SIMPLE CLASSIFICATION: Implementing Decision Trees with Python Scikit-Learn

In this section, we will implement the decision tree algorithm using Python's Scikit-Learn library. In the following examples we'll solve classification problems using the decision tree.

1. Decision Tree for Classification
In this section we will predict whether a bank note is authentic or fake depending upon the four different attributes of the image of the note. The attributes are Variance of wavelet transformed image, kurtosis of the image, entropy, and skewness of the image (refer to the dataset).

Dataset

The dataset for this task is provided.

The rest of the steps to implement this algorithm in Scikit-Learn are identical to any typical machine learning problem, we will import libraries and datasets, perform some data analysis, divide the data into training and testing sets, train the algorithm, make predictions, and finally we will evaluate the algorithm's performance on our dataset.

Importing Libraries

The following script imports required libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Importing the Dataset

Since our file is in CSV format, we will use panda's read_csv method to read our CSV data file. Execute the following script to do so:

```
dataset = pd.read_csv("bill_authentication.csv")
```

In this case the file "bill_authentication.csv" is located in the "Datasets" folder of "D" drive. You should change this path according to your own system setup.

Data Analysis

Execute the following command to see the number of rows and columns in our dataset:

```
dataset.shape
```

The output will show "(1372,5)", which means that our dataset has 1372 records and 5 attributes.

Execute the following command to inspect the first five records of the dataset:

```
dataset.head()
```

The output will look like this:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

Preparing the Data

In this section we will divide our data into attributes and labels and will then divide the resultant data into both training and test sets. By doing this we can train our algorithm on one set of data and then test it out on a completely different set of data that the algorithm hasn't seen yet. This provides you with a more accurate view of how your trained algorithm will actually perform.

To divide data into attributes and labels, execute the following code:

```
X = dataset.drop('Class', axis=1)
y = dataset['Class']
```

Here the X variable contains all the columns from the dataset, except the "Class" column, which is the label. The y variable contains the values from the "Class" column. The X variable is our attribute set and the y variable contains corresponding labels.

The final preprocessing step is to divide our data into training and test sets.
The model_selection library of Scikit-Learn contains the train_test_split method, which we'll use to randomly split the data into training and testing sets. Execute the following code to do so:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)
```

In the code above, the test_size parameter specifies the ratio of the test set, which we use to split up 20% of the data into the test set and 80% for training.

Training and Making Predictions

Once the data has been divided into the training and testing sets, the final step is to train the decision tree algorithm on this data and make predictions. Scikit-Learn contains the tree library, which contains built-in classes/methods for various decision tree algorithms. Since we are going to perform a classification task here, we will use the DecisionTreeClassifier class for this example. The fit method of this class is called to train the algorithm on the training data, which is passed as parameter to the fit method. Execute the following script to train the algorithm:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

Now that our classifier has been trained, let's make predictions on the test data. To make predictions, the predict method of the DecisionTreeClassifier class is used. Take a look at the following code for usage:

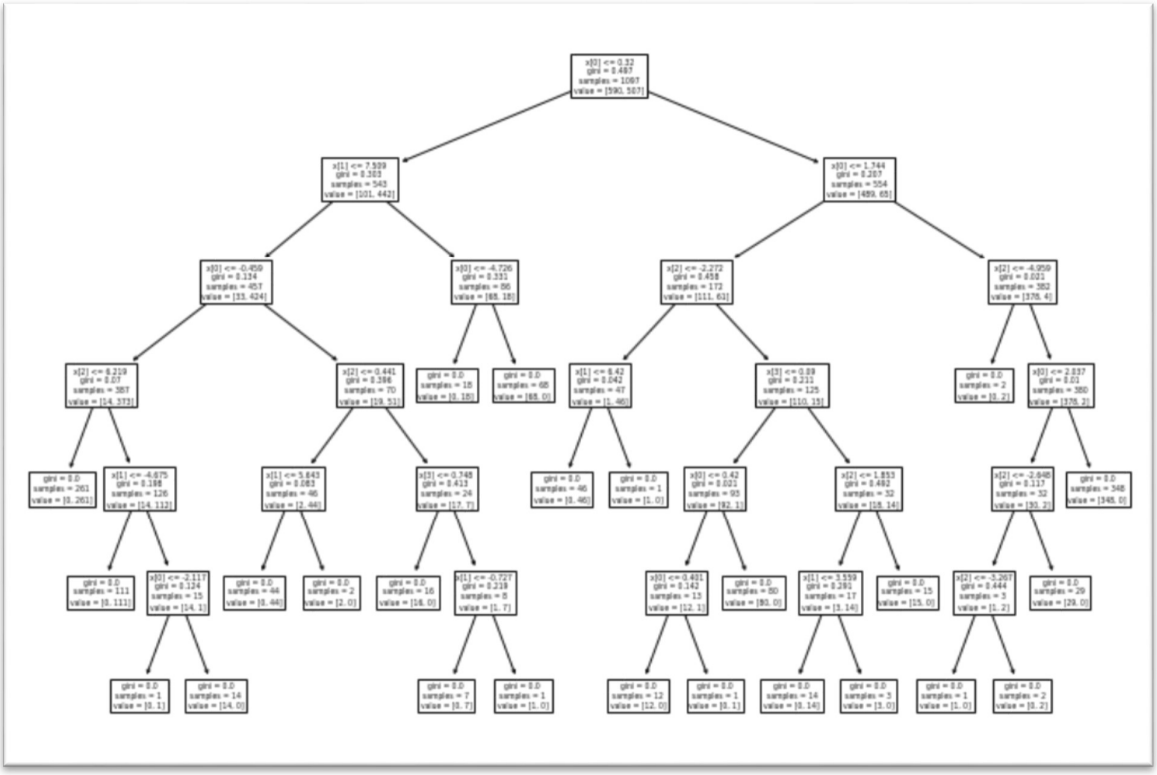
```
y_pred = classifier.predict(X_test)
```

Data Visualization

To visualize the decision tree, we use the matplotlib to plot the figure using the following `tree` function `plot_tree` as follows.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
from sklearn import tree
tree.plot_tree(classifier.fit(X_train, y_train))
```

