Decision Trees

Learning Objectives

- Apply decision trees to classification/regression problems
- Calculate different measurements (Entropy, Gini) for quantifying branching in decision trees
- Apply methods to avoid overfitting
- Apply recursive methods in your programming

Success Criteria

Today I will be successful if I can…

- Define root, parent, child, leaf node on a decision tree
- Explain how a Decision Tree determines a split
- Calculate mini impurity and Shannon entropy of example trees
- Express a danger of a fully grown decision tree in terms of bias/variance
- Explain the following parameters of a decision tree:
  - criterion
  - max_depth
  - min_impurity_split

Example

Imagine a factory wants to build a machine for its production line that identifies the type of fruits based on their color and size.

A sample of data is given to you and you want to create a mechanism to find the fruit's type based on the given data (color and size).

What is the best way to do this?

Fruit Type	Size	Color
Apple	6	Yellow
Apple	5	Green
Orange	7	Orange
Orange	5	Orange
Lemon	6	Yellow

Example - Con’t

Our goal is to achieve the best classification with the least number of decisions!
We use a decision tree to do this task.

A decision tree is a (binary) tree, with each branch (non-leaf) node having a left and right child node. Each branch node specifies a feature (predictor, attributes) to split along with information on which values should pass to the left/right node.

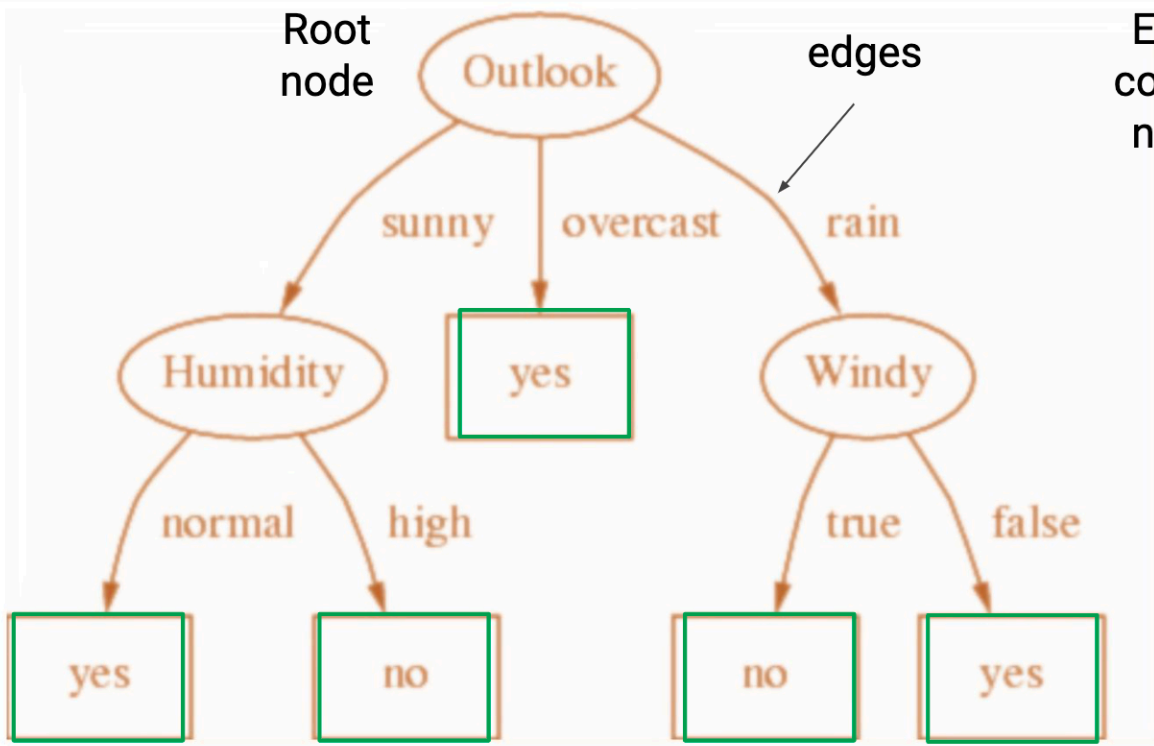
Decision tree :

