Scikit-Learn

Decision Tree Classification example with Iris dataset

In this task 3.1P I will demonstrate the commands and their functionalities in helping us classify an Iris datset using decision trees.

First I will start with importing the *pandas* library and *iris* dataset.

In [1]:
```
import pandas as pd
from sklearn.datasets import load_iris
```

We are now using the variable *iris* to store the loaded iris dataset.

In [2]:
```
iris = load_iris()
```

The variable *df* is used alongside with the **pandas.DataFrame()** function to load the data contained in Iris to a proper dataset with the column names. We also add a *target* column for future use.

In [3]:
```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
```

Now that the dataset has been properly formatted we can now prepare the dataset to train and test the decision tree classifier. We do this by importing the *train_test_split* function from the *sklearn.model_selection* library.

Then the imported command is used to organize the data that will be trained and tested accordingly. Here we can also pass additional inputs such as the test size, which divides the data at a percentage passed to it(0.33 in this case) the remainder would then be the train size, and random state, which controls the shuffling applied to the data before splitting it.

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(df[iris.feature_names], df['target'],
                                                    test_size=0.33, random_state=2)
```

The output of the above function is shown below. It is clearly shown that the rows from *df* is divided at $^{1}/_{3}$ for testing and $^{2}/_{3}$ for training the algorithm.

Now we will import the decision tree module from sklearn library.

In [5]:
```
from sklearn import tree
```

Now we use *clf* to hold the **DecisionTreeClassifier()** command. What I meant by hold is the *clf* variable is used in place of *tree.DecisionTreeClassifier()* so we have less work to do down the line.

In [6]:
```
clf = tree.DecisionTreeClassifier()
```

Now we use our shortened *clf* to run the **fit()** function. The *clf.fit(X_train, Y_train)* method is passed the *X_train* and *Y_train* datasets to build a decision tree classifier from the training set (X_train, Y_train).

Here clf.fit(X_train, Y_train) expands to tree.DecisionTreeClassifier.fit(X_train, Y_train).

In [7]:
```
clf.fit(X_train, Y_train)
```

Out[7]:
```
DecisionTreeClassifier()
```

Now that the decision tree classifier has been trained we can test it by using the **predict()** function. This will let us run the function **tree.DecisionTreeClassifier.predict()** which will take the *X_test* dataset we split earlier to make it predict values it think is right for it. We then put this predicted dataset into *Y_pred*.

In [8]:
```
Y_pred = clf.predict(X_test)
```

Now that we have the predicted output (Y_pred) and real output(Y_test), we can test how our decision tree classifier worked by passing to the **accuracy_score()** function from sklearn. This function takes both inputs and gives an output from 0 - 1 which represents how successful out classifier has been. Here we have an accuracy score of 0.96 which is pretty good.

In [9]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_pred, Y_test))
```

```
0.96
```

As the above method to calculate the accuracy score has a bit of an overhead by importing another library module we can also use the **tree.DecisionTreeClassifier.score()** function to give us an accuracy score. Which takes the *X_test* and *Y_test* datasets as inputs to replace the last 3 lines of code we executed; thereby making our lives easier.

In [10]:
```
x = clf.score(X_test, Y_test)
print(x)
```

```
0.96
```

The **tree.DecisionTreeClassifier.get_params()** function gives us an output which contains all the parameters set for the current classifier. Some of which are values we assigned to it and some are default.

In [11]:
```
clf.get_params()
```

Out[11]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

The functions **tree.DecisionTreeClassifier.get_n_leaves()** and **tree.DecisionTreeClassifier.get_depth()** as the name sugests are used to output the values they hold. *get_n_leaves()* means it returns the number of leaves in the decision tree. *get_depth()* means the output will be the depth of the decision tree which will be the maximum distance between the root and any leaf.

In [12]:
```
print(clf.get_n_leaves())
print(clf.get_depth())
```

```
7
4
```

Now that we have successfully made a demo of the decision tree classifier and taken a look at its workings, it is now time to visualize the tree. It is possible to generate the the below diagram without importing *matplotlib.pylot* but I have done so that the diagram generated will be pleasing to look at. The only use of *pyplot* is to set the *dpi* parameter so that the diagram will be big.

