Classification Trees using Python

One of the reasons why it is good to learn how to make decision trees in a programming language is that working with data can help in understanding the algorithm.

Load the Dataset

The Iris dataset is one of datasets scikit-learn comes with that do not require the downloading of any file from some external website. The code below loads the iris dataset.

```
import pandas as pd
from sklearn.datasets import load_irisdata = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Original Pandas df (features + target)

Splitting Data into Training and Test Sets

The code below puts 75% of the data into a training set and 25% of the data into a test set.

```
X_train, X_test, Y_train, Y_test =
train_test_split(df[data.feature_names], df['target'],
random_state=0)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0

X_train
X_test
Y_train
Y_test

The colors in the image indicate which variable (X_train, X_test, Y_train, Y_test) the data from the dataframe df went to for this particular train test split.

Note, one of the benefits of Decision Trees is that you don't have to standardize your data unlike PCA and logistic regression which are sensitive to effects of not standardizing your data.

Scikit-learn 4-Step Modeling Pattern

Step 1: Import the model you want to use

In scikit-learn, all machine learning models are implemented as Python classes

```
from sklearn.tree import DecisionTreeClassifier
```

Step 2: Make an instance of the Model

In the code below, I set the `max_depth = 2` to preprune my tree to make sure it doesn't have a depth greater than 2. I should note the next section of the tutorial will go over how to choose an optimal `max_depth` for your tree.

Also note that in my code below, I made `random_state = 0` so that you can get the same results as me.

```
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state = 0)
```

Step 3: Train the model on the data

The model is learning the relationship between X(sepal length, sepal width, petal length, and petal width) and Y(species of iris)

```
clf.fit(X_train, Y_train)
```

Step 4: Predict labels of unseen (test) data

```
# Predict for 1 observation
clf.predict(X_test.iloc[0].values.reshape(1, -1))# Predict for
multiple observations
clf.predict(X_test[0:10])
```

Remember, a prediction is just the majority class of the instances in a leaf node.

Measuring Model Performance

While there are other ways of measuring model performance (precision, recall, F1 Score, ROC Curve, etc), we are going to keep this simple and use accuracy as our metric.

Accuracy is defined as:

(fraction of correct predictions): correct predictions / total number of data points

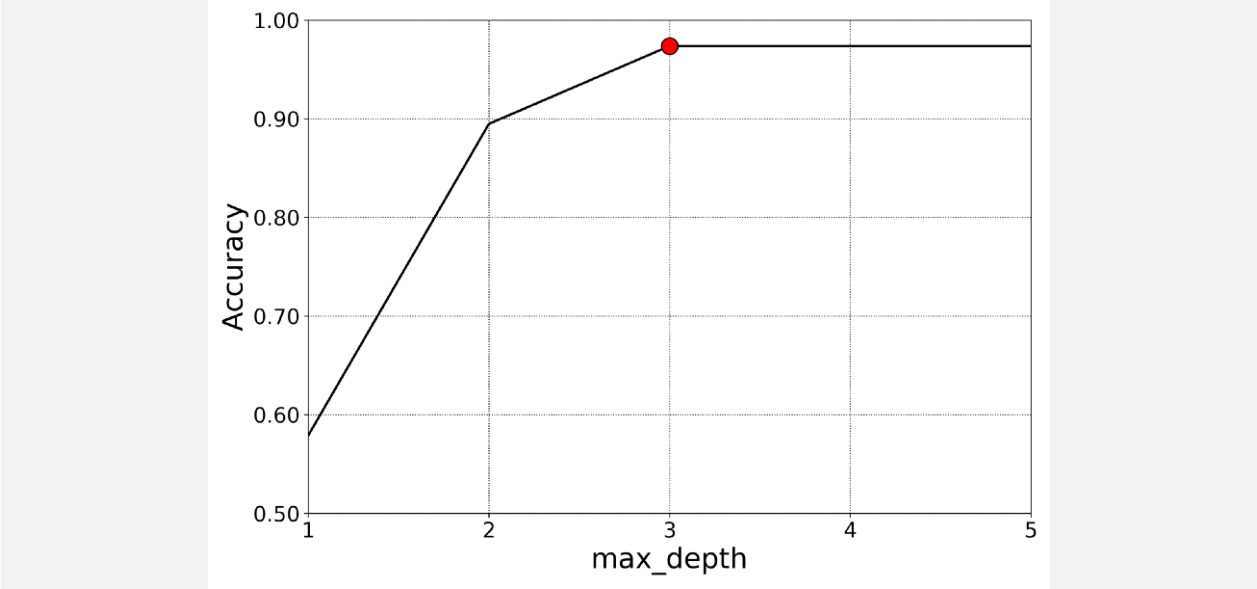
```
# The score method returns the accuracy of the model
score = clf.score(X_test, Y_test)
print(score)
```

Tuning the Depth of a Tree

Finding the optimal value for`max_depth` is one way way to tune your model. The code below outputs the accuracy for decision trees with different values for `max_depth`.

