JawadAhmed_20P-0165

February 19, 2023

1 Lab Task 4 (Decision Tree)

1.1 1. Load the iris dataset using scikit-learn.

[1]:
```
from sklearn.datasets import load_iris
from sklearn import tree
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn import metrics
```

[2]:
```
data = load_iris()
df = pd.DataFrame(data=data.data, columns=data.feature_names)
Y = data.target
target_names = data.target_names
df.head()
```

[2]:
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
```

[3]:
```
print("Target Names: ", target_names)
```

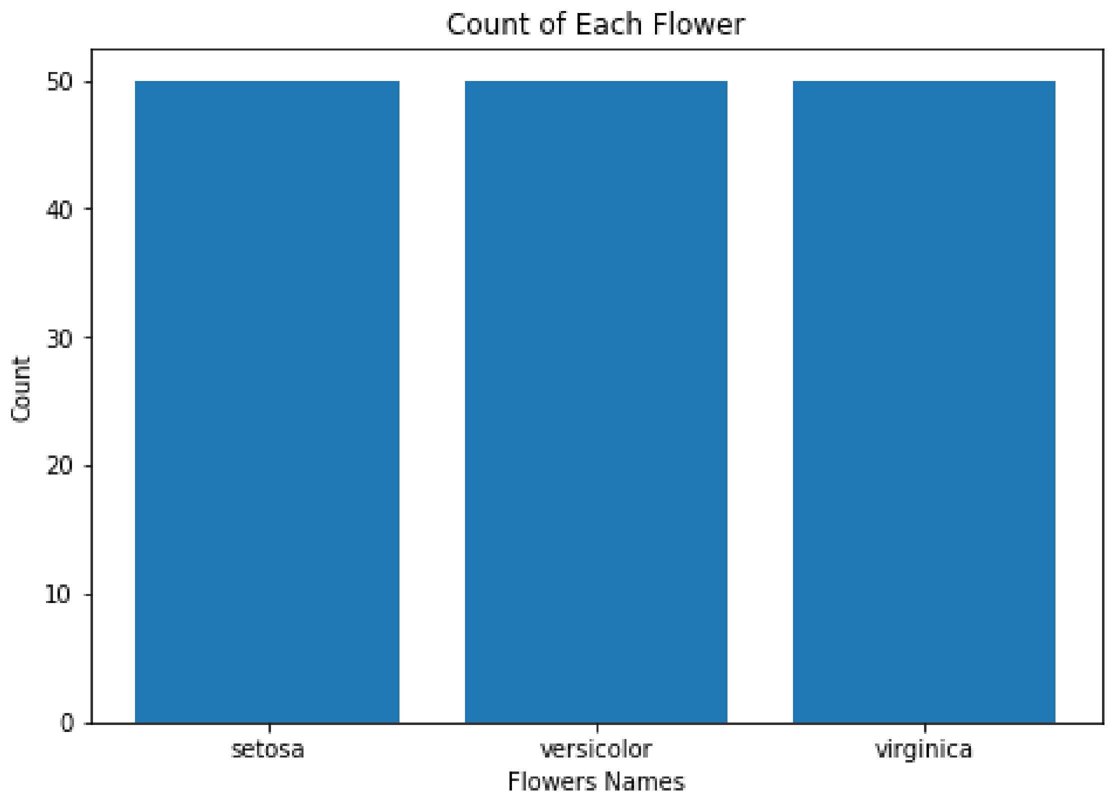
```
Target Names:  ['setosa' 'versicolor' 'virginica']
```

[4]:
```
Y
```

[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

1.2 Visualize the DataSet

[5]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
freq = np.bincount(Y)
ax.bar(target_names,freq)
plt.xlabel('Flowers Names')
plt.ylabel('Count')
plt.title('Count of Each Flower')
plt.show()
```



The graph show that each target has same frequency.

1.3 2. Split the dataset into training and testing sets with 70% of the data for training and 30% for testing.

[6]:
```
x_train,x_test,y_train,y_test=train_test_split(df,Y,test_size=0.3)
```

[7]:
```
x_train
```

```
[7]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
     29                 4.7               3.2                1.6               0.2
     60                 5.0               2.0                3.5               1.0
     119                6.0               2.2                5.0               1.5
     66                 5.6               3.0                4.5               1.5
     134                6.1               2.6                5.6               1.4
     ..                 …                 …                  …                 …
     34                 4.9               3.1                1.5               0.2
     87                 6.3               2.3                4.4               1.3
     110                6.5               3.2                5.1               2.0
     76                 6.8               2.8                4.8               1.4
     3                  4.6               3.1                1.5               0.2

     [105 rows x 4 columns]
```

```
[8]: x_test
```

```
[8]:      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
     83                 6.0               2.7                5.1               1.6
     38                 4.4               3.0                1.3               0.2
     90                 5.5               2.6                4.4               1.2
     8                  4.4               2.9                1.4               0.2
     47                 4.6               3.2                1.4               0.2
     77                 6.7               3.0                5.0               1.7
     70                 5.9               3.2                4.8               1.8
     143                6.8               3.2                5.9               2.3
     2                  4.7               3.2                1.3               0.2
     36                 5.5               3.5                1.3               0.2
     22                 4.6               3.6                1.0               0.2
     27                 5.2               3.5                1.5               0.2
     68                 6.2               2.2                4.5               1.5
     46                 5.1               3.8                1.6               0.2
     15                 5.7               4.4                1.5               0.4
     114                5.8               2.8                5.1               2.4
     111                6.4               2.7                5.3               1.9
     125                7.2               3.2                6.0               1.8
     72                 6.3               2.5                4.9               1.5
     113                5.7               2.5                5.0               2.0
     31                 5.4               3.4                1.5               0.4
     136                6.3               3.4                5.6               2.4
     41                 4.5               2.3                1.3               0.3
     74                 6.4               2.9                4.3               1.3
     105                7.6               3.0                6.6               2.1
     140                6.7               3.1                5.6               2.4
     81                 5.5               2.4                3.7               1.0
     1                  4.9               3.0                1.4               0.2
     75                 6.6               3.0                4.4               1.4
```

```
149         5.9         3.0         5.1         1.8
58          6.6         2.9         4.6         1.3
102         7.1         3.0         5.9         2.1
59          5.2         2.7         3.9         1.4
5           5.4         3.9         1.7         0.4
116         6.5         3.0         5.5         1.8
122         7.7         2.8         6.7         2.0
44          5.1         3.8         1.9         0.4
131         7.9         3.8         6.4         2.0
51          6.4         3.2         4.5         1.5
145         6.7         3.0         5.2         2.3
120         6.9         3.2         5.7         2.3
18          5.7         3.8         1.7         0.3
7           5.0         3.4         1.5         0.2
14          5.8         4.0         1.2         0.2
39          5.1         3.4         1.5         0.2
```

[9]:
```
y_train
```

[9]:
```
array([0, 1, 2, 1, 2, 1, 0, 2, 1, 0, 0, 2, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 2, 0, 2, 2, 0, 2, 2, 1, 1, 0, 0, 2, 0, 1, 2, 0, 1, 1, 2, 0, 2,
       2, 2, 1, 1, 2, 2, 1, 2, 0, 0, 0, 1, 1, 0, 2, 0, 1, 2, 1, 0, 1, 1,
       0, 0, 2, 1, 2, 0, 0, 2, 0, 1, 1, 2, 1, 0, 2, 2, 0, 2, 2, 1, 0, 2,
       1, 0, 0, 1, 1, 2, 2, 2, 1, 1, 2, 2, 0, 1, 2, 1, 0])
```

[10]:
```
y_test
```

[10]:
```
array([1, 0, 1, 0, 0, 1, 1, 2, 0, 0, 0, 0, 1, 0, 0, 2, 2, 2, 1, 2, 0, 2,
       0, 1, 2, 2, 1, 0, 1, 2, 1, 2, 1, 0, 2, 2, 0, 2, 1, 2, 2, 0, 0, 0,
       0])
```

1.4 3. Train a decision tree classifier using the entropy criterion and evaluate its accuracy on the testing set.

