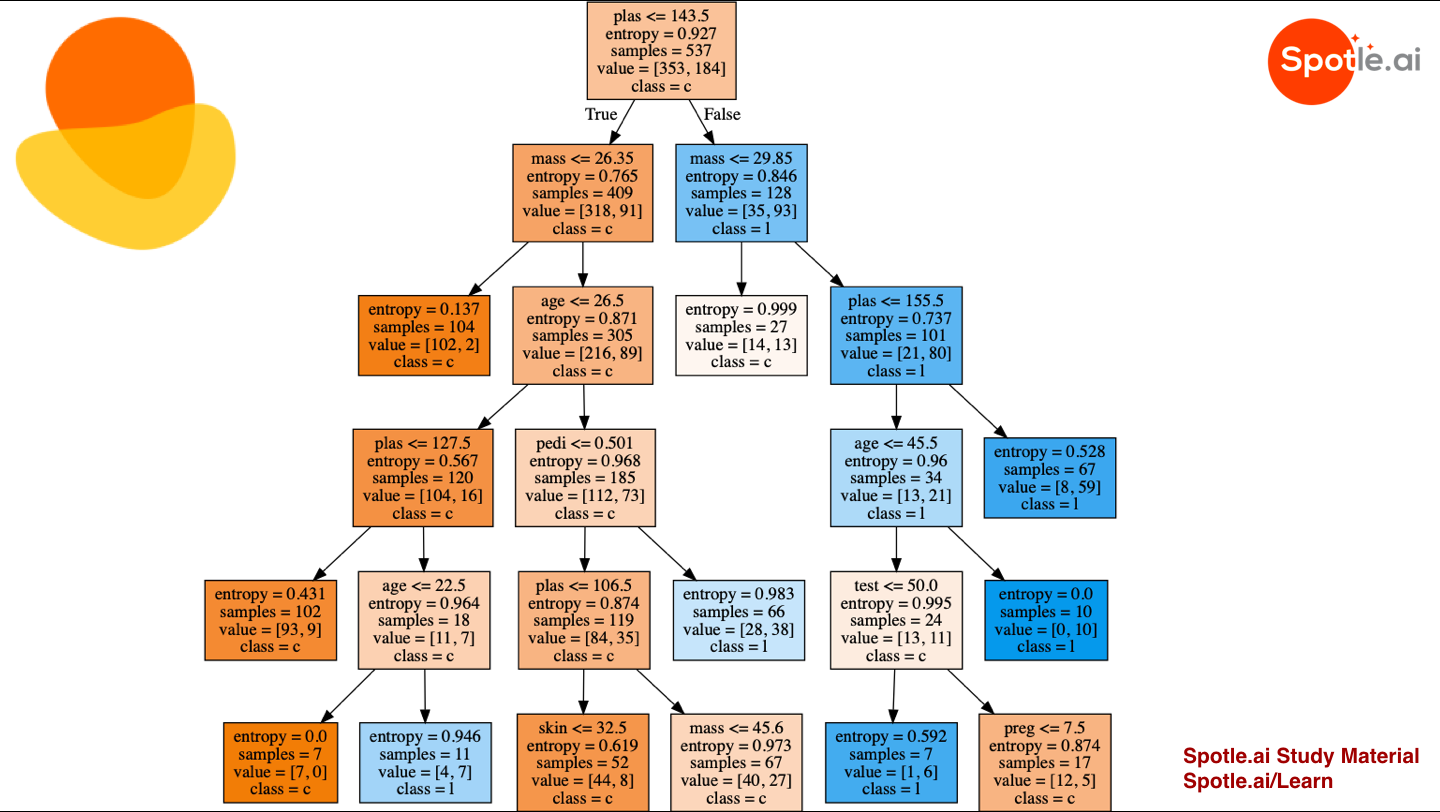
**Decision Tree with Python**



* **615ACTIVITIES**
* **20SPOTCOINS**

Sep 10, 2019

**The objective of this study is to devise a decision tree based model, that can predict whether a subject has diabetes or not, given the values of certain diagnostic measurements.**

**Decision Tree with Python**

The following is the dataset we will be using to learn decision tree hands-on in python.

