DataWhisperer – Interact with Your Dataset

- **Introduction**

  Automating the general processes behind creating ML pipelines is essential for improving the efficiency in model development. This project addresses this very issue. It achieves this by creating a system that can autonomously generate ML pipeline for a given dataset.

  This automation is achieved by breaking down the problem itself into six smaller problems that require automation themselves:

  1. **Data Preprocessing**: The idea of data cleaning, selecting features, encoding categorical data, scaling numerical data.
  2. **Data Visualization**: Presenting a basic understanding about the dataset by visualizing the contents of the dataset using graphs and charts.
  3. **Model Generation**: Finding the best model for the dataset and the problems, by utilizing cross-validation search to find out the best ML pipelines.
  4. **Finding Patterns**: Training various ML models to find patterns frequently occurring patterns.
  5. **Insight Generation**: To utilize the finding of patterns, and realize the consequences of finding them.
  6. **User Interface**: To provide a welcoming user interface, for interaction with a dataset.

  The idea behind automating creation of ML pipelines is to provide abstraction from the rigorous world of selecting ML models, and to provide a starting point in generating customized ML models pertaining to the dataset.

- **Dataset Description:**

The idea is to create a website that can handle any dataset, for any classic ML task, like regression, ranking or clustering.

But for now, the website is fully-functional in handling datasets which have the following constraints:

1. **Target Feature**: The dataset can only have one target feature, and it must be the last column of the dataset.
2. **Data**: A website is purely designed to work with raw data. So a minimum of one categorical and numerical feature is required in the dataset.
3. **Headers**: *This is not much of a problem.* It is highly encouraged that the first row of the data is the names of the features.

The website can only handle .csv or .xlsx files with a size limit of 200MB.

But, for now the website can only handle classification, either binary or multi-class classification. So dataset pertaining to these problems are only supported.

- **Methodology:**

As explained above; to solve this problem, we need to automate six sub-problems.

- **Data Preprocessing**: To automate data preprocessing, every dataset is passed to a class called DataPreProcessor, which performs the following tasks in the following order:
  - Separating features from targets.
  - Separating numerical and categorical features.
  - Finds the total amount of missing values in the dataset.
  - Handles missing values by imputation in the following way:

        - Numerical features are imputed using K-Nearest Neighbors Imputation.
        - Categorical features are imputed using most frequent (Mode) imputation.

    Missing values can also be handled using removal, but have been set to imputation by default.

    - Handles duplicates using removal. Only exact duplicates are removed.
    - Outliers are found using Inter-Quantile Range (IQR) and removed.
    - Categorical features are encoded using OneHotEncoder, only if they have not been encoded before.
    - Categorical target is encoded using LabelEncoder, only if it is not encoded before.
    - Dataset is split in the proportion of 3:1:1, or 60%, 20% and 20% for train, test and validation.
    - Training data is fit into a StandardScaler, testing and validation data are transformed using the scaler.
    - Finally, a method performs all the following tasks in order, and return the preprocessed train, test and validation splits.

    Tools Used: Dask, Dask-ML, Scikit-Learn, Streamlit.

- **Data Visualization:** To automate visualization of the dataset, a class called Knowledge services all the graphs, charts and dataframes required to display basic knowledge from the dataset. All the data from this class is displayed using streamlit components. A brief rundown of the class is provided here:
    - The constructor requires the dataframe, names of the numerical and categorical columns.
    - Basic data, such as dataset descriptions, The amount of space the dataframe occupies, and also complicated data like the correlation matrix, and the feature histogram are serviced by the method. The final return value is a

dictionary, containing all the dataframes and images, which are held as BytesIO objects.

- Column wise data takes two inputs, a column index and a Boolean representing if the column is categorical. It is responsible for providing fundamental data about the column like unique values, total number of missing values, etc, and finally provides a scatter plot of the data, visualized against the original normal curve for the dataset.

Tools Used: Streamlit, PyPlot from MatPlotLib, Pandas, Numpy and Dask.

- **Model Generation:** The process of model generation is completely automated using a tool called 'EvalML'. It can perform Cross-Validation checks for every possible binary, or multiclass classification model, on the validation data. It finally finds the best model.

  This best model is then trained on training data, and tested on test data. A view of the classification report after training, after testing and the final confusion matrix is then displayed.

  Tools Used: Streamlit, Dask, EvalML, Seaborn, Scikit-Learn

- **Finding Patterns:** To find patterns, a number of classes are used. A detailed description of each of these classes is given below:
  - *LinearSeperability:* This class is responsible for checking if the dataset is linearly separable, that is, if it can be classified using linear models. This is achieved by comparing the training accuracy of a linear SVC (Support Vector Classifier) with a non-linear SVC (rbf kernel). It finally returns a dictionary containing all the training and

test accuracies, and a final verdict. This dictionary is serviced on the webpage using streamlit.

  - *FeatureImportance:* This class is responsible for finding the ranking of the most important features in the dataset. This is achieved by using Random Forest Classifier's 'feature_importance' attribute. This attribute is generated after ensembling all the decision trees. This feature importance is return from a method to visualize_importance(), which returns a BytesIO object that of a simple bar graph of features.
  - *Apriori:* This class is responsible for finding mining patterns in the dataset. It utilizes the apiori implementation from mlextend. The class provides two methods, one to return a dataframe of frequently occurring itemsets, and another to mine associate rules from the dataset.
  - *DBScan:* This class is responsible for finding partitional clusters withing the dataset. The class utilizes scikit-learn's DBSCAN implementation. It finds clusters within the dataset, by firstly running Parameter Grid Search, and finding the best hyperparameters for the dataset. Then, it performs clustering on the dataset, using these hyperparameters. The class finally returns a simple scatter plot of clusters with a silhouette score.
  - *Hierarchies:* This class is responsible for finding hierarchical clusters within the dataset. It utilizes scipy's linkage, and dendrogram implementation for this. It finally returns the dendrogram as a BytesIO object.

- **User Interface:** The user interface is completely implemented using 'Streamlit', a package built especially for representing AI/ML models. Navigation of the pages in the below order is best advised for the best experience on the webpage. The interface contains multiple pages, each page has the following functionality:

    - *DataWhisperer.py*: This page is the landing page for the website. The webpage includes a basic description of the project, and a small warning.
    - *Upload Your Dataset.py:* This page allows the user to upload their dataset, and pass the column-wise 0-indexed index of the target feature. After passing the required data, a small preview of the dataset is shown. It also displays a set of prerequisites required before uploading your dataset.
    - *Preprocess Your Data.py:* This page is the initializer of the DataPreProcessor.py class mentioned above. It displays the basic preprocessing techniques that have been applied on the dataset. It finally generates a train, test and validation split of the dataset, which you can download as .csv
    - *Data Visualiztion.py:* This page is the initializer of the Knowledge class mentioned above. It displays the basic data, as well as a sidebar to select columns, to display column-wise data.
    - *Model Generation.py:* This page is the initializer of the 'EvalML' module mentioned earlier. It explains how the train, test and validation split is used, and checks for the best model, then displays training and test accuracies as well as the confusion matrix.
    - *Fundamental Patterns And Insights:* This page simply services all the data acquired via the classes in the 'models' directory.
- **Results And Discussion:** The results from this project cannot be displayed as the input of the dataframe is left to the user. A trail run using a sample dataset can be seen over here: ***https://youtu.be/lo0_Wnk3mqM***

- **Conclusion:**

“The DataWhisperer Project makes a significant amount of progress in the field of insight and ML pipeline automation. It has managed to do so by automating six dependent steps, that are linked to one another. “

- **Future Work:**
  The amount of work possible for this project in the future is enormous. These are the following prospects of the project:

  - ✐ For now, the project can only handle any binary or multiclass classification task. It must be brought up to speed in regression, clustering, ranking, dimensionality reduction, and any other non-neural network-based ML task.
  - ✐ To achieve the above objective, it is very much necessary to bring another part automatedly, that being EDA (Exploratory Data Analysis), as different tasks need to be approached differently.
  - ✐ Small bugs are present in the code here and there. A testing mechanism is much required.
  - ✐ The deployment of the website is very limited. It has been provided 0.6 vCPU and 512 MB Memory. This has to be upgraded. Due to this reason, the time to find a model is enormous, approximately 1 to 10 minutes in time.

I would love to hear review of my work, as it has taken me a lot of time and resources to build this website.