Decision Trees

Overview

Decision trees are a supervised, non-parametric learning method whose trained form is a series of sequential, binary “splits” on features with the goal of minimizing predictive error.

What distinguishes decision trees are the sequential splits on the features, where the information gained in the split determines which split should be made.

Walk through decision tree animation.

Objectives

By the end of this lecture you:

- Can apply decision tree to classification/regression problems
- Can apply different measures (Entropy, Gini, MSE, etc.) to quantify branching in decision trees
- Can discuss advantages and disadvantages of decision trees
- Can discuss methods used to combat overfitting decision trees
- Can apply recursive methods in your programming

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 - Continued

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

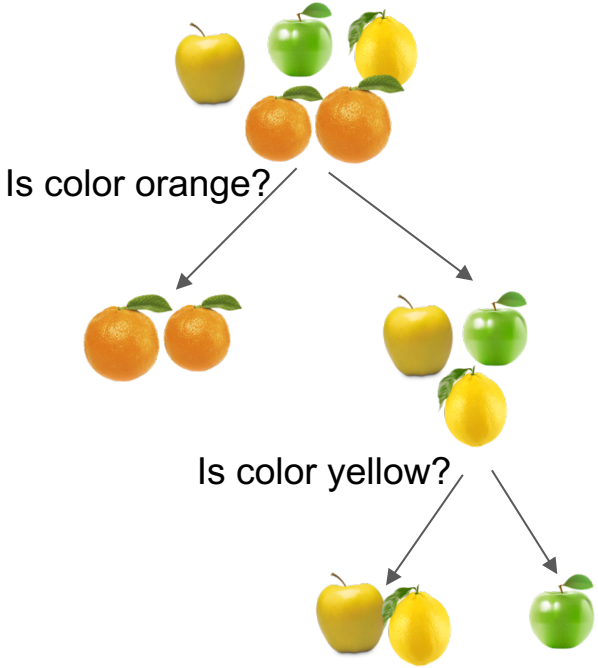
A decision tree is a (binary) tree, with each branch (non-leaf) node having a left and right child node. Each branch node specifies a feature (predictor, attributes) to split along with information on which values should pass to the left/right node.

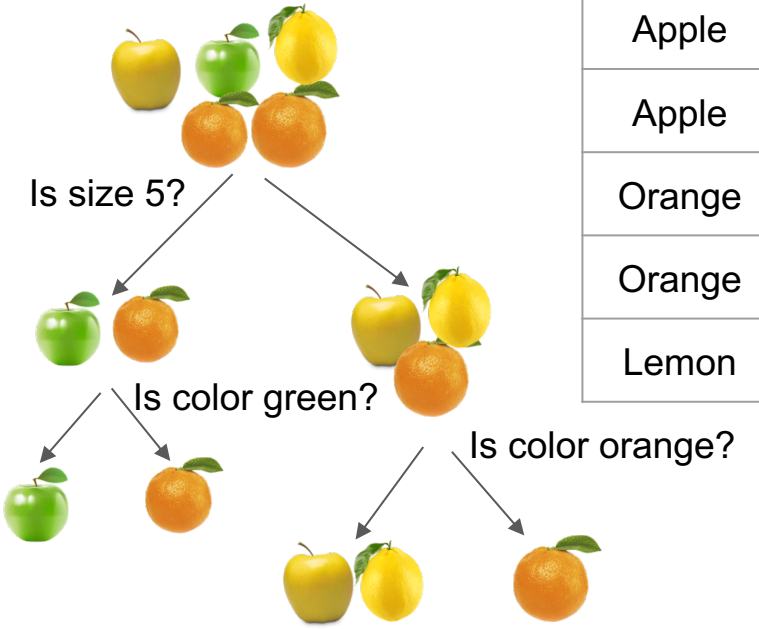
Decision tree :

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

Example - Continued

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 - Continued

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 checking the branch we can notice that at any step we reduce the amount of randomness and gain more information!

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 is identified correctly multiply by it is identified incorrectly, sum over all objects:

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

How does this help us to ask the right question?

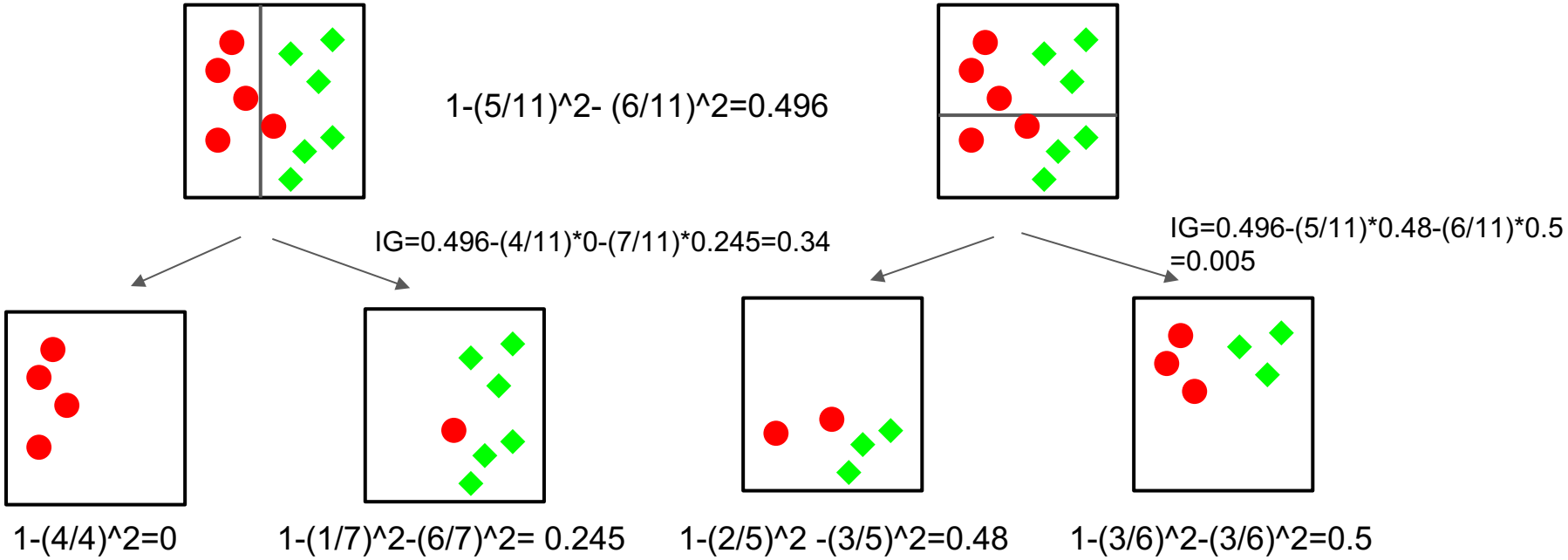
Or what is the right question to ask at every step?

The right question is a question that gives the minimum gini impurity (or gain the most information) at each step! The information gain is calculated by:

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

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

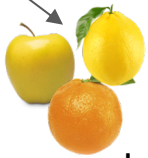
How Do We Find the Best Tree?

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



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

Is color green?

IG=0.5

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
- Commit to the decision which results in the largest information gain

Decision Tree - Pseudocode

Recursive way to build a tree using the Splitting Algorithm

Resort to a greedy heuristic:

1. Start from empty decision tree
2. Split on next best feature based on the splitting algorithm
3. Recurse

Different Types of Decision Trees

1. **Classification Trees**

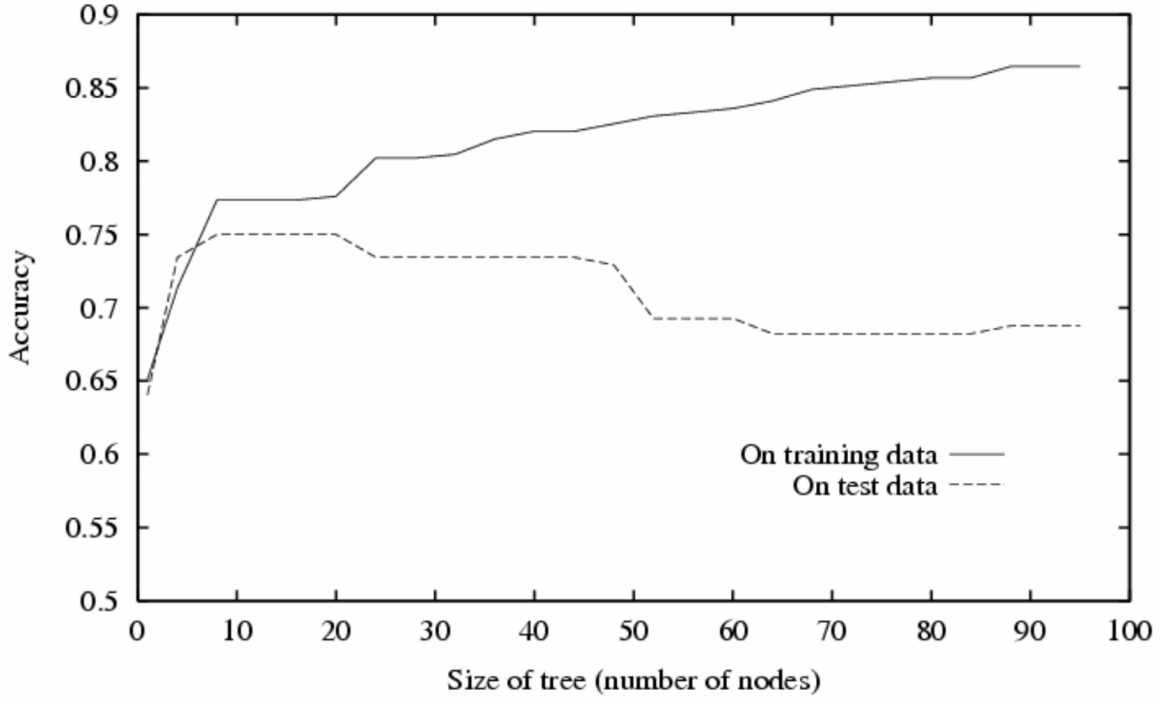
   In classification trees our outcomes (target, output) are discrete. Leaf values are typically set to the most common outcomes.

1. **Regression Trees**

   In regression trees our outcomes (target, output) are numerical. 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 categorical or numerical for both classification and regression trees.

Overfitting



Overfitting

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

Pre-pruning 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

Post-pruning ideas (prune after you've finished building the tree):

- **merge leaves:** if doing so decreases test-set error
- **number of leaf nodes:** stop splitting when number #nodes get too high

Pruning (pseudo-code)

Algorithm prune_tree(T,D)
Input : decision tree T; labelled data D
Output : pruned tree T^p

```
def prune_tree(T, D):
    for every internal node N of T (starting from the bottom):
            T^N = subtree of T rooted at N;
            D^N ={Data covered by N};
            if accuracy of T^N over D^N is worse than predicting majority class in D^N :
                replace T^N in T by a leaf labelled with the majority class in D^N
            return pruned version of T
```

Other Splitting Measures - Entropy

There are other measures beside gini impurity that quantify the randomness and gaining of information when we are branching.

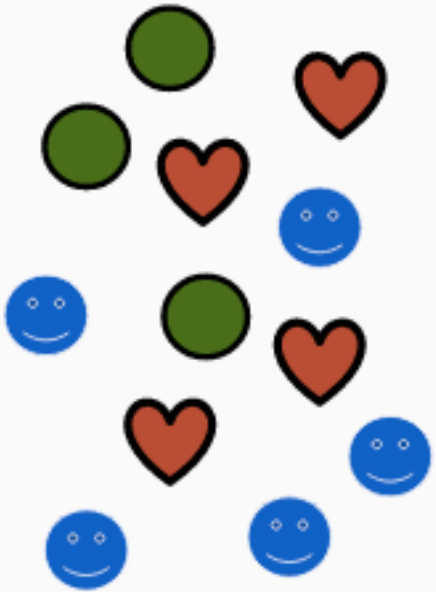
Entropy is another measure that quantify the randomness. If the probability of identifying the object *j* correctly is p_j and the probability of identifying it incorrectly is 1-p_j then the entropy is defined:

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

The information gain is calculated by:

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

Example 2 – Entropy



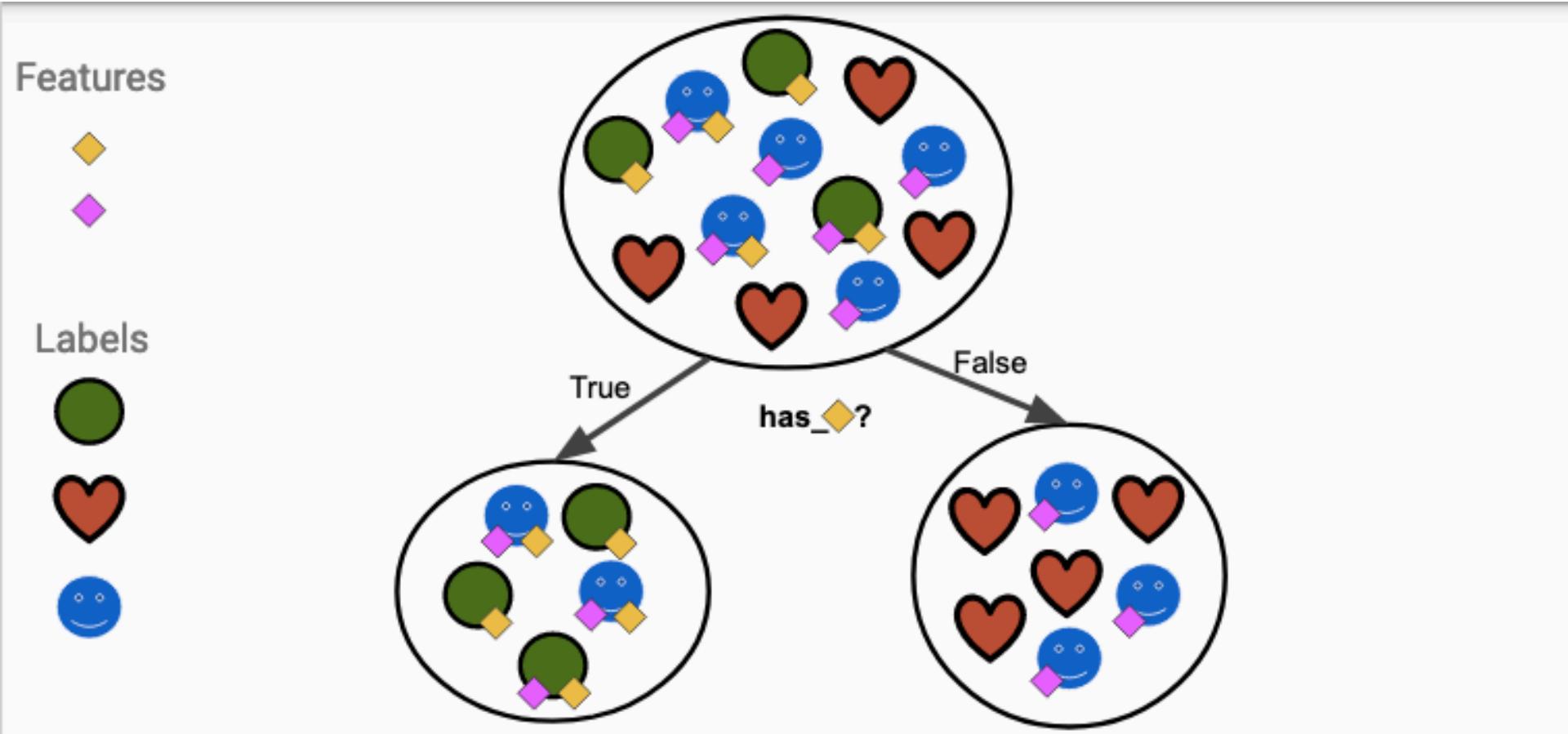
Estimate:

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

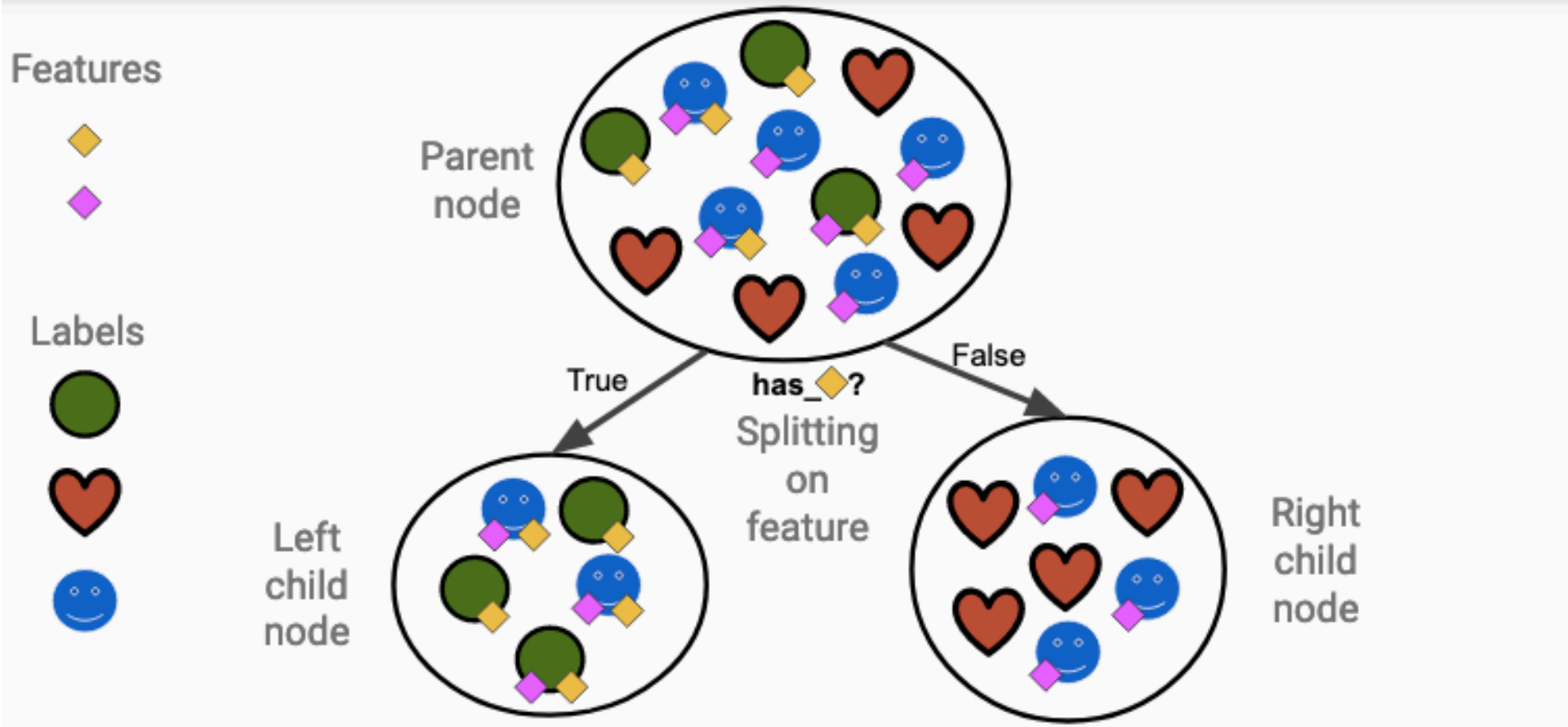
P( ) = 3/12 = 0.25
P( ) = 4/12 = 0.33
P( ) = 5/12 = 0.42

$H = -0.25*\log_2(0.25) +$
$-0.33*\log_2(0.33) +$
$-0.42*\log_2(0.42)$
$H = 1.55$

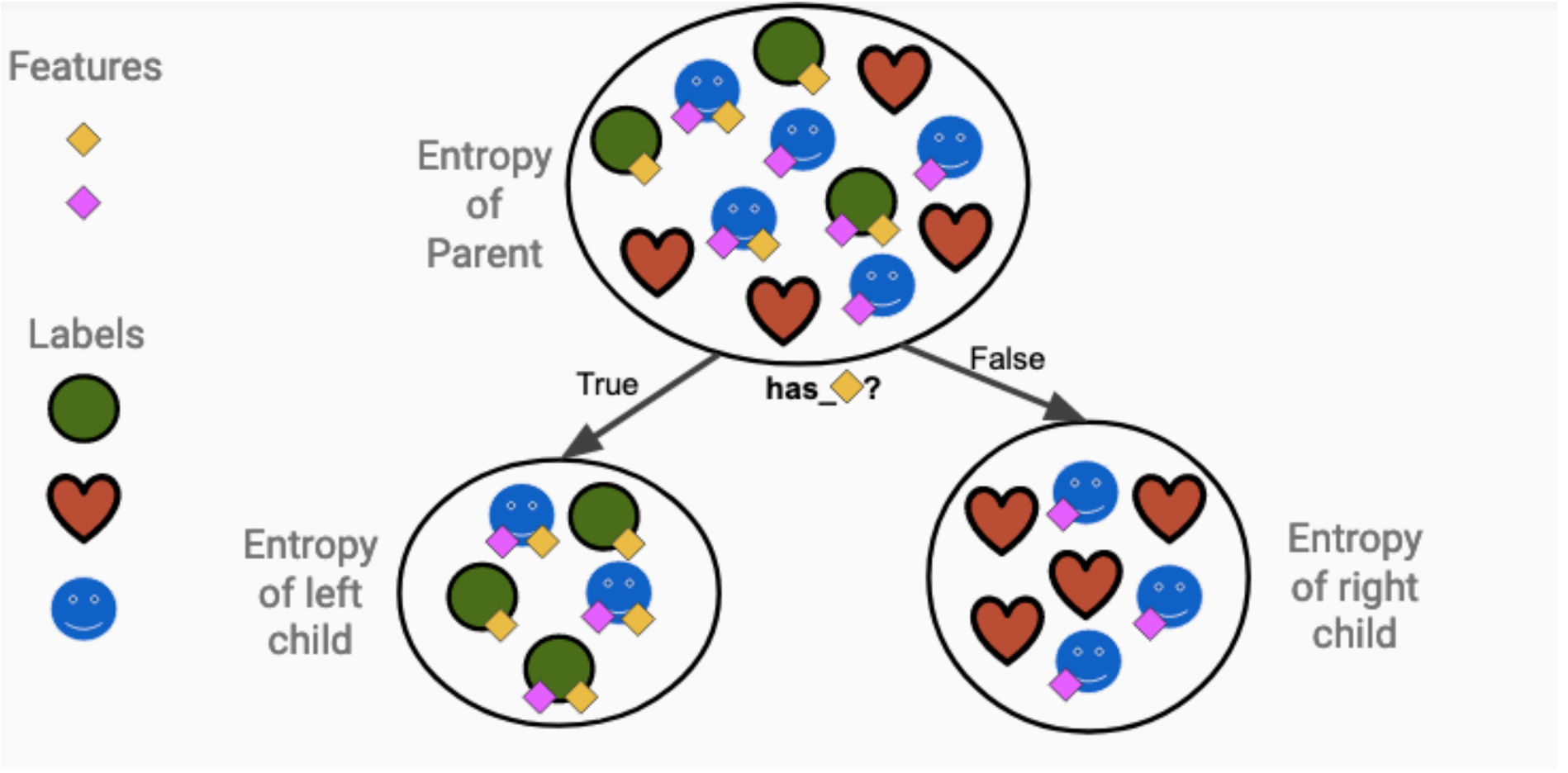
Example 2 Continued



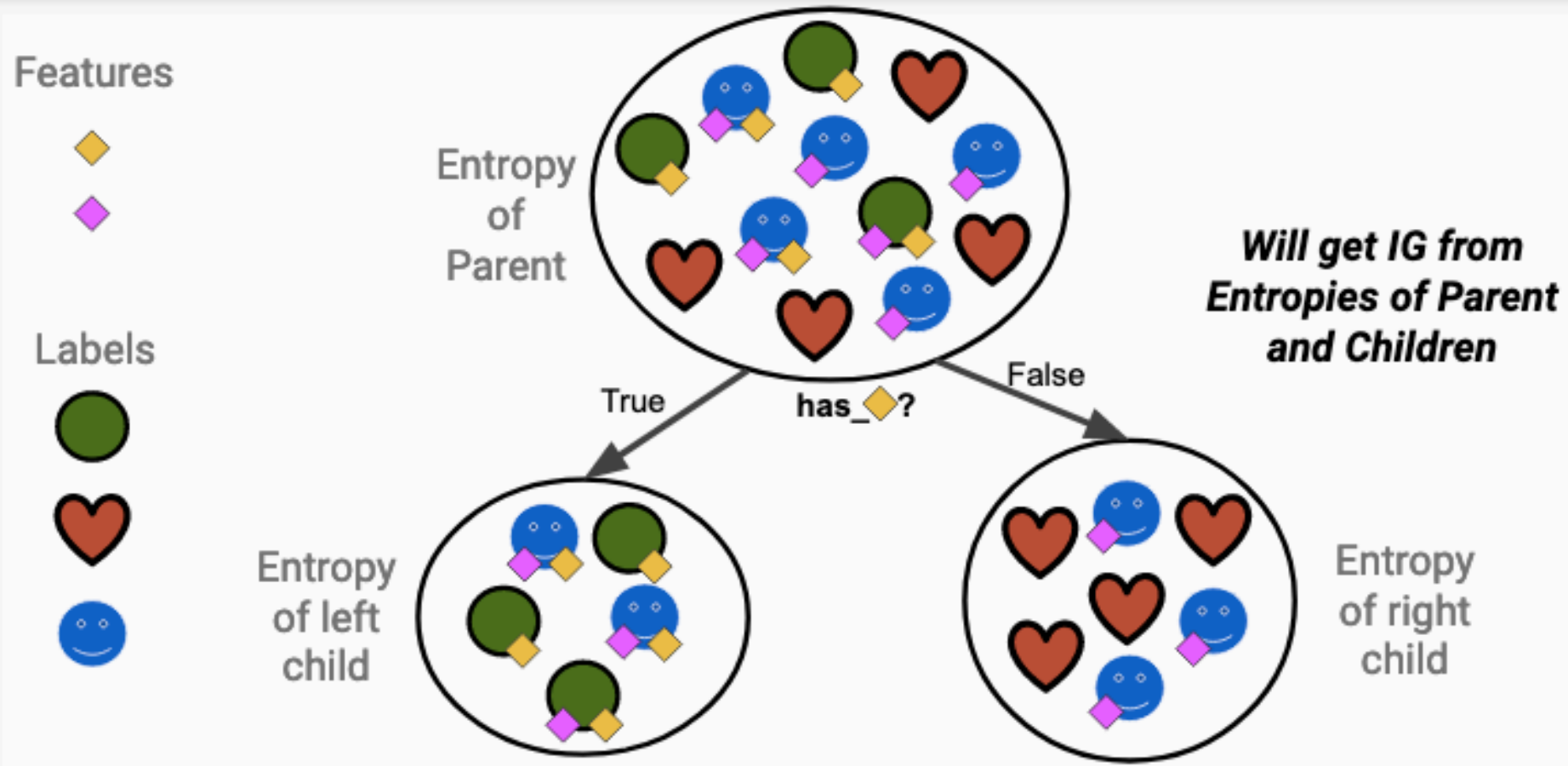
Example 2 Continued



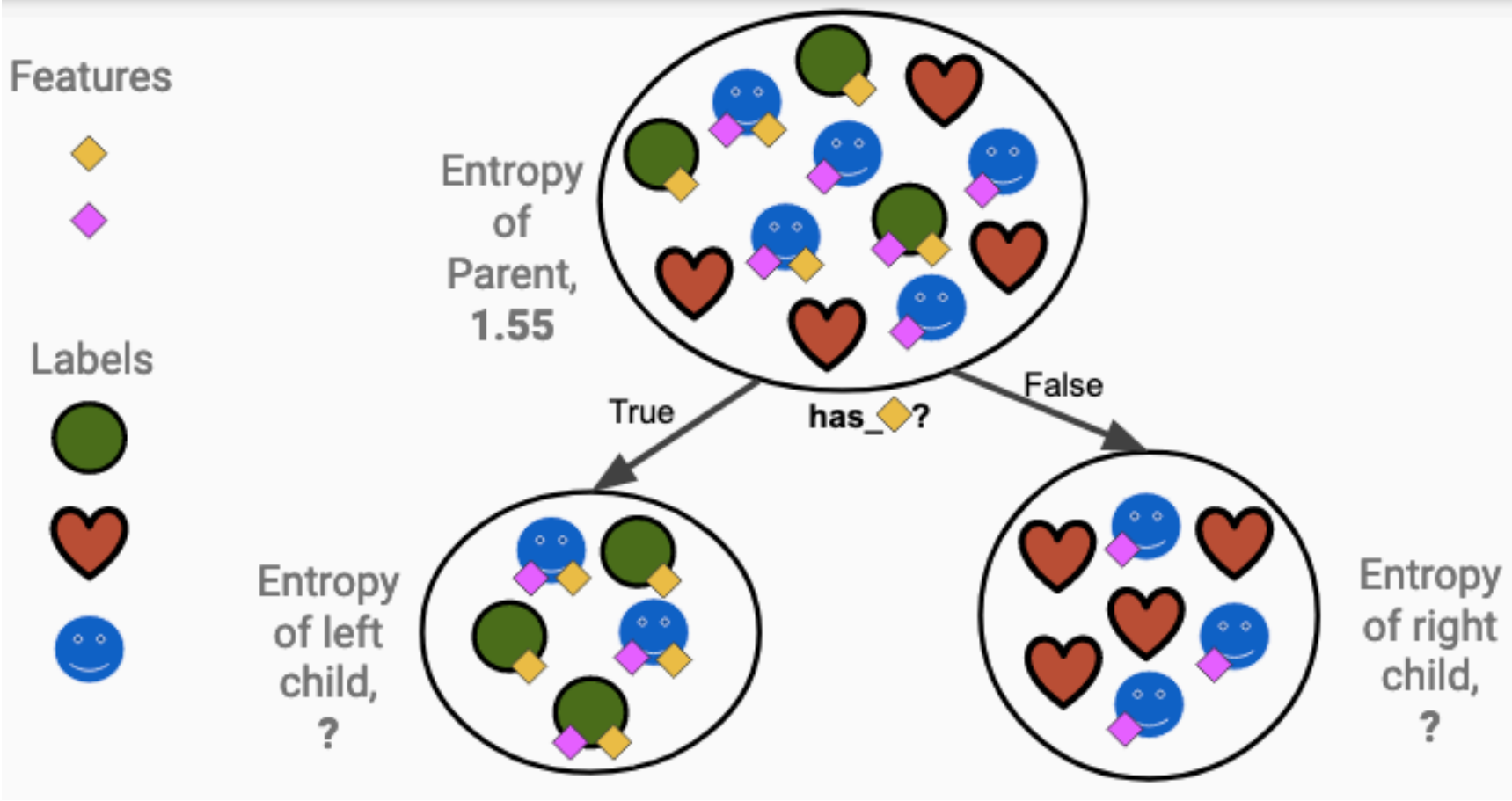
