Lab Assignment 6 - Decision Trees

In this lab exercise, you will learn a popular machine learning algorithm, Decision Tree. You will use this classification algorithm to build a model from historical data of patients, and their respond to different medications. Then you use the trained decision tree to predict the class of a unknown patient, or to find a proper drug for a new patient.

Import the Following Libraries:

- **numpy (as np)**
- **pandas**
- **DecisionTreeClassifier** from **sklearn.tree**

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

About dataset

Imagine that you are a medical researcher compiling data for a study. You have collected data about a set of patients, all of whom suffered from the same illness. During their course of treatment, each patient responded to one of 5 medications, Drug A, Drug B, Drug c, Drug x and y.

Part of your job is to build a model to find out which drug might be appropriate for a future patient with the same illness. The feature sets of this dataset are Age, Sex, Blood Pressure, and Cholesterol of patients, and the target is the drug that each patient responded to.

It is a sample of binary classifier, and you can use the training part of the dataset to build a decision tree, and then use it to predict the class of a unknown patient, or to prescribe it to a new patient.

now, read data using pandas dataframe:

```
my_data = pd.read_csv("/content/drug200.csv", delimiter=",")
my_data[0:5]
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

Practice

What is the size of data?

```
# write your code here
data_size = my_data.size
print("The size of this data is: ",data_size)
```

```
The size of this data is:  1200
```

Pre-processing

Using **my_data** as the Drug.csv data read by pandas, declare the following variables:

- **X** as the **Feature Matrix** (data of my_data)
- **y** as the **response vector (target)**

Remove the column containing the target name since it doesn't contain numeric values.

```
X = my_data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[0:5]
```

```
array([[23, 'F', 'HIGH', 'HIGH', 25.355],
       [47, 'M', 'LOW', 'HIGH', 13.093],
       [47, 'M', 'LOW', 'HIGH', 10.114],
       [28, 'F', 'NORMAL', 'HIGH', 7.798],
       [61, 'F', 'LOW', 'HIGH', 18.043]], dtype=object)
```

As you may figure out, some featurs in this dataset are catergorical such as **Sex** or **BP**. Unfortunately, Sklearn Decision Trees do not handle categorical variables. But still we can convert these features to numerical values. **pandas.get_dummies()** Convert categorical variable into dummy/indicator variables.

```
from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
X[:,1] = le_sex.transform(X[:,1])


le_BP = preprocessing.LabelEncoder()
le_BP.fit([ 'LOW', 'NORMAL', 'HIGH'])
X[:,2] = le_BP.transform(X[:,2])


le_Chol = preprocessing.LabelEncoder()
le_Chol.fit([ 'NORMAL', 'HIGH'])
X[:,3] = le_Chol.transform(X[:,3])

X[0:5]

```

```
array([[23, 0, 0, 0, 25.355],
       [47, 1, 1, 0, 13.093],
       [47, 1, 1, 0, 10.114],
       [28, 0, 2, 0, 7.798],
       [61, 0, 1, 0, 18.043]], dtype=object)
```

Now we can fill the target variable.

```
y = my_data["Drug"]
y[0:5]
```

```
0    drugY
1    drugC
2    drugC
3    drugX
4    drugY
Name: Drug, dtype: object
```

Setting up the Decision Tree

We will be using **train/test split** on our **decision tree**. Let's import **train_test_split** from **sklearn.cross_validation**.

```
from sklearn.model_selection import train_test_split
```

Now **train_test_split** will return 4 different parameters. We will name them:
X_trainset, X_testset, y_trainset, y_testset

The **train_test_split** will need the parameters:
X, y, test_size=0.3, and random_state=3.

The **X** and **y** are the arrays required before the split, the **test_size** represents the ratio of the testing dataset, and the **random_state** ensures that we obtain the same splits.

```
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=3)
```

Practice

Print the shape of X_trainset and y_trainset. Ensure that the dimensions match

```
# your code
print("The X_trainset is: ",X_trainset.shape)
print ("The y_trainset is: ",y_trainset.shape)

if X_trainset.shape[0] == y_trainset.shape[0]:
    print("Everything is ok, both values from X_trainset and y_trainset are equal")
else:
    print("Error, X_trainset and y_trainset has different values")
```

```
The X_trainset is:  (140, 5)
The y_trainset is:  (140,)
Everything is ok, both values from X_trainset and y_trainset are equal
```

Print the shape of X_testset and y_testset. Ensure that the dimensions match

```
# your code
print("The X_trainset is: ",X_testset.shape)
print ("The y_trainset is: ",y_testset.shape)

if X_testset.shape[0] == y_testset.shape[0]:
    print("Everything is ok, both values from X_trainset and y_trainset are equal")
else:
    print("Error, X_trainset and y_trainset has different values")


```

```
The X_trainset is:  (60, 5)
The y_trainset is:  (60,)
Everything is ok, both values from X_trainset and y_trainset are equal
```

Modeling

We will first create an instance of the **DecisionTreeClassifier** called **drugTree**.
Inside of the classifier, specify *criterion="entropy"* so we can see the information gain of each node.

```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree # it shows the default parameters
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Next, we will fit the data with the training feature matrix **X_trainset** and training response vector **y_trainset**

```
drugTree.fit(X_trainset,y_trainset)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Prediction

Let's make some **predictions** on the testing dataset and store it into a variable called **predTree**.

```
predTree = drugTree.predict(X_testset)
```

You can print out **predTree** and **y_testset** if you want to visually compare the prediction to the actual values.

```
print (predTree [0:5])
print (y_testset [0:5])

```

```
['drugY' 'drugX' 'drugX' 'drugX' 'drugX']
40     drugY
51     drugX
139    drugX
197    drugX
170    drugX
Name: Drug, dtype: object
```

Evaluation

Next, let's import **metrics** from sklearn and check the accuracy of our model.

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
xteste = np.array(predTree)
xteste[0]

#predTree[0:1]
```

```
DecisionTrees's Accuracy:  0.9833333333333333
'drugY'
```

Accuracy classification score computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.

In multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

Practice

Can you calculate the accuracy score without sklearn ?

```
# your code here

i = y_testset.size
x = 0
ok = 0
fail = 0
y_testset_comp = np.array(y_testset)
predTree_comp = np.array(predTree)

for x in range(0,i):
    if y_testset_comp[x] == predTree_comp[x]:
        ok = ok + 1
    else:
        fail = fail + 1

accuracy = ok/i
print(accuracy)

```

```
0.9833333333333333
```

Visualization

Lets visualize the tree

```
!pip install six
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (1.16.0)
```

```
from six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

```
dot_data = StringIO()
filename = "drugtree.png"
featureNames = my_data.columns[0:5]
targetNames = my_data["Drug"].unique().tolist()
out=tree.export_graphviz(drugTree,feature_names=featureNames, out_file=dot_data, class_names= np.unique(y_trainset), filled=True,  spe
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img,interpolation='nearest')
```

```
<matplotlib.image.AxesImage at 0x7f06502332e8>
```