IBM Developer
SKILLS NETWORK

Decision Trees

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Develop a classification model using Decision Tree Algorithm

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

Table of contents

Import the Following Libraries:

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

In [1]:

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

About the 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 features of this dataset are Age, Sex, Blood Pressure, and the Cholesterol of the patients, and the target is the drug that each patient responded to.

It is a sample of multiclass 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 an unknown patient, or to prescribe a drug to a new patient.

Downloading the Data

To download the data, we will use !wget to download it from IBM Object Storage.

In [2]:

```
!wget -O drug200.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
```

```
--2021-11-07 01:05:52--  https://cf-courses-data.s3.us.cloud-object-storage.appdomai
n.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/drug20
0.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses
-data.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5827 (5.7K) [text/csv]
Saving to: ‘drug200.csv’

drug200.csv         100%[===================>]   5.69K  --.-KB/s    in 0s
```

```
2021-11-07 01:05:52 (129 MB/s) - ‘drug200.csv’ saved [5827/5827]
```

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: Sign up now for free

Now, read the data using pandas dataframe:

In [3]:

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

Out[3]:

	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?

In [4]:

```
my_data.shape
```

Out[4]:

```
(200, 6)
```

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.

In [5]:

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

Out[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 features in this dataset are categorical, such as **Sex** or **BP**.

Unfortunately, Sklearn Decision Trees does not handle categorical variables. We can still convert these features to numerical values using **pandas.get_dummies()** to convert the categorical variable into dummy/indicator variables.

In [6]:

```
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]
```

Out[6]:

```
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.

In [7]:

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

Out[7]:

```
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**.

In [8]:

```
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.

In [9]:

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

Practice

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

In [10]:

```
print('Shape of X training set {}'.format(X_trainset.shape),'&',' Size of Y training
```

```
Shape of X training set (140, 5) &  Size of Y training set (140,)
```

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

In [11]:

```
print('Shape of X test set {}'.format(X_testset.shape),'&',' Size of Y test set {}'.
```

```
Shape of X test set (60, 5) &  Size of Y test set (60,)
```

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.

In [12]:

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

Out[12]:

```
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**

In [13]:

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

Out[13]:

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

Prediction

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

In [14]:

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

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

In [15]:

```
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.

In [16]:

```
from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))
```

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

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 matches with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

Visualization

Let's visualize the tree

In [ ]:

```
#!conda install -c conda-forge pydotplus -y
#!conda install -c conda-forge python-graphviz -y
```

In [21]:

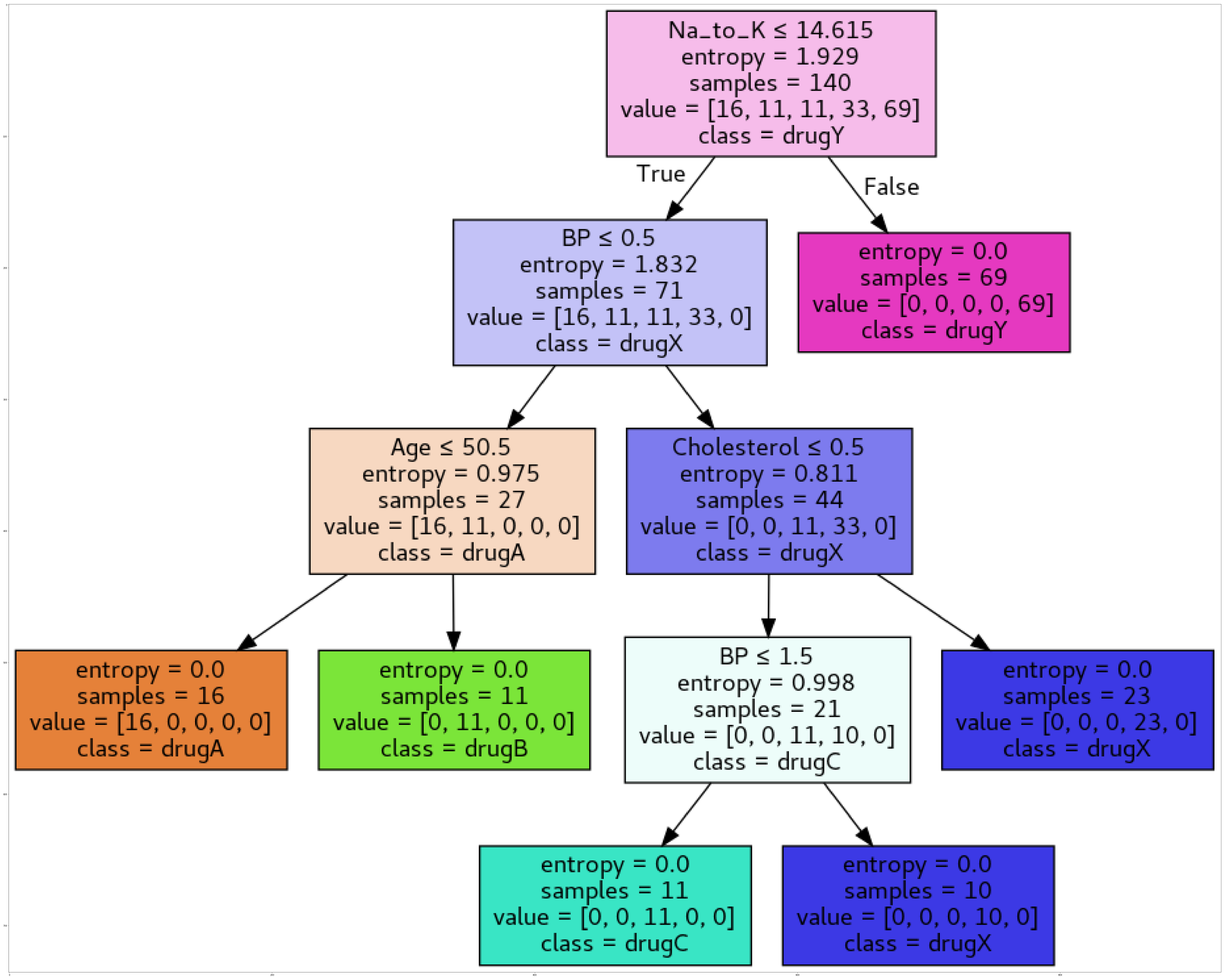
```
from  io import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

In [19]:

```
dot_data = StringIO()
filename = "drugtree.png"
featureNames = my_data.columns[0:5]
out=tree.export_graphviz(drugTree,feature_names=featureNames, out_file=dot_data, cla
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')
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x7efe309ef670>
```



Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

Thank you for completing this lab!

Author

Saeed Aghabozorgi

Other Contributors

Joseph Santarcangelo

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-11-20	2.2	Lakshmi	Changed import statement of StringIO
2020-11-03	2.1	Lakshmi	Changed URL of the csv
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab