DEPAUL UNIVERSITY

DSC478: Programming Machine Learning Applications

Roselyne Tchoua

rtchoua@depaul.edu

School of Computing, CDM, DePaul University

Classification: 3 Step Process

1. **Model construction** (**Learning**):
   - Each record (instance, example) is assumed to belong to a predefined class, as determined by one of the attributes
     - This attribute is called the target attribute
     - The values of the target attribute are the **class labels**
   - The set of all instances used for learning the model is called **training set**
2. **Model Evaluation** (**Accuracy**):
   - Estimate accuracy rate of the model based on a **test set**
   - The known labels of test instances are compared with the predicts class from model
   - Test set is independent of training set otherwise over-fitting will occur
3. **Model Use** (**Classification**):
   - The model is used to classify unseen instances (i.e., to predict the class labels for new unclassified instances)
   - Predict the value of an actual attribute

Decision Trees: Basic Idea

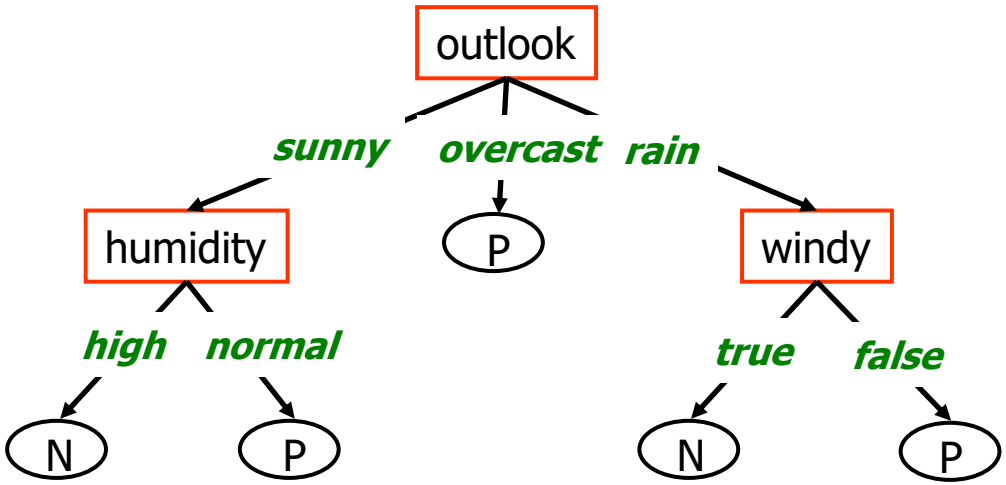
- An *inductive learning task*
  - Use particular facts to make more generalized conclusions

- A predictive model based on a branching series of Boolean tests
  - These smaller Boolean tests are less complex than a one-stage classifier

Decition Tress

A decision tree is a flow-chart-like tree structure

- Internal node denotes a test on an attribute (feature)
- Branch represents an outcome of the test
- All records in a branch have the same value for the tested attribute
- Leaf node represents class label or class label distribution



DT Classification

Example: "is it a good day to play golf?"

- a set of attributes and their possible values:

outlook	sunny, overcast, rain
temperature	cool, mild, hot
humidity	high, normal
windy	true, false

A particular *instance* in the training set might be:

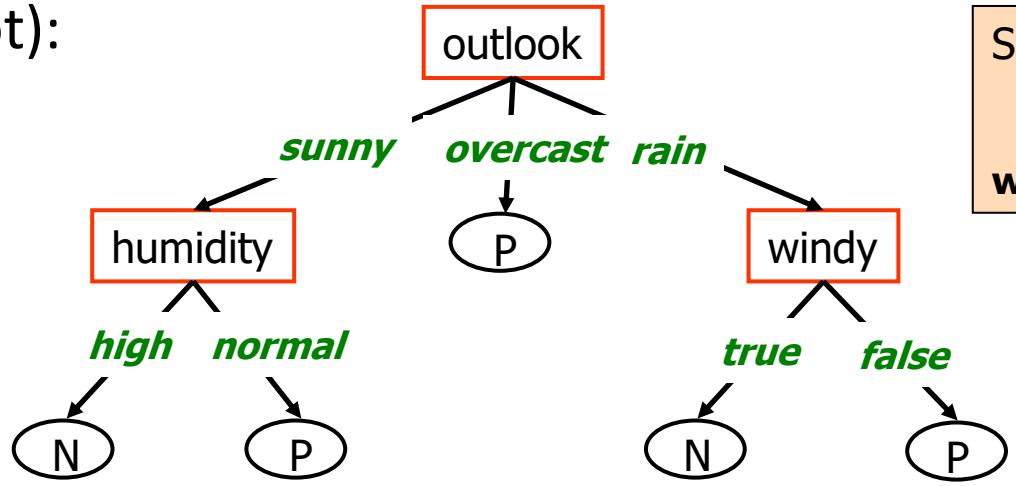
<overcast, hot, normal, false>: play

In this case, the target class is a binary attribute, so each instance represents a positive or a negative example.

