in unsupervised learning. With a loss function $L()$ defined, in the supervised case, one can evaluate the empirical error $L_m(f, g)$ of a learning function $g()$ based on m samples. For example, the loss function can be based on the average square error that gives the empirical error as $\frac{1}{m}\sum_{i=1}^{m}(f(\vec{x}_i) - g(\vec{x}_i))^2$.

In the supervised case, an empirical error can be calculated for a collection of samples because each $\vec{x}_i$ has a label $y_i = f(\vec{x}_i)$. In the unsupervised case, it is trickier because one does not have a way to access the true density value $p(\vec{x}_i)$. To apply the loss function concept, for example, one can construct an empirical distribution function $p_m()$ as the reference and calculate an empirical error as $L_m(p_m(), g())$ [20] — The error is evaluated in a relative sense, not in an absolute sense.

An empirical error $L_m()$ evaluates a learning model $g()$ based on the given m samples. However, the validity of a model should be evaluated based on future unseen samples. In machine learning, the concept of *overfitting* characterizes the situation where the performance of a learning model on the future samples deviates from the performance of the model on the current samples used in learning. Fig. 8 illustrates the concept.

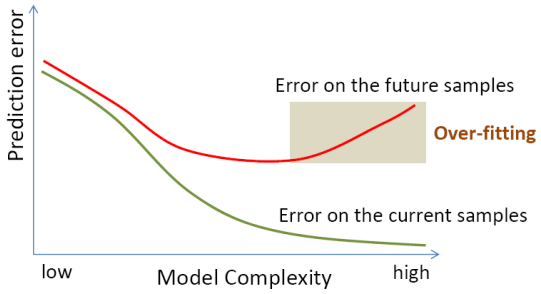

Fig. 8. Illustration of the concept overfitting in learning

In theory [5], one can obtain a model to minimize the empirical error by increasing the model complexity. However, as a model becomes more complex, at some point its performance on the future samples becomes worse while its performance on the current samples continues to improve. If a model falls into this situation, the model is said to be overfitting the data.

To evaluate overfitting, a common way is to perform cross-validation. The data is divided into two sets: a training set consisting of $X\%$ of the samples and a validation set consisting of the remaining samples. For example, $X\%$ could be 90%. The learning is applied to the training set and validated through the validation set to ensure that the empirical errors calculated on both sets are consistent. The process can be repeated k times by randomly selecting the training set and taking the average.

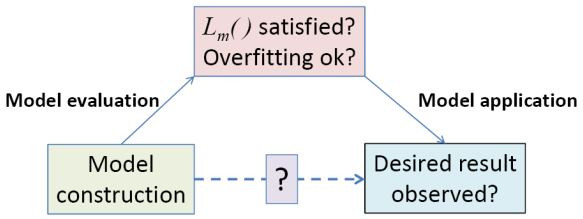

Fig. 9. Challenge faced in many EDA/test data mining applications

In a typical learning problem setting, evaluation of model validity can take place in two stages. Fig. 9 depicts the two stages. In the first stage, empirical errors are calculated and overfitting is evaluated. If both are satisfactory, in the second stage the model is applied and its resulting performance in the actual application is observed.

For example, if the objective of a model is to predict consumer behavior and increase sales, ultimately one needs to observe the sales data for a period of time to conclude the validity of the model. However, evaluating model validity in an actual application can be time and effort consuming. Hence, in many applications it is an acceptable practice to consider a model to be valid if it can pass the first stage of the model evaluation.

In EDA and test applications, it is often the case that the first evaluation stage is not even a viable option. Consider a dataset with n positive samples and m negative samples where $n < 10$ and $n \ll m$. In this case, one might not have enough positive samples to pursue cross-validation. Consequently, a model needs to be validated directly in its application context.

As an application example, in building models for capturing customer returns [18], a model is validated by showing that it can capture future potential customer returns. As another application example, in building models for improving yield, silicon experiments need to be conducted to demonstrate actual yield improvement [21]. Pursuing model evaluation in its application context is usually costly and time consuming. Hence, for many EDA and test applications, one of the key challenges is to devise a viable model evaluation scheme before actual model application. This challenge is denoted in Fig. 9 by "?".

For functional test context optimization, the model evaluation issue is less challenging. This is because the common objective of test context optimization is to improve coverage and coverage can be evaluated through simulation (or silicon application). For example, if a model is learned for hitting a particular event, evaluation of the model can mean (1) modifying the test template (or test) according to the model, (2) applying the resulting test(s), and (3) observing the event coverage. In comparison to other applications such as customer return analysis and yield improvement, model evaluation in functional test content optimization is much less costly.

C. The filtering component

The early work in [22] proposed to implement the filtering component by solving a novelty detection problem with unsupervised learning. Fig. 10 illustrates the idea. Suppose a set of tests T has been simulated. In novelty detection the learning machine builds a model to capture the "space" represented by the tests in T. The model is then used to filter future tests. If a test falls inside the boundary of the model, it is discarded. If a test falls outside, it is novel and selected for simulation.

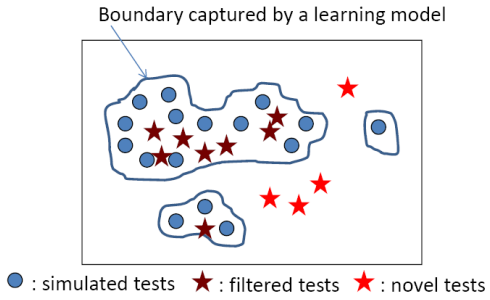

Fig. 10. Filtering by performing novelty detection

One of the learning algorithms which can be used to build such a novelty model is the Support Vector Machine (SVM) one-class algorithm [13] which was used in [22]. In [22], the tests were assumed to be sequences of binary vectors.

To extend the work to analyze assembly programs, the work in [23] developed a kernel function that could measure the similarity between two assembly programs. Similarity between two programs was measured based on their program flow graphs, more specifically based on the *minimal edit distance* to convert one program graph to another.

What is the "space" represented by a set of tests? With kernel-based learning, this space is dictated by the similarity measure function, i.e. the kernel function. Fig. 11 depicts the concept. Suppose t_1 has been simulated. Two more tests t_2 and t_3 are available. Suppose only one test can be simulated. Should it be test t_2 or test t_3? Suppose a kernel function $k()$ is defined. The similarity between t_1 and t_2 is measured as $s_{12} = k(t_1, t_2)$ and the similarity between t_1 and t_3 is measured as $s_{13} = k(t_1, t_3)$. It is intuitive to see that if $s_{12} > s_{13}$, i.e. t_2 is closer to t_1 than t_3 to t_1, then t_3 should be chosen.

The SVM one-class algorithm essentially works on such a kernel-induced space to find a model that includes most of the sample points. For example, one can choose a parameter of the algorithm to guarantee that the SVM learning model would leave at most one point outside the modeled space [13].

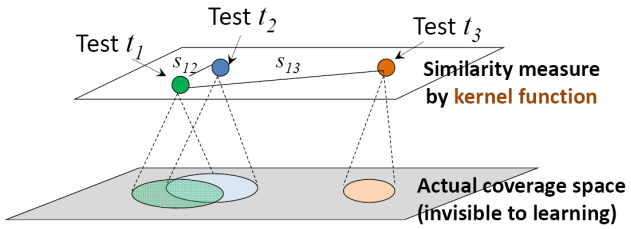

Fig. 11. Learning space vs. actual coverage space - ideal situation [23]

Fig. 11 also illustrates the fundamental challenge in defining a proper kernel function. Ideally, the similarity measure between two tests should reflect their similarity in terms of the actual coverage in the coverage space. However, before tests are simulated or applied, their actual coverages are unknown. Hence, the information in the actual coverage space is not available to the learning machine.

