Defect Detection on Semiconductor Wafers by Distribution Analysis

Thomas Olschewski, Technische Universität Dresden

November 9, 2021

Abstract

A method for object classification that is based on distribution analysis is proposed. In addition, a method for finding relevant features and the unification of this algorithm with another classification algorithm is proposed. The presented classification algorithm has been applied successfully to real-world measurement data from wafer fabrication of close to hundred thousand chips of several product types. The presented algorithm prefers finding the best rater in a low-dimensional search space over finding a good rater in a high-dimensional search space. Our approach is interesting in that it is fast (quasi-linear) and reached good to excellent prediction or detection quality for real-world wafer data.
### Contents

1 Acknowledgements

2 Introduction
   2.1 Overview

3 The Algorithms
   3.1 Definitions
      3.1.1 Scaling
      3.1.2 Histogram Functions
      3.1.3 Finding Candidate Indicator Columns
      3.1.4 Computing Indicator Values \( S_C(i) \)
      3.1.5 Computing Predictions \( F(i, c) \)
      3.1.6 Cutoff Selection
      3.1.7 Classification Algorithm
   3.2 Free Parameters: \( b^+ \), \( b^- \) And \( nb \)
   3.3 Considerations on \( \mu_j \) And \( \sigma_j \)
   3.4 Limits

4 Experimental Setup
   4.1 Task And Data
   4.2 The Data Basis
   4.3 The Implementation

5 Results
   5.1 Properties
   5.2 Notational Remark
   5.3 Predicting or Detecting Frontend Defect Status of Chips
      5.3.1 Predicting Frontend Defect Status S2 of Chips
      5.3.2 Detecting Frontend Defect Status S1 of Chips
      5.3.3 Predicting Frontend Defect States Knowing Only Part of Measurement Data
   5.4 Predicting Backend Defect Status of Chips
      5.4.1 Predicting Backend Defect Status of Chips Knowing Frontend Data
      5.4.2 Predicting Backend Defect States Knowing Only Part of Measurement Data
      5.4.3 Addendum: Batch-Wise Cutoff Optimization
   5.5 Classifying Iris Types
   5.6 Dimensional Reduction
   5.7 Automatizing The Finding of Cics
      5.7.1 Practical Results by Using AutoCics()
      5.7.2 Obsoleting Free Parameters by Using AutoCics()
6 Algorithmic Complexity
  6.1 Scaling (Algorithm 1) .................................................. 32
  6.2 Computing one histogram $H_{a, nb}(x_1, \ldots, x_m)$ .......... 32
  6.3 Computing FindCics(Scale($X$), $T^+, T^-, b^+, b^-, nb$) (Algorithm 2) 33
  6.4 Computing $S_C(I_{-T})$ (Algorithm 3) .............................. 33
  6.5 Computing predictions $F(i, c)$ (Algorithm 4) ................... 33
  6.6 Naïve cutoff selection ................................................... 34
  6.7 Optimizing cutoff $c$ (Algorithm 5) ................................. 34
  6.8 Overall complexity ...................................................... 34

7 Uniting Two Classification Algorithms .............................. 36

8 Conclusion ................................................................. 39
1 Acknowledgements

I would like to thank Zoltán Sasvári for carefully reading the manuscript and for providing the environment which made this research possible.

Research leading to these results has received funding from the iRel40 project. iRel40 is a European co-funded innovation project that has been granted by the ECSEL Joint Undertaking (JU) under grant agreement no 876659. The funding of the project comes from the Horizon 2020 research programme and participating countries. National funding is provided by Germany, including the Free States of Saxony and Thuringia, Austria, Belgium, Finland, France, Italy, the Netherlands, Slovakia, Spain, Sweden, and Turkey.

Disclaimer: The document reflects only the author’s view and the JU is not responsible for any use that may be made of the information it contains.

2 Introduction

In this paper we propose a fast stochastic method for object classification which can be applied to detecting and predicting defective semiconductor devices by analyzing high-dimensional measurement data from wafer fabrication. The most important goal is predicting early in the production chain which semiconductor chips of a set of wafers will turn out to be defective later.

Our approach is interesting in that it led to good to excellent results in frontend classification tasks with real-world wafer data and is based on distributional analysis of the input data with quasilinear time complexity displaying computation times of some minutes for typical classification tasks of ours on normal PC hardware.

The function principles we are using in the classification algorithm presented in this paper are deriving certain distributional properties from the scaled features of two sets of positive and negative samples in order to select a subset of features with specific properties as candidate indicator features. These latter features are then used for attempting to classify objects hitherto unknown to the algorithm.

We will describe example applications of how this algorithm is used for
detecting frontend defects using only tiny sample sets. Another application we will present here is trying to predict backend defects using only a very small number of frontend measurements.

Part of the tasks we are tackling here have also been attacked by the algorithm described in [7] so we can compare the quality of detection or prediction by these two algorithms which are based on completely different principles. Similar as with the latter algorithm and unlike, for example, typical neural-net classification methods [8] with back-propagation, we try to avoid optimizations over high-dimensional search ranges and prefer finding some best rater in a limited set of raters over trying to find a good rater in a giant search range.

The algorithms presented in this paper have quasi-linear time complexity. They have been implemented in Python and have been used successfully for classifying close to hundred thousand semiconductor chips of different product types, where classifying 10000 chips took some minutes on normal PC hardware. We expect that re-implementation in C/C++ and parallelization would bring a considerable speedup.

The main application of the classification algorithm described in this paper is improving quality control in wafer fabrication. In an ongoing series of research projects working towards this goal, the following properties—already mentioned in [7]—have turned out to be important for some algorithm to be economically useful:

1. Not dependent on Gaussian distribution of some or all features.
2. High TP/FP quotients: as few as possible good chips should be scrapped for sorting out defective chips.
3. Efficiency: necessity of classification in fabrication real time.
4. Ability of coping with large amount of data, for example $\approx 30 - 100$ MiB per data lot.
5. Means for reducing the number of tests: measurements are costly and time consuming to different extents.

Algorithm 6 and its auxiliary functions explained in section 3 have been designed to meet these goals.

In some sense, this paper can be seen as a follow-up to [7]. Whereas in the latter paper we reported on frontend detection and prediction tasks only, in this paper we also consider backend predictions based on frontend measurements, using a classification method based on completely different principles.

### 2.1 Overview

The rest of this paper is organized as follows. In section 3 we will define notations and describe our algorithms. In section 4 we describe the experimental setup, the tasks to be solved and the data material. In section 5 we will list and analyse a number of results we obtained from a larger number of runs attempting to
classify semiconductor chips with respect to the overall defect state of one of several possible measurement steps as the main application, and results from classifying iris data. We will also propose and apply a feature relevance indicator in section 5. Section 6 will be about the worst-case complexity of the algorithms described in section 3. In section 7 we will propose an algorithm which unites the classification Algorithm 5 of this paper with the classification algorithm described in 7 by implanting some main ingredients of the latter algorithm into the former. Section 8 will close this paper by concluding thoughts.
3 The Algorithms

3.1 Definitions

In what follows, we will use the abbreviations $\mathbb{B} = \{0, 1\}$, $\mathbb{Q}^* = \mathbb{Q} \cup \{-\infty, \infty\}$. Let $X \subseteq \mathbb{Q}^{m \times n}$ be a data matrix describing $m$ objects by $n$ numbers each.

3.1.1 Scaling

To begin with, the first step of our classification algorithm is column-wise scaling the input data matrix $X$. Scale($X$) refers to the following function.

\[
\text{Input/Output: } X \in \mathbb{Q}^{m \times n} \\
1 \text{ for } j = 1 \ldots n \text{ do} \\
2 \quad \text{compute } \mu_j, \sigma_j \\
3 \quad \text{// mean and standard deviation of the } j\text{-th column of } X \\
4 \text{ end} \\
5 \text{ for } i = 1 \ldots m \text{ do} \\
6 \quad \text{ for } j = 1 \ldots n \text{ do} \\
7 \quad \quad x_{i,j} = \begin{cases} \\
8 \quad \quad \quad \frac{x_{i,j} - \mu_j}{\sigma_j}, & \sigma_j \neq 0 \\
9 \quad \quad \quad 0, & \sigma_j = 0 \\
10 \quad \end{cases} \\
11 \text{ end} \\
12 \text{ end}
\]

Algorithm 1: Scale($X$)

Every column of $X$ has mean 0 and standard deviation 1 after scaling. After scaling $X$, the algorithms do not operate on the original rows $x_{i,1}, \ldots, x_{i,n}$ but on the corresponding numbers that quantify how far (by how many standard deviations $\sigma_j$) $x_{i,j}$ is away from the mean $\mu_j$ of its column $j$.

Note. We always assume Scale($X$) $\in \mathbb{Q}^{m \times n}$. This is the case if Scale($X$) is computed numerically using floating point arithmetics. With symbolic computation, the element type of Scale($X$) would be an extension field $\mathbb{Q}(\sigma_1, \ldots, \sigma_n)$.

