03_Decision_Tree

June 26, 2022

0.1 Decision Trees

A decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. It is popularly known as the **Classification and Regression Trees (CART)** algorithm.

A decision tree is a flowchart-like structure in which each `internal node` represents a `test on an attribute`, each `branch represents the outcome of the test`, and each `leaf node represents a class label` (decision taken after computing all attributes). The paths from the root to leaf represent `classification rules`.

0.1.1 Types of Decision Trees

Types of the decision tree are based on the type of target variable we have. It can be of two types:

1. **Classification tree**: has a categorical target variable.
2. **Regression tree**: has continuous target variable.

Example:-

Let's say we have a problem to predict whether a bike is good or not. This can be judged by using a decision tree classifier.

However, to qualify the bike into the good or bad category, mileage becomes an important factor. Mileage is measured using a contiguous value hence it can be measured using the decision tree regressor.

0.1.2 Important Terminology

Here's a look at the basic terminology used with Decision trees:

- **Root Node**: It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
- **Splitting**: It is a process of dividing a node into two or more sub-nodes.
- **Decision Node**: When a sub-node splits into further sub-nodes, then it is called a decision node.
- **Leaf/ Terminal Node**: Nodes that do not split are called Leaf or Terminal nodes.
- **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.

- **Branch / Sub-Tree**: A sub-section of entire tree is called a branch or sub-tree.
- **Parent and Child Node**: A node, which is divided into sub-nodes is called the parent node of sub-nodes whereas sub-nodes are the children of a parent node.

0.1.3 Decision Tree Algorithm Pseudocode

1. Place the best attribute of our dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

While building our decision tree classifier, we can improve its accuracy by tuning it with different parameters. But this tuning should be done carefully since by doing this our algorithm can overfit on our training data & ultimately it will build bad generalization model.

0.2 Splitting criteria for classification trees

Common options for the splitting criteria:

- **Classification error rate:** fraction of training observations in a region that don't belong to the most common class
- **Gini index:** measure of total variance across classes in a region

0.2.1 Example of classification error rate

Pretend we are predicting whether someone buys an iPhone or an Android:

- At a particular node, there are **25 observations** (phone buyers), of whom **10 bought iPhones and 15 bought Androids**.
- Since the majority class is **Android**, that's our prediction for all 25 observations, and thus the classification error rate is **10/25 = 40%**.

Our goal in making splits is to **reduce the classification error rate**. Let's try splitting on gender:

- **Males:** 2 iPhones and 12 Androids, thus the predicted class is Android
- **Females:** 8 iPhones and 3 Androids, thus the predicted class is iPhone
- Classification error rate after this split would be **5/25 = 20%**

Compare that with a split on age:

- **30 or younger:** 4 iPhones and 8 Androids, thus the predicted class is Android
- **31 or older:** 6 iPhones and 7 Androids, thus the predicted class is Android
- Classification error rate after this split would be **10/25 = 40%**

The decision tree algorithm will try **every possible split across all features**, and choose the split that **reduces the error rate the most.**

0.2.2 Gini impurity

Decision trees use the concept of **Gini impurity** to describe how homogeneous or "pure" a node is. A node is pure ($G = 0$) if all its samples belong to the same class, while a node with many samples from many different classes will have a Gini closer to 1.

More formally the Gini impurity of n training samples split across k classes is defined as:

where p[k] is the fraction of samples belonging to class k.

Example of Gini index Calculate the Gini index before making a split:

$$1 - \left(\frac{iPhone}{Total}\right)^2 - \left(\frac{Android}{Total}\right)^2 = 1 - \left(\frac{10}{25}\right)^2 - \left(\frac{15}{25}\right)^2 = 0.48$$

- The **maximum value** of the Gini index is 0.5, and occurs when the classes are perfectly balanced in a node.
- The **minimum value** of the Gini index is 0, and occurs when there is only one class represented in a node.
- A node with a lower Gini index is said to be more "pure".

Evaluating the split on **gender** using Gini index:

$$\text{Males: } 1 - \left(\frac{2}{14}\right)^2 - \left(\frac{12}{14}\right)^2 = 0.24$$

$$\text{Females: } 1 - \left(\frac{8}{11}\right)^2 - \left(\frac{3}{11}\right)^2 = 0.40$$

$$\text{Weighted Average: } 0.24\left(\frac{14}{25}\right) + 0.40\left(\frac{11}{25}\right) = 0.31$$

Evaluating the split on **age** using Gini index:

$$\text{30 or younger: } 1 - \left(\frac{4}{12}\right)^2 - \left(\frac{8}{12}\right)^2 = 0.44$$

$$\text{31 or older: } 1 - \left(\frac{6}{13}\right)^2 - \left(\frac{7}{13}\right)^2 = 0.50$$

$$\text{Weighted Average: } 0.44\left(\frac{12}{25}\right) + 0.50\left(\frac{13}{25}\right) = 0.47$$

Again, the decision tree algorithm will try **every possible split**, and will choose the split that **reduces the Gini index (and thus increases the "node purity") the most.**

0.2.3 Iris Data Set

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Ronald Fisher in his 1936 paper 'The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis'.

This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray

The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

[1]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
```

[2]:
```python
# read the iris data into a DataFrame
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
col_names =␣
↪['sepal_length','sepal_width','petal_length','petal_width','species']
iris = pd.read_csv(url, header=None, names=col_names)
iris.head()
```

[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[3]:
```python
#shape
print(iris.shape)
```

```
(150, 5)
```

[4]:
```python
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[6]:
```python
# store feature matrix in "X"
feature_cols = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
X = iris[feature_cols]
```

[7]:
```python
# store response vector in "y"
y = iris.species
```

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=5)
clf = DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
```

[9]:
```
# predict for unknown sample
clf.predict([[5.3,3.6,1.4,1.1]])
```

```
C:\Users\Administrator\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but DecisionTreeClassifier was
fitted with feature names
  warnings.warn(
```

[9]:
```
array(['Iris-versicolor'], dtype=object)
```

[10]:
```
y_pred = clf.predict(X_test)
```

[11]:
```
y_pred
```

[11]:
```
array(['Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-virginica', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica'],
      dtype=object)
```

[12]:
```
print(classification_report(y_test, y_pred))
print('\nAccuracy: {0:0.2f}'.format(accuracy_score(y_test, y_pred)))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       1.00      0.82      0.90        11
 Iris-virginica       0.85      1.00      0.92        11

       accuracy                           0.93        30
      macro avg       0.95      0.94      0.94        30
   weighted avg       0.94      0.93      0.93        30


Accuracy: 0.93
```

- **Precision** is the ability of a classiifer not to label an instance positive that is actually negative - **positive prediction rate**.

- **Recall** is the ability of a classifier to find all positive instances - **sensitivity**.
- The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0.
- Support is the number of actual occurrences of the class in the specified dataset.

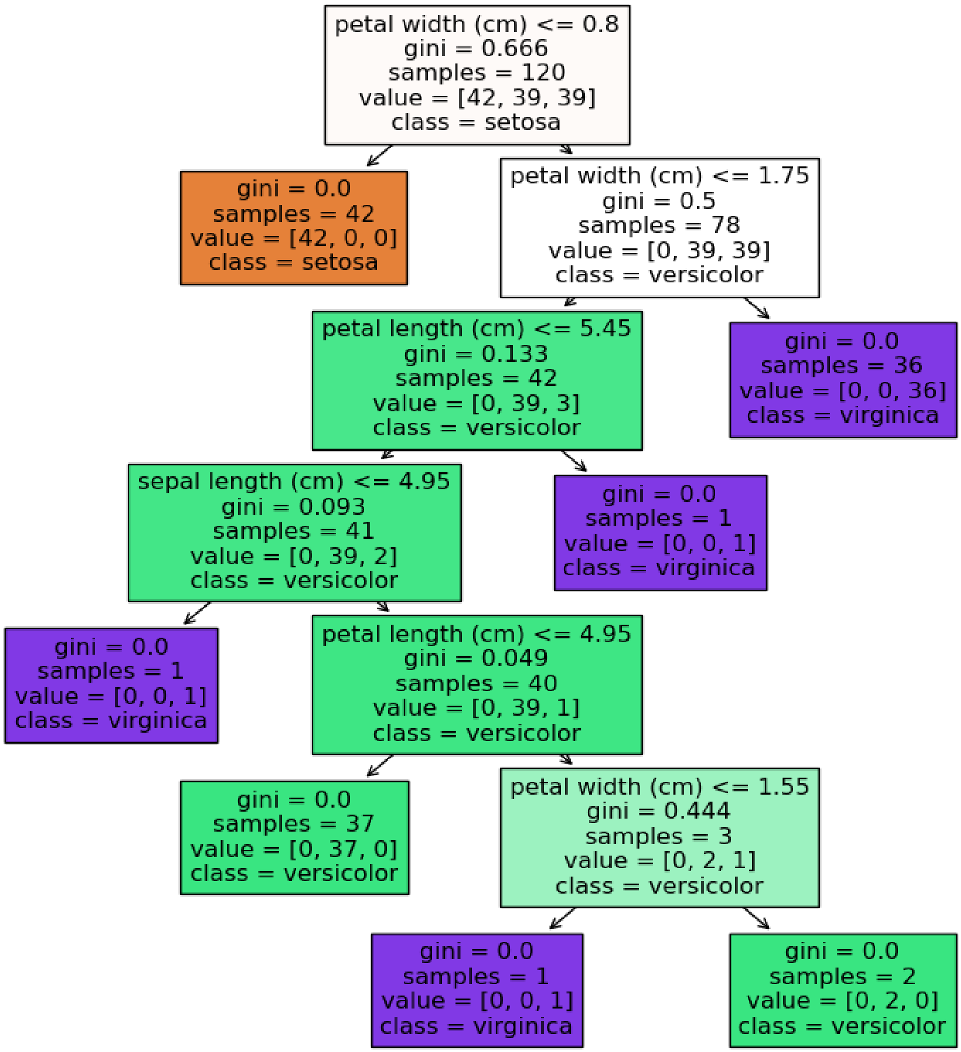
[13]:
```
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
```

[13]:

	Actual	Predicted
82	Iris-versicolor	Iris-versicolor
134	Iris-virginica	Iris-virginica
114	Iris-virginica	Iris-virginica
42	Iris-setosa	Iris-setosa
109	Iris-virginica	Iris-virginica
57	Iris-versicolor	Iris-virginica
1	Iris-setosa	Iris-setosa
70	Iris-versicolor	Iris-virginica
25	Iris-setosa	Iris-setosa
84	Iris-versicolor	Iris-versicolor
66	Iris-versicolor	Iris-versicolor
133	Iris-virginica	Iris-virginica
102	Iris-virginica	Iris-virginica
107	Iris-virginica	Iris-virginica
26	Iris-setosa	Iris-setosa
23	Iris-setosa	Iris-setosa
123	Iris-virginica	Iris-virginica
130	Iris-virginica	Iris-virginica
21	Iris-setosa	Iris-setosa
12	Iris-setosa	Iris-setosa
71	Iris-versicolor	Iris-versicolor
128	Iris-virginica	Iris-virginica
48	Iris-setosa	Iris-setosa
72	Iris-versicolor	Iris-versicolor
88	Iris-versicolor	Iris-versicolor
148	Iris-virginica	Iris-virginica
74	Iris-versicolor	Iris-versicolor
96	Iris-versicolor	Iris-versicolor
63	Iris-versicolor	Iris-versicolor
132	Iris-virginica	Iris-virginica

[14]:
```
from sklearn import tree
import matplotlib.pyplot as plt
fn=['sepal length (cm)','sepal width (cm)',
    'petal length (cm)','petal width (cm)']
cn=['setosa', 'versicolor', 'virginica']
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (10,10), dpi=100)
tree.plot_tree(clf,
```

