**Silicon Wafer Defect Classification**

Carson Branham

CS 4375.003 – Intro to ML

Professor Rishabh Iyer

I. Introduction

Silicon wafers, thin slices of semiconductors no greater than 3/4mm thick, play a crucial role in modern computing. Utilizing their unique properties of being either conductive or nonconductive to transmit electricity based on digital input, these wafers are essential in the production of integrated circuits. A single wafer can be worth as much as $10,000 and yield hundreds of integrated circuits for a variety of applications. Without integrated circuits, personal computers and many other electronic devices would not be feasible as computers would likely still be multi-ton devices using vacuum tubes for computation. [4]

The process to create functional ICs from a blank wafer takes hundreds of steps over the course of months - throughout this process various factors such as contamination, defects, and human error can lead to chip failure. [3] Due to this, thorough testing is required. Traditionally human operators must go through the wafer failure maps generated by these tests to categorize the type of failure into fail patterns. Fail patterns show not only which chips are faulty but also provide invaluable insight into where the failure occurred in the production process. Automating the classification of failures will speed this analysis up and increase throughput of the production line, allowing manufacturers to identify and address issues more quickly.

II. Dataset Analysis

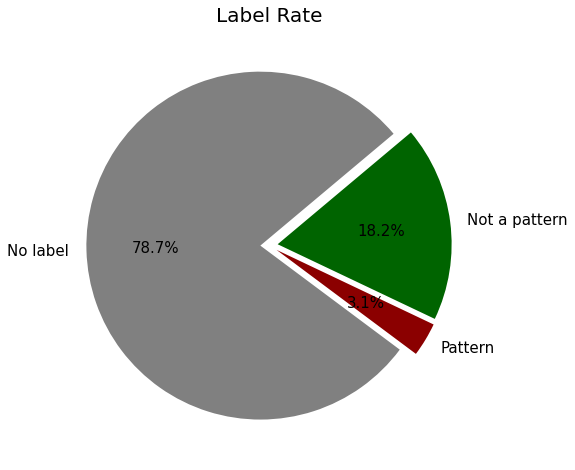
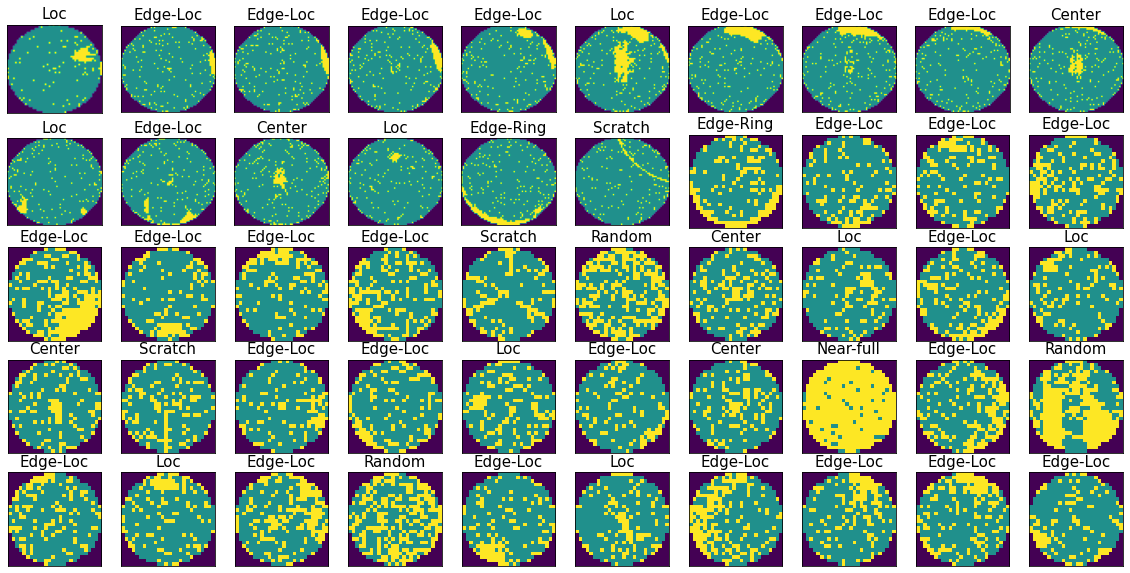
To achieve this, the dataset WM-811K [1] was selected. It was not only the most reliable dataset, but the largest available publicly. Studying the characteristics of the data we can quickly see that the waferIndex column is unnecessary for the purposes of our study so it can be safely dropped. In addition, lotName was considered momentarily as a candidate to be dropped however it is reasonable to assume that wafers within the same lot, especially those nearby each other, may be subjected to the same production defect meaning this could prove useful for future use. The dataset boasts a substantial 811,457 wafer maps. Though upon inspection only 172,950 of these are actually labelled. Additionally, of those labelled a mere 25,519 have failure patterns. Using this data, we create a new data frame with only labelled wafers that have a failure pattern. We have now reduced the amount of data in the data frame to 3.1% of its original size, greatly increasing the speed at which operations performed on it can run at. (see fig. 1) Continuing to analyze the columns, waferMap is a large array containing the values 0, 1, and 2 representing empty, working chip, and defective chip respectively. This data looks incomprehensible within the code but it’s relatively simple to use it to visually represent our samples. To begin with we will go ahead and draw the first 50 samples within our dataset. (see fig. 2)

