Python (Decision Tree)

To construct a decision tree in Python, we can import the required module from `scikit-learn` as follows:

```
from sklearn import tree
```

To import a dataset from an external file, we can use the `pandas` library to generate a `DataFrame` object. A `DataFrame` object provides easy access to the entries in the dataset. For example, we can load a dataset as follows:

```
import pandas as pd
dftrain = pd.read_csv(your file path, header= None, sep = separator of the data file)
```

We can then access the entries using `dataframe.iloc[row index, column index]`:

```
training_labels = dftrain.iloc[:,-1]
```

In the above operation, we extract all rows (‘:’) and the last column from the dataset as training labels.

If the labels are non-numeric, we can transform them into integers as follows:

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
label = le.fit_transform(training_labels)
```

Using the Iris dataset as an example, we can construct the decision tree as follows:

```
clf = tree.DecisionTreeClassifier(max_depth=3)
clf = clf.fit(iris.data, iris.target)
```

(To avoid overfitting, you can limit the depth of the decision tree by setting an appropriate `max_depth` parameter.)

To visualize the decision tree, you need to install the `python-graphviz` package.

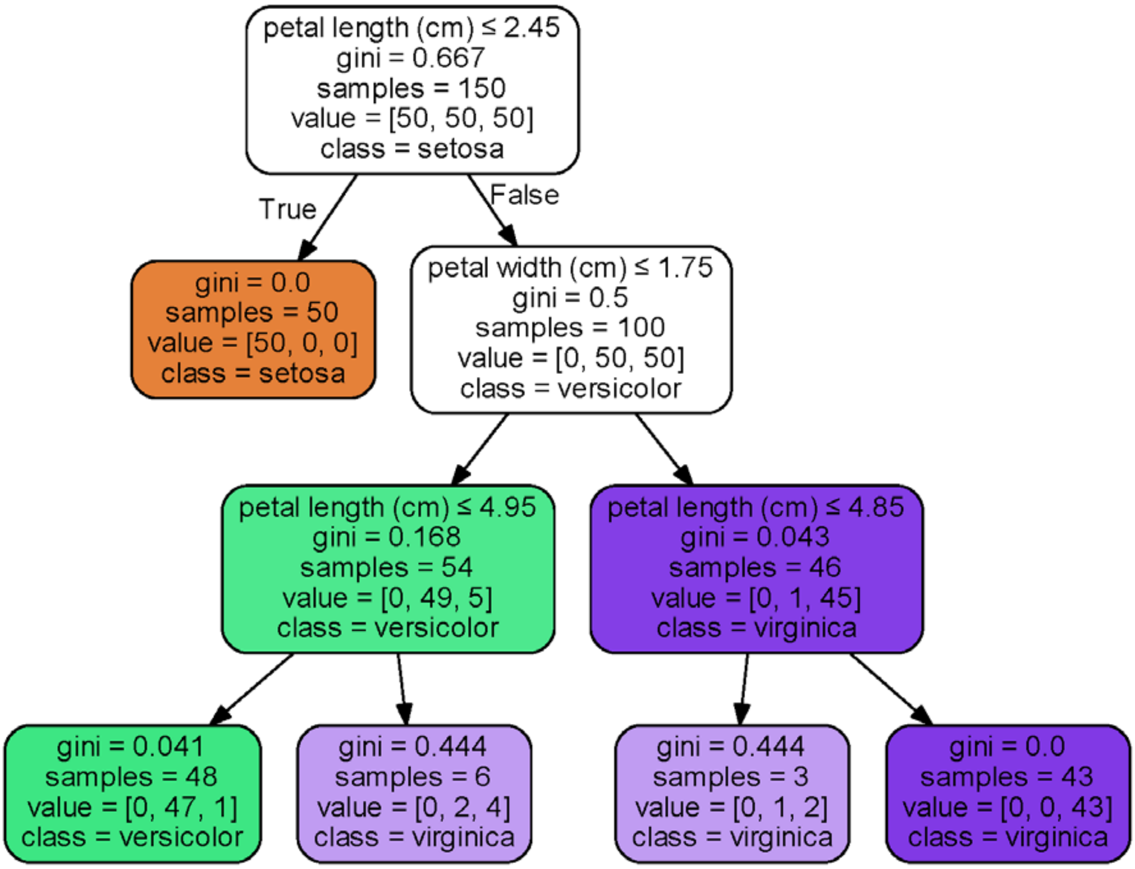
We can then plot the decision tree as follows:

