**EXPERIMENT NO 11**

**AIM:** To explore decision tree models for both classification and regression tasks using the Iris dataset.

**SOFTWARE USED:** Jupyter Notebook

# THEORY:

**Decision trees** are versatile and widely used algorithms in machine learning for both classification and regression tasks. They are hierarchical structures consisting of nodes that represent decisions based on feature values. At each node, the dataset is split based on a chosen feature, aiming to maximize the homogeneity of the resulting subsets with respect to the target variable. This process is repeated recursively until certain stopping criteria are met, such as reaching a maximum depth or when further splits no longer improve the model's performance.

**Entropy and Information Gain** play crucial roles in decision tree algorithms, particularly when using entropy as the criterion for splitting. Entropy measures the level of impurity or disorder in a dataset, with lower entropy indicating higher homogeneity. Information gain quantifies the reduction in entropy achieved by splitting the data based on a specific feature. Decision trees aim to maximize information gain at each split, selecting the feature that provides the most significant reduction in entropy. By iteratively choosing the most informative features, decision trees efficiently partition the data, resulting in a series of decision rules that can accurately classify or predict the target variable.

**Gini Impurity** is an alternative criterion used in decision tree algorithms for evaluating potential splits in the data. Unlike entropy, which measures the level of disorder in a dataset, Gini impurity computes the probability of misclassifying a randomly chosen element if it were labeled according to the class distribution in the subset. Decision trees that utilize Gini impurity aim to minimize this impurity at each split, selecting features that result in the most homogeneous subsets in terms of class labels. This approach often leads to similar results as entropy-based decision trees but may offer computational advantages in certain scenarios.

**Model Evaluation:** In classification tasks, model performance is assessed using metrics such as accuracy, precision, recall, and F1-score. Accuracy gauges the proportion of correctly classified instances, while precision and recall measure the quality of positive predictions concerning all positive instances. F1-score balances precision and recall, offering a comprehensive evaluation of classification performance.

**Regression Tasks:** Decision trees are applicable to regression tasks, where the target variable is continuous. In regression trees, leaf nodes contain predicted values, typically representing the mean or median of the target

variable within the node. Evaluation in regression tasks often involves metrics like mean squared error (MSE) or R-squared (coefficient of determination), assessing the model's ability to predict continuous outcomes accurately.

**Overfitting:** Decision trees are susceptible to overfitting, particularly when the tree's depth is unrestricted. Overfitting arises when the model captures noise in the training data, hindering generalization to unseen data. To combat overfitting, techniques like pruning and limiting the tree's maximum depth are employed, ensuring the model learns meaningful patterns rather than memorizing noise.

In [1]:

**----------------------------------------------------------------------**



**-----**

**ImportError** Traceback (most recent call last)

**~\AppData\Local\Temp/ipykernel\_4704/1055167153.py** in <module>

6 **from** sklearn**.**tree **import** DecisionTreeClassifier

7 **from** sklearn **import** tree

**----> 8 from** sklearn**.**metrics **import** classificiation\_report

9 **from** sklearn **import** preprocessing

**ImportError**: cannot import name 'classificiation\_report' from 'sklear n.metrics' (C:\Users\hp\anaconda3\lib\site-packages\sklearn\metrics\ init .py)



**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt **from** sklearn **import** datasets **import** numpy **as** np

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.tree **import** DecisionTreeClassifier **from** sklearn **import** tree

**from** sklearn.metrics **import** classification\_report

**from** sklearn **import** preprocessing



iris**=**pd.read\_csv("iris.csv") iris.head()

Out[12]:

In [11]:

In [12]:

**Unnamed: 0 Sepal.Length Sepal.Width Petal.Length Petal.Width Species**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| **1** | 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| **2** | 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| **3** | 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |

In

**4** 5 5.0 3.6 1.4 0.2 setosa

In [13]: 

label\_encoder **=** preprocessing.LabelEncoder()

iris['Species'] **=** label\_encoder.fit\_transform(iris['Species'])

In [14]: 

iris

x**=**iris.iloc[:,0:4] y**=**iris["Species"]

[15]:

Out[15]:

**Unnamed: 0 Sepal.Length Sepal.Width Petal.Length Petal.Width Species**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 5.1 | 3.5 | 1.4 0.2 | 0 |  |
| **1** | 2 | 4.9 | 3.0 | 1.4 0.2 | 0 |
| **2** | 3 | 4.7 | 3.2 | 1.3 0.2 | 0 |
| **3** | 4 | 4.6 | 3.1 | 1.5 0.2 | 0 |
| **4** | 5 | 5.0 | 3.6 | 1.4 0.2 | 0 |
| **...** | ... |  | ... | ... | ... ... | ... |
| **145** | 146 | 6.7 | 3.0 | 5.2 2.3 | 2 |  |
| **146** | 147 | 6.3 | 2.5 | 5.0 1.9 | 2 |  |
| **147** | 148 | 6.5 | 3.0 | 5.2 2.0 | 2 |  |
| **148** | 149 | 6.2 | 3.4 | 5.4 2.3 | 2 |  |
| **149** | 150 | 5.9 | 3.0 | 5.1 1.8 | 2 |  |
| **150** rows × 6 columns | | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In | [16]: |  |  |  |  | x |
|  | Out[16]: | **Unnamed: 0** | **Sepal.Length** | **Sepal.Width** | **Petal.Length** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0** | 1 | 5.1 | 3.5 | 1.4 |
| **1** | 2 | 4.9 | 3.0 | 1.4 |
| **2** | 3 | 4.7 | 3.2 | 1.3 |
| **3** | 4 | 4.6 | 3.1 | 1.5 |
| **4** | 5 | 5.0 | 3.6 | 1.4 |
| **...** | ... |  | ... | ... ... |
| **145** | 146 | 6.7 | 3.0 | 5.2 |
| **146** | 147 | 6.3 | 2.5 | 5.0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In |  | | | | |
|  | **147** | 148 | 6.5 | 3.0 | 5.2 |
|  | **148** | 149 | 6.2 | 3.4 | 5.4 |
|  | **149** | 150 | 5.9 | 3.0 | 5.1 |

**150** rows × 4 columns

[17]:

y

|  |  |  |  |
| --- | --- | --- | --- |
| Out[17]: | 0 | 0 |  |
|  | 1 | 0 |
|  | 2 | 0 |
|  | 3 | 0 |
|  | 4 | 0 | .. 145 2 |
|  | 146 | 2 |  |
|  | 147 | 2 |  |
|  | 148 | 2 |  |
|  | 149 | 2 |  |

Name: Species, Length: 150, dtype: int32

iris['Species'].unique()

In [18]: 

Out[18]: array([0, 1, 2])

iris.Species.value\_counts()

In [19]: 

|  |  |
| --- | --- |
| Out[19]: 0 | 50 |
| 1 | 50 |
| 2 | 50 |

Name: Species, dtype: int64

colnames**=**list(iris.columns) colnames

In [20]: 

Out[20]: ['Unnamed: 0',

'Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width', 'Species']

In [21]: x\_train,x\_test,y\_train,y\_test**=**train\_test\_split(x,y,test\_size**=**0.2,random\_

**building decision tree classifier using entropy criteria**



In

Out[22]: [23]:

*#plot decision tree*

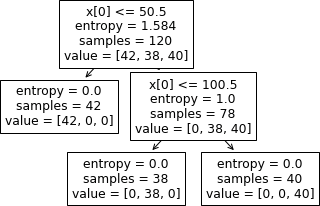
tree.plot\_tree(model)

Out[23]: [Text(0.4, 0.8333333333333334, 'x[0] <= 50.5\nentropy = 1.584\nsamples = 120\nvalue = [42, 38, 40]'),

Text(0.2, 0.5, 'entropy = 0.0\nsamples = 42\nvalue = [42, 0, 0]'), Text(0.6, 0.5, 'x[0] <= 100.5\nentropy = 1.0\nsamples = 78\nvalue = [0, 38, 40]'),

Text(0.4, 0.16666666666666666, 'entropy = 0.0\nsamples = 38\nvalue = [0, 38, 0]'),

Text(0.8, 0.16666666666666666, 'entropy = 0.0\nsamples = 40\nvalue = [0, 0, 40]')]



In

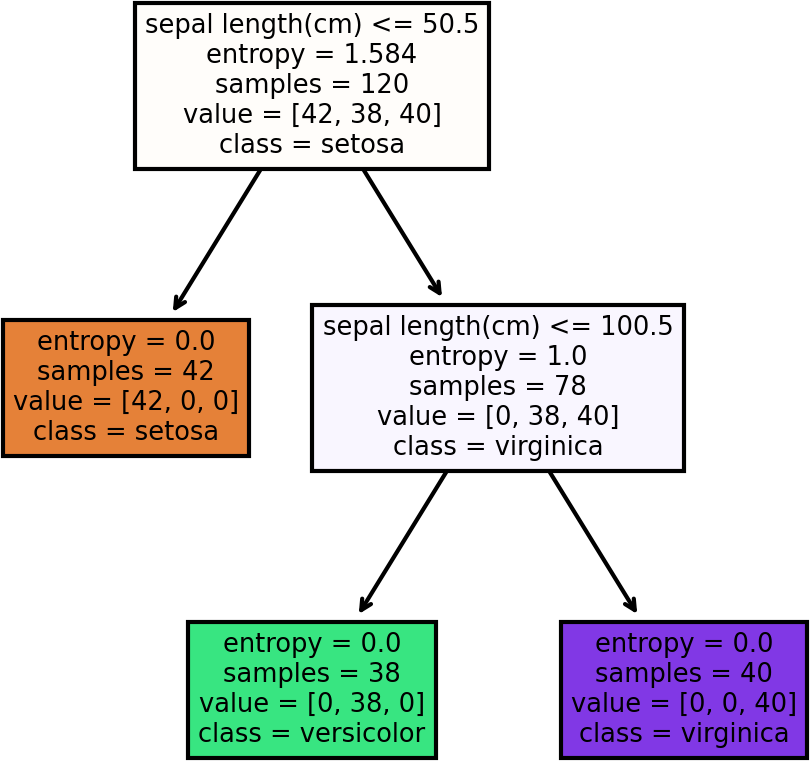
[24]:

fn**=**['sepal length(cm)', 'sepal width(cm)','petal length(cm)','peatl widt cn**=**['setosa', 'versicolor', 'virginica'] fig,axes**=**plt.subplots(nrows**=**1,ncols**=**1, figsize**=**(4,4),dpi**=**300) tree.plot\_tree(model,



feature\_names**=**fn, class\_names**=**cn, filled**=True**

);



In [25]: 

preds**=**model.predict(x\_test)

pd.Series(preds).value\_counts()

Out[25]: 1 12

2 10

0 8 dtype: int64

In [26]:



preds

In

Out[26]: array([0, 1, 2, 2, 1, 2, 1, 1, 1, 0, 1, 0, 0, 2, 1, 2, 2, 2, 1, 1, 2,

2,

1, 0, 1, 0, 0, 2, 0, 1])

pd.crosstab(y\_test,preds)

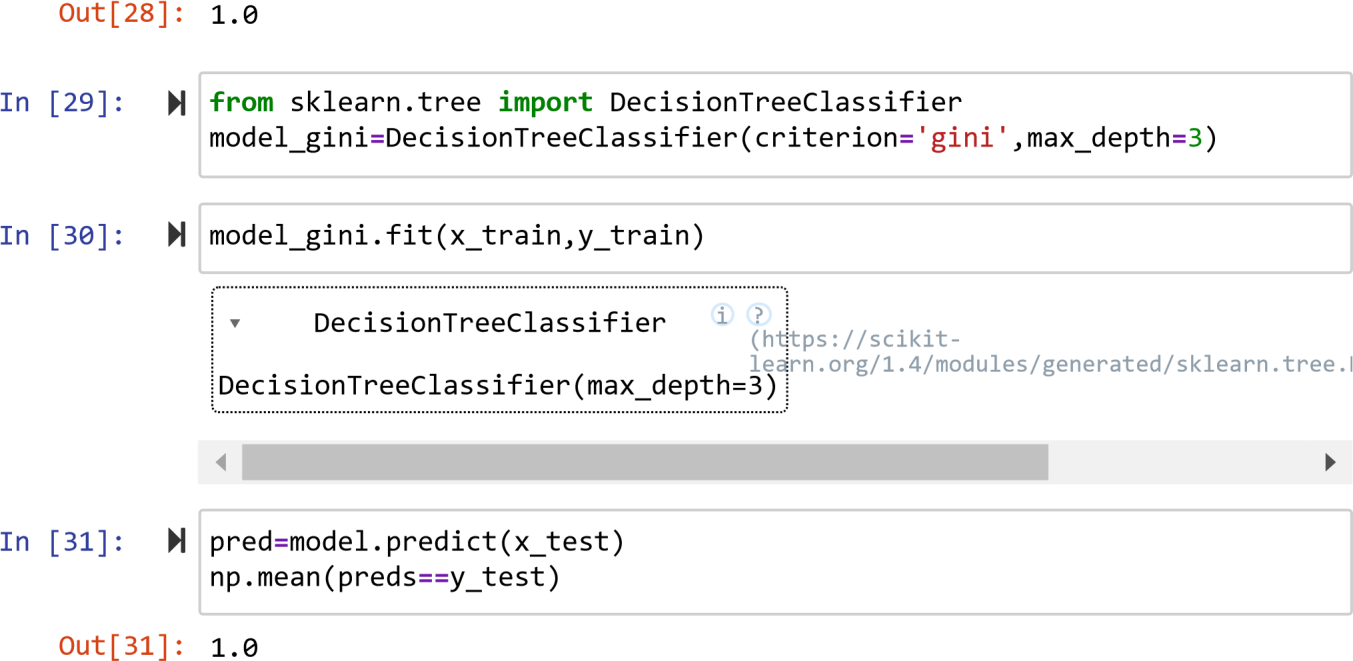
[27]:

Out[27]:

In [28]: 

**col\_0 0 1 2 Species**

|  |  |  |
| --- | --- | --- |
| **0** 8 | 0 | 0 |
| **1** 0 | 12 | 0 |
| **2** 0  np.mean(pred | 0  s | 10  **==**y\_test) |



Out[30]:



In

[35]:

model.score(X\_test,y\_test)

Out[35]: 0.9429207164394147



In [ ]: 