(https://www.bigdatauniversity.com)

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 response 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**

In [1]:
```
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.

Downloading Data

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

In [2]:
```
!wget -O drug200.csv https://s3-api.us-geo.objectstorage.softlayer.net/c
f-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
print('Downloaded....Got It!')
```

```
--2018-11-20 19:46:20--  https://s3-api.us-geo.objectstorage.softlayer.
net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.obje
ctstorage.softlayer.net)... 67.228.254.193
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.
objectstorage.softlayer.net)|67.228.254.193|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6027 (5.9K) [text/csv]
Saving to: ‘drug200.csv’

drug200.csv         100%[===================>]   5.89K  --.-KB/s   in
0s

2018-11-20 19:46:20 (99.9 MB/s) - ‘drug200.csv’ saved [6027/6027]

Downloaded....Got It!
```

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 (http://cocl.us/ML0101EN-IBM-Offer-CC)

now, read 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]:
```
# write your code here
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', 10.113999999999999],
       [28, 'F', 'NORMAL', 'HIGH', 7.797999999999999],
       [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 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.

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, 10.113999999999999],
       [28, 0, 2, 0, 7.797999999999999],
       [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, te
st_size=0.3, random_state=3)
```

Practice

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

In [10]:
```
# your code
print('Train set size:', X_trainset.shape, y_trainset.shape)
```

```
Train set size: (140, 5) (140,)
```

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

In [11]:
```
# your code
print('Test set size:', X_testset.shape, y_testset.shape)
```

```
Test set size: (60, 5) (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(class_weight=None, criterion='entropy', max_dept
h=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
```

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(class_weight=None, criterion='entropy', max_dept
h=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
```

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 prediction to the actual values.

In [15]:
```
print('PREDTREE')
print (predTree [0:5])
print('\n')
print('Y_TESTSET')
print (y_testset [0:5])
```

```
PREDTREE
['drugY' 'drugX' 'drugX' 'drugX' 'drugX']


Y_TESTSET
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, pr
edTree))
```

```
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 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 ?

In [17]:
```
# your code here
#NOPE!
```

Visualization

Lets visualize the tree

In [18]:
```
# Notice: You might need to uncomment and install the pydotplus and grap
hviz libraries if you have not installed these before
!conda install -c conda-forge pydotplus -y
!conda install -c conda-forge python-graphviz -y
```

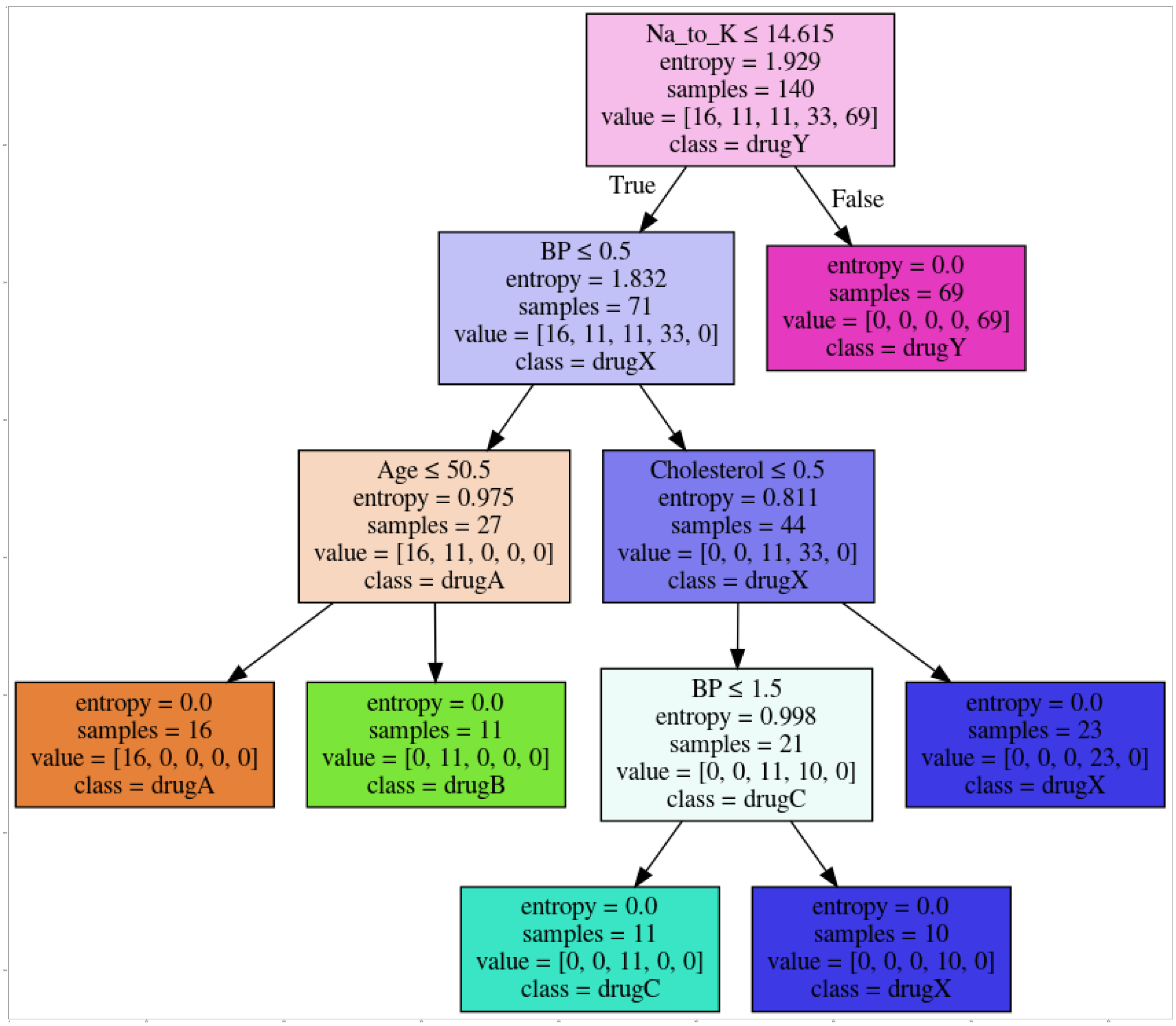
```
Solving environment: done