- Each internal node tests a predictor (attribute, variable, feature)
- Each branch corresponds to a predictor value
- Each leaf node assigns a classification

Parts of our Tree

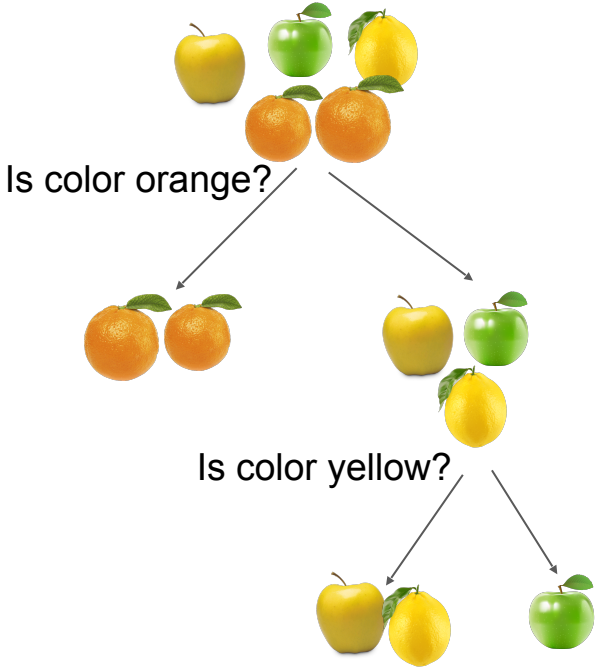
"Nodes" are where data are split on features

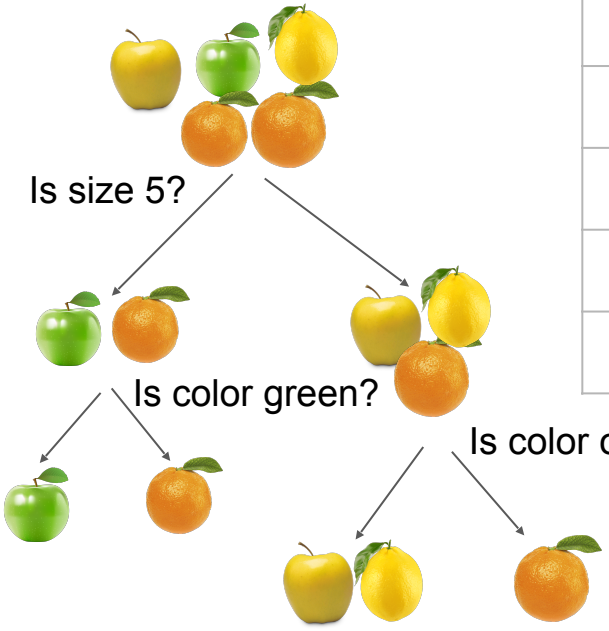
Terminal node, a.k.a "leaf", the final result



Example - Con’t

There are different ways to classify the fruits:





Fruit Type	Size	Color
Apple	6	Yellow
Apple	5	Green
Orange	7	Orange
Orange	5	Orange
Lemon	6	Yellow

Example - Con’t

What is the best question to ask at each step?

Do we have any measure (quantity) which tells us what is the best question to ask?

After proposing a split, we can check how much we reduce the amount of randomness and thus how much information is potentially gained!

Different quantities have been introduced to measure randomness and information gain in a system. One of these quantities is Gini impurity.

Gini Impurity

Gini impurity is defined by the probability of object *j* in a set being identified correctly multiply by the probability it is identified incorrectly, summed over all objects:

$$\mathrm{Gini}(S) = 1 - \sum_{i \in S} p_i^2$$

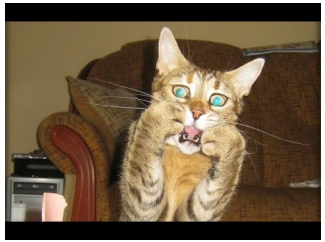
How does this help us find the right split?