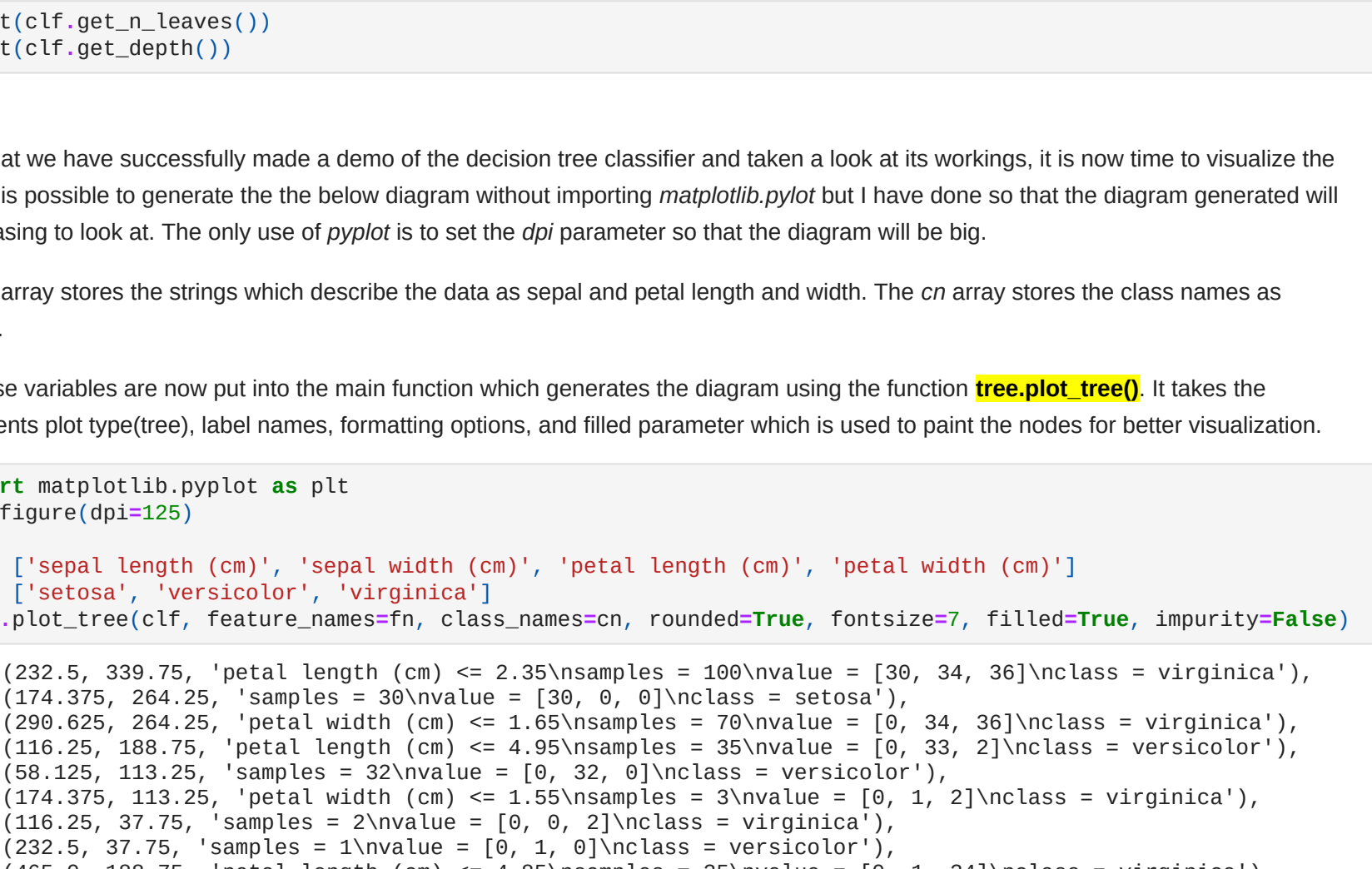
The *fn* array stores the strings which describe the data as sepal and petal length and width. The *cn* array stores the class names as strings.

All these variables are now put into the main function which generates the diagram using the function **tree.plot_tree()**. It takes the arguments plot type(tree), label names, formatting options, and filled parameter which is used to paint the nodes for better visualization.

In [13]:
```
import matplotlib.pyplot as plt
plt.figure(dpi=125)

fn = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
cn = ['setosa', 'versicolor', 'virginica']
tree.plot_tree(clf, feature_names=fn, class_names=cn, rounded=True, fontsize=7, filled=True, impurity=False)
```

Out[13]:
```
[Text(232.5, 339.75, 'petal length (cm) <= 2.35\nsamples = 100\nvalue = [30, 34, 36]\nclass = virginica'),
 Text(174.375, 264.25, 'samples = 30\nvalue = [30, 0, 0]\nclass = setosa'),
 Text(290.625, 264.25, 'petal width (cm) <= 1.65\nsamples = 70\nvalue = [0, 34, 36]\nclass = virginica'),
 Text(116.25, 188.75, 'petal length (cm) <= 4.95\nsamples = 35\nvalue = [0, 33, 2]\nclass = versicolor'),
 Text(58.125, 113.25, 'samples = 32\nvalue = [0, 32, 0]\nclass = versicolor'),
 Text(174.375, 113.25, 'petal width (cm) <= 1.55\nsamples = 3\nvalue = [0, 1, 2]\nclass = virginica'),
 Text(116.25, 37.75, 'samples = 2\nvalue = [0, 0, 2]\nclass = virginica'),
 Text(232.5, 37.75, 'samples = 1\nvalue = [0, 1, 0]\nclass = versicolor'),
 Text(465.0, 188.75, 'petal length (cm) <= 4.85\nsamples = 35\nvalue = [0, 1, 34]\nclass = virginica'),
 Text(406.875, 113.25, 'sepal width (cm) <= 3.1\nsamples = 3\nvalue = [0, 1, 2]\nclass = virginica'),
 Text(348.75, 37.75, 'samples = 2\nvalue = [0, 0, 2]\nclass = virginica'),
 Text(465.0, 37.75, 'samples = 1\nvalue = [0, 1, 0]\nclass = versicolor'),
 Text(523.125, 113.25, 'samples = 32\nvalue = [0, 0, 32]\nclass = virginica')]
```



Made by Andrew Eymard 220415619