```
                    feature_names = fn,
                    class_names=cn,
                    filled = True);
fig.savefig('classification_tree_iris.png')
```



We're using the Gini impurity as our metric

Now let's mess with the minimum impurity decrease parameter to see if we can prune this tree and how that affects the accuracy. Let's set it to 0.1. This will tell our classifier that a **node will split if its impurity is above the threshold, otherwise it is a leaf**.

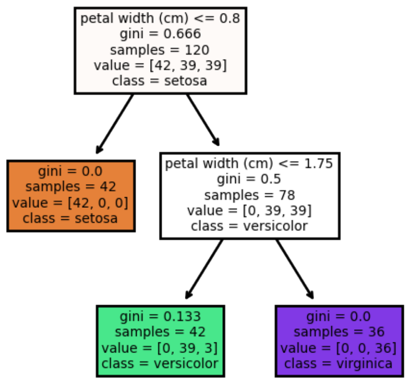
[15]:
```
clf1 = DecisionTreeClassifier(random_state=5, min_impurity_decrease=0.1)

clf1.fit(X_train, y_train)
```

[15]: DecisionTreeClassifier(min_impurity_decrease=0.1, random_state=5)

[ ]:
```
DecisionTreeClassifier?
```

[16]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (3,3), dpi=200)
tree.plot_tree(clf1,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('classification_tree_iris1.png')
```



If we set this parameter to be too large, then everything will collapse into a single node with a poor accuracy. Remember that the larger the value, the more aggressive pruning!

Advantages of decision trees:

- Can be used for regression or classification
- Can be displayed graphically
- Highly interpretable, can be specified as a series of rules, and more closely approximate human decision-making than other models
- Prediction is fast
- Features don't need scaling
- Automatically learns feature interactions
- Tends to ignore irrelevant features

Disadvantages of decision trees:

- Performance is (generally) not competitive with the best supervised learning methods
- Can easily overfit the training data (tuning is required)
- Small variations in the data can result in a completely different tree (high variance)
- Doesn't tend to work well if the classes are highly unbalanced
- Doesn't tend to work well with very small datasets

```
[ ]:
```