3.1.2 Histogram Functions

We consider the input objects—represented by lines of the input data matrix $X$—as samples of random processes the true distributions of which are unknown to us. Furthermore, we consider positive objects as results of a random process generating positive objects and negative objects as results of a possibly different random process generating negative objects. For guessing certain statistical properties of the true distributions, we need a means to re-construct approximations of the shape of the true distributions. For doing this, we use histograms derived from the sample objects $T^+, T^-$ which will be described now.

Let $v \in \mathbb{B}^m$ be a 0-1 column vector. We call the object belonging to line $i$ a
positive object if $v_i = 1$ and a negative object if $v_i = 0$. Let

$I^+ = \{i \in 1, \ldots, m \mid v_i = 1\}$

and

$I^- = \{i \in 1, \ldots, m \mid v_i = 0\}$

be the sets of line indices belonging to all positive resp. negative objects. Let $T^+ \subseteq I^+$ and $T^- \subseteq I^-$ be two subsets which will serve as index sets of a set of positive training objects and a set of negative training objects. Let be

$I_{\sim T} = \{1, \ldots, m\} \setminus (T^+ \cup T^-)$

the index set of objects outside of the training sets. This is the set of indices of all objects to be classified. We will skip training objects in the classification loop for two reasons: firstly, in order to keep a clean separation of what is known to the algorithm ($v(i)$ for $i \in I_{\sim T}$) and what is to be predicted ($v(i)$ for $i \notin T^+ \cup T^-$). And secondly, we consider classifying objects already known to the algorithm as a different, simpler problem than classifying objects not yet known to the algorithm. Later, we will also need the index set of all training objects

$I_T = T^+ \cup T^-.$

Since what we actually will be doing is using these two sets of training objects as samples for guessing certain statistical properties of the full distributions of the input data, we can call the two training sets positive samples and negative samples as well.

For convenience and if there is no danger of misinterpretation, we will not always make a difference when referring to objects and referring to lines of $X$.

We assume $\text{nb} \geq 3$ ("number of bins") to be a constant which will be used for setting the number of bins in computing all occurring histograms.

Let $a$ be an increasing sequence of numbers

$a = (-\infty < a_1 < a_2 < \cdots < a_{\text{nb} - 1} < \infty) \in \{-\infty\} \times \mathbb{Q}_{\text{nb} - 1} \times \{\infty\}$

whereby we always set $a_0 = -\infty$, $a_{\text{nb}} = \infty$.

Every such $a$ induces a partitioning

$\mathbb{Q}^* = \bigsqcup_{k=0}^{\text{nb} - 1} I_i$

into mostly half-open intervals\footnote{The only exception being the first interval $I_0 = (-\infty, a_1)$ which is open at both ends.} where

$I_0 \quad = \quad (a_0, a_1) \quad = \quad (-\infty, a_1)$

$I_1 \quad = \quad [a_1, a_2)$

$\ldots$

$I_i \quad = \quad [a_i, a_{i+1}) \quad \quad \quad (i = 0, \ldots, \text{nb} - 1)$

$\ldots$

$I_{\text{nb}-2} \quad = \quad [a_{\text{nb}-2}, a_{\text{nb}-1})$

$I_{\text{nb}-1} \quad = \quad [a_{\text{nb}-1}, a_{\text{nb}}) \quad = \quad [a_{\text{nb}-1}, \infty)$
The assumption \( nb \geq 3 \) guarantees that there is at least one interval \([a_1, a_2]\) with both limits being finite.

**Definition.** Let \( H_{a, nb} : Q^m \rightarrow Q^{nb} \) be the histogram function that, given a sequence of numbers \( x = (x_1, \ldots, x_m) \) and a sequence of boundaries \( a = (-\infty < a_1 < a_2 < \cdots < a_{nb-1} < \infty) \), assigns to every bin index \( k \in \{0, 1, \ldots, nb - 1\} \) the relative frequency of this bin, i.e. the number of elements lying in \( I_k \) divided by \( m \):

\[
H_{a, nb}(x) = (h_0, \ldots, h_{nb-1})
\]

where

\[
h_k = \frac{1}{m} \cdot | \{ i \in \{1, \ldots, m\} \mid x_i \in I_k \} | \quad (k = 0, 1, \ldots, nb - 1)
\]

Obviously, \( \sum_{k=0}^{nb-1} h_k = 1 \). Note that \( a \) is required to be in strictly ascending order whereas \( x \) may be unordered.

Given \( nb \) and \( x \in Q^m \), we can specify boundaries \( a^* = (-\infty < a_1^* < a_2^* < \cdots < a_{nb-1}^* < \infty) \) in a way that the inner intervals \( I_1^*, \ldots, I_{nb-2}^* \) have equal width \( \frac{\max(x) - \min(x)}{nb-2} \). We just set \( a_1^* = \min(x) \) and \( a_{nb-1}^* = \max(x) + \varepsilon \) for some \( \varepsilon > 0 \). Then

\[
[\min(x), \max(x) + \varepsilon] = \bigcup_{k=1}^{nb-2} I_k^*
\]

When using these boundaries \( a^* \), the outmost intervals \( I_0 = (-\infty, a_1^*) \) and \( I_{nb-1} = [a_{nb-1}^*, \infty) \) do not contain any element of \( x \) and thus \( h_0 = h_{nb-1} = 0 \) in this case. Every \( nb \) setting induces a function which maps \( x \in Q^m \) to the special boundaries vector \( a^* \) defined above:

\[
\begin{align*}
a_{nb}^* : Q^m &\rightarrow Q^{nb+1} \\
\quad a_{nb}^*(x) &= (-\infty, a_1^*, a_2^*, \ldots, a_{nb-1}^*, \infty)
\end{align*}
\]

**Proposition.** Let \( j \in \{1, \ldots, n\} \) be some column index. Let \( x_{1,j}, \ldots, x_{m,j} \) be independent and identically distributed with distribution \( P^j \). Let \( I = \{i_1, \ldots, i_s\} \subset \{1, \ldots, m\} \) some fixed subset of indices and \( H_{a, nb}^I : Q^s \rightarrow Q^{nb} \), \( H_{a, nb}^I(x_{i_1,j}, \ldots, x_{i_s,j}) = (h_0^I, \ldots, h_{nb-1}^I) \) the histogram of \( (x_{i_1,j}, \ldots, x_{i_s,j}) \). Then \( h_k^I \) is an unbiased estimator of \( P[x_{i,j} \in [a_k, a_{k+1}]] \) for \( k = 0, \ldots, nb - 1 \).

**Proof.** Let \( k \in \{0, \ldots, nb - 1\} \) be fixed. Define \( X_{i_1}, \ldots, X_{i_s} \) by

\[
X_{i_\theta} = \begin{cases} 1, & x_{i_\theta} \in [a_k, a_{k+1}) \\ 0, & \text{else} \end{cases}
\]

Then by linearity of the expectation \( E \):

\[
E[h_k^I] = E\left[\frac{X_{i_1} + \cdots + X_{i_s}}{s}\right] = E[X_{i_1}] = P[x_{i,j} \in [a_k, a_{k+1}]], \quad \text{qed.}
\]

Thus by using \( T^+ \cup T^- \) for \( I \), we can take histograms derived from the training set as unbiased estimators for the distribution of every column \( j \).

\[^2\text{As an exception, the left-most interval } (a_0, a_1) = (-\infty, a_1) \text{ is open on the left.}\]
3.1.3 Finding Candidate Indicator Columns

Input: \( \text{Scale}(X) \in \mathbb{Q}^{m \times n}, T^+ \subseteq I^+, T^- \subseteq I^-, b^+ \in \mathbb{Q}, b^- \in \mathbb{Q}, \text{nb} \in \mathbb{N}_{\geq 3} \)

Output: list of Cics \( L = \{(j, a_k^*, a_{k+1}^*) | s = 1, \ldots, S\} \)

1 for \( j = 1 \ldots n \) do
2 set \( x^+ = (x_{i,j} | i \in T^+) \)
3 compute \( a^* = a^*_\text{nb}(x^+) \)
4 compute \( h^+ = H_{a^*, \text{nb}}(x^+) \in \mathbb{Q}^\text{nb} \)
5 \( h^+_{\text{max}} = \max(h^+_0, \ldots, h^+_{\text{nb}-1}) \)
6 \( k^+ = \min\{k \in \{0, \ldots, \text{nb} - 1\} | h^+_k = h^+_{\text{max}}\} \)
7 set \( x^- = (x_{i,j} | i \in T^-) \)
8 compute \( h^- = H_{a^*, \text{nb}}(x^-) \in \mathbb{Q}^\text{nb} \)
9 if \( h^+_{\text{max}} > b^+ \land h^-_{k^+} < b^- \) then
10 append \((j, a_k^*, a_{k+1}^*)\) to \( L \)
11 end
12 end

Algorithm 2: FindCics(\( X, T^+, T^-, b^+, b^-, \text{nb} \))

FindCics(\( X, T^+, T^-, b^+, b^-, \text{nb} \)) takes as input the scaled input data matrix \( \text{Scale}(X) \), two indices sets of positive and negative training objects, respectively, a lower bound \( b^+ \) and an upper bound \( b^- \), and the number of bins \( \text{nb} \) to be used in the histograms.