## Package Plan ##

  environment location: /home/jupyterlab/conda

  added / updated specs:
    - pydotplus


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-1.0.2p             |       h470a237_1         3.1 MB  conda
-forge
    certifi-2018.10.15         |        py36_1000         138 KB  conda
-forge
    pydotplus-2.0.2            |             py_1          22 KB  conda
-forge
    ca-certificates-2018.10.15 |       ha4d7672_0         135 KB  conda
-forge
    conda-4.5.11               |        py36_1000         651 KB  conda
-forge
    ------------------------------------------------------------
                                           Total:         4.0 MB

The following packages will be UPDATED:

    ca-certificates: 2018.8.24-ha4d7672_0 conda-forge --> 2018.10.15-ha
4d7672_0 conda-forge
    certifi:         2018.8.24-py36_1001  conda-forge --> 2018.10.15-py
36_1000  conda-forge
    conda:           4.5.11-py36_0        conda-forge --> 4.5.11-py36_1
000      conda-forge
    openssl:         1.0.2p-h470a237_0    conda-forge --> 1.0.2p-h470a2
37_1     conda-forge
    pydotplus:       2.0.2-py36_0         anaconda    --> 2.0.2-py_1
         conda-forge


Downloading and Extracting Packages
openssl-1.0.2p       | 3.1 MB    | ####################################
# | 100%
certifi-2018.10.15   | 138 KB    | ####################################
# | 100%
pydotplus-2.0.2      | 22 KB     | ####################################
# | 100%
ca-certificates-2018 | 135 KB    | ####################################
# | 100%
conda-4.5.11         | 651 KB    | ####################################
# | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Solving environment: done
```

```
## Package Plan ##

  environment location: /home/jupyterlab/conda

  added / updated specs:
    - python-graphviz


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    python-graphviz-0.8.4      |        py36_1002          27 KB  conda
-forge

The following NEW packages will be INSTALLED:

    python-graphviz: 0.8.4-py36_1002 conda-forge


Downloading and Extracting Packages
python-graphviz-0.8. | 27 KB     | #####################################
# | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

In [19]:
```
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

In [20]:
```
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=d
ot_data, class_names= np.unique(y_trainset), filled=True,  special_chara
cters=True,rotate=False)
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[20]: <matplotlib.image.AxesImage at 0x7f38ea4f8748>



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 (http://cocl.us/ML0101EN-SPSSModeler).

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 (https://cocl.us/ML0101EN_DSX)

Thanks for completing this lesson!

Notebook created by: Saeed (https://ca.linkedin.com/in/saeedaghabozorgi">Saeed) Aghabozorgi

---