Lecture 6 pre-video

Recursion, big-O, and information entropy

Recursively split trees to minimize disorder

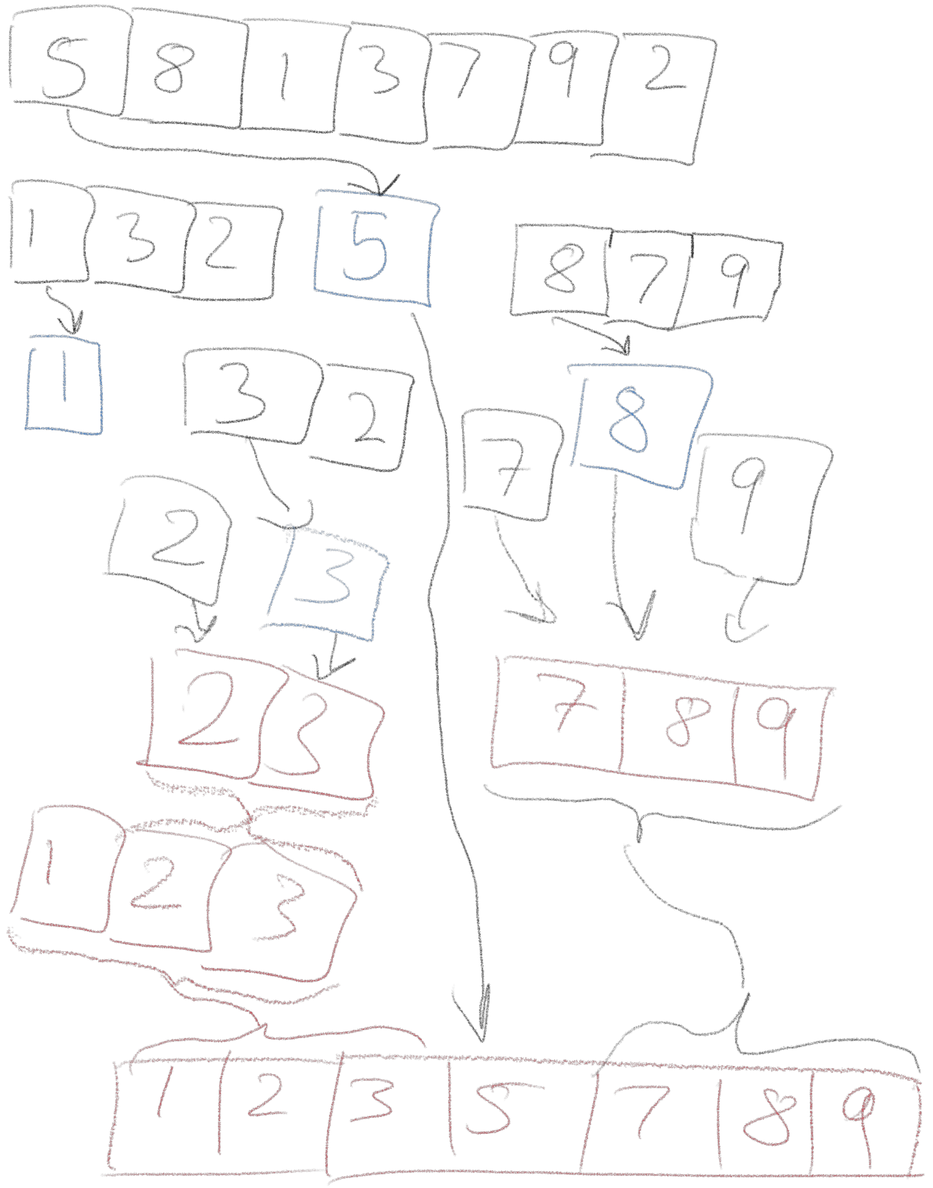
Recursion / Recursive Algorithms

```
1  def some_fun(x):
2      if x == []:
3          return 0
4      else:
5          return 1 + some_fun(x[1:])
```

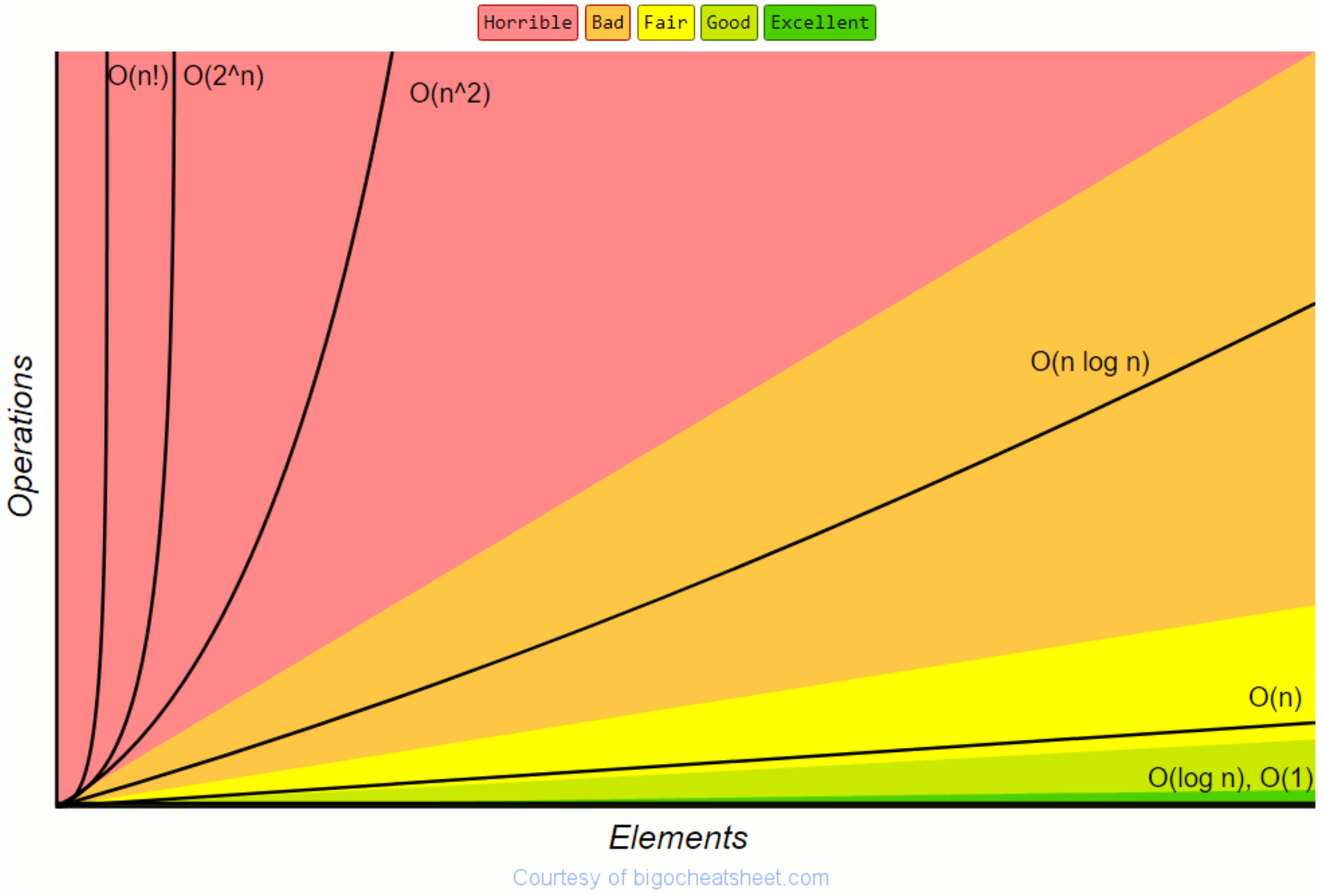
What does this function do?