Definition. A candidate indicator column (Cic) is any column \( j \) of \( \text{Scale}(X) \) which has the property that the most-frequent bin \( I_{k^+} \) of the column \( j \) entries of all lines belonging to positive samples is \( > b^+ \) whereas the same bin \( I_{k^+} \) (i.e. with the same boundaries) of the column \( j \) entries of all lines belonging to negative samples is \( < b^- \).

If there are multiple \( k \in \{0, \ldots, \text{nb} - 1\} \) with \( h^+_k = h^+_{\text{max}} \) then we take the one \( k \) with the smallest index in the above algorithm 2:

\[ k^+ = \min\{k \mid h^+_k = h^+_{\text{max}}\} \]

In I\(_{k^+} = [a^*_{k^+}, a^*_{k^++1}]\), the 2nd and 3rd entry of the triplet \((j, a^*_{k^+}, a^*_{k^++1})\) being appended to \( L \) are the lower and upper boundary of the most-frequent bin \( I_{k^+} \) of the histogram of positive objects \( h^+ \).

Clearly, FindCics(\( \text{Scale}(X), T^+, T^-, b^+, b^-, \text{nb} \)) returns the list of all Cics of \( \text{Scale}(X) \).

3.1.4 Computing Indicator Values \( S_C(i) \)

Definition. Let \( C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, \text{nb}) \) be the output of Algorithm 2. Then define

\[ S_C : I_-T \rightarrow \mathbb{N}_{\geq 0} \]

\[ S_C(i) = \left| \{(j, a^*_{k^+}, a^*_{k^++1}) \in C \mid x_{i,j} \in [a^*_{k^+}, a^*_{k^++1}]\} \right| \]
In words, $S_C$ assigns to every index $i$ of a non-training object $(x_{i,1},\ldots,x_{i,n})$ the number of those Cics $j$ for which the $j$-th column entry of this object lies in the most-frequent bin of the histogram of positive samples.

| Input: $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ |
| Output: $S_C(I_{-T})$ |
| 1 for $i \in I_{-T}$ do |
| 2 | compute $S_C(i) = |\{(j,a^*_k,a^*_k+1) \in C \mid x_{i,j} \in [a^*_k,a^*_k+1]\}|$ |
| 3 end |

Algorithm 3: Computing indicator values $S_C(i)$

### 3.1.5 Computing Predictions $F(i, c)$

After having computed an indicator value $S_C(i)$ for every object $i$ outside the training sets we need to obtain binary predictions, 1 for predicting object $i$ to be positive, 0 for negative:

| Input: $S_C(I_{-T})$ as computed by Algorithm 3, $c \geq 0$ |
| Output: $F(c) \in \mathbb{B}^{|I_{-T}|}$ |
| 1 for $i \in \{1,\ldots,m\} \setminus (T^+ \cup T^-)$ (= $I_{-T}$) do |
| 2 | $F(i, c) = \begin{cases} 1, & S_C(i) \geq c \\ 0, & S_C(i) < c \end{cases}$ |
| 3 end |

Algorithm 4: Computing predictions

Of the wide range of methods for specifying $C$ we describe just two.

### 3.1.6 Cutoff Selection

Let be

\[
I^+_{-T} = I^+ \cap I_{-T} \\
I^-_{-T} = I^- \cap I_{-T}
\]

the index sets of objects to be classified which are positive or negative, respectively, in truth. Their average indicator values are:

\[
Av^1 = \frac{1}{|I^+_{-T}|} \sum_{i \in I^+_{-T}} S_C(i) \\
Av^0 = \frac{1}{|I^-_{-T}|} \sum_{i \in I^-_{-T}} S_C(i)
\]
In what follows, we assume $A^v_1 - A^v_0 \geq 0$.

**Naïve cutoff selection.**

$$c = \frac{A^v_1 - A^v_0}{2}$$

This better-than-nothing selection can be computed very fast but may make predictions $F(i, c)$ far from what can be achieved with a more sophisticated cutoff.

**Optimizing cutoff for some statistical quantity $Q(v, w)$.**

Let $(i^1, \ldots, i^{|I-\tau|})$ be an enumeration of $I-\tau$ in some fixed order and $v = (v(i^1), \ldots, v(i^{|I-\tau|}))$ the 0-1 vector of their true positive/negative states. Let be $F(c) = (F(i^1, c), \ldots, F(i^{|I-\tau|}, c))$ the 0-1 vector of all $F(i, c)$ for $i \in I-\tau$ using the same index ordering as in $v$. Furthermore, let $Q(v, w)$ be some statistical quantity measuring similarity of two 0-1 vectors $v$ and $w$. This may be accuracy 4 or—by considering $F(c)$ and $v$ as two raters of the same feature vector—Cohen’s kappa 5, among other possibilities.

Denote

$$S_C^{\min} = \min S_C(I-\tau)$$
$$S_C^{\max} = \max S_C(I-\tau)$$

3If this does not hold true in a specific application then this may be an indication of insufficient training sets: not characteristic enough, or with contradictive positive/negative bits $v(i)$.

4Accuracy is the amount of coincident bits of two $r$-bit vectors:

$$\text{accu}(v, w) = \frac{1}{r} \sum_{i=1}^{r} (v_i \cdot w_i + (1 - v_i)(1 - w_i))$$

5Cohen’s Kappa 2 is used for measuring agreement of two binary $r$-bit raters $v$ and $w$ and defined as

$$\text{kappa}(v, w) = \frac{\text{accu}(v, w) - p_e}{1 - p_e}$$

where

$$p_e = \frac{1}{r^2}(n_{v=0 \cdot w=0} + n_{v=1 \cdot w=1}).$$
Input: $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$

Output: $C_{opt} \in \mathbb{Q}, F(C_{opt}) \in \mathbb{B} | I \cap T |$

1 for $c = S^\text{min}_C, S^\text{min}_C + 1, \ldots, S^\text{max}_C$ do
2 \hspace{1em} compute $F(c)$ by Algorithm 4
3 \hspace{1em} compute $Q(F(c), v)$
4 end
5 $Q_{opt} = \max\{Q(F(c), v) \mid c \in C\}$
6 $C_{opt} = \text{argmax}(Q_{opt})$

Algorithm 5: Optimizing cutoff $c$

Since the indicator values $S_C(i)$ are small integers which are limited to the number of Cics there is little practical advantage in employing more refined methods than Algorithm 5 for finding an optimal cutoff $c$ like the hill-climbing approach described in the subsection "Refined Method For Cutoff Selection" of [7].

If the optimal cutoff can not be computed for the whole set of objects because only some subset of all objects is known to the user then the cutoff optimization has to be performed for the known set of objects first and used as an estimator for the whole lot of data later.

3.1.7 Classification Algorithm

By putting it all together, we arrive at the following algorithm for object classification.

Input: $X \in \mathbb{Q}^{m \times n}; b^+, b^- \in \mathbb{Q}; nb \in \mathbb{N}_{\geq 3}; t^+, t^- \in \mathbb{N}_{\geq 1}$

Output: $F(C_{opt}) \in \mathbb{B} | I \cap T |$

1 Scale(X) by Algorithm 1
2 select random subsets $T^+ \subseteq I^+$ of size $t^+$, $T^- \subseteq I^-$ of size $t^-$
3 $C := \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ by Algorithm 2
4 compute $S_C(I-T)$ by Algorithm 3
5 $S^\text{min}_C := \min S_C(I-T)$
6 $S^\text{max}_C := \max S_C(I-T)$
7 compute $C_{opt}$ and $F(C_{opt})$ by Algorithm 5

Algorithm 6: Classification algorithm

3.2 Free Parameters: $b^+$, $b^-$ And nb

$b^+$ and $b^-$ specify which features are to be considered as Cics in the definition of Cic on page 9. They are used in Algorithm 2. In practice, the value of $b^+$ seems to be far more important than $b^-$ as most small positive values $b^- \approx 0$.
served equally well in our applications.

In section 5.7, we will present Algorithm 7 as an alternative for Algorithm 2 in order to not control the selection of Cics by \( b^+ \) and \( b^- \) but by specifying how many of the top ranks according to their \( n_{diff} \) value are to be used as indicator columns. This replaces \( b^+ \), \( b^- \)—which are obsolete then—by a new parameter \( t \) which in turn may be obsoleted by defining a convenient default value depending on the feature count of the input data matrix \( X \).

\( nb \) specifies how many intervals are to be filled by the histogram function \( H_{a,nb} : Q^m \rightarrow Q^{nb} \) defined on page 8.

Note that the output of \( \text{Scale}(X) \)—which serves as input to \( H_{a,nb} \)—may be quantized already, firstly because every continuous feature of the input data matrix \( X \in Q^{m,n} \)—measurements of an analog quantity, for example—is already quantized, and secondly by the well-known limitations of floating point processing and the number format of \( X \) and \( \text{Scale}(X) \). As is typical of algorithms which are processing real-world data using floating point, quantizations may be obvious (number format of numerical data in files, for example) or silent (pre-quantized measurement data, limitations of floating point processing). In our chip classifications, we limited the entries of \( \text{Scale}(X) \) to 3 significant digits with exponent notation if necessary and set \( nb = 1000 \) or 2000.