More importantly, the coverage space can change from one verification task to another. Consider that one task is to verify unit A and the other task is to verify unit B. Coverage of unit A is defined based on the signals in unit A, which would be different from the coverage defined for unit B. This means that one should have one kernel for unit A and another kernel for unit B. In other words, the intent of the verification task should be taken into account in the definition of the kernel function.

While the early works demonstrate promises of using a filtering component to improve test content, they did not address the question of how to incorporate verification intent (i.e. domain knowledge) into the learning machine. This is a crucial consideration when the proposed learning approach is applied in an actual industrial verification environment.

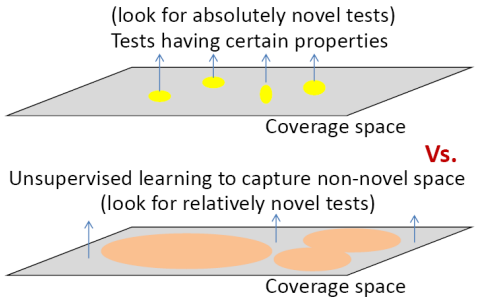

Fig. 12. Absolute novelty vs. relative novelty

With unsupervised learning, a novel test is the one that is dissimilar to the simulated tests. In this setting, the novelty is defined relatively. In some applications, one may want to look for novel tests that have certain properties. For those applications, the definition of novelty becomes absolute. Fig. 12 depicts the difference between the two definitions. For example,

suppose the dataset comprises n novel tests and m non-novel tests with $n \ll m$. To build a model to filter in future novel tests that have the same properties of the existing novel tests, one can formulate it as a supervised learning problem (see e.g. [2]). Therefore, novelty detection can be approached by either unsupervised learning or supervised learning. Which approach to choose in Fig. 12 depends on the specific application.

D. The learning and feedback component

The early work in [26] formulated an implicant learning problem for the learning and feedback component in Fig. 4. Fig. 13 depicts the setting of the problem.

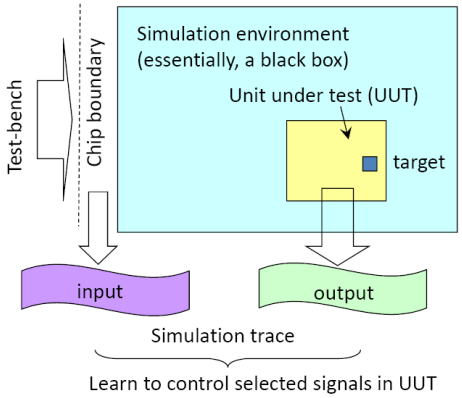

Fig. 13. Learning to enhance controllability of selected signals [26]

Suppose the input $\mathbf{X}$ comprises m vectors $\vec{x}_1, \ldots, \vec{x}_m$ where each $\vec{x}_i$ is a binary vector of length n. The output $\vec{y}$ comprises m binary values, corresponding to the m values of a particular signal when the m inputs are applied. Essentially, one can think of $(\mathbf{X}, \vec{y})$ as m samples drawn from an unknown Boolean function. The implicant learning problem is to uncover the implicants of this function based on the samples.

Suppose in verification one desires to hit an event. Hitting the event requires controlling three signals inside the unit under test (UUT). Then, this leads to three implicant learning problems. In general, the implicant learning problem can be thought of as learning the Boolean function itself. However, as studied in [24] learning a Boolean function works only when the function has very limited complexity. In fact, learning a Boolean function in general is a difficult problem [25].

The work in [26] applied the concepts in association rule mining [27] to implement a method for learning implicants. The work in [28] implemented a learning and feedback method in the context of unit-level verification. While these early works showed promises of applying automatic learning to improve signal controllability, the methods were not designed to work with assembly programs directly.