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

DT Classification

- Examples can be classified as follows
  1. look at the example's value for the feature specified
  2. move along the edge labeled with this value
  3. if you reach a leaf, return the label of the leaf
  4. otherwise, repeat from step 1
- Example (a decision tree to decide whether to play golf or not):



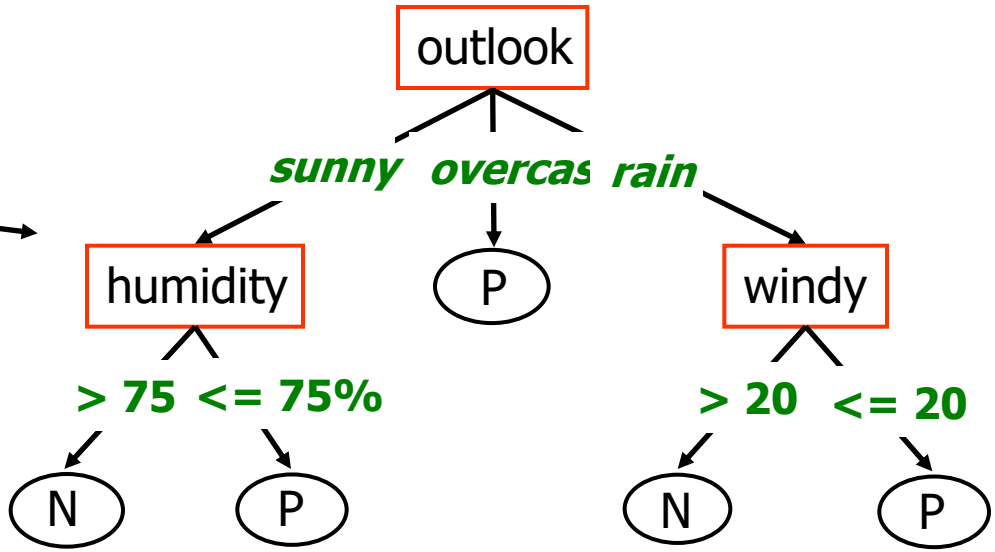
So, a new instance:

<rainy, hot, normal, true>: ?

will be classified as "noplay"

DT and Decision Rules

If attributes are continuous, internal nodes may test against a threshold.



Each path in the tree represents a decision rule:

Rule1:
If (outlook="sunny") AND (humidity<=0.75)
Then (play="yes")

Rule2:
If (outlook="rainy") AND (wind>20)
Then (play="no")

Rule3:
If (outlook="overcast")
Then (play="yes")

. . .

Top-Down DT Generation

The basic approach usually consists of two phases:

- Tree construction
  - At the start, all the training instances are at the *root*
  - Partition instances recursively based on selected attributes
- Tree pruning (to improve accuracy)
  - remove tree branches that may reflect noise in the training data and lead to errors when classifying test data

Top-Down DT Generation

Basic Steps in Decision Tree Construction

- Tree starts a single node representing all data
- If instances are all same class, then node becomes a leaf labeled with class label
- Otherwise, ***select feature*** that best distinguishes among instances
  - Partition the data based the values of the selected feature (with each branch representing one partitions)
- Recursion stops when:
  - instances in node belong to the same class (or if too few instances remain)
  - There are no remaining attributes on which to split

DT Construction Algorithm (ID3)

- Decision Tree Learning Method (ID3)
  - **Input**: a set of training instances ***S***, a set of features ***F***
  - 1. If every element of ***S*** has a class value "yes", return "yes"; if every element of ***S*** has class value "no", return "no"
  - 2. Otherwise, choose the **best feature** ***f*** from ***F*** (if there are no features remaining, then return failure);
  - 3. Extend tree from ***f*** by adding a new branch for each attribute value of ***f***
    - 3.1. Set ***F'*** = ***F*** – {***f***},
  - 4. Distribute training instances to leaf nodes (so each leaf node ***n*** represents the subset of examples S_n of ***S*** with the corresponding attribute value
  - 5. Repeat steps 1-5 for each leaf node ***n*** with S_n as the new set of training instances and ***F'*** as the new set of attributes
- Main Question:
  - how do we choose the best feature at each step?

