Machine Learning Approaches for IC Manufacturing Yield Enhancement

Chapter · January 2019

DOI: 10.1007/978-3-030-04666-8_6

2 authors:

Hongge Chen
Massachusetts Institute of Technology
25 PUBLICATIONS **535** CITATIONS
SEE PROFILE

Duane Boning
Massachusetts Institute of Technology
324 PUBLICATIONS **4,425** CITATIONS
SEE PROFILE

Machine Learning Approaches for IC Manufacturing Yield Enhancement

Hongge Chen and Duane Boning

Abstract Modern semiconductor manufacturing is highly automated and generates a large amount of data with every wafer and process step. There is an increasing interest in machine learning and data mining techniques to improve yield to take advantage of this increasing volume of data. In this chapter, we introduce machine learning yield models for integrated circuit (IC) manufacturing yield enhancement. Challenges in this area include class imbalance due to high manufacturing yield, and concept drift due to the time varying environment. We present batch and online learning methods to tackle the imbalanced classification problem, and incremental machine learning frameworks to overcome concept drift. We consider the packaging and testing process in chip stack flash memory manufacturing as an application. By testing our methods on real data from industry, we show the possibility of packaged memory yield improvement with machine learning based classifiers detecting bad dies before packaging. Experimental results show significant yield improvement potential. In a time-invariant environment, for stacks of eight dies, we can achieve an approximately 9% yield improvement. In a longer period of time with realistic concept drift, a naive approach using a fixed false alarm rate-based decision boundary is shown to fail to improve yield. In contrast, with optimal thresholds, our incremental learning approach achieves approximately 1.4% yield improvement in the eight die stack case and 3.4% in the 16 die stack case.

Hongge Chen
Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, e-mail: chenhg@mit.edu

Duane Boning
Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, e-mail: boning@mtl.mit.edu

1 Introduction

Semiconductor manufacturing processes are highly automated with large amounts of data generated every single day. Due to the uncertainty in nanoscale fabrication and the growing complexity of the process, various machine learning and data mining techniques have been proposed to improve different steps of manufacturing [1][2][3]. In this chapter, we consider machine learning to predict and enhance the yield of integrated circuit (IC) manufacturing. More specifically, we want to predict the results of final device tests from early stage data. The results of the final test are assumed to be binary, for example, pass/fail or good/bad. Two main challenges, class imbalance and concept drift, in building machine learning models for integrated circuit yield enhancement are discussed and addressed in this chapter, based on extending the work presented in [4] and [5].

1.1 Challenge One: Imbalanced Classification

In high volume semiconductor manufacturing, high manufacturing yield is often achieved. As a result, the number of failure devices may only be a very small fraction of the sample set. The issue of extreme differences in prior class probabilities is called class imbalance [6]; building classifiers on highly imbalanced datasets is challenging. In scenarios we consider here, class imbalance arises because we have much smaller numbers of dies belonging to the "failure" class compared to those in the "good die" class. In this chapter, we discuss batch and online learning ensemble methods to learn on highly imbalanced datasets from IC manufacturing. Our approach will be presented in Sections 4.1 and 4.2.

1.2 Challenge Two: Concept Drift

The data from a manufacturing process is often available in a data stream manner. Due to subtle changes in the process, the underling probability distributions of the data are expected to change over time. This issue is called concept drift, with the unfortunate effect that learning models trained on old data may be inconsistent with new data [7]. To overcome this challenge, we present an incremental learning approach to learn in nonstationary environments. Our approach for incremental learning and continuous classifier updating is discussed in Section 4.3.

1.3 Application

As an application, our learning methods for class imbalance and concept drift are tested on a flash memory packaging process. There are two main reasons why a packaging process, the last step of semiconductor device fabrication, is a good and important object of study. First, packaging is almost the last chance for a semiconductor manufacturing company to improve its product quality before the final test. Second, at packaging, we have more input features, including unpackaged die electrical test results, for the learning model than at most other previous manufacturing stages. Opportunities for machine learning based yield improvement in packaging processes have not attracted much attention with few previous works reported. In this chapter, we build on our previous discussion in [4] and [5] to show that with a good machine learning model classifying the flash memory dies before packaging, we can improve the final part yield significantly. Our key ideas in this chapter are as follows:

1. Batch and online ensemble machine learning techniques are explained to address the class imbalance problem.
2. An incremental learning framework is designed to overcome the problem of concept drift.
3. We build a mathematical model and demonstrate the possibility of yield improvement using a classifier to detect bad dies before packaging.
4. The experimental results show the potential for significant improvement of yield using industrial data.

The rest of this chapter is organized as follows. Section 2 is an introduction to the flash memory manufacturing process. Section 3 provides the preliminaries of the problem, including evaluation metrics and mathematical formulation. In Section 4 the batch, online and incremental learning frameworks are discussed. Then the experimental results are presented in Section 5. Section 6 concludes this chapter.

2 Background of the Manufacturing Process

As illustrated in Figure 1, there are six high-level steps in the IC memory manufacturing process [8], Wafer Fabrication, Cherry Pick, KGD, Assembly (Packaging), Die Preparation and Memory Test. In flash memory chip manufacturing, wafers are fabricated in wafer fab facilities, after which dies on each wafer go through a standard electrical test process. Based on the results of these tests, wafers and dies are marked with different parameters and bins. The two main tests in the fabrication facility are SME1 and SME2. These two tests are basically the same type of tests at different temperature. Note that in order to reduce cost, SME1 and SME2 are performed in a sequential manner, meaning that dies that fail SME1 will not be tested in SME2. Dies that fail either SME1 or SME2 are marked on the wafer by the fab. Then the wafers, together with their data, are shipped to an assembly and

test facility. This facility can then bin or "cherry pick", by sorting the wafers into different quality tiers according to the test data from the fab. A further testing step called the Known Good Die (KGD) test is carried out. KGD is product-oriented: dies from wafers in different tiers undergo different KGD procedures. Those dies on the "prime" wafers identified in the cherry pick go through a strict KGD test and will be packaged into high-end products. Then the packaging process begins. After necessary preparations and processing steps, the wafers are diced into singular dies. Then wire bonding attaches the dies to the substrate, with other passive components mounted on. Importantly, multiple dies are typically stacked together. Then the device is encapsulated and separated [8]. After packaging, there is a final memory test which verifies the functionality of the product in the extreme environment of the real world. Though the dies are packaged into products, memory test also still assesses the performances of each die within the package. For each die, the output

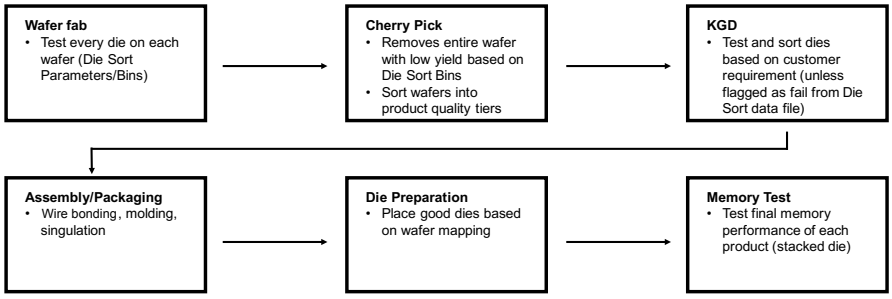


Fig. 1 Flash memory chip manufacturing process as discussed in [8].

of the memory test is pass (good) or fail (bad). Note that all dies selected for packaging have already passed earlier strict die electrical tests, but some of them still fail the high-performance final memory test after packaging. In other words, there exists "actually" bad or low-performance dies despite being judged by the earlier tests as good or high-performance, and with the limitations of the existing test decision process they can only be detected later by the final memory test. Because the dies are sequentially connected in the package, if one die in the package fails, the whole package is considered as fail. The company does not necessarily discard such "failed" packages, but rather downgrades them to low-end products with less profit. Since the total number of packages is fixed, to achieve high profit, the company desires more high-end products. Because a single fail die will lead to the failure of the whole package, good dies stacked in that package may be wasted. If we know that the failure probability of a single die is $p_{\text{die fail}}$, which is usually very small, the failure rate of the final packages will be

$$p_{\text{package fail}} = 1 - (1 - p_{\text{die fail}})^s \approx s p_{\text{die fail}}, \tag{1}$$

where s ($= 4$, 8 or even 16) is the number of dies in a package. Thus, the real failure rate of the product can be much higher than the failure rate of dies.

The conceptual goal of a machine learning based classifier is shown in Figure 2. Each die from wafer fab (shown at left in Figure 2) has a large number of features based on electrical test measurements. Based on these measurements, some dies are identified by the test program to be bad (e.g., red dies in Firegure 2), while the rest are assumed to be good dies and are sent to packaging, having passed earlier SME1 and SME2 electrical tests. So the packaging process proceeds on the assumption that all of these dies are good and will meet high performance specifications. But actually some of these dies are not good (pictured as orange dies in Figure 2), and presently these failures can only be detected by the final memory test. If we can build a classifier to predict the result of the final memory test based on the available die electrical test data from the fab, more good dies (shown as blue dies in Figure 2) will be stacked together for more high-end products.

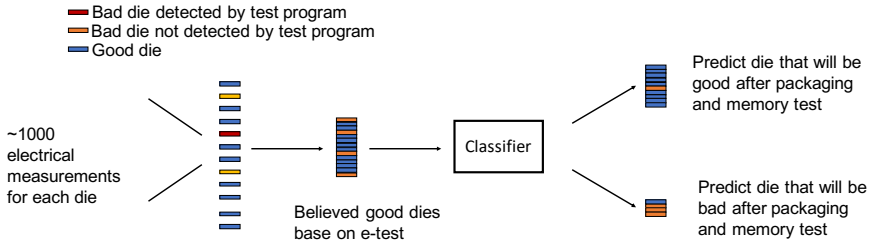


Fig. 2 Manufacturing process with a classifier.

3 Preliminaries

As mentioned in Section 1, a good classifier for memory test results before packaging will help to increase yield and profit for the following reasons.

- One can stack predicted good dies together for high-end products and package predicted lower performance ("bad") dies together for low-end products. If the classifier is good enough, the number of good packages will increase.
- Dies predicted as "bad" can go through a lower cost low-end packaging and test process directly and the cost of downgrading is saved.

For each die, all of its early test results and fabrication parameters can be used as the input of the classifier. In our data, approximately 1000 measured features (including results from SME1, SME2 and KGD) are available for each die. The output of the classifier is a prediction of the die's memory test result. Figures 3 and 4 illustrate the packaging and testing process without and with a classifier, respectively. In these two simple examples, 4 out of 16 incoming dies (all of which passed the fab die tests) are actually bad. If we have a fairly good classifier (even if imperfect as illustrated in Figure 4) to predict the memory test result and package dies predicted

as good together as high-end products and dies predicted as bad together as low-end products, our high-end yield can be improved from 25% to 50%. A similar issue also exists in the novel 3D integrated circuit packaging technology. If one chip in a stack fails, the whole package fails. In 3D packaging cases, where different types of chips or devices are stacked in a package, we would need classifiers for each kind of chip.

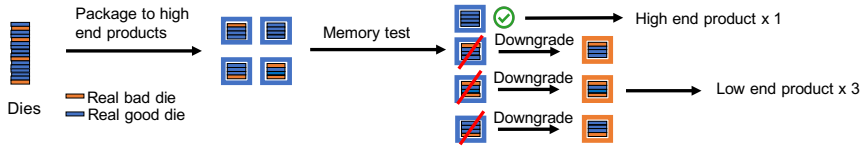


Fig. 3 The packaging and memory testing process without a classifier.

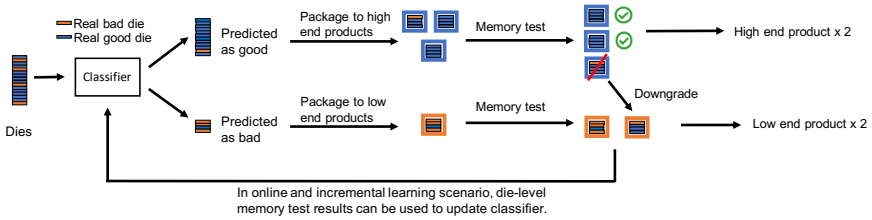


Fig. 4 The packaging and memory testing process with a classifier.

In Sections 3.1 and 3.2, we next introduce terms and evaluation metrics used in this work. Then the mathematical insight of the yield improvement and profit gain will be provided in Section 3.3.

3.1 Evaluation Metric: Confusion Matrix

Confusion matrices are perhaps the easiest and most widely used performance metric in classification problems. Suppose we have a classification problem with two possible classes; the confusion matrix is a 2×2 matrix visualizing the performance of a supervised learning method. Each column of a confusion matrix represents a possible predicted condition (output of the classifier) while each row of the matrix represents a possible true condition (ground truth). The confusion matrix and corresponding terminology with respect to a binary classification problem are shown in Table 1. In our problem, fail dies are labelled as 1 (positive or signaling a failure) in the final memory test and pass dies are labelled as 0 (negative). The confusion matrix is constructed based on the results of a classifier and known final memory

test on a test set of N dies. Each die in the test set must be placed into one cell in the matrix. Fail dies that are correctly predicted as fail are placed into the true positive (TP) cell, and those that are incorrectly predicted as pass are placed into the false negative (FN) cell. Good dies that are correctly predicted as pass are placed into the true negative (TN) cell, while those incorrectly predicted as fail are placed into the false positive (FP) cell. Then we sum up the number of dies in each cell and obtain the confusion matrix of the classifier. FN is also called Type II error and FP is also called Type I error.

Table 1 The confusion matrix and corresponding terminology with respect to a binary classification problem.

		Predicted condition	
		Predicted positive (fail)	Predicted negative (pass)
True condition	Condition positive (fail)	True positive (TP)	False negative (FN)
	Condition negative (pass)	False positive (FP)	True negative (TN)

Based on the confusion matrix, some terminology and derivations are developed. First, it is obvious that we have $N_{\text{fail}} = TP + FN$ true fail dies and $N_{\text{pass}} = FP + TN$ true pass dies, where total number of dies $N = N_{\text{fail}} + N_{\text{pass}}$. The classification accuracy is defined as

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} = \frac{TP+TN}{N}.$$

In our high manufacturing yield and highly imbalanced classification scenario, classification accuracy is in fact not a good metric since we can achieve fairly high accuracy by simply labeling all dies as pass. For example, if in our data $N_{\text{fail}} = 10$ and $N_{\text{pass}} = 990$ (which means the yield is 99%), then a trivial classifier predicting all dies as good can achieve 99% classification accuracy since $TN = 990$, $FN = 10$ and $TP = FP = 0$. However, this trivial classifier cannot give any useful information.

To overcome this, we instead need to consider both true and false positive rates. The true positive rate (TPR) is defined as

$$TPR = \frac{TP}{TP+FN},$$

which gives the fraction of positive instances that are correctly detected as positive. The false positive rate (FPR) is defined as

$$FPR = \frac{FP}{FP+TN},$$

which gives the fraction of negative instances that are sacrificed or lost to false alarms. In the aforementioned example, the trivial classifier has $TPR = FPR = 0$. If a classifier detects 8 out 10 bad dies while mistakenly labels 99 good dies as bad, we have $TPR = 0.8$ and $FPR = 0.1$. In general, a good classifier is expected to have both low FPR and high TPR.

3.2 Evaluation Metric: ROC Curves

The Receiver Operating Characteristic (ROC) curve is a graphic illustration of the performance of a binary classifier based on TPR and FPR. We denote "positive" or fail by 1, and "negative" or pass by 0.

For a given die, the vector of input features of our classifiers is denoted as $x \in X$, where X is the space of input features. The prediction result y belongs to Y, the set of all possible prediction results. In our binary classification, $Y = \{0,1\}$. A classifier $h(x,y)$ is assumed to output a non-negative real value, the score of an instance with features x to be classified in a class $y \in Y$. Moreover, we have

$$\sum_y h(x,y) = 1 \text{ and } h(x,y) \in [0,1]. \tag{2}$$

Then $h(x,y)$ can be interpreted as the probability of x to be classified into class y. To make a prediction, a threshold η is set to map the scores to $\{1,0\}$. For example, when $\eta = 0.6$, this die with feature vector x is predicted to be a member of class 1 if $h(x,1) > 0.6$ and to be a member of class 0 otherwise. Under each threshold, we thus have different TPR and FPR. The Receiver Operating Characteristic (ROC) curve is obtained by plotting TPR against FPR under different thresholds. Note that when $\eta = 1$, all instances will be classified as 0, thus $TPR = FPR = 0$, and when $\eta = 0$, $TPR = FPR = 1$. So an ROC curve must go through $(0,0)$ and $(1,1)$. Figure 5 shows different example ROC curves. An ideal ROC curve goes through $TPR = 1, FPR = 0$, which means that under some threshold, we can achieve 100% accuracy in classification with an ideal classifier. A classifier based purely on a random guess is a straight line, while a practical classifier is somewhere in between these two cases.

The area undeor an ROC curve (AUC) is a useful overall metric for classifier performances with all possible thresholds, as presented in Figure 5. For an ideal classifier, $AUC = 1$; for random guess, $AUC = 0.5$; and for a practical classifier, $AUC \in (0.5,\ 1)$.

3.3 Mathematical Formulation

With a large number of testing dies, we can estimate the underlying positive and negative probabilities by the relative frequency. Let H be the true condition and

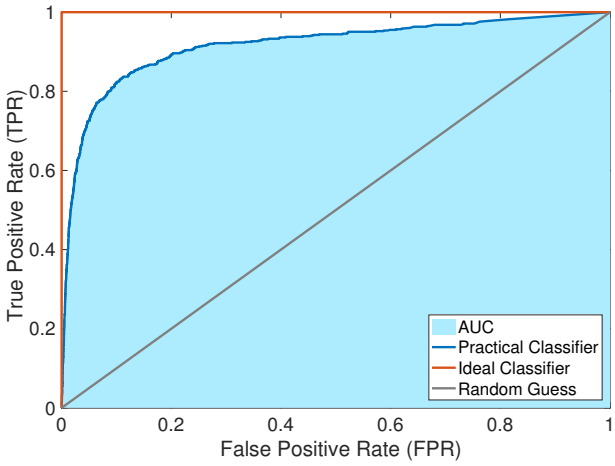


Fig. 5 Receiver Operating Characteristic (ROC) curves and the area under ROC curve, or *AUC*.

y be the predicted condition. For given input features x and threshold η, $y = 1$ if $h(x,y) > \eta$ and 0 otherwise. Then the joint distribution of true condition H and predicted condition y can be estimated by

$$\begin{aligned} p(H=1,y=1) &= \frac{TP}{N},\ p(H=1,y=0) = \frac{FN}{N} \\ p(H=0,y=1) &= \frac{FP}{N},\ p(H=0,y=0) = \frac{TN}{N}. \end{aligned} \tag{3}$$

Here $p(E)$ indicates the probability of event E. Without any classifiers, if we randomly package the dies into packages of s chips in each stack the failure rate of the packages is

$$p_{\text{package fail}} = 1 - p(H=0)^s. \tag{4}$$

Then if n dies are produced, we can produce n/s packages in total. The expected number of good packages is

$$\mathbb{E}[m_1(s)] = \frac{n}{s} p(H=0)^s, \tag{5}$$

where round-off error is neglected (number of leftover dies $< s$ not packaged). With a classifier to predict the memory test result before packaging, we can stack the dies predicted as fail together and the dies predicted as pass together. Recall that our "fail" classification still indicates that the die will work in low performance products, while "pass" dies are required for high performance products. Using the law of total expectation, the expected number of good (high performance) packages in this case is given by

$$\mathbb{E}[m_2(s)] = \mathbb{E}_k[\mathbb{E}[m_2(s)|k]] = \mathbb{E}[kp(H=0|y=0)^s] = \frac{n}{s}p(H=0|y=0)^s p(y=0), \tag{6}$$

where k is the number of stacks packaged as high end products and is subject to a binomial distribution $B(\frac{n}{s},\ p(y=0))$. Again, round-off error is neglected. If the classifier is ideal, all good dies are packaged into high-end products and the maximum number of high-end products is achieved,

$$\mathbb{E}[m_2^*(s)] = \frac{n}{s}p(y=0) = \frac{n}{s}p(H=0). \tag{7}$$

It is easy to prove that $\mathbb{E}[m_2^*] \geq \mathbb{E}[m_1]$ and $\mathbb{E}[m_2^*] \geq \mathbb{E}[m_2]$. The expected yield improvement is $\mathbb{E}[m_2(s)] - \mathbb{E}[m_1(s)]$ which can be either positive or negative. Note that TP, TN, FP and FN are uniquely determined by N_{pass}, N_{fail}, FPR and TPR. Subtracting (5) from (6), plugging in (3) and with a little manipulation, we have the expected yield improvement

$$\begin{aligned}
\frac{\mathbb{E}[m_2(s) - m_1(s)]}{n/s} &= \frac{p(H=0, y=0)^s}{p(y=0)^{s-1}} - p(H=0)^s \\
&= \frac{(p(H=0)p(y=0|H=0))^s}{[(p(H=0)p(y=0|H=0) + (p(H=1)p(y=0|H=1)]^{s-1}} - p(H=0)^s \\
&= \frac{[N_{\text{pass}}(1-FPR)]^s}{N[N_{\text{pass}}(1-FPR) + N_{\text{fail}}(1-TPR)]^{s-1}} - \frac{N_{\text{pass}}^s}{N^s},
\end{aligned} \tag{8}$$

which is a function of TPR and FPR. In this work's imbalanced classification scenario we have $N_{\text{fail}} \ll N_{\text{pass}}$. As shown in Figure 5, $TPR \geq FPR$ should hold, so we also have $N_{\text{fail}}(1-TPR) \ll N_{\text{pass}}(1-FPR)$. Then (8) can be linearized as

$$\begin{aligned}
\frac{\mathbb{E}[m_2(s) - m_1(s)]}{n/s} &= \frac{[N_{\text{pass}}(1-FPR)]^s}{N[N_{\text{pass}}(1-FPR) + N_{\text{fail}}(1-TPR)]^{s-1}} - \frac{N_{\text{pass}}^s}{N^s} \\
&\approx \frac{N_{\text{pass}}}{N}(1-FPR)[1-(s-1)\frac{N_{\text{fail}}(1-TPR)}{N_{\text{pass}}(1-FPR)}] - (1 - \frac{sN_{\text{fail}}}{N}) \\
&= (s-1)\frac{N_{\text{fail}}}{N}TPR - \frac{N_{\text{pass}}}{N}FPR \\
&= \frac{1}{N}[(s-1)TP - FP],
\end{aligned} \tag{9}$$

where we use the fact that $(1+x)^n \approx 1+nx$ when $|x| \ll 1$. This gives us an easy way to estimate the yield improvement. For example, when we have 4 dies in a package, the yield improvement will be positive if the number of bad dies correctly detected is at least $\frac{1}{3}$ of the number of good dies mistakenly predicted as bad.

Here we implicitly assume that the underlying joint probability distribution $p(H,\ y)$ does not change over time. If it does, as in Section 1.2 and Section 4.3, equation (8) from old data may not be valid for future prediction.

4 Learning Models

Here we describe the learning models used to overcome imbalanced classification and concept drift. The goal is to increase TPR and reduce FPR so as to maximize the expected yield improvement as in equation (8).

4.1 Imbalanced Classification and Batch RUSBoost Learning

The imbalanced classification problem is pervasive in many real world problems, especially when connected with anomaly detection [6][9], such as in Email fraud detection, medical diagnosis or computer intrusion detections. Since most conventional learning techniques assume fairly balanced distributions, their performances may be compromised on a highly imbalanced dataset. Class imbalance is a classical problem with some state-of-the-art solutions [10]. In our flash memory data, the fail dies only consist of around 2% of total dies and the problem is highly imbalanced.