### 3.3 Considerations on \( \mu_j \) And \( \sigma_j \)

In practical applications, the means \( \mu_j \) and the standard deviations \( \sigma_j \) \((j = 1, \ldots, n)\) of the total population as they are needed in the scaling Algorithm 1 may not be known in advance. We can obtain canonical estimations \( \bar{\mu}_j \) and \( \bar{s}_j \) by computing the sample mean and sample variance of the training objects, i.e.

\[
\bar{\mu}_j = \frac{1}{n} \cdot \sum_{i \in I_T} x_{i,j}
\]

\[
\bar{s}_j^2 = \frac{1}{n-1} \cdot \sum_{i \in I_T} (x_{i,j} - \bar{\mu}_j)^2
\]

where \( I_T = T^+ \cup T^- \). It is well-known that \( \bar{\mu}_j \) is an unbiased estimator for \( \mu_j \) and \( \bar{s}_j^2 \) is an unbiased estimator for the variance \( \sigma_j^2 \). For concave functions \( f \) of some random variable \( X \) it holds \( E[f(X)] \leq f(E[X]) \) but not necessarily equality. Since the square root is a concave function, \( s_j := \sqrt{\bar{s}_j^2} \) may not be an unbiased estimator for \( \sigma_j \) and generally, there is no formula for unbiased \( \sigma_j \) estimation being true for all distributions. In case the bias of the estimation of \( \sigma_j \) being especially low is of elevated importance there are improved estimators. See, for example, [1] and the references therein on using the estimator \( \sqrt{\frac{1}{n-1} \cdot \sum_{i \in I_T} (x_{i,j} - \mu_j)^2} \) in case of normal distribution.

Of course, there may be good reasons for decoupling the sample set for estimating \( \mu_j \) and \( \sigma_j \) from \( T^+ \) and \( T^- \) by choosing a different set than \( I_T \) for
the former because the size of $I_T$ may be influenced by considerations of avoiding over- and undersampling in the classification.

**Note.** It proved useful in our applications to have the possibility to optionally bypass the FindCics stage (Algorithm 2) by implementing a means to specify the Cics manually, especially in case of relevant Cics being known in advance from former runs with similar data. Then the C in $S_C$ is not the output of FindCics(...) but a list of triples $(j_s, a_{k_s}^+, a_{k_s+1}^+)$ created for those columns $j_s$ specified by the user. This has been applied successfully in cases where several lots of measurement data are to be classified whereby the chips are of the same product type.

### 3.4 Limits

Clearly, the classification Algorithm 6 requires the existence of some columns of $X$ where the conditional distribution on the positive samples differs from the conditional distribution on the negative samples. In a stochastic sense, this presupposes the existence of some features the true distributions of which differ when restricted to the positive and to the negative objects separately.
4 Experimental Setup

4.1 Task And Data

The tasks to be solved are as described in [7]:

*Given measurements and the defect states for chips of a small training set, predict the defect state of the remaining chips of the lot based on the measurements only.*

The data we are given consist of measurement data from chip fabrication. The data is organized in lots where one lot is a set of wafers from production. Each wafer carries a fixed number of chips of the same product type.

In what follows, we ignore the partitioning of a lot into wafers and consider a lot as a series of \( m \) chips of the same product type where each chip is represented by \( n \) measurements. Using the notation of subsection 3.1, the input to our algorithm consist of:

- \( X \in \mathbb{Q}^{m \times n} \) where the \( i \)-th line represents the \( i \)-th chip and the \( j \)-th column contains the \( j \)-th measurement

- \( v \in \mathbb{B}^n \) where \( v_i \) is the true defect state of the \( i \)-th chip.

Every column represents a feature because measurements in the same column are presupposed to be of the same type.

The general hypothesis the classification algorithm presented in [7] is based on is that defective devices may possess abnormalities in patterns derived from deviations by thresholding from the component-wise means of measurement data. The approach presented in this paper in turn resides on the hypothesis that there may exist features—columns of \( X \) in the above algorithms—which we call Cics where defective devices differ from normal devices in their distributional properties.

The columns of \( X \) are grouped into so-called **MeasSteps** (measurement steps) named S1, S2, ... for front-end MeasSteps. Every column of \( X \) belongs to one unique MeasStep. To every chip and every MeasStep there is assigned either an error code specifying the type of defect which had occurred in this step, or "0" meaning “passed”. In addition to this, if the tasks consists in predicting backend defects then to every chip there is assigned a backend error code or "BEpass".

In the tasks we are describing here, the true defect state \( v_i \in \mathbb{B}^n \) of the \( i \)-th chip may be

- the frontend defect state “anything but "0"” or

- the backend defect state “anything but "BEpass"” in our tasks.

**Notation.** We will occasionally abbreviate “frontend” as **FE** and “backend” as **BE**.
The research problem to be solved is: Find algorithms that are able to learn the function \( f : \mathbb{Q}^n \rightarrow \mathbb{B} \) which maps the vector of measurements of the \( i \)-th chip to the defect state of this chip:

\[
(x_{i,1}, \ldots, x_{i,n}) \mapsto v_i
\]

So, given \((x_{i,1}, \ldots, x_{i,n})\) for a small set of training chips with indices \( i \in T^+ \cup T^- \) we want an algorithm that predicts \( v_i \) for chips of the same product type with defect state yet unknown, knowing only their measurement vectors \((x_{i,1}, \ldots, x_{i,n})\).

We are not only interested in predicting defect states of chips when a full set of measurements is available but also in predictions when only part of measurement data is available. This bears the possibilities of saving costs by reducing the amount of measurements and predicting defects earlier in the production process.

### 4.2 The Data Basis

In our experiments we considered chips of 4 different product types A, B, C and D with the following properties:

| Product | #Chips | #Measurements per chip | continuous only |
|---------|--------|------------------------|-----------------|
| A       | 8280   | 385/1084               | yes/no          |
| B       | 11952  | 332                    | yes             |
| C       | 11328  | 915                    | no              |
| D       | 34550  | 150                    | no              |

The data of Product B contains only continuous features (currents, voltages, ...) whereas products C and D also include discrete features (flag words, counts, ...).

An important feature of our method for object classification is that we do not use any meta-knowledge about the data. For chip measurement data, this means that we do not know the types or units nor the meanings of the measurements. All we know is their numerical values.

### 4.3 The Implementation

A program for the application described in subsection 4.1 including several extra features (graphical output etc.) has been realized in 3400 lines of Python 3 using the NumPy [5] and Matplotlib [6] packages. The algorithms presented above contain several loops which could be parallelized by using the multiprocessing package: the first \( j \) loop and the \( i \) loop in Algorithm [1] the \( j \) loop in Algorithm [2] the \( i \) loop in Algorithm [3] and the \( c \) loop in [4]. We keep this optimization for future versions of this implementation.
5 Results

5.1 Properties

The main Algorithm meets all of the five goals listed in section which seems to make it well-suited for detecting or predicting the overall frontend defect states of chips using measurement data.

When using the classification algorithm presented here together with the dimensional reduction method dim-reduce(Scale(X), sharpness) described in subsection “Dimensional Reduction” of [7] as a preprocessing step, the algorithm presented here computed better results than the algorithm of [7] with the dimensional reduction in a series of classification runs on the same input. In what follows, we refer to this dimensional reduction by dim-reduce. See [7] for more details. See subsection 5.3.3 for numbers.

The hardware demands of the algorithms presented in this paper are low. All classifications the results of which are listed in this paper together took 22 minutes on a Pentium i5-750 using a Python implementation of Algorithm without parallelization on task- or thread-level.

5.2 Notational Remark

We will abbreviate “true positive” as TP, “false positive” as FP, “true negative” as TN and “false negative” as FN henceforth.

5.3 Predicting or Detecting Frontend Defect Status of Chips

5.3.1 Predicting Frontend Defect Status $S_2$ of Chips

The goal of this application is to predict the frontend overall defect state "0" of MeasStep $S_2$ if the $S_1$ and $S_2$ measurements are known to the classification algorithm. In the following example, we are going to classify 11328 chips of Product C using 20%/5% of all positive/negative chips for training, setting $b^+ = 0.3$. The prediction quality is summarized by the 4 fields and their various quotients in the following table:
The next two figures are created by the Algorithms 3 and 5 while solving the aforementioned task. Figure 1 shows the value $S_C(i)$ for each of the 10669 objects $i$ to be classified.

**Note.** For the beholder’s convenience, all positive objects $i$ are rearranged to the left in this diagram whereas all negative objects $i$ are moved to the right. So the differences in $S_C(i)$ of positive objects and negative objects are clearly visible. This will be done in all diagrams of this type from now on. Figure 2 shows the plots of cutoff-vs-accuracy and cutoff-vs-kappa which are created by Algorithm 5 while finding some cutoff $C_{opt}$ for optimizing accuracy or kappa.

![Figure 1: $S_C(i)$ in the S2 classification with product C.](image)

![Figure 2: plots of cutoff-vs-accuracy and cutoff-vs-kappa.](image)
The kappa value of this result is slightly better than the value (0.979) computed by the algorithm of [7]. All in all, this task is easy to both algorithms even though they differ vastly in the function principles they are based on.

