Classification using a Decision Tree

This notebook demonstrates generating a practice dataset, visualising it, and using a decision tree to classify it. It is important that you generate a unique dataset so that you have different results to your peers so that you can undertake a unique analysis. To achieve this, where a random number seed is specified you need to replace the *0* with the *last 3 digits of your student number*.

You are required to document your work in *markdown cells*. Empty cells have been included, but you can add more if you want for either code experimentation or futher explanation. Concise documentation for *markdown* can be found at https://sourceforge.net/p/jupiter/wiki/markdown_syntax/#md_ex_pre and https://github.com/adam-p/markdown-here/wiki/Markdown-Here-Cheatsheet

Original notebook by Dr Kevan Buckley, University of Wolverhampton, 2019. This submission by ***your name and student number***

In markdown cells like this one explain the code or results below

Name:Gaurav Raut ID:np03a180058 group:L6GC6

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pydotplus
from sklearn import tree
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Numpy is an universally useful cluster preparing bundle.It gives an elite multidimensional array objects, and tools for working with these arrays. It is the major bundle for logical registering with Python.Otherthan its undeniable logical uses, Numpy can likewise be utilized as a proficient multi-dimensional holder of nonexclusive information.

Pandas is an opensource library that permits one to perform information control in Python. Pandas library is based over Numpy, which means Pandas needs Numpy to work. Pandas give a simple method to make, control and wrangle the information.

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures.

PyDotPlus is an improved version of the old pydot project that provides a Python Interface to Graphviz's Dot language.

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests,and k-neighbours, and it also supports Python

numerical and scientific libraries like NumPy and SciPy.

In [2]:
```
import os
os.environ["PATH"] += os.pathsep + 'C:/Users/user/miniconda3/Library/bin/graphviz'
```

Make sure that you replace the zero with the last 3 digits of your student number. If this includes a leading zero use the last 4 digits.

In [3]:
```
features, target = make_classification(
    n_samples=200, n_features=4, n_classes=3, n_clusters_per_class=1, random_state=0
```

make_classification() is a inbuilt scikit-learn function which is used to randomly generate 2D classification datasets. This allows us to have multiple features, add noise and imbalance to out data. Some of its other features includes, adding Redundant features(linear combination of existing features), non-informative features to check overfitting with the randomly generated features that are of no use and adding directly repeated features as well. We also can create multiple clusters of our classes to increase the complexity of the classification problem and to force complex non-linear boundary for classifiers, we can decrease the separation between classes. It consists of following parameters: -n_samples --> total number of samples -n_features --> total number of features -n_classes --> total number of labels required in a classification problem -n_clusters_per_class --> total number of clusters per class -random_state --> it determines random number generation while creating datasets. This functions returns X: Array of shape [n_samples, n_features] and y: Array of shape [n_samples] In the above cell, For make_classification(), 200 samples containing multi-class classification datasets with 3 classes are generated, with 4 informative features and 1 clusters per class. We obtain features and target from this function.

In [4]:
```
features.shape
```

Out[4]:
```
(200, 4)
```

-features.shape is used to obtain the dimesion of the features array. -It returns the shape in tuple as we can see in the above output. -We obtained a tuple (200, 4), which means there are 4 features for 200 samples as we have defined number of features equal to 4 per class and total of 200 examples per each samples. -This also means that there are 200 rows of data and corresponding 4 columns of features.

In [5]:
```
target.shape
```

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

-The target.shape provides the shape of the target values. -As seen above, we obtained a tuple of (200,), which means there are 200 rows but no columns. -There are total of 200 rows which means that for each training samples, we have one expected target output.

In [6]:
```
features[0]
```

Out[6]:
```
array([ 0.70451998, -0.12039885, -0.27340318,  0.28677273])
```

In [ ]:
```
-This gives the array of features of the first data/sample (first row) in the traini
-As there are 4 features to each training samples, we can see the array containing 4
data.
```

In [7]:
```
target[0]
```

Out[7]: 2

-This gives the target label/values for the first training sample. -As there is output 0 in the above cell, this means that the first training sample belongs to the 3rd class. [Note: Assuming that the class label starts at 0 and there are total of 3 classes.]

In [8]:
```
feature_names = ['feature_0', 'feature_1', 'feature_2', 'feature_3']
feature_names
```

Out[8]: ['feature_0', 'feature_1', 'feature_2', 'feature_3']

-In the above cell, the features name were added to the variable feature_name in the form of list. -As there are 4 features- feature_0, feature_1, feature_2, and feature_3 name were given to each features. -As a result, we obtain a list of name of all the features of training samples.

In [ ]:
```

```

In [9]:
```
features_df = pd.DataFrame(features, columns=feature_names)
```

-pd.DataFrame is a two dimensional data structure which helps to align data in a tabular fashion in rows and columns. -The three principle components it consists of are the data, rows, and columns. -It consists of following parameters: -data (ndarray, iterable, dict or DataFrame) -index (Index to use for resulting frame) -columns (column labels to use for resulting frame) -dtype (data type to force) -copy (to copy data from inputs) In the above cell, we have used pd.DataFrame to align our features in the tablur form where each row consists of the sample data and its 4 features. In the parameters, we added a features and then the columns as feature_names that we have created before.

In [10]:
```
features_df.head()
```

Out[10]:

	feature_0	feature_1	feature_2	feature_3
0	0.704520	-0.120399	-0.273403	0.286773
1	1.353608	1.673007	-1.124737	-1.260748
2	-2.483329	0.863287	0.825550	-1.428388
3	1.526224	0.835742	-0.937457	-0.421999
4	0.560243	-0.643432	-0.045013	0.749102

-This returns the first n rows of the data form the previously created DataFrame. -It takes n:int as a parameter (default = 5), to identify how many rows to selects and display. -In the above cell, we have displayed first five rows of data with features_df.head().

In [11]:
```
target_df = pd.DataFrame(target, columns=['target'])
```

-This sets our labels into DataFrame which alings the target matrix into tabular form. -The column name was set as "target". -There is only one column in the target dataframe.

In [12]:
```
target_df.head()
```

Out[12]:

	target
0	2
1	0
2	0
3	1
4	2

In the above cell, we have displayed first five rows of the target dataframe which shows labels(targets) for the each samples with target_df.head().

In [13]:
```
dataset = pd.concat([features_df, target_df], axis=1)
```

-pd.concat is a panda's function to easily combine Series, DataFrame and Panel objects. -This concat function helps to perform concatenation operations along an axis. -It consists of the following parameters: -objs: this is a sequence or mapping of DataFrame, Series, or Panel objects. -axis: this is the axis to concatenate along -join: {'inner', 'outer'}: inner join for intersection and outer join for union -ignore_index: True: does not uses the index value on the concatenation axis -join_axes: list of index objects -In the above cell, pd.concat function is used to create a dataset by combining each features of the sample data with their corresponding target values. -This will create a single DataFrame from two dataframes of features and labels.

In [14]:
```
dataset.head()
```

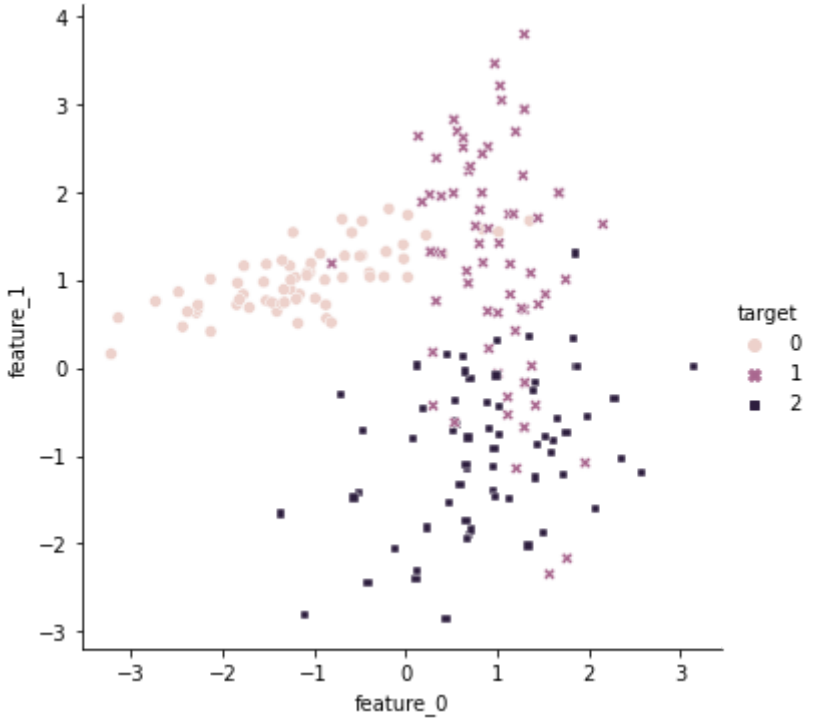
Out[14]:

	feature_0	feature_1	feature_2	feature_3	target
0	0.704520	-0.120399	-0.273403	0.286773	2
1	1.353608	1.673007	-1.124737	-1.260748	0
2	-2.483329	0.863287	0.825550	-1.428388	0
3	1.526224	0.835742	-0.937457	-0.421999	1
4	0.560243	-0.643432	-0.045013	0.749102	2

In the above cell, we have displayed first five rows of the combined dataframe which was obtained by using the concat function which previously was used to combine features and its target values.

The above dataframe shows the 4 features columns for first 5 sample data and target column for the target output of each corresponding samples.

In [15]:
```
sns.relplot(
    x='feature_0', y='feature_1', hue='target', style='target', data=dataset)
plt.show()
```



In the above cell, the sns.relplot function was used to shows the joint distribution of two variables i.e. two features- feature_0 and feature_1 where each point represent an observation in the dataset and plotting points according to 3rd varibale "target". Here, this is reffered to as "hue semantic" as the color of the points gains meaning. Also, different marker style such as circle, cross and square was used to emphasize the difference between the classes and to improve the accessibility. This allows us to indentify whether there is any meaningful relationship between them or not. In the above figure, hue semantic is categorical with 3 classes. Here, it is noticed that among the features, feature_0 and feature_1, it looks like the blue class (which is the third class i.e. 2) of targets is seperable from the other two.

The association between feature_0 and feature_1: For class 0, -There is a moderately strong, positive, linear association. There don't appear to be any outliers in the data.

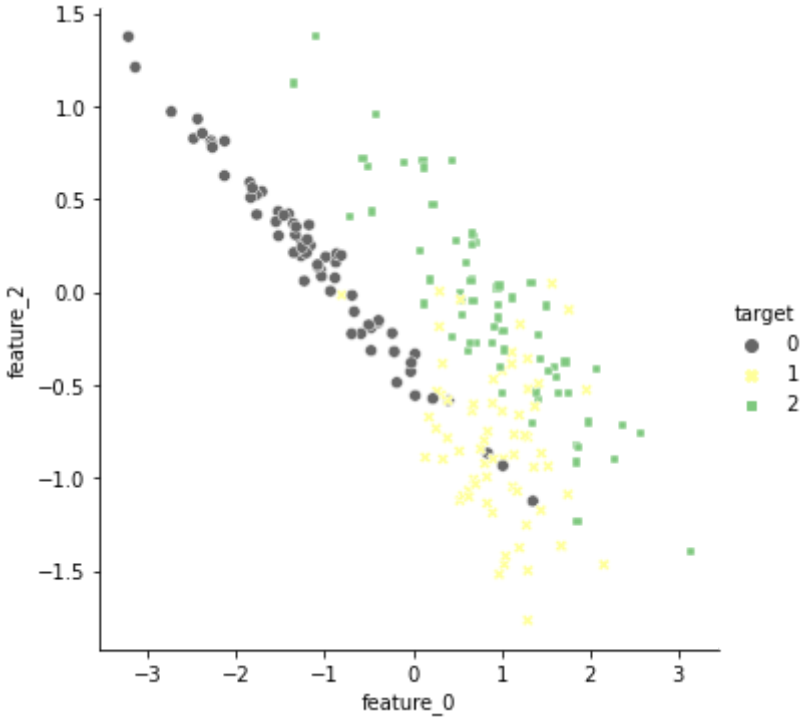
For class 1, -There is a strong, positive, linear association. There don't appear to be any outliers in the data.

For class 2, -There is a strong, negative, linear association with some outliers.

Try different palettes until you find find a suitable one then remove this comment. Available palettes on the lecture development machine were: Accent, Accent_r, Blues, Blues_r, BrBG, BrBG_r, BuGn, BuGn_r, BuPu, BuPu_r, CMRmap, CMRmap_r, Dark2, Dark2_r, GnBu, GnBu_r, Greens, Greens_r, Greys, Greys_r, OrRd, OrRd_r, Oranges, Oranges_r, PRGn, PRGn_r, Paired, Paired_r, Pastel1, Pastel1_r, Pastel2, Pastel2_r, PiYG, PiYG_r, PuBu, PuBuGn, PuBuGn_r, PuBu_r, PuOr, PuOr_r, PuRd, PuRd_r, Purples, Purples_r, RdBu, RdBu_r, RdGy, RdGy_r, RdPu, RdPu_r, RdYlBu, RdYlBu_r, RdYlGn, RdYlGn_r, Reds, Reds_r, Set1, Set1_r, Set2, Set2_r, Set3, Set3_r, Spectral, Spectral_r, Wistia, Wistia_r, YlGn, YlGnBu, YlGnBu_r, YlGn_r, YlOrBr, YlOrBr_r, YlOrRd, YlOrRd_r, afmhot, afmhot_r, autumn, autumn_r, binary, binary_r, bone, bone_r, brg, brg_r, bwr, bwr_r, cividis, cividis_r, cool, cool_r, coolwarm, coolwarm_r, copper, copper_r, cubehelix, cubehelix_r, flag, flag_r, gist_earth, gist_earth_r, gist_gray, gist_gray_r, gist_heat, gist_heat_r, gist_ncar, gist_ncar_r, gist_rainbow, gist_rainbow_r, gist_stern, gist_stern_r, gist_yarg, gist_yarg_r, gnuplot, gnuplot2, gnuplot2_r, gnuplot_r, gray, gray_r, hot, hot_r, hsv, hsv_r, icefire, icefire_r, inferno, inferno_r, jet, jet_r, magma, magma_r, mako, mako_r, nipy_spectral, nipy_spectral_r,

ocean, ocean_r, pink, pink_r, plasma, plasma_r, prism, prism_r, rainbow, rainbow_r, rocket, rocket_r, seismic, seismic_r, spring, spring_r, summer, summer_r, tab10, tab10_r, tab20, tab20_r, tab20b, tab20b_r, tab20c, tab20c_r, terrain, terrain_r, twilight, twilight_r, twilight_shifted, twilight_shifted_r, viridis, viridis_r, vlag, vlag_r, winter, winter_r.

In [16]:
```
sns.relplot(
    x='feature_0', y='feature_2', hue='target', style='target', palette='Accent_r',
plt.show()
```

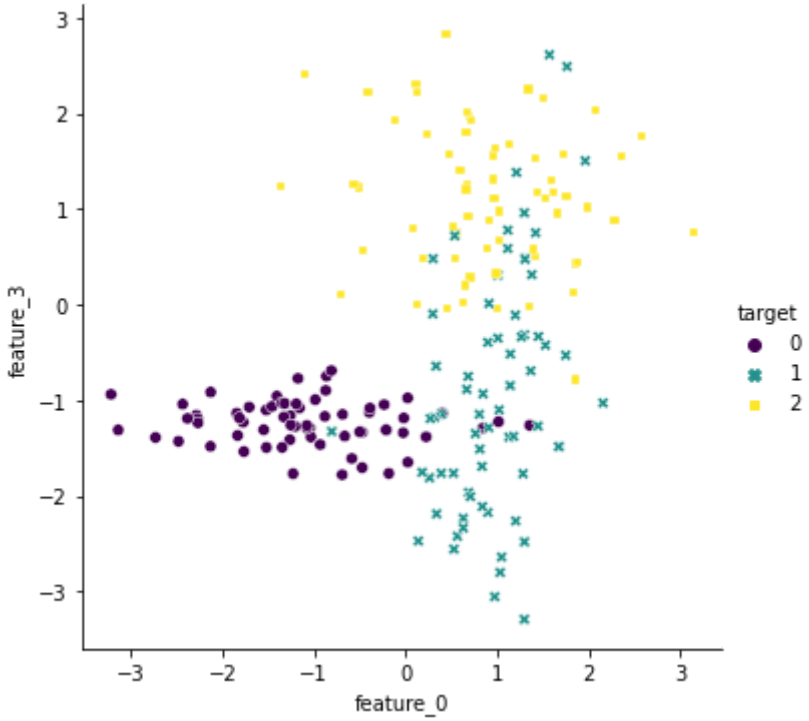


There is a moderately strong, positive, linear association. There don't appear to be any outliers in the data.

For class 1, -There is a strong, positive, linear association. There don't appear to be any outliers in the data.

For class 2, -There is a strong, negative, linear association with some outliers.

In [17]:
```
sns.relplot(
    x='feature_0', y='feature_3', hue='target', style='target', palette='viridis', d
plt.show()
```



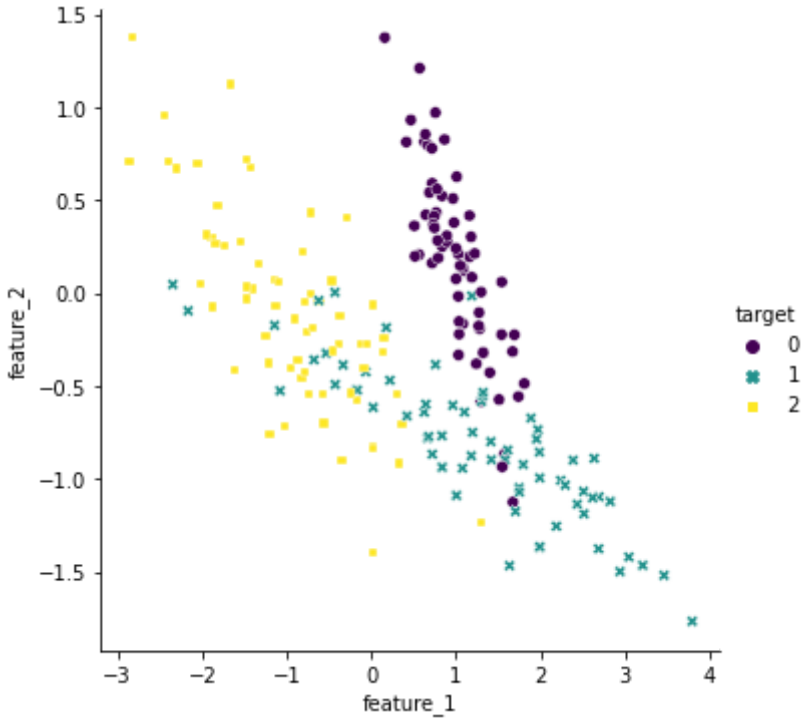
The association between feature_0 and feature_3:

For class 0, -There is moderately strong, positive, linear association. There don't appear to be any outliers in the data.

For class 1, -There is a strong, positive, no-linear association. There don't appear to be any outliers in the data.

For class 2, -There is a strong, positive, linear association with some outliers.

In [18]:
```
sns.relplot(
    x='feature_1', y='feature_2', hue='target', style='target', palette='viridis', d
plt.show()
```



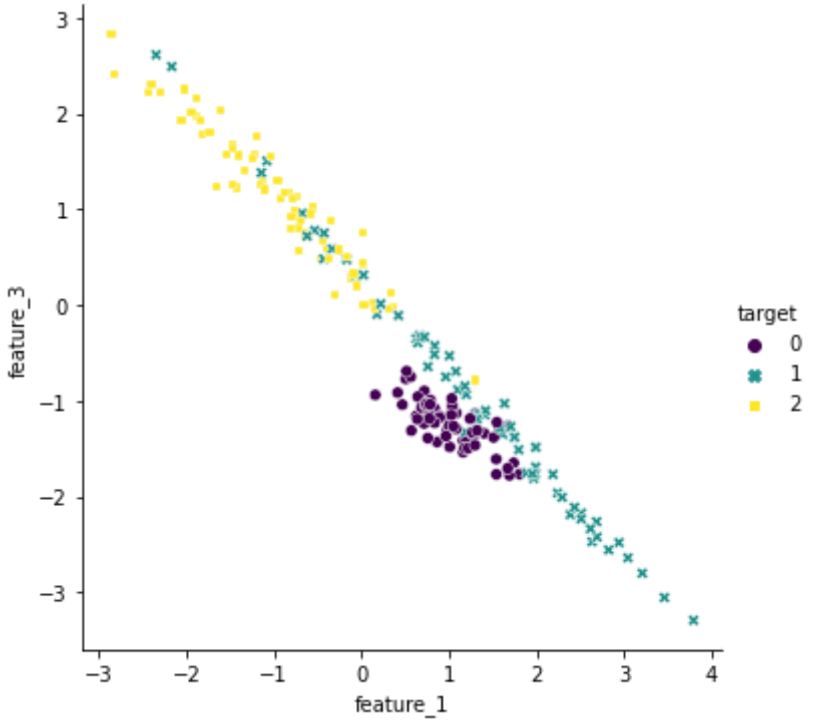
The association between feature_1 and feature_2:

For class 0, -There is a strong, positive, linear association with some outliers.

For class 1, -There is a strong, positive, linear association with a few potential outliers.

For class 2, -There is a strong, positive, linear association with a few potential outliers.

In [19]:
```
sns.relplot(
    x='feature_1', y='feature_3', hue='target', style='target', palette='viridis', d
plt.show()
```



The association between feature_1 and feature_3:

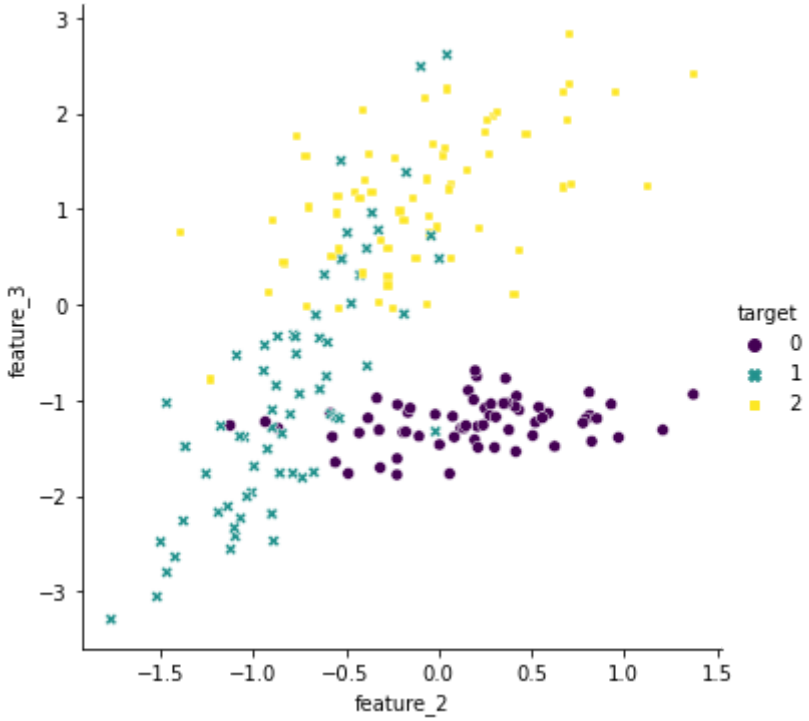
For class 0, -There is a moderately strong, positive, linear association. There don't appear to be any outliers in the data.

For class 1, -There is a strong, positive, linear association. There don't appear to be any outliers in the data.

For class 2, -There is a strong, negative, linear association with some outliers.

In [ ]:
```
```

In [20]:
```
sns.relplot(
    x='feature_2', y='feature_3', hue='target', style='target', palette='viridis', d
plt.show()
```



The association between feature_2 and feature_3:

For class 0, -There is a moderately strong, positive, linear association. There don't appear to be any outliers in the data.

For class 1, -There is a strong, positive, linear association. There don't appear to be any outliers in the data.

For class 2, -There is a strong, negative, linear association with some outliers.

In [ ]:

In [21]:
```
training_features, test_features, training_target, test_target = train_test_split(
    features, target, random_state=0)
```

Before training a model, we need to split our dataset into training and test sets because when training a model in machine learning, this is about the machine that is going to learn something. Here, our decision tree algorithm model is going to learn from data to make predictions. So our machine learning model is going to learn something from the dataset by understanding some correlations in the datasets. And imagine our machine learning model is running too much in the dataset like its learning too much of correlations then it will not perform great to the new datasets with slightly different correlations. So, we are going to build a machine learning models in the datasets and then we have to test it on the new sets which is going to be slightly different from the datasets on which we build the machine learning model. Now, we have to make two different sets, training sets on which we build the model and a test set on which we test the performance of this machine learning model. And the performance on the test sets should not be that different from the performance on the training set because this would mean that machine learning model understood well, the correlations and can adapt to the new sets of data.

We will use the train_test_split function from the scikit learn's model selection library. The features and targets arrays were passed to the function to perform train_test_split with random state of 727. The random state is used as a seed to the random number generation. This actually

ensures that the random numbers are generated in the same order every time we generate the random numbers.

In [ ]:

In [22]:
```
print(training_features.shape, test_features.shape)
```

```
(150, 4) (50, 4)
```

The above print statement is executed to print the shapes of training features and test features after splitting the features and targets into training and testing sets. The original datasets was of shape (200, 4) but now after this function, they are divided into (150, 4) and (50, 4) sets as train and test sets respectively. The shape of training_features= (150, 4) means that now there are 150 samples and 4 features each for training the model and test_features = (50, 4) means that now there are 50 samples and 4 features each for testing the performance of the model.

In [23]:
```
dtc = DecisionTreeClassifier(criterion='entropy')
dtc
```

Out[23]:
```
DecisionTreeClassifier(criterion='entropy')
```

class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, ccp_alpha=0.0)

As in this week, we are carrying out classification problem using a decision tree, the DecisionTreeClassifier() function from scikit learn was used. Decision trees are tree based learning algorithms, nonparametric supervised learning method that is used for classification and regression problems. The tree based methods empower predictive models with high accuracy, stability and ease of interpretation. It is a flowchart_like structure in which all the internal nodes represents a "test" on an attribute, each branch represents the outcome of the test and each leaf node represents a target label. In decision trees classification rules are the paths from root to leaf nodes.

The DecisionTreeClassifier() function was used above to classify and this is also capable of performing multi-class classification on a given datasets. Our dataset contains 3 target labels, so it is also a multi class classification problem. The DecisionTreeClassifier takes two input arrays, an array of X, of size [n_samples, n_features] which consists of the training samples and an array Y of integer values of size [n-samples] consisting the class labels of the training samples. This method takes different arguments which are discussed below:

-criterion: It is the function to measure the quality of a split. The supported criteria are “entropy” for the information gain and “gini” for the Gini impurity.

-splitter: It is the strategy to choose the split at each node. Supported strategies are best for the best split and random for the random split.

-max_depth: To provide the depth of the tree. And other parameters that can also be used are, min_samples_split, min_sample_leaf,min_weight_fraction_leaf, max_features, random_state, max_leaf_nodes, min_impurity_decrease, min_impurity_split, class_weight, presort. For the above functions, we have used criterion as entropy and all other default parameters.Entropy and information gain are the mathematical methods of choosing the best split. Entropy measures

the impurity of the training dataset and information gain measures the overall purity of the dataset. Decision Trees actually uses multiple algorithms to decide to split a node in two or more sub-nodes. The purity of the nodes increases with respect to the target variable. This algorithm splits the node on all the available variables and then selects the splits which results in most homogenous sub-nodes.

Gini index: It states, if we select two items from a population at random, then they must be of same class and probability. It is a metric for classification tasks in CART. It stores sum of squared probabilities of each class. Gini = 1- summation(Pi^2 for i = 1 to number of classes)

In [24]:
```
model = dtc.fit(training_features, training_target)
```

In [25]:
```
predictions = model.predict(test_features)
```

Now, we fit the decision tree classifier model. Fitting is same as training and after the model is trained, the model can used to make predictions. Usually with a .predict() method call. Here we are fitting our model (using .fit() method) to the training data, which is essentially the training part of the modeling process. This function finds the coefficients for the equation via the decision tree classification algorithm that we used earlier. Then, for a classifier, we can classify incoming new data points from either test sets or other source using the predict methods.

In our example above, the fit function is used which takes training_features and training_targets and feed this data to the decision tree model for training process. After it has been trained, the predict method is used on the test_features to test the performance of our decision tree model.

In [ ]:
```

```

In [26]:
```
matrix = confusion_matrix(predictions, test_target)
```

Confusion matrix is the performance measurement for machine learning classification problem. Here, output can be two or more than two classes. It is also used to evaluate the accuracy of a classification. In our program, it is a multiclass classification with 3 class labels. The confusion matrix is calculated with the scikit learns built in function matrix.confusion_matrix(). This function takes the following parameters: -y_true: it is an array of correct target values and its shape is [n_samples] -y_pred: it is an array of estimated targets which are returned by a classifier and its shape is [n_samples] -labels: it is the list of labels to index the matrix. This is the optional parameter. -sample_weight: sample weights

In the above function, to compute the confusion matrix, test_target and predictions were passed. Test target consists of the true label from the test datasets and predictions consists of the predicted labels. Based on these two inputs, the confusion_matrix function computes the confusion matrix.

Before evaluating the confusion matrix of our problem, let's discuss about it on more detail. Confusion matrix is extremely useful to measure Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve. In confusion matrix, the number of correct and incorrect predictions are summarized with count values and broken down by each class. It gives information on not only the errors that is being made by the classifier (decision trees in our case) but more importantly the types of errors that are being made. There are two types of error: Type 1 error: Error of the first kind. It is the rejection of true null hypothesis (also known as "false

positive" finding or conclusion). Type 2 error: Error of the second kind. It is the failure to reject a false null hypothesis (also known as "false negative" findings or conclusion).

Let's understand TP, FP, FN, TN: True Positive (TP): We predicted positive and it is true. True Negative (TN): We predicted negative and it is true. False Positive (FP): This is type 1 error. We predicted positive and it is false. False Negative (FN): This is type 2 error. We predicted negative and it is false.

In the above example, I have described predicted values as Positive and Negative and actual values as True and False. Now based on these values, we can create a confusion matrix.

In [27]:
```
print(matrix)
```

```
[[12  0  0]
 [ 2 11  2]
 [ 0  5 18]]
```

In [ ]:
```
```

In [28]:
```
print(classification_report(test_target, predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        14
           1       0.73      0.69      0.71        16
           2       0.78      0.90      0.84        20

    accuracy                           0.82        50
   macro avg       0.84      0.81      0.82        50
weighted avg       0.83      0.82      0.82        50
```

The above print statement is used to print the classification report ofthis problem. Here, the classification_report is used to show the main classification metrics for the decision tree classification problem. Thismethod takes y_true (correct target values) and y_pred (predicted target values returned by classifier) as a main parameters. This report gives us precision, recall, f1-score, support, accuracy, macro average, weighted average. These terms with their values from our classification report are discussed below. Precision: It is Positive Predictive Value (PPV). It gives, out of all the positive classes that we have predicted correctly, how many are actually positive. (True Positives / Predicted Positives)

Precision = TP / (TP + FP)

Here, from our model, we got precision of 0.83, 0.67, 1.00 respectively for classes 0, 1 and 2. Recall (Sensitivity): It gives, out of all the positive classes, how much our model predicted correctly. This should be high as possible. (TruePositives / All Actual Positives) Recall (Sensitivity) = TP / (TP + FN)

Here, from our model, we got recall of 0.75, 0.80, 0.93 respectively for classes 0, 1 and 2. Accuracy: It gives, out of all the classes, how much our model predicted correctly. (All correct / All) Accuracy = (TP + TN) / (TP + TN + FP + FN)

We got the accuracy of 82% from our model. F1-Score: This is the measures of test accuracy. The traditional FMeasure is the Harmonic Mean (HM) of the precision and recall. It helps to measure Recall and Precision at the same time.

F-Measure = (2 *Recall* Precision) / (Recall + Precision)

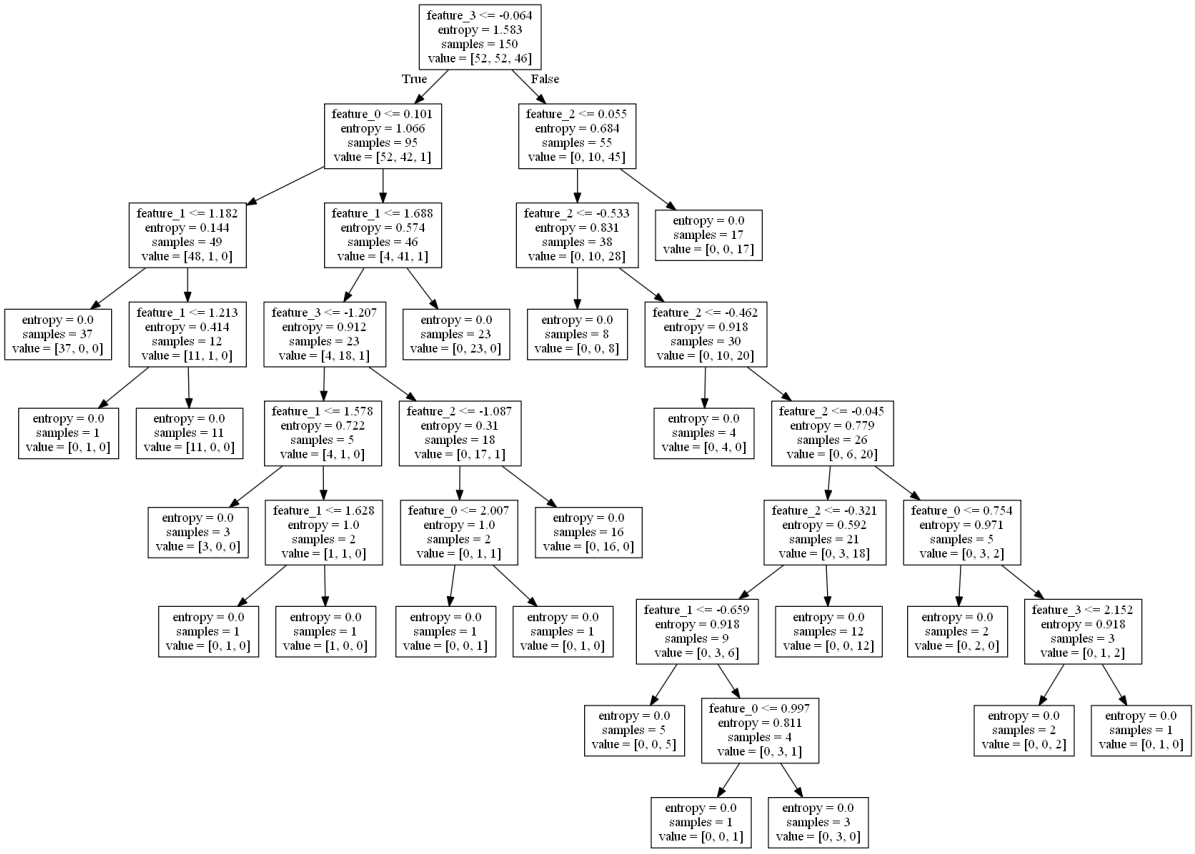
We got the f1-score of 0.79, 0.73 and 0.97 respectively for classes 0, 1 and 2.

Support: It is the total number of elements in each predicted class. Support for classes 0, 1 and 2 are 20, 15 and 15 respectively. Macro Average: it is the normal average. Micro Average = [Sum (Correct Classification) / Total Samples] Weighted Average = [Sum (Precision * Total Predicted)] / Total Sample Specificity: It is Negative Predicted Value (NPV). It gives, out of all the negative classes, how much our model predicted them correctly. (True Negatives / All Actual Negatives) Specificity = TN / (TN + FP)

In [29]:
```
dot_data = tree.export_graphviz(
    model, out_file=None, feature_names=feature_names)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png('decision_tree.png')
```

Out[29]: True

The above piece of code is used to plot the Decision Tree. We can also export the tree in Graphviz format using the export_graphviz function exporter. The export_graphivz exporter also supports a variety of aesthetic options, including coloring nodes according to their class and using explicit variables and class names if needed or wanted.



In [ ]:

In [ ]:

In [ ]: