Decision Trees

Ryan Henning
(customized by F. Burkholder)

galvanize

- Decision Trees
- Entropy
- Information Gain
- Recursion
- How to build a tree

Historical log of times I played tennis:

Temp	Outlook	Humidity	Windy	Played
Hot	Sunny	High	False	No
Hot	Sunny	High	True	No
Hot	Overcast	High	False	Yes
Cool	Rain	Normal	False	Yes
Cool	Overcast	Normal	True	Yes
Mild	Sunny	High	False	No
Cool	Sunny	Normal	False	Yes
Mild	Rain	Normal	False	Yes
Mild	Sunny	Normal	True	Yes
Mild	Overcast	High	True	Yes
Hot	Overcast	Normal	False	Yes
Mild	Rain	High	True	No
Cool	Rain	Normal	True	No
Mild	Rain	High	False	Yes

```python
def will_play(temp, outlook, humidity,\
              windy):

    if outlook == 'sunny':
        if humidity == 'normal':
            return True
        else: # humidity == 'high'
            return False

    elif outlook == 'overcast':
        return True

    else: # outlook == 'rain'
        if windy == True:
            return False
        else: # windy == False:
            return True
```

You must like working hard all the time to code like this.

galvanize

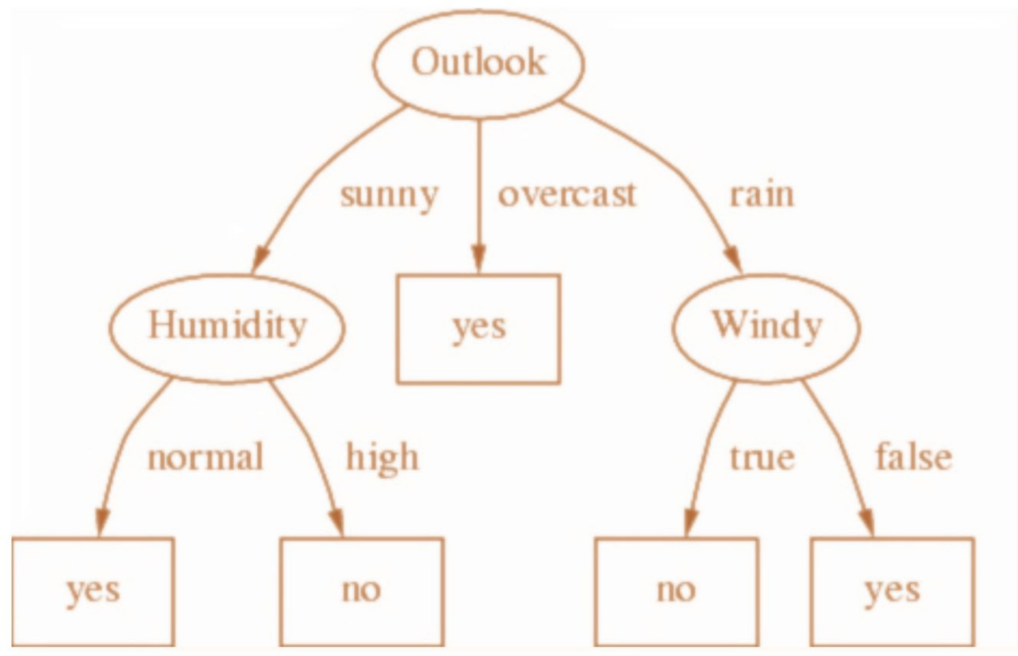
```
def will_play(temp, outlook, humidity,\
              windy):

    if outlook == 'sunny':
        if humidity == 'normal':
            return True
        else: # humidity == 'high'
            return False

    elif outlook == 'overcast':
        return True

    else: # outlook == 'rain'
        if windy == True:
            return False
        else: # windy == False:
            return True
```

Instead, let's write an algorithm to build a **Decision Tree** for us, based on the training data we have.



Decision Tree Terminology

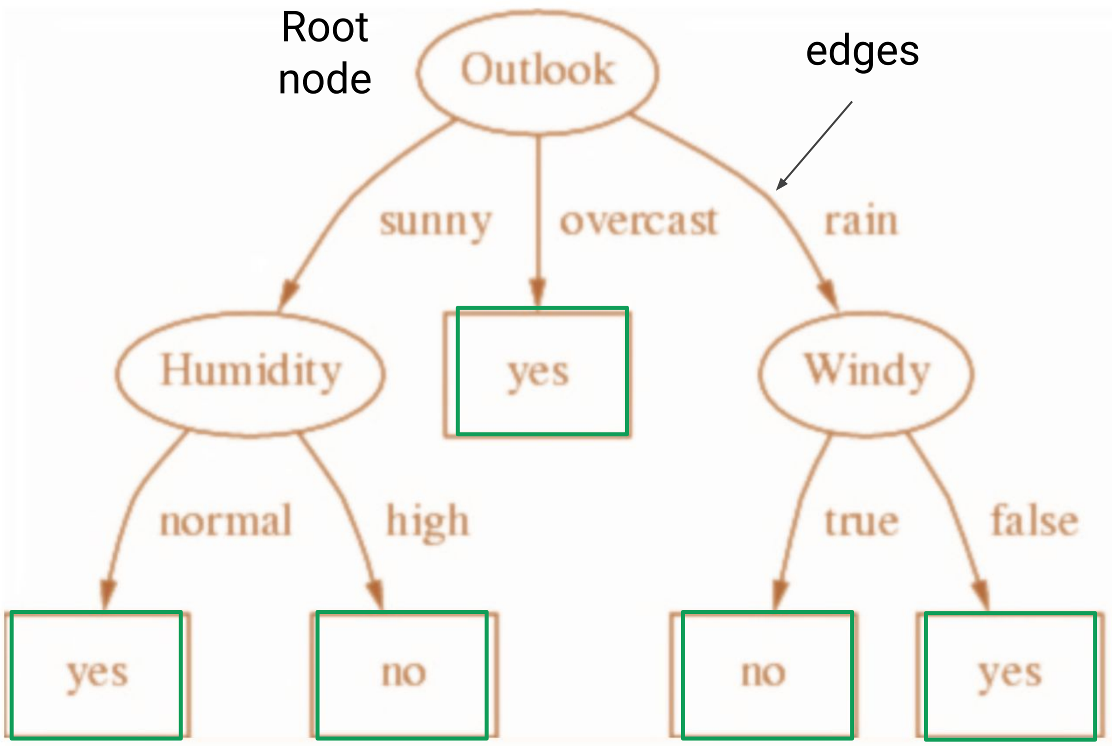
Where-ever there is data is called a "node"

Root node

edges

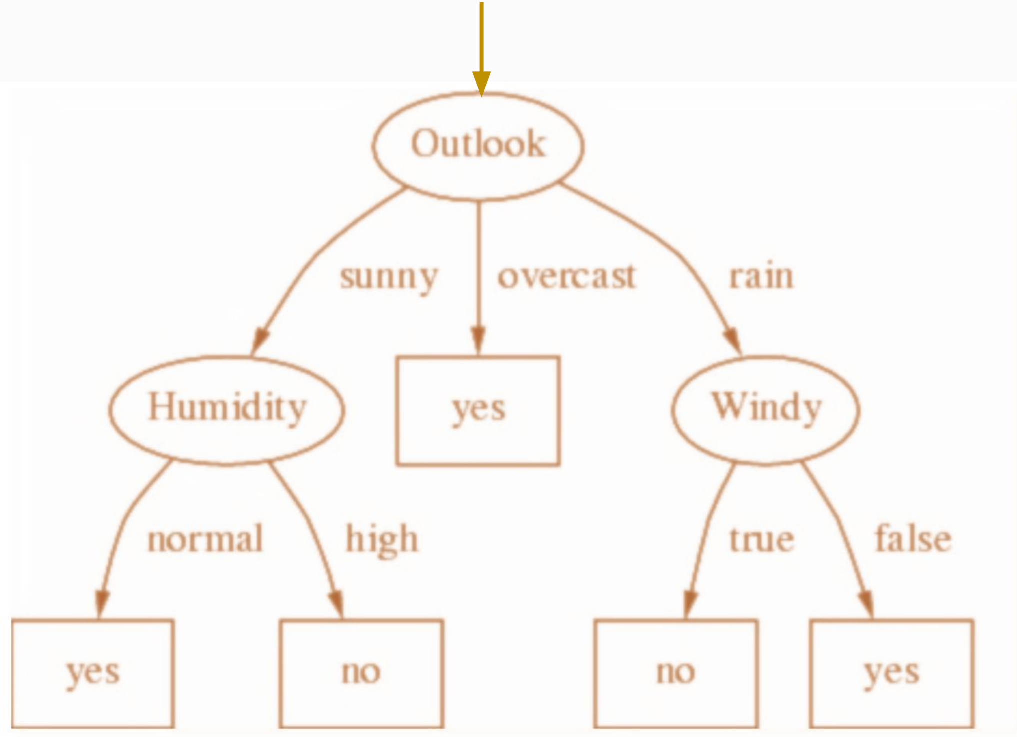
Edges connect nodes

Terminal node a.k.a "leaf"



Benefits of a decision tree:

Will I play tennis?

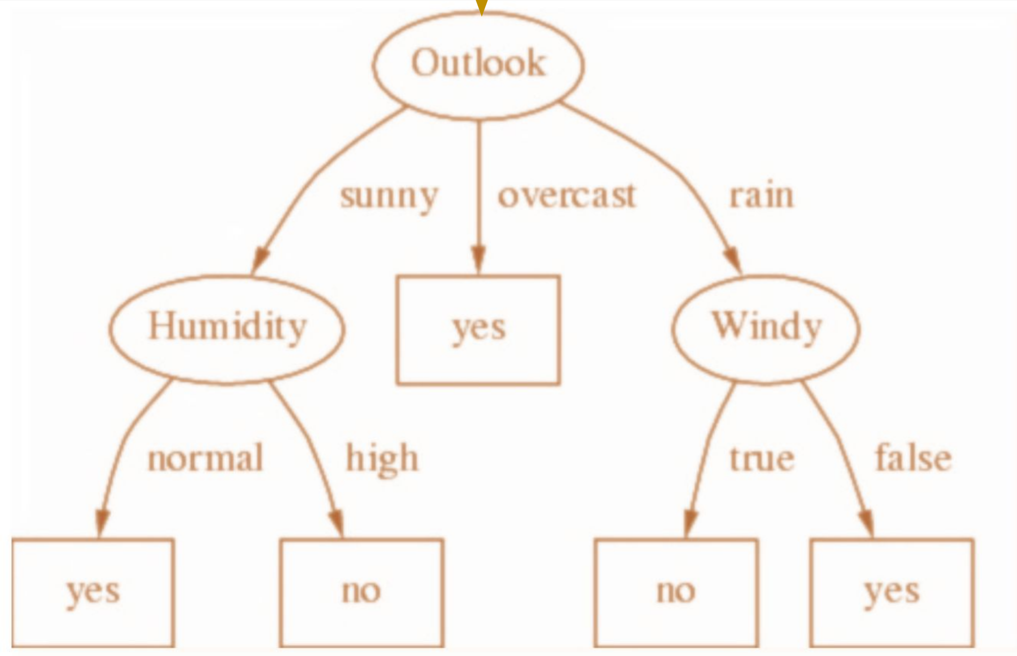


Benefits:

- non-parametric, non-linear
- can be used for classification and for regression
- real and/or categorical features*
- easy to interpret
- computationally cheap prediction
- handles missing values and outliers*
- can handle irrelevant features

*Caveats in sklearn

Will I play tennis?