Divide & Conquer Algorithms: Quicksort

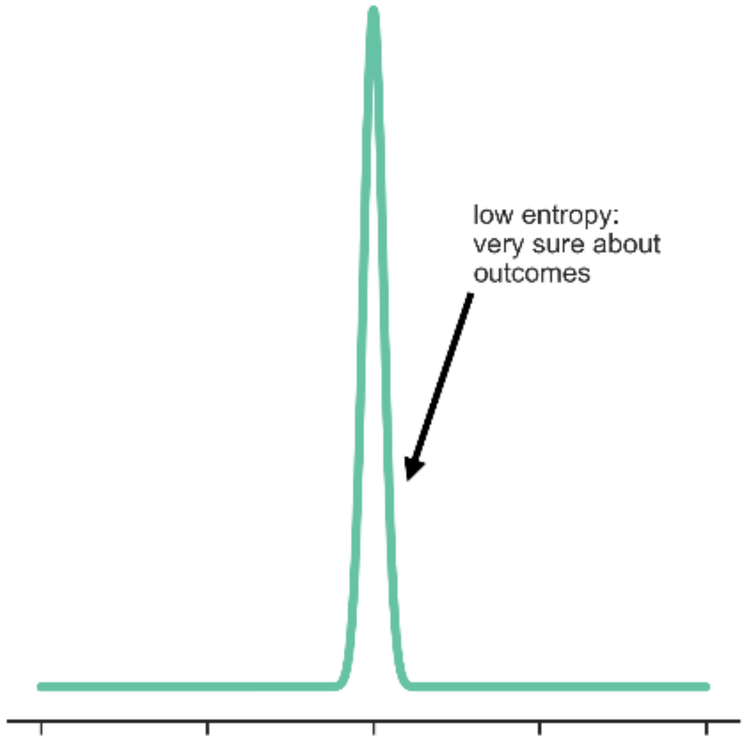
```
1  def quicksort(array):
2      if len(array) < 2:
3          return array
4      else:
5          pivot = array[0]
6          smaller, bigger = [], []
7          for ele in array[1:]:
8              if ele <= pivot:
9                  smaller.append(ele)
10             else:
11                 bigger.append(ele)
12         return quicksort(smaller) + [pivot] + quicksort(bigger)
```

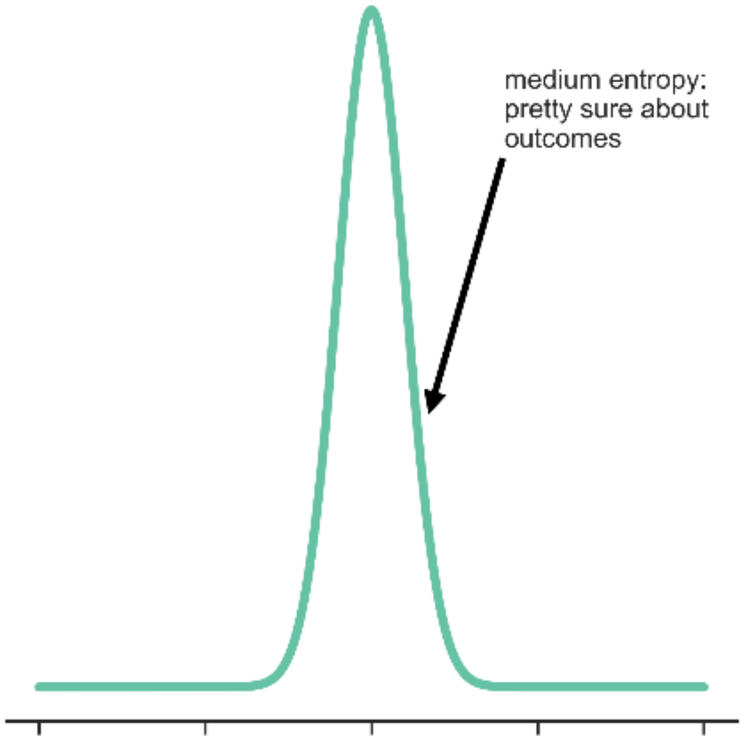


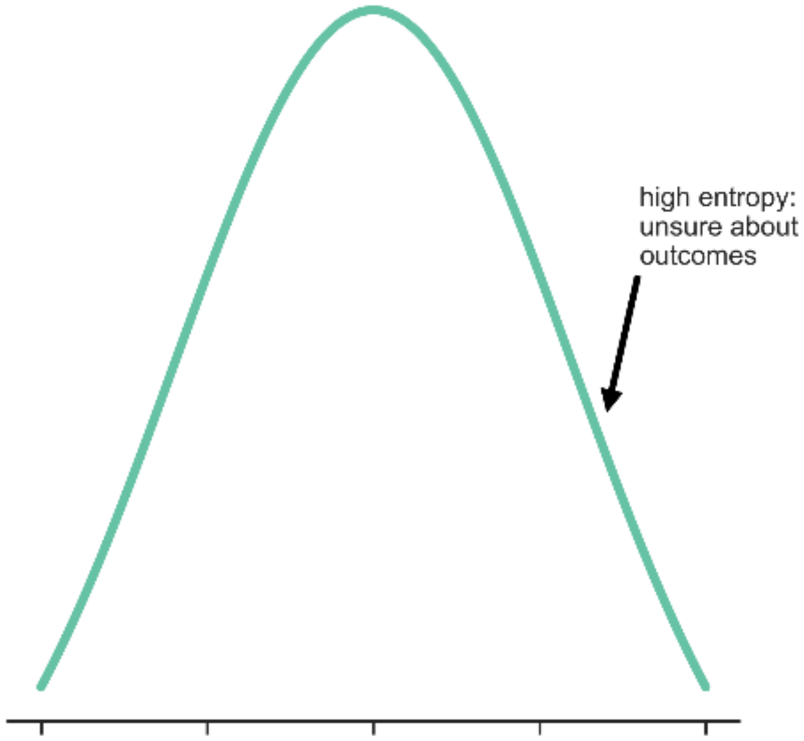
Complexity



Big-O notation	Computations for 10 elements	Computations for 100 elements	Computations for 1000 elements
O(1)	1	1	1
O(log n)	3	7	10
O(n)	10	100	1000
O(n log n)	33	664	9966
O(n^2)	100	10000	1000000
O(2^n)	1024	1.26765E+30	1.0715E+301
O(n!)	3628800	9.3326E+157	4.0238726E+2567







Distribution is mostly just one thing

This is ordered and therefore predictable

You won't be surprised once you actually draw a sample
You wouldn't pay to know the outcome before the draw

Distribution has lots of different things

This is disordered and therefore unpredictable

Drawing a sample could yield surprising results
It would be valuable to know the outcome before the draw

Entropy

General measure for knowing the underlying uncertainty of a random variable.

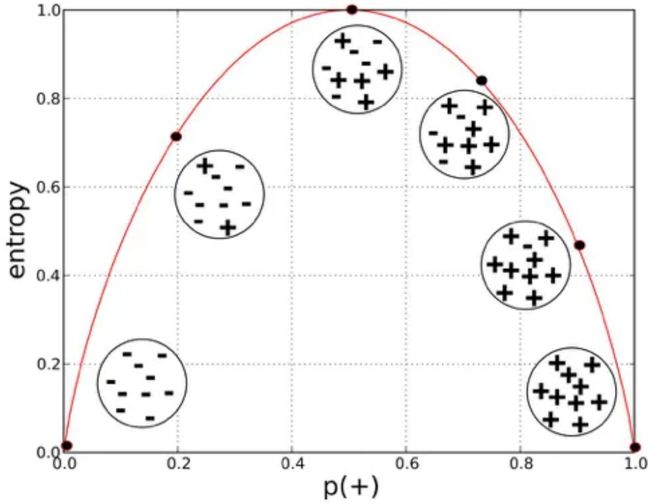
Discrete random variable:

$$H(X) = -\sum_i P(X = x_i) \log P(X = x_i)$$

Continuous random variable:

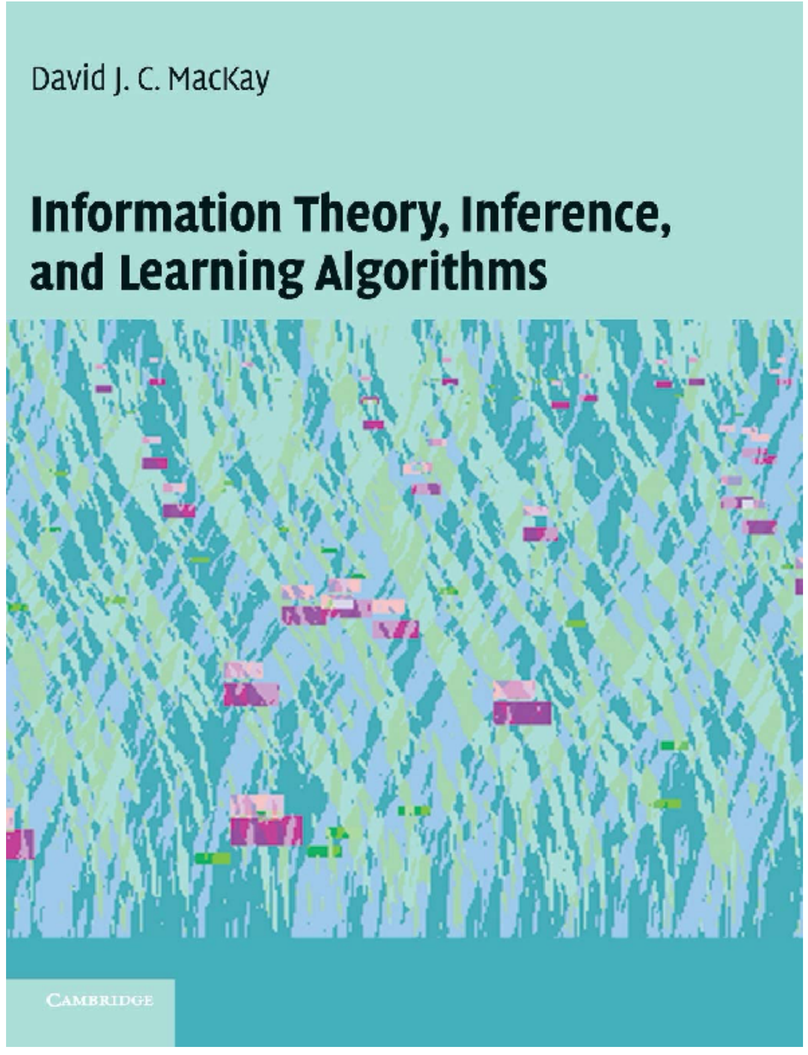
$$H(X) = -\int p(x) \log p(x) dx$$

0.5	**0.5**	$H(X) = -(0.5 \times \log 0.5 + 0.5 \times \log 0.5) \approx 0.30$
0.9	**0.1**	$H(X) = -(0.9 \times \log 0.9 + 0.1 \times \log 0.1) \approx 0.14$
0.1	**0.9**	$H(X) = -(0.9 \times \log 0.9 + 0.1 \times \log 0.1) \approx 0.14$