E. Two unanswered questions

The early works, while demonstrating promises of applying learning machine in functional test content optimization, did not explicitly address two important questions:

- What is the domain knowledge in the application context?
- How to incorporate the domain knowledge when the samples to analyze are assembly programs?

Answering these questions is essential for practical implementation of the proposed two data mining components.

IV. Implementing the Filtering Component

As discussed in Section II, learning space can be defined explicitly or implicitly and the definition can dictate the complexity of the learning. Therefore, it is intuitive to see that domain knowledge can be applied in two ways: (1) to influence the computation of the kernel function, or (2) to influence the feature selection. Fig. 14 illustrates these two approaches.

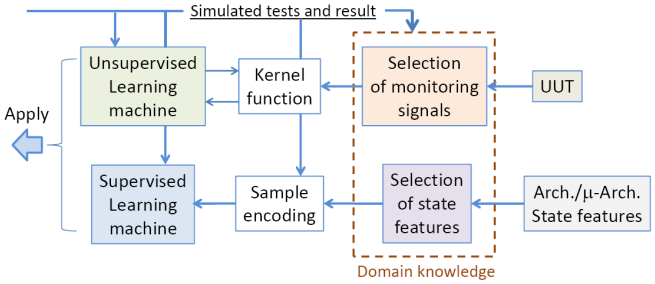

Fig. 14. Two ways to implement the filtering component

The work in [1] proposed to implement a kernel function based on a set of monitoring signals selected in the unit under test. Initially, this set is selected by the user. As the flow iterates, in each iteration the monitoring set is automatically adjusted based on the result from the simulated tests. The adjustment is based on the toggle coverage of the monitoring signals. For example, if a signal has a high coverage, it is removed from the monitoring set.

Given a monitoring set, the kernel function measures the similarity between two assembly programs based on their *estimated* coverage on the monitoring set. Because the kernel computation has to be very fast, the estimation cannot be based on simulation. In [1] the coverage estimation is based on a precomputed coverage database. The database consists of single-instruction samples and multi-instruction sequences, and their coverage on the signals of UUT. Then, coverage estimation of a given assembly program is based on matching segments of the assembly program to the stored single-instruction samples and multi-instruction sequences. Note that the precomputed coverage database can also be seen as the domain knowledge that captures how individual instructions or certain primitive instruction sequences are supposed to behave.

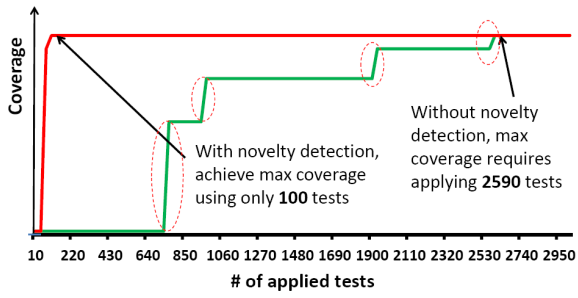

Fig. 15. Effect of the filtering component - Example 1

Fig. 15 and Fig. 16 show two examples to demonstrate the effectiveness of the filtering component. The experiments were

conducted on a commercial processor [1]. In Fig. 15, notice that in the original simulation without novelty detection (i.e. without the filtering), the coverage curve has a significant jumps in four places (circled with dotted red ovals). These jumps indicate that there were few tests much more important than others. With novelty detection these important tests were recognized as novel tests. As a result, 100 novel tests could achieve the same coverage of the original 2590 tests.

In Fig. 16, the actual saving in terms of the simulation time is shown. Note that this simulation time was based on a server farm. If the simulation was carried out on a single server, the difference would be around one week.