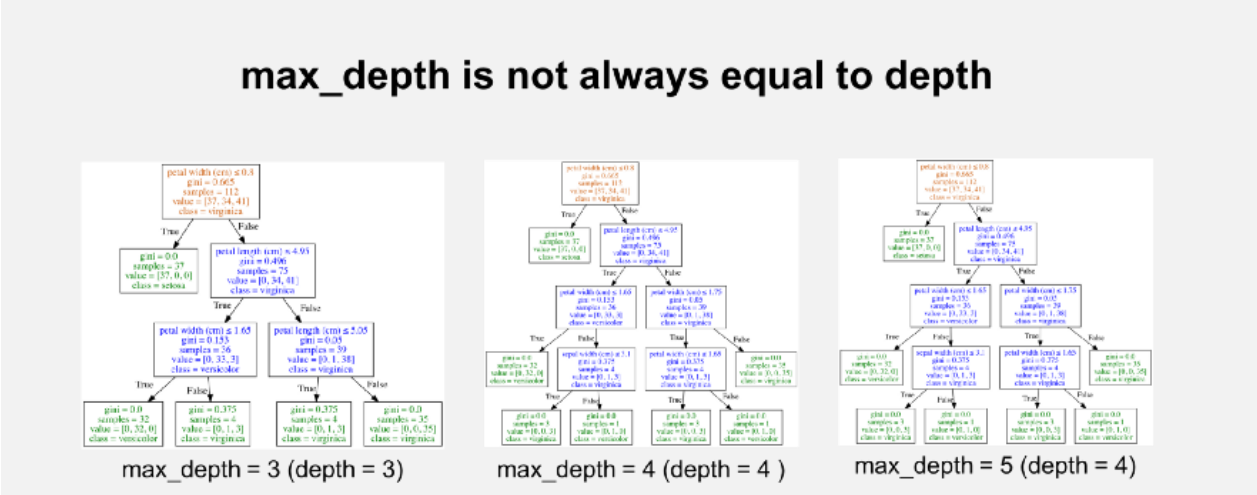
```
# List of values to try for max_depth:
max_depth_range = list(range(1, 6))# List to store the accuracy for each value of max_depth:
accuracy = []for depth in max_depth_range:

    clf = DecisionTreeClassifier(max_depth = depth,
                                 random_state = 0)
    clf.fit(X_train, Y_train)    score = clf.score(X_test, Y_test)
    accuracy.append(score)
```

Since the graph below shows that the best accuracy for the model is when the parameter `max_depth` is greater than or equal to 3, it might be best to choose the least complicated model with `max_depth = 3`.



I choose max_depth =3 as it it seems to be an accurate model and not the most complicated.

It is important to keep in mind that `max_depth` is not the same thing as depth of a decision tree. `max_depth` is a way to preprune a decision tree. In other words, if a tree is already as pure as possible at a depth, it will not continue to split. The image below shows decision trees with `max_depth` values of 3, 4, and 5. Notice that the trees with a `max_depth` of 4 and 5 are identical. They both have a depth of 4.



Notice how we have two of the exact same trees.

If you ever wonder what the depth of your trained decision tree is, you can use the `get_depth` method. Additionally, you can get the number of leaf nodes for a trained decision tree by using the `get_n_leaves` method. While this tutorial has covered changing selection criterion (Gini index, entropy, etc) and `max_depth` of a tree, keep in mind that you can also tune minimum samples for a node to split (`min_samples_leaf`), max number of leaf nodes (`max_leaf_nodes`), and more.

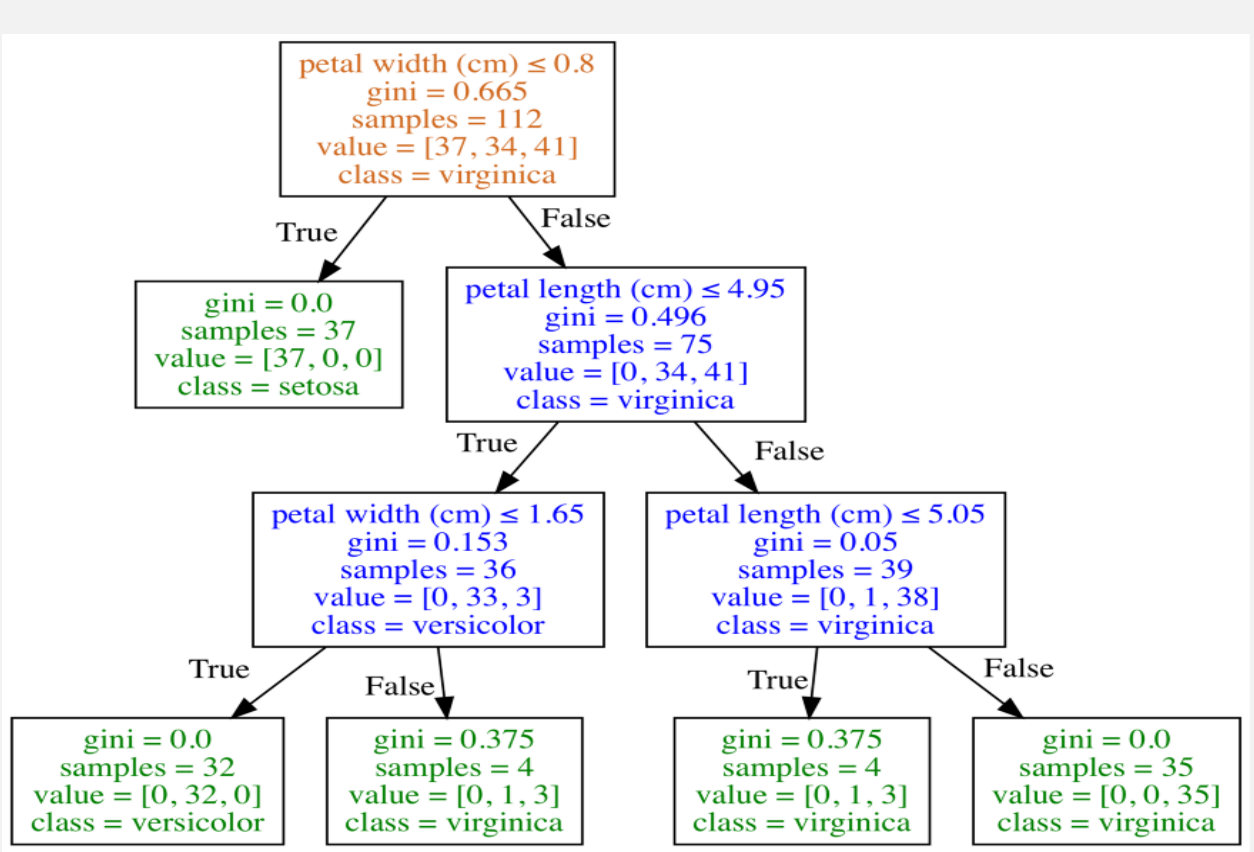
Feature Importance

One advantage of classification trees is that they are relatively easy to interpret. Classification trees in scikit-learn allow you to calculate feature importance which is the total amount that gini index or entropy decrease due to splits over a given feature. Scikit-learn outputs a number between 0 and 1 for each feature. All feature importances are normalized to sum to 1. The code below shows feature importances for each feature in a decision tree model.

```
importances =
pd.DataFrame({'feature':X_train.columns,'importance':np.round(clf.f
eature_importances_,3)})
importances = importances.sort_values('importance',ascending=False)
```

	feature	importance
3	petal width (cm)	0.578
2	petal length (cm)	0.422
0	sepal length (cm)	0.000
1	sepal width (cm)	0.000

In the example above (for a particular train test split of iris), the petal width has the highest feature importance weight. We can confirm by looking at the corresponding decision tree.



The only two features this decision tree splits on are petal width (cm) and petal length (cm).

Keep in mind that if a feature has a low feature importance value, it doesn't necessarily mean that the feature isn't important for prediction, it just means that the particular feature wasn't chosen at a particularly early level of the tree. It could also be that the feature could be identical or highly correlated with another informative feature. Feature importance values also don't tell you which class they are very predictive for or relationships between features which may influence prediction. It is important to note that when performing cross validation or similar, you can use an average of the feature importance values from multiple train test splits.