



(<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.

Table of contents

1. [About the dataset](#)
2. [Downloading the Data](#)
3. [Pre-processing](#)
4. [Setting up the Decision Tree](#)
5. [Modeling](#)
6. [Prediction](#)
7. [Evaluation](#)
8. [Visualization](#)

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 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 the 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-gio.objectstorage.softlayer.net/cf-course
s-data/CognitiveClass/ML0101ENv3/labs/drug200.csv

--2020-02-03 13:26:03-- https://s3-api.us-gio.objectstorage.softlayer.net/cf-
courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
Resolving s3-api.us-gio.objectstorage.softlayer.net (s3-api.us-gio.objects
torage.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-gio.objectstorage.softlayer.net (s3-api.us-gio.objects
torage.softlayer.net)|67.228.254.196|: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

2020-02-03 13:26:03 (17.1 MB/s) - 'drug200.csv' saved [6027/6027]
```

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 \(http://cocl.us/ML0101EN-IBM-Offer-CC\)](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_Cholesterol = preprocessing.LabelEncoder()
le_Cholesterol.fit(['NORMAL', 'HIGH'])
X[:,3] = le_Cholesterol.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, test_size=
        0.3, random_state=3)
```

Practice

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

```
In [19]: # your code
        print("X: ", X_trainset.shape)
        print("y: ", y_trainset.shape)

X:  (140, 5)
y:  (140,)
```

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

```
In [20]: # your code
        print("X_test: ", X_testset.shape)
        print("y_test: ", y_testset.shape)

X_test:  (60, 5)
y_test:  (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 [21]: drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
         drugTree # it shows the default parameters
```

```
Out[21]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=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=None,
                                splitter='best')
```

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

```
In [22]: drugTree.fit(X_trainset,y_trainset)
```

```
Out[22]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=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=None,
                                splitter='best')
```

Prediction

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

```
In [23]: 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 [24]: 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 [25]: 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 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 [28]: # your code here
```

Visualization

Lets visualize the tree

```
In [29]: # Notice: You might need to uncomment and install the pydotplus and graphviz 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

```
==> WARNING: A newer version of conda exists. <==
current version: 4.5.11
latest version: 4.8.1
```

Please update conda by running

```
$ conda update -n base -c defaults conda
```

Package Plan

environment location: /home/jupyterlab/conda/envs/python