Example 2 Continued



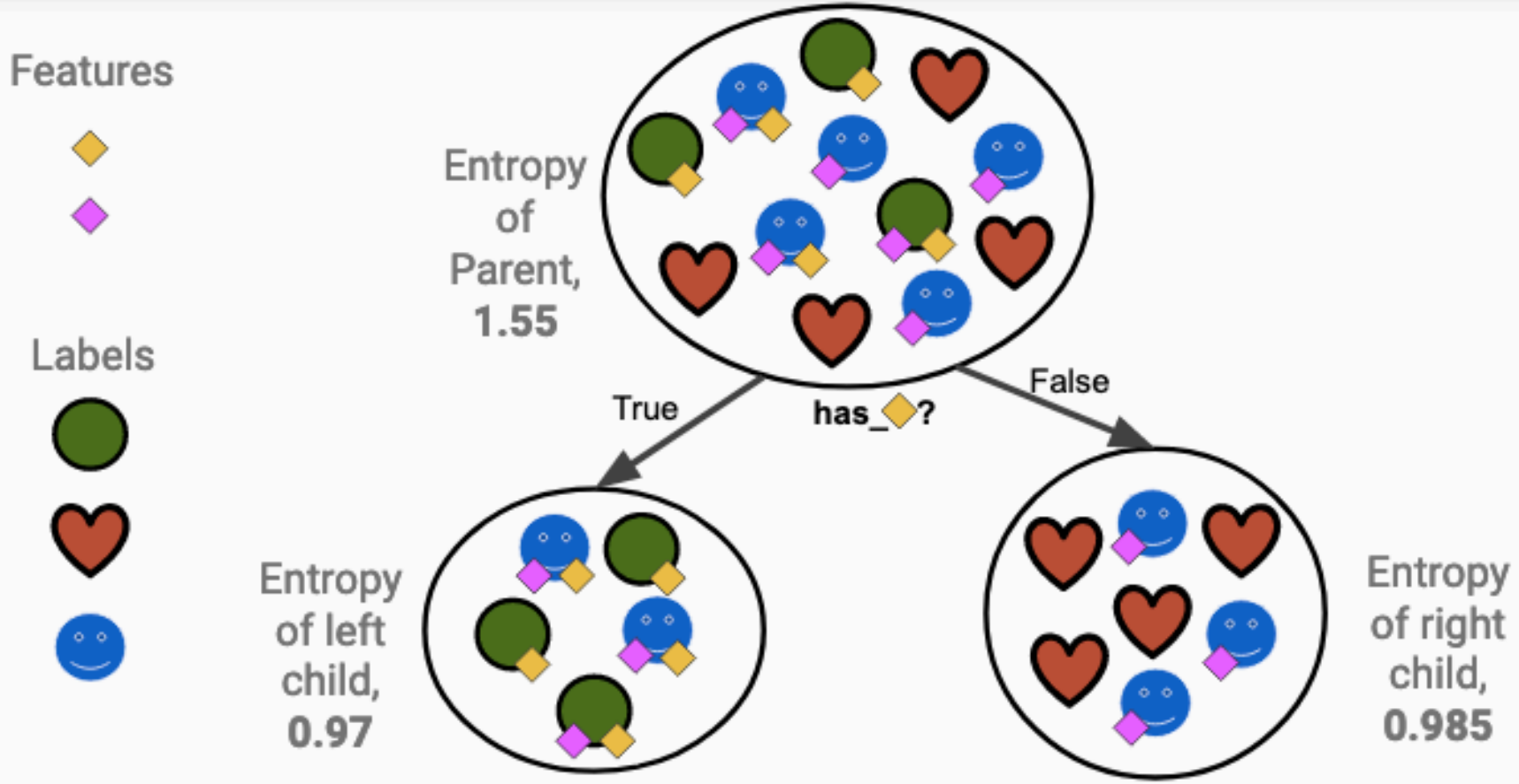
Example 2 Continued



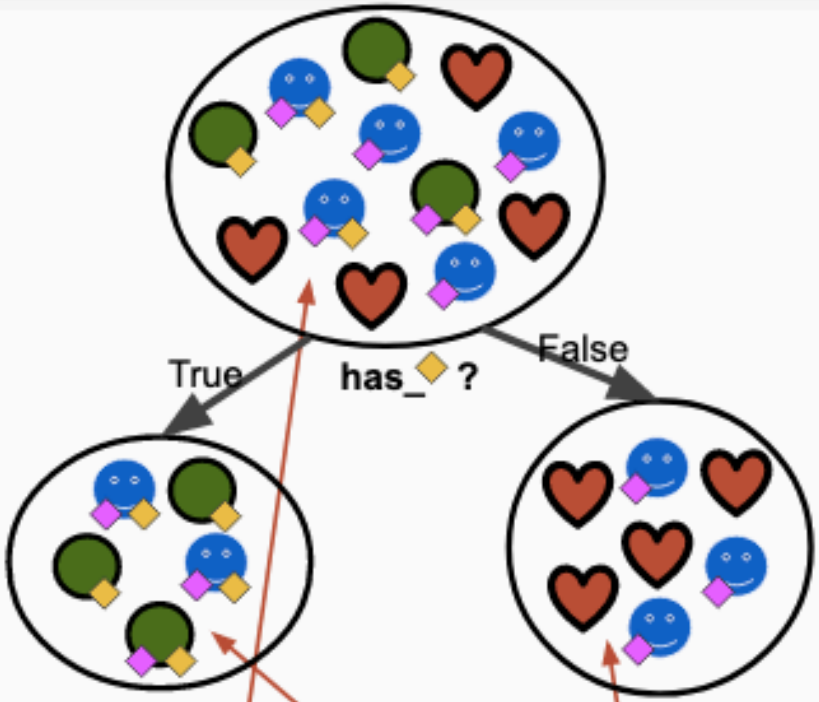
Example 2 Continued



Example 2 Continued



Example 2 Continued