To better understand information theory

… and its relationship to machine learning



Chapter 8

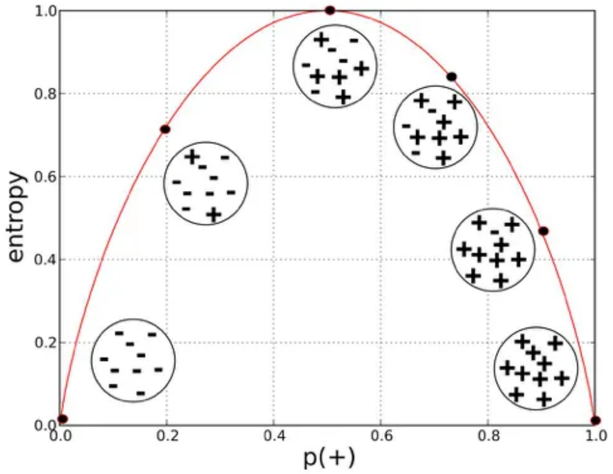
http://www.inference.org.uk/itprnn/book.pdf

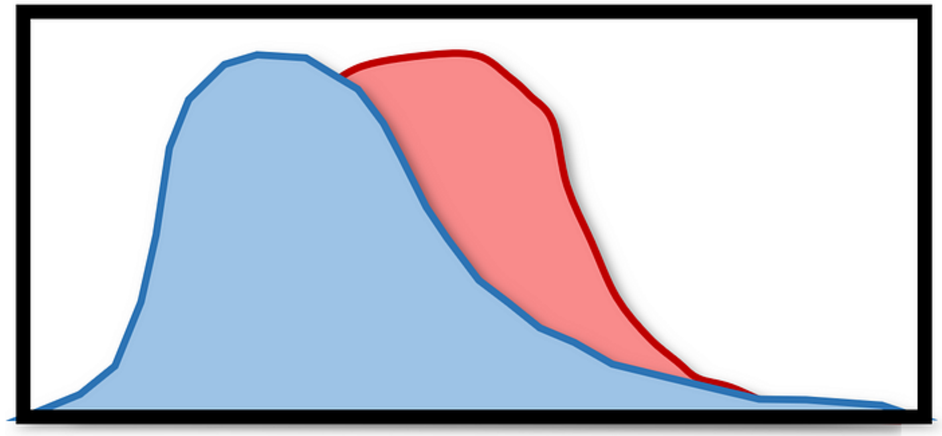
Decision stumps

Stumps

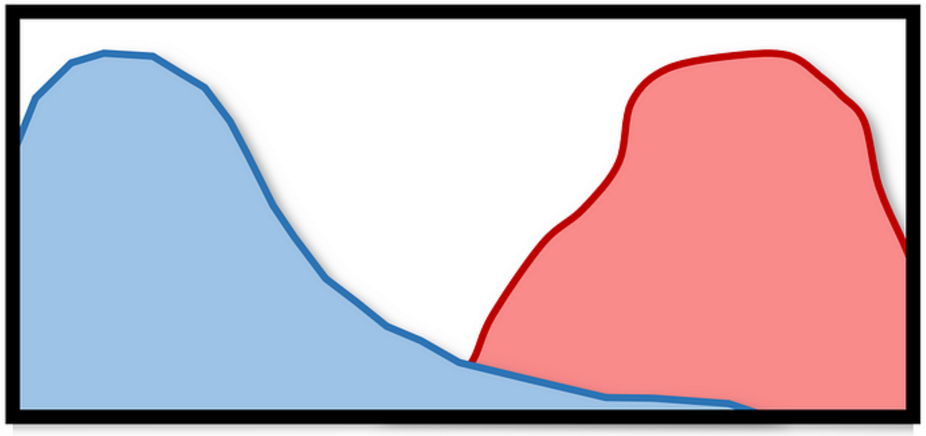
Pick a feature that best separates two classes

- Problem: you've got + and - class samples, with measurements of several features
- Pick ONE feature to predict with. Pick the one that is going to make it easiest to predict accurately