A fundamental approach is to not make any changes in the learning algorithm, and instead to modify the dataset itself to provide a balanced training set for the classifier. The easiest and most intuitive ways to alleviate class imbalance are oversampling the minority class or undersampling the majority class. Though these seem to be equivalent in function, both oversampling and undersampling have problems that may be harmful to the performance of learning. For undersampling, reducing the number of the training instances means that some important patterns or information may be lost and the data becomes noisy. Oversampling, on the other hand may lead to increase in computation time and overfitting due to a large number of identical minority training instances. To tackle the problems of simple oversampling and undersampling, more sophisticated techniques are available.

Sampling techniques can be integrated with ensemble methods to interact with the models. Ensemble methods combine the results from multiple base classifiers (simple classifiers such as decision trees and naive Bayes classifiers) to achieve better performance. One of the most well-known ensemble methods to combine classifiers is Adaptive Boosting (AdaBoost) [11]. In Adaboost, the first base classifier is trained on the original data set. Then when training the subsequent base classifiers, instances misclassified by previous classifiers will be assigned larger weights. After training all the base classifiers, the ensemble classifier's final result is a weighted average of the base classifiers' output. The individual base classifiers may be very weak, but as long as the classification performances of each of them is slightly better than random guessing, the final ensemble classifier can be proven to converge to a strong classifier.

In our problem, RUSBoost is used to tackle the problem of imbalanced dataset in classification. Here RUS stands for "random undersampling" [12], which means removing instances in the majority class at random. RUSBoost is a widely-used hybrid approach embedding under-sampling in the AdaBoost framework. Before training each new base classifier, the instances in the majority class are randomly

undersampled without replacement as in step 4 of Algorithm 1. For example, if the desired class ratio for training is $50:50$, then instances in the majority class are randomly removed until the number of majority instances equals to the number of minority instances. The class ratio here is a hyper-parameter to be determined by cross validation. The main advantage of random undersampling is its simplicity. With multiple classifiers in an ensemble, the problem of noisy base classifiers can be alleviated. Random undersampling the majority class generates a new training set S_t' with weight distribution D_t', which is used to train base classifier h_t. Then the pseudo-loss on the original dataset S with weights D_t is calculated as in step 7. In step 9, we update the weights. Misclassified instances at t will have large weights at $t+1$, so the training of the base classifier at $t+1$ will concentrate more on these instances.

Algorithm 1 Batch RUSBoost [12]

1: **input**: S instances: $(x_1,y_1),(x_2,y_2),...,(x_m,y_m), y_i \in \{1,0\}$ with majority class $y_i = 0$ and minority class $y_i = 1$; base classfiers $h_1,...,h_T$
2: **initialize** $D_1(i) = \frac{1}{m}$ for any i
3: **for** $t = 1,2,...,T$ **do**
4: Create new training set S_t' by undersampling the majority class, the weight distribution of the remaining instances is D_t'
5: Train base classifier h_t with S_t' and D_t'
6: Test h_t on S, $h_t(x_i,y_i) \in [0,1]$
7: Calculate pseudo-loss on S and D_t: $\varepsilon_t = \sum_{i:y_i \neq y} D_t(i)(1 - h_t(x_i,y_i) + h_t(x_t,y))$.
8: $\alpha_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
9: Update $D_{t+1}(i) = D_t(i)\alpha_t^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_t,y))}$
10: Normalize $D_{t+1}(i) : D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_j D_{t+1}(j)}$
11: **end for**
12: **output**: $H(x,y) = \sum_{t=1}^{T} h_t(x,y) \log \frac{1}{\alpha_t}$ with normalization

4.2 Online RUSBoost Learning

Conventional batch machine learning models are trained on a fixed training set. In our problem, however, all dies, no matter predicted as pass or fail, go through the memory test, and thus we eventually have access to the true labels for all test instances. This enables us to further improve our classifier with new test data in an online approach, as shown in Figure 4. Due to the large number of dies produced each day, it is expensive to store all the old training data and retrain a new classifier with both old and new data when more data is available. Therefore we use an online learning model to update our existing model when the information about new dies arrives. A similar model updating technique has also been studied in the CAD community, such as for hotspot detection in [13].

Online machine learning is a vibrant sub area in machine learning research. Online learning methods have been proven to be very powerful in applications where training data becomes available with a sequential order. In contrast with traditional batch learning which needs to see the entire training data set before giving an output, online learning requires instant output after seeing each datum. The online classifier is built to output prediction p_t for input x_t at each time step t, $t = 1, 2, ..., T$. At time t, classifier receives x_t from "nature." In our work, "nature" refers to the early test data from SME1, SME2 or KGD. The classifier is asked for a prediction p_t for this x_t. Then at some later point the classifier receives the true label y_t. Comparing p_t and y_t, the classifier then updates its internal model.

The online learning model used in our work is an online version of RUSBoost proposed in [14], as shown in Algorithm 2. This Online RUSBoost is based on the online boosting framework proposed in [15]. The online versions of bagging and boosting approaches are discussed in [15]. The key idea of these approaches is to approximate the corresponding batch bagging and boosting version with the training dataset observed as a data stream. Mathematically, this approximation depends on the well-known limiting case of a binomial distribution (when the number of samples is very large and the probability of success is very small) approaching a Poisson distribution. Indeed, for a given data point, if we draw N independent samples with probability of success $\frac{\lambda}{N}$, the probability of having n copies is

$$p_n = \binom{N}{n}(\frac{\lambda}{N})^n(1-\frac{\lambda}{N})^{N-n} \rightarrow \frac{e^{-\lambda}\lambda^n}{n!} \tag{10}$$

when $N \rightarrow \infty$. The reason for this approximation is that in the online learning, we may not know the length of the whole data stream before observing all the data. But with this approximation, we do not need the number of data points N when it is very large. Then instead of changing the probability distribution on the samples as in batch RUSBoost, we simply change the λ values as in Algorithm 2. Experimental results have shown that the online learning can achieve comparable performance to the corresponding batch learning method [15].

4.3 Incremental Learning for Concept Drift and Class Imbalance

The classical set up of a batch machine learning problem is based on learning from data drawn from an unknown but fixed distribution. However, problems in the real world are usually more complicated, especially with parameters in the learning scenario involving time. The data we receive may not be generated at the same time, and a critical issue is that the underling probability distribution of the training data may change with time [7]. The issue of concept drift in the statistics community refers to the unexpected or unknown changes in the statistical properties of the dataset used to train and test the model. The most obvious problem caused by concept drift is that the model built on previous training data may lose its accuracy over time, as

Algorithm 2 Online RUSBoost [14]

input: Data stream: $(x_1,y_1),(x_2,y_2),...,(x_m,y_m),y_i \in \{1,0\}$ with majority class $y_i = 0$ and minority class $y_i = 1$; sampling rate C
Base classifiers $h_1,...,h_K$, $\lambda_k^{SC} = \lambda_k^{SW} = 0$ for all k.
Initialize $\lambda_k^{SC} = \lambda_k^{SW} = \lambda_k^{POS} = \lambda_k^{NEG} = 0$ for all k, $n^+ = n^- = 0$
when (x_i,y_i) **arrives**:
Reset $\lambda = 1$
for $k = 1,2,...,K$ **do**
    if $y_i = 1$ **then**
        $\lambda_k^{POS} \leftarrow \lambda_k^{POS} + \lambda, n^+ = n^+ + 1$
        $\lambda_k^{RUS} \leftarrow \lambda \frac{n^+}{n^+ + n^-} / \frac{\lambda_k^{POS}}{\lambda_k^{POS} + \lambda_k^{NEG}}$
    else
        $\lambda_k^{NEG} \leftarrow \lambda_k^{NEG} + \lambda, n^- = n^- + 1$
        $\lambda_k^{RUS} \leftarrow \lambda \frac{Cn^+}{n^+ + n^-} / \frac{\lambda_k^{NEG}}{\lambda_k^{POS} + \lambda_k^{NEG}}$
    end if
    Generate random variable a_i according to $Poisson(\lambda^{RUS})$
    Do a_i times: train base classifier h_k with (x_i,y_i)
    if $h_k(x_i) = y_i$ **then**
        $\lambda_k^{SC} \leftarrow \lambda_k^{SC} + \lambda$
        $\varepsilon_k \leftarrow \frac{\lambda_k^{SW}}{\lambda_k^{SW} + \lambda_k^{SC}}$
        $\lambda \leftarrow \frac{\lambda}{2(1-\varepsilon_k)}$
    else
        $\lambda_k^{SW} \leftarrow \lambda_k^{SW} + \lambda$
        $\varepsilon_k \leftarrow \frac{\lambda_k^{SW}}{\lambda_k^{SW} + \lambda_k^{SC}}$
        $\lambda \leftarrow \frac{\lambda}{2\varepsilon_k}$
    end if
end for
output: $H(x) = \arg\max_y \sum_{t=1}^{T} \mathbf{1}_{\{h_k(x)=y\}} \log \frac{1-\varepsilon_k}{\varepsilon_t}$

illustrated in Figure 6. In this figure, the blue crosses and circles are data points at t_1 while the red crosses and circles are data points at t_2. The blue dashed line is the decision boundary at t_1, which successfully classifies most blue data points at t_1. However, as the distribution of the data changes in t_2 (concept drift), if we still use the blue dashed line as the decision boundary, red data points in the right upper corner will be misclassified. In fact, at t_2, our decision boundary should be changed to the red dashed line. In the data stream from a semiconductor manufacturing process, concept drift also exists. One obvious reason is inherent change in the manufacturing process, due to continued improvement, equipment aging, or environmental changes in the fabrication, in assembly or in the testing process.

It is believed that dealing with concept drift and class imbalance simultaneously is an inherently difficult problem [16]. Due to the scarcity of fail dies (instances from minority class), the time interval between two consecutive fail dies can be large. Substantial changes in the manufacturing process may happen in this interval and the underlying distribution of these two fail dies' parameters may not be identical.

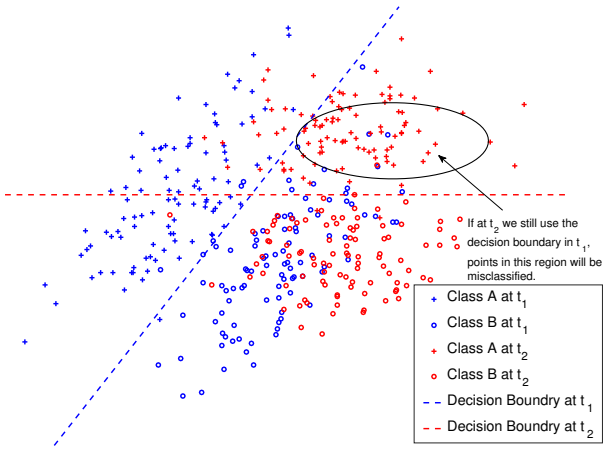


Fig. 6 Example of concept drift at two time steps t_1 and t_2. If at t_2 we still use the decision boundary at t_1, points in the ellipse will be misclassified.

With this problem formulation, Online RUSBoost in Algorithm 2 is not the best choice to handle a non-stationary environment since it is designed to approximate the batch learning algorithm.

Intuitively, in an evolving environment, learning models should only be trained on the most "recent" data and should only be used to predict the near future. However, as the concept drift can be very fast, the qualified recent data may be inadequate to train a good classifier due to overfitting. This problem is exaggerated in the class imbalance scenario where the qualified recent minority instances are even fewer. To learn in a non-stationary environment, the idea of incremental learning has been proposed. In contrast to the online learning scenario, where the information from old instances is discarded after training, incremental learning emphasizes extending the model's knowledge by leveraging new data while preserving information from old data. To implement this idea, we store only the classifiers trained on old data (since storing all old data is expensive) and invoke those classifiers together with the new classifiers to help us to make predictions in the future. While a new classifier trained only on recent data may be incapable of making a good prediction, old classifiers trained on a similar pattern may participate in decision making and improve performance.

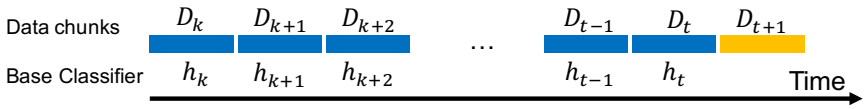


Fig. 7 The idea of incremental learning is that we divide our data stream into data chunks. Then we train base classifiers on each data chunk and combine them to predict dies in the new data chunk.

In this area, the Learn++.NIE [17] algorithm is a state-of-the-art ensemble method. Learn++.NIE assumes there is a data stream of chunks (mini batches). For example, in our case, each data chunk can be dies produced on one day. At each time step t, a new chunk $\mathscr{D}_t = \{(x_1,\ y_1),\ (x_2,\ y_2),\ ...,(x_{m^{(t)}},\ y_{m^{(t)}})\}$ arrives. A new base classifier h_t is trained on $\mathscr{D}_t$ in a batch manner, as shown in Figure 7. The final ensemble combines the base classifiers by calculating a weighted average of each base classifier's output. The weight of the base classifier trained at time step $k \leq t$ is determined by that base classifier's overall performance on $\mathscr{D}_k,\ \mathscr{D}_{k+1}, ...,\ \mathscr{D}_t$. Then a sigmoid error weighting is used to emphasize the performance on the recent data chunks as shown in Figure 8. The sigmoid values a and b are hyper-parameters to be tuned by cross validation, where a controls the slope and b controls the halfway crossing point. A detailed introduction of the Learn++.NIE is available in [17].

Algorithm 3 Modified Learn++.NIE

1: **input**: Data chunks: $\mathscr{D}_t = \{(x_i,\ y_i)\},\ i = 1,\ 2, ...,\ m^{(t)}$ with majority class $y_i = 0$ and minority class $y_i = 1$; Positive sigmoid parameters a and b.
2: **for** $t = 1, 2, ...$ **do**
3: **when** $\mathscr{D}_t$ **arrives**:
4: Call RUSBoost on $\mathscr{D}_t$ to generate a new base classifier $h_t(\cdot,\ \cdot) : X \times Y \rightarrow [0,1]$
5: For $k \in \{1,\ 2,\ ...,\ t\}$, evaluate $\text{auc}_k^{(t)}$, area under ROC curve obtained by h_k on $\mathscr{D}_t$.
6: For $k \in \{1,\ 2,\ ...,\ t\}$, calculate pseudo-loss $\varepsilon_k^{(t)} = 1 - \text{auc}_k^{(t)}$.
7: **if** $\varepsilon_t^{(t)} > 0.5$ **then**
8: Discard h_t and train a new one as in line 4 and redo lines 5 and 6.
9: **end if**
10: **if** $\varepsilon_k^{(t)} > 0.5$ **and** $k < t$ **then**
11: Set $\varepsilon_k^{(t)} = 0.5$
12: **end if**
13: $\beta_k^{(t)} = \varepsilon_k^{(t)} / (1 - \varepsilon_k^{(t)})$
14: For $k \in \{1,\ 2,\ ...,\ t\}$, calculate weights:
Define: $\omega_k^{(t)} = [1 + \exp(-a(t-k-b))]^{-1}$
Normalize: $\hat{\omega}_k^{(t)} = \omega_k^{(t)} / \sum_{j=0}^{t-k} \omega_k^{(t-j)}$
$\hat{\beta}_k^{(t)} = \sum_{j=0}^{t-k} \hat{\omega}_k^{(t-j)} \beta_k^{(t-j)}$
15: $W_k^{(t)} = \log \frac{1}{\hat{\beta}_k^{(t)}}$
16: **end for**
17: **output**: $H^{(t)}(x,y) = \sum_{k=1}^{t} W_k^{(t)} h_k(x,y)$ with normalization

In order to make it more suitable to our problem, we discuss a modified version of Learn++.NIE in this chapter, as expressed in Algorithm 3. Key ideas in this approach include the following:

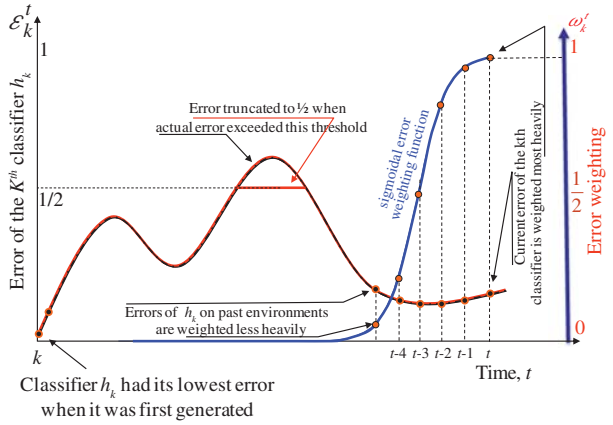


Fig. 8 An illustration of the sigmoidal error weighting in [18]. Learn++.NIE in [17] uses a similar error weighting.

- Instead of undersampling bagging as in [17], we use RUSBoost as our base classifier. Note that each RUSBoost base classifier is again an ensemble model with its own base classifiers.
- Since the output of the base classifier is a probability, we use $\text{auc}_k^{(t)}$, the area under ROC curve, to evaluate the performance of the base classifier. The pseudo-loss is obtained by $\varepsilon_k^{(t)} = 1 - \text{auc}_k^{(t)} \in [0,\ 1]$.
- The final output of the ensemble is a weighted average of the probability from each base classifier.

5 Experimental Results

The proposed batch, online and incremental learning methods are implemented and applied to memory die test data from our industrial partner. We choose a decision tree [19] as the base classifier in this work due to its simplicity, minimal assumptions required and potential robustness against class imbalance with sampling [20]. One RUSBoost ensemble model consists of 40 decision tree base classifiers. In Sections 5.1 and 5.2, as the batch and online learning methods in this chapter are not designed for concept drift, we use the data of around 100k dies (2.5% actual failure rate) produced in a short period of time, 8 days, to implement the algorithms. We assume that no major change in the manufacturing process takes place in this period and thus the concept drift in the data stream over this interval is negligible. In Section 5.3, the data of dies produced in a larger time span, one month, is used (around 370k dies in total), and the modified Learn++.NIE is used to handle concept drift and class imbalance simultaneously.

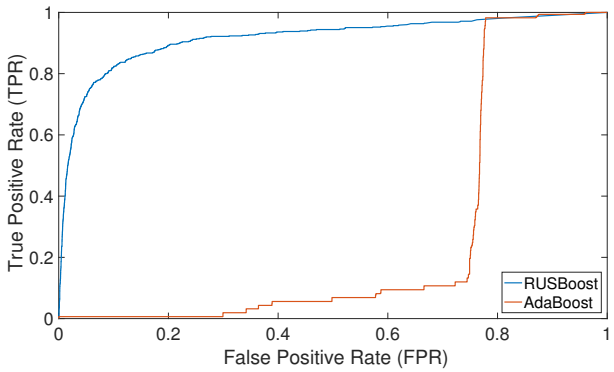


Fig. 9 The ROC curves of RUSBoost and AdaBoost on an imbalanced memory test dataset.

5.1 RUSBoost on Imbalanced Dataset

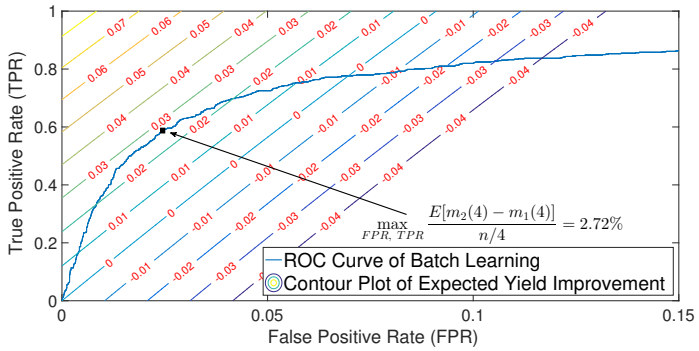
This section examines the performance of RUSBoost on the imbalanced memory test dataset. Our first experiment is to show the effectiveness of RUSBoost on an extremely imbalanced memory test dataset. We use 40 decision trees in an ensemble and the learning rate in fitting is set as 0.15. Figure 9 shows typical ROC curves of RUSBoost and AdaBoost on an imbalanced memory test dataset, AdaBoost largely fails on the data set while RUSBoost maintains high performance. Since AdaBoost tends to classify nearly all dies as pass, its TPR is very low. For example, typical confusion matrices by setting threshold to 0.5 are shown in Tables 2 and 3. As expected, AdaBoost tends to label nearly all dies as pass to achieve both simplicity and high accuracy. Then only 0.2% (1 out of 634) of the real bad dies are detected by the classifier and the economic benefit generated is negligible, or even negative. In contrast, RUSBoost identifies 523 of 634 bad dies, with significant potential economic saving as a result.

Table 2 A typical confusion matrix using AdaBoost, threshold 0.5.

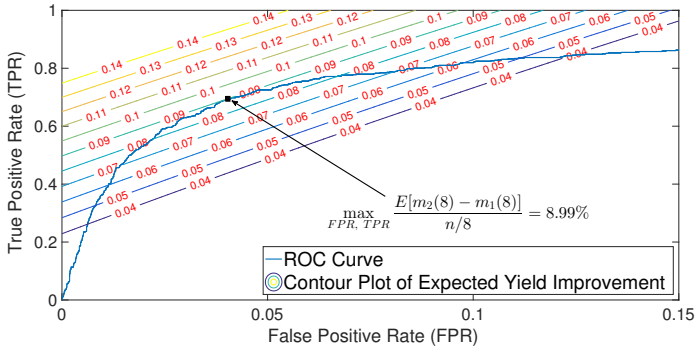
	Predicted as fail	Predicted as pass	
True fail	1	633	$TPR = 0.2\%$
True pass	6	29360	$FPR = 0.0\%$

Table 3 A typical confusion matrix using RUSBoost, threshold 0.5.

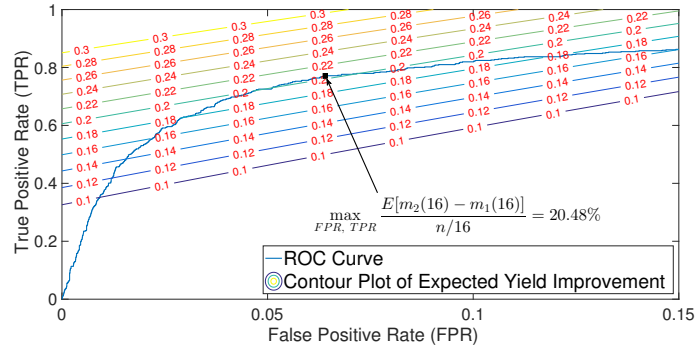
	Predicted as fail	Predicted as pass	
True fail	523	111	$TPR = 82.5\%$
True pass	2518	26848	$FPR = 8.8\%$



$s = 4$, yield without a classifier is 88.06%, yield with a classifier is 90.78%, $\eta^* = 0.75$.



$s = 8$, yield without a classifier is 77.54%, yield with a classifier is 86.53%, $\eta^* = 0.68$.



$s = 16$, yield without a classifier is 60.12%, yield with a classifier is 80.60%, $\eta^* = 0.60$.

Fig. 10 Contour plots of yield improvement $\frac{\mathbb{E}[m_2(s)-m_1(s)]}{n/s}$ with number of dies in a package $s = 4$, $s = 8$ and $s = 16$. The yield improvement is optimized under the constraint of ROC curve that is determined by the classifier. With more dies in a package, we can potentially achieve larger yield improvement.

Recalling equation (8), the expected packaged memory yield improvement is a function of TPR and FPR. However, FPR and TPR are constrained by the ROC curve of the classifier. Then the optimal point $(FPR^*,\ TPR^*)$ is where the contour plot of expected yield improvement is tangent to the ROC curve and the corresponding optimal threshold η^* is determined for future prediction. Figure 10 gives the ROC curve and contour plots of expected yield improvement $\frac{\mathbb{E}[m_2(s)-m_1(s)]}{n/s}$ with different s (number of dies in a package) for the data in our application. As expected, with larger s, we can achieve a larger yield improvement with more dies in a package. When $s = 16$, the maximum yield improvement is as large as 20.48% for this case of no concept drift, over an 8 day period. With smaller s, the yield without a classifier itself is very high, so even 1% of improvement is difficult, though still economically valuable. Also, the optimal threshold η^* decreases as s increases, which means more dies will be classified as fail. Intuitively, as s increases, the cost of mistakenly labeling a real bad die as a good die increases, since more good dies in its package will be wasted. Then the optimal classifier tends to decrease the threshold to detect more real bad dies. Additionally, note that the contour plots of the expected yield improvement using (8) are approximately straight lines, which proves that the linearization in (9) is valid in our case.

5.2 The Effectiveness of Online Learning

With the same data set, once we know the memory test result of a die in the test set, it becomes our new training instance for updating the online learning classifier in Algorithm 2. As online boosting methods are designed to approximate the batch learning asymptotically, each die in the test set is predicted by a classifier trained on the training set and all the test instances before it. As shown in Figure 11, all base classifiers of both batch and online models are trained on the training set. In the batch learning scenario, these base classifiers are combined to test the dies in the test set without any further training. In the online learning scenario, however, the dies in the test set are received sequentially. The test results are then used to update the online classifiers to achieve higher prediction accuracy. The online classifier is tested on the same dataset we use in Section 5.1. As Figure 12 indicates, online learning could enable as much as an extra 4.5% of expected yield improvement compared to the batch classifier when $s = 4$, primarily by a much improved classifier and corresponding ROC curve, when there is no concept drift present.

5.3 Incremental Learning with Concept Drift

Finally, we consider concept drift. We first show that concept drift is indeed present in our data within relevant time intervals. We then demonstrate the effectiveness of our incremental learning approach with this concept drift.

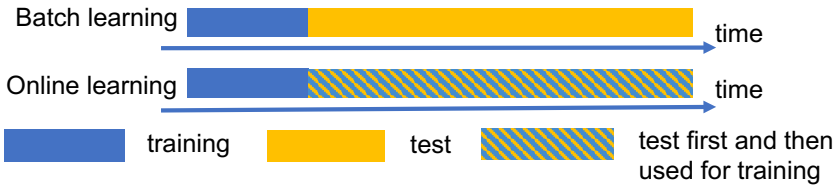


Fig. 11 The batch and online learning scenarios. All base classifiers of both batch and online models are trained on the training set. In the batch learning scenario, these base classifiers are combined to test the dies in the test set without any further training. In the online learning scenario, however, the test results are then used to update the model.

We consider the following experiment to prove the existence of concept drift. In this experiment, our larger data stream (about 370k dies) is divided into five sequential chunks, each covering approximately one week. Half of the third chunk is randomly selected as the training set. The rest of the dies in chunk 3 are used as test set 3. Then the classifier is tested on all five test sets. Figure 13 shows the ROC curves obtained on the five test sets. Clearly, the classifier's performance on test sets 1, 2, 4 and 5 are worse than on test set 3, especially in the low false alarm region that we care about most. This phenomenon indicates the existence of concept drift, where the earlier time periods (1 and 2) and the later time periods (4 and 5) have different underlying statistical structures than period 3 used to build the classifier; indeed, the degradation in the ROC is quite large in our data. Figure 14 shows the degradation of *AUC* value over time. A RUSBoost classifier is trained with data from week 0 and tested on data from week 1, 2, ... , 9. If we choose 0.85 as the threshold of major concept drift, only data within two weeks may be used.

As mentioned in Section 3.3, in a drifting environment, expected yield improvement $\frac{\mathbb{E}(m_2-m_1)}{n/s}$ in the future can no longer be estimated by old data. The choice of the optimal threshold in classification then requires expert advice or evaluation of cost tradeoff [8]. Here we assess the overall performance of the classifiers by their *AUC* values over time. We compare the *AUC* value of our modified Learn++.NIE ensemble (incremental learning) against a single RUSBoost classifier trained only on the most recent data chunk. Note that instead of using decision trees, the incremental learning takes RUSBoost classifiers trained on each data chunk as base classifiers, since data chunks are imbalanced. We divide our 370k dataset from one month into 14 time steps with two to three days in a time step. As the number of dies from each day may differ, we make sure that in each data chunk we have approximately the same number of dies. Then we test the incremental learning method and single classifier on this data stream. This procedure is repeated 30 times, and the average *AUC* results are shown in Figure 15.

With old base classifiers stored, incremental learning outperforms a single RUSBoost classifier on the whole. The average *AUC* value with a 95% confidence interval of the single classifier is 0.79 ± 0.002 while that of the incremental learning

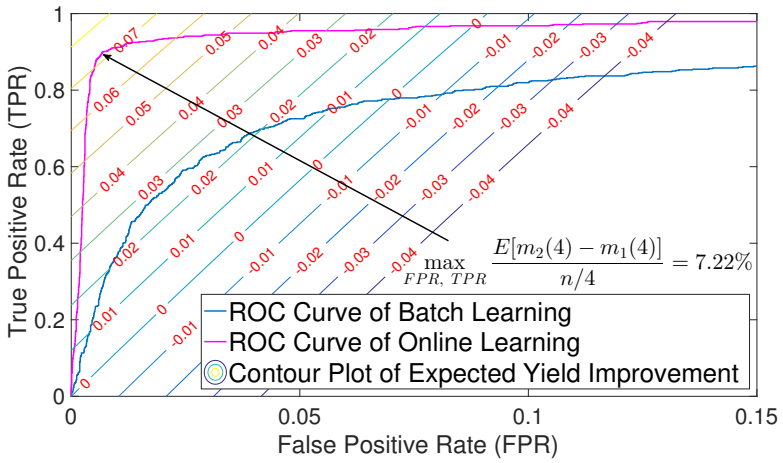


Fig. 12 Contour plots of yield improvement with number of dies in a package $s = 4$. Yield with a batch classifier is 90.78% while yield with an online classifier is 95.28%, compared to yield with no classifier of 88.06%.

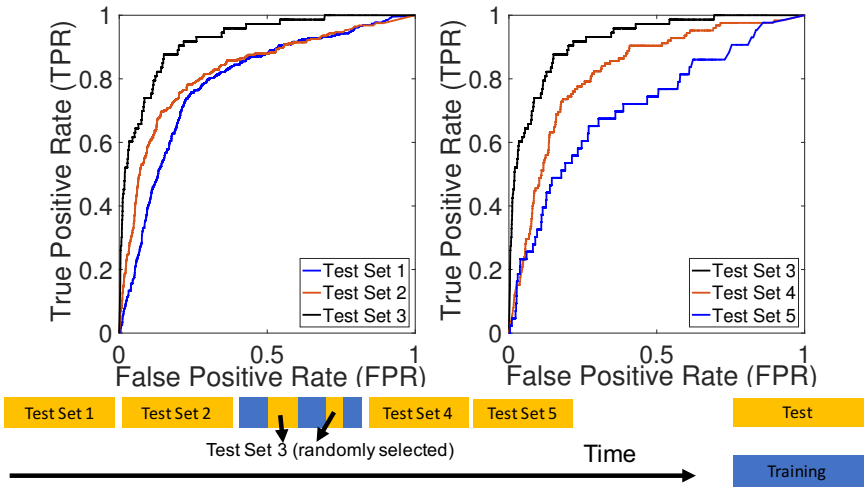


Fig. 13 Existence of concept drift. The data stream is divided into 5 chunks. The training set is randomly selected from chunk 3. The remaining dies in chunk 3 are used as test set 3. The classifier is tested on all 5 test sets. The performance of the classifier decays over time (both forward and backward) as the ROC curves show, confirming substantial concept drift.

is 0.82 ± 0.001 (a 4% improvement). If we do have expert advice for the optimal threshold, average expected yield improvement with a 95% confidence interval by our incremental learning and a single classifier is given in Table 4, where our incre-

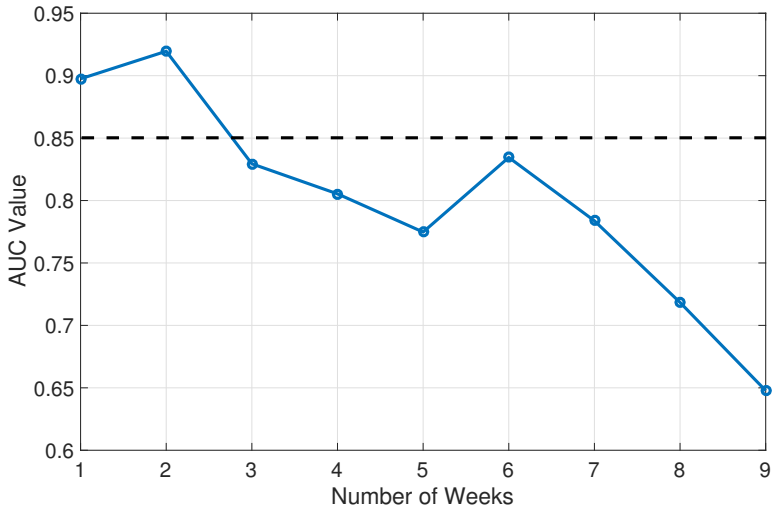


Fig. 14 The *AUC* values obtained on test sets from each week. The model is trained on data from week 0. The threshold of major concept drift is set as 0.85 (dashed line).

mental learning increases the average expected yield improvement by $1.4\times$ to $4.2\times$ compared to the single classifier.

It is important to note that, in a non-stationary environment with class imbalance, a sophisticated method to choose the threshold for classification is necessary. For example, an intuitive way is to fix the False Positive Rate (FPR). Though this method also uses assumed or historical information about the joint distribution of H (true condition) and y (predicted condition), we may not achieve expected yield improvement with this intuitive naive approach. The result of expected yield improvement using thresholds with fixed FPR at 5% for our dataset is shown in Table 5, which indicates negative yield improvement for both incremental learning and single classifier. In contrast, optimal selection of thresholds can generate substantial yield improvements, as shown in Table 4.

Table 4 Average expected yield improvement with a 95% confidence interval by incremental learning and by a single classifier when optimal thresholds are given.

s	Incremental Learning	Single Classifier	Improvement Factor
4	$\mathbf{0.36\%\pm0.03\%}$	$0.08\%\pm0.02\%$	$4.2\times$
8	$\mathbf{1.4\%\pm0.03\%}$	$0.6\%\pm0.1\%$	$2.2\times$
16	$\mathbf{3.4\%\pm0.05\%}$	$2.4\%\pm0.1\%$	$1.4\times$

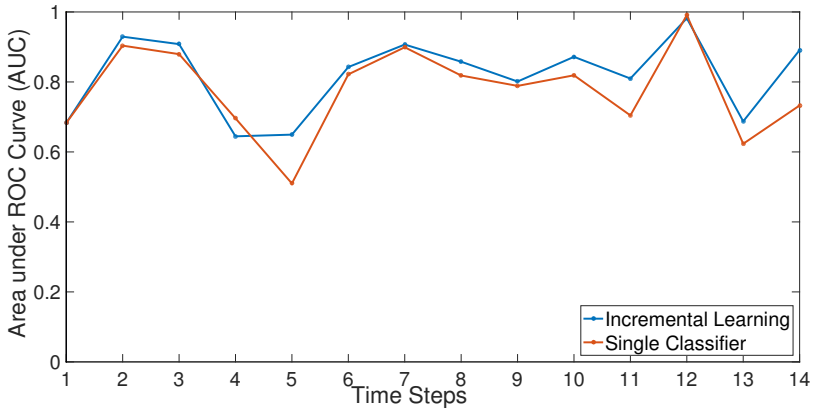


Fig. 15 Average area under ROC curves obtained on data chunks by incremental learning, and by a single classifier trained only on the most recent data chunk.

Table 5 Average expected yield improvement with a 95% confidence interval by incremental learning and by a single classifier when $FPR = 5\%$ thresholds are given.

s	Incremental Learning	Single Classifier	Improvement Factor
4	$\mathbf{-3.6\% \pm 0.02\%}$	$-4.2\% \pm 0.10\%$	–
8	$\mathbf{-2.9\% \pm 0.03\%}$	$-3.4\% \pm 0.08\%$	–
16	$\mathbf{0.0\% \pm 0.03\%}$	$-1.9\% \pm 0.08\%$	–

6 Conclusions

In this chapter, we identify two key challenges in building practical learning models for semiconductor manufacturing process yield improvement, class imbalance and concept drift. Batch, online and incremental learning frameworks are presented to resolve these challenges. As an application, we study the packaging process of flash memory. We first show the possibility of memory package yield improvement in expectation by adding a classifier before packaging. The effectiveness of the algorithms are demonstrated on real data from industry. In the future we will investigate methods to determine the optimal classification threshold in non-stationary environments. We will also devise more sophisticated models to predict defect categories. We hope our work will attract more researchers in the CAD and machine learning community to the semiconductor manufacturing area.

Acknowledgements We thank SanDisk Semiconductor (Shanghai) Co. Ltd. (SDSS) for assistance and support. We also thank Professor Roy Welsch from MIT Sloan School of Management, and Honghong Chen, Fangfang You, Wenting Zhang and Yew Wee Cheong from SDSS for helpful discussions.

References

1. A. Kusiak, "Rough set theory: a data mining tool for semiconductor manufacturing," *IEEE Trans. on Electronics Packaging Manuf.*, vol. 24, no. 1, pp. 44–50, 2001.
2. C. K. Shin and S. C. Park, "A machine learning approach to yield management in semiconductor manufacturing," *International Journal of Production Research*, vol. 38, no. 17, pp. 4261–4271, 2000.
3. S. M. Weiss, A. Dhurandhar, and R. J. Baseman, "Improving quality control by early prediction of manufacturing outcomes," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 1258–1266.
4. H. Chen and D. S. Boning, "Online and incremental machine learning approaches for IC yield improvement," in *IEEE/ACM 36th International Conference on Computer-Aided Design*, 2017.
5. H. Chen, "Novel machine learning approaches for modeling variations in semiconductor manufacturing," Master's thesis, Massachusetts Institute of Technology, 2017.
6. N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, 2002.
7. A. Tsymbal, "The problem of concept drift: definitions and related work," *Technical Report*, vol. 106, no. 2 Computer Science Department, Trinity College Dublin, 2004.
8. N. A. Arnold, "Wafer defect prediction with statistical machine learning," Master's thesis, Massachusetts Institute of Technology, 2016.
9. Y. Sun, A. K. Wong, and M. S. Kamel, "Classification of imbalanced data: A review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 04, pp. 687–719, 2009.
10. H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. on Knowledge and Data Eng,*, vol. 21, no. 9, pp. 1263–1284, 2009.
11. Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *ICML*, vol. 96, 1996, pp. 148–156.
12. C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Trans. on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.
13. H. Zhang, B. Yu, and E. F. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *IEEE/ACM 35th International Conference on Computer-Aided Design*, 2016.
14. B. Wang and J. Pineau, "Online bagging and boosting for imbalanced data streams," *IEEE Trans. on Knowledge and Data Eng.*, vol. 28, no. 12, pp. 3353–3366, 2016.
15. N. C. Oza, "Online bagging and boosting," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 3, 2005, pp. 2340–2345.
16. T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.
17. G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Trans. on Knowledge and Data Eng.*, vol. 25, no. 10, pp. 2283–2301, 2013.
18. R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.
19. S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.
20. J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, "Experimental perspectives on learning from imbalanced data," in *Proceedings of the 24th ICML*, 2007, pp. 935–942.