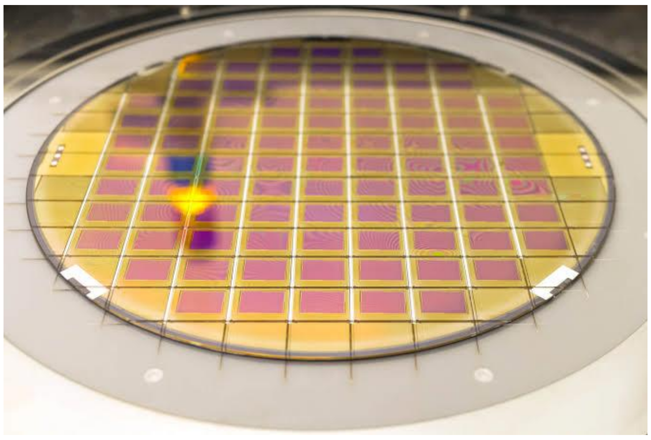


WAFER FAULT DETECTION

Machine Learning Project

ABSTRACT

The purpose of this project is to build a machine learning model that can effectively predict the faulty wafers by using the data given by the sensors in the wafers.

Sai Srinivas

Image Source: www.waferworld.com

Contents

Problem Statement

The goal is to build a programme to predict the faulty wafers by using classification methods in Machine Learning with the data provided by the client.

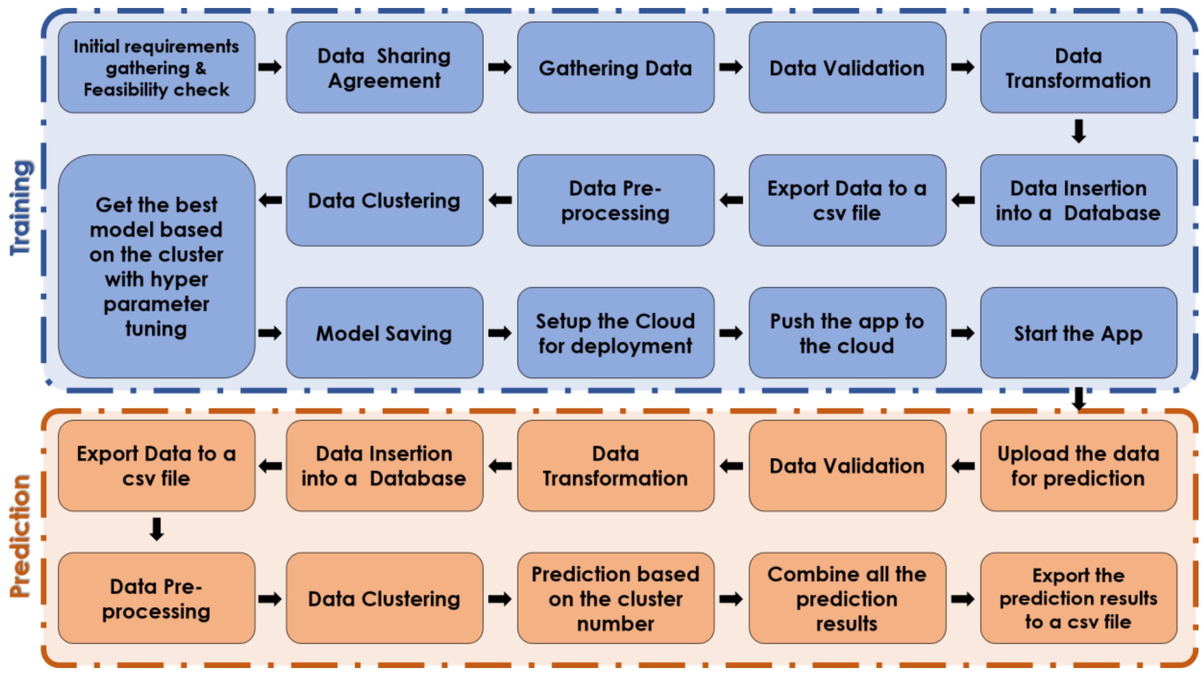
Wafer

Wafer (in electronics) is a thin slice of semiconductor, and generally in circular forms and each wafer is divided into a grid-iron format, the number of sensors we have in our dataset is the number of square we could get in each wafer. As far as this project is concerned, we are only taking same sized wafers as we want the number of columns in the project to be the same i.e. 590 columns.

Data Sharing Agreement (DSA)

In the data sharing agreement, the client mentions the number of files, mode of data to be transferred, feature details, language in which the data will be transferred and other specifications. The client should agree that whatever specifications of the data he mentioned in the DSA, should be transferred accordingly.

Architecture


Training
Initial requirements gathering & Feasibility check
Data Sharing Agreement
Gathering Data
Data Validation
Data Transformation
Data Insertion into a Database
Export Data to a csv file
Data Pre-processing
Data Clustering
Get the best model based on the cluster with hyper parameter tuning
Model Saving
Setup the Cloud for deployment
Push the app to the cloud
Start the App
Prediction
Upload the data for prediction
Data Validation
Data Transformation
Data Insertion into a Database
Export Data to a csv file
Data Pre-processing
Data Clustering
Prediction based on the cluster number
Combine all the prediction results
Export the prediction results to a csv file

Data Description

As per the Data Sharing agreement, we will receive data files in a shared location by the client, in the pre-decided formats. The format of files agreed are comma-separated value (.csv) files, with 590 columns each representing a sensor in a wafer, wafer name column, and an output column with values +1 (Working wafer) and -1(Faulty wafer).