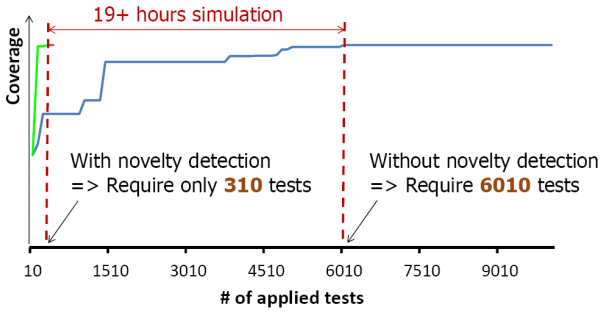

Fig. 16. Effect of the filtering component - Example 2

The work in [2] implemented a filtering component to look for tests that exercised the worst-case peak power. In the context of Fig. 14, the work follows the second approach based on feature selection and encoding. With respect to Fig. 12, the learning looked for absolute novel tests with certain properties. The properties are based on a set of architecture and/or micro-architecture state features selected by the user (as depicted in Fig. 14). The learning essentially tries to uncover the mapping between these high-level states and the low-level behavior observed. In [2], the low-level behavior to model is the worst-case peak power in the execution of an assembly program.

To see that the learning is about learning the mapping, consider two classes of tests T^+ and T. Suppose tests in T^+ cause worst-case peak power and tests in T do not. With selected state features, suppose tests in T^+ are converted into n positive samples, where each sample is a state vector. Each vector characterizes the hardware state during the cycle when the worst-case peak power is observed. Similarly, from the tests in T, m negative samples are derived. This conversion is done by the "sample encoding" step in Fig. 14.

Given the two sets of samples, a supervised learning machine is applied to model the differences between the positive samples and the negative samples, for example as a decision tree [2]. The model essentially characterizes a mapping function from the high-level hardware states to the low-level worst-case peak power events. Therefore, the essence of the learning is to uncover this mapping.

V. Implementing the Feedback Component

Fig. 17 depicts the learning and feedback component implemented for works [3][4]. The goal is to uncover the properties on a given set of important tests, with respect to a large number of unimportant tests (refer to the discussion with Fig. 4). The approach looks similar to the second approach shown in Fig. 14. However, there are some fundamental differences.

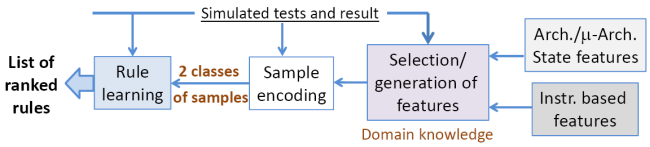

Fig. 17. Implementing the learning and feedback component

As suggested in [3], features are separated into two levels: state features and instruction-based features. When the analysis is carried out with state features, the learning tries to uncover important state feature combinations. Then, with the instruction-based features, the learning tries to uncover what instruction properties are likely to achieve those combinations.

In Fig. 17, features not only can be selected by the user but also can be "generated." Generation of a feature means implementing a function based on some selected features. For example, one may have two state features, A and B and use them to derive a new feature C=A+B.

The sample encoding is similar to that discussed above. Once the two classes of samples are derived, classification rule learning can be applied to uncover properties of the positive samples [3]. Because the emphasis is on the discovery of properties of the positive samples, the number of the positive samples is much more important than the number of the negative samples. As discussed in [4], as the number of negative samples grows beyond a point (e.g. 10K), adding more negative samples would not help the learning.

In Fig. 14, a learning model is applied to filter tests. In Fig. 17, the output of the rule learning is a ranked list of rules. This list is supposed to be examined by the user. Hence, it is up to the user to interpret a rule and decide on its usefulness.