**Pima-Indians-Diabetes**

preg plas pres skin test mass pedi age class

0 6 148 72 35 0 33.6 0.627 50 1

1 1 85 66 29 0 26.6 0.351 31 0

2 8 183 64 0 0 23.3 0.672 32 1

3 1 89 66 23 94 28.1 0.167 21 0

4 0 137 40 35 168 43.1 2.288 33 1

5 5 116 74 0 0 25.6 0.201 30 0

6 3 78 50 32 88 31.0 0.248 26 1

7 10 115 0 0 0 35.3 0.134 29 0

8 2 197 70 45 543 30.5 0.158 53 1

9 8 125 96 0 0 0.0 0.232 54 1

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. All patients here are females at least 21 years old from Pima Indian heritage. Please download the dataset from the link: <https://www.kaggle.com/uciml/pima-indians-diabetes-database>

The dataset consists of 9 columns and 768 observations. Columns are namely preg, plas, pres, skin, test, mass, pedi, age and class. First 8 columns are input variables or independent variables or features. The last column, that is the class variable is a dependent variable, which is 1 if a subject has diabetes, 0 otherwise.

1. Number of times a subject was pregnant,
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test,
3. Diastolic blood pressure (mm Hg),
4. Triceps skin fold thickness (mm),
5. 2-Hour serum insulin (mu U/ml),
6. Body mass index (weight in kg/(height in m)^2),
7. Diabetes pedigree function,
8. Age (years),
9. Class variable (0 or 1)

The dataset contains information for 500 non diabetic subjects and 268 diabetic patients.

**Let's restate the objective. Our objective is to build a decision tree based model, that will predict the class variable (that means whether a subject has Diabetes or not) of the subject based on the observation in 8 features.**

We will be using sklearn.tree.DecisionTreeClassifier from scikit-learn for this model.

In [1]:

from sklearn.tree import DecisionTreeClassifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

Some advantages of decision trees are:

* Simple to understand and to interpret. Trees can be visualised.
* Requires little data preparation. Other techniques often require data normalisation, creation of dummy variables and removal of blank values. Note that, this module does not support missing values.
* The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
* Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
* Able to handle multi-output problems.
* Uses a white box model. If a given situation is observed in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
* Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
* Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

We are now ready to build and train the model. Let us first load the dataset. Variable X contains 8 features and variable Y contains class information, as explained earlier.

In [2]:

from pandas import read\_csv

filename = '/common/AI & ML/data/pima-indians-diabetes.csv'

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = read\_csv(filename, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

The following section will split the dataset randomly into two groups, training dataset and test dataset. We will use 70% data as training data and remaining 30% as test data.

In [3]:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=1234)

Let's create a decision tree classifier.

In [4]:

model = DecisionTreeClassifier()

**Description of parameters**

* criterion : string, optional (default=”gini”). The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
* splitter : string, optional (default=”best”). The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.
* max\_depth : int or None, optional (default=None). The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
* min\_samples\_split : int, float, optional (default=2). The minimum number of samples required to split an internal node:
* If int, then consider min\_samples\_split as the minimum number.
* If float, then min\_samples\_split is a fraction and ceil(min\_samples\_split \* n\_samples) are the minimum number of samples for each split.
* min\_samples\_leaf : int, float, optional (default=1). The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
* If int, then consider min\_samples\_leaf as the minimum number.
* If float, then min\_samples\_leaf is a fraction and ceil(min\_samples\_leaf \* n\_samples) are the minimum number of samples for each node.
* min\_weight\_fraction\_leaf : float, optional (default=0.). The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample\_weight is not provided.
* max\_features : int, float, string or None, optional (default=None). The number of features to consider when looking for the best split:
* If int, then consider max\_features features at each split.
* If float, then max\_features is a fraction and int(max\_features \* n\_features) features are considered at each split.
* If “auto”, then max\_features=sqrt(n\_features).
* If “sqrt”, then max\_features=sqrt(n\_features).
* If “log2”, then max\_features=log2(n\_features).
* If None, then max\_features=n\_features.
* random\_state : int, RandomState instance or None, optional (default=None). If int, random\_state is the seed used by the random number generator; If RandomState instance, random\_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.
* max\_leaf\_nodes : int or None, optional (default=None). Grow a tree with max\_leaf\_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
* min\_impurity\_decrease : float, optional (default=0.). A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
* min\_impurity\_split : float, (default=1e-7). Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf. Use min\_impurity\_decrease instead.
* class\_weight : dict, list of dicts, “balanced” or None, default=None. Weights associated with classes in the form {class\_label: weight}.
* presort : bool, optional (default=False). Whether to presort the data to speed up the finding of best splits in fitting. For the default settings of a decision tree on large datasets, setting this to true may slow down the training process. When using either a smaller dataset or a restricted depth, this may speed up the training.

