(https://www.bigdatauniversity.com)

Decision Trees

En este ejercicio, aprenderás un algoritmo muy popular de machine learning llamado Árboles de Decisión. Utilizarás un algoritmo de clasificación para construir un modelo basado en datos históricos de pacientes y sus respectivos medicamentos. Luego, utilizarás el árbol de decisión recién entrenado para predecir la clase de paciente desconocido o para encontrar la droga adecuada para el mismo.

Import the Following Libraries:

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

In [127]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

Acerca del set de datos

Imagina que eres un investigador médico recolectando datos para un estudio. Has colectado datos de un grupo de pacientes, todos sufrieron la misma enfermedad. Durante su tratamiento, cada paciente respondio a una de 5 medicaciones, Droga A, Droga B, Droga c, Droga x e y.

Parte de tu trabajo es construir un modelo para encontrar la droga apropiada para un próximo paciente con la misma enfermedad. El conjunto de características son Edad, Sexo, Presión Sanguínea y Colesterol. El objetivo es la droga ante la cual cada paciente respondió.

Este es un ejemplo de un clasificador binario donde puedes utilizar un set de entrenamiento del set de datos para construir un árbol de decisión para predecir la clase de pacientes desconocidos o para prescribirle a un nuevo paciente.

Descargando los Datos

Para descagar los datos, utilizaremos !wget desde IBM Object Storage.

In [128]:
```
!wget -O drug200.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-cours
es-data/CognitiveClass/ML0101ENv3/labs/drug200.csv
```

```
--2020-03-31 19:05:57--  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.objectstora
ge.softlayer.net)... 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-geo.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 0.001s

2020-03-31 19:05:57 (10.1 MB/s) - 'drug200.csv' saved [6027/6027]
```

¿Sabías? Cuando se trata de Machine Learning, seguro trabajarás con grandes datasets (juego de datos). Entonces, ¿dónde podrás guardar esos datos? IBM ofrece una oportunidad única para las empresas, con 10 Tb de IBM Cloud Object Storage: Regístrate ahora gratuitamente (http://cocl.us/ML0101EN-IBM-Offer-CC)

ahora, lee los datos utilizando el marco de datos de panda:

In [129]:
```
my_data = pd.read_csv("drug200.csv", delimiter=",")
my_data[0:5]
```

Out[129]:

	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

Práctica

¿Cuál es el tamaño de los datos?

In [ ]:
```
# escribe tu código aquí
```

Pre-procesamiento

Utilizando **my_data** como los datos de panda el archivo Drug.csv, declara las siguientes variables:

- **X** as the **Feature Matrix** (datos de my_data)
- **y** como el **vector de respuesta (target)**

Elimina la columna que contiene el target ya que no posee valores numéricos.

In [130]:
```
X = my_data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[0:5]
```

Out[130]:
```
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)
```

Como te puedes imaginar, algunas características son de categoría, tales como **Sex** o**BP**. Desafortunadamente, los árboles de Decisión Sklearn no manejan variables categóricas. Pero las podemos convertir en valores numéricos. **pandas.get_dummies()** Convertir variable categórica en indicadores de variables.

In [131]:
```
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[131]:
```
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)
```

Ahora, podemos completar la variable objetivo (target).

In [132]:
```
y = my_data["Drug"]
y[0:5]
```

Out[132]:
```
0    drugY
1    drugC
2    drugC
3    drugX
4    drugY
Name: Drug, dtype: object
```

Configurando el Arbol de Decisión

Estaremos utilizando **entrenar/probar separar** en nuestro **árbol de decisión**. Importemos **train_test_split** de **sklearn.cross_validation**.

In [133]:
```
from sklearn.model_selection import train_test_split
```

Ahora **train_test_split** devolverá 4 parámetros diferentes. Los nombraremos:
X_trainset, X_testset, y_trainset, y_testset

El **train_test_split** necesitará los parámetros:
X, y, test_size=0.3, and random_state=3.

La **X** e **y** son los arreglos necesarios ántes de la operación dividir/separar, **test_size** representa el grado del dataset de pruebas, y el **random_state** asegura que obtendremos las mismas divisiones.

In [126]:
```
X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size
=0.3, random_state=3)
```

Práctica

Imprimir la forma X_trainset e y_trainset. Asegurarse que las dimensiones coincidan

In [ ]:
```
# your code
```

Imprimir la forma de X_testset e y_testset. Asegurarse que las dimensiones coincidan

In [ ]:
```
# tu código
```

Modelando

Primero crearemos una instancia del **DecisionTreeClassifier** llamada **drugTree**.
Dentro del clasificador, especificaremos *criterion="entropy"* para que podamos ver la nueva información de cada nodo.

In [134]:
```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree # muestra los parámetros por omisión
```

Out[134]:
```
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')
```

Luego, adaptaremos los datos con la matriz de entrenamiento **X_trainset** y el vector de respuesta **y_trainset**

In [135]:
```
drugTree.fit(X_trainset,y_trainset)
```

Out[135]:
```
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')
```

Predicción

Ahora hagamos algunas **predicciones** en el dataset de pruebas y guardémoslas en una variable llamada **predTree**.

In [136]:
```
predTree = drugTree.predict(X_testset)
```

Puedes imprimir **predTree** y **y_testset** si quieres comparar visualmente la predicción con los valores actuales.

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

Evaluación

Luego, importemos **metrics** de sklearn y revisemos la precisión de nuestro modelo.

In [137]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
print("Precisión de los Arboles de Decisión: ", metrics.accuracy_score(y_testse
t, predTree))
```

```
Precisión de los Arboles de Decisión:  0.9833333333333333
```

Accuracy classification score calcula la precisión del subconjunto: las etiquetas predichas para una muestra deben coincidir con las correspondientes etiquetas en y_true.

