### Author: Souvik Nath

### Dated: 21.12.2019

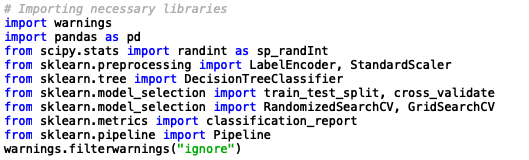
**Objective**: The objective of this document is to walk the user through the ‘Decision Tree algorithm python code file’ on the different functionalities and techniques being used to attain the end outcome of implementing the algorithm

**SECTION 1: Importing python libraries**

This is the first section of the code, where all the necessary libraries are being imported. Python programmers might find this resemblance with almost all python programs being written.

*NOTE: Line number 10 is a command used to hide all irrelevant python warnings*

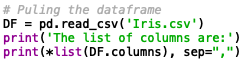
**CODE:**

****

**SECTION 2: Importing flat files as Pandas Objects**

This section of the code is where we pull the .csv file from the provided web url.

**CODE:**

****

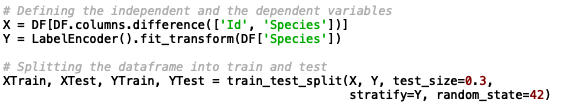
**Explanation:**

1. **pd.read\_csv / pandas.read\_csv:** This command provides the programmer with the functionality to pull flat files from both local directory or web urls

**SECTION 3: Defining the dependent and Independent Variables and Splitting into Train and Test datasets**

This section of the code is where we define the **Dependent** and **Independent** variables required for model training. For any supervised learning algorithm like **Decision Trees** one needs to define the **Dependent** and **Independent** variables.

**CODE:**

****

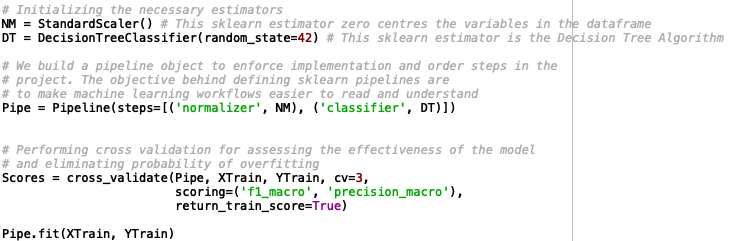
**Explanation:**

1. Declaration of Independent Variables(*X*):   
     
   The first line of code is where we select the Independent Variables from the **Analytical Dataset.** The df.columns.difference() command is a **Pandas** functionality which allows the programmer to select all columns but the ones mentioned within the command.
2. Declaration of Dependent Variables(*Y*):   
     
   The second line of code is where we select and transform the **dependent variable**. The **dependent variable** in our case is of string data-type.  
     
   The **Decision Tree** algorithm in the python library being used here(**scikit-learn**) has the **CART(***Classification and Regression Trees***)** implementation, which inherently cannot handle string data-types in the **dependent variable**.  
     
   **Label encoding**(*sklearn.preprocessing.LabelEncoder*): Label Encoder assigns numeric values to each unique value in the **dependent variable** field**.** The fit\_transform()methodwithin the LabelEncoder()estimator performs the transformation of assigning numeric values to each unique value in the **dependent variable** column.
3. Splitting into **Train** and **Test** datasets:  
     
   The sklearn.model\_selection.train\_test\_split() enables splitting the X and Y into training and testing datasets.   
     
   The test\_size keyword argument allows the programmer to select the size of the training and test set from the original frame.  
     
   The stratify keyword argument enables splitting in a stratified fashion based on the dependent column.

**SECTION 4: Training the model**

This section of the code is where we train the model.

**CODE:**



**Explanation:**

1. Initializing the estimators:   
     
   The first two lines of code is where we have initialized the estimators StandardScaler() and DecisionTreeClassifier().  
     
   StandardScaler() normalizes the features in X to have a mean of ‘0’ and a standard deviation of ‘1’. Though such monotonic transformations don’t affect the learning and predicting capability of such tree-based algorithms, it is usually suggested as a best practice to do so in case of visualizations.  
     
   The DecisionTreeClassifier() is the **scikit-learn** object for the **decision tree algorithm** which comes with built-in methods like, fit(), predict(), etc. The **‘..Classifier’** in the name signifies that this **scikit-learn** object for the **decision tree algorithm** is only suited for **classification** problems.
2. Creating the pipeline:   
     
   We build a **pipeline** object to enforce implementation and order steps in the

project. The objective behind defining **scikit-learn** pipelines are to make **machine learning workflows** easier to read and understand.