You will then get the following decision tree shown in the plot.



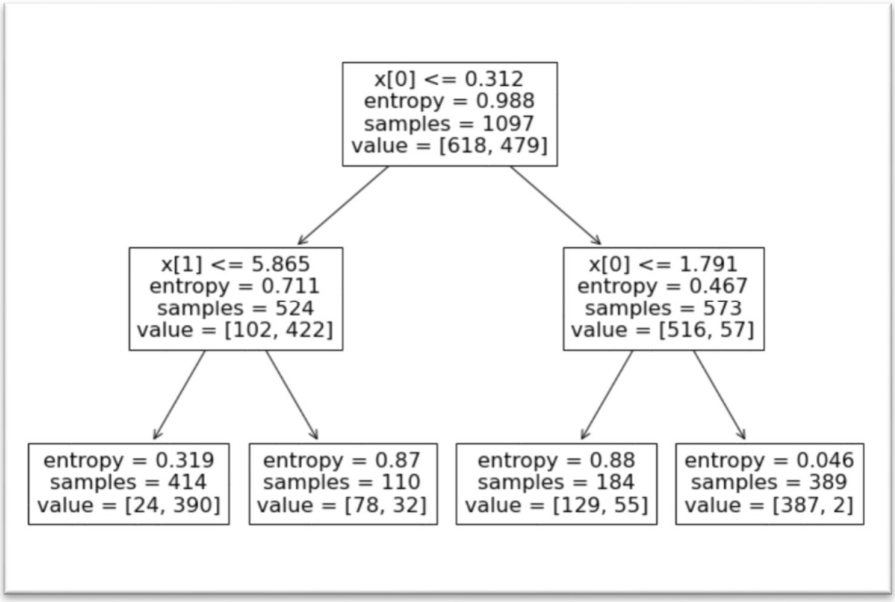
To modify the decision tree, try to use the following codes and rerun the program again.

```
classifier = DecisionTreeClassifier(criterion='entropy',
                max_depth=2, random_state=0)
```

Your decision tree now has less branches as the `max_depth` has set to 2 now and the `random_state` was set to 0. The `entropy` criterion was now used as the criterion for the decision tree splitting. A tree is composed of nodes, and those nodes are chosen looking for the optimum split of the features. For that purpose, different criteria exist. In the decision tree Python implementation of the scikit-learn library, this is made by the parameter 'criterion'. This parameter is the function used to measure the quality of a split and it allows users to choose between 'gini' or 'entropy'. Entropy is a measure of information that indicates the disorder of the features with the target. Similar to the Gini Index, the optimum split is chosen by the feature with less entropy. It gets its maximum value when the probability of the two classes is the same and a node is pure when the entropy has its minimum value, which is 0.

By default, the gini criterion will be used as the criterion for the split.

You can now modify the criterion and compare the result of your prediction using some evaluation report in the next section.



Evaluating the Algorithm

At this point we have trained our algorithm and made some predictions. Now we'll see how accurate our algorithm is. For classification tasks some commonly used metrics are confusion matrix, precision, recall, and F1 score. Lucky for us Scikit-Learn's metrics library contains the classification_report and confusion_matrix methods that can be used to calculate these metrics for us:

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

This will produce the following evaluation:

```
[[142   2]
 [  2 129]]
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	144
1	0.98	0.98	0.98	131
avg / total	0.99	0.99	0.99	275

From the confusion matrix, you can see that out of 275 test instances, our algorithm misclassified only 4. This is 98.5 % accuracy. Compare the result when using different criterion and check their accuracy.

In the next activity, you will compare several classifier using scikit-learn on synthetic datasets. The point of this example is to illustrate the nature of decision boundaries of different classifiers. This should be taken with a grain of salt, as the intuition conveyed by these examples does not necessarily carry over to real datasets.

Particularly in high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as naive Bayes and linear SVMs might lead to better generalization than is achieved by other classifiers.

The plots show training points in solid colors and testing points semi-transparent. The lower right shows the classification accuracy on the test set.