Data Mining In EDA - Basic Principles, Promises, and Constraints *

Li-C. Wang
University of California at Santa Barbara

Magdy S. Abadir
Freescale Semiconductor

ABSTRACT

This paper discusses the basic principles of applying data mining in Electronic Design Automation. It begins by introducing several important concepts in statistical learning and summarizes different types of learning algorithms. Then, the experience of developing a practical data mining application is described, including promises that are demonstrated through positive results based on industrial settings and constraints explained in their respective application contexts.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Miscellaneous; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Design

Keywords

Computer-Aided Design, Data Mining, Test, Verification

1. INTRODUCTION

Electronic Design Automation (EDA) has become a major application area for data mining in recent years. In design and test processes, tremendous amounts of simulation and measurement data are generated and collected. These data present opportunities for applying data mining.

Many EDA problems have complexity that is NP-hard or beyond (e.g. #P). In theory, data mining does not make an NP-hard problem easier. For example, the power of "learning" is limited that "learning" a 3-term DNF formulae by itself is NP-hard [1]. This raises the fundamental question: If not for solving a difficult EDA problem, what problems is data mining good for in EDA?

*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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
DAC'14 June 01 - 05, 2014San Francisco, CA, USA.
Copyright 2014 ACM 978-1-4503-2730-5/14/06 ...$15.00
http://dx.doi.org/10.1145/2593069.2596675.

Before jumping into an answer, we should first consider a more fundamental question: What is learning? The NP-hard learnability result for 3-term DNF is based on the Probable Approximately Correct (PAC) learning model [2], which was intended to define the concept of *learnability* in supervised learning. In a PAC learning, the learning result is *guaranteed* by two parameters (1) $0 < \delta < \frac{1}{2}$: that the learning algorithm with $1-\delta$ probability will output the desired result (Probable) and (2) $0 < \epsilon < \frac{1}{2}$: that the desired result has an error bounded by ϵ (Approximately Correct).

In other words, if one desires to simultaneously guarantee the success rate and the result quality of the learning, then the learning problem can be hard.

In practice, one basic principle for avoiding this hardness can be to formulate a problem such that the simultaneous guarantee is not required. For example, the work in [3] shows that if one only seeks for good results without guarantee, learning a Boolean function with a high percentage of accuracy can be quite feasible.

While the computational learning theory addresses learning from the computational complexity perspective [1], the statistical learning theory developed by V. Vapnik [4] provides the necessary and sufficient conditions for a learning process to *asymptotically* guarantee its performance.

In almost all EDA applications, one has to assume that the data is somewhat limited. With limited data, it is likely that the data has not yet reflected the total complexity of the underlying behavior one seeks to model. In this case, the learning problem can be viewed as choosing the best model for the given data [1]. However, due to incomplete information, the best model may not be good enough.

To provide the missing information to a learning machine, domain knowledge is required. In fact, the learning theories tell us that some knowledge is always required for learning, i.e. "Learning = Data + Knowledge." The question is how much. In learning theories, one desires to use as little knowledge as possible - to make the theories as general as possible. In a practical EDA application, however, the question for a learning task is often about finding an optimal tradeoff between the need for the data and the need for the knowledge.

Data availability and knowledge availability are therefore key considerations that impact the formulation of a data mining application in EDA. Data availability concerns the information content of the data for the learning result to show some statistical significance. Almost all applications in EDA demand time-sensitive solutions. Hence, one may not have the time to wait for more data. In some cases, collecting more data can also be expensive or prohibited.

In an application, domain knowledge can be applied in: (1) formulating a learning task simple enough for the learning to be effective and (2) judging the quality of the learning result. The first relaxes the demand for more data and the second relaxes the need for guaranteed learning result.

For a data mining application in EDA to be useful, it has to provide added value to the existing tools and methodologies. This does not mean that a data mining approach has to out-perform an existing approach to be useful. This more often means that a data mining approach has a clear complementary value to the existing approach. This also often means that data mining is not used as a sole approach to solve a problem, but more as an approach to assist other tools or to assist an engineer to solve the problem.