```
added / updated specs:
- pydotplus
```

The following packages will be downloaded:

package	build		
scikit-learn-0.20.1	py36h22eb022_0	5.7 MB	
certifi-2019.11.28	py36_0	149 KB	conda-forge
liblapack-3.8.0	11_openblas	10 KB	conda-forge
liblapacke-3.8.0	11_openblas	10 KB	conda-forge
libopenblas-0.3.6	h5a2b251_2	7.7 MB	
scipy-1.4.1	py36h921218d_0	18.9 MB	conda-forge
libcbblas-3.8.0	11_openblas	10 KB	conda-forge
libblas-3.8.0	11_openblas	10 KB	conda-forge
numpy-1.17.5	py36h95a1406_0	5.2 MB	conda-forge
blas-2.11	openblas	10 KB	conda-forge
Total:		37.7 MB	

The following NEW packages will be INSTALLED:

libblas:	3.8.0-11_openblas	conda-forge
libcbblas:	3.8.0-11_openblas	conda-forge
liblapack:	3.8.0-11_openblas	conda-forge
liblapacke:	3.8.0-11_openblas	conda-forge
libopenblas:	0.3.6-h5a2b251_2	

The following packages will be UPDATED:

blas:	1.1-openblas	conda-forge --> 2.11-
openblas	conda-forge	
certifi:	2019.9.11-py36_0	conda-forge --> 2019.
11.28-py36_0	conda-forge	
numpy:	1.16.2-py36_blas_openblas	conda-forge [blas_ope
nblas] --> 1.17.5-py36h95a1406_0	conda-forge	
scipy:	1.2.1-py36_blas_openblas	conda-forge [blas_ope
nblas] --> 1.4.1-py36h921218d_0	conda-forge	

The following packages will be DOWNGRADED:

scikit-learn:	0.20.1-py36_blas_openblas	conda-forge [blas_ope
nblas] --> 0.20.1-py36h22eb022_0		

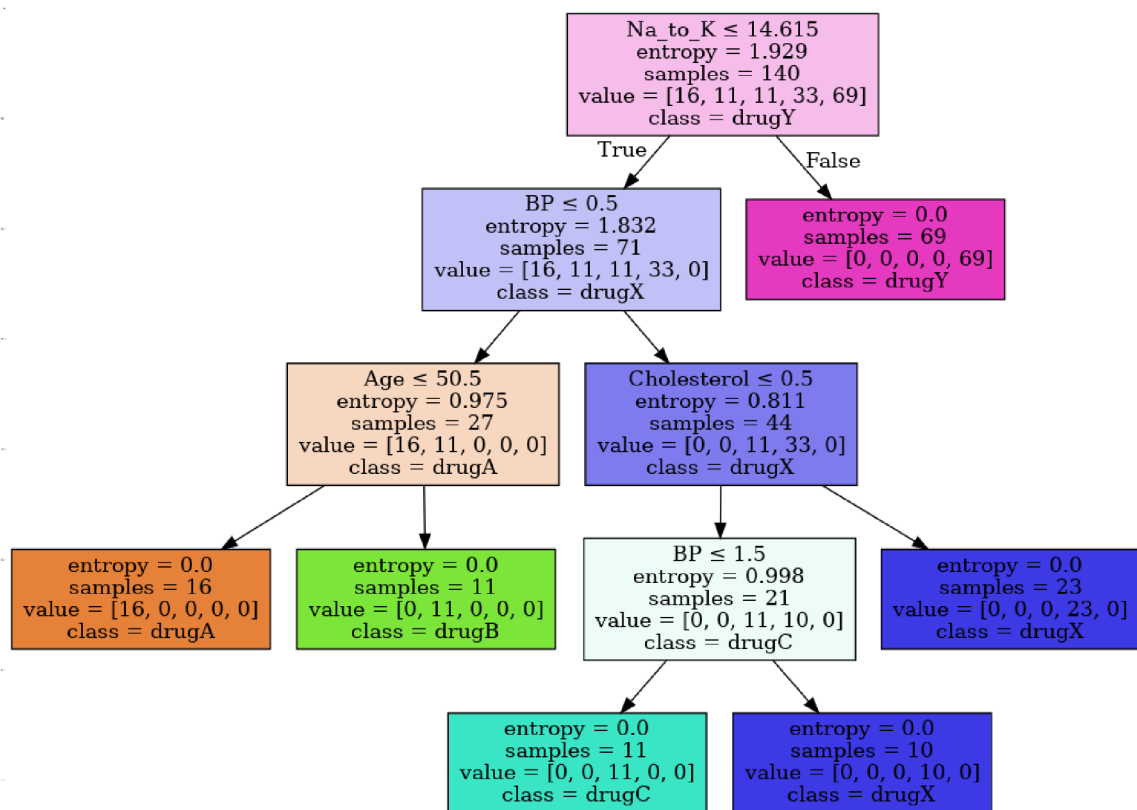
Downloading and Extracting Packages

scikit-learn-0.20.1	5.7 MB	#####	10
0%			
certifi-2019.11.28	149 KB	#####	10
0%			
liblapack-3.8.0	10 KB	#####	10


```
In [30]: from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

```
In [31]: 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, special_characters=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[31]: <matplotlib.image.AxesImage at 0x7ffbc845c5f8>



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 \(http://cocl.us/ML0101EN-SPSSModeler\)](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 \(https://cocl.us/ML0101EN_DSX\)](https://cocl.us/ML0101EN_DSX).

Thanks for completing this lesson!

Author: [Saeed Aghabozorgi \(https://ca.linkedin.com/in/saeedaghabozorgi\)](https://ca.linkedin.com/in/saeedaghabozorgi)

[Saeed Aghabozorgi \(https://ca.linkedin.com/in/saeedaghabozorgi\)](https://ca.linkedin.com/in/saeedaghabozorgi), PhD is a Data Scientist in IBM with a track record of developing enterprise level applications that substantially increases clients' ability to turn data into actionable knowledge. He is a researcher in data mining field and expert in developing advanced analytic methods like machine learning and statistical modelling on large datasets.

Copyright © 2018 [Cognitive Class \(https://cocl.us/DX0108EN_CC\)](https://cocl.us/DX0108EN_CC). This notebook and its source code are released under the terms of the [MIT License \(https://bigdatauniversity.com/mit-license/\)](https://bigdatauniversity.com/mit-license/).