Information gain from this split

the set of children

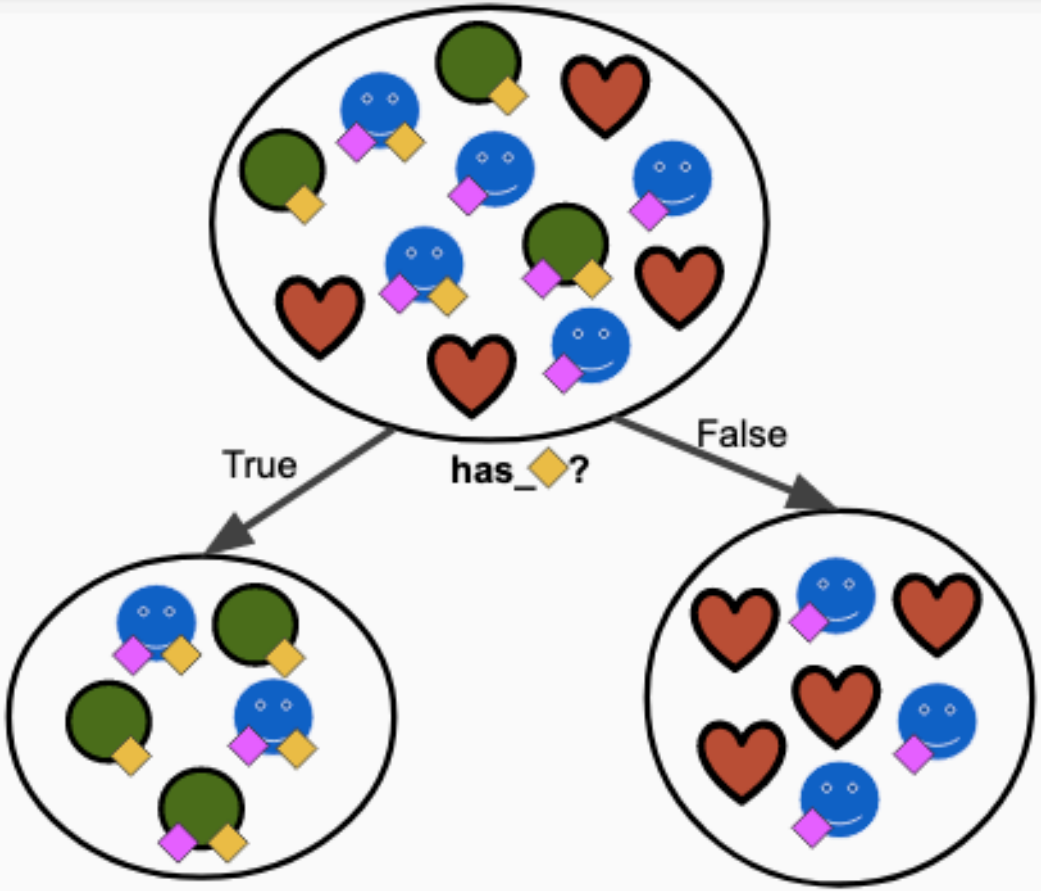
$$\mathrm{IG}(S, C) = H(S) - \sum_{C_i \in C} \frac{|C_i|}{|S|} H(C_i)$$