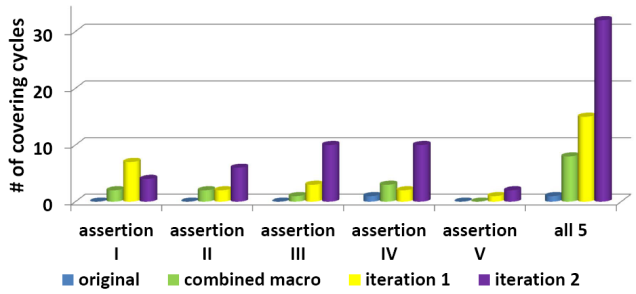

Fig. 18. Effect of the learning and feedback component - Example 1

Fig. 18 is an example to show the effectiveness of the learning and feedback component [3]. The experiments were conducted on a commercial processor verification environment. Originally, the tests based on the test template written by the verification team were not able to deliver good coverage on a family of five assertions - only assertion IV was hit once by 2000 tests generated. By learning the properties on the test hitting the assertion IV, the test template was modified and used to generate 100 more tests. This resulted in (labeled as "combined macro") coverage of assertions I to IV. With more positive samples available, the learning was carried out again. Test template was modified accordingly and 100 more tests were produced. With the 100 tests (labeled as "iteration 1") all five assertions

could be covered. The learning was applied again and the result (labeled as "iteration 2") showed further improved coverage.

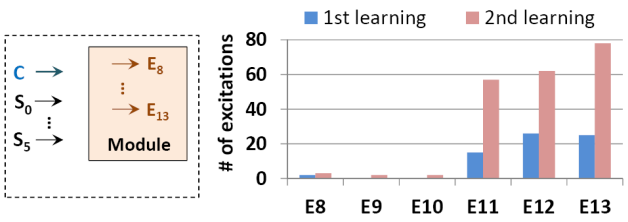

Fig. 19. Effect of the learning and feedback component - Example 2

Fig. 19 is an example to explain how the learning could be applied when the initial simulation had zero coverage on the events under consideration [4]. In this example, the events to cover were E_8 to E_{13}. Simulation of more than 30K tests generated from a given test template could not cover any of the six events. Because there was no test to cover any of the events, there was no positive sample to learn from.

For this example, domain knowledge was applied to recognize that signal values on C and S_0 to S_5 were highly related to the activation of the six E's events. Hence, learning was used to learn about the properties related to C and S_0 to S_5. Fig. 19 shows that after one iteration of learning, the modified test template could cover 4 out of the 6 events (with 1K new tests). By learning on the new positive tests, the modified test template could cover all 6 events with 100 tests [4].

VI. Final Remark

As explained in [7], data mining in EDA and test applications can often be treated as an iterative *knowledge discovery* (KD) process [29] to uncover *interpretable* and *actionable* knowledge. Fig. 20 illustrates a KD process.

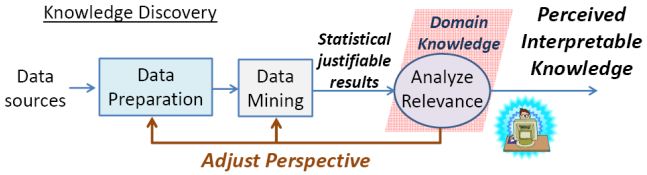

Fig. 20. Data mining as an iterative knowledge discovery (KD) process

Fig. 17 can be viewed as an example of the KD process in Fig. 20. In Fig. 17, the learning *perspective* is defined by the features in use. The data preparation is the sample encoding step. The data mining is the rule learning step. More importantly, the learning results are interpreted by a user and actions are pursued according to the interpretation. It is important to recognize that in order to produce interpretable results, the features themselves have to be interpretable to begin with. Interpretability is a crucial consideration in practice. This can apply to the features, the kernel functions, and the resulting learning models. In practice, if a user cannot make sense of a model, the model is often not applied, especially when application of the model has a non-neglectable cost. How to develop a data mining system to maximize the interpretability is a problem subjected to future research.

Acknowledgments

This work is supported in part by Semiconductor Research Corporation projects 2012-TJ-2268, 2013-TJ-2466, and by National Science Foundation Grant No. 1255818.

References

[1] Wen Chen, et. al. Novel Test Detection to Improve Simulation Efficiency — A Commercial Experiment. *ACM/IEEE ICCAD*, 2012.