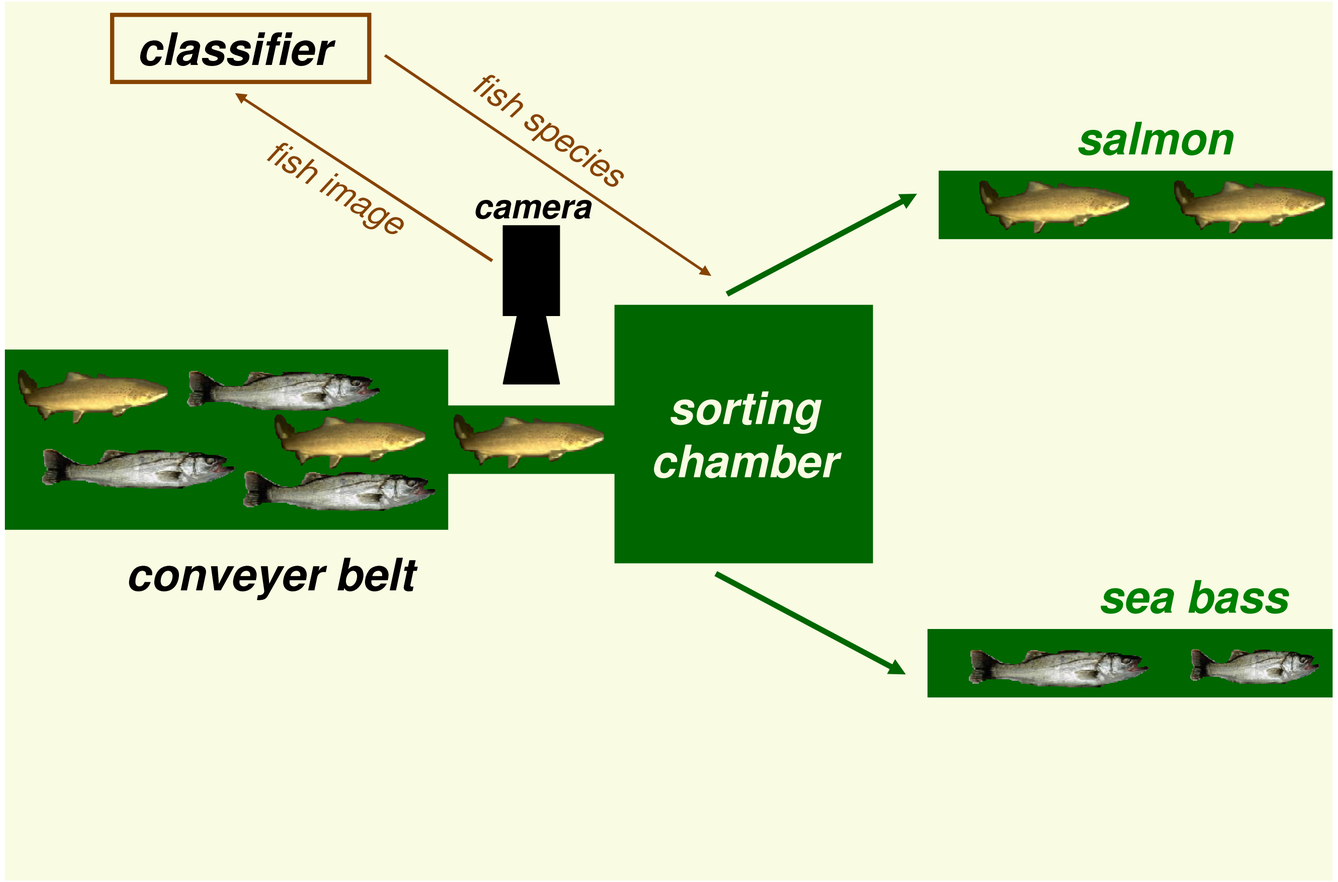


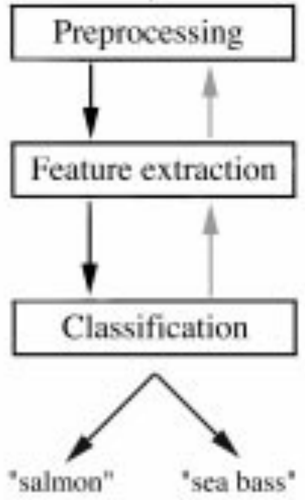
Low information gain
High entropy



High information gain
Low entropy

An example


classifier
fish species
fish image
camera
salmon
sorting chamber
conveyer belt
sea bass


Preprocessing
Feature extraction
Classification
"salmon"
"sea bass"

How to design a Classification system?

- Collect data and classify by hand

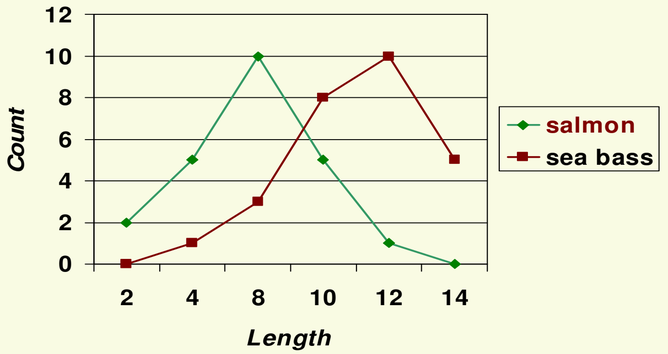
- Preprocess by segmenting fish from background

- Extract possibly discriminating features
  - length, lightness,width,number of fins,etc.
- Classifier design
  - Choose model for classifier
  - Train classifier on part of collected data (training data)
- Test classifier on the rest of collected data (test data) i.e. the data not used for training
  - Should classify new data (new fish images) well

Classifier design

- Notice salmon tends to be shorter than sea bass
- Use *fish length* as the discriminating feature
- Count number of bass and salmon of each length

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0



Fish length as discriminating feature

- Find the best length ***L*** threshold

fish length < L ⇩ ***classify as salmon***

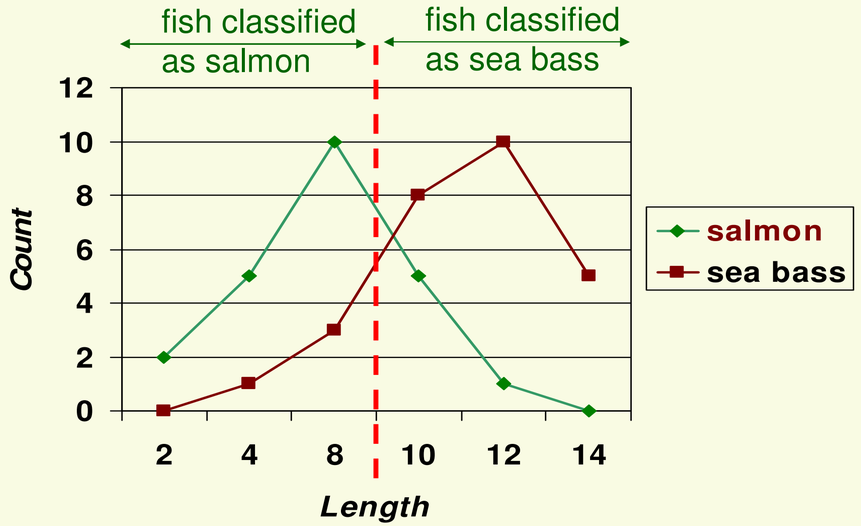
fish length > L ⇩ ***classify as sea bass***

- For example, at ***L*** = 5, misclassified:
  - 1 sea bass
  - 16 salmon

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0

- Classification error (total error): $\frac{17}{50} = 34\%$

Fish Length as discriminating feature



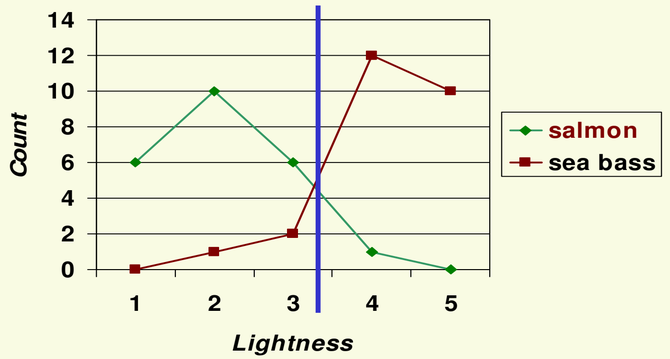
- After searching through all possible thresholds ***L***, the best ***L***= 9, and still 20% of fish is misclassified

Next Step

- Lesson learned:
  - Length is a poor feature alone!
- What to do?
  - Try another feature
  - Salmon tends to be lighter
  - Try average fish lightness

Fish lightness as discriminating feature

	1	2	3	4	5
bass	0	1	2	10	12
salmon	6	10	6	1	0



- Now fish are well separated at lightness threshold of 3.5 with classification error of 8%