As per the data sharing agreement, we will also get two schema files from the client, which contains all the relevant information about the training and prediction datafiles.

Data Validation & Transformation

Once we gather all the data from the client. We will start the data validation process as per the data sharing agreement and the requirements of machine learning process as a part of our training process.

Filename validation:

First, we will start with the file name validation, as per the schema file given for the training datasets. We will create a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we will check for the length of date and time in the file name. If all the values are as per the schema file, we will move such files to Good_Data_Folder. Else, we will move such files to Bad_Data_Folder.

Number of Columns:

We will validate the number of columns present in each file in the Good_Data_Folder, if the number of columns present in a file matches the number of columns present in the schema file for training, that file will be retained in the same folder. Else, that file will be moved to Bad_Data_Folder.

Name of Columns:

The names of the columns present in each file in the Good_Data_Folder is validated and should be as per the schema file. Else, those files will be moved to the Bad_Data_Folder.

Datatype of Columns:

The datatypes of the columns present in each file in the Good_Data_Folder is validated and should be as per the schema file. Else, those files will be moved to the Bad_Data_Folder.

Null values in columns:

If any of the columns in a file have all the values as Null or missing, we discard that file and move it to Bad_Data_Folder. And, we will replace the other null values with a string code “Null”.

Data Insertion into Database

Database Creation:

Create an SQLite database, if it is not already present in the given directory. If it is present, open the database by creating a connection to that database.

Table Creation in the Database:

Create a table in the database with name "Good_Data" for inserting the files in the Good_Data_Folder based on given column names and datatypes in the schema file, if it is not already present in the database. If that table is already present in the database, no need to create a new table, append this data to the existing table.

Insertion of files in the table:

All the files in the Good_Data_Folder are inserted in this table. If any files are raising errors while inserting the data to the table due to the invalid datatypes, those files will be moved to the Bad_Data_Folder.

Export the data to a csv file

The data from the database will be exported to the csv file, and is used for model training in the later stages.

All the files in the Bad_Data_Folder will be moved to Archives, as we want to show the rejected files to the client.

All the files in the Good_Data_Folder will be deleted as we have already captured this data in our database.

Data Pre-processing

We will read the exported csv file, and impute all the values with Null String code using KNN Imputer. We will also perform necessary feature engineering techniques as part of our pre-processing step, like dropping all the columns with zero standard deviation etc.

Data Clustering

We are using a semi-supervised machine learning process. So, once our data is clean, we cluster the data into different clusters (using KMeans Clustering Algorithm) which we later use for training different models on each cluster, this will increase the overall accuracy of the project. So first, we divide the data based on implicit patterns in the data. Then, we give a number to each cluster, and a add new column that consists of cluster names.

Model Training

By using four machine learning algorithms, “Support-Vector Classifier”, “Decision-Tree Classifier”, “Random-Forest Classifier”, and “XGBoost Classifier”, we will train each cluster by Grid Searching few hyperparameters that are already defined using Cross-validation. We decide the best model for each cluster based on their auc scores, if auc score cannot be produced due to the testing dataset, we will use accuracy score. Once we find the best models for each cluster, we will save them in the “/models” directory with their clusters numbers in their names and folder names.

Deployment

After training the model in the local system, and testing it. We will create a CICD pipeline using *circleci* and *dockerhub* for our model deployment.

We will deploy our trained program in Heroku platform.

As the training process takes a lot of time, *we disabled the option of training the model using the website*.

The model training can be performed using API, by providing the location of the training batch files {"folderPath":"Training_Batch_files"}.

Prediction

Before predicting the output of the data present in the data files, we have to perform some similar actions we did in the training process. These steps are required to insert our data for prediction. We will also have a schema file for prediction, with a little difference when compared to the schema file for training i.e., no output column present in the prediction schema file or in the batch files given for the prediction. We will perform the following steps:

- File name validation
- File type validation
- Number of columns validation
- Name of columns validation
- Validation of the column datatypes
- Null values validation and transformation

The validated data will be inserted into a database, and after completing the insertion process for all the files. The files present in the Good_Data_Folder will be deleted and Bad_Data_Folder will be moved to archive.

The data from the database will be exported into a csv file and similar pre-processing steps will be form on the data loaded from this csv file.

After the pre-processing steps, the data will be divided into the clusters by using the already trained Kmeans clustering model.

And, the best model created for each cluster will be used for prediction and all the data will be compiled together with their respective wafer names.

Finally, the prediction results will be exported as a csv file.

Logging Framework

The logs of our model are maintained to monitor its performance and fix any bugs while it is working. Using the logs, we can find the issues easily and fix it quickly.

In this project, we are using synchronous logging method.

Synchronous logging method: Logging is done after each step, until that logging step is completed, the next steps will not be initiated.

Model Retraining

Once we train our model, it is the not final, we have to updated our model based on the new patterns present in the new data. So, whenever we get new data, we will append it to our database, that way we will not lose the previous learnings and also use the current patterns.

Application Flow

main.py

Training Route

1. *Validation*

- Read the data
- Validate the data
- Transform the data
- Insert the data in to a database
- Export the data from the database into a single csv file

2. *Model Training*

- Read the data from the csv file created in the previous step
- Data Pre-processing
- Data Clustering, and adding the cluster number in the dataframe
- Finding the best model for each cluster after hyperparameter tuning.
- Deploy

Prediction Route

1. *Validation*

- Read the data
- Validate the data
- Transform the data
- Insert the data in to a database
- Export the data from the database into a single csv file

2. *Prediction*

- Loading the previous models into the memory
- Predicting the outcome of the data using these models
- Exporting the prediction data into a .csv files and sharing it with the client.