Introduction of a data mining flow should make a target task easier for its user, not harder. For example, a user should not be spending more time and effort on preparing the data and interpreting the mining results than solving the problem using an existing flow. From this end, designing an effective usage model is crucial. This includes effective presentation of the mining results to facilitate user interaction and decision making.

Applying data mining to an EDA problem begins with a proper problem formulation. This often means developing a novel methodology such that data mining tools can be applied effectively. The problem formulation and methodology development determine what specific problems are to be solved by the data mining tools and hence, determine the overall effectiveness of the data mining approach.

In summary, a data mining methodology can be designed by considering several principles: (1) It does not always require guaranteed results from a data mining tool for the methodology to be useful and effective, (2) The required data is either readily available or the time and effort to collect the data are acceptable, (3) It provides added value to the existing tools and methodologies, and (4) It does not impose more engineering effort for solving the problem than that required without taking the data mining approach.

2. LEARNING ALGORITHMS

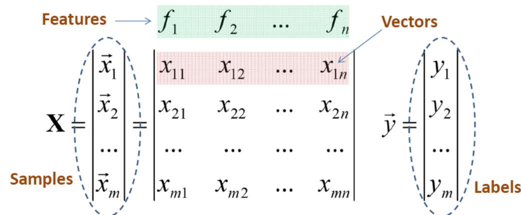

Figure 1: Dataset seen by a learning algorithm

Figure 1 illustrates a typical dataset seen by a learning algorithm. 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*.

A typical assumption to the x's values is that they are continuous values. If x's are binary, for example, then the learning is closer to that studied in computational learning [1] than that in statistical learning [4].

Note that in some learning problem, the y can be multivariate as well. Hence, instead of $\vec{y}$, the right hand side can be a matrix $\mathbf{Y}$. For example, the partial least square regression is designed for regression between two matrices. Canonical correlation analysis is a multivariate correlation analysis applied to a dataset of $\mathbf{X}$ and $\mathbf{Y}$ (see, e.g. [5]).

2.1 Basic ideas in learning

Take classification as an example. There can be four basic ideas to design a learning algorithm: (1) Nearest neighbor (2) Model estimation (3) Density estimation and (4) Bayesian inference.

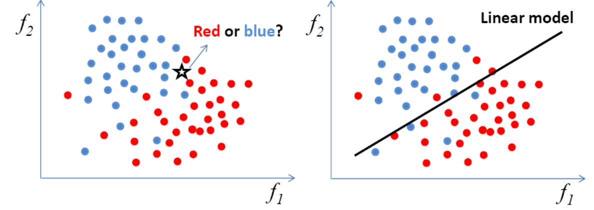

Figure 2: Nearest neighbor vs. model based

For example, Figure 2 depicts a simple classification problem in a two-dimensional space. The basic principle for nearest neighbor is that the category of a point (red or blue) can be inferred by the "majority" of data points surrounding it. Then, the trick is in how to define "majority" (see, e.g. [6]).

In a model based approach, one begins with assuming a model. For example, in binary classification one can assume a linear hyperplane to separate the two classes. A linear hyperplane in an n-dimensional space can be modeled as a linear equation with $n+1$ parameters. For example, in the figure the linear model can be modeled as $M(f_1, f_2) = w_1 f_1 + w_2 f_2 + b$ where w_1, w_2, b are parameters to be estimated based on the data.

The assumed model can be complex. For example, a neural network model may contain multiple hidden layers where the parameters are based on a collection of linear equations. The assumed model does not have to be an equation. For example, the model can be a tree [7], a collection of trees [8], or a collection of rules [9]. In a model based approach, the learning algorithm is specific to the underlying model structure it assumes.