The right split is a split that results in the minimum gini impurity (or gains the most information). The information gain is calculated by:

$$\mathrm{IG}(S, C) = H(S) - \sum_{C_i \in C} \frac{|C_i|}{|S|} H(C_i)$$

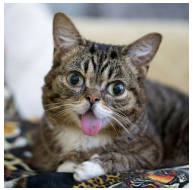
Making the Information Gain formula less intimidating

$$\mathrm{IG}(S, C) = H(S) - \sum_{C_i \in C} \frac{|C_i|}{|S|} H(C_i)$$



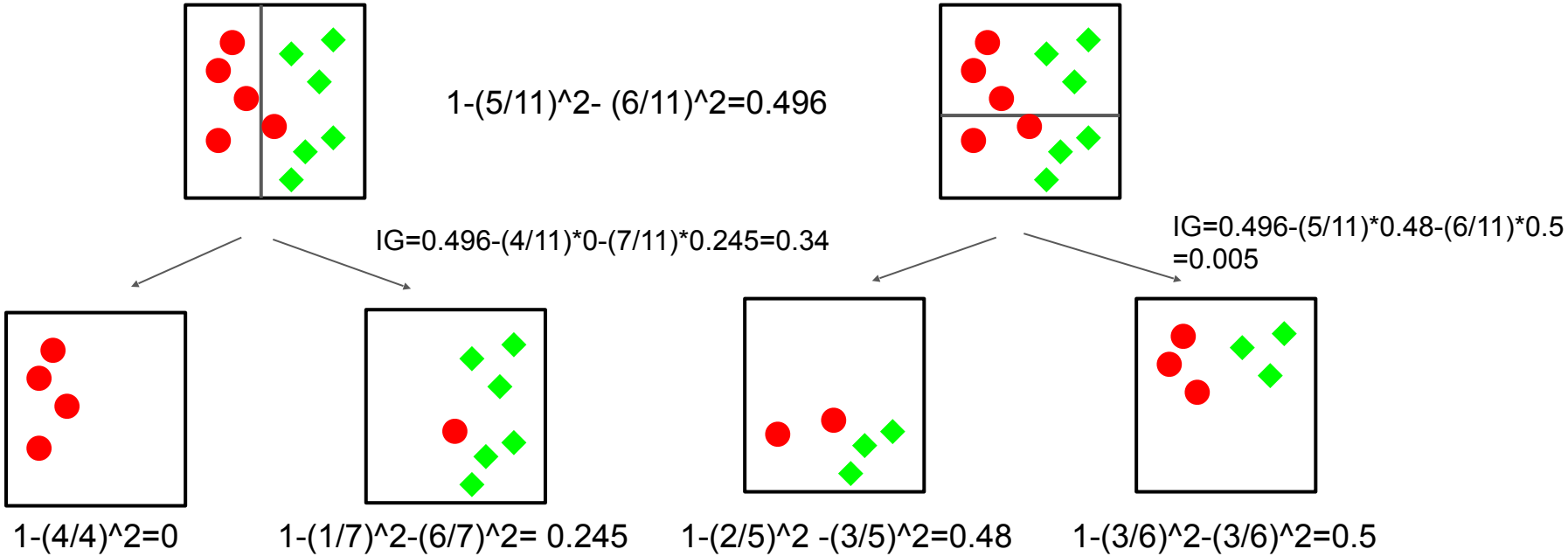
PI = Parent Impurity
C = Child
CI = Child Impurity

PI - sum of ((proportion of data that ends up in C_i) * (CI_i) for each child node)



Example

Below are two examples of calculating Gini impurity and information gain.



How Do We Find the Best Tree?

1-2*(2/5)^2-1*(1/5)^2=0.64

Is color orange?

IG = 0.64-(2/5)*0-(3/5)*0.44=0.376

1 - 1 =0

1-1*(2/3)^2-1*(1/3)^2=0.44

Is color yellow?

IG = 0.44-(2/3)*0.5-(1/3)*0=0.106

1 - 2*(1/2)^2=0.5

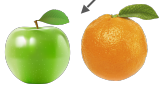
1 - 1 = 0

How Do We Find the Best Tree?