[11]:
```
dtc_entropy = tree.DecisionTreeClassifier(criterion='entropy',␣
 ↪max_depth=4,min_samples_leaf=4)
dtc_entropy.fit(x_train, y_train)
```

[11]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=4, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[12]:
```
dtc_entropy.score(x_train, y_train)
```

[12]: 0.9714285714285714

[13]:
```
y_pred = dtc_entropy.predict(x_test)
```

[14]:
```
print("Prediction By Decision Tree (Using Entropy): ", y_pred)
```

```
Prediction By Decision Tree (Using Entropy):  [2 0 1 0 0 2 2 2 0 0 0 0 1 0 0 2 2
2 2 2 0 2 0 1 2 2 1 0 1 2 1 2 1 0 2 2 0
 2 1 2 2 0 0 0 0]
```

[15]:
```
print("Actual Result: ",y_test)
```

```
Actual Result:  [1 0 1 0 0 1 1 2 0 0 0 0 1 0 0 2 2 2 1 2 0 2 0 1 2 2 1 0 1 2 1 2
1 0 2 2 0
 2 1 2 2 0 0 0 0]
```

[16]:
```
# Return the mean accuracy on the given test data and labels.
print("Accuracy Entropy: ", dtc_entropy.score(x_test, y_test))
```

```
Accuracy Entropy:  0.9111111111111111
```

[17]:
```
# Take input the y_true(ground truth that you already know) and␣
 ↪y_pred(prediction by your model) based on that find accuracy
print("Accuracy Entropy: ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy Entropy:  0.9111111111111111
```

1.5 4. Train another decision tree classifier using the gini criterion and evaluate its accuracy on the testing set.

[18]:
```
dtc_gini = tree.DecisionTreeClassifier(criterion='gini',␣
 ↪max_depth=4,min_samples_leaf=4)
dtc_gini.fit(x_train, y_train)
```

[18]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=4, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[19]:
```
dtc_gini.score(x_train, y_train)
```

[19]: 0.9714285714285714

[20]:
```
y_pred = dtc_gini.predict(x_test)
```

```
[21]: print("Prediction By Decision Tree (Using Gini): ", y_pred)
```

```
Prediction By Decision Tree (Using Gini):  [2 0 1 0 0 2 2 2 0 0 0 0 1 0 0 2 2 2
2 2 0 2 0 1 2 2 1 0 1 2 1 2 1 0 2 2 0
 2 1 2 2 0 0 0 0]
```

```
[22]: print("Actual Result: ",y_test)
```

```
Actual Result:  [1 0 1 0 0 1 1 2 0 0 0 0 1 0 0 2 2 2 1 2 0 2 0 1 2 2 1 0 1 2 1 2
1 0 2 2 0
 2 1 2 2 0 0 0 0]
```

```
[23]: # Return the mean accuracy on the given test data and labels.
      print("Accuracy Gini: ", dtc_entropy.score(x_test, y_test))
```

```
Accuracy Gini:  0.9111111111111111
```

```
[24]: # Take input the y_true(ground truth that you already know) and␣
       ↪y_pred(prediction by your model) based on that find accuracy
      print("Accuracy Gini: ", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy Gini:  0.9111111111111111
```

1.6 5. Compare the performance of the two classifiers and discuss the differences in terms of accuracy.

- ### 1.6.1 Comparison of two classifiers in term of Performance

  - Entropy involves logarithmic computation that is computationally expensive and Gini impurity is calculated as the sum of the squared probabilities of each class in the dataset.
  - Gini Impurity is more efficient than entropy in terms of computing power.
  - As shown in graph entropy first increases up to 1 and then start decreasing on the other hand Gini impurity it only goes up to 0.5 and then start decreasing hence it requires less computational power.
  - Entropy lies in [0,1] interval and gini impurity lies in [0,0.5] interval.