The third basic idea is to estimate the probability distribution of a class. For example, for each class of data points shown in Figure 2, one can estimate its probability distribution as a two-dimensional normal distribution, i.e. the red samples with mean μ_1 and covariance Σ_1 as $\mathcal{N}(\mu_1, \Sigma_1)$ and the blue samples as $\mathcal{N}(\mu_2, \Sigma_2)$. Then, the decision function for a new sample x can be stated as:

$$D(x) = \log \frac{Prob(x \text{ based on } \mathcal{N}(\mu_1, \Sigma_1))}{Prob(x \text{ based on } \mathcal{N}(\mu_2, \Sigma_2))} \quad (1)$$

Equation 1 is the basic idea of discriminant analysis [6]. Of course, the probability density estimation can be more general than assuming a normal distribution (e.g. [11]).

The fourth idea is following the Bayes' rule. Let $\vec{x} = (x_1, \ldots, x_n)$ be an input sample to be predicted. The Bayes' rule states that:

$Prob(class|\vec{x}) = \frac{Prob(class)Prob(\vec{x}|class)}{Prob(\vec{x})} = \frac{prior \times likelihood}{evidence}$

Assume that sample occurrence is uniformly distributed. Then, $Prob(\vec{x})$ is a constant. $Prob(class)$ can be estimated by counting the number of samples in each class. Hence, the only term left to be estimated is the likelihood.

In naive Bayes classifier, it is assumed that all features are mutually independent. Hence, $Prob(\vec{x}|class)$=$Prob(x_1|class)$ $Prob(x_2|class) \cdots Prob(x_n|class)$. Each $Prob(x_i|class)$ is estimated using the f_i column of the dataset in Figure 1.

In practical application, the mutual independence assumption rarely holds. Hence, more sophisticated algorithms are designed to explore the mutual dependence among the features (see, e.g. [10]).

2.2 Learning space and kernel methods

A learning algorithm design can be based on more than one of the basic ideas discussed above. Nevertheless, there is one important issue that is not yet covered in the basic ideas - the space for carrying out the learning.

In Figure 2, the space is defined with two features f_1 and f_2. Typically, these are the input features provided with the dataset like Figure 1. However the fact that they are given as inputs does not mean that the space they define is necessarily good for applying a particular learning algorithm.

In kernel based learning (see, e.g. [11][12]), the learning algorithm and the learning space definition are separated. In the learning, a kernel function $k()$ is used which measures the *similarity* between any pair of samples $\vec{x}, \vec{x}'$ as $k(\vec{x}, \vec{x}')$. A kernel function *implicitly* defines the learning space. This is called the "kernel trick."

To see how the kernel trick works, consider the simple binary classification example shown in Figure 3. In the input space where the data samples are provided, the two classes of samples are not linearly separable. However, one can define a kernel as $k(\vec{x}, \vec{x}') = \langle \vec{x}, \vec{x}' \rangle^2$ where $\langle , \rangle$ denotes the dot-product of the two vectors.

Let Φ be a mapping function that for a sample $\vec{x}$, $\Phi(\vec{x} = (x_1, x_2)) \to (|x_1|^2, |x_2|^2, \sqrt{2}|x_1||x_2|)$. In other words, Φ maps a two-dimensional vector $\vec{x}$ into a new three-dimensional vector. We call the space defined by Φ the *feature space*.

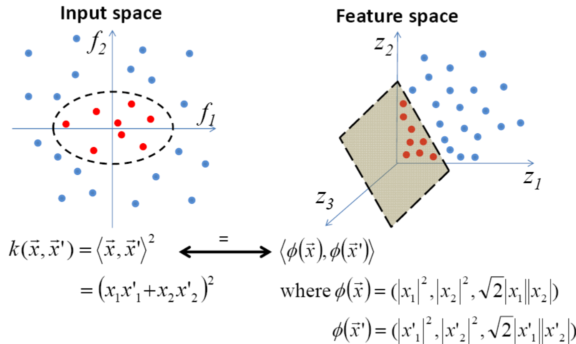

Figure 3: Illustration of kernel method

Figure 3 shows that while the two classes of samples are not linearly separable in the input space, they are in the feature space. In other words, if one desires to apply a learning algorithm that only assumes a linear model, one still can do that in the feature space and obtain a model that completely separates the two classes of samples.

