Code Documentation

Assignment-4

-By Gursimar Singh Bedi

Task in hand

To create a machine learning model using decision tree classifier using the iris dataset.

Procedure

# Installing Libraries

**In [1]:pip install** **pandas**

**In [2]:pip install** **seaborn**

**In [3]:pip install** **matplotlib**

**In [4]:pip install** **sklearn**

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. We will be the data-frame data structure to read the dataset in hand and manipulate it.

Seaborn is a Python data visualization library based on [matplotlib](https://matplotlib.org/). It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn will simplify the process of creating a heat map for correlation values.

Scikit-learn is a [free software](https://en.wikipedia.org/wiki/Free_software) [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [library](https://en.wikipedia.org/wiki/Library_(computing)) for the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) [programming language](https://en.wikipedia.org/wiki/Programming_language). It features various [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and [clustering](https://en.wikipedia.org/wiki/Cluster_analysis) algorithms including [support vector machines](https://en.wikipedia.org/wiki/Support_vector_machine), [random forests](https://en.wikipedia.org/wiki/Random_forests), [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting), [*k*-means](https://en.wikipedia.org/wiki/K-means_clustering) and [DBSCAN](https://en.wikipedia.org/wiki/DBSCAN), and is designed to interoperate with the Python numerical and scientific libraries [NumPy](https://en.wikipedia.org/wiki/NumPy" \o "NumPy) and [SciPy](https://en.wikipedia.org/wiki/SciPy). It will help us in efficiently applying the decision tree classfier

# Importing Libraries

**In [1]:import** **pandas as pd**

**In [2]:import** **seaborn as sns**

**In [3]:import** **matplotlib.pyplot as plt**

**In [4]:from** **sklearn.tree import DecisionTreeClassifier**

**In [5]:from** **sklearn.metrics import f1\_score, accuracy\_score, confusion\_matrix**

**In [6]: from** **sklearn.model\_selection import train\_test\_split, GridSearchCV**

pd and sns are optional aliases for both the libraries and help reducing the length of the code. Plt is being used to plot the heat map and view the results.

# Importing the Dataset

**In [7]: df=** **pd.read\_csv("C:/Users/Predator/Desktop/iris.csv")**

Using pandas we will store the dataset in a variable named df.

# Analyzing the Dataset

**In [8]: print(df.head())**

**In [9]: print(df.describe())**

**In [10]: print(df.isnull.sum())**

head() returns the first five rows of the dataset. We will use this to find out the names of the column and get a brief idea about the dataset.

Describe() is used to return a statistical summary of all columns in a dataset. We can use this to gain insights and look for anomalies.

.isnull() checks for null values and sum() returns the column-wise sum, hence in amalgamation the command returns the number of null values in each row. This was used to ensure that the data was relatively clean

# Dropping ID Row

After analyzing the dataset, it was observed that a column named id was being used to uniquely identify each observation. As it will serve no purpose to the prediction we will be dropping the column from the dataset.

**In [11]: df= df.drop(‘Id’,axis=1)**

Axis=1 argument is used to drop columns from a dataset.

# Converting target variables into numerical data for predictions

Our target variable ‘Species’ is in the form of string values which is not fit for prediction. So to convert it into numerical values we are going to apply the following process.

**In [12]: print(df[‘Species’].value\_counts())**

**In [13]: species= {‘Iris-versicolor’:0, ‘Iris-virginica’:1, ‘Iris-sentosa’:2}**

**In [14]: numerical\_df= df**

**In [15]: numerical\_df[‘Species’]=[species[val] for val in numerical\_df[‘Species’]]**

Value\_counts() is used to get the number of unique value in the column alongside their frequency. We received three columns each of frequency 50 suggesting that the dataset is balanced.

We create a dictionary with keys as values from the species column and values as the assigned category. The code in line number 15 creates a loop in which each key is converted to its corresponding value from the dictionary thereby resulting in numerical values.