As another illustration of the performance of Algorithm 6 at predicting front-end defects, we will show next the results of classifying 34550 chips of product D predicting the overall defect state "0" of the second MeasStep S2. We used 50%/10% of all positive/negative objects as samples for training.

|   |   |
|---|---|
| TP | 740 |
| FP | 0  |
| TN | 31347 |
| FN | 36  |
| TP/(TP+FN)% | 95.4 |
| TN/(TN+FP)% | 100.0 |
| FP/(TP+FN)% | 0.0  |
| FN/(TN+FP)% | 0.1  |
| TP/FP | ∞   |
| TN/FN | 870.8 |
| Accuracy% | 99.9 |
| Kappa | 0.976 |

The latter result has been achieved by Algorithm 6 using only 3 columns as Cics. We used the same 3 columns here as we did in our experiment for minimizing the number of input features for BE defect prediction, set out in subsection 5.4.2. When using only 2 columns for Cics or specifying the Cics by $b^+$, the resulting quality of predictions was only marginally different.

The limits mentioned in subsection 3.4 of the distributional analysis approach of Algorithm 6 showed up when classifying Product A. Unlike with the
algorithm of [7] where 5000 chips could be classified with a kappa value of 0.707 and accuracy 98.3%, using Algorithm [6] the best result with the same positive/negative training set sizes 70%/10% was kappa 0.469 and accuracy 96.3%. In accordance with this, looking at the histograms of Product A, one sees that when leaving out all discrete features, the positive and negative histograms of many features actually do not seem to differ much in a way that histogram analysis could detect with some certainty. Classifying Product A was also somewhat harder than classifying the other products mentioned there to the classification algorithm of [7].

5.3.2 Detecting Frontend Defect Status S1 of Chips

In this application, the task was detecting the frontend defect status S1 of 10000 chips of product B while knowing measurements of S1 and S2. In order to know how the detection quality depends on the training set sizes, we tested a wide range of sizes. In the following table we list a small representative selection of results. We set \( n_b \) to 2000 and \( b^+ \) to 0.98 throughout this series.

| Train Pos% | Train Neg% | FP | FN | Kappa |
|------------|------------|----|----|-------|
| 90         | 10         | 0  | 3  | 0.979 |
| 50         | 10         | 1  | 10 | 0.983 |
| 20         | 10         | 14 | 1  | 0.985 |
| 1          | 10         | 16 | 21 | 0.970 |
| 0.125      | 10         | 21 | 16 | 0.971 |
| 0.125      | 5          | 22 | 1  | 0.979 |
| 0.125      | 0.125      | 23 | 0  | 0.982 |
| 0.125      | 0.01       | 12 | 10 | 0.983 |
| 0.125      | 0.005      | 23 | 1  | 0.981 |
| 0.1        | 10         | 91 | 2 | 0.741 |

This table shows that the detection quality is hardly sensitive to the selection of the training set sizes in this series. Remarkably, when setting TrainPos=0.125% and TrainNeg=0.005%, the training sets consist of only 3 chips: 2 positive samples and 1 negative sample.

Figure 3 shows the plots of cutoff-vs-accuracy and cutoff-vs-kappa which are created by Algorithm [5] while classifying 10000 chips of product B using 2 positive sample chips and 1 negative sample chip for training.

The next 3 figures show some histograms of the above series, created by Algorithm [2]. In each of the 3 figures, there are 25 diagrams in a 5 x 5 grid. The five rows correspond to columns #81 to #85 of the matrix Scale(X). There are 5 histograms in each row, each rendered by using 100 bins:

1. the (signed) difference of the pos and neg histograms
2. their absolute difference
3. the histogram of positive objects (“pos”)
Figure 3: Cutoff optimization of the S1 classification with product B.

Cutoff vs Accuracy (left), cutoff vs Kappa (right) knowing only samples.

4. the histogram of negative objects ("neg")

5. the histogram of all objects ("all")

Figure 4 shows how the histograms would be looking like if the classification algorithm would get to see all objects as input data. In real applications the algorithms get to see only the training sets. So in the next Figure 5, the histograms are displayed when using 20%/10% of all objects as samples for the positive/negative training sets. The topmost row displays column #81 which is not used as a Cic (see page 9) in classification, whereas the next 4 rows (displaying columns #82 to #85) are used as Cics. In accordance with this, if we compare the 3rd column of Figure 5 with the 4th column, we see that in the 4th column, the peak of the topmost diagram is much higher than in the remaining 4 diagrams of this column. This means, the difference in relative frequencies of the most frequent bin of the "pos" histogram (column 3) and the "neg" histogram (column 4) is much lower in column #81 than in the columns #82 to #85. Thus Algorithm 2 will use columns #82 to #85 as Cics, but not #81.

Finally, in Figure 6, we see the histograms created just like the former, but this time using only 4 chips for training: 2 positive objects and 2 negative objects. As one can see, the only relative frequencies occurring are 0, 0.25, 0.5, 0.75 and 1. Again, by comparing the diagrams of the 3rd column ("pos") with the diagrams of the 4th column ("neg"), we see that the peaks in the 4 bottom "pos" histograms are located differently than in the belonging 4 bottom "neg" histograms of the same row. But this is not the case when comparing the topmost "pos" with the topmost "neg" histogram, which again gives a hint why the FindCics Algorithm 2 takes columns #82 to #85 as Cics, but not #81, even when knowing only 2+2 samples for training.
5.3.3 Predicting Frontend Defect States Knowing Only Part of Measurement Data

In order to evaluate the suitability of the algorithm described in this paper to the prediction of the frontend overall defect state \"0\" with a successively reduced set of features as input, we repeated the application described in section \"Dimensional Reduction\" of [7] with the same input data but replaced the algorithm of loc. cit. by the classification algorithm described in this paper.

After scaling the input matrix $X$ by $\text{Scale}(X)$ in Algorithm 6, we proceeded by processing the scaled matrix by running $\text{dim-reduce}(\text{Scale}(X), \text{sharpness})$ with increasing sharpness settings 0,1,2,3,4. Both algorithms have been extended by this dim-reduce preprocessing.

The next two tables show the results of classifying 11328 chips of product C (429 S1-measurements per chip). The algorithms are given measurements of MeasStep S1 only, and the task is predicting the overall defect state \"0\" of MeasStep S2 without knowing any S2 measurements.

The following table shows the quality of prediction by the classification algorithm of [7] using abs-t-excess thresholding.
Figure 5: Histogram collection of 5 features, knowing 20%/10% sample sets only.

In this second table we list the corresponding results by the algorithm described in this paper, using 50%/5% of all positive/negative chips as samples:

| Sharpness of reduction | #Features omitted | %Features omitted | Accuracy | Kappa | TP  | FP |
|------------------------|------------------|------------------|----------|-------|-----|----|
| 0                      | 0                | 0%               | 0.989    | 0.885 | 33.3|    |
| 1                      | 283              | 66.0%            | 0.974    | 0.664 | +∞  |    |
| 2                      | 346              | 80.7%            | 0.973    | 0.656 | 34.4|    |
| 3                      | 379              | 88.3%            | 0.946    | 0.194 | 234 |    |
| 4                      | 387              | 90.2%            | 0.946    | 0.188 | 201 |    |

As for the accuracy and kappa values, the algorithm described in this paper is superior to the classification algorithm of [7] in tackling this task with sharpness 0 to 3. For example, with sharpness 1—leaving out 66% of the features—the
5.4 Predicting Backend Defect Status of Chips

All results described up to now were obtained by trying to detect or to predict the frontend (FE) overall defect state "0" of chips from a series of wafers. A related problem is trying to predict the backend (BE) overall defect state "BEpass" knowing only frontend measurement data. The latter problem of BE defect prediction has turned out to be harder than FE defect prediction or detection to all algorithms we have tried.

Note. Whenever we write “predicting the BE overall defect state "BEpass"", what we actually are doing is predicting the negation of "BEpass". This is completely equivalent because the predictions are binary so negating the predicted bit converts the former into the latter and vice versa.
5.4.1 Predicting Backend Defect Status of Chips Knowing Frontend Data

In this application, input data consists of 150 measurements per chip taken at different frontend fabrication steps, and, as in the FE application, all chips are of the same product type. The task consists in predicting the BE overall defect state "BEPass" for a large sets of chips after training with two smaller sets of positive and negative samples, respectively. In addition, we wanted to know how many and which measurements could be omitted without reducing the quality of prediction unduly.

In the following two figures 7 and 8 we see plots created by Algorithm 6 while classifying 10000 chips of product D for BE overall defect status. See the explanations for Figure 2 and 3 in subsection 5.3.1 for details on the type of results these plots show. For this BE classification of 10000 chips of product D we set $b^+ = 0.41$ and used 60% of all positive and 10% of all negative chips as samples for training.

Figure 7: $S_C(i)$ in BE classification with product D, using all features.

![Graph showing number of conditions satisfied per object knowing only sampled distribution.]