The kernel trick is then based on two observations: (1) $k(\vec{x}, \vec{x}') = \langle \Phi(\vec{x}), \Phi(\vec{x}') \rangle$, i.e. the dot-product of two vectors in the feature space is the same as the kernel computation in the input space. (2) Learning a linear model in the feature space requires *only* the dot-product operator.

Based on the two observations, learning a linear model in the feature space does not have to be explicitly carried out with the mapping function Φ. All computations are based on the kernel $k()$. This is why with the kernel trick, the feature space exists only implicitly.

It is interesting to note that in kernel based learning, a learning algorithm (for the most part of its operation) no longer directly accesses the data matrix $\mathbf{X}$ shown in Figure 1. The information in $\mathbf{X}$ is accessed through the kernel function. This is illustrated in Figure 4.

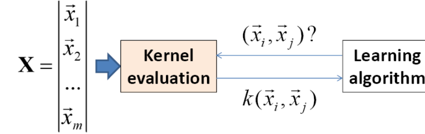

Figure 4: Kernel function vs. learning algorithm

A kernel based learning algorithm relies on the *relative information* provided by a kernel function to compute its learning model. Because $\mathbf{X}$ is not directly used, the samples do not have to be represented as vectors like that shown in Figure 1. As long as the kernel function $k()$ can be defined, the samples can be represented in any form.

Kernel based learning provides great flexibility to enable the application of data mining in EDA, especially when the data to be learned is not provided in matrix form like Figure 1. For example, in assessing the variability of a layout, each sample exists simply as a piece of layout image (see, e.g [13]). With a proper kernel, one does not need to explicitly convert a layout piece into a vector. As another example, in assessing the effectiveness of a functional test for processor verification, each sample (functional test) exists as an assembly program. To apply a learning algorithm to identify novel programs, one defines a kernel to measure the similarity between two assembly programs [14].

2.3 Overfitting, model complexity and SVM

In plain terms, *overfitting* is the situation where a learning model performs very well on the training data (data used to build the model) but not as well on the future data (or validation data - data not used in the learning).

In statistical learning theory [4], the concept of overfitting can be understood in view of the model complexity. This is depicted in Figure 5.

Consider the example in Figure 3 again. Suppose the learning algorithm is fixed and assumes a linear model. A linear model cannot separate the two classes in the input space. Hence, such a model would mis-classify many training samples. In the feature space, however, a linear model can perfectly separate the two classes, resulting in no error.

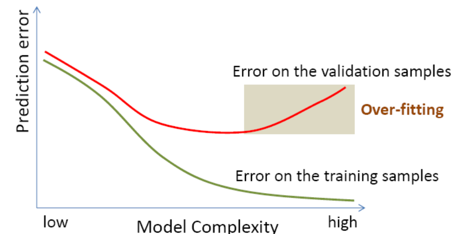

Figure 5: Overfitting in view of model complexity

The linear model in the feature space is more "complex" than the linear model in the input space because the feature space has a higher dimensionality and also each feature is more complex (to compute). From this simple example, we see that by increasing the *complexity* of a model, the error on the training samples can be reduced.

Figure 5 illustrates that by employing a more complex model, the training error can always be reduced, i.e. a more complex model fits the training data better.

The performance on the validation samples is different. At some point when the model complexity is too high, even though the training error can continue to improve, the validation error will start to increase. When this happens, the model has *overfitted* the training data.

Given Figure 5, there are two fundamental ideas to avoid the overfitting [6]. The first idea is to predefine a model structure with a limited complexity and then try to minimize the training error. A model based learning discussed in Section 2.1 like Neural Networks follows this basic idea. The second idea is to make no assumption to limit the model complexity (e.g. VC dimension [4]) and try to find the minimal complexity model to fit the data.