Fig. 1: Amount of data containing labels and patterns.

Fig. 2: First 50 samples that have labels & a fail pattern



With the ability to easily visualize what each pattern looks like we can now reasonably see with our own eyes what the classification for each fault pattern is. (fig. 3) With this in mind let us also take a look at how many samples of each pattern we have in our dataset. (fig. 4) Going forward we should keep in mind that we have significantly less data for donut, random, scratch, and near-full patterns as prediction accuracy for them could be affected by this.

Fig. 4: Amount of each pattern in our dataset

Fig. 3: Ideal fail pattern for each label

A graph of different sizes of blue bars

Description automatically generated with medium confidenceA group of images of different colored circles

Description automatically generated

III. Feature Extraction

Dataset now analyzed; we understand the information we are working with. It is time to use waferMap to derive features for our machine learning algorithms. The first and most obvious method of doing so is by dividing wafers into regions. We can divide the wafers into 13 regions, 9 in the middle forming a 3x3 grid and 4 on the sides. This division will facilitate the capture of any local clustering around the center or edges more than anything else. (fig. 5) Then, theoretically based on the density of failures in each of the regions, our machine learning algorithm can work to determine the label. Examining fig. 6, we can see that this method will prove very effective at classifying center, both loc, and near full. Issues arise when we start to look at the others, especially scratch which is nearly impossible to classify using this method alone as it involves a near-random distribution across multiple regions. It is clear that we will have to introduce more features into our dataset before implementing any machine learning algorithms.

A grid of squares in a circle

Description automatically generatedA group of images of different shapes

Description automatically generated with medium confidence

Fig. 5: Region division

Fig. 6: Region weights on the different patterns

Our next feature set to derive is geometry-based features: the idea of reducing noise and then isolating the most common values in regions. [2] Using Radon transformation – a variant of Fourier transformation, we can map projections of the values present in each pattern across all wafers of that pattern (fig. 6) and isolate the geometric shapes that are relevant to that pattern. (fig. 7) This is especially useful for scratches as interconnectedness is key. Once we have done so we look at the formed shape’s area, perimeter, length, width, solidity, and eccentricity. These values make up for our geometry-based features. Thinking back to our scratch example, they will generally be long but thin shapes with little area so the model we build will be able to see this and group all of such shapes together. Geometric features make up for our shortcomings in region density features and we are now ready to give features to our model.