En la clasificación multietiqueta, la función devuelve un subconjunto de precisión. Si el conjunto de etiquetas predichas para una muestra coincide totalmente con el conjunto de etiquetas, entonces la precisión del subconjunto es 1.0; de no ser así, es 0.0.

Práctica

¿Puedes calcular la precisión sin sklearn ?

In [ ]:
```
# tu código aquí
```

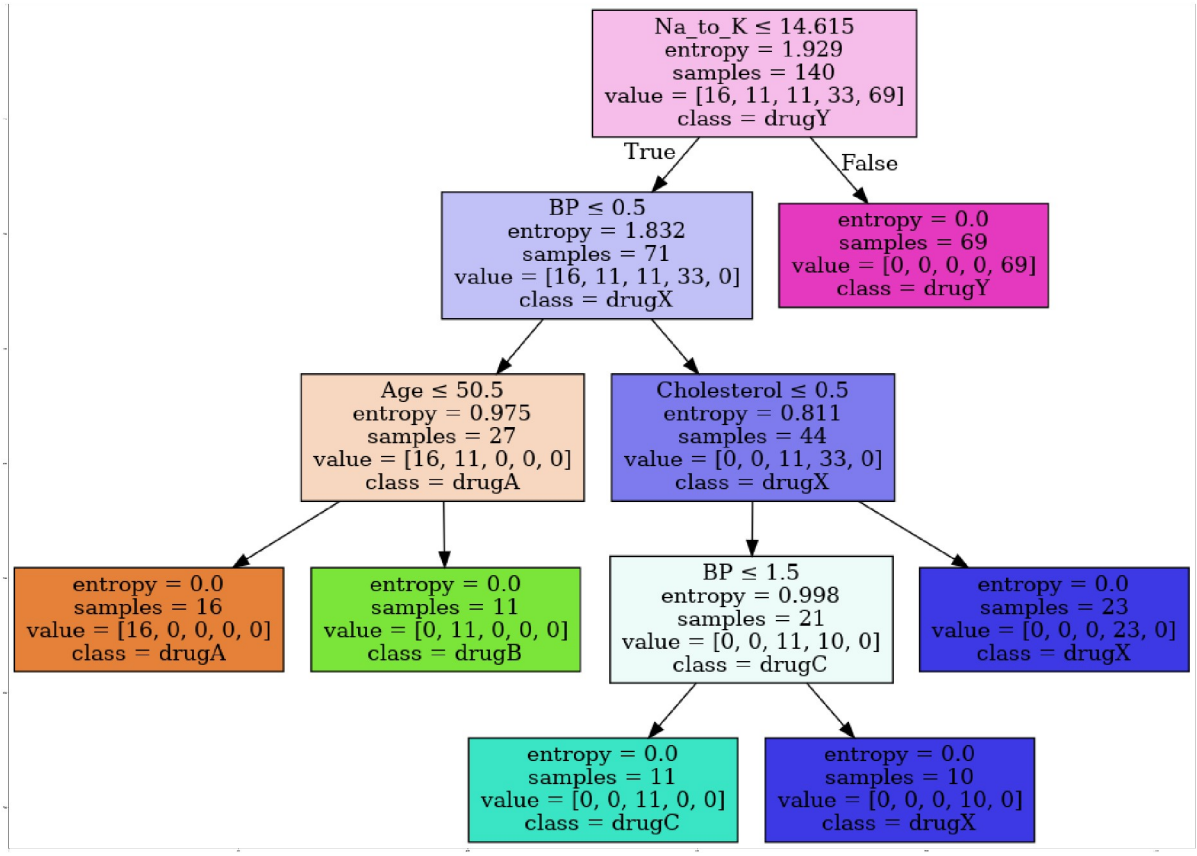
Visualización

Observemos el árbol

In [138]:
```
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

In [139]:
```
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_dat
a, class_names= np.unique(y_trainset), filled=True,  special_characters=True,ro
tate=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[139]: <matplotlib.image.AxesImage at 0x7f63dd9f5828>



¿Deseas aprender más?

IBM SPSS Modeler es una plataforma para analytics que contiene varios algoritmos de machine learning. Fue diseñada para acercar inteligencia predictiva a las decisiones hechas por individuos, grupos, sistemas, toda la empresa. Un free trial está disponible a través de este curso en: SPSS Modeler (http://cocl.us/ML0101EN-SPSSModeler).

Asi mismo, puedes utilizar Watson Studio para ejecutar estos notebooks más rápido y con datasets más grandes. Watson Studio es una solución en la nube lider de IBM's para científicos de datos, construída por científicos de datos. Con Jupyter notebooks, RStudio, Apache Spark y librerías conocidas pre instaladas en la nube, Watson Studio posibilita a los científicos de datos colaborar en sus proyectos sin tener que instalar nada. Sumate a la comunidad de usuarios Watson Studio hoy mismo por medio de una cuenta gratuita en Watson Studio (https://cocl.us/ML0101EN_DSX)

¡Gracias por completar esta lección!

Laboratorio creado por: Saeed Aghabozorgi (https://ca.linkedin.com/in/saeedaghabozorgi)