5.4.2 Predicting Backend Defect States Knowing Only Part of Measurement Data

Next, we tried how far we can reduce the number of features without inducing unacceptable loss in quality of prediction. In result, we could shrink the input data to just 2 specific features out of 150. We used the same data lot and
the same amount of positive (60%) and negative (10%) chips for training as in subsection 5.4.1 above. The plots in Figure 9 and 10 are created by Algorithm 6 using only these 2 features. The following table shows example results for the aforementioned BE classification of 10000 chips of product D, using all 150 features or using just 2 or 3 specific ones.

| #Features | #Cics | TP/FP | TN/FN | Accuracy% | Kappa |
|-----------|-------|-------|-------|-----------|-------|
| 150       | 7     | ∞     | 36.4  | 97.4      | 0.638 |
| 3         | 3     | ∞     | 36.4  | 97.4      | 0.638 |
| 2         | 2     | 204   | 36.4  | 97.4      | 0.637 |

As can be seen in this table, if using all 150 columns, Algorithm 2 finds 7 Cics. When we replaced this by 3 or even 2 very specific columns, the prediction quality by Algorithm 6 was only marginally worse.

5.4.3 Addendum: Batch-Wise Cutoff Optimization

If measurement data of several wafers of the same product type is put together to some data lot, inhomogenities may occur, for example caused by parameter shifts or by accumulation of defects in certain wafers. As a consequence, optimizing the cutoff over a complete data lot containing the measurement data of—for example—some dozens of wafers, as Algorithm 5 does by default, may be too coarse. In order to explore this further, we performed a series of experiments by not optimizing the cutoff once over the whole series of wafers but by repeating Algorithm 5 on a sequence of batches each containing a fixed number of chips instead.

In the rest of this subsection, results relate to predicting the BE overall defect state “BEpass” of product D (34550 chips) using 70%/5% of all positive/negative
Figure 9: $S_C(i)$ in BE classification with product D, using only 2 features.

![Figure 9: $S_C(i)$ in BE classification with product D, using only 2 features.](image)

objects as samples for training, using 3 columns out of 150 as Cics as described in subsection 5.4.2. The following table shows the results of some sequences of runs of the classification Algorithm 3 when performing the optimization step by Algorithm 4 on a single batch instead of the complete lot. There is considerable variation in the prediction quality for separate batches, as the following table shows.

| Batchsize | 200  | 400  | 1000 | 1382 | 4000 |
|-----------|------|------|------|------|------|
| Max Kappa | 1.0  | 0.920| 0.729| 0.669| 0.619|
| Min Kappa | 0.0  | 0.196| 0.433| 0.423| 0.535|

Using a batch size of 1382 means wafer-wise cutoff optimization. Looking at how many batches could be classified with given kappa values, we obtained the following numbers.

| #Batches with $\kappa$ | Batchsize 200 | Batchsize 400 |
|------------------------|---------------|---------------|
| $\kappa = 1.0$         | 13            | 0             |
| $\kappa \in (0.9, 1.0)$ | 1             | 2             |
| $\kappa \in (0.8, 0.9)$ | 11            | 2             |
| $\kappa \in (0.7, 0.8)$ | 29            | 16            |
5.5 Classifying Iris Types

As a completely different application in multiple respects—object count, feature count and type of measurements—we tested Algorithm 6 with the classic iris flower data set [4] with the correction of [3]. It turns out that Algorithm 6 performs well on this data set despite of its small number of objects and features. The following results have all been obtained using nb = 5 or 6, 60%/1% positive/negative training set size and specifying columns #3 and #4 as Cics. The selection of Cics can also be done by prescribing b* (see Algorithm 2), with slightly worse quality of prediction.

| Type    | TP | FP | TN | FN | Kappa | Accuracy% |
|---------|----|----|----|----|-------|-----------|
| setosa  | 18 | 0  | 99 | 2  | 0.937 | 98.3      |
| versicolor | 13 | 1  | 98 | 7  | 0.727 | 93.3      |
| virginica | 13 | 0  | 99 | 7  | 0.756 | 94.1      |

In comparison, the classification algorithm of [7] performed well only when classifying two of the three types.

5.6 Dimensional Reduction

As rendered by examples in subsections 5.4.2 and 5.3.1, there are applications where not all features are necessary for chip classifications. Some features or even groups of features in the baselying measurement data may be eminently important whereas others may be dispensable without much degradation in prediction quality. We will be using the notations of section 3 here.

Let \( H_{a^*, nb} : \mathbb{Q}^m \rightarrow \mathbb{Q}^{nb} \) be the histogram function mapping \( m \) numbers to the relative frequencies \( (h_0, \ldots, h_{nb-1}) \) of these \( m \) numbers if distributing them to \( nb \) equidistant intervals \( I_0, \ldots, I_{nb-1} \) with bounds \( a^* \).
Definitions. Let $j$ be some column index in $\{1, \ldots, n\}$. Consider $(h_0, \ldots, h_{nb-1}) = H_{a^{*}, nb}(x_{+}, j)$ where $x_{+} = (x_{i,j}: i \in T^+) \in \mathbb{Q}^{|T^+|}$ is the $j$-th column of Scale($X$), reduced to those rows representing positive training objects. Let $k^{*}$ be the smallest index in $\{0, \ldots, nb-1\}$ with $h_{k^{*}} = \max(h_0, \ldots, h_{nb-1})$. Then $I_{k^{*}}$ is the (left-most) interval of those occurring with maximum relative frequency. Define

$$n_{pos}(j) = |\{i \in T^+: x_{i,j} \in I_{k^{*}}\}|$$
$$n_{neg}(j) = |\{i \in T^- : x_{i,j} \in I_{k^{*}}\}|$$
$$n_{diff}(j) = n_{pos}(j) - n_{neg}(j)$$

As follows from the definition of Cic in subsection 3.1.3, some column $j$ is suited for being used as one Cic in computing the indicators by Algorithm 3 if the property of $x_{i,j}$ lying in the most-frequent interval of the $j$-th column’s histogram can be used as a criterion separating as many as possible positive objects $(x_{i,1}, \ldots, x_{i,n})$ from negative objects. Since all the classification algorithm knows in training is the samples, we must take histograms of the columns reduced to the positive objects, $(h_0, \ldots, h_{nb-1}) = H_{a^{*}, nb}(x_{+}, j)$ as defined above.

So some column $j$ is suited for being used as one Cic in computing the indicators by Algorithm 3 if it has the following two properties.

- For many positive samples $(x_{i,1}, \ldots, x_{i,n})$ ($i \in T^+$), $x_{i,j}$ lies in the most-frequent interval $I_{k^{*}}$ of $H_{a^{*}, nb}(x_{+}, j)$.

- For few negative samples $(x_{\bar{i},1}, \ldots, x_{\bar{i},n})$ ($\bar{i} \in T^-)$, $x_{\bar{i},j}$ lies in the aforementioned $I_{k^{*}}$.

Therefore, those columns $j$ that satisfy these two criteria to a high extent possess high $n_{diff}(j) = n_{pos}(j) - n_{neg}(j)$ values which suggests using $n_{diff}(j)$ as a relevance indicator of column $j$.

As an example, here is the top of some table listing column indices $j$ ordered by decreasing $n_{diff}(j)$:

| Rank | $j$ | $n_{diff}$ | $n_{pos}$ | $n_{neg}$ |
|------|-----|------------|----------|----------|
| 1    | 144 | 844        | 879      | 35       |
| 2    | 145 | 841        | 899      | 58       |
| 3    | 142 | 836        | 903      | 67       |
| 4    | 141 | 831        | 897      | 66       |
| 5    | 143 | 825        | 901      | 76       |
| ...  | ... | ...        | ...      | ...      |
| 146  | 1   | 0          | 24       | 24       |
| 147  | 5   | -11        | 64       | 75       |
| 148  | 17  | -112       | 271      | 383      |
| 149  | 16  | -130       | 346      | 476      |
| 150  | 6   | -230       | 765      | 995      |
This table has been extracted while classifying product D for the BE overall defect state as described in 5.4.2. As is clearly visible, the 3 columns \( j = 142, 143, 144 \) occur among the top five ranks. These 3 columns were those that made it possible to use only 3 out of 150 columns as Cics with a negligible degradation of prediction quality in FE prediction—see subsection 5.3.1—and in BE prediction—see subsection 5.4.2.

5.7 Automatizing The Finding of Cics

The \( n_{\text{diff}} \) ranking detailed in section 5.6 can be used in order to try finding relevant features, i.e., features that should be taken into respect when computing the indicator values \( S_C(i) \) by Algorithm 8. The resulting classification algorithm is the same as Algorithm 6 with the only difference being that the step

\[
C := \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, \text{nb}) \text{ by Algorithm 2}
\]

is replaced by

\[
C := \text{AutoCics}(\text{Scale}(X), T^+, T^-, \text{nb}, t) \text{ by Algorithm 7}
\]

which refers to the following algorithm. Let \( t \) be some number in \( \{1, 2, \ldots, n\} \) where \( n \) is the number of columns of the input data matrix \( X \).

**Algorithm 7: AutoCics(\( X, T^+, T^-, \text{nb}, t \))**

If \( t \) is set to some number in \( \{1, \ldots, n\} \) then Algorithm 7 returns this many columns with the highest \( n_{\text{diff}} \) values and thus can be plugged into the classification algorithm as an alternative for computing \( C \) by Algorithm 2.

5.7.1 Practical Results by Using AutoCics()

In order to get an impression of the practicability of the AutoCics() variant of Algorithm 6 we tested the degree of stability of the column list \( L \) computed by the AutoCics() Algorithm 7 against changing training set selection, training set size and changing data lots of the same product type.
In order to compare the AutoCics() variant with results mentioned earlier in this paper, we repeated the task as described in section 5.3.2: predict the S1 overall defect state of 10000 chips of Product B using minimal training sets of only 2 positive and 1 negative samples. But unlike 5.3.2, we do not control the selection of Cics by $b^+$ but by the parameter $t$ of the AutoCics() algorithm. $b^+$ and $b^-$ are not used here.

The following table shows results for three different settings of the parameter $t$—called “#Top Cics Used” in this table—in Algorithm 7. When setting $t = 50$, the difference in detection quality is small with Kappa changing from 0.983 to 0.980 and FP+FN increasing from 10+12 to 2+23. Because of these tiny training sets, only $50 \cdot 3$ numbers of the matrix Scale(X) are used for classifying 10000 chips here.

| #Top Cics Used | FP  | FN  | Accuracy% | Kappa |
|---------------|-----|-----|-----------|-------|
| 50            | 2   | 23  | 99.7      | 0.980 |
| 45            | 81  | 23  | 99.0      | 0.921 |
| 40            | 277 | 23  | 97.0      | 0.799 |

5.7.2 Obsoleting Free Parameters by Using AutoCics()

Using AutoCics() by Algorithm 7 offers the possibility to get rid of the necessity of controlling the selection of Cics by the user. We can specify a default value $t^*$ for $t$, derived from $n$ (= number of columns of $X$), which is to be used if the user does not specify $t$ himself. This makes it possible to do without specifying $b^+$, $b^-$ in Algorithm 6 as well as $t$ in Algorithm 7.

We set $t^* = \lceil 0.1 \cdot n \rceil$ so that the top 10% ranks of the $n_{diff}$ table described in sections 5.6 and 5.7 are used as indicator columns.

The following table shows results of the setting described in 5.7.1, but this time $t^*$ is used for $t$. As Product B contains 332 measurements per chip, $t^*$ is 34 here. We use 5 chips for training, 3 positive samples and 2 negative ones.

| TP  | 657 |
| FP  | 13  |
| TN  | 9301|
| FN  | 25  |
| TP/(TP+FN)% | 96.3 |
| TN/(TN+FP)% | 99.9 |
| FP/(TP+FN)% | 1.9 |
| FN/(TN+FP)% | 0.3 |
| TP/FP | 50.5 |
| TN/FN | 372.0 |
| Accuracy% | 99.6 |
| Kappa | 0.970 |
6 Algorithmic Complexity

In what follows, we count operations $+,-,\times,\div,\leq,\lfloor \cdot \rfloor$ and $\sqrt{\cdot}$ at unit cost for all occurring numbers. This is a realistic model when using fixed-width rational number types like 64-bit floating point on a real computer. By this, we assume a somewhat idealized computer by ignoring certain limitations of data types frequently used for unit-cost- (or constant-cost)-arithmetics on real computer hardware: limited value range, rounding errors in representing rational numbers whose denominator is not a power of 2, approximations of $\sqrt{\cdot}$ etc. Furthermore, we ignore costs associated with accessing and modifying simple data structures—mainly, accessing single numbers, vectors and matrices of numbers in memory. Counting these may increase costs by polylogarithmic factors, depending on the chosen data structures, the input size and the computer hardware the user is going to use.

6.1 Scaling (Algorithm 1)

For $X \in \mathbb{Q}^{m \times n}$ we need to compute $n$ means and $n$ standard deviations in $O(m \cdot n)$, and compute $\frac{x_{i,j} - \mu_j}{\sigma_j}$ for all $i,j$. Altogether, this takes $O(m \cdot n)$ arithmetic operations, including $n$ square roots.

6.2 Computing one histogram $H_{a,nb}(x_1, \ldots, x_m)$

Let $a = (-\infty < a_1 < \cdots < a_{nb-1} < \infty)$.  

Case 1: The inner boundaries $a_1, \ldots, a_{nb-1}$ are chosen by the user: For every $i$ we find the unique bin $[a_k, a_{k+1})$ which contains $x_i$ by binary search on the boundaries vector $a$, which takes $\log(nb)$ comparisons plus incrementing one of $nb$ counters. So the overall complexity is in $O(\bar{m} \cdot \log(nb) + nb)$.  

Case 2: The inner boundaries $a_1, \ldots, a_{nb-1}$ are fixed like when using $a^*$: Let $x = (x_1, \ldots, x_m)$. For every $i$ we can compute the unique bin $[a_k, a_{k+1}) (k \geq 1)$ containing $x_i$ by first computing the constant bin width $w = \frac{\max(x) - \min(x)}{nb-2}$, then $k = 1 + \left\lfloor \frac{x_i - \min(x)}{w} \right\rfloor$. Obviously, $k \in \{1, \ldots, nb-1\}$ and $x_i \in [a_k, a_{k+1})$. The left-most interval $[a_0, a_1) = (-\infty, a_1)$ stays empty. We need to compute $\min(x)$ and $\max(x)$ only once and have $O(1)$ arithmetic operations plus incrementing one of $nb$ counters for every $x_i$ which sums up to $O(\bar{m} + nb)$ operations altogether.

Note that in our applications $\bar{m}$ is $|T^+|$ or $|T^-|$ which are both $\leq m$ (= row number of $X$).

Note that the right-most inner interval must be $[a_{nb-2}, a_{nb-1} + \varepsilon)$ instead of $[a_{nb-2}, a_{nb-1})$ for $a_{nb-1} = \max(x_1, \ldots, x_m)$ for formal reasons, see subsection 3.1.2.
6.3 Computing $\text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ (Algorithm 2)

For every column index $j = 1, \ldots, n$: Computing $\min(x^+)$ and $\max(x^+)$ takes $O(|T^+|)$ operations, thus computing $a^*$ is in $O(|T^+|)$ operations. Computing the histograms $h^+$ and $h^-$ takes $O(|T^+|) + O(|T^-|) + O(nb)$ operations according to Case 2 of above. Computing $h^+_{\text{max}}$ and $k^+$ takes $nb$ comparisons at most.

The output consists of at most $3 \cdot n$ numbers.

So computing $\text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ from $\text{Scale}(X)$ takes at most

$$O(n \cdot (|T^+| + |T^-| + nb))$$

operations.

**Remark.** This is in $O(n \cdot (m + nb))$ since $|T^+| + |T^-| \leq m$.

**Theorem.** Computing $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ from $X \in \mathbb{Q}^{m \times n}$ takes at most

$$O(n \cdot (m + nb))$$

operations.

**Proof.** This follows directly from the complexity of computing $\text{Scale}(X)$, then computing $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ from $\text{Scale}(X)$ and from the above Remark, qed.

6.4 Computing $S_C(I_{\neg T})$ (Algorithm 3)

Given $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$, for each one of $I_{\neg T}$ non-training objects we must perform at most $|C|$ interval containment decisions which sums up to

$$|C| \cdot O(|I_{\neg T}|)$$

operations.

6.5 Computing predictions $F(i, c)$ (Algorithm 4)

Given $S_C(I_{\neg T})$ and $c$, this sums up to

$$O(|I_{\neg T}|)$$

operations for all $i \in I_{\neg T}$.

6.6 Naive cutoff selection

Given $I^+_{\neg T}$, $I^-_{\neg T}$ and $S_C(I_{\neg T})$ from Algorithm 3, Computing $Av^0, Av^1$ takes

$$O(|I^+_{\neg T}| + |I^-_{\neg T}|) \subseteq O(|I_{\neg T}|)$$

operations.
6.7 Optimizing cutoff $c$ (Algorithm 5)

Let $Q : \mathbb{B}^r \times \mathbb{B}^r \rightarrow \mathbb{R}$ be a function measuring some type of similarity of two 0-1 vectors and let be $E(r)$ the maximum number of operations it takes to evaluate $Q(v,w)$ for any $v,w \in \mathbb{B}^r$. Given $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ from Algorithm 2, Algorithm 5 repeats $(S_C^{\text{max}} - S_C^{\text{min}} + 1)$ times Algorithm 4 for obtaining a prediction vector $F(c)$ using $O(|I\neg T|)$ operations each, and evaluates $Q(\cdot, \cdot)$ on two 0-1 vectors of length $r = |I\neg T|$ using at most $E(|I\neg T|)$ operations each. From $S_C^{\text{min}} \geq 0$ and $S_C^{\text{max}} \leq |C|$ follows $S_C^{\text{max}} - S_C^{\text{min}} + 1 \leq |C| + 1$. Computing $Q_{\text{opt}}$ and $C_{\text{opt}}$ both take $O(S_C^{\text{max}} - S_C^{\text{min}} + 1)$ as $c$ takes just this many values in the for-loop.