Fig. 7: Geometric isolation

Fig. 6: Radon maps

A screenshot of a video game

Description automatically generatedA screenshot of a computer generated image

Description automatically generated

IV. Machine Learning Model

K-nearest neighbors (KNN) algorithm was chosen for this project because of its usefulness in classification tasks. It does not assume any distribution of the data and more importantly is robust to noise and outliers in the data. We did use the radon transformation previously to get around noise in one of our feature sets but there is always going to be some amount of noise. Figure 2 clearly shows that sometimes there is so much noise it can even be difficult for a human to classify the fault pattern. The initial selection of k used was 5 as it strikes a balance between bias and variance. This first selection proved to outperform the first tests of SVM that were made to compare the two which was a large deciding factor. To ensure that we were getting the highest possible accuracy we compared every k value between 0 and 30 – through this process it was found that 9 was the perfect balance. (Fig. 8) Not satisfied with 83.75%, there was a final effort made to improve accuracy by implementing Weighted KNN. This resulted in 83.97% which seemed to be as good a result as KNN can produce. To see where exactly our model was predicting incorrectly, a confusion matrix was created. Looking at fig. 9 we can see the exact number predicted for each pattern. The least amount of data and most accurate (with 100% prediction) was near-full while center and edge-ring ended up with significantly more data and yet also highly accurate! Our greatest shortcoming ended up being scratch – which was already predicted to be a problem. Scratches were commonly classified as edge-loc perhaps due to the fact that there were indeed A screenshot of a computer

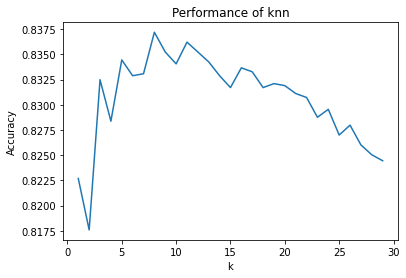
Description automatically generatedmany more edge-loc in the dataset than scratch samples.

Fig. 9: Confusion Matrices

Fig. 8: Ideal K selection

V. Concluding Notes

Regarding my final accuracy I feel that further feature engineering and data augmentation could greatly improve the performance of the model. Even without more features, simply more data on the less common patterns would also help to boost the accuracy rate, as seen in the confusion matrix. These less common patterns also seem to be the most difficult to classify as simply one or the other by both humans and machines. Perhaps if one wafer could have multiple patterns like in the OPTICS paper [2] previously referenced such problems would become less of a concern. Unfortunately, the training data provided did not have multiple possible classifications for individual wafers.

This project was an incredible learning experience, particularly in the realm of feature generation. It was exciting taking visual data and dividing it up into meaningful regions for the algorithm to be able to use it. Mapping the noise generated by each pattern was also intriguing and helped me understand Fourier/sigmoid quite a bit more. This was a very rewarding endeavor and even if the final accuracy did not reach 85% like I had wanted I did obtain valuable knowledge of the real-world applications of machine learning.

References:  
[1] “WM-811K” MIR Corpora. <http://mirlab.org/dataSet/public/>

[2] Mengying Fan, Qin Wang and B. van der Waal, "Wafer defect patterns recognition based on OPTICS and multi-label classification," 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, China, 2016, pp. 912-915, doi: 10.1109/IMCEC.2016.7867343.

[3]

“Silicon Wafer Manufacturing: From Sand to Silicon | Wafer World,” www.waferworld.com. https://www.waferworld.com/post/silicon-wafer-manufacturing-from-sand-to-silicon#:~:text=Definition%20of%20Silicon%20Wafer%20Processing&text=The%20process%20involves%20converting%20raw (accessed May 08, 2024).

‌ [4]

“What is a silicon wafer? What is it used for?,” WaferPro, May 28, 2017. https://waferpro.com/what-is-a-silicon-wafer/

*As an aside the mapping for failure name to failure number along with a short description of what each looks like are as follows:*

* ***Center*** *– 0 – faults common in wafer center*
* ***Donut*** *– 1 – faults form a ring around the center*
* ***Edge-Loc*** *– 2 – faults centralized in one edge location*
* ***Edge-Ring*** *– 3 – faults spread out along an edge*
* ***Loc*** *– 4 – faults centralized in one non edge location*
* ***Random*** *– 5 – no pattern found*
* ***Scratch*** *– 6 – faults form at least one line*
* ***Near****-****full*** *– 7 – faults nearly make up entirety of the wafer*
* ***None*** *– 8 – no faults*