Now, let's build the model with training data.

In [5]:

model.fit(X\_train, Y\_train)

Out[5]:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

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')

**We have now trained our model. Next, we need to test our model with the test data. The following section will test how efficient (or accurate) our decision tree based model is.**

OK. What are we going to test? We are going to test that for a given subject in the test data whether the predicted value of class variable (that is, whether a subject has diabetes or not) matches with the actual value of the class variable in the test data. The following piece of code shows how the class variable is predicted for a given set of test data.

In [6]:

import pandas as pd

import numpy as np

import random as rnd

rnd.seed(123458)

X\_new = X[rnd.randrange(X.shape[0])]

X\_new = X\_new.reshape(1,8)

YHat = model.predict(X\_new)

df = pd.DataFrame(X\_new, columns = names[:-1])

df["predicted"] = YHat

df.head(1)

Out[6]:

pregplaspresskintestmasspediagepredicted08.0196.076.029.0280.037.50.60557.01.0

The above table shows, the person is likely to be a diabetes patient.

The next step is to calculate the accuracy of the prediction. And we will do this using accuracy\_score function.

In [7]:

from sklearn.metrics import accuracy\_score

YHat = model.predict(X\_test)

# calculate accuracy

print (round(accuracy\_score(Y\_test, YHat)\*100,2))

67.97

Now we will try a different model. Let us change classification criterion to entropy and assign 20 to max\_leaf\_nodes.

**Note:** Entropy is the measures of impurity, disorder or uncertainty in a bunch of examples. Entropy controls how a Decision Tree decides to split the data. It actually effects how a Decision Tree draws its boundaries.

In [8]:

model = DecisionTreeClassifier(criterion='entropy', max\_leaf\_nodes = 20)

dt = model.fit(X\_train, Y\_train)

YHat = model.predict(X\_test)

print (round(accuracy\_score(Y\_test, YHat)\*100,2))

72.73

We can see that, accuracy of our model is increased by approximately 5%. So for a new subject, we can predict her diabetes status with 72.73 percent accuracy, which is the accuracy of our model.

**The following section will show you how to visualize a decision tree.**

**The decision tree.**

To draw the decision tree we need to install following Python libraries pydotplus and graphviz. You can install these libraries using following commands: -

* (sudo) pip3 install pydotplus
* (sudo) pip3 insatll graphviz

Additionally you have to install graphviz executable.

Windows user can download from the following site, <https://graphviz.gitlab.io/_pages/Download/Download_windows.html>

Mac users can install using following commands,

* brew install graphviz

Ubuntu users can install using apt-get

* (sudo) apt-get install graphviz

To know more about graphviz, visit <https://www.graphviz.org/>

In [10]:

from IPython.display import Image

from sklearn import tree

import matplotlib.image as mpimg

import pydotplus

import io

import matplotlib.pyplot as plt

import sys

str1 = "/usr/local/Cellar/graphviz/2.40.1\_1/bin/"

sys.path.append(str1)

dot\_data = io.StringIO()

tree.export\_graphviz(dt, out\_file=dot\_data, filled=True, feature\_names = names[:-1], class\_names = names[-1])

# Draw graph

pydotplus.graph\_from\_dot\_data(dot\_data.getvalue()).write\_png('dt.png')

plt.figure(figsize=(20, 80))

img = mpimg.imread('dt.png')

imgplot = plt.imshow(img)

plt.show()