***ML Thyroid Detection***

1. **Problem Statement**

* To build classification model to predict the type of thyroid the person is suffering based on the given training data.
* Hospital can decide the type of thyroid the person suffers like hyper thyroid or not, treatment can vary based on the severity of the illness. Can assign the special doctor or junior doctors based on the level of illness. If illness severe then the seniors can be notified about that.

1. **Description of data**

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

1. Age: age of the patient
2. Sex: gender of the patient
3. On\_thyroxine
4. Query\_on\_thyroxine
5. On\_antithyroid\_medication
6. Sick: whether the patient suffers any sickness
7. Pregnant: whether patient is pregnant or not
8. Thyroid\_surgery
9. I131\_treatment
10. Query\_hypothyroid
11. Query\_hyperthyroid
12. Lithium
13. Goitre
14. Tumor
15. Hypopituitary
16. Psych
17. TSH\_measured
18. TSH
19. T3\_measured
20. T3
21. TT4\_measured
22. TT4
23. T4U\_measured
24. T4U
25. FTI\_measured
26. FTI
27. TBG\_measured
28. TBG
29. Referral\_source: organization referring the patient
30. Class (Target Label): negative or positive or componensated, primary, secondary\_hypothyroid
31. **Application architecture and module division**

* Bigger problem: How to classify the data to decide whether the individual suffers from thyroid?
* 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. Add quotes to all values present in the categorical values for insertion into the DB.
* 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.

Strip or remove the white spaces in categorical values.

Replace the “?” with NaN in the dataset and check number of missing values. Check if the values like ‘?’ present through whole data column, then drop it

The T4U\_measured and other measured columns are not necessary as they additional values we remove. Replace the ‘?’ by np.nan for numerical values

Custom mapping for sex, referral\_source and other true and false columns

Label encoder for the target class

Impute the missing values using the KNNImputer with neighbors=3

Check the distribution of the numerical values and if skewness present use the log transformation (+1 for log zero)

Check the distribution of class and whether its balanced or not

If imbalanced then use the random over sampling technique for handling the imbalance

* 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 KNN and RF. 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. Reassign the encode class category back on the predicted file.

* 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. Follow Heroku deployment steps