```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None,
                         feature_names=iris.feature_names,
                         class_names=iris.target_names,
                         filled=True, rounded=True,
                         special_characters=True)

graph = graphviz.Source(dot_data)
```

You can save your plot to a file and display the tree as follows:

```
graph.render('Iris', view=True)
```

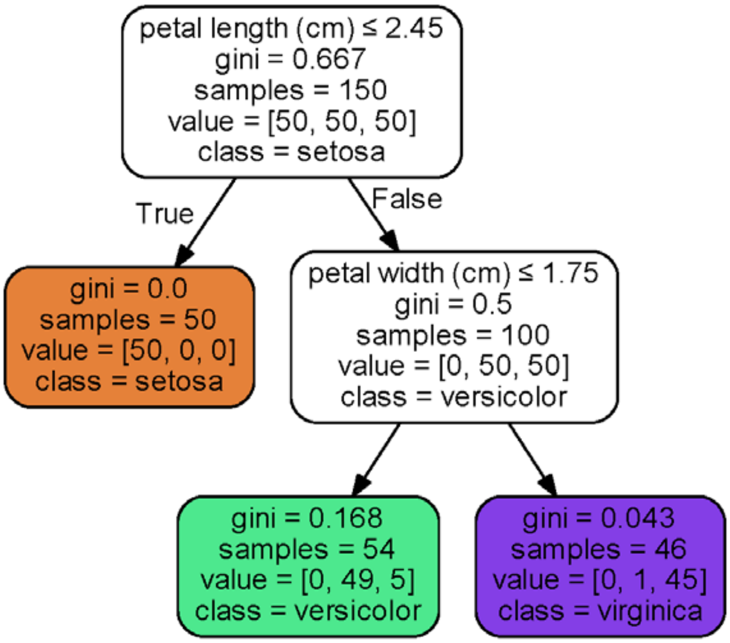


The Gini index is used by default. To construct the decision tree using information gain, the following form of `DecisionTreeClassifier` can be used:

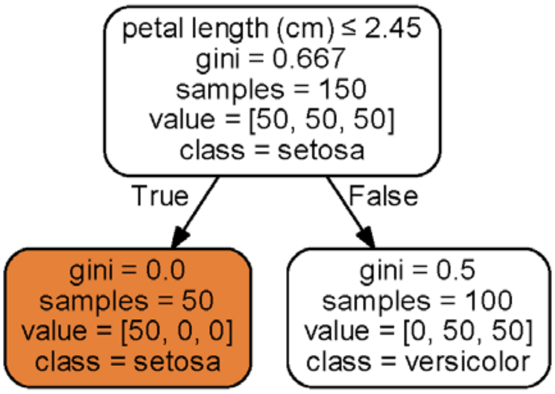
```
clf = tree.DecisionTreeClassifier(max_depth=3,criterion='entropy')
```

We can explore different possible ways of pruning the tree by limiting the maximum depth of the decision tree:

```
clf = tree.DecisionTreeClassifier(max_depth=2)
```



```
clf = tree.DecisionTreeClassifier(max_depth=1)
```



The decision tree can be applied to a new dataset as follows:

```
import numpy as np
irisTest = np.array([[4.6,3.5,1.1,0.25],[5.7,2.5,2.8,1.2],[7.3,2.8,6.6,2.2]])
prediction = clf.predict(irisTest)
print(iris.target_names[prediction])
```

Result:

```
['setosa' 'versicolor' 'virginica']
```

There are several modules in `scikit-learn` that can help you to evaluate the classification performance. One of them is `accuracy_score`:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(target, prediction))
```

The other module is `confusion_matrix`. In the resulting matrix, rows are true labels and columns are predicted labels.

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(target, prediction))
```

We can access the number of training instances belonging to each class in a node through the `tree_.value` attribute of the classifier using `print(clf.tree_.value)`

Result:

```
[[[ 50. 50. 50.]]
[[ 50. 0. 0.]]
[[ 0. 50. 50.]]
[[ 0. 49. 5.]]
[[ 0. 47. 1.]]
[[ 0. 2. 4.]]
[[ 0. 1. 45.]]
[[ 0. 1. 2.]]
[[ 0. 0. 43.]]]
```

We can observe that tree nodes are arranged in a depth-first order. We can use the method `decision_path` to analyze the decision sequence of each sample as follows:

```
print(clf.decision_path(irisTest).todense())
```

Result:

```
[[1 1 0 0 0 0 0 0 0]
 [1 0 1 1 1 0 0 0 0]
 [1 0 1 0 0 0 1 0 1]]
```

For example, the first row [1 1 0 0 0 0 0 0 0] indicates that the first test sample goes through node 0 and node 1.