

# Decision Trees

Ryan Henning  
Frank Burkholder



- Decision Tree Overview
  - Animation
  - Advantages/Disadvantages
- How to make a split
  - Information Gain
    - Entropy/Gini (Classification)
    - Variance (Regression)
- Decision tree algorithm(s)
  - Recursion
- Implementation in sklearn

- **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](#).

# 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

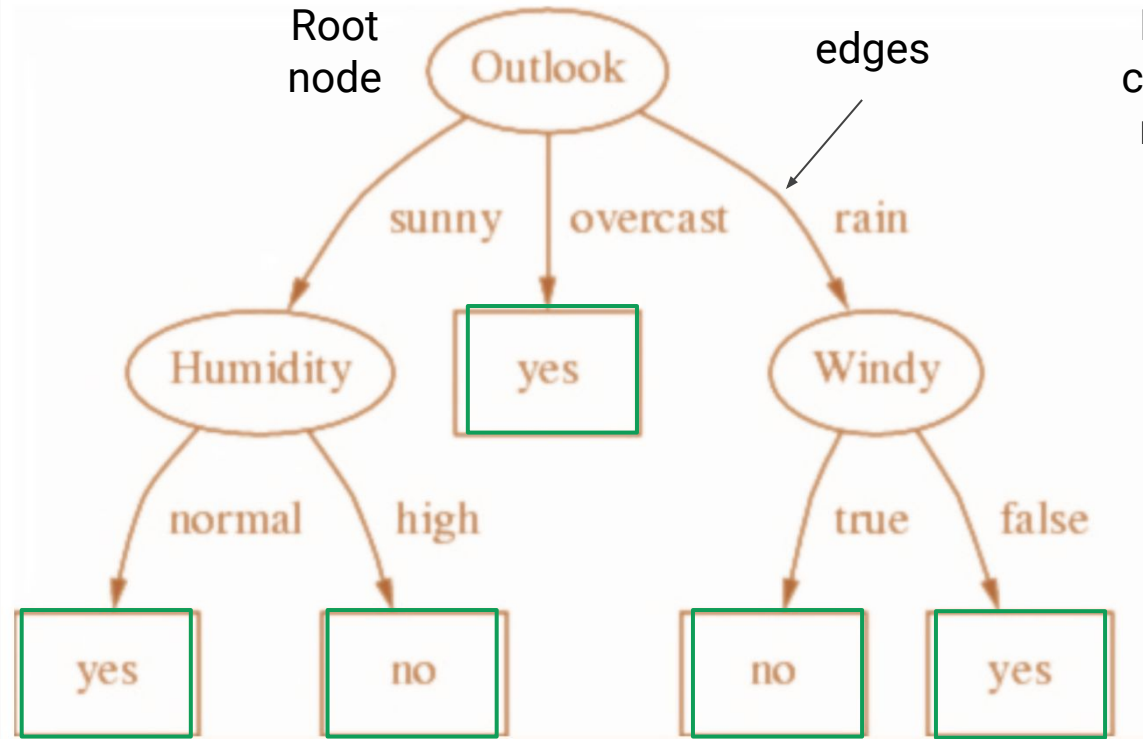
```
def will_play(temp, outlook, humidity,\n              windy):\n\n    if outlook == 'sunny':\n        if humidity == 'normal':\n            return True\n        else: # humidity == 'high'\n            return False\n\n    elif outlook == 'overcast':\n        return True\n\n    else: # outlook == 'rain'\n        if windy == True:\n            return False\n        else: # windy == False:\n            return True
```

Not a decision tree!

# Decision Tree Terminology

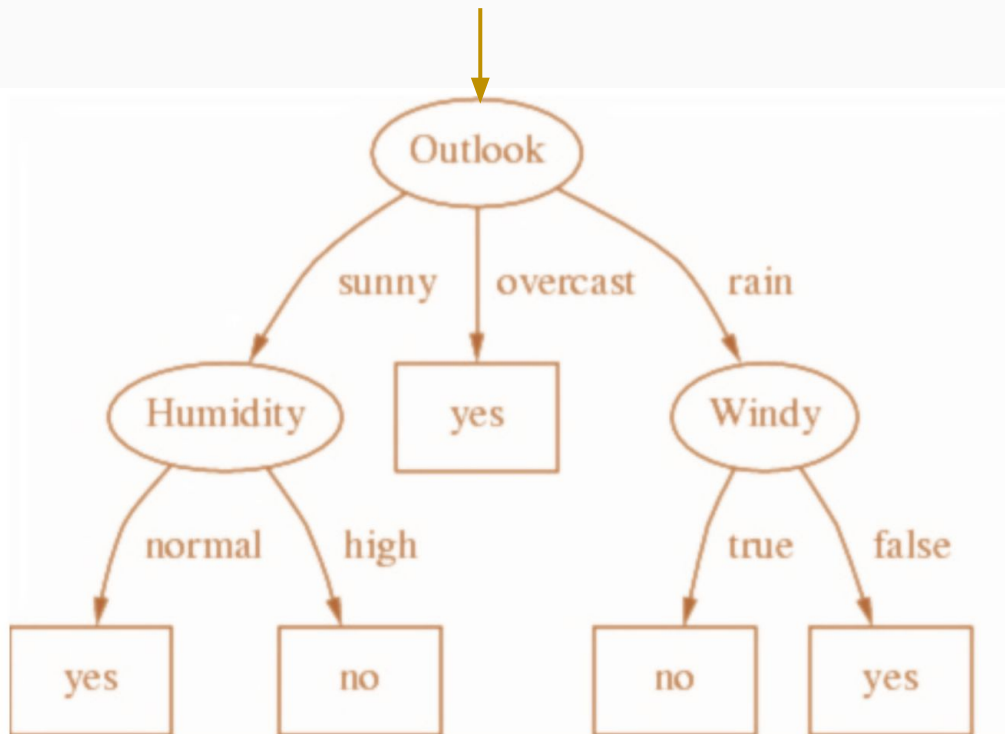
“Nodes” are where data are split on features

Terminal node, a.k.a “leaf”, the final result



Edges connect nodes

Will I play tennis?

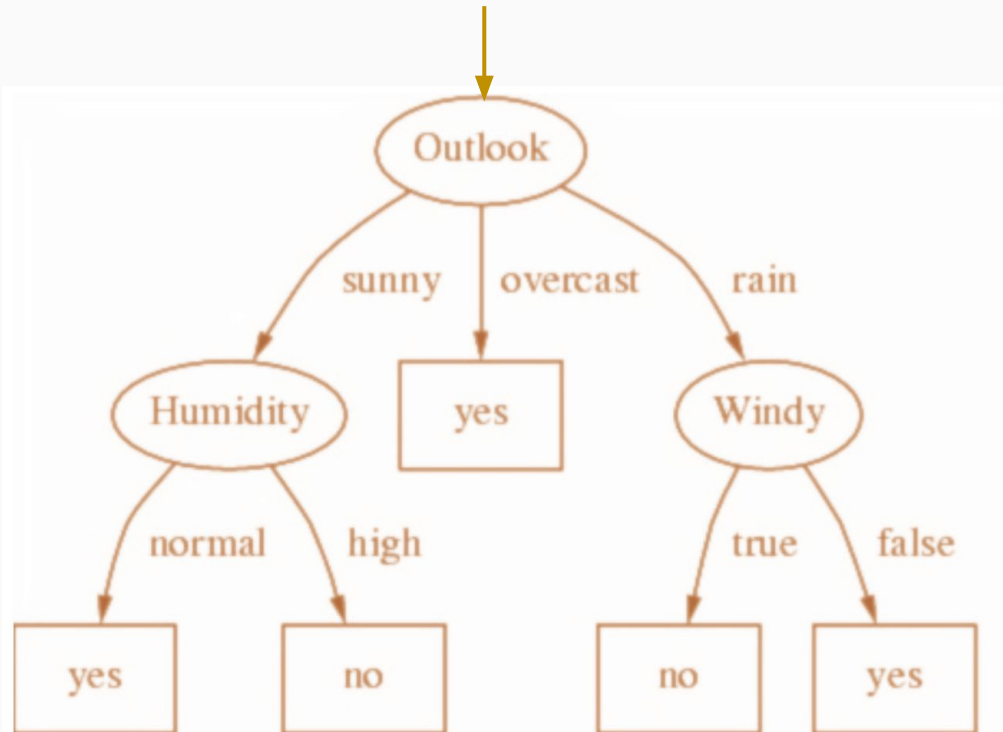


## Benefits:

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

\*Caveats in sklearn

Will I play tennis?



Drawbacks:

- expensive to train
- greedy algorithm (local maxima)
- easily overfits
- right-angle decision boundaries only
- deterministic (you'll get the same model every time)

# Decision Trees:

## 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.

# Decision Trees:

## 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

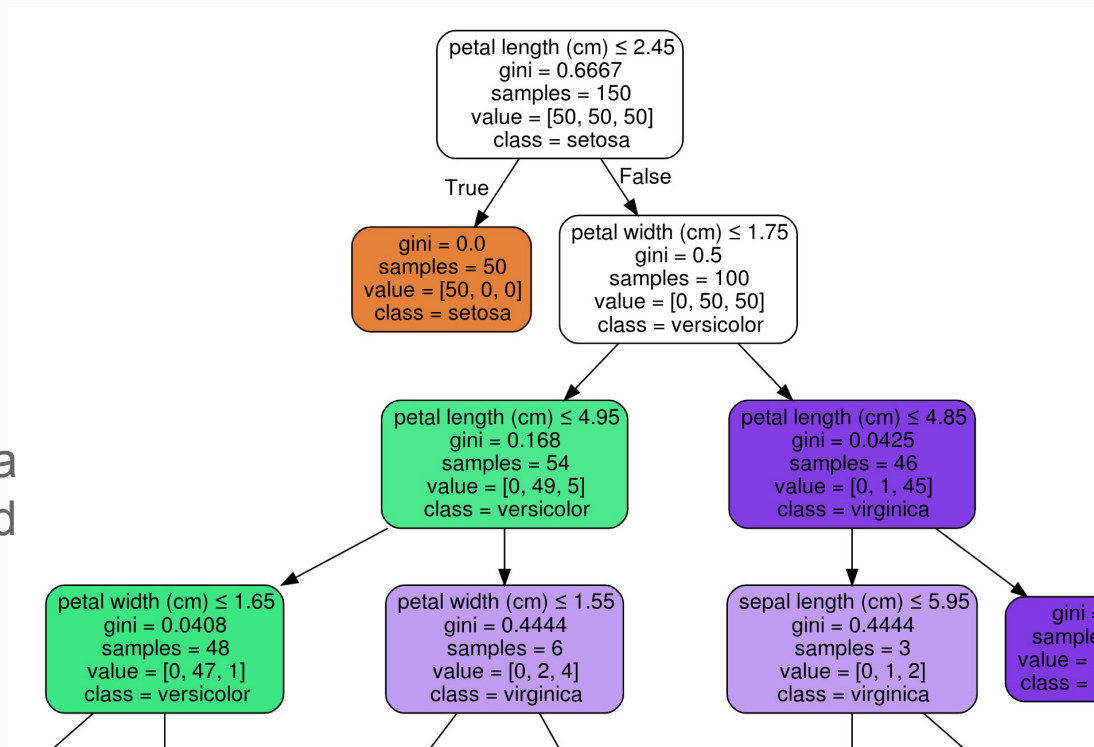
## Splitting Algorithm:

1. Calculate the information gain for all possible splits.
2. Commit to the split that has the highest information gain.



## Need two things:

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



$$\begin{aligned} H(X) &= E[I(X)] = E[\log_2(\frac{1}{P(X)})] \\ &= -E[\log_2(P(X))]\end{aligned}$$

