***ML Wafer Fault Detection***

1. **Problem Statement**

* To build classification methodology to predict the quality of wafer sensors based on given training data
* Wafer is thin slice of semiconductor used in many electronic devices/circuits used with photovoltaic cells
* Client has a production line and various wafers are deployed on the production lines
* If fault in wafer, then they have to check manually which wafer is faulty, stop the production line and replace the faulty wafer with the new one
* Solution: Create model to identify which wafer is faulty and only that section of the production line is stopped. Client can set an alarm whenever a wafer is faulty.

1. **Description of data**

* Dataset contains wafer name (id) and 590 sensor features for each wafer id
* Target column Good/Bad - +1/-1
* Alarm sends wafer id and client can replace the wafer id particularly.

1. **Application architecture and module division**

* Bigger problem: Whether the wafer is faulty or not
* Better to break down the development into small parts so that changes to be made in end of module doesn’t affect the other modules. Multiple members part of the project so better to divide the modules
* Broker into small subgroups:

1. How to read the data
2. How to validate the data
3. How to do data preprocessing and how to train a model on the data
4. How to do hyperparameter tuning for the model

* **Step 1**: Data ingestion

1. Data for training - client provides or stores the data needed at a particular location, aggregate multiple data sources
2. Data validation – discuss with client the datatype of variables, number of variables, whether any columns contain only null values
3. Data transformation – missing values conversion to null, categorical values in commas or “” and maybe not accepted in DB
4. Data insertion in DB – after transformation insert the data inside the database for further development

* **Step 2**: Training Pipeline / Step

1. Export the data in CSV from DB and csv acts as train data
2. Data preprocessing – perform EDA, check if there are any null values present, convert categorical values into numerical values, if data is imbalanced or normalized
3. Data clustering - to increase the accuracy of the model we divide the data into individual clusters and build model for each cluster separately
4. Hyperparameter tuning - to increase the performance of the individual model selected for each cluster
5. Model saving – save the model for each cluster individually

* **Step 3:** Deploy on cloud

1. Create metadata for pushing the app onto the cloud server
2. Start and test the application

* **Step 4**: Prediction Pipeline

1. Data validation – discuss with client the datatype of variables, number of variables, whether any columns contain only null values
2. Data transformation – missing values conversion to null, categorical values in commas or “” and maybe not accepted in DB
3. Data insertion in DB – after transformation insert the data inside the database for further development
4. Export the data in CSV from DB and csv acts as train data
5. Data preprocessing – perform EDA, check if there are any null values present, convert categorical values into numerical values, if data is imbalanced or normalized
6. Data clustering - to increase the accuracy of the model we divide the data into individual clusters and build model for each cluster separately
7. Call the model for specific cluster number stored
8. Make prediction and export the prediction in a csv file

* **Step 5:** Model retraining

1. When new patterns detected these changes must be aggregated to the model.
2. Provide the prediction + train data to the model for retraining
3. Logging and monitoring framework
4. **Code**:

Main .py -🡪 1. Validation step – read data, validation, transformation, insert into DB, export

to csv file.

* 2. Training – read train data, data preprocessing, data clustering, model finding,

Model tuning, deployment

* 3. Prediction – validation,

Prediction – model saved loaded into memory and make predictions

* User provides different training batch files
* Synchronize logging , asynchronous – code doesn’t wait for completing the logging faster and individual
* Validation – file name is correct or not based on agreement, if we reject the data push into bad else put in good data folder. We use the schema files created.

We delete the good and bad directory as the good data is stored inside the database

* Use the good data from db and split the input and output data and remove the columns which are not necessary for the further process. Check if the null values are present inside the csv file and create a separate csv for storing the null information.
* Used KNN imputer for missing values – we use 3 nearest neighbor of particular value

Average of the 3 nearest values, for the missing values. Weights = ‘uniform’, here we give equal weightage to all 3 nearest values. If ‘weighted’, more weight is given to the nearest point

* Remove the data with std = 0 and remove the columns from further analysis. All the data cleaned now.
* Clustering for improving the accuracy of the model. K-means: we use the elbow method or Kneed library. Save the KMeans model for further reference
* Model training after clustering the data: apply 4-5 algorithms to all the clusters and check which model is best for each cluster available. Highest AUC score chose the model. Perform hyperparameter tuning for each model. RF (bagging) and XGBoost (boosting) algorithm. Save the models for respective clusters
* Prediction –

1. perform validation – for filename, no of columns, all of null values
2. perform transformation – replace nan, insert into db, export csv as input
3. data preprocessing – missing values and imputation, std == 0 dropped, drop unnecessary columns
4. Perform clustering to determine which cluster it belongs to using Kmeans.predict
5. Then based on cluster assigned use the respective model for each individual cluster

* Index.html – default for every browser it call webage it returns httpget

Render template to display respective html pages.

* Deployment for cloud 🡪

1. Requirements.txt – import of packages are included here, as cloud needs instruction for cloud deployment (pip freeze requirements.txt)
2. Inform the python version inside runtime.txt
3. Manifest.yml – name of application, RAM, HDD, random route, buildpack
4. Procfile – how to run the application. No of processes that run parallely and threads to be used.
5. Pivotal web services; cloud foundry cli (run cf command)

* Go inside the project folder
* Cf login
* Cf push