[2] Vinayak Kamath, et. al., Functional Test Content Optimization for Peak-Power Validation — An Experimental Study. In *International Test Conference*, 2012.

[3] Wen Chen, et al., Simulation knowledge extraction and reuse in constrained random processor verification. In *ACM/IEEE Design Automation Conference*, 2013.

[4] Kou-Kai Hsieh, et al., On Application of Data Mining in Functional Debug. In *ICCAD*, 2014.

[5] V. Vapnik, The nature of Statistical Learning Theory. 2nd ed., *Springer*, 1999.

[6] Frank Rosenblatt. *Principles of Neurodynamics*. Washington, DC. Spartan Books, 1962.

[7] Li-C. Wang, Magdy Abadir. Data Mining In EDA - Basic Principles, Promises, and Constraints in *Design Automation Conference*, 2014.

[8] Olivier Bousquet, et. al. Introduction to Statistical Learning Theory. Springer *LNCS*, V 3176, 2004, pp. 169-207.

[9] Frank E. Grubbs. Procedures for Detecting Outlying Observations in Samples in *Technometrics*, v11, no 1, 1969, pp. 1-21.

[10] Trevor Hastie, et al. The Elements of Statistical Learning - Date Mining, Inference, and Prediction. *Springer Series in Statistics*, 2001

[11] Y. Katz and et al. Learning microarchitectural behaviors to improve stimuli generation quality. In *ACM/IEEE Design Automation Conference*, pages 848 –853, 2011.

[12] Olivier Chapelle, Bernhard Schlkopf and Alexander Zien. *Semi-Supervised Learning*. The MIT press, 2010.

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

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

[15] Nicholas Callegari, et. al. Classification rule learning using subgroup discovery of cross-domain attributes responsible for design-silicon mismatch. *DAC*, 2010, pp. 374-379.

[16] Janine Chen, et. al. Mining AC Delay Measurements for Understanding Speed-limiting Paths. In *IEEE ITC*, 2010.

[17] G. Batista. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *Sigkdd Explor.*, 6(1), pp. 20-29, 2004.

[18] Nik Sumikawa, et. al. Screening Customer Returns With Multivariate Test Analysis. In *IEEE ITC*, 2012.

[19] http://scikit-learn.org/stable/user_guide.html#user-guide http://scikit-learn.org/stable/modules/classes.html

[20] Sayan Mukherjee and Vladimir Vapnik. Support Vector Method for Multivariate Density Estimation. A.I. Memo No. 1653, C.B.C.L. Paper No. 170, MIT 1999.

[21] Jeff Tikkanen et. al., Yield Optimization Using Advanced Statistical Correlation Methods. In *IEEE ITC*, 2014.

[22] Onur Guzey, et. al., Functional Test Selection Based on Unsupervised Support Vector Analysis. in *Design Automation Conference*, 2008.

[23] Po-Hsien Chang, et. al., A Kernel-Based Approach for Functional Test Program Generation. In *International Test Conference*, 2010.

[24] Onur Guzey, et. al. Extracting a Simplified View of Design Functionality Based on Vector Simulation. Lecture Note in Computer Science, *LNCS*, Vol 4383, 2007, pp. 34-49.

[25] Michael J. Kearns and Umesh V. Vazirani. An Introduction to Computational Learning Theory, *MIT Press*, 1994.

[26] Onur Guzey, et. al., Enhancing signal controllability in functional testbenches through automatic constraint extraction. In *International Test Conference*, 2007.

[27] Chengqi Zhang and Shichao Zhang. Association Rule Mining, Models and Algorithms. *Lecture Notes in CS* Vol. 2307, Springer 2002.

[28] C.H.-P. Wen, et. al., An Incremental Learning Framework for Estimating Signal Controllability in Unit-Level Verification. In *ICCAD*, 2007.

[29] Krzysztof J. Cios, et. al., Data Mining - A Knowledge Discovery Approach, Springer, 2007.