Machine Learning
Lab 3

Advantages of Decision Trees

There are several advantages of using decision trees for predictive analysis:

- Decision trees can be used to predict both continuous and discrete values i.e. they work well for both regression and classification tasks.
- They require relatively less effort for training the algorithm.
- They can be used to classify non-linearly separable data.
- They're very fast and efficient compared to KNN and other classification algorithms.
- 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 both classification as well as regression problems using the decision tree.

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, curtosis of the image, entropy, and skewness of the image.

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
%matplotlib inline
```

Importing the Dataset

Since our file is in CSV format, we will use panda's read_csv method to read our CSV data file. Download the data ‘bill_aythentication.csv’ from Moodle and store the data into variable ‘dataset’.

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 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 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 in to 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)
```

Optional: Evaluating the Algorithm (we will learn this next week)

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. Not too bad!

Activity:

Download the data ‘PastHires.csv’ from Moodle and use it as training dataset. Build a decision tree, a random forest, and a naïve bayes classifier. Generate a test data sample yourself and check if classification results are the same.

Hint: You will need to convert the categorical variable into values.