The popular Support Vector Machine (SVM) family of learning algorithms follow the second idea [6][11]. An SVM learning model is of the form:

$$M(\vec{x}) = [\sum_{i=1}^{m} \alpha_i k(\vec{x}, \vec{x}_i)] + b. \quad (2)$$

where $\vec{x}_1, \ldots, \vec{x}_m$ are the training samples. Each $k(\vec{x}, \vec{x}_i)$ is the similarity between the new input $\vec{x}$ (to be predicted) and the training sample $\vec{x}_i$. Each $\alpha_i \geq 0$ denotes the importance of the sample $\vec{x}_i$ in the computation of the model. The model $M()$ can be seen as a weighted average similarity between $\vec{x}$ and all training samples where the weights are determined by the α's. In SVM, the model complexity can be measured as $C = \sum_{i=1}^{m} \alpha_i$.

Let E denote the training error. An SVM algorithm tries to minimize the objective of the form $E + \lambda C$. This is called *regularization* and λ is a *regularization* constant [11]. Hence, in Figure 5 the overfitting is avoided by controlling the complexity of the model with λ. Regularization is not specific to SVM. In many modern learning algorithms, the regularization is applied to avoid overfitting [11].

2.4 Types of learning algorithms

SVM algorithm [11], tree based algorithms [7][8] and neural networks [6] are popular choices for classification problems. In practice, one may encounter the issue of imbalance dataset where there are much more samples from one class than from the other. Techniques were proposed to rebalance a dataset [15]. However, if the imbalance is quite extreme, rebalancing will not solve the problem. In those cases, it is no longer a typical classification problem.

For example, to learn a model to predict customer returns, one usually encounters a dataset where there are only a few customer returns and millions of passing parts [16]. Given an extremely imbalanced dataset, the problem becomes more like a feature selection problem [17][18] than a traditional classification problem.

For regression, there are many types of algorithms, including the straightforward nearest neighbor algorithm [6], the least square fit (LSF) [6], the regularized LSF [6], SVM regression (SVR) [11] and Gaussian Process (GP) [19]. For example, the work in [20] studied these five types of regression algorithms in the context of learning a model to predict the maximum frequency (Fmax) of a chip.

Clustering is among the most widely used unsupervised learning methods in data mining. Popular algorithms for clustering include, K-means, Affinity propagation, Mean-shift, Spectral clustering, Hierarchical clustering, DBSCAN, etc. (see, e.g. [21]). Clustering is easy to apply but the result may not be robust. The performance of a clustering algorithm largely depends on the definition of the learning space in which the samples are clustered.

Novelty detection is another widely applied unsupervised learning method. Novelty detection looks for *outliers* in a set of samples. The one-class SVM is a popular choice for novelty detection [11]. However, the performance of the method can largely depend on the kernel function in use.

Principal Component Analysis (PCA) [22] and Independent Component Analysis (ICA) [23] are popular data transformation methods. For example, PCA can be useful for reducing the dimensionality of a dataset by transforming a high-dimensional **X** matrix into a low-dimensional **X'** matrix. PCA explores correlations among the input features to extract *uncorrelated* new features called *principal components*. ICA is similar to PCA except that instead of looking for uncorrelated components, ICA looks for (statistically) independent components. Both PCA and ICA have found applications in test data analysis [24][25].

Classification rule learning such as the CN2-SD algorithm [9] is applied for supervised learning. A rule learning algorithm uncovers rules where each rule tries to model a subset of samples in a given class. Rule learning in unsupervised context is called *association rule mining* [26]. In those applications, an algorithm tries to uncover *frequent* patterns (represented as rules) in the dataset.

3. APPLICATION EXAMPLES

In a design process, the design evolves over time. Consequently, functional verification is an iterative process where extensive simulation is run on a few relatively stable design versions. In this context, data mining can be applied to reduce simulation time and improve coverage. For example, Figure 6 shows two places that data mining can be applied in a constrained random processor verification environment. Here a test is a sequence of instructions.

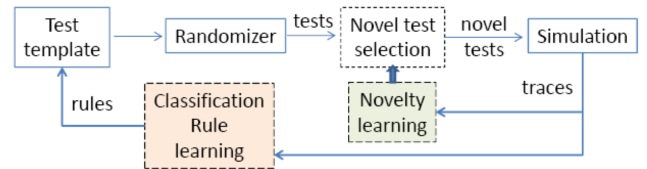

Figure 6: Two places to apply data mining

The work in [14] implemented the novel test selection idea proposed in [27] in a processor verification environment. The idea is to learn a novelty detection model based on the tests that have been simulated and use the model to filer out redundant tests coming from the constrained-random test generator ("randomizer"). The work in [14] uses the one-class SVM algorithm [11] for novelty detection model building. However, the real challenge in the implementation is not in the learning algorithm, but in developing a proper kernel evaluation software module [14].

Figure 7 shows a typical result observed in this application. Without the novel test selection, all tests coming out of the randomizer would be simulated. In this case, it took more than 6K tests to reach the maximum coverage for the unit under test (this was for the load-store unit; see [14]). With the novel test selection, only 310 tests would be simulated to reach the same coverage. The saving is 95%, or

19+ hours in server farm simulation (or multi-day if using only one server).

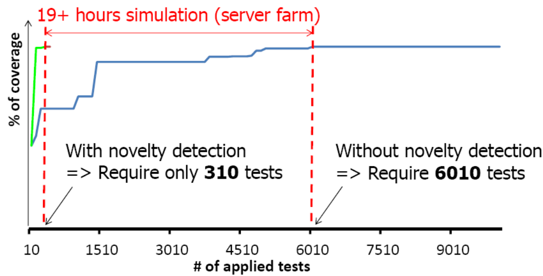

Figure 7: Simulation run time saving example

The work in [28] applied a rule learning methodology in the same processor verification environment. The idea is to learn the properties of a special test (e.g. a test hitting a coverage point of interest) and feedback those properties to the verification engineer for improving the test template.

Table 1: Coverage improvement after learning

Stage	# of tests	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7
Original	400	10	17	0	0	0	0	0	0
1st learning	100	3	11	10	10	4	2	1	1
2nd learning	50	72	59	71	83	79	97	96	87

Table 1 depicts a typical result. In this case, the original test template provided by the engineer was instantiated to 400 tests by the randomizer. On coverage points A_0 to A_7, only A_0 and A_1 received some coverage (# of cycles the coverage point was hit). Learning from the special tests hitting A_0 and A_1 resulted in rules used to improve the test template. The new test template was instantiated to 100 tests to achieve the coverage result shown in the row "1st learning." The new tests were added to the data for learning again. Then, the further improved test template was instantiated to 50 more tests. As we see in the row "2nd learning," all points were covered with high frequencies.

Figure 8 depicts the setup in [13] to apply data mining in layout variability prediction. Using lithography simulation as golden reference, the data comprises a set of good layout samples and a set of bad layout samples. The work applied both SVM binary classification and one-class SVM to learn from the data. The goal is to construct a model M that can be used for *fast* layout variability prediction. Similar to the work in [14], the real challenge in this implementation was developing an effective kernel. The work in [13] used the the Histogram Intersection (HI) kernel.

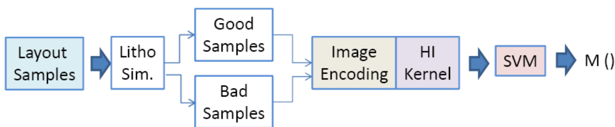

Figure 8: Data mining in litho. sim. context

Figure 9 shows a result to compare the prediction accuracy of the model M to the lithography simulation. Most of the high variability areas identified by the simulation were correctly identified by the learning model M.

Design-silicon timing correlation (DSTC) is another application area where data mining techniques were used [29][30]. In DSTC, the objective is to understand why the timing of a path observed on silicon is different from that predicted by a timer. The work in [31] applied a feature-based rule learning framework to analyze the speed-limiting paths that were not

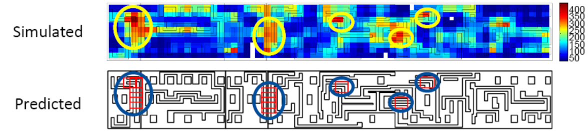

Figure 9: Fast prediction of variability

predicted by the timer as the top (12K) critical paths. It was shown that the learning approach could uncover design features causing the design-silicon mismatch [31].

Figure 10 shows a result of applying a similar data mining methodology. The left plot shows two clusters of paths: those whose silicon timing is faster than the predicted timing and those whose silicon timing is slower. These paths belong to the same design block and the mismatch result was totally unexpected. The right shows a learning rule uncovered by the methodology. This rule basically says that if the path contains a large number of layers-4-5 and layers-5-6 vias it would be a slow path. Later it was confirmed that the issue causing the slow paths occurred on metal layer 5.

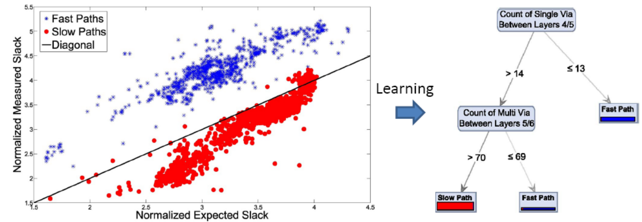

Figure 10: Diagnose unexpected timing paths

In test, large amounts of test data are available for mining. Test data mining has been an active research area for more than a decade. In a recent work [16], we applied data mining in predicting customer returns. For automotive products, the goal is to have zero customer return. Hence, whenever there is a return sent back from the customer, it is analyzed thoroughly to avoid similar returns from happening.

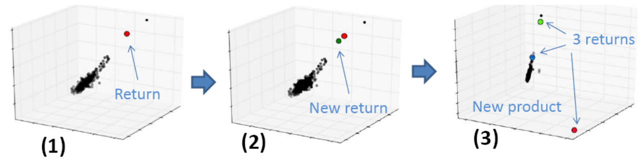

Figure 11: Modeling customer returns

Figure 11 shows a latest result based on the data mining methodologies suggested in [16][32]. Plot (1) shows how a return is learned and projected as an outlier in a 3-dimensional test space. Plot (2) shows how the outlier model, when applied, could have captured another return manufactured several months later. Plot (3) shows how the same model, when applied, could have identified three returns as outliers from a sister product line manufactured one year later.

4. DIFFICULT CASE FOR DATA MINING

In contrast to the promising results discussed above, Figure 12 depicts a scenario where data mining might not help. This is in the context of removing tests for test cost reduction [33]. In the two plots, our primary interest was to ask the question: could we drop test A and test B?

The left plot shows, based on test data from 1M chips, that all chips that failed test A were outside the test limit bounding box defined by tests 1 and 2 - i.e. all test A fails were

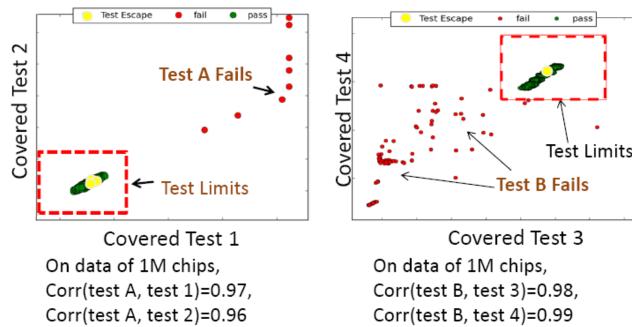

Figure 12: Difficult cases for data mining

also captured by test 1 or test 2. Moreover, the measured values of test A across the 1M chip are 0.97 correlated to those of test 1 and 0.96 correlated to those of test 2. Hence, based on the data of 1M chip, a data mining method would suggest to drop test A "safely." Similar situation applied to test B on the right. Both were reasonable results.

It turned out that in the next 0.5M chips, there were chips (yellow dots) that failed test A, but not test 1 nor test 2. Same situation occurred for test B. Therefore, if one demands to learn a model from the 1M chip data with a guarantee of, say, ≤ 1 test escape (escapes as the yellow dots) in the next 0.5M chips, the problem becomes very difficult.