Drawbacks:

- expensive to train
- greedy algorithm (local maxima)
- easily overfits
- right-angle decision boundaries only

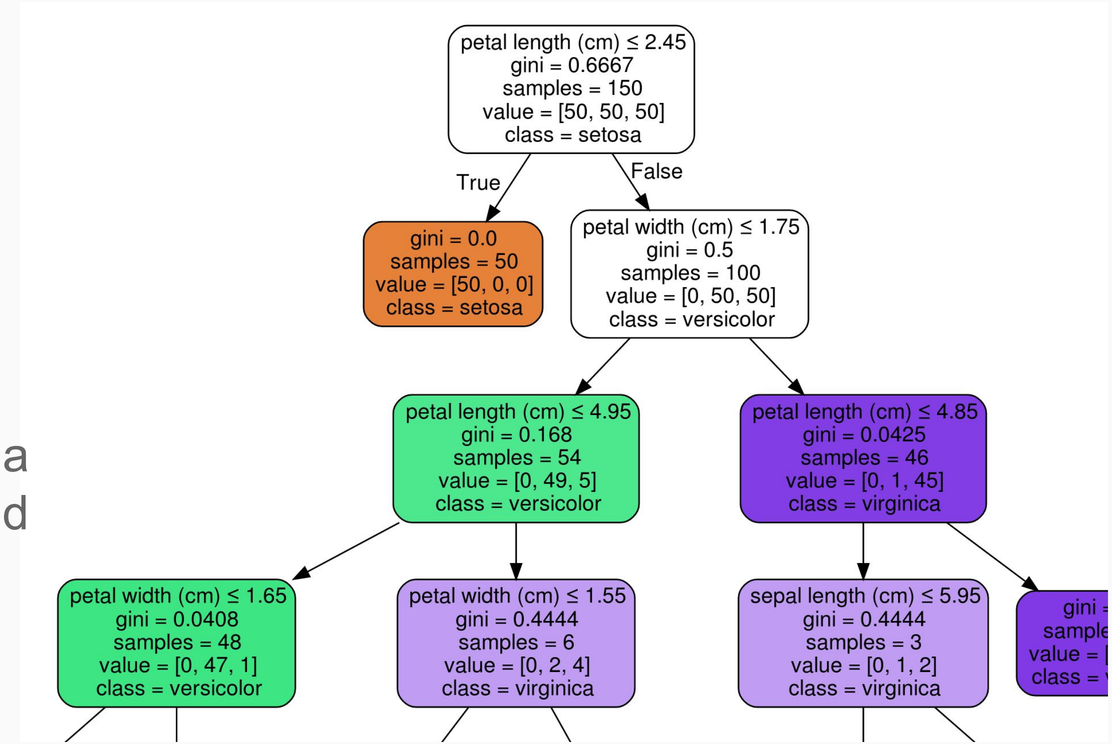
But how can we build one of these from training data?

A tree is a series of binary splits. How to pick the splits and build one?

Need two things:

1) A way to quantify how disordered a node is.
   Classification: Entropy or Gini
   Regression: RSS

2) A way to see how much disorder is reduced by making a split. How much information did we gain (how much disorder was reduced) by making that split?