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

Shannon  
Entropy

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

Discrete  
random  
variable



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

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

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



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

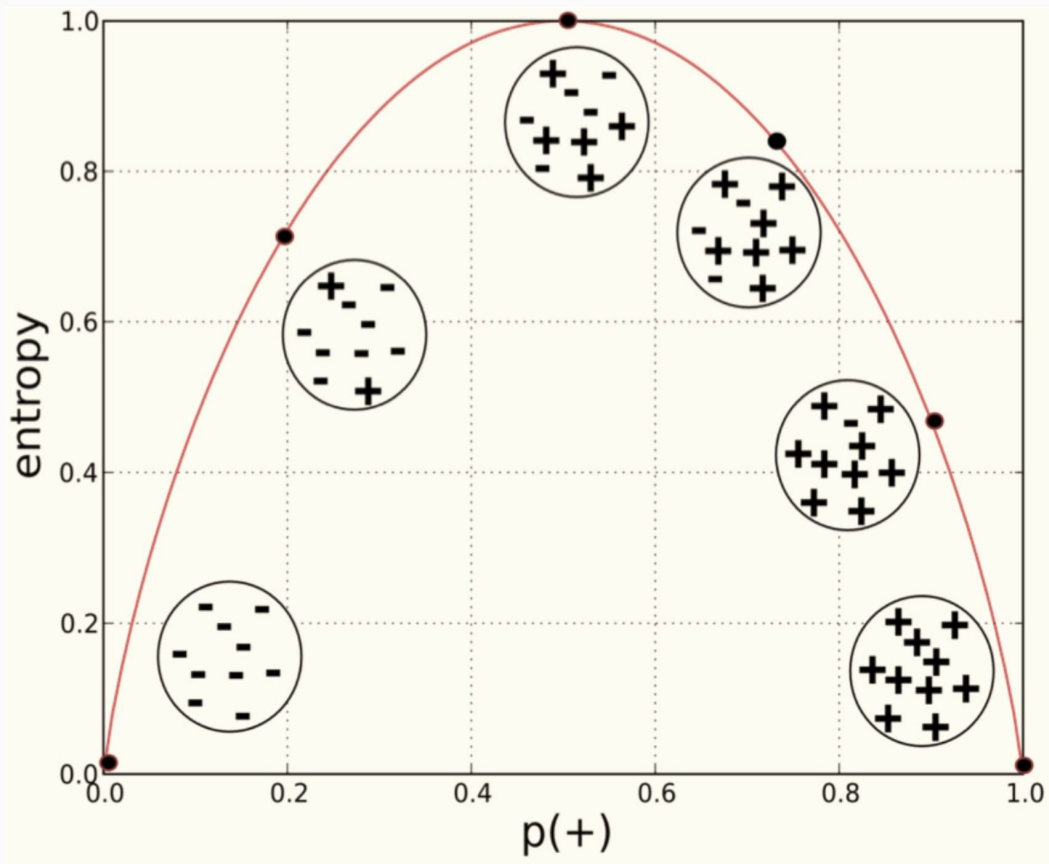
$$H(X) = - \sum_i p_i \log_2(p_i)$$

probability of  
each possible  
discrete outcome

$i$  iterate over pmf

# Entropy graph of a Bernoulli random variable X

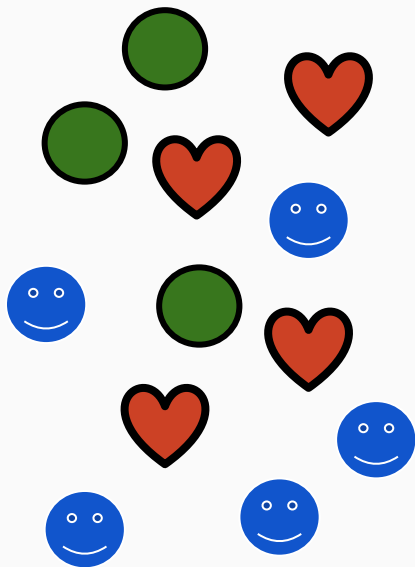
Conceptual/  
visual basis  
for Shannon  
Entropy



