

Árboles de decisión

Objetivos

- Desarrollar un modelo de clasificación usando el algoritmo de árboles de decisión

Utilizaremos el algoritmo de clasificación de árboles de decisión para construir un modelo a partir de datos históricos de pacientes y su respuesta a diferentes medicaciones. Luego utilizará el árbol entrenado para predecir la clase de un paciente desconocido, o para encontrar la droga apropiada para un nuevo paciente.

Tabla de contenido

- Acerca del dataset
- Descargando los datos
- Pre-procesamiento
- Estableciendo el árbol de decisión
- Modelando
- Predicción
- Evaluación
- Visualización

Importamos las siguientes librerías:

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

```
In [1]: import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

Acerca del dataset

Imagine que es un investigador médico que reúne datos para un estudio. Ha coleccionado datos acerca de un conjunto de pacientes; todos ellos sufrieron la misma enfermedad. Durante el curso de su tratamiento, cada paciente respondió a 1 de 5 medicamentos: Drug A, Drug B, Drug c, Drug x e y. Parte de su trabajo es construir un modelo para encontrar qué droga es apropiada para un futuro paciente que tenga la misma enfermedad. El conjunto de características es Edad, Sexo, presión sanguínea y colesterol, y el objetivo es la droga a la que responde cada paciente. Es un ejemplo de clasificador multiclase. Se usa la parte de entrenamiento del dataset para construir un árbol de decisión y luego éste se utiliza para predecir la clase de un paciente desconocido o para prescribirle la droga apropiada a un nuevo paciente.

Descargando los datos

```
In [2]: !wget -O drug200.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDvelopement/01/NB_Samples/data/drug200.csv
```

```
Out[2]: ('drug200.csv', <http.client.HTTPMessage at 0x1db7109cd30>)
```

Los leemos en un dataframe Pandas:

```
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

```
In [5]: # tamaño de los datos
my_data.shape
```

```
Out[5]: (200, 6)
```

Pre-procesamiento

Utilizando **my_data** declaramos las siguientes variables:

- X** como la **matriz de características**
- y** como el **vector respuesta (objetivo)**

Removemos la columna que contiene el nombre objetivo ya que no contiene valores numéricos.

```
In [6]: X = my_data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[0:5]
```

```
Out[6]: 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)
```

Algunas de las características en el dataset son categóricas, como **sex** o **BP**. Desafortunadamente, los árboles de decisión de Sklearn no trabajan con variables categóricas, así que debemos convertirlas a valores numéricos. **pandas.get_dummies()** convierte variables categóricas en variables dummy/indicador.

```
In [7]: 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[7]: 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)
```

Variable objetivo

```
In [8]: y = my_data["Drug"]
y[0:5]
```

```
Out[8]:
```

0	drugY
1	drugC
2	drugC
3	drugX
4	drugY

Name: Drug, dtype: object

Estableciendo el árbol de decisión

Utilizaremos **train/test split**

```
In [9]: from sklearn.model_selection import train_test_split
```

Ahora, **train_test_split** devolverá 4 parámetros diferentes. Los llamaremos:

X_trainset, X_testset, y_trainset, y_testset

train_test_split necesita los parámetros:

X, y, test_size y **random_state**.

X e **Y** son los arreglos antes de la división. **test_size** representa el cociente del dataset de testing.

random_state asegura que obtengamos las mismas divisiones.

```
In [10]: X_trainset, X_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, random_state=1)
```

Dimensiones de **X_trainset** e **y_trainset**. Deben coincidir.

```
In [11]: print('Shape of X training set {}'.format(X_trainset.shape), '&', 'Size of Y training set {}'.format(y_trainset.shape))
Shape of X training set (140, 5) & Size of Y training set (140,)
```

Dimensiones de **X_testset** e **y_testset**. Deben coincidir.

```
In [12]: print('Shape of X training set {}'.format(X_testset.shape), '&', 'Size of Y training set {}'.format(y_testset.shape))
Shape of X training set (60, 5) & Size of Y training set (60,)
```

Modelado

Primero creamos una instancia de **DecisionTreeClassifier** llamada **drugTree**.

Dentro del clasificador, especificamos **criterion="entropy"** para poder ver la ganancia de información en cada nodo.

```
In [13]: drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
drugTree # muestra los parámetros por defecto
```

```
Out[13]: DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Ajustamos los datos usando la matriz de características de entrenamiento **X_trainset** y el vector de respuestas de entrenamiento **y_trainset**.

```
In [14]: drugTree.fit(X_trainset,y_trainset)
```

```
Out[14]: DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Predicción

Hagamos algunas predicciones en el dataset de testing y almacenémoslas en una variable llamada **predTree**.

```
In [15]: predTree = drugTree.predict(X_testset)
```

Imprimimos **predTree** e **y_testset** para comparar visualmente.

```
In [16]: 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
```

Evaluación

Importemos metrics de sklearn y chequeemos la precisión del modelo.

```
In [17]: 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 computa la precisión del subconjunto: el conjunto de etiquetas predichas para una muestra debe coincidir exactamente con el conjunto de etiquetas correspondiente en **y_true**.

En la clasificación multietiquetas, la función devuelve la precisión del subconjunto. Si todo el conjunto de etiquetas predichas para una muestra coincide estrictamente con el conjunto verdadero de etiquetas, entonces la precisión del subconjunto es 1.0; de lo contrario, es 0.0.

Visualización