So the total number of operations Algorithm 5 takes is limited by

$$(|C| + 1) \cdot O(|I\neg T| + E(|I\neg T|)) \leq O(|C| \cdot (|I\neg T| + E(|I\neg T|)))$$

6.8 Overall complexity

Adding the upper bounds for Scale($X$), for computing $\text{FindCics}(\text{Scale}(X), \ldots)$ and for computing $S_C(I\neg T)$ gives an operation count of

$$O(n \cdot m + n \cdot (m + nb)) \leq O(n \cdot (m + nb + |I\neg T|)) \leq O(n \cdot (m + nb + E(|I\neg T|)))$$

(since $|C| \leq n$)

(since $|I\neg T| \leq m$)

If we add steps for cutoff selection or optimization and use $|I\neg T| \leq m$ twice we get:

$$O(n \cdot (m + nb)) \quad \text{with naive cutoff selection}$$

$$O(n \cdot (m + nb + E(|I\neg T|))) \quad \text{when optimizing cutoff } c$$

Using $|I\neg T| \leq m$ once again, we get:

**Theorem.** Given $X \in \mathbb{Q}^{m \times n}$, $v \in \mathbb{B}^m$, training sets $T^+, T^- \subseteq \{1, \ldots, m\}$, $b^+, b^- \in \mathbb{Q}, nb \in \mathbb{N}_{\geq 3}$, the overall number of operations it takes to

- compute Scale($X$)
- compute $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$
- compute $S_C(I\neg T)$ (where $I\neg T = \{1, \ldots, m\} \setminus (T^+ \cup T^-)$)
- select cutoff $c$
- compute predictions $F(i,c)$ from this by Algorithm 4

is limited by

$$O(n \cdot (m + nb)) \quad \text{with naive cutoff selection}$$

$$O(n \cdot (m + nb + E(m))) \quad \text{when optimizing cutoff } c$$
where $E(m)$ is the number of operations it takes to evaluate the chosen similarity measure $Q : \mathbb{E}^m \times \mathbb{E}^m \rightarrow \mathbb{R}$.

**Note.** The direct input of Algorithms 3 and 5 is $C$, not $X$ or Scale($X$), so their complexity is actually linear in the number of Cics $|C|$. In the worst case $|C|$ equals $n$ but in practical settings $|C|$ is frequently much smaller than $n$, for example 20 versus 150. So Algorithms 3 and 5 become the faster the smaller the number of Cics $|C|$ is. Indeed, the implementation on a real computer shows that $|C|$ is an important determinant for time consumption.
7 Uniting Two Classification Algorithms

There are several ways to combine multiple raters with the goal of obtaining a better rater. Some well-known ways are making majority decisions, or making a decision according to the number of the “1” (or “0”) of the single raters exceeding a certain threshold or not. What we are doing in this section instead is combining the baselying principles of two classification algorithms—Algorithm 6 and the one of [7]—into one algorithm.

In this section, we will use the same notations and designations as in section 3 Definition.

We will say that a Cic \( j \) is active for object \( i \) represented by \( x_i, 1, \ldots, x_{i,n} \) if in Algorithm 3, \( x_{i,j} \in [a_{k^+}, a_{k^+}+1) \). Here, \( i \in \{1, \ldots, m\} \), \( j \in \{1, \ldots, n\} \).

In \( S_C(i) \) computed by Algorithm 3, the number of active Cics for object \( i \) is used as an indicator for positivity of object \( i \). Actually, there is no reason for just counting and not trying to go further. In order to be more specific, one could replace \( S_C(i) \) by the bit pattern induced by the active Cics for object \( i \).

Definition. Let

\[
A_C(i) : \{1, \ldots, m\} \to \mathbb{B}^{|C|}
\]

\[
i \mapsto a_i
\]

where

\[
a_i(j) = \begin{cases} 
1 & \text{if Cic } j \text{ is active for object } i \\
0 & \text{else}
\end{cases}
\]

Now let be \( U^+ \subseteq I^+ \) some set of positive objects we will use as a third set of training indices along with \( T^+, T^- \). We need to reduce \( I_{-T} \) defined in subsection 3.1.2 to

\[
I_{-T,-U} = \{1, \ldots, m\} \setminus (T^+ \cup T^- \cup U^+)
\]

in order to skip the objects of three training sets in the classification loop.

Let be \( \langle a, b \rangle = \sum_{k=1}^{r} a_k \cdot b_k \) the inner product of two vectors of size \( r \), and \( H(a) = \sum_{k=1}^{r} a_k \) the Hamming weight of \( a \in \mathbb{B}^r \).

Definition. Let be \( i \in \{1, \ldots, m\} \) and \( U^+ \) as above.

\[
q_{\max}(i, U^+) := \max_{j \in U^+} \frac{\langle A_C(i), A_C(j) \rangle}{H(A_C(i))}
\]

\[
q_{\min}(i, U^+) := \min_{j \in U^+} \frac{\langle A_C(i), A_C(j) \rangle}{H(A_C(i))}
\]

Remark. \( q_{\max}(i, U^+) \) is in strict analogy to what is denoted by \( s(x; T) \) in subsection “Dense Formulation” of section “The Algorithms” in [7], the difference being that in [7] we operate on bit patterns \( (P_1(x^{(i)}), \ldots, P_n(x^{(i)})) \) induced by component-wise thresholding \( x^{(i)} = i \)-th line of the scaled input matrix, whereas by computing \( q_{\max}, q_{\min} \) here we operate on the Cic activity pattern of line \( i \) of the (scaled) input matrix.
Definition. As in subsection 3.1.4, let $C = \text{FindCics}(\text{Scale}(X), T^+, T^-, b^+, b^-, nb)$ be the output of Algorithm 2. Define two alternative indicators by:

$$Q^\text{max}_{C, U^+} : I \rightarrow [0, 1] \subseteq \mathbb{Q} \quad i \mapsto q^\text{max}(i, U^+)$$

$$Q^\text{min}_{C, U^+} : I \rightarrow [0, 1] \subseteq \mathbb{Q} \quad i \mapsto q^\text{min}(i, U^+)$$

The following algorithm is formulated in a way that it can be used as a drop-in replacement for Algorithm 3 in subsection 3.1.4.

Algorithm 8: Computing alternative indicators $Q^\text{max}_{C, U^+}(i), Q^\text{min}_{C, U^+}(i)$

We can now formulate a variant of the classification Algorithm 6 of subsection 3.1.7 which uses $Q^\text{max}_{C, U^+}(I \sim T, \sim U), Q^\text{min}_{C, U^+}(I \sim T, \sim U)$ as indicators instead of $S_C(I \sim T)$.

Algorithm 9: Classification algorithm

By “an adaption of Algorithm 5 and Algorithm 4” we mean the following two modifications:

- In Algorithm 5 replacing the looping $c = S^\text{min}_C, S^\text{min}_C + 1, \ldots, S^\text{max}_C$ by looping over some grid $C \subset [0, 1] \subset \mathbb{Q}$ or by the method of subsection “Refined Method For Cutoff Selection” in \[7\].
In Algorithm 4 (called by Algorithm 5), replacing the computation of $S_C(i)$ by $Q^{\text{max}}_{C,U^+}(i)$, $Q^{\text{min}}_{C,U^+}(i)$ as defined above.

The potential of Algorithm 9 has yet to be explored and will be a subject of future research.
8 Conclusion

We have presented a histogram-based algorithm for object classification in subsection 3.1.7 together with some worst-case complexity analysis in section 6 and rendered the detection and prediction of frontend and backend defects in semiconductor chips based on wafer measurement data as main application by discussing classification results of data from real-world wafer fabrication in section 5. Furthermore, we laid out a method of searching for candidate indicator columns in subsection 5.6 which enables the feature count to be reduced dramatically in some applications. We presented an algorithm using $n_{\text{diff}}$ as a feature relevance indicator in section 5.7.

Finally, we presented an algorithm that unites some baselying principle of the classification algorithm of [7] with the principles of the classification Algorithm 6 as detailed in this paper.

In subsection 5.4 of this paper, we have started tackling the problem of predicting the backend overall defect states of semiconductor chips which has been mentioned in the last paragraph of [7] as one topic of ongoing research. The results which we report on in section 5 confirm that predicting the backend overall defect state for the type of data our classifications are performed on using only frontend measurements seems to be harder of a machine learning problem than predicting the frontend overall defect state.
List of Figures

| Figure | Description                                                                 | Page |
|--------|-----------------------------------------------------------------------------|------|
| 1      | $S_C(i)$ in the S2 classification with product C.                           | 18   |
| 2      | Cutoff optimization of the S2 classification with product C.                | 19   |
| 3      | Cutoff optimization of the S1 classification with product B.                | 21   |
| 4      | Histogram collection of 5 features, knowing all objects.                    | 22   |
| 5      | Histogram collection of 5 features, knowing 20%/10% sample sets only.       | 23   |
| 6      | Histogram collection of 5 features, knowing 2+2 sample objects only.        | 24   |
| 7      | $S_C(i)$ in BE classification with product D, using all features.           | 25   |
| 8      | Cutoff optimization of BE classification with product D, using all features.| 26   |
| 9      | $S_C(i)$ in BE classification with product D, using only 2 features.         | 27   |
| 10     | Cutoff optimization of BE classification with product D, using only 2 features. | 28   |

References

[1] Richard M. Brugger. A Note on Unbiased Estimation of the Standard Deviation. *The American Statistician*, 23(4):32, 1969.

[2] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[4] Ronald Aylmer Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Martin H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[6] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[7] Thomas Olschewski. Fast Accurate Defect Detection in Wafer Fabrication. https://arxiv.org/abs/2108.11757v1, 2021.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.