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:**

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



$$P(\text{green circle}) = 3/12 = 0.25$$

$$P(\text{red heart}) = 4/12 = 0.33$$

$$P(\text{blue smiley}) = 5/12 = 0.42$$

---

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

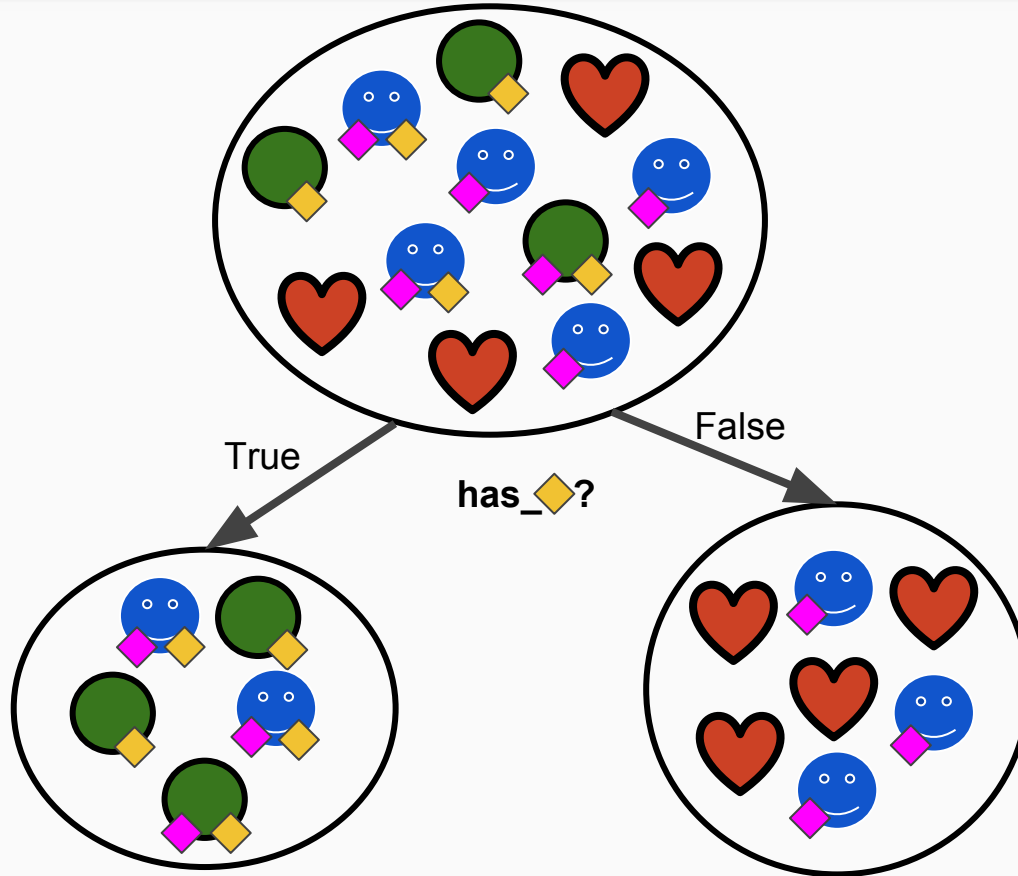
$$H = 1.55$$

# Determining Information Gain from a Split

Features



Labels

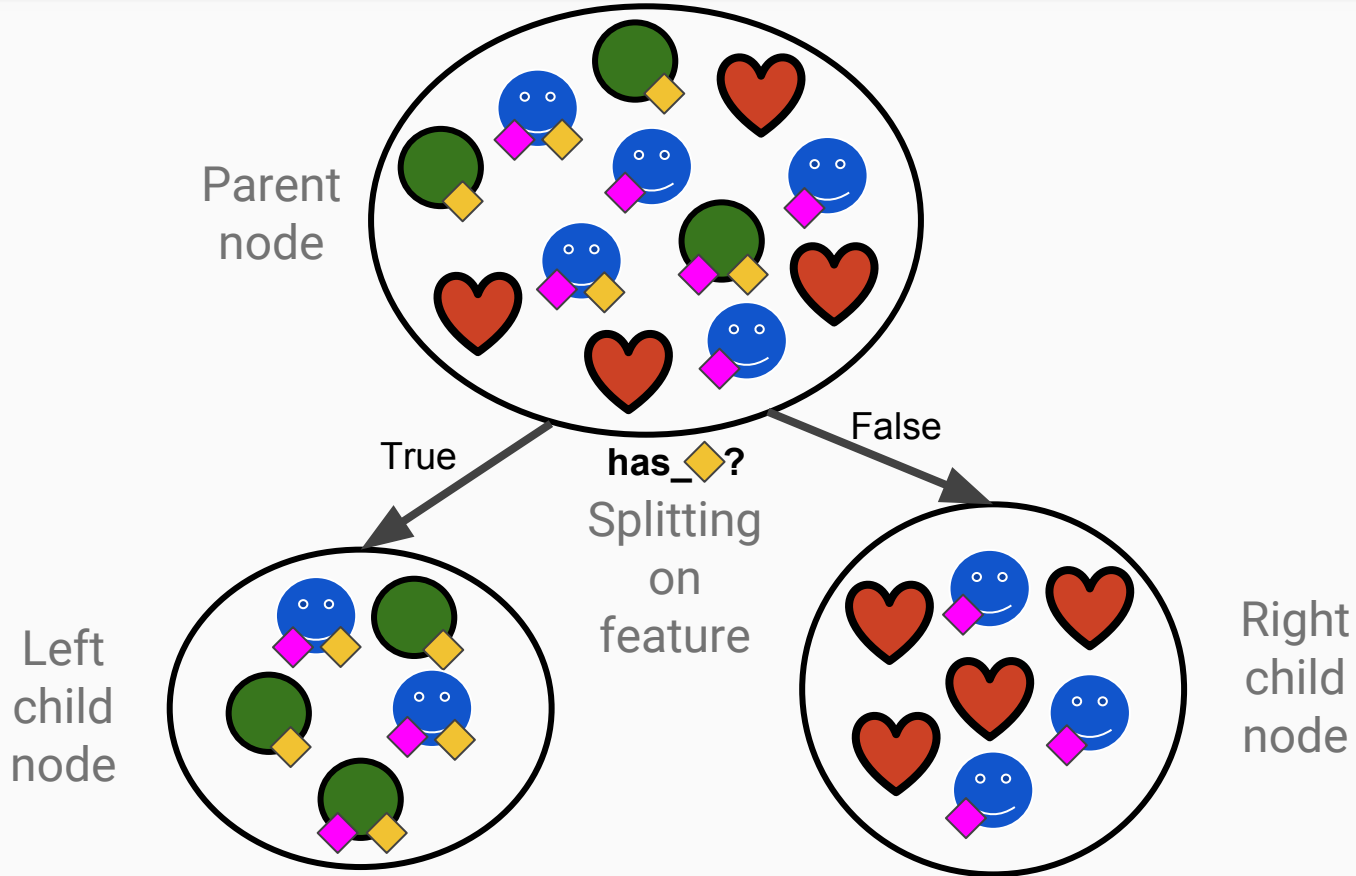


# Determining Information Gain from a Split

Features



Labels



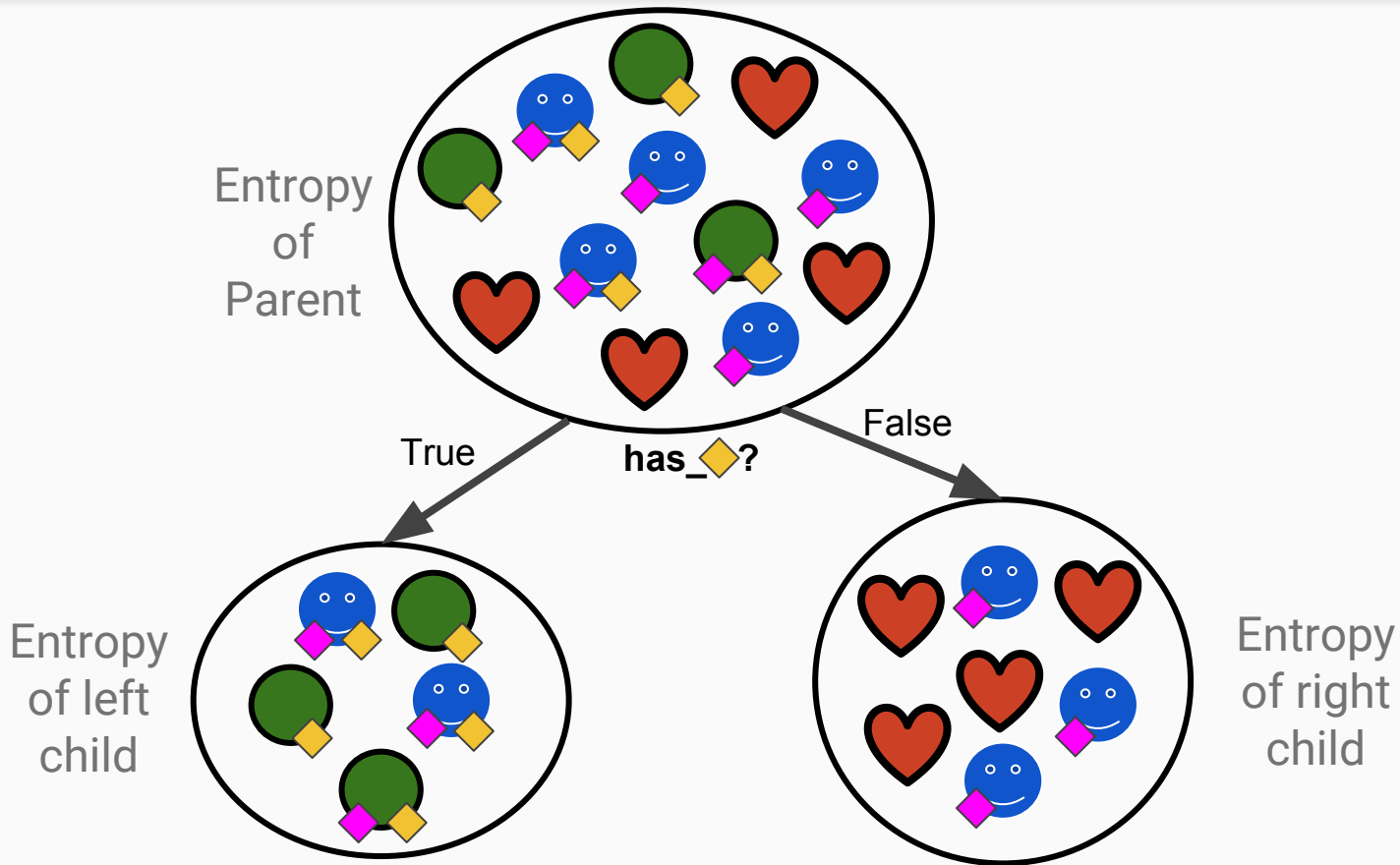


# Determining Information Gain from a Split

Features