1 - 2*(2/5)^2 - 1*(1/5)^2 = 0.64

Is size 5?

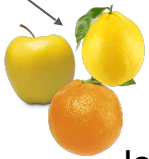
IG=0.64-(2/5)*0.5-(3/5)*0.67 = 0.038



1-2*(1/2)^2=0.5

Is color green?

IG=0.5



1-3*(1/3)^2=0.67

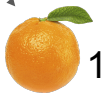
Is color orange?

1-1 = 0

1-1=0

IG =0.67-(2/3)*0.5-(1/3)*0=0.37

1-2*(1/2)^2=0.5



1-1 = 0

Decision Tree - Splitting Algorithms

How do we build a tree with many features (predictors, attributes) and data?

We follow the same procedure as we did in our example:

- Start with the whole data and create all the possible binary decisions based on each feature (predictor):
  - For discrete features the decision is class no class
  - For continues features the decision is threshold <value or threshold ≥value
- Calculate the gini impurity for every decision
- Pick the decision which reduces the impurity the most

Different Types of Decision Trees

1. **Classification Trees**

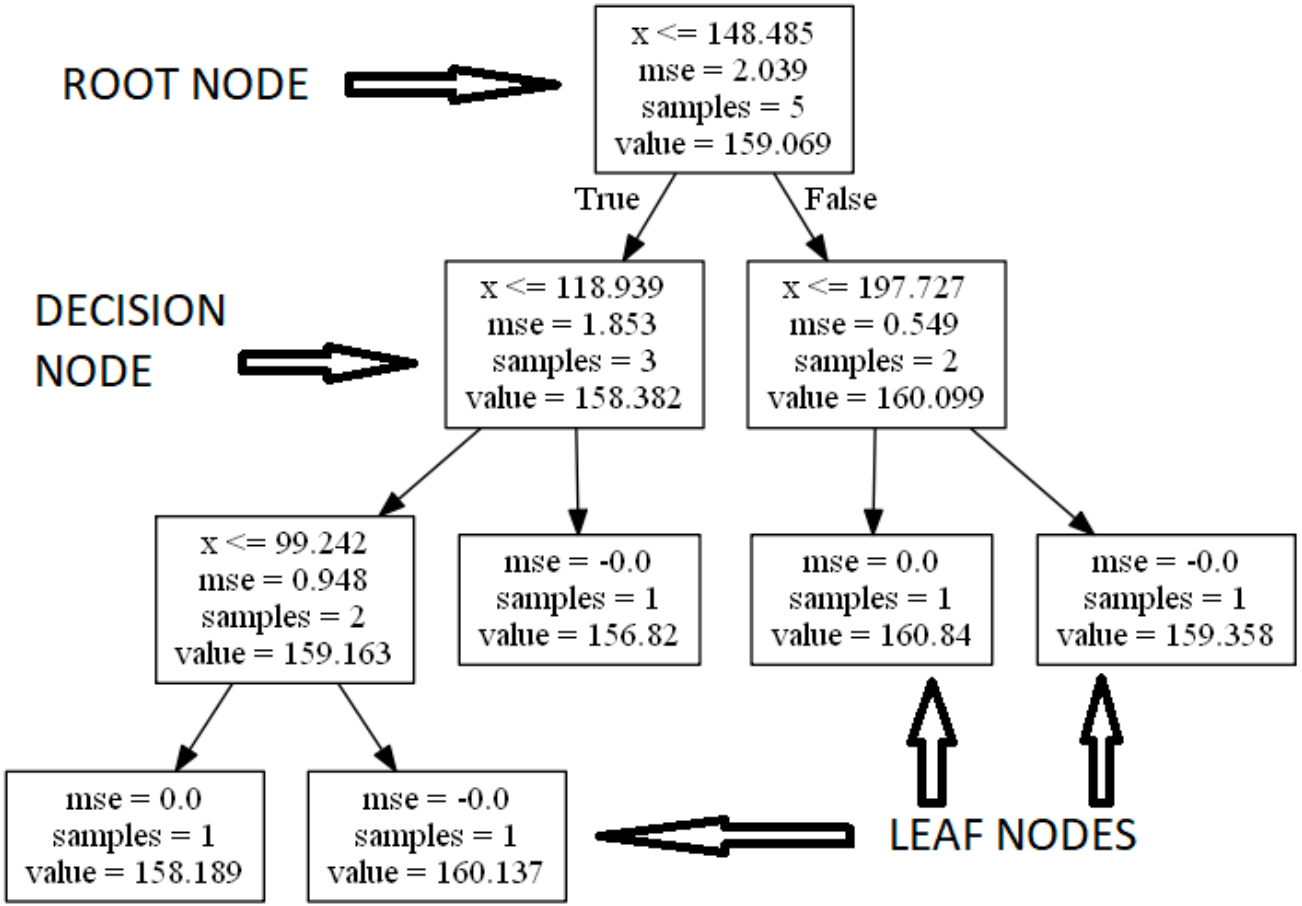
   In classification trees our outcomes (target, output) are discrete. Leaf values are typically set to the most common outcomes.