- ### 1.6.2 Difference of two classifiers in term of Accuracy

  - Gini is useful when classes in the dataset are well balanced and Entropy is used when classes in the dataset are highly imbalanced. **For Example:** In the above dataset there are 50 'setosa' 50 'versicolor' and 50 'virginica' flowers in this case classes are well balanced (gini is preffered). Let's assume another case there are 10 'setosa', 40 'versicolor' and 100 'virginica' flowers. In this cases classes are not well balanced. (entropy is preffered

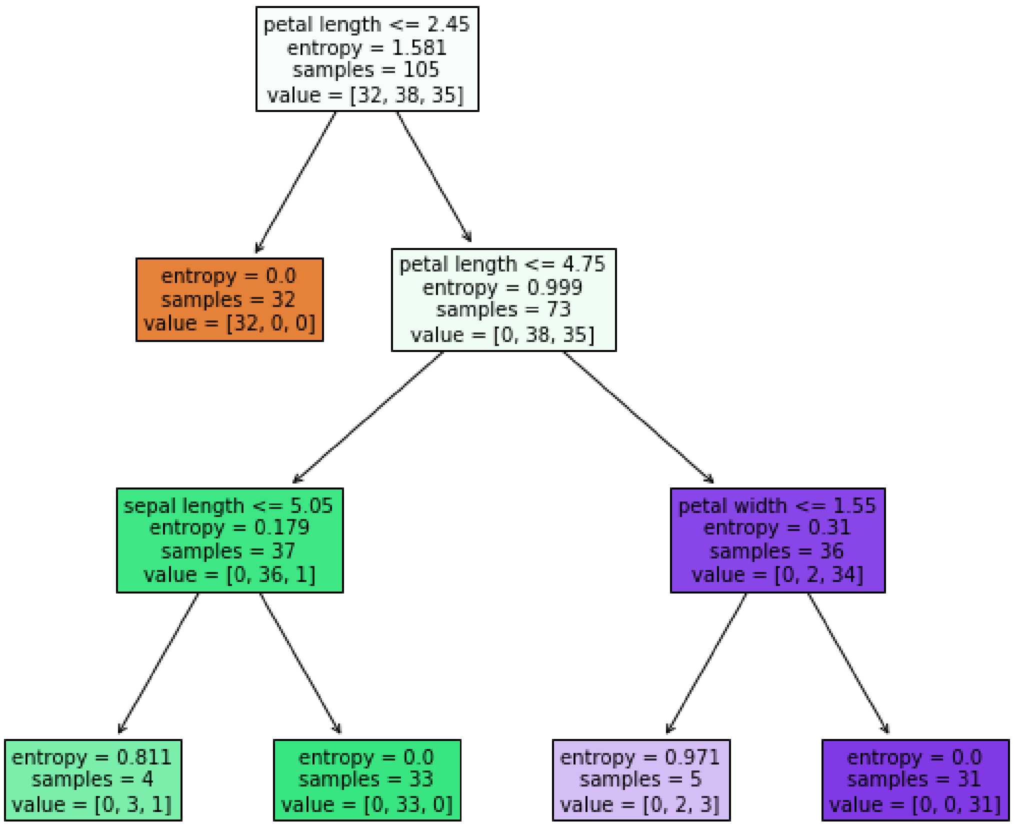
- As in above iris dataset the dataset is well balanced and we can see that accuracy for both the algorithm is same.
- It depends on the problem apply both the gini impurity and entropy and then check which criteria of decision tree suits you most.

1.7 6. Visualize the decision tree for each classifier and compare the tree structure and feature importance.

1.8 Decision Tree Using Entropy

[25]:
```
# DTC_tree = tree.plot_tree(DTC_Model)
plt.figure(figsize=(10, 9))

DTC_tree = tree.plot_tree(dtc_entropy, filled=True,
                          feature_names=['sepal length','sepal width','petal␣
 ↪length','petal width'],
                          fontsize=10,
                          )
```



Let's understand the decison tree structure with an example. **petal_length = 5** * First Question => Is petal_length less then or equal to 2.45 (No) * False so we will move to right node of the root * Second Question => Is petal length less than or equal to 4.75 (No) * Move to right node (Because of False) * Third Question => Is petal length less than or equal to 1.55 (No) * Move to Right node (Because of False) * That is the lead node so value [0, 0, 31] which is equal to ['setosa', 'versicolor', 'virginica'] * Highest count is for virginica so it will be virginica answer retured by decision tree

