Build your own prediction model

Please open the Read_me before this execution. See to with that all the installations are perfect. Please see to with that you've understood all the concepts in the Chapter-01 thoroughly, so that execution becomes easier.

At the end of this tutorial, you'll be able to build your own prediction model, by using Decision Trees. This model takes certain parameters(features) of a student and based on these features, it predicts whether the student passes or fails in a particular test. I strongly recommend you to download and view the data set to understand the data parameters (parameters in Machine Learning language are called features)
https://archive.ics.uci.edu/ml/datasets/student+performance – To download the data set

USP: From the above data, we are going to predict whether a student is going to pass or fail.

Step-0:

Download the data-set from the above-mentioned URL. If its in zip format, unzip it otherwise, you'll see error while loading the data set. It has 3 scores of 3 tests G1-1st test, G2-2nd test, G3-3rd test. The problem is simplified to compute just pass or fail. This can be done by checking if G1+G2+G3 >= 35. This brings us to a 50% split of students

Step-1: Loading the dataset into the algorithm

This is done by using **pandas.read_csv('name')** function offered by pandas library. The reason "sep=';'" is used as the usual reference of separation for csv files is , but in our dataset ; is used.

```
# In[1]:

# load dataset (student Portuguese scores)

import pandas as pd

d=pd.read_csv('file-location\student-por.csv', sep=';')

#Copy+Paste the address of the  student-por.csv file

len(d)
```

Step-2:

Add columns for pass or fail indication (1-pass ; 0-fail). Now compute G1+G2+G3.
apply function of pandas is used to apply the lambda; axis=1 for row-wise operation; axis=0 for coloumn-wise operation. The other variables need to be dropped.

```
# In[2]:
# generate binary label (pass/fail) based on G1+G2+G3 (test grades, each 0-20 pts); threshold for passing is sum>=30
d['pass'] = d.apply(lambda row: 1 if (row['G1']+row['G2']+row['G3']) >= 35 else 0, axis=1)
d = d.drop(['G1', 'G2', 'G3'], axis=1)
d.head()
```

Step-3:

If you observe the output in Step-2, some colomouns are words/phrases/strings. These need to be converted to numbers. This is done by *get_dummies()* which is a Pandas feature, and we need to mention which columns are the ones that we want to turn into numeric form.

```
# In[3]:

# use one-hot encoding on categorical columns
d = pd.get_dummies(d, columns=['sex', 'school', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',

                    'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',

                    'nursery', 'higher', 'internet', 'romantic'])
d.head()
```

Note: Refer “one-hot encoding method” for detailed explanation.

Step-4:

This is one of the most important steps. Remember that the better the training, the better is the accuracy of prediction. Now how can we train the model better? By giving more data for the training set. In order to do this, we can see that 500 rows of data is fed to the training set and only 149 rows to the test set.

Here we need to shuffle the rows and produce a training set with first 500 rows and rest 149 rows for test set and then we just need to get attributes form the training set which means we will get rid of the pass column and save the pass column separately. The same is repeated for the testing set. We will apply the attributes to the entire dataset and save the pass column separately for the entire dataset. Now we will find how many passed and failed from the entire dataset. This can be done by
computing the percentage number of passed and failed which will give us a result of 328
out of 649.

```
# shuffle rows
d = d.sample(frac=1)
# split training and testing data
d_train = d[:500]
d_test = d[500:]
d_train_att = d_train.drop(['pass'], axis=1)
d_train_pass = d_train['pass']


d_test_att = d_test.drop(['pass'], axis=1)

d_test_pass = d_test['pass']



d_att = d.drop(['pass'], axis=1)

d_pass = d['pass']



# number of passing students in whole dataset:

import numpy as np

print("Passing: %d out of %d (%.2f%%)" % (np.sum(d_pass), len(d_pass), 100*float(np.sum(d_pass)) / len(d_pass)))
```

Step-5: Building the decision tree
This is done by fitting a decision tree to the model. Scikit-learn package offers DecisionTreeClassifier() which is capable of performing multi-class classification on a data set. Max_depth=5 is only a trial value as an initial tree depth to get a feel of how the model is trained for a depth=5.

```
# In[5]:
# fit a decision tree

from sklearn import tree
t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=5)
t = t.fit(d_train_att, d_train_pass)
```

Here we will use the entropy or information gain metric to decide when to split.

Step-6: Visualizing the data
To get an overview of our dataset, we need to create a visual representation of the tree. This can be achieved by using one more function of the scikit-learn package: export_graphviz.
Interpretation of the visual data: In this case if failure is greater than or equal to 0.5, that means it is true and it placed on left-hand side of the tree. Consider tree is always true on left side and false on right side, which means there are no prior failures. In the representation we can see left side of the tree is mostly in blue which means it is predicting a pass even though there are few questions as compared to the failure maximum of 5 questions. The tree is o n right side if failure is less than 0.5, this makes the student fail, which means the first question is false. Prediction is failure if in orange color but as it proceeds further to more questions since we have used i.e., depth=5.

```
# visualize tree

        import graphviz

        dot_data = tree.export_graphviz(t, out_file=None, label="all", impurity=False, proportion=True,

                                feature_names=list(d_train_att), class_names=["fail", "pass"],

                                filled=True, rounded=True)

        graph = graphviz.Source(dot_data)

        graph
```

Step-7: The following code block shows a method to export the visual representation which by clicking on Export and save to PDF or any format if you want to visualize later:

```
# In[7]:


# save tree

tree.export_graphviz(t, out_file="student-performance.dot", label="all", impurity=False, proportion=True,

            feature_names=list(d_train_att), class_names=["fail", "pass"],

            filled=True, rounded=True)
```

Step-8: Next we check the score of the tree using the testing set that we created earlier:

```
# In[8]:

t.score(d_test_att, d_test_pass)
```

The result we had was approximately 60%. Now let's cross verify the result to be assured
that the dataset is trained perfectly:

Step-9:

Performing cross-validation on the entire dataset which will split the data on a of 20/80 basis, where 20% is the on testing set and 80% is on the training set. The average result is 67%. This shows that we have a well-balanced dataset. Here we have various choices to make regarding.

```
# In[9]:

from sklearn.model_selection import cross_val_score
scores = cross_val_score(t, d_att, d_pass, cv=5)

# show average score and +/- two standard deviations away (covering 95% of scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

Step-10:

As mentioned above, to test a range of values for the max_depth value, we trial and error with a range from 1-20. Please make a note of all the observations.

```
# In[10]:

for max_depth in range(1, 20):

   t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=max_depth)

   scores = cross_val_score(t, d_att, d_pass, cv=5)

   print("Max depth: %d, Accuracy: %0.2f (+/- %0.2f)" % (max_depth, scores.mean(), scores.std() * 2))
```

Step-11:

We use various max_depth values from 1 to 20 as mentioned above. Considering we make a tree with one question or with 20 questions having depth value of 20 which will give us questions more than 20 which is you will have to go 20 steps down to reach a leaf node. Here we again perform cross- validation and save and print our answer. This will give different accuracy and calculations. On analyzing it was found that on have depth of 2 and 3 the accuracy is the best which was compared accuracy from the average we found earlier. The result obtained in this step is the required data to plot the final graph. This is the penultimate step of the prediction.

After executing step 11, you observe that:

The error bars shown in the following screenshot are the standard deviations in the score,
which concludes that a depth of 2 or 3 is ideal for this dataset, and that our assumption of 5
was incorrect:

```
# In[11]:

depth_acc = np.empty((19,3), float)

i = 0

for max_depth in range(1, 20):

   t = tree.DecisionTreeClassifier(criterion="entropy", max_depth=max_depth)

   scores = cross_val_score(t, d_att, d_pass, cv=5)

   depth_acc[i,0] = max_depth

   depth_acc[i,1] = scores.mean()

   depth_acc[i,2] = scores.std() * 2

   i += 1


depth_acc
```

Step-12:

More depth doesn't give any more power, and just having one question, which would be *did you fail previously?*, isn't going to provide you with the same amount of information as two or three questions would. Our model shows that having more depth does not necessarily help, nor does having a single question of *did you fail previously?* provide us with the same amount of information as two or three questions would give us.

```
# In[12]:

import matplotlib.pyplot as plt

fig, ax = plt.subplots()

ax.errorbar(depth_acc[:,0], depth_acc[:,1], yerr=depth_acc[:,2])

plt.show()
```