5. KNOWLEDGE DISCOVERY

In practice, data mining is often used for *knowledge discovery* to uncover *interpretable* and/or *actionable* knowledge [34]. For knowledge discovery, the involvement of domain knowledge is almost always necessary. Further, the data mining process is *iterative*, where results from each iteration are (manually) evaluated to adjust the mining in the next iteration. In a data mining methodology, domain knowledge can be incorporated in two places: (1) In kernel based learning, the domain knowledge can be incorporated into the kernel module (e.g. [13][14]). (2) In feature-based learning, the domain knowledge is incorporated into the definition of the *features* (e.g. [31]). Our experiences show that the challenges in practical implementation are often related to the kernel or feature development, while choosing an existing learning algorithm to apply is relatively easy.

Our experiences also show that for practical success it is essential to develop a methodology to define mining problems where data mining techniques can be applied effectively. As illustrated in Section 4, if a problem formulation demands a stringent and guaranteed result, data mining might no longer be suitable for the application.

6. REFERENCES

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

[2] L. G. Valiant. A theory of learnable. *Communications of ACM*, 27 (11), pp. 1134-1142, 1984.

[3] 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.

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

[5] David R. Hardoon, Sandor Szedmak, John Shawe-Taylor. Canonical correlation analysis; An overview with application to learning methods *Neural Computation*, 16 (12), pp. 2639-2664, 2004.

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

[7] Leo Preiman, et al. Classification and Regression Trees. Wadsworth, 1984.

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

[9] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Rule induction for subgroup discovery with CN2-SD *Journal of Machine Learning Research*, 5:153–188, Dec. 2004.

[10] David MacKay, Information Theory, Inference, and Learning Algorithms. Cambridge Univ. Press, 2003.

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

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

[13] Dragoljub (Gagi) Drmanac, Frank Liu, Li-C. Wang. Predicting Variability in Nanoscale Lithography Processes. *ACM/IEEE DAC*, 2009, pp. 545-550.

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

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

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

[17] Nik Sumikawa, et. al. Important Test Selection For Screening Potential Customer Returns. *IEEE VLSI Design Automation and Test Symposium*, 2011, pp. 171-174.

[18] Z. Zheng, X. Wu, and R. Srihari. Feature Selection for Text Categorization on Imbalanced Data. *Sigkdd Explorations*, 6 (1), pp. 80-89, 2004.

[19] C. E. Rasmussen and C. K. I. Williams, Gaussian Processes for Machine Learning. MIT Press, 2006.

[20] Janine Chen, et al. Data learning techniques and methodology for Fmax prediction. *IEEE ITC*, 2009.

[21] http://scikit-learn.org/stable/modules/clustering.html

[22] I.T. Jolliffe, Principal Component Analysis. Springer, 1986.

[23] A. Hyvarinen, et. al. Independent Component Analysis. Wiley Series on Adaptive and Learning Systems, 2001

[24] Peter M. O'Neill. Production Multivariate Outlier Detection Using Principal Components. *IEEE International Test Conference*, 2008.

[25] Ritesh Turakhia, et al. Defect Screening Using Independent Component Analysis on IDDQ. *IEEE VLSI Test Symposium*, 2005, pp. 427-432.

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

[27] Onur Guzey, et al. Functional test selection based on unsupervised support vector analysis. *ACM/IEEE Design Automation Conference*, 2008, pp. 262-267.

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

[29] Li-C. Wang, Pouria Bastani, Magdy S. Abadir. Design-silicon timing correlation — a data mining perspective. In *ACM/IEEE DAC* 2007, pp. 384-389.

[30] P. Bastani, et. al. Statistical Diagnosis of Unmodeled Timing Effect. *ACM/IEEE DAC*, 2008, pp. 355-360.

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

[32] Nik Sumikawa, et al. A Pattern Mining Framework for Inter-Wafer Abnormality Analysis. *IEEE ITC*, 2013

[33] Dragoljub (Gagi) Drmanac, et al., Wafer Probe Test Cost Reduction of an RF/A Device by Automatic Testset Minimization: A Case Study. *IEEE ITC*, 2011.

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