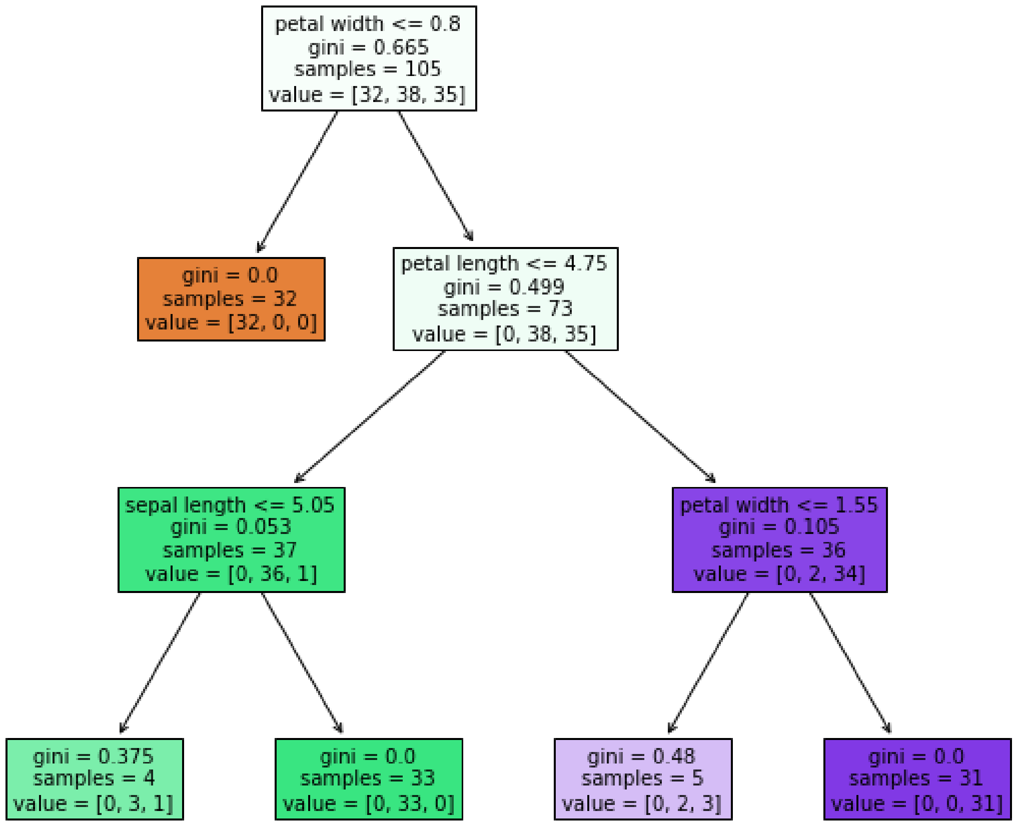
1.8.1 petal_length has the highest Entropy so it is placed on the root node

samples = 31 in the decision tree means that there are 31 flowers that lies in this category. As in the root node the samples is 105 as no question asked and all the flowers lies in this category.

2 Decision Tree Using Gini

[26]:
```
# DTC_tree = tree.plot_tree(DTC_Model)
plt.figure(figsize=(10, 9))

DTC_tree = tree.plot_tree(dtc_gini, filled=True,
                          feature_names=['sepal length','sepal width','petal␣
 ↪length','petal width'],
                          fontsize=10,
                          )
```



Let's understand the decison tree structure with an example. **petal_length = 5** * First Question => Is petal_length less then or equal to 0.8 (No) * False so we will move to right node of the root * Second Question => Is petal length less than or equal to 4.75 (No) * Move to right node (Because of False) * Third Question => Is petal length less than or equal to 1.55 (No) * Move to Right node (Because of False) * That is the lead node so value [0, 0, 31] which is equal to ['setosa', 'versicolor', 'virginica'] * Highest count is for virginica so it will be virginica answer retured by decision tree

2.0.1 petal_width has the highest gini so it is placed on the root node