# Plotting Correlation Heat Map

**In [16]: sns.heatmap(numerical)df.corr(),annot=True)**

**In [17]: plt.show()**

Using the Seaborn library a heat map of the correlation values of all the columns was created. The heatmap displayed high correlation between petalWidth and petalLength. This implies that dropping one of the columns will result in better accuracy of the model.



We dropped the petalLength Column as it shared higher aggregate correlation with respect to the petal length column using the following code

**In [18]: corr\_mat=numerical\_df.corr().abs()**

**In [19]: upper=corr\_mat.where(np.triu(np.ones(corr\_mat.shape,k=1).astype(np.bool)**

**In [20]: to\_drop= [columns for columns in upper.columns if any(upper.columns>0.9)]**

**In [21]: numerical\_df= numerical\_df.drop(to\_drop, axis=1)**

The first line aggregates the values of correlation for each column. The second line selects the upper triangle and the third line is used to select the columns which have correlation values higher than 0.9.

# Train Test Split

First we will store all the independent values in a variable named x and our target column in a variable named y. Then using train\_test\_split from sklearn we are going to split the data in train and test values. Train\_size is used to dictate the percentage of split. The value 0.7 signifies that 70% percent of dataset will be used for training.

Note: The variable storing the values should always be in the order given below (line number 24)

**In [22]: x= numerical\_df.drop(‘Species’, axis=1)**

**In [23]: y= numerical\_df[‘Species’]**

**In [24]: x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, train\_size=0.7, random\_state=123)**

# Decision tree algorithm

**In [25]: tree=DecisionTreeClassifier(random\_state=27)**

**In [26]: tree\_para = {'criterion':['gini','entropy'],'max\_depth':[4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]}**

**In [27]: clf\_GS = GridSearchCV(tree, tree\_para)**

**In [28]: clf\_GS.fit(x\_train , y\_train)**

**In [29]: print(clf\_GS.best\_params\_)**

We import the decision tree algorithm from sklearn. As the algorithm has some hyper parameters which can be manipulated to give us varying results we will be using grid search cv. The first argument for girdsearchcv is the model while the second is a dictionary containing hyper parameters and their varying values which the algorithm can choose from.

The resultant model is then trained on our training data using fit() method and the model with the best accuracy is selected. We can use best\_params\_ to receive the values of the best parameters.

# Predictions

Using the .predict() method we will now make predication based of off our machine learning model on both training and test data.

**In [30]: train\_predict = clf\_GS.predict(x\_train)**

**In [31]: test\_predict = clf\_GS.predict(x\_test)**

# Results

To measure the performance of the model we will be using metrics like accuracy f1\_score and confusion matrix with our predicted values and original values as the arguments. Sklearn makes it easier for us to apply these metrics.

In measurement of a set, accuracy is closeness of the measurements to a specific value, while precision is the closeness of the measurements to each other.

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.

**In [32]: print("Train Accuracy:"+str(accuracy\_score(y\_train, train\_predict)\*100))**

**In [33]: print("Test Accuracy:"+str(accuracy\_score(y\_test, test\_predict)\*100))**

**In [32]: print("F1 Score on Train:"+str(f1\_score(y\_train, train\_predict, average="macro")\*100))**

**In [33]: print("F1 Score on Test:"+str(f1\_score(y\_test, test\_predict, average="macro")\*100))**

**In [32]: print("Confusion Matrix on Train:")**

**In [33]: print(confusion\_matrix(y\_train, train\_predict))**

**In [32]: print("Confusion Matrix on Test:")**

**In [33]: print(confusion\_matrix(y\_test, test\_predict))**

Model Used

Name: Species, dtype: int64

{'criterion': 'gini', 'max\_depth': 4}

Train Accuracy

99.04761904761905

Test Accuracy

93.33333333333333

F1 Score on Train

99.07565685343464

F1 Score on Test

92.42636746143057

Confusion Matrix on Train

[[40 0 0]

[ 1 32 0]

[ 0 0 32]]

Confusion Matrix on Test

[[10 0 0]

[ 3 14 0]

[ 0 0 18]]

Given that the dataset was relatively clean and simple. The results produced are satisfactory.