1. k-Fold Cross Validation:  
     
   **Cross-validation** is a resampling procedure used to evaluate **machine learning** models on a limited data sample.We perform **cross validation** to assess the effectiveness of the model and eliminating all possibilities of model overfitting on the training dataset  
     
   We perform a 3-fold cross validation for this exercise. There is essentially no rule of thumb in deciding the ‘k’ to perform cross validation. The choice of ‘k’ is however arbitrary and it varies from analyst to analyst and problem-complexity to problem complexity.  
     
   Since this is a **multi-classification** problem, we choose the scoring parameters as to be **‘f1-macro’** and **‘precision-macro’** which are basically macro averages of this more popular model evaluation metric in **classification-type** problems.

**SECTION 4: Hyper-parameter tuning**

This section of the code is where we tune the **hyper-parameters** of the model.

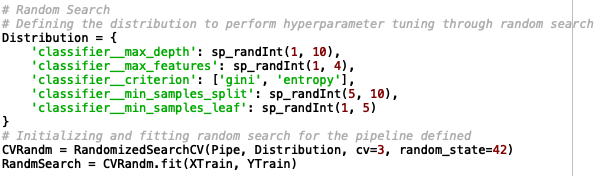
**Hyper-parameter** **tuning** is an essential step in machine-learning, and is defined as the process of choosing an optimal set of parameters for any learning algorithm.. It can be related to as a tightrope walk in achieving balance between overfitting and underfitting of the model.

There are a plethora of techniques for **hyper-parameter tuning**. But for purposes of maintaining simplicity we will restrict our scope to approaches of searching for the best configuration using the more popular **Random Search** and **Grid Search** techniques.

**RANDOM SEARCH:**

One of the very popular hyper-parameter techniques amongst data science enthusiasts to data scientists have observed great practical real life implementations across the industry for a variety of problem complexities. To define Random Search, it is a family of numerical optimization methods, that don’t require the gradient of the problem to be optimized.

**CODE:**

****

**Explanation:**

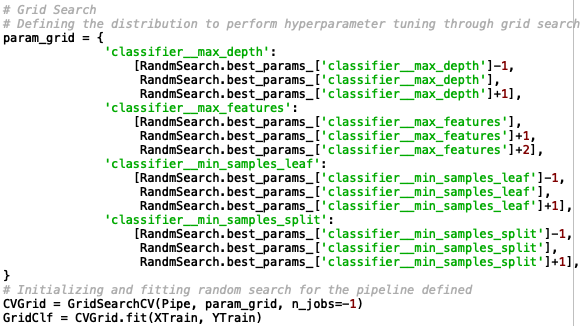
1. Defining the parameter grid(*Distribution*):  
     
   Approaches of searching for finding the best configuration requires us to define the parameter grid within a range of values for each parameter that needs to be tuned or we want to tune. The **Random Search** algorithm searches for the best configuration within this defined space of values. In our case, I decided to go ahead and tune the following parameters for the **Decision Tree** algorithm:  
   1. Maximum depth of the tree i.e. classifier\_\_max\_depth
   2. Maximum Features in the leaf node i.e. classifier\_\_max\_features
   3. Tree split criterion for calculating classification error i.e. classifier\_\_criterion
   4. Minimum number of samples for splitting i.e. classifier\_\_min\_samples\_split
   5. Minimum number of samples per leaf i.e. classifier\_\_min\_samples\_leaf
2. We then perform the **Random Search** over a 3-fold **Cross Validation** of the training set

**GRID SEARCH:**

**Grid Search** comes as the next step to **Random Search** in typical machine learning workflows.

**Grid Search** or **parameter sweep** is simply an exhaustive searching technique of finding the best configuration within a manually defined subset of the **hyperparameter space** of a learning algorithm. A **Grid Search** algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

**CODE:**



**Explanation:**

1. Defining the parameter grid(*param\_grid*):  
     
   As per definition, since the **hyperparameter space** has to be manually defined for **Grid Search**. I decided to define the space relying on the results of the **Random Search** algorithm. For **Grid Search**, I decided to go ahead and define the **hyperparameter space** for the following parameters for the **Decision Tree** algorithm:  
   1. Maximum depth of the tree i.e. classifier\_\_max\_depth
   2. Maximum Features in the leaf node i.e. classifier\_\_max\_features
   3. Minimum number of samples for splitting i.e. classifier\_\_min\_samples\_split
   4. Minimum number of samples per leaf i.e. classifier\_\_min\_samples\_leaf
2. We then perform the **Grid Search** over a 3-fold **Cross Validation** of the training set