***ML Credit Card Defaulters***

1. **Problem Statement**

* To build classification model to predict whether the customer will default in is credit card bill payment or not.
* If the customers default in their bill payment it can lead to loss of revenue for the bank and it impacts their business strategies. The bank makes money majorly on the interest rate money it earns through the bill payment of credit cards.

1. **Description of data**

* Dataset contains continuous values and categorical values both.
* Columns inside the dataset are:

1. Limit\_Bal (continuous): Credit limit of the person
2. Sex (categorical): 1= male, 2= female
3. Education (categorical): 1= graduate school, 2= university, 3= high school, 4= others
4. Marriage (categorical): 1= married, 2= single, 3= others
5. Age (continuous): age of the person
6. Pay\_0 to Pay\_6: history of the past payment of the bills. Tracked for the past monthly payment records
7. Bill\_amt1 to Bill\_amt6: amount of bill payments
8. Pay\_amt1 to Pay\_amt6: amount of previous payments
9. Default payment next month (Target Label): 1= Yes, 0= No
10. **Application architecture and module division**

* Bigger problem: How to classify the data to decide whether the person defaults 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

* Data for training: User provides different training batch files
* Synchronize logging, asynchronous – code doesn’t wait for completing the logging faster and individual
* Data Validation: Whether the data sent by the client is valid or not as per requirement given. File name is correct or not based on agreement, if we reject the data push into bad folder 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. Check the length of the timestamp given in the file name format, check the number of columns present in the data, check if any column contains just all column values as null values.
* Data Transformation: DB doesn’t accept Nan value so we transform to NULL using fillna.
* Data Insertion in DB: Put the good raw data inside the DB using the datatypes given inside the schema. Export the good raw data table into csv.
* Data preprocessing: Use the exported csv file as input. Perform EDA to understand the data and what processing is needed for the data.

Correlation exists use the domain knowledge whether to drop the column or not. For example, we remove column when they convey the same information, but here dropping the columns shall result in loss of bill and payment history data (if person cannot pay the bill in the first install, possibility of the bill being increased and customer not able to pay again is much higher). Perform PCA to removes the correlation but keeps the variance or just use the data as it is.

* Model Training: Fetch the data and perform the preprocessing decided after doing the EDA. Separate the target and features. Perform clustering on the cleaned data and create clusters. We can apply cluster specific algorithms for better performance. Loop individual clusters and apply different models. Perform standard scaler on the input data. Logistic regression, DT, RF, SVM, NB, KNN, XGBoost algorithms can be used for classification.
* We perform hyperparameter tuning for the selected algorithms and check the performance of the model. Then save the model for further prediction. We used Naïve bayes and XGBoost. If same label classes present then we use the accuracy score else we can use the roc\_auc\_score for the check. Save the model for each cluster created.
* 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. Create the PROCFIlE (web: gunicorn main:app)

Web – indicates it’s a web application

Gunicorn- WSGI web server gateway interface (web server -> application) for to send messages from server to application.

Main – entry point file and app is the name of flask application.

1. Log into Heroku and create app.

Cmd: download Heroku cli

1. Heroku login
2. Git init
3. Heroku git:remote -a test-ml-deploy
4. git add .
5. git commit
6. git push Heroku master