Decision Stump Classifier

Use exhaustive search to find a threshold value best separating the positive class from the negative class

Given a training set:

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

For each sample $\mathbf{x}$ with its corresponding label y:

$$\mathbf{x} = (x_1, ..., x_m), x_j \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^m$$

$$y \in \{0, 1\}$$

We look for a decision stump classifier $f(\mathbf{x}, j, Th)$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

Decision Stump Classifier

Use exhaustive search to find a threshold value best separating the positive class from the negative class

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

The optimal decision stump classifier $f(\mathbf{x}, j^*, Th^*)$ minimizes the training error:

$$e_{training} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i))$$

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th)$$

$$= \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Decision Stump

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

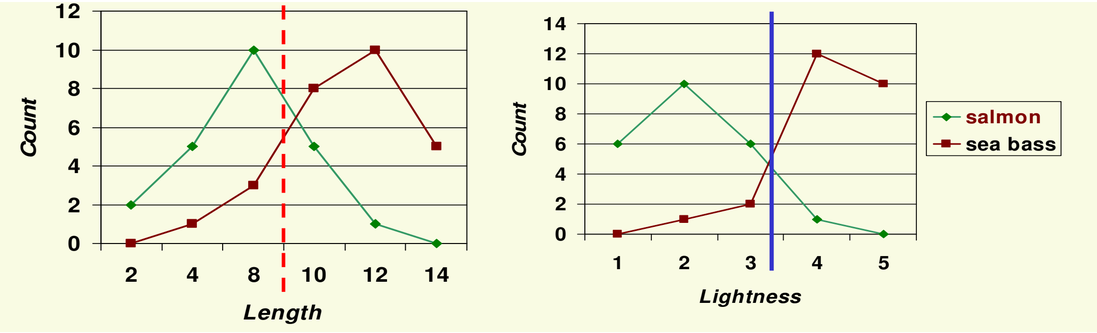
$$\mathbf{x} = (x_1, x_2), x_1, x_2 \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^2$$

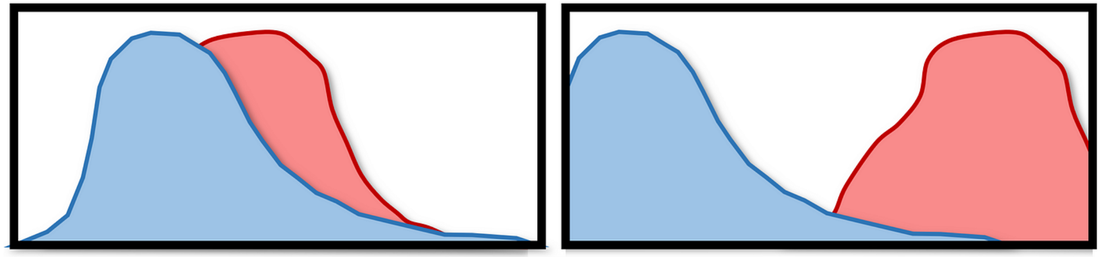
$$y \in \{0, 1\}$$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

$y = 1$

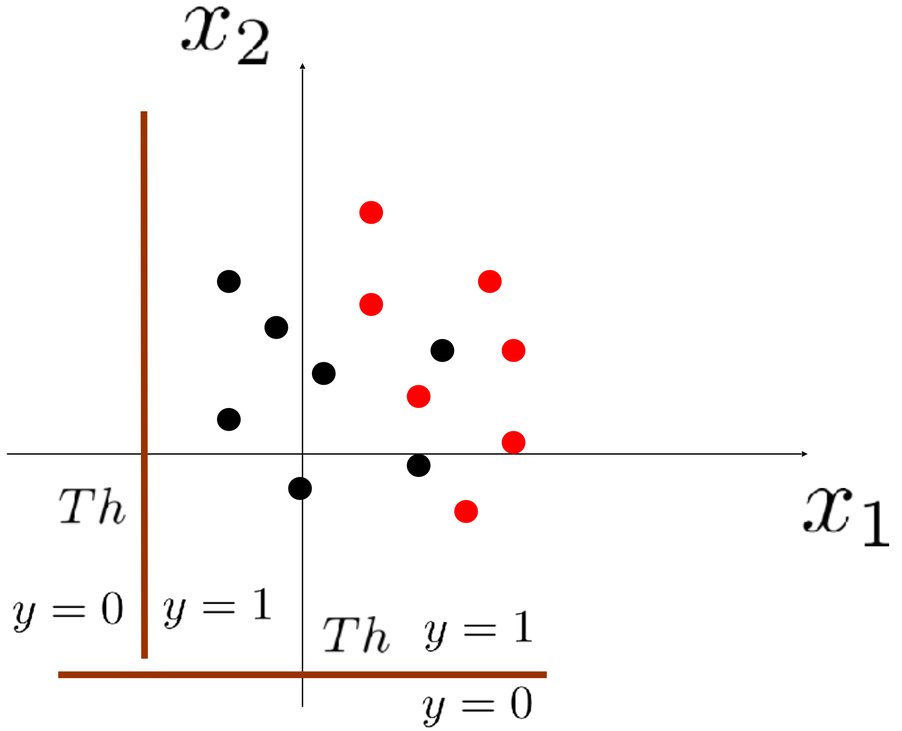
$y = 0$





Low information gain
High entropy

High information gain
Low entropy



Decision Stump - Training

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

$$\mathbf{x} = (x_1, ..., x_m), x_j \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^m \qquad y \in \{0, 1\}$$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th) = \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Pseudo-code:

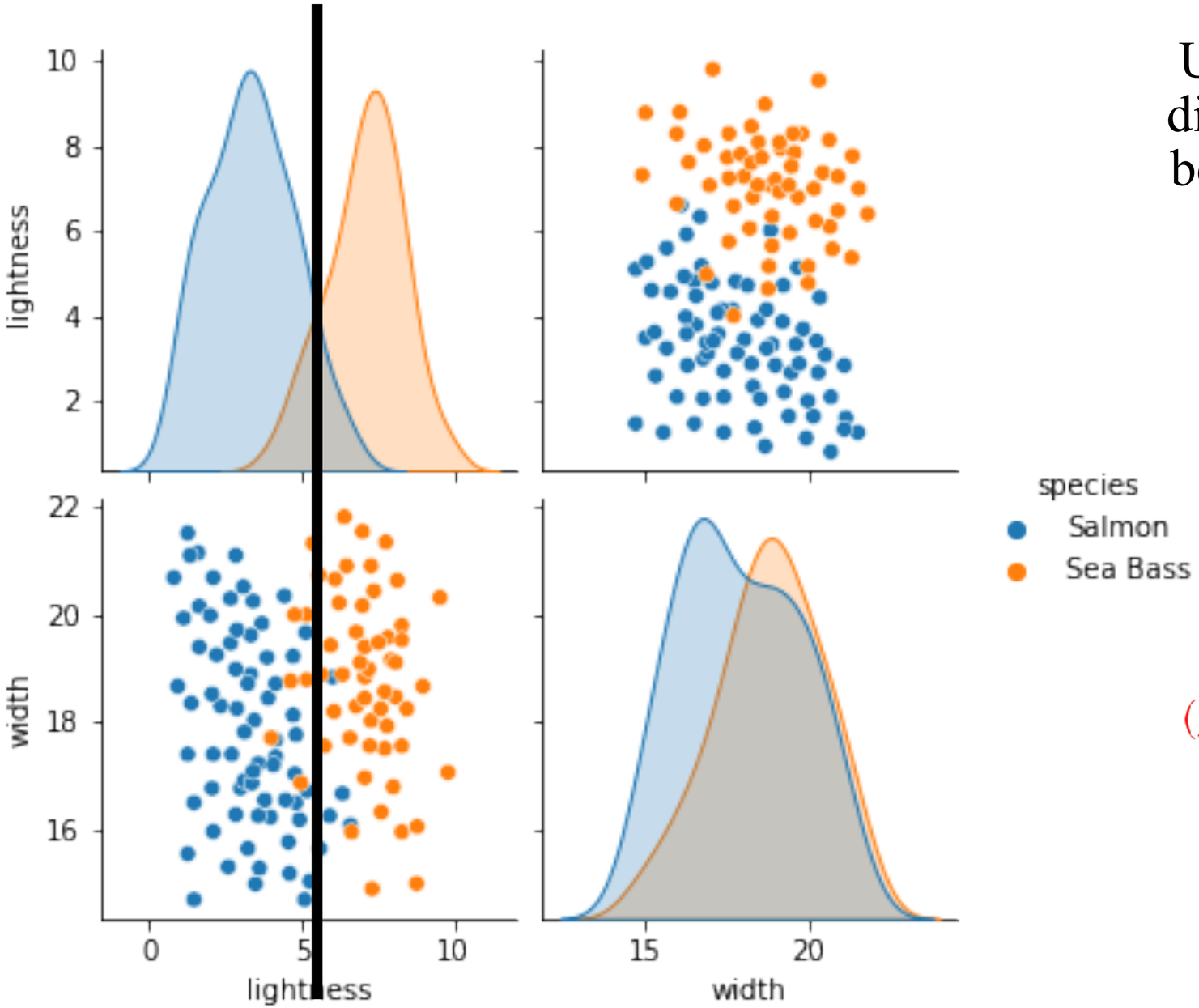
for j=1 to m

for $Th = min\_val\_j$ to max_val_j

Compute $e_{training}(j, Th) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$

Report the (j^*, Th^*) with the lowest $e_{training}(j, Th)$

Seaborn: sns.pairplot()



Decision stump

Use exhaustive search to find which dimension and which threshold value best separates the positive class from the negative class

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th)$$

$$= \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Not Secure — r2d3.us

R2
D3

http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

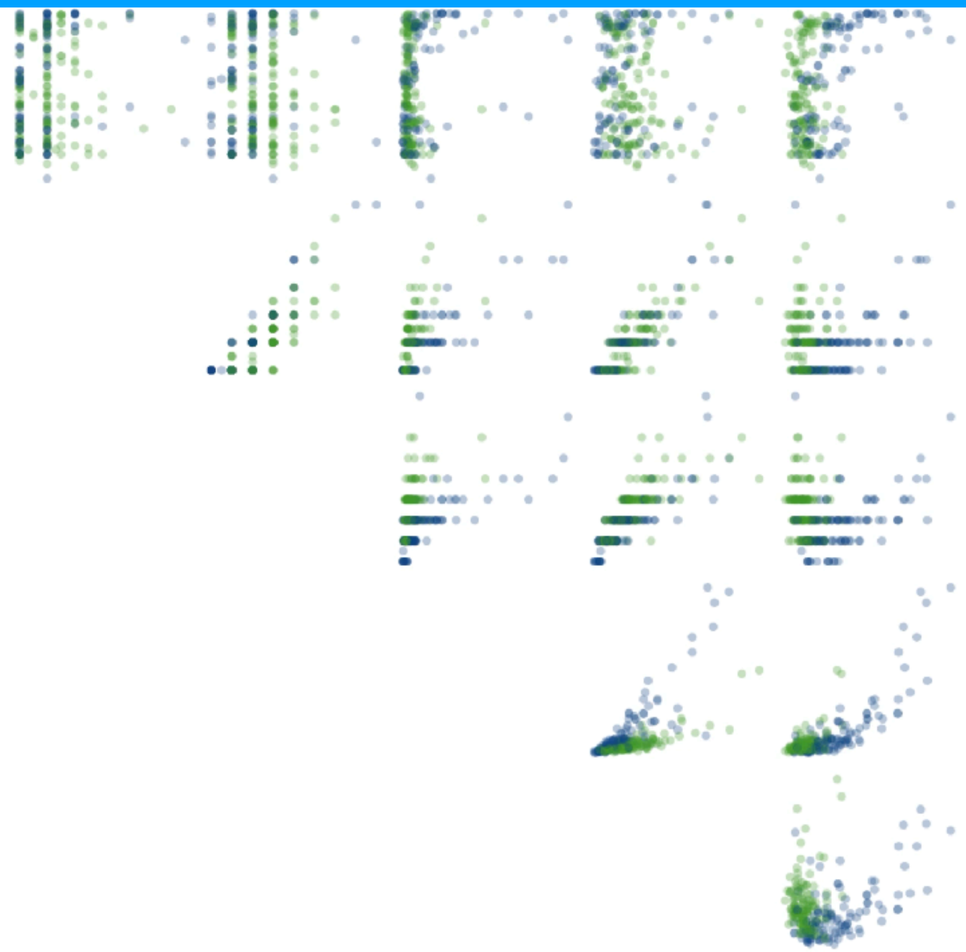
A visual introduction to machine learning

English

In machine learning, computers apply **statistical learning** techniques to automatically identify patterns in data. These techniques can be used to make highly accurate predictions.

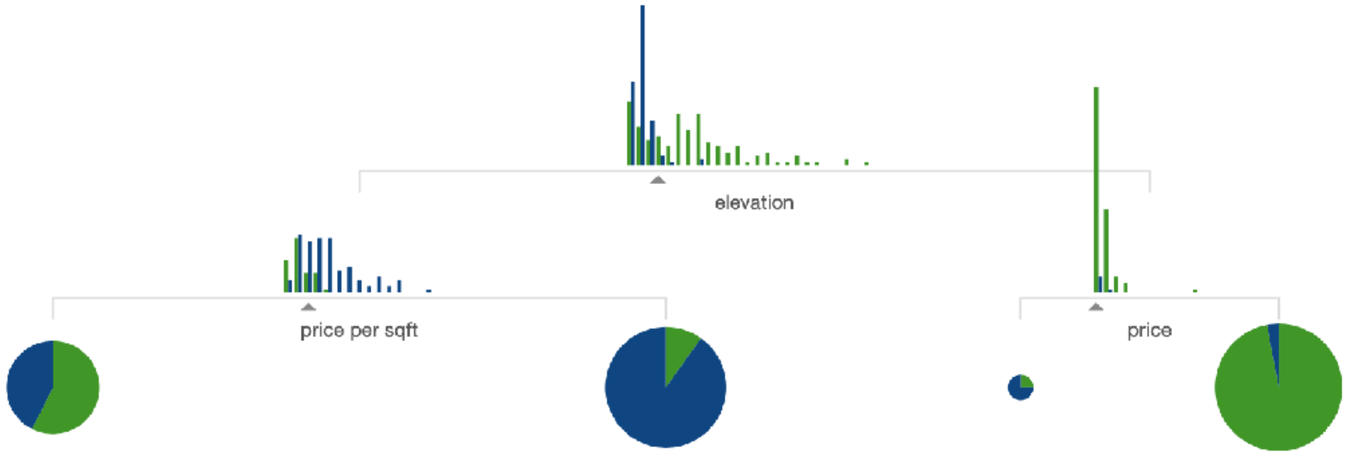
Keep scrolling. Using a data set about homes, we will create a machine learning model to distinguish homes in New York from homes in San Francisco.

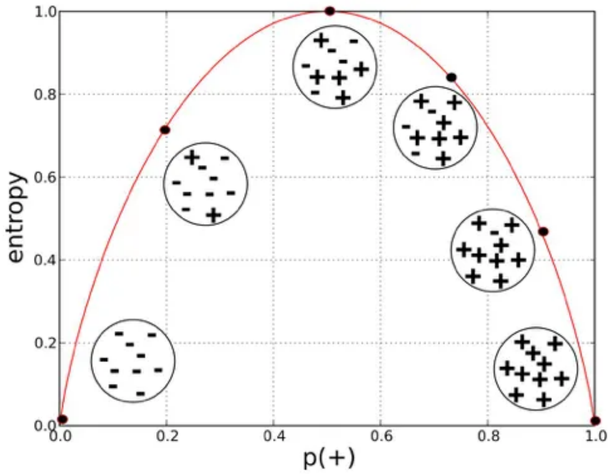
SCROLL



Growing a tree

Additional forks will add new information that can increase a tree's **prediction accuracy**.



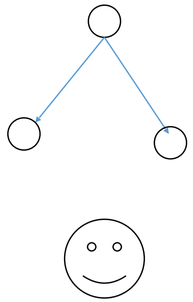


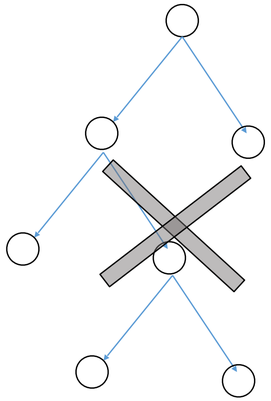
A rule of thumb when constructing a decision tree classifier

What is a good decision tree classifier?

1. For the same training error, a shallow tree is more preferred than a deep tree.

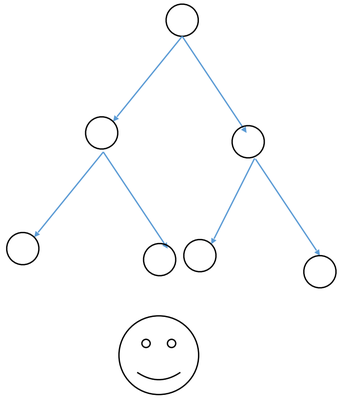
Low Complexity!

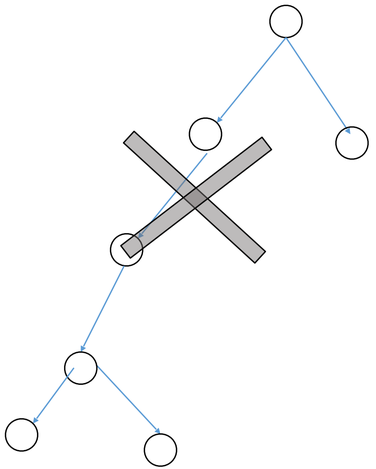




2. For the number of nodes, a balanced tree is more preferred than an unbalanced tree.

Less Overfitting!





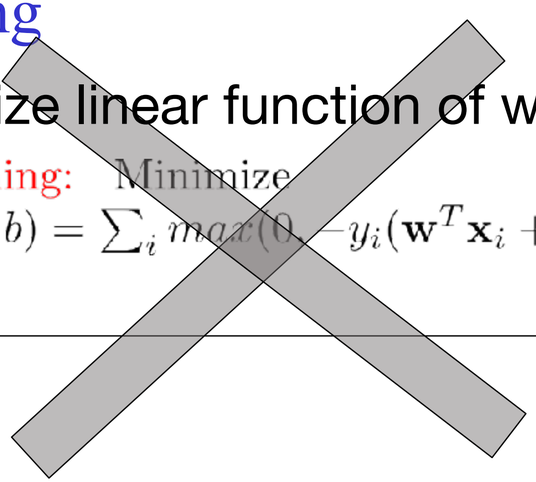
Very different than other classifiers

Training