1. **Regression Trees**

   In regression trees our outcomes (target, output) are continuous. Leaf values are typically set to the mean value in outcomes. In Regression trees we use RSS Instead of Gini/entropy.

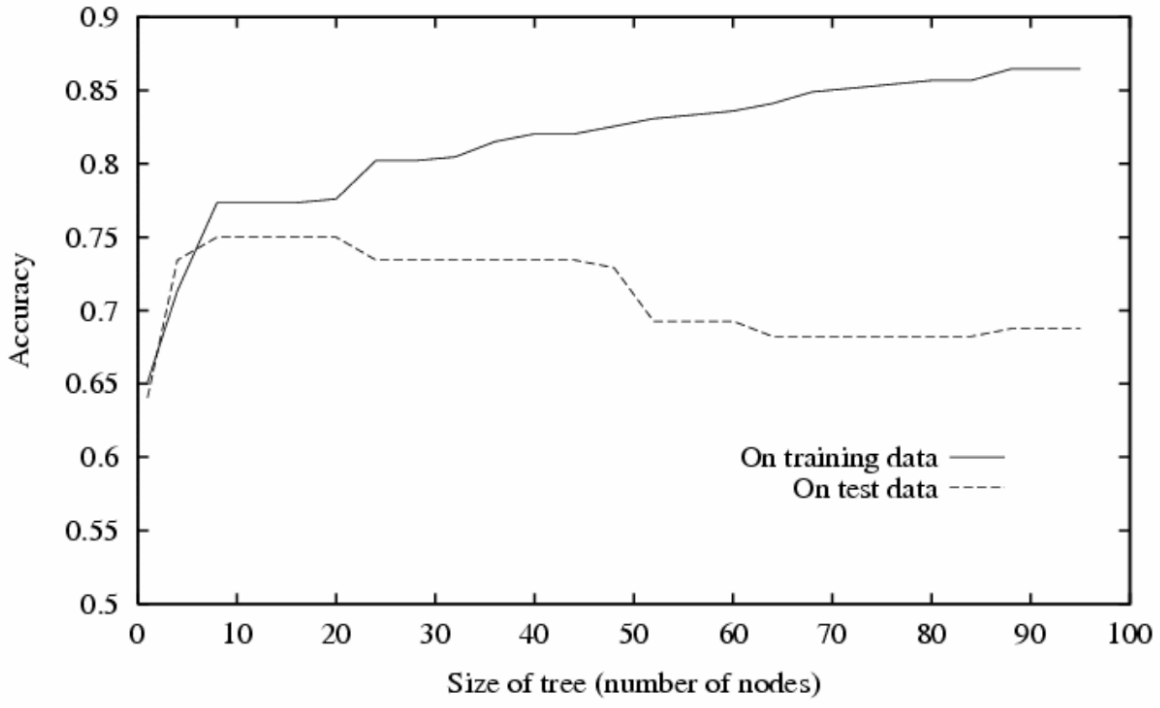
Features (inputs, predictors) can be either discrete or continuous for both classification and regression trees.

Example of Regression Tree



https://medium.com/@dhiraj8899

Overfitting



Overfitting

Overfitting is likely if you build your tree all the way until every leaf is pure.

Prepruning ideas (prune while you build the tree):

- **leaf size:** stop splitting when #examples gets small enough
- **depth:** stop splitting at a certain depth
- **purity:** stop splitting if enough of the examples are the same class
- **gain threshold:** stop splitting when the information gain becomes too small

Postpruning ideas (prune after you've finished building the tree):

- merge leaves if doing so decreases test-set error (very similar to how we removed features from our regression models using a function we called EliminateOne)

Other Splitting Measures - Entropy

There are other measures besides gini impurity that quantify the randomness and gain of information when we are creating branches.

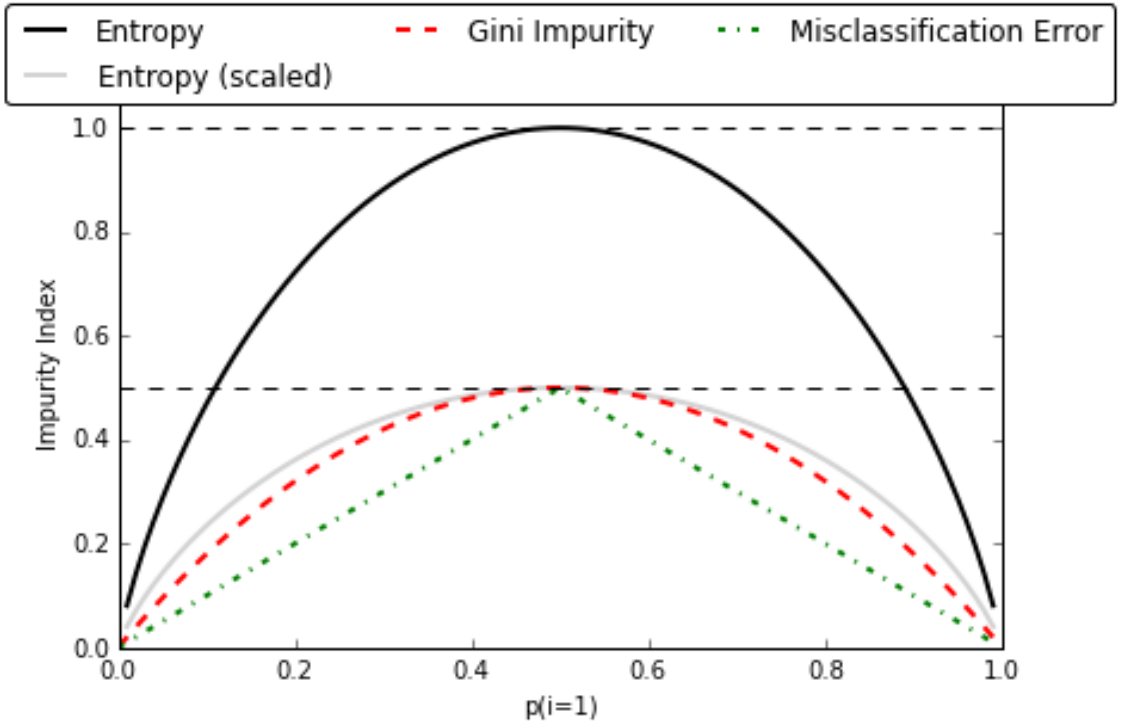
Entropy is another measure that quantifies randomness. If the probability of identifying the object *j* correctly is p_j and the probability of identifying it incorrectly is 1-p_j then the entropy is defined:

$$H(X) = -\sum_i p_i log_2(p_i)$$

The information gain is calculated by:

$$\mathrm{IG}(S, C) = H(S) - \sum_{C_i \in C} \frac{|C_i|}{|S|} H(C_i)$$

Splitting Measure



Sklearn

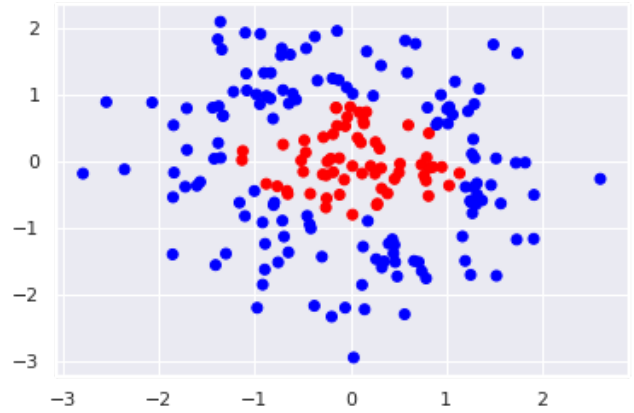
```
from sklearn import tree
tree.DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=2,
        max_features=None, max_leaf_nodes=None,
          min_impurity_decrease=0.0, min_impurity_split=None,
          min_samples_leaf=1, min_samples_split=2,
          min_weight_fraction_leaf=0.0, presort=False, random_state=None,
          splitter='best')

tree.DecisionTreeRegressor(max_depth=2)
```

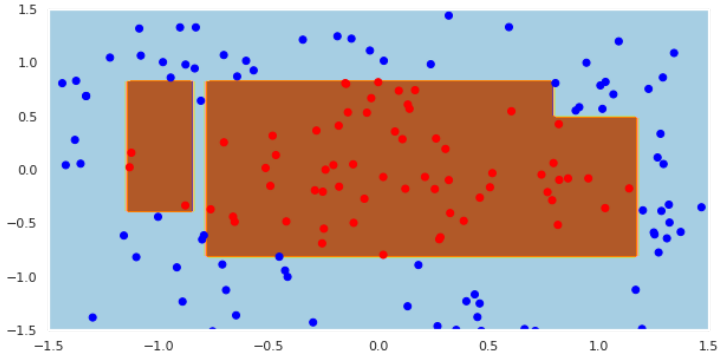
Pruning with max_depth, min_samples_split, min_samples_leaf or max_leaf_nodes

- Gini is default, but you can also choose entropy

Example

```
from sklearn import tree
import matplotlib.pyplot as plt

X_ring = np.random.normal(size=(200,2))
y_ring = (X_ring[:,0]**2 + X_ring[:,1]**2) < 0.7
plt.scatter(X_ring[:,0], X_ring[:,1], c=y_ring, cmap='bwr')

dt = tree.DecisionTreeClassifier(criterion='entropy')
dt.fit(X_ring, y_ring)

plot_colors = "br"
plot_step = 0.02
plt.figure(figsize=(10, 5))

# Plot the decision boundaries
x_min, x_max = X_ring[:, 0].min() - 1, X_ring[:, 0].max() + 1
y_min, y_max = X_ring[:, 1].min() - 1, X_ring[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

Z = dt.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.scatter(X_ring[:,0], X_ring[:,1], c=y_ring, cmap='bwr');
plt.axis("tight")
plt.xlim(-1.5,1.5)
plt.ylim(-1.5,1.5)
```





Recursion

Recursion uses the idea of "divide and conquer" to solve problems. It divides a complex problem you are working on into smaller sub-problems that are easily solved, rather than trying to solve the complex problem directly.

Three Laws of Recursion

1. A recursive algorithm must have a base case.
2. A recursive algorithm must change its state and move toward the base case.
3. A recursive algorithm must call itself, recursively.

Example - Recursion

```
def factorial(x):
    """Recursively calculate x!"""
    # base case is when we get to x=0, which is 0! = 1
    if x == 0:
        return 1
    # otherwise, recursive case, notice how we are reducing x
    else:
        return x * factorial(x-1)

def power(base, exp):
    """Recursively caclulate base ** exp"""
    # base case is when exp = 0, base ** 0 = 1
    if exp == 0:
        return 1
    #  otherwise, recursive case, reduce exp
    return base * power(base, exp - 1)
```