Labels

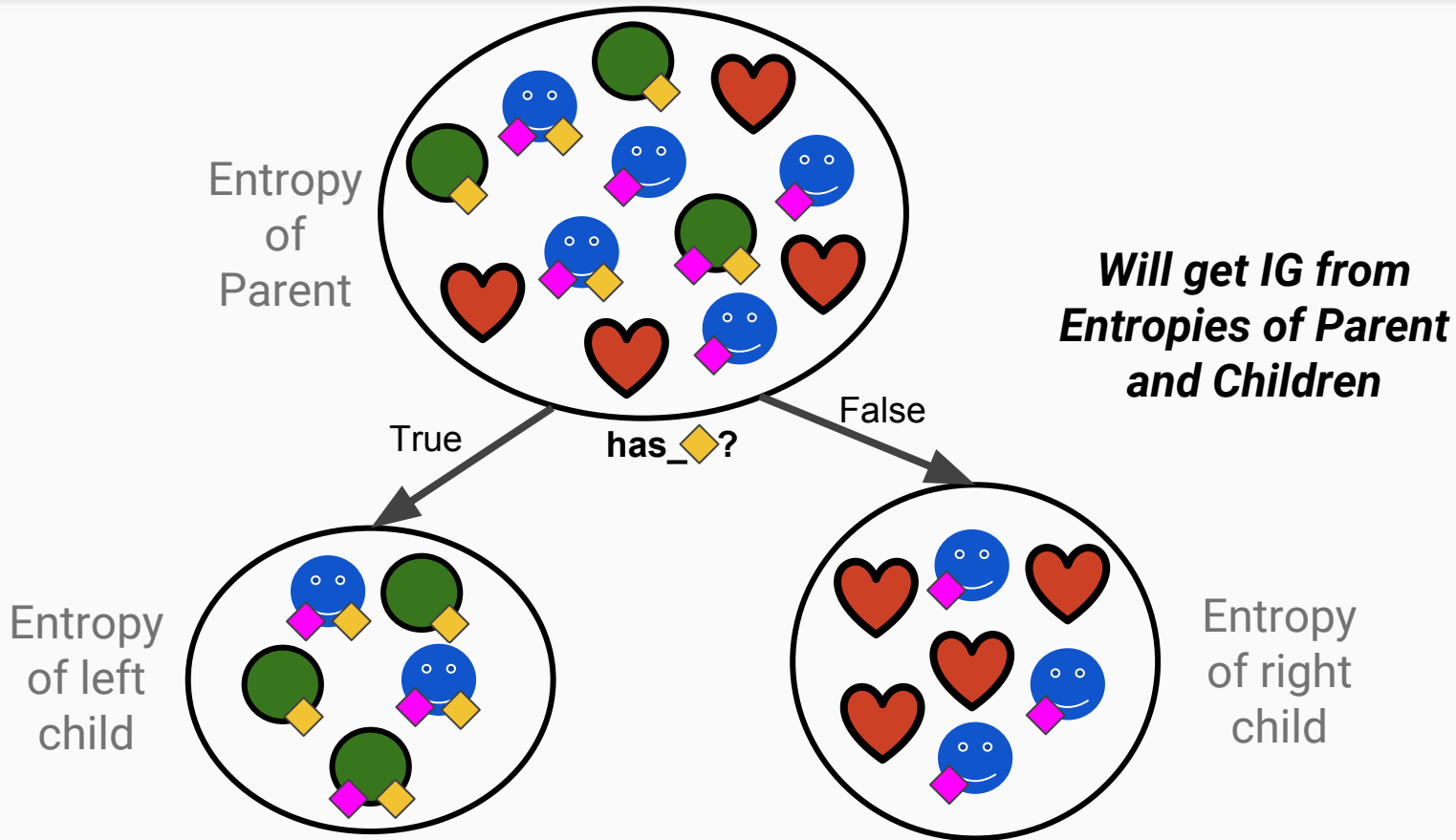


# Determining Information Gain from a Split

Features



Labels

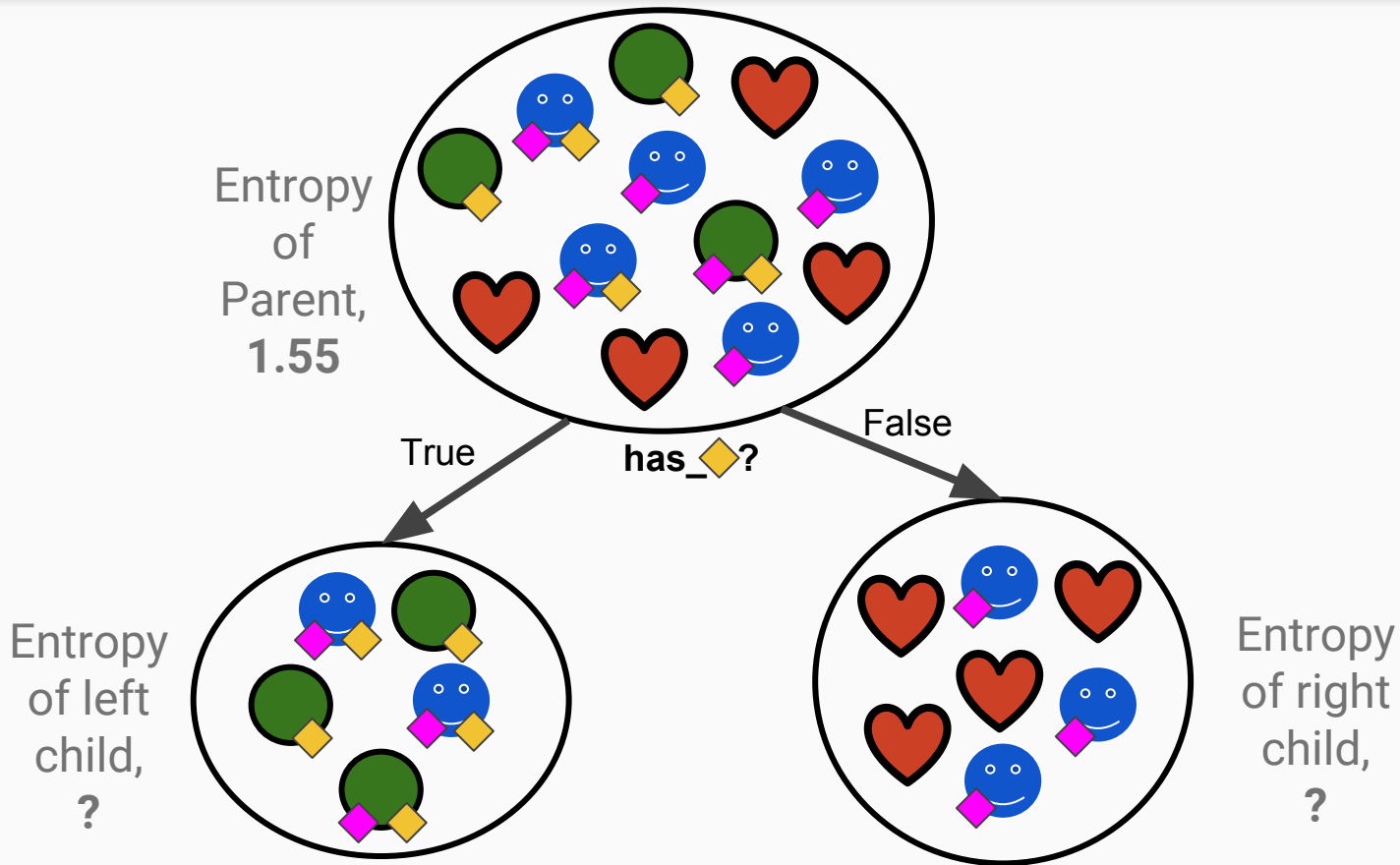


# Determining Information Gain from a Split

Features



Labels

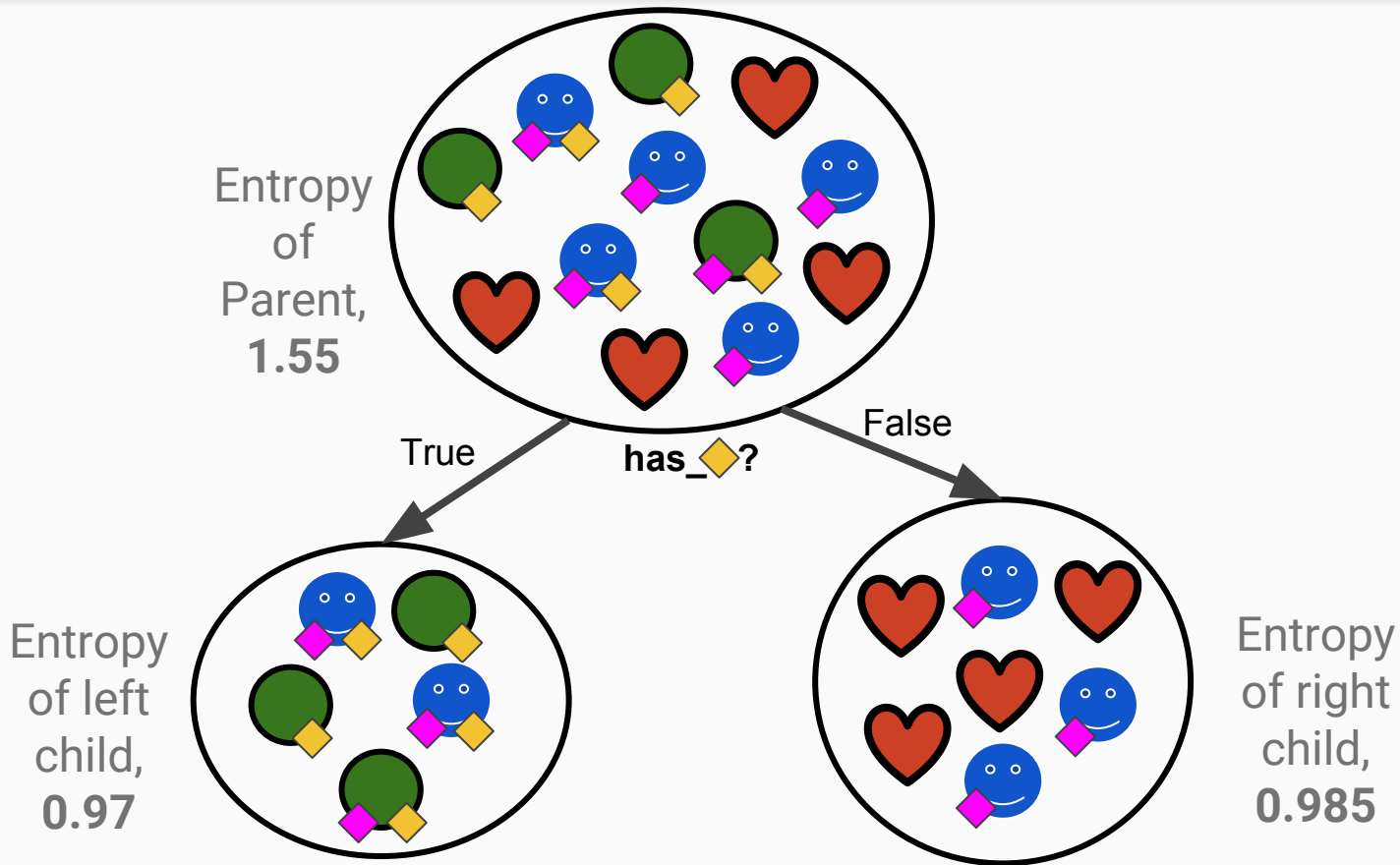


