**Wafer Fault Detection**

1. **About the Project:**

The name of the project is Wafer Fault Detection.

**Problem Statement**: The problem statement is to build a model which classifieds whether a wafer is working or not. Wafer actually is slice of semiconductor material which is used in various electronic circuits electronic devices it is also used with photovoltaic cells so just imagine that the scenario is like our client has a production line and on that production line various wafers are deployed and whenever there is a fault in any wafer they have to manually check all the wafers and see which wafer is faulty stop the entire production line and then replace that faulty wafer so this is the problem that our client is facing and now for a solution of this problem our client has asked us to create a model which successfully identifies at which for wafer is faulty and start their operation so that the entire production line it does not get injured so client at their site based on our predictions whatever we are going to make with the wafer is faulty or not based on that client is going to set an alarm at their end that and it will be triggered whenever we say that a certain wafer is faulty so it will get triggered at that point of time and only that particular section on the production line will be stored there that wafer is deployed it that wafer will be replaced and again the production line will resume without any hindrance so this is what our problem statement is.

The goal is to build a Machine Learning model which predicts whether a wafer needs to be replaced or not (i.e., whether it is working or not) based on the inputs from various sensors.

There are two classes:

+1 represents Good Wafer

-1 represents Bad Wafer

1. **Size of the Data:**

There is a total of 41 .csv files for Training and 19 .csv files for Prediction

The client will send data in multiple sets of files in batches at a given location. Data will contain Wafer names and 590 columns of different sensor values for each wafer. The last column will have the "Good/Bad" value for each wafer.

The "Good/Bad" column will have two unique values +1 and -1.

Apart from training files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:

Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns, and their datatype.

1. **Data Type:**

The data used for training consisted of 591 columns of float data type.

And the data used in Prediction consisted of 590 columns of float data type.

1. **Architecture of the Project:**



1. **Techniques used for Data Pre-Processing:**

* Null values are checked. If Null values are present, then they are imputed using KNN Imputer.
* Columns having zero standard deviation were removed, as they don’t give any information during model training.
* Unnecessary columns were removed during training like Wafer's name.

1. **Validation of Files:**

Name Validation- We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to "Good\_Data\_Folder" else we move such files to "Bad\_Data\_Folder."

Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad\_Data\_Folder."

Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Bad\_Data\_Folder".

The datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "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 such a file and move it to "Bad\_Data\_Folder".

Files from the "Bad\_Data\_Folder" are moved to “Archived Bad\_Data\_Folder” before deleting the "Bad\_Data\_Folder".

The “Archived\_Bad\_Data\_Folder” is then sent to the client for rechecking.

"Good\_Data\_Folder" is also deleted to save memory.

1. **Data Insertion in Database:**

Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

Table creation in the database - table with the name - "Good\_Data", is created in the database for inserting the files in the "Good\_Data\_Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created and new files are inserted in the already present table as we want training to be done on new as well as old training files.

Insertion of files in the table - All the files in the "Good\_Data\_Folder" are inserted in the above-created table. If any file has an invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad\_Data\_Folder".

1. **Model Training**

Data Export from Db - The data in a stored database is exported as a CSV file to be used for model training.

Data Pre-processing

a) Check for null values in the columns. If present, impute the null values using the KNN imputer.

b) Check if any column has zero standard deviation, remove such columns as they don't give any information during model training.

Clustering - KMeans algorithm is used to create clusters in the pre-processed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using the "KneeLocator" function. The idea behind clustering is to implement different algorithms

To train data in different clusters. The KMeans model is trained over pre-processed data and the model is saved for further use in prediction.

Model Selection - After clusters are created, we find the best model for each cluster. We are using two algorithms, "Random Forest" and "XGBoost". For each cluster, both the algorithms are passed with the best parameters derived from Grid Search. We calculate the AUC scores for both models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

1. **Deployment:**

This model is successfully deployed on AWS Elastic Beanstalk.

Advantages of using AWS Elastic Beanstalk:

* AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, Python, .NET, PHP, Node.js, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.
* You can simply upload your code and EB automatically handles the deployment, from capacity provisioning, load balancing, and auto-scaling to application health monitoring.
* There is no additional charge for EB – you pay only for the AWS resources needed to store and run your application.
* Increased productivity - Since all the provisions and operations regarding the infrastructure and managing the application are handled by Elastic Beanstalk, you as the developer don’t have to waste your time or learn the skills.

### **Auto Scaling -** Based on what the specific needs of your application are, Elastic Beanstalk will automatically scale your application up and down. For this, it uses its Auto scaling settings that you can adjust to suit your needs.

### **Resource Control -** For all the AWS resources like Amazon EC2 that are powering your application, you will have full control. This will help you take manual control of your application’s infrastructure if and whenever you desire to do so.