Homework 3: Wafer Map Failure Pattern

Due Date: Sunday, November 9th, 2025, 11:59PM

Start early!

Introduction

Homework 3 is slightly different from the previous assignments. In earlier assignments, your goal was to craft a **single precise prompt**. This time, however, your goal is to **leverage an LLM to complete a long piece of code**. Your task is to directly call your desired LLM from within the provided `ipynb` file to complete the code yourself. The notebook also includes a motivating story and helpful guidelines, so please follow the flow step by step as you build your solution.

In addition to code generation, if any questions come to mind while working through the process, you should also include those questions in the `ipynb` file by making API calls to the LLM.
The grading for this assignment will focus on two main criteria:

1. Whether you successfully completed the code using the LLM effectively.
2. What kinds of questions you asked the LLM during the process.

You may use any LLM you want. Of course, since you'll be calling the LLM directly within your .ipynb file, there is no limit on the number of LLM calls you can make. You will not lose points for asking questions that are too simple or for asking similar questions multiple times. So don't try to include only "perfect" questions and answers. Instead, include as many of your real questions as possible that came up while you were working on the assignment. There are also **no submission limits** on your `ipynb` and `csv` files.

In homework 3, we will introduce you to the WM-811K wafer dataset[1], which is real-world data produced at a fab. This is a well-known dataset used for wafer failure analysis research in recent years. Unlike the second homework, where you are given processed features, this homework requires you to perform feature engineering to define and extract features from wafer failure maps. From your features, you will build a decision tree model and Support Vector Machine (SVM) model to evaluate how well those features function for classification.

Background: Wafer Map Failure Pattern Analysis

In the semiconductor manufacturing process, batches of functional circuits are fabricated on a slice of silicon material which we call a *wafer*. After going through complicated test stages, the wafer is cut into pieces, each containing one copy of the circuit. Each of these pieces is called a die. At companies, one of the important factors for their revenue is ensuring a high *yield* of their manufactured wafers. *Yield* is the fraction of dies on the wafers that are not discarded during the manufacturing process due to some defect. The *yield per wafer* is quantified as the number of

working (good/passing) dies divided by the total number of dies fabricated on that wafer.

Wafer maps provide visual patterns of the bad/failing dice on the wafer. By analyzing the failure pattern on a wafer, experienced engineers can often identify issues in manufacturing and/or test processes that caused the failure, then take actions accordingly. Thus, these wafer maps are important for increasing the quality of production and ultimately increasing profit.

Dataset

We have taken a subset of the original data as training data of this homework (wafermap_train.npy). It contains 1693 real-world wafers and their attributes as shown in Table 1. There are 5 types of known labels: `Center`, `Edge-Loc`, `Scratch`, `Donut`, and `Near-full`.

Column	Type	Description
dieSize	float	the number of dice on a wafer
failureType	string	the type of failure pattern from 5 categories: "Center", "Edge-Loc", "Scratch", "Donut", "Near-full"
lotName	string	the manufacturing lot name of a wafer
trainTestLabel	string	denotes whether the sample is for training or testing in the original dataset
waferIndex	integer	denotes the wafer index within a lot
waferMap	numpy array	contains wafer maps *of varying size*. 0s denote no dice, 1s denote passing dice, 2s denote failing dice

Table 1: Real-world wafers [1]

Tasks

For this homework, we will perform feature engineering to extract features from the wafer maps. We will use these features to build decision tree and SVM models to classify the wafer map failure patterns.

Dataset Inspection

Before we dive into designing the features, it is important to inspect the dataset and visualize the failure patterns.

- Implement a function that plots an example wafer for each failure type.
- Implement a function that outputs all wafer maps to corresponding failure type directories. This allows you to simply click through to see all the wafer maps of each failure type.

Data Preparation (Part 1)

After inspecting the dataset, you should see the following problems:

- The dieSize column varies, meaning the wafer maps have different shapes. Implement a function that takes a wafer map and resizes it to (64, 64).

- The failureType is a string type, but the model training requires numerical number. Implement a function that takes the failure type string and converts it to numerical value.

Feature Engineering

Now that we have a better understanding of the failure patterns, we will start creating features for the wafer maps. Most of the features we implement will be based on the salient region of the wafer map. Salient region (for this homework) is defined as the 8-neighborhood connected component with the largest area. We will implement the ten features listed below:

- Area Ratio: Ratio of the area of the salient region to the area of the wafer map
- Perimeter Ratio: Ratio of the perimeter of the salient region to the radius of the wafer map
- Max Distance from Center: Maximal distance between the salient region and the center of the wafer map
- Min Distance from Center: Minimal distance between the salient region and the center of the wafer map
- Major Axis Ratio: Ratio of the length of the major axis of the estimated ellipse surrounding the salient region to the radius of the wafer map
- Minor Axis Ratio: The ratio of the length of the minor axis of the estimated ellipse surrounding the salient region to the radius of the wafer map
- Solidity: The proportion of failed dice in the estimated convex hull in the salient region
- Eccentricity: The shape of the estimated ellipse surrounding the salient region, where the value is 0 for a circle, or 1 for a line
- Yield Loss: Ratio of the failed dice on the wafer map to the total number of dice on the wafer map
- Edge Yield Loss: Ratio between the number of failing dice within two pixels of a wafer edge to the total number of dice within two pixels of a wafer edge.

Note: many of these functions can be implemented easily with the Scikit-Image library.

Data Preparation (Part 2)

Select the features defined above as your dataset and prepare the dataset for training and validation as we did in the previous homework. Do not normalize your data, as DecisionTree does not benefit from data normalization, and we would like the DecisionTree's rules to tell us the exact splitting values of our important features.

Model Training and Validation (Decision Tree)

We will train a decision tree model on the training set, evaluate its performance on the validation set, and analyze which feature is important based on the decision tree plot. For simplicity, you can start with a decision tree depth of 3 and increase it as needed. For this part you will need to show the following:

- **Per failure type accuracy for the training samples and validation samples**.
- **Confusion matrix for training samples and validation samples. (confusion matrix should be labeled using the original failureType string)**
- **The decision tree plot.**

Model Testing (Decision Tree)

Load the test dataset, preprocess it, predict the failure type with decision tree model, output prediction to a `dt_scores.csv` with a single column, `failureType`, and submit it to Gradescope.

Model Testing (Support Vector Classifier)

We will train a support vector classifier (SVC) model on the training set, evaluate its performance on the validation set.

- **Per failure type accuracy for the training samples and validation samples**.
- **Confusion matrix for training samples and validation samples. (confusion matrix should be labeled using the original failureType string)**

Model Testing (Support Vector Classifier)

Load the test dataset, preprocess it, predict the failure type with SVC model, output prediction to a `svc_scores.csv` with a single column, `failureType`, and submit it to Gradescope.

What to turn in

- dt_scores.csv and svc_scores.csv: CSV files with one column, failureType, containing your models' predictions for the wafers in wafermap_testing.npy.
- .ipynb file: Fill in the code and call LLM APIs to ask questions.

References

[1] Jyh-Shing R. Jang Ming-Ju Wu and Jui-Long Chen. *MIR-WM811K*. 2015. URL: `http://mirlab.org/dataset/public/`.