# Determining Information Gain from a Split

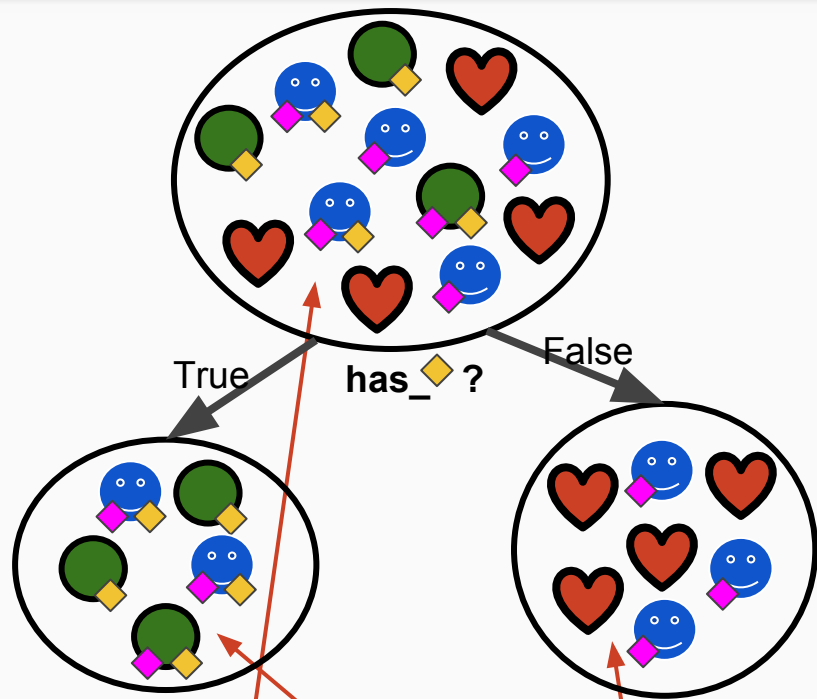
Features



Labels



# Determining Information Gain from a Split



Information gain  
from this split

the set of  
children

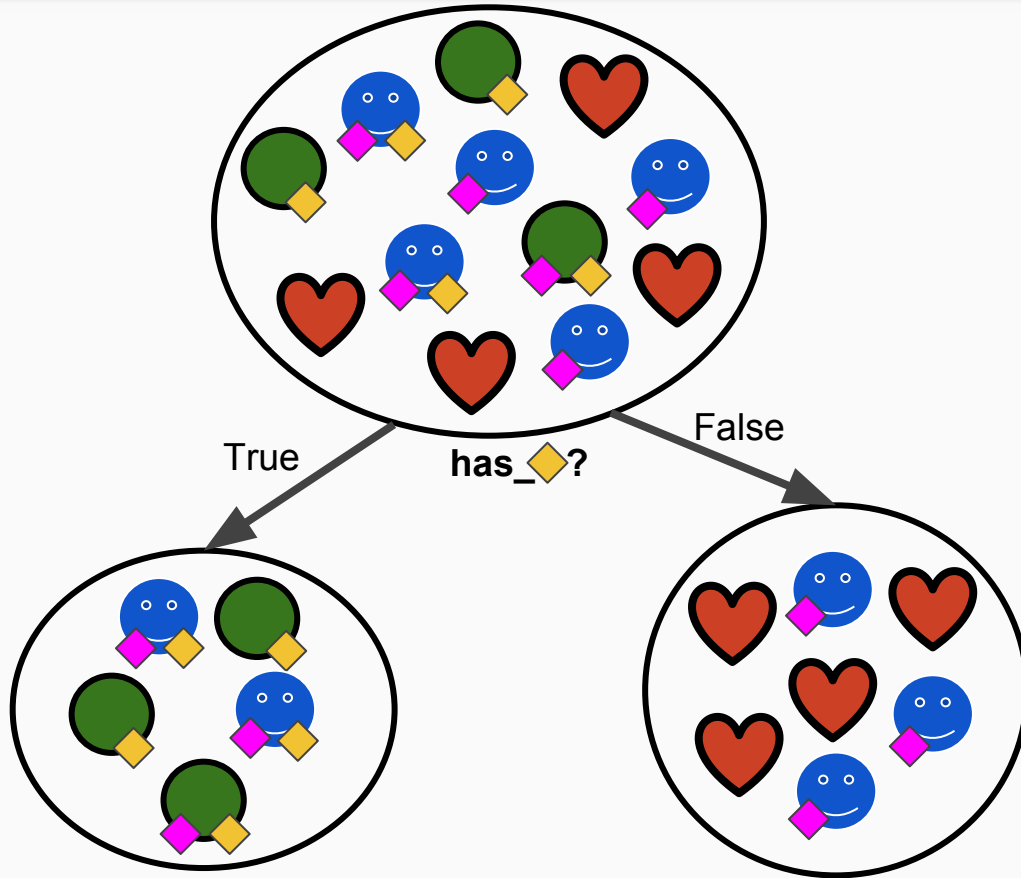
$$IG(S, C) = H(S) - \sum_{C_i \in C} \frac{|C_i|}{|S|} H(C_i)$$