$$H(X) = E[I(X)] = E[log_2(\frac{1}{P(X)})]$$

$$= -E[log_2(P(X))]$$

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

Shannon
Entropy

$$H(X) = E[I(X)] = E[log_2(\frac{1}{P(X)})]$$

$$= -E[log_2(P(X))]$$

Discrete
random
variable

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

Quantifying node disorder, classification - Entropy

galvanize

Shannon Entropy

information content of X

number of bits needed to encode each X event

$$H(X) = E[I(X)] = E[log_2(\frac{1}{P(X)})]$$

Discrete random variable

$$= -E[log_2(P(X))]$$

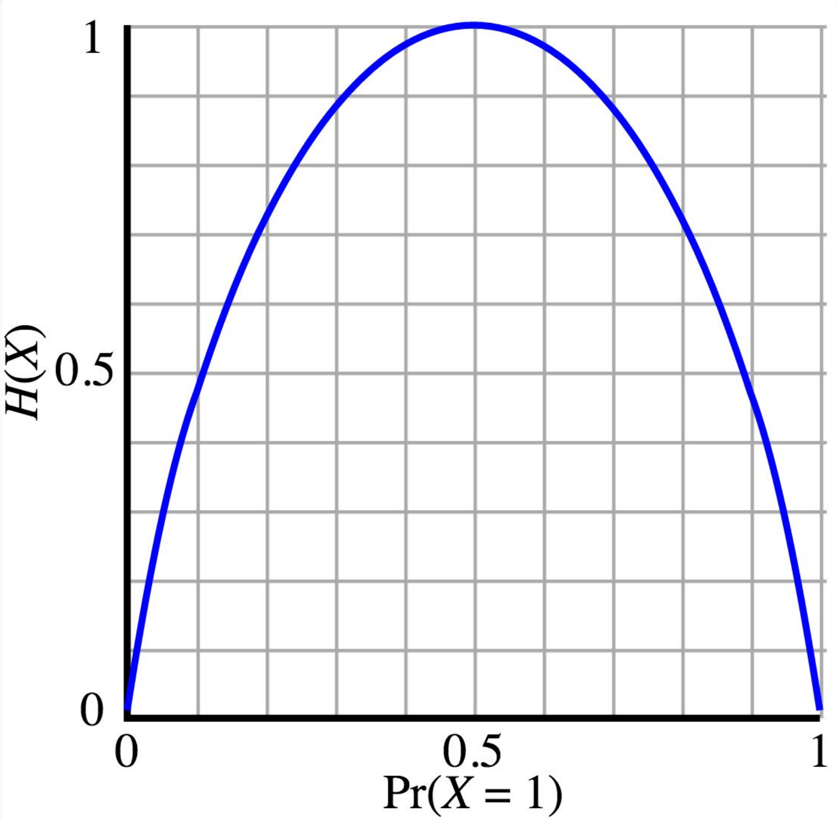
$$H(X) = -\sum_i p_i log_2(p_i)$$

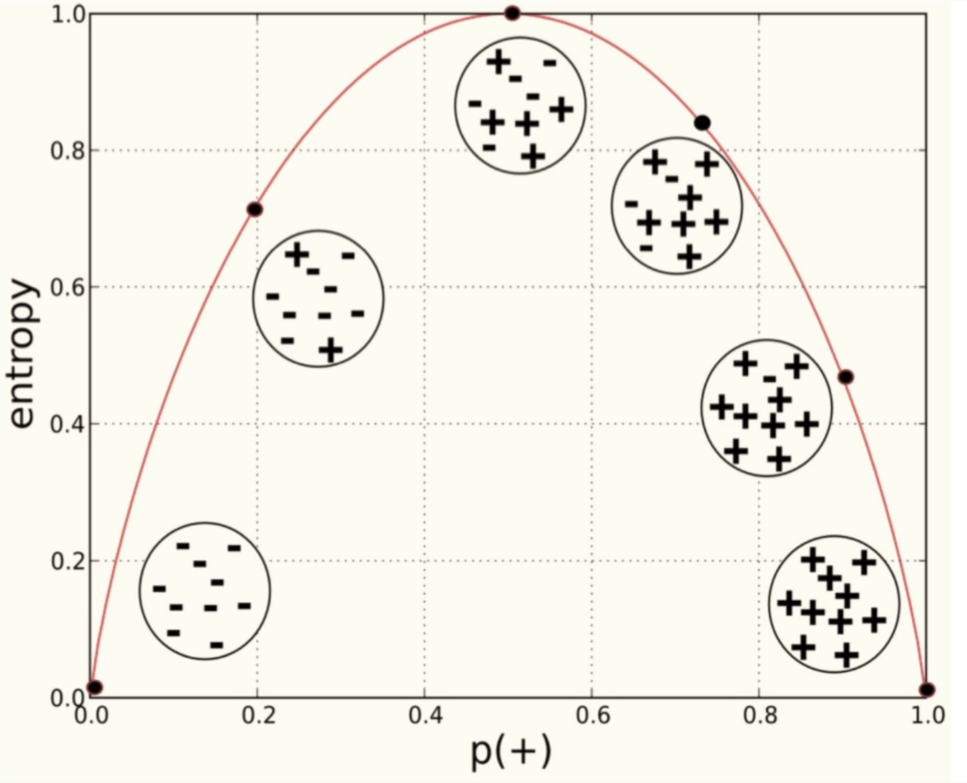
probability of each possible discrete outcome

Shannon Entropy

iterate over pmf

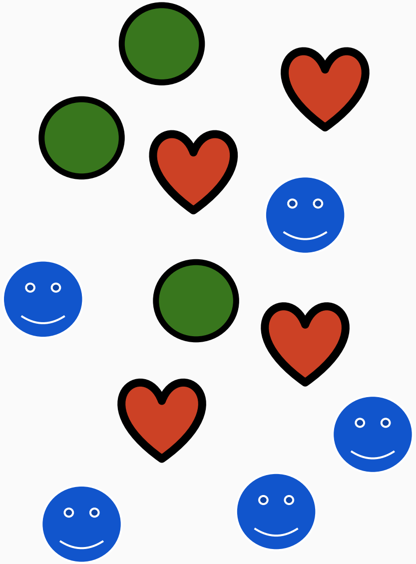
Entropy graph of a Bernoulli random variable X





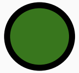
Shannon Entropy Diversity Index (aka, the Shannon Index)

We can measure the diversity of a set using Shannon Entropy (H) if we interpret the frequency of elements in the set as probabilities.



Estimate:

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

P() = 3/12 = 0.25

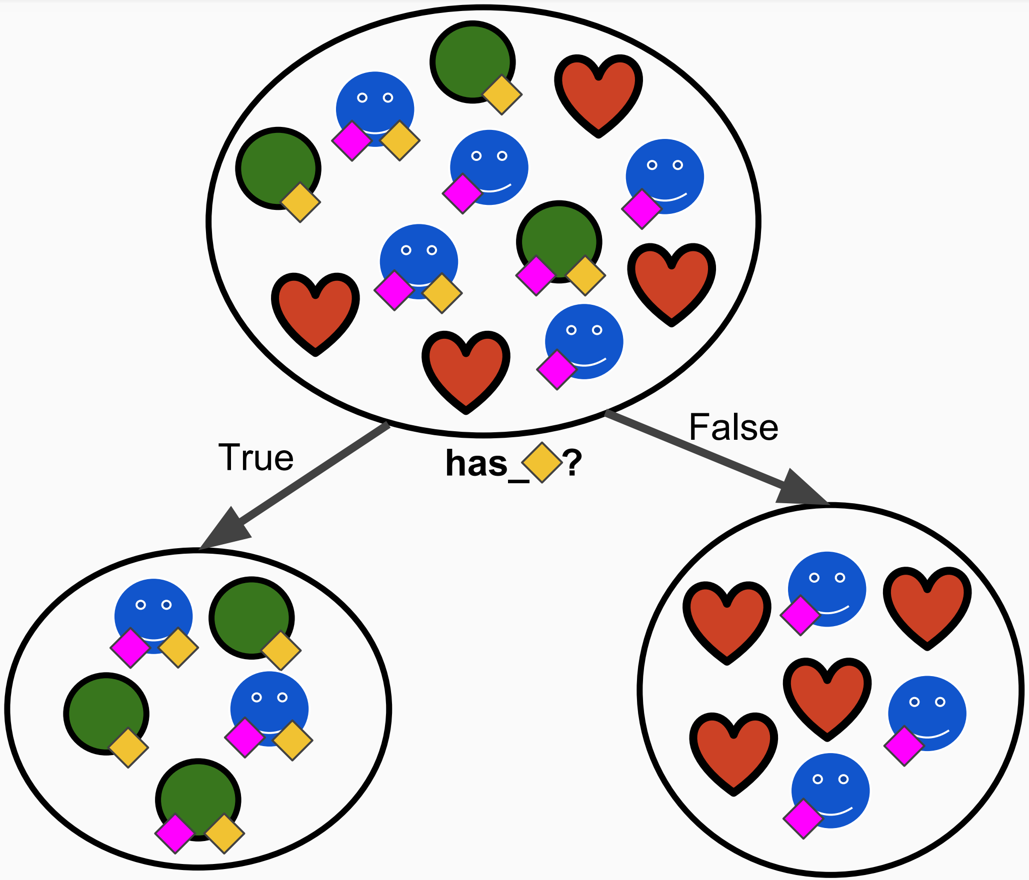
P() = 4/12 = 0.33

P() = 5/12 = 0.42

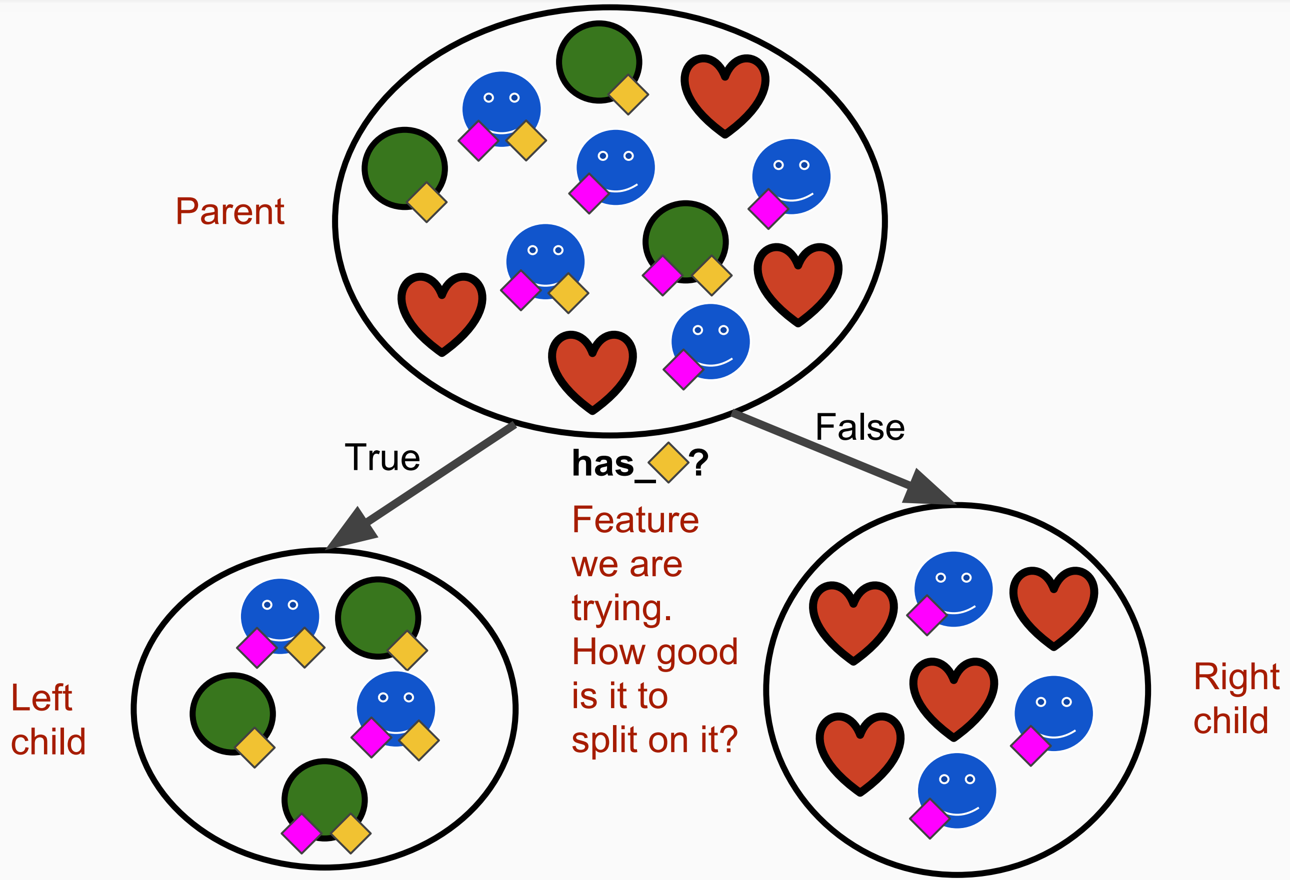
H = -0.25*$\log_2$(0.25) +
-0.33*$\log_2$(0.33) +
-0.42*$\log_2$(0.42)

H = 1.55

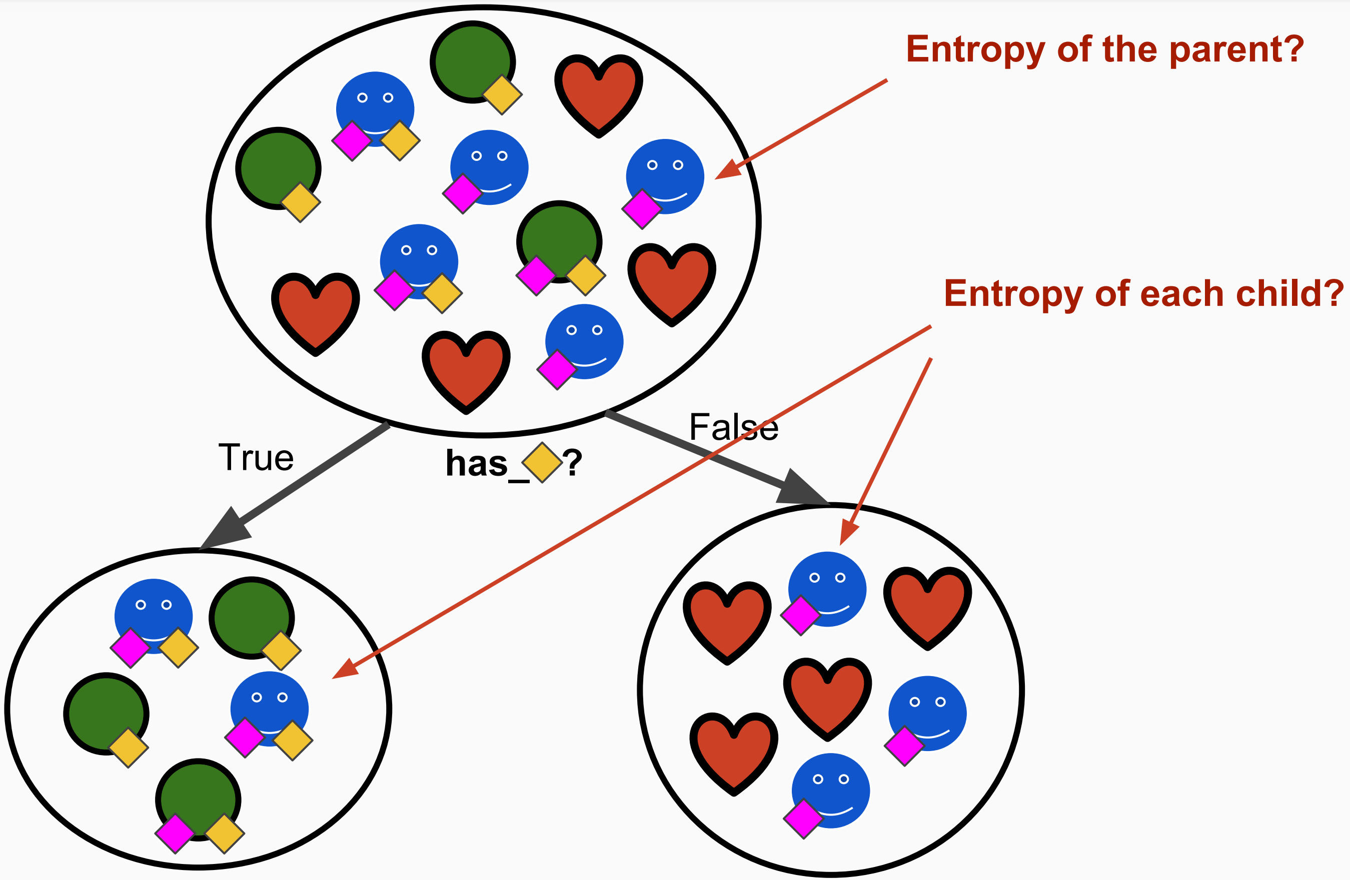
How much information is gained from making a split?



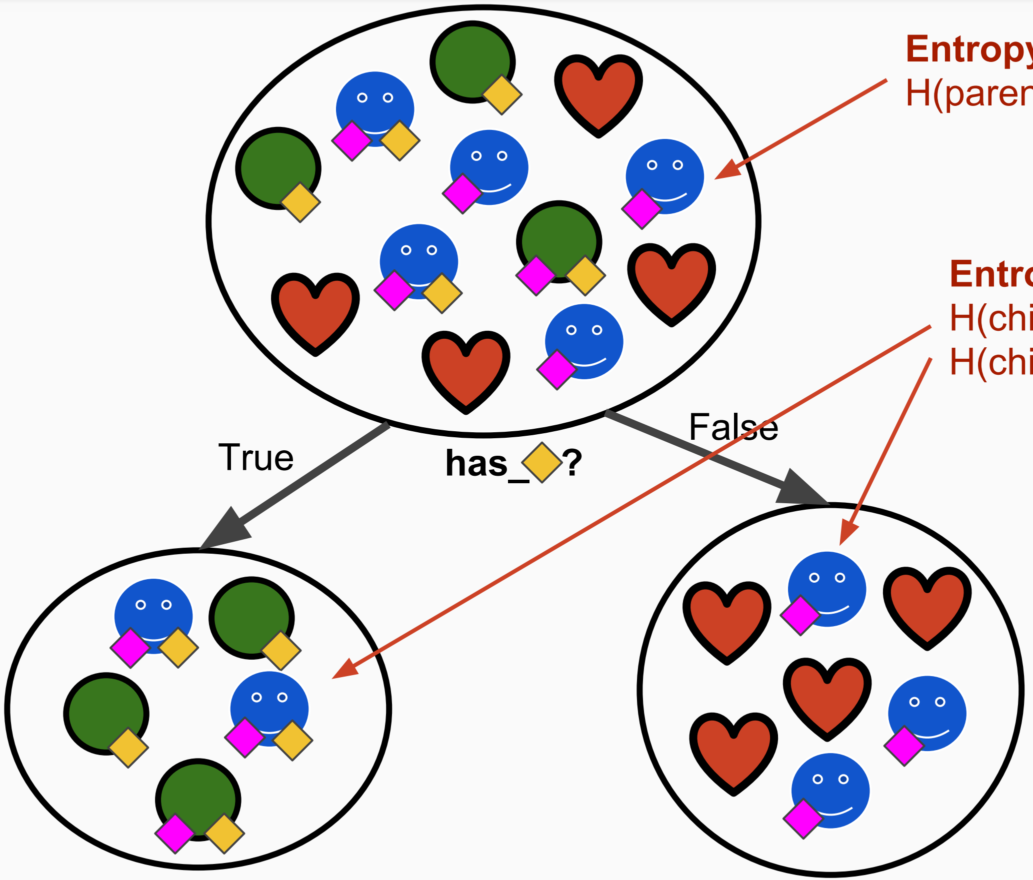
How much information is gained from making a split?



How much information is gained from making a split?



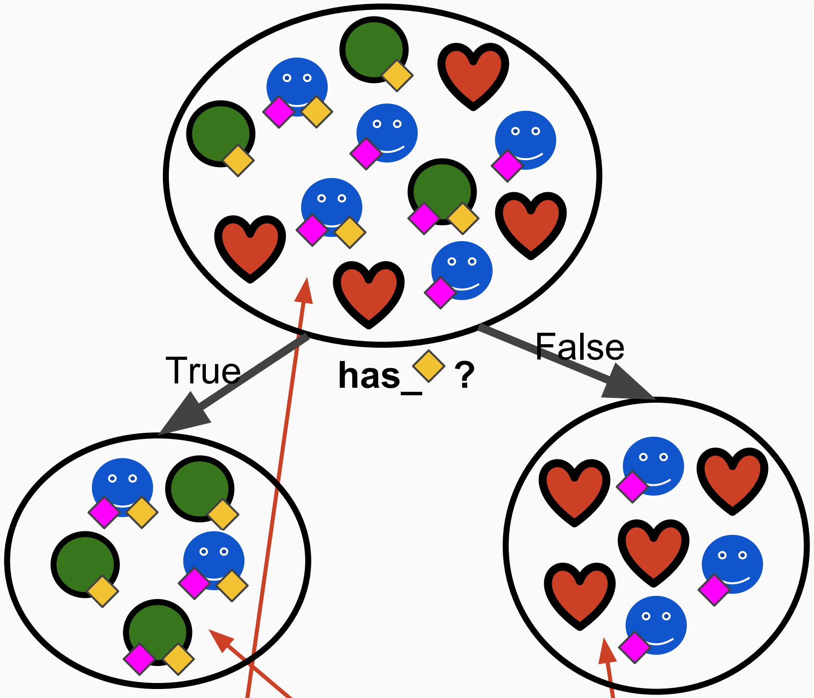
How much information is gained from making a split?



Entropy of the parent?
H(parent) = 1.55

Entropy of the children?
H(child_1) = 0.97
H(child_2) = 0.985

Information Gain (using Shannon Entropy Diversity Index)



Information gain from this split

the set of children

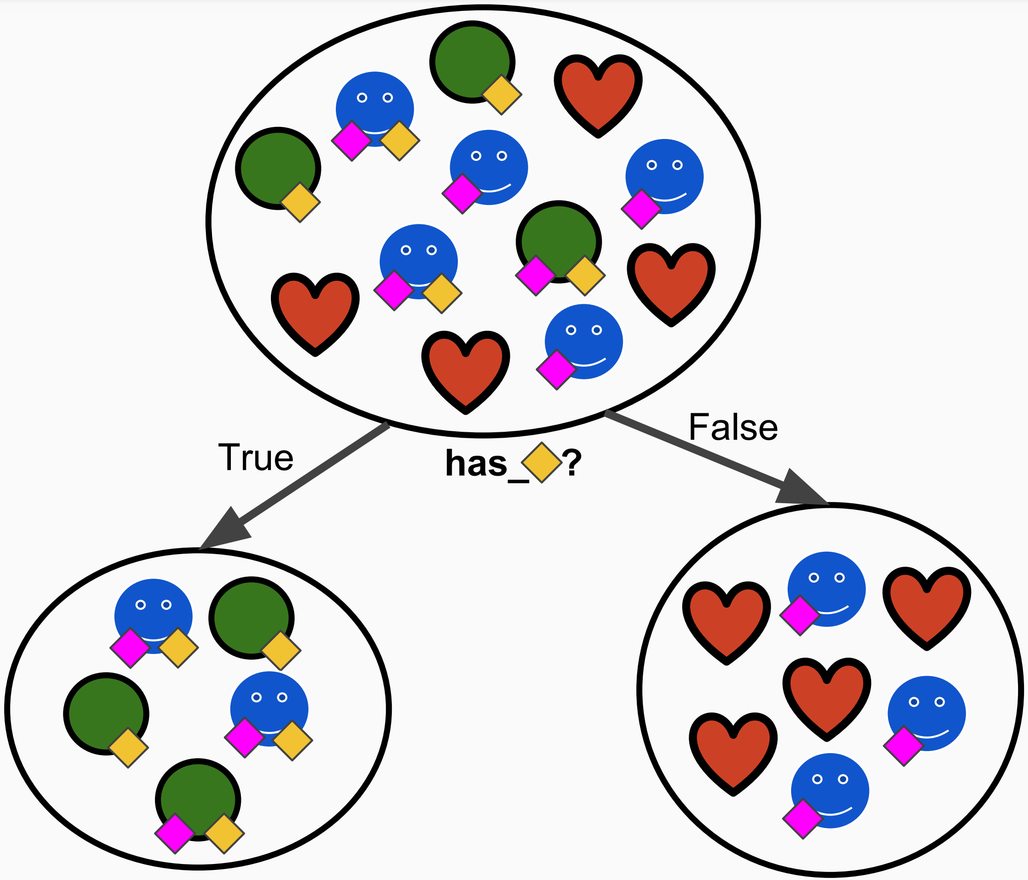
$$\mathrm{IG}(S, C) = H(S) - \sum_{C_i \in C} \frac{|C_i|}{|S|} H(C_i)$$

the parent's set of examples

the set of examples in each child

$$\mathrm{IG}(\text{parent}, \{\text{child\_1}, \text{child\_2}\}) = 1.55 - 5/12 * 0.97 - 7/12 * 0.985 = 0.57$$

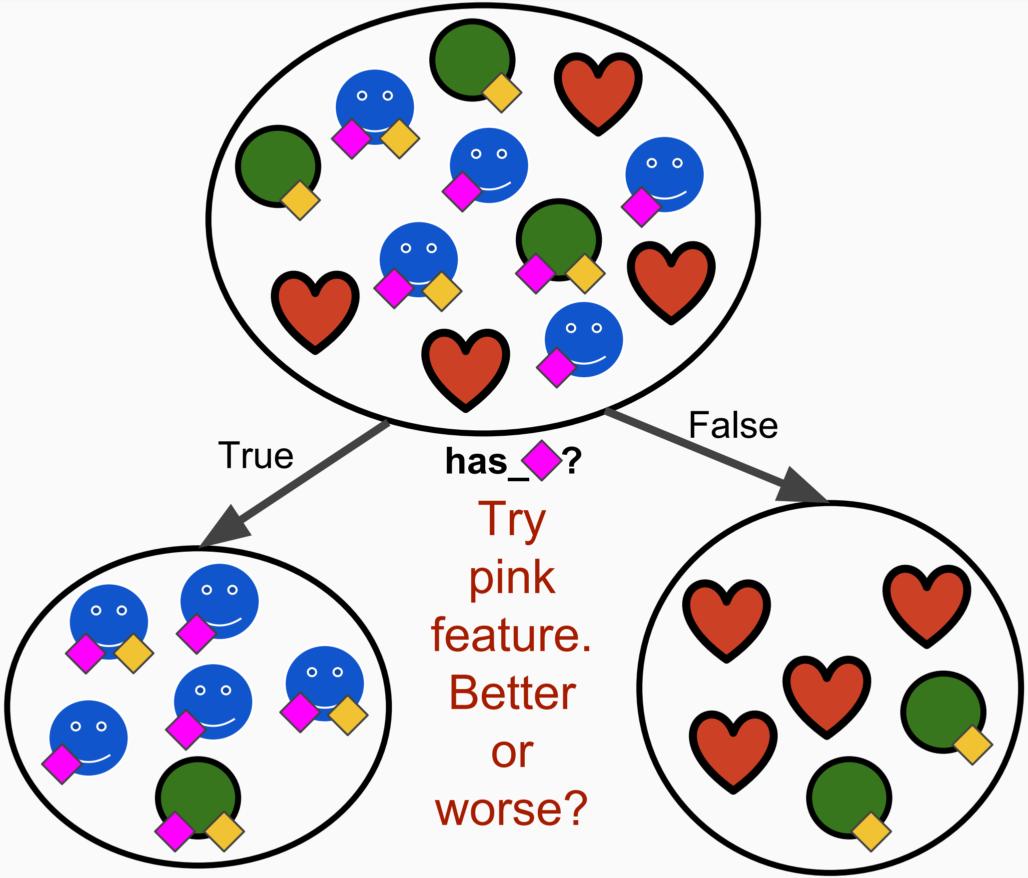
Information Gain



Information Gain = 0.57

...for splitting on yellow feature.

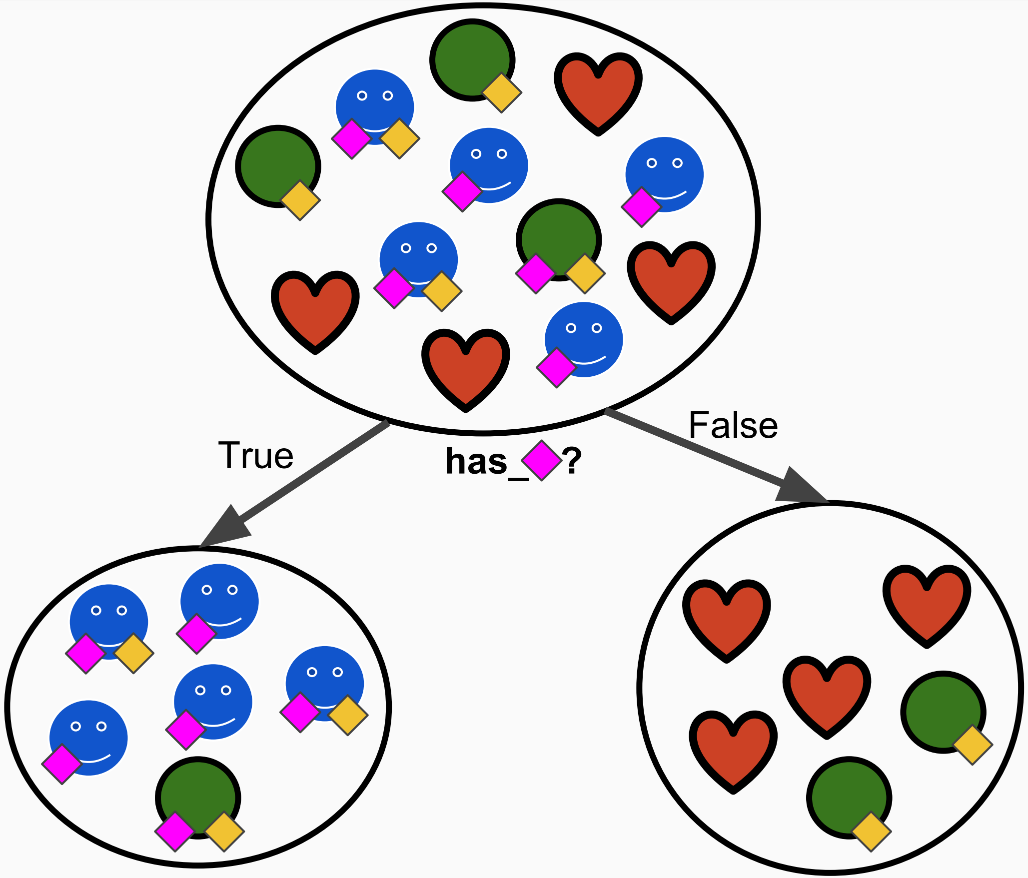
Try other features - which one has the highest information gain?



Information Gain = ??

Information Gain

galvanize



Information Gain = 0.765

Better! In this case we would choose to split on the pink feature (higher information gain)

Splitting Algorithm:

Possible Splits:

Consider all binary splits based on a single feature:

- if the feature is categorical, split on value or not value.
- if the feature is numeric, split at a threshold: >threshold or <=threshold

Splitting Algorithm:

1. Calculate the information gain for all possible splits.

2. Commit to the split that has the highest information gain.

Recursion

What is this function?

$$f(x) = \prod_{i=1}^{x} i$$

Is this an equivalent function?

$$f(x) = \begin{cases} 1, & \text{if } x \leq 1 \\ x f(x-1), & \text{otherwise} \end{cases}$$

```
def f(x):
    '''
    This function returns x!.
    >>> f(5)
    120
    '''
    if x <= 1:
        return 1
    else:
        return x * f(x-1)

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

How to build a decision tree (pseudocode):

```
function BuildTree:
    If every item in the dataset is in the same class
    or there is no feature left to split the data:
        return a leaf node with the class label
    Else:
        find the best feature and value to split the data
        split the dataset
        create a node
        for each split
            call BuildTree and add the result as a child of the node
        return node
```

The Gini Index

A measure of impurity: the probability of a misclassification if a random sample drawn from the set is classified according to the distribution of classes in the set

Scikit-learn doesn't use *Shannon Entropy Diversity* by default. It uses the *Gini Index*:

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

Information gain using the *Gini Index*:

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

Regression Trees

Training

Targets are real values so can't use Gini or Entropy as basis for IG. But you can maximize the reduction in RSS.

Predict

Either predict the mean value of the leaf, or do linear regression within the leaf!

Pruning - preventing 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 (after a certain number of splits)
- **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
- Set the maximum number of leaf nodes (form of regularization - see pair.md for details)

Algorithm Names:

The details of training a decision tree vary… each specific algorithm has a name. Here are a few you'll often see:

- **ID3:** category features only, information gain, multi-way splits, ...
- **C4.5:** continuous and categorical features, information gain, missing data okay, pruning, ...
- **CART:** continuous and categorical features and targets, gini index, binary splits only, …
- Sklearn uses CART. See http://scikit-learn.org/stable/modules/tree.html#tree section 1.10.6

In sklearn:

- Gini is default, but you can often choose entropy (I frequently get same tree & splits)
- Prune with `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_leaf_nodes`
- Need to use one-hot-encoding for categorical features, e.g. ['Red', 'Green', 'Blue'] encoded as X_red = 1, X_green = 0, X_blue = 0 if feature is 'Red'. See **Feature Binarization** and **Encoding Categorical Features** at http://scikit-learn.org/stable/modules/preprocessing.html
- Does not support missing values (even though it's CART)