the parent's set of examples

the set of examples in each child

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

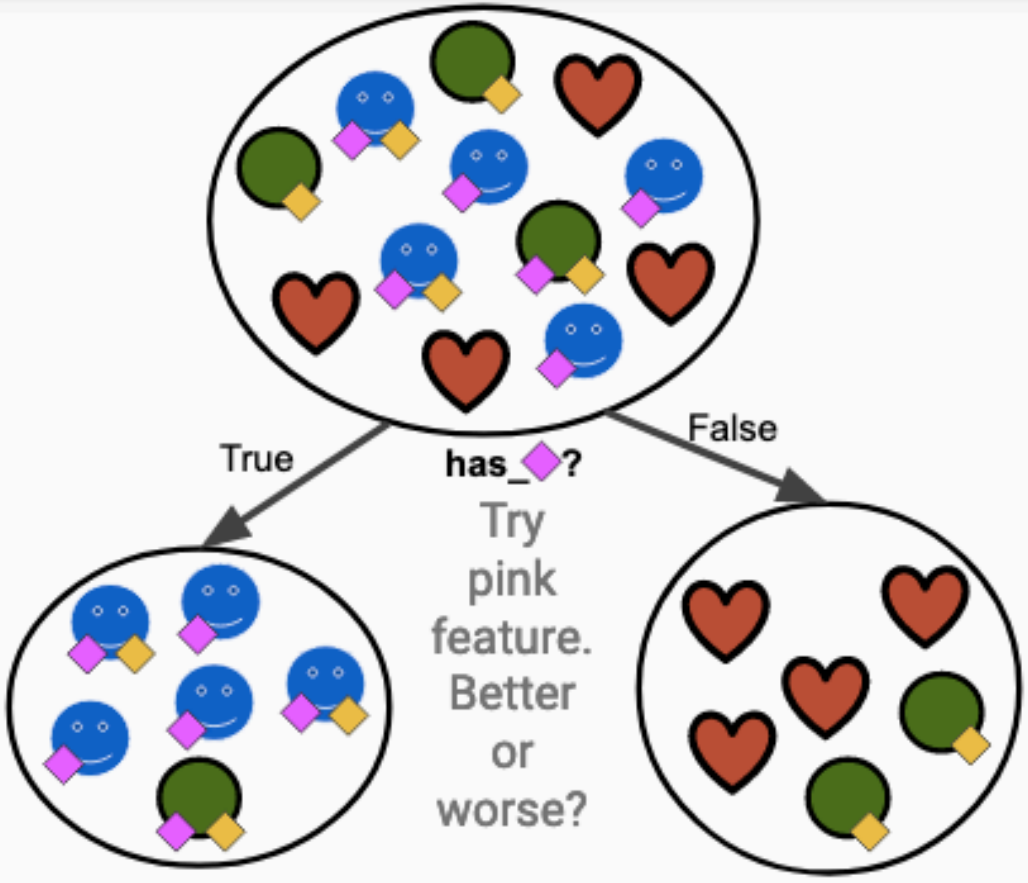
Example 2 Continued



Information Gain = **0.57**

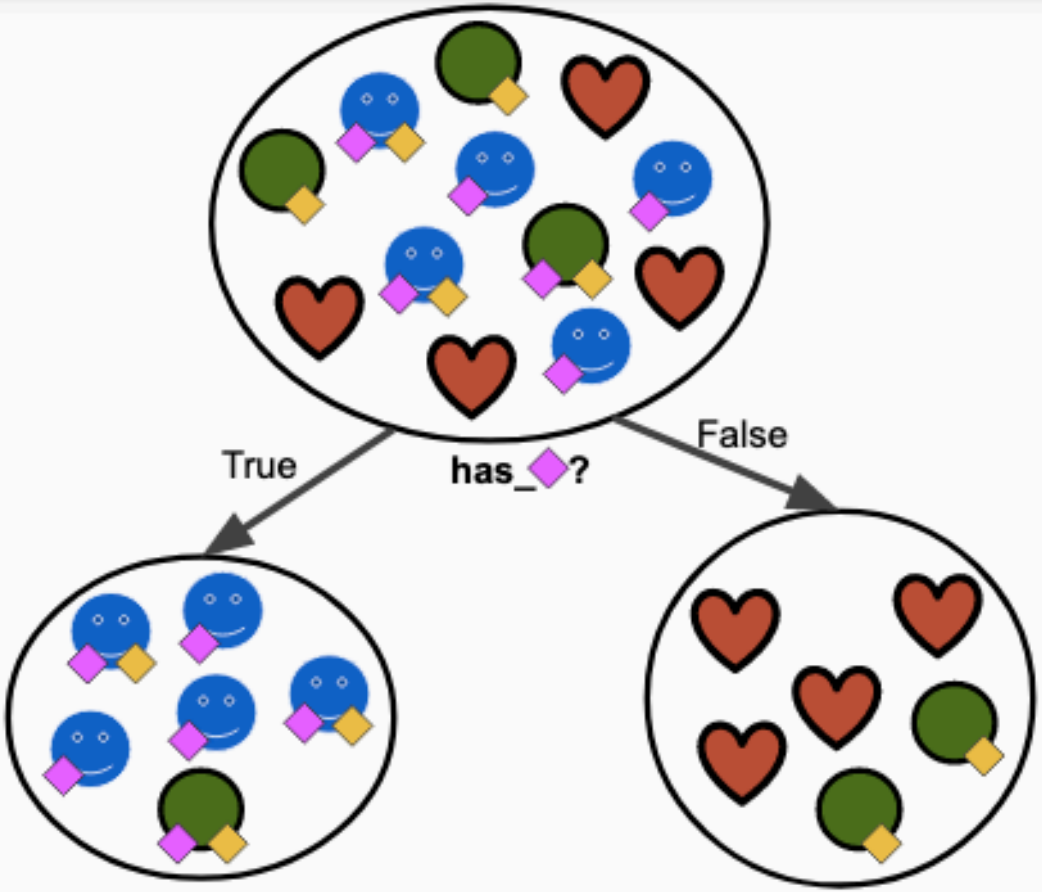
...for splitting on yellow feature.

Example 2 Continued



Information Gain = ??

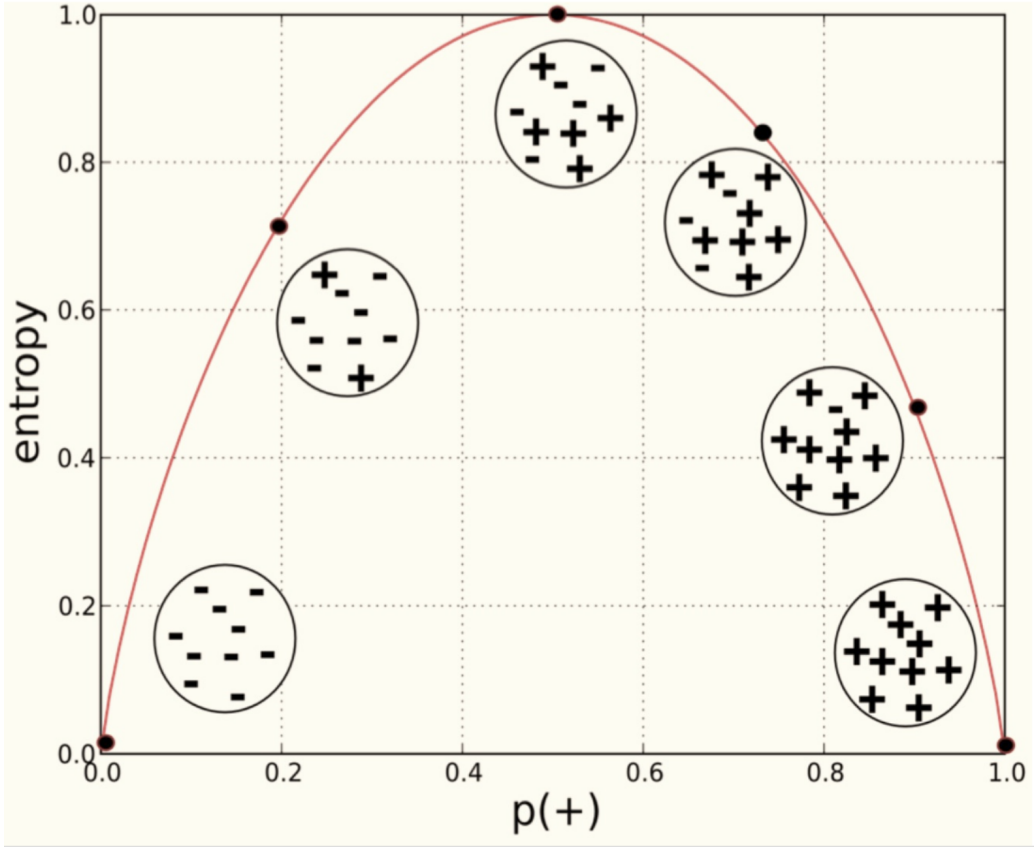
Example 2 Continued



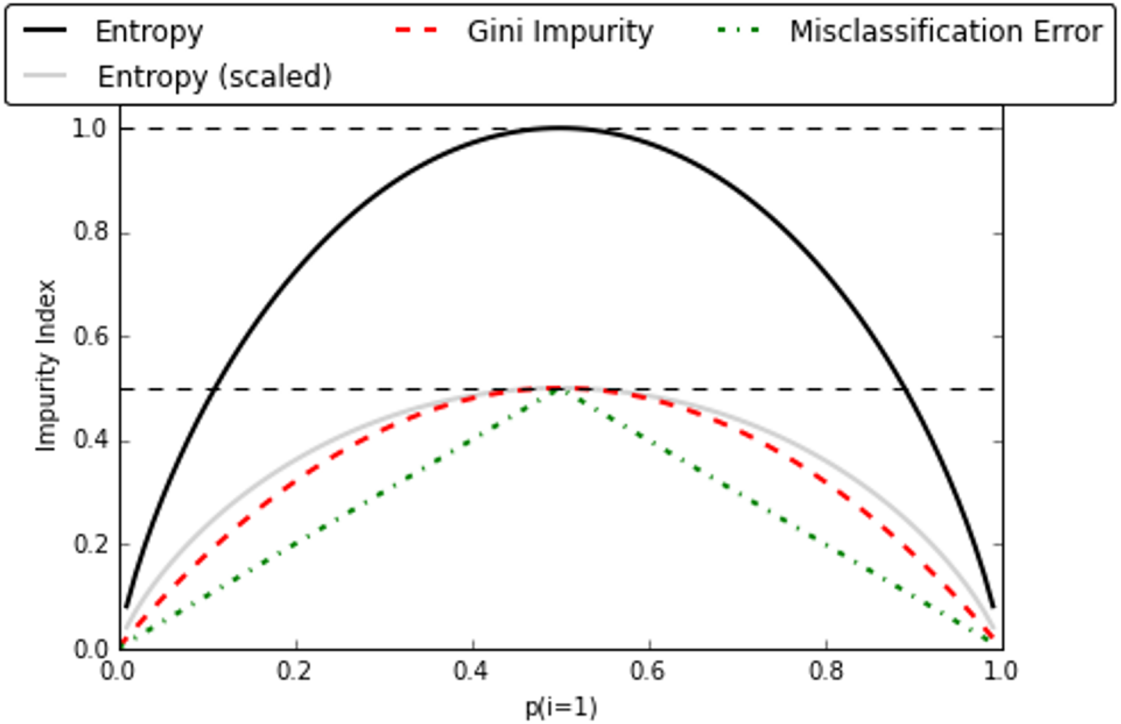
Information Gain = 0.765

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

Visualizing Entropy



Splitting Measure



What about Regression?

- **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:**
- Calculate the information gain for all possible splits.
- Commit to the split that has the highest information gain (as measured by reduction in variance)

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

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

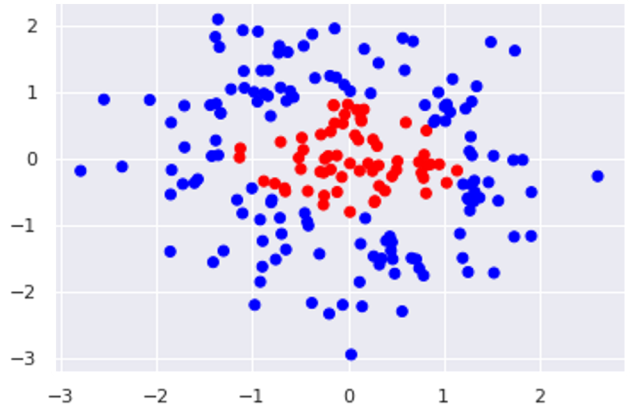
- Gini is default, but you can also choose entropy

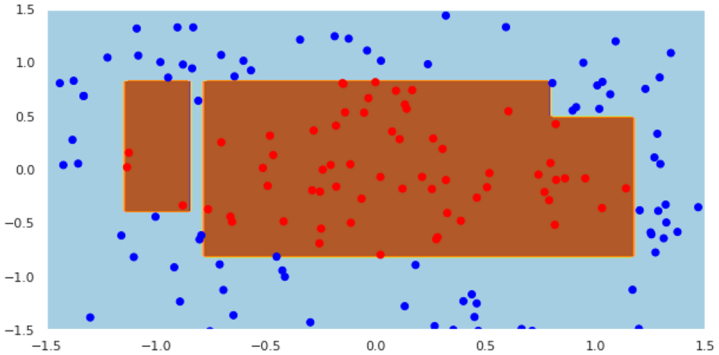
Sklearn - tips

- 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)

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)
```





Algorithms for Training a Decision Tree

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, …
- ...

Decision Tree Advantages:

- Compared to other algorithms decision trees requires less data preparation
- A decision tree does not require normalization or scaling of data
- Missing values in the data also do NOT affect the process of building a decision tree to any considerable extent**
- A Decision tree model is very intuitive and easy to explain to technical teams as well as stakeholders
- Predictions are fast to compute

**SKLearn implementation cannot handle missing values

Decision Tree Disadvantages

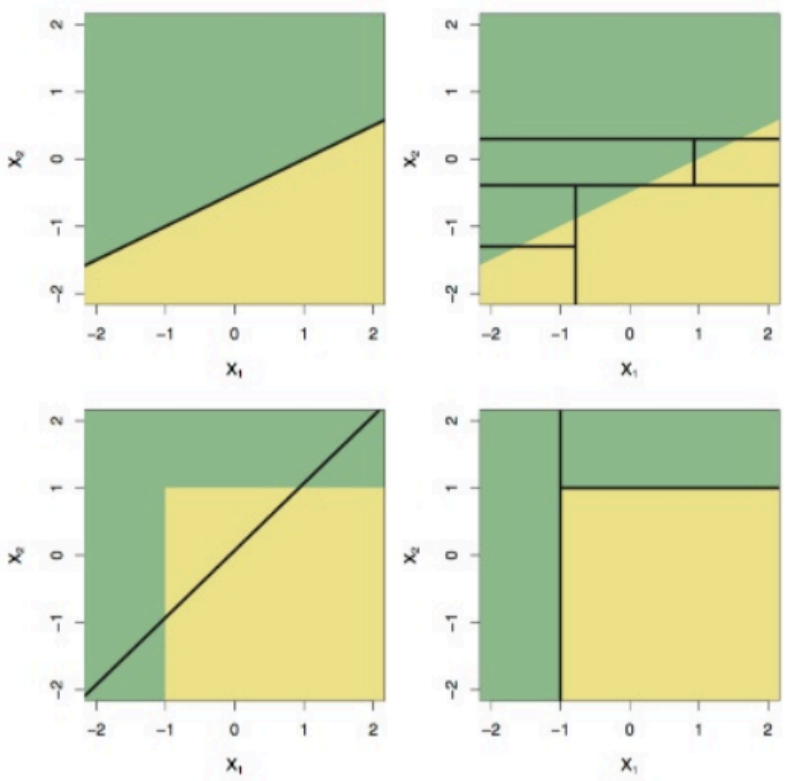
- A small change in the data can cause a large change in the structure of the decision tree causing instability
- Decision tree calculations can be far more complex than to other algorithms
- Decision trees often involve longer training times
- The Decision Tree algorithm is inadequate for applying regression and predicting continuous values
- Decision boundaries are only constructed at right angles

Decision Tree - Decision Boundaries

Figure 8.7 from *ISLR*

Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).

Bottom Row: Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).



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)
```

Example - Recursion for Decision Trees

```
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 #Base Case
    Else:
        find the best feature and value to split the data
        split the dataset
        create a node #Change state and Move Towards Base Case
        for each split
            call BuildTree add the result as a child of the node #Call self
        return node
```

Review

Explain how you:

- Can apply decision tree to classification/regression problems
- Can apply different measures (Entropy, Gini, MSE, etc.) to quantify branching in decision trees
- Would discuss advantages and disadvantages of decision trees
- Would discuss methods used to combat overfitting decision trees
- Apply recursive methods in your programming