the parent's set  
of examples

the set of  
examples in  
each child

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

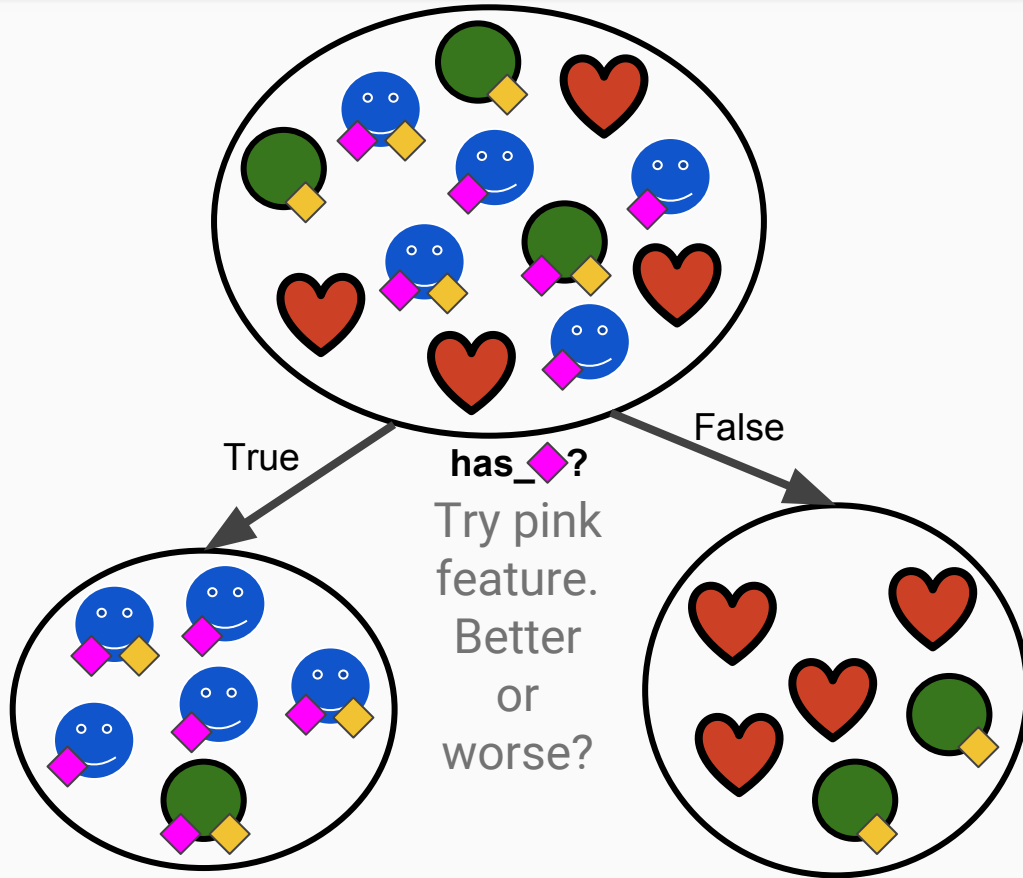
# Determining Information Gain from a Split



Information Gain = **0.57**

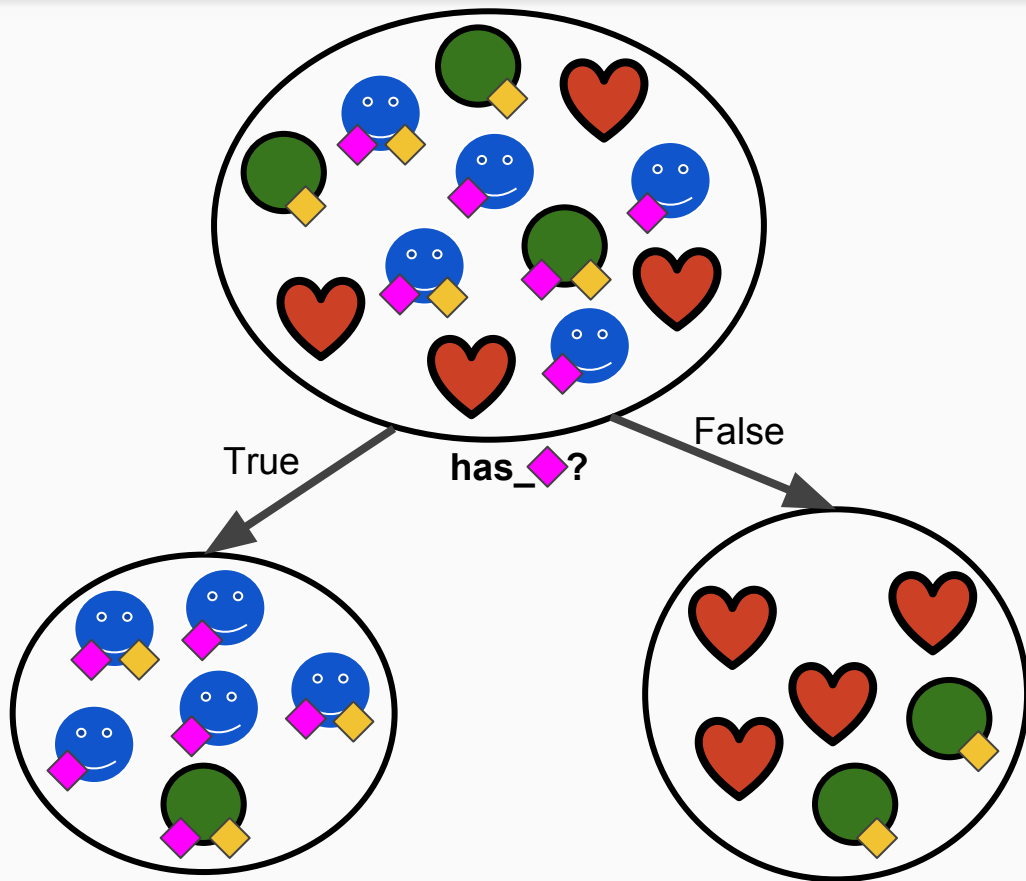
...for splitting on yellow feature.

# Determining Information Gain from a Split



Information Gain = ??

# Determining Information Gain from a Split



Information Gain = 0.765

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



# Splitting Algorithm (Classification):

## 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.

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

# Splitting Algorithm (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:

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

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

# 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*:

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

Information gain using the *Gini Index*:

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

# Recursion

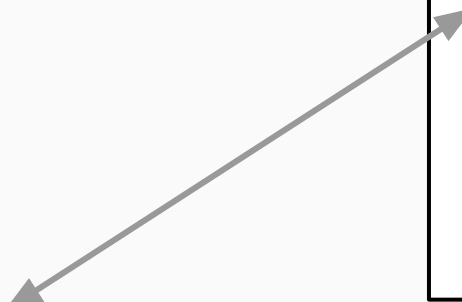
Product notation:

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

A recursive 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)
```



# Build decision tree (note recursive call)

```
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
```

# 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

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)

# 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)