```
In [20]: # Notice: You might need to uncomment and install the pydotplus and graphviz libraries
!conda install -c conda-forge pydotplus -y
!conda install -c conda-forge python-graphviz -y
```

Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

Package Plan

environment location: C:\Users\marco\Anaconda3

added / updated specs:
- pydotplus

The following packages will be downloaded:

package	build		
conda-4.9.2	py38haa244fe_0	3.1 MB	conda-forge
graphviz-2.38.0	h6538335_10l1	41.0 MB	conda-forge
pydotplus-2.0.2	py_2	23 KB	conda-forge
python_abi-3.8	1_cp38	4 KB	conda-forge
Total:		44.1 MB	

The following NEW packages will be INSTALLED:

graphviz	conda-forge/win-64:graphviz-2.38.0-h6538335_10l1
pydotplus	conda-forge/noarch:pydotplus-2.0.2-py_2
python_abi	conda-forge/win-64:python_abi-3.8-1_cp38

The following packages will be SUPERSEDED by a higher-priority channel:

conda	pkgs/main::conda-4.9.2-py38haa95532_0 --> conda-forge::conda-4.9.2-py38haa244fe_0
-------	---

Downloading and Extracting Packages

pydotplus-2.0.2	23 KB	#####9	69%
pydotplus-2.0.2	23 KB	#####	100%
python_abi-3.8	4 KB	#####	100%
python_abi-3.8	4 KB	#####	100%

graphviz-2.38.0	41.0 MB		0%
graphviz-2.38.0	41.0 MB		1%
graphviz-2.38.0	41.0 MB	2	2%
graphviz-2.38.0	41.0 MB	4	4%
graphviz-2.38.0	41.0 MB	6	6%
graphviz-2.38.0	41.0 MB	7	8%
graphviz-2.38.0	41.0 MB	9	12%
graphviz-2.38.0	41.0 MB	#1	10%
graphviz-2.38.0	41.0 MB	#3	14%
graphviz-2.38.0	41.0 MB	#5	15%
graphviz-2.38.0	41.0 MB	#7	17%
graphviz-2.38.0	41.0 MB	#8	19%
graphviz-2.38.0	41.0 MB	#####	21%
graphviz-2.38.0	41.0 MB	#2	23%
graphviz-2.38.0	41.0 MB	#4	25%
graphviz-2.38.0	41.0 MB	#6	27%
graphviz-2.38.0	41.0 MB	#8	31%
graphviz-2.38.0	41.0 MB	#	32%
graphviz-2.38.0	41.0 MB	#4	34%
graphviz-2.38.0	41.0 MB	#6	36%
graphviz-2.38.0	41.0 MB	#8	38%
graphviz-2.38.0	41.0 MB	#####	40%
graphviz-2.38.0	41.0 MB	#####2	42%
graphviz-2.38.0	41.0 MB	#####4	44%
graphviz-2.38.0	41.0 MB	#####6	46%
graphviz-2.38.0	41.0 MB	#####8	48%
graphviz-2.38.0	41.0 MB	#####	50%
graphviz-2.38.0	41.0 MB	#####2	52%
graphviz-2.38.0	41.0 MB	#####4	54%
graphviz-2.38.0	41.0 MB	#####6	56%
graphviz-2.38.0	41.0 MB	#####8	58%
graphviz-2.38.0	41.0 MB	#####	60%
graphviz-2.38.0	41.0 MB	#####2	62%
graphviz-2.38.0	41.0 MB	#####4	64%
graphviz-2.38.0	41.0 MB	#####6	66%
graphviz-2.38.0	41.0 MB	#####8	68%
graphviz-2.38.0	41.0 MB	#####9	70%
graphviz-2.38.0	41.0 MB	#####1	72%
graphviz-2.38.0	41.0 MB	#####3	74%
graphviz-2.38.0	41.0 MB	#####5	76%
graphviz-2.38.0	41.0 MB	#####7	78%
graphviz-2.38.0	41.0 MB	#####9	80%
graphviz-2.38.0	41.0 MB	#####1	82%
graphviz-2.38.0	41.0 MB	#####3	84%
graphviz-2.38.0	41.0 MB	#####5	85%
graphviz-2.38.0	41.0 MB	#####7	87%
graphviz-2.38.0	41.0 MB	#####9	89%
graphviz-2.38.0	41.0 MB	#####1	91%
graphviz-2.38.0	41.0 MB	#####3	93%
graphviz-2.38.0	41.0 MB	#####5	95%
graphviz-2.38.0	41.0 MB	#####7	97%
graphviz-2.38.0	41.0 MB	#####9	99%
graphviz-2.38.0	41.0 MB	#####	100%

conda-4.9.2	3.1 MB		0%
conda-4.9.2	3.1 MB	#3	14%
conda-4.9.2	3.1 MB	#####7	38%
conda-4.9.2	3.1 MB	#####4	64%
conda-4.9.2	3.1 MB	#####9	90%
conda-4.9.2	3.1 MB	#####	100%

Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

Package Plan

environment location: C:\Users\marco\Anaconda3

added / updated specs:
- python-graphviz

The following packages will be downloaded:

package	build		
python-graphviz-0.16	pyhd3deb0d_0	20 KB	conda-forge
Total:		20 KB	

The following NEW packages will be INSTALLED:

python-graphviz	conda-forge/noarch:python-graphviz-0.16-pyhd3deb0d_0
-----------------	--

Downloading and Extracting Packages

python-graphviz-0.16	20 KB		0%
python-graphviz-0.16	20 KB	#####9	80%
python-graphviz-0.16	20 KB	#####	100%

Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done

```
In [21]: from io import StringIO
import pydotplus
from matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

```
In [22]: 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=tree.class_name,
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[22]: <matplotlib.image.AxesImage at 0x1db72fec220>
```