Note: ID3 algorithm only deals with categorical attributes, but can be extended (as in C4.5) to handle continuous attributes

Choosing the "Best" Feature

- Use *Information Gain* to find the "best" (most discriminating) feature
- Assume there are two classes, *P* and *N* (e.g., P = "yes" and N = "no")
  - **Let the set of instances *S* (training data) contains *p* elements of class *P* and *n* elements of class *N***
  - **The amount of information, needed to decide if an arbitrary example in *S* belongs to *P* or *N* is defined in terms of entropy, *I*(*p*,*n*):**

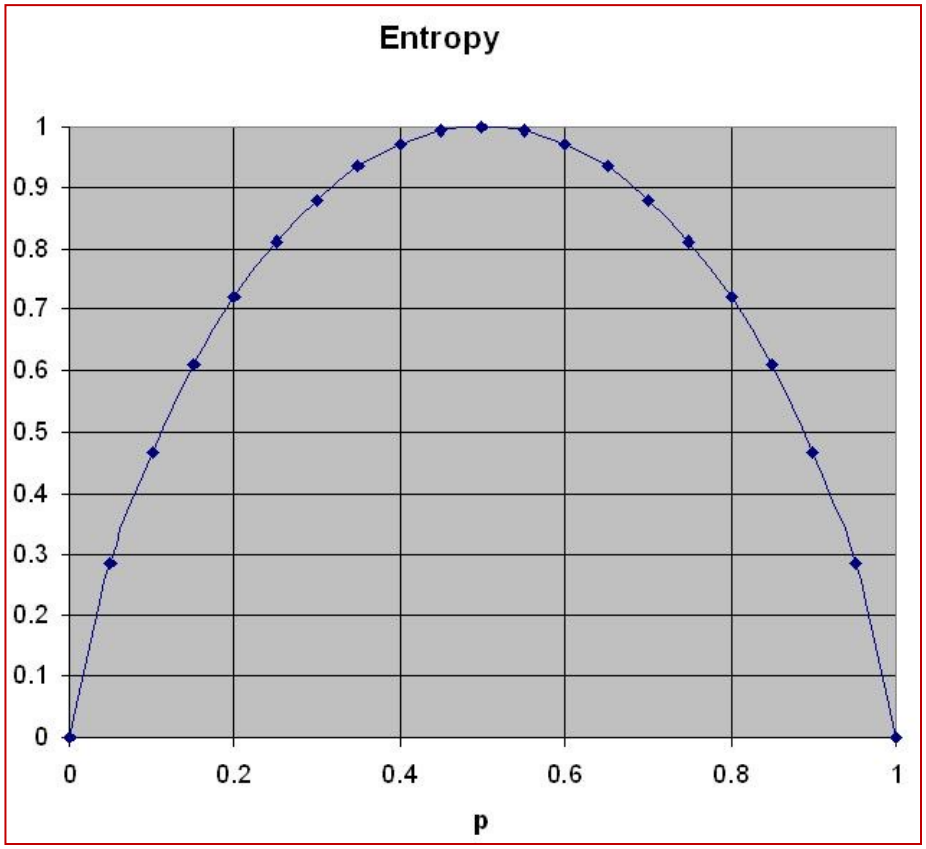
$$I(p,n) = -\Pr(P)\log_2 \Pr(P) - \Pr(N)\log_2 \Pr(N)$$

  - **Note that Pr(*P*) = *p* / (*p*+*n*) and Pr(*N*) = *n* / (*p*+*n*)**

- In other words, entropy of a set on instances *S* is a function of the probability distribution of classes among the instances in *S*.

Entropy

Entropy for two class problems



Entropy with Multi-Class Problems

- More generally, if we have *m* classes, $c_1, c_2, \ldots, c_m$, with $s_1, s_2, \ldots, s_m$ as the numbers of instances from *S* in each class, then the entropy is:

$$I(s_1, s_2, \ldots, sn) = -\sum_{i=1}^{m} p_i \log_2 p_i$$

- where p_i is the probability that an arbitrary instance belongs to the class c_i.

Information Gain

- Now, assume that using attribute A a set *S* of instances will be partitioned into sets S_1, S_2 , …, S_v each corresponding to distinct values of attribute A.
  - **If S_i contains p_i cases of *P* and n_i cases of *N*, the entropy, or the expected information needed to classify objects in all subtrees S_i is**

$$E(A) = \sum_{i=1}^{v} \Pr(S_i) I(p_i, n_i)$$

where,

$$\Pr(S_i) = \frac{|S_i|}{|S|} = \frac{p_i + n_i}{p + n}$$

The probability that an arbitrary instance in S belongs to the partition S_i

- The encoding information that would be gained by branching on *A*:

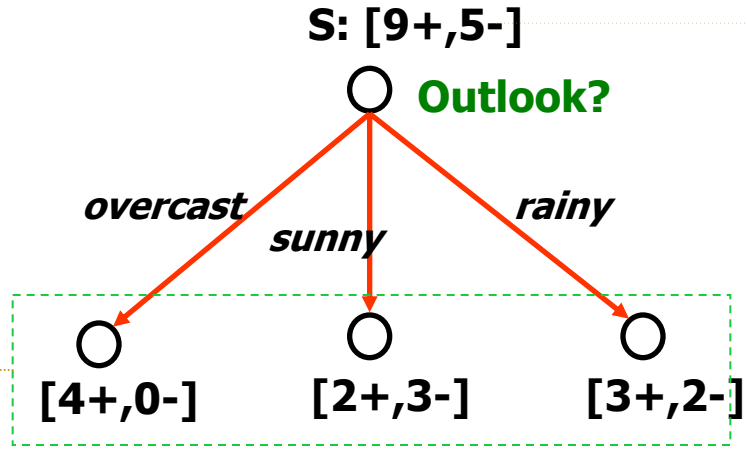
$$Gain(A) = I(p, n) - E(A)$$

- At any point we want to branch using an attribute that provides the highest information gain.

Attribute Selection Example

The "Golf" example: what attribute should we choose as the root?

Day	outlook	temp	humidity	wind	play
D1	sunny	hot	high	weak	*No*
D2	sunny	hot	high	strong	*No*
D3	overcast	hot	high	weak	*Yes*
D4	rain	mild	high	weak	*Yes*
D5	rain	cool	normal	weak	*Yes*
D6	rain	cool	normal	strong	*No*
D7	overcast	cool	normal	strong	*Yes*
D8	sunny	mild	high	weak	*No*
D9	sunny	cool	normal	weak	*Yes*
D10	rain	mild	normal	weak	*Yes*
D11	sunny	mild	normal	strong	*Yes*
D12	overcast	mild	high	strong	*Yes*
D13	overcast	hot	normal	weak	*Yes*
D14	rain	mild	high	strong	*No*



I(9,5) = -(9/14).log(9/14) - (5/14).log(5/14)
= 0.94

I(4,0) = -(4/4).log(4/4) - (0/4).log(0/4)
= 0

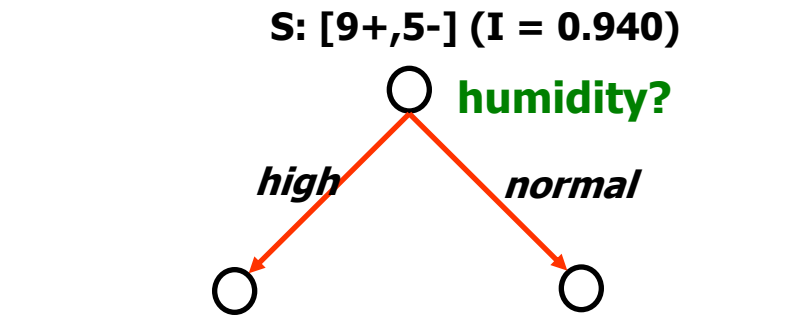
I(2,3) = -(2/5).log(2/5) - (3/5).log(3/5)
= 0.97

I(3,2) = -(3/5).log(3/5) - (2/5).log(2/5)
= 0.97

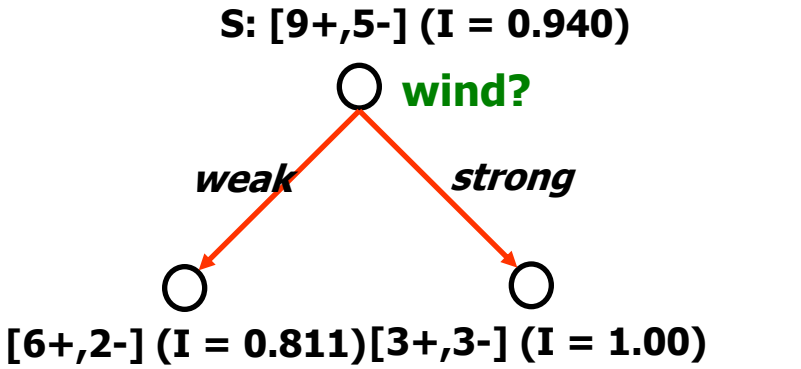
Gain(outlook) = .94 - (4/14)*0
- (5/14)*.97
- (5/14)*.97
= .24

Attribute Selection Example (cont.)

Day	outlook	temp	humidity	wind	play
D1	sunny	hot	high	weak	*No*
D2	sunny	hot	high	strong	*No*
D3	overcast	hot	high	weak	*Yes*
D4	rain	mild	high	weak	*Yes*
D5	rain	cool	normal	weak	*Yes*
D6	rain	cool	normal	strong	*No*
D7	overcast	cool	normal	strong	*Yes*
D8	sunny	mild	high	weak	*No*
D9	sunny	cool	normal	weak	*Yes*
D10	rain	mild	normal	weak	*Yes*
D11	sunny	mild	normal	strong	*Yes*
D12	overcast	mild	high	strong	*Yes*
D13	overcast	hot	normal	weak	*Yes*
D14	rain	mild	high	strong	*No*



Gain(humidity) = .940 - (7/14)*.985 - (7/14)*.592 = .151

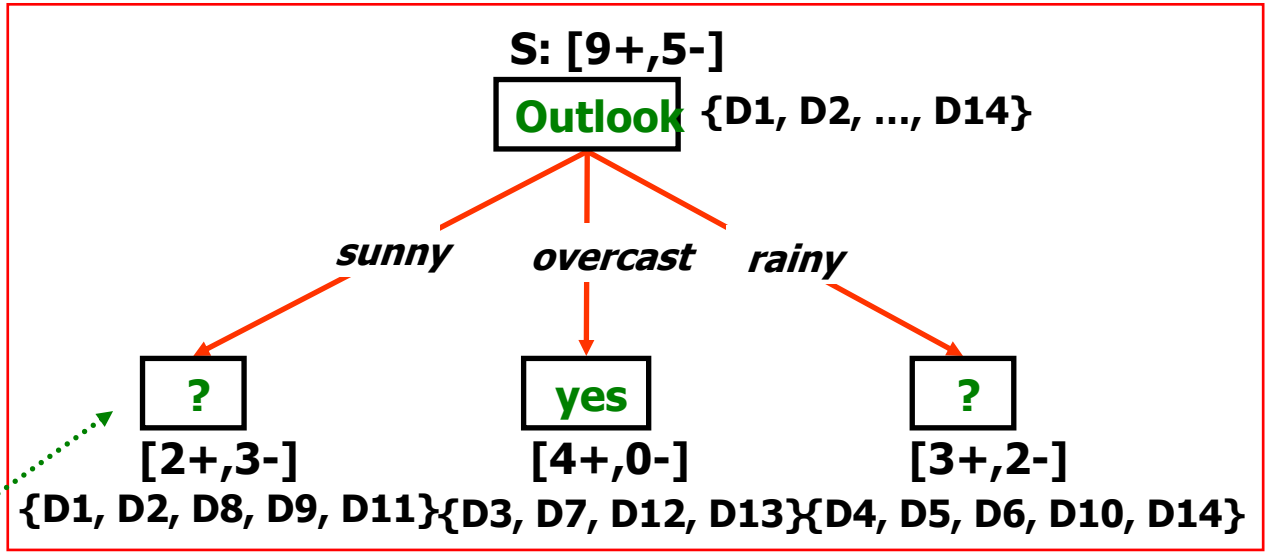


Gain(wind) = .940 - (8/14)*.811 - (8/14)*1.0 = .048

So, classifying examples by humidity provides more information gain than by wind. Similarly, we must find the information gain for "temp". In this case, however, you can verify that outlook has largest information gain, so it'll be selected as root

Attribute Selection Example (cont.)

- Partially learned decision tree



- which attribute should be tested here?

S_{sunny} = {D1, D2, D8, D9, D11}

Gain(S_{sunny}, humidity) = .970 - (3/5)*0.0 - (2/5)*0.0 = .970

Gain(S_{sunny}, temp) = .970 - (2/5)*0.0 - (2/5)*1.0 - (1/5)*0.0 = .570

Gain(S_{sunny}, wind) = .970 - (2/5)*1.0 - (3/5)*.918 = .019

Other Attribute Selection Measures

- **Gain ratio**:
  - Information Gain measure tends to be biased in favor attributes with a large number of values
  - Gain Ratio normalizes the Information Gain with respect to the total entropy of all splits based on values of an attribute
  - Used by C4.5 (the successor of ID3)
  - But tends to prefer unbalanced splits (one partition much smaller than others)
- **Gini index**:
  - A measure of ***impurity*** (based on relative frequencies of classes in a set of instances)
    - The attribute that provides the smallest Gini index (or the largest reduction in impurity due to the split) is chosen to split the node
  - Possible Problems:
    - Biased towards multivalued attributes; similar to Info. Gain.
    - Has difficulty when # of classes is large

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Some splits or leaf nodes may be the result of decision based on very few instances, resulting in poor accuracy for unseen instances
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the error rate going above a pre-specified threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree
    - Get a sequence of progressively pruned trees
    - Use a test data different from the training data to measure error rates
    - Select the "best pruned tree"

Enhancement to Basic DT Approach

- Allow for **continuous-valued attributes**
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
  - A leaf node is marked with a real value or a linear function: e.g., the mean of the target values of the examples at the node
- Handle **missing attribute values**
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- **Attribute construction**
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

DTs Pros

- is relatively fast, even with large data sets (10^6) and many attributes (10^3)
  - advantage of recursive partitioning: only process all cases at root
- Small-medium size trees usually intelligible
- Can be converted to rules
- Does feature selection ➔ tells you which features were informative
- Often yields compact models (Occam's Razor)
- Decision tree representation is understandable!

DTs Cons

- **Overfitting** – Trees will often need tuning and pruning to avoid overfitting, they are low bias and high variance algorithm
  - This is why we tune trees
  - This is why we use tress in ensemble models: Random Forest (covered later in the course)
- Large or complex trees can be just as unintelligible as other models
- If model depends on summing contribution of many different attributes, DTs probably won't do well
  - Pre-processing has to create combined features
- DTs that look very different can be same/similar
- Usually poor for predicting continuous values (regression)

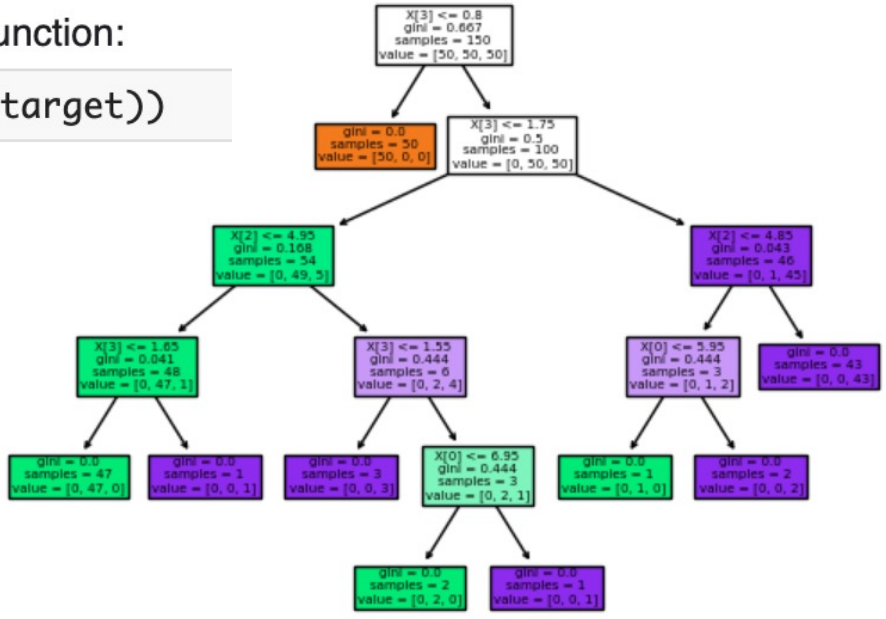
DTs in Scikit-Learn

Using the Iris dataset, we can construct a tree as follows:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(iris.data, iris.target)
```

Once trained, you can plot the tree with the plot_tree function:

```
>>> tree.plot_tree(clf.fit(iris.data, iris.target))
```



DTs in Scikit-Learn

- Input:
  - min_samples_leaf : int, float, optional (default=1) The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
  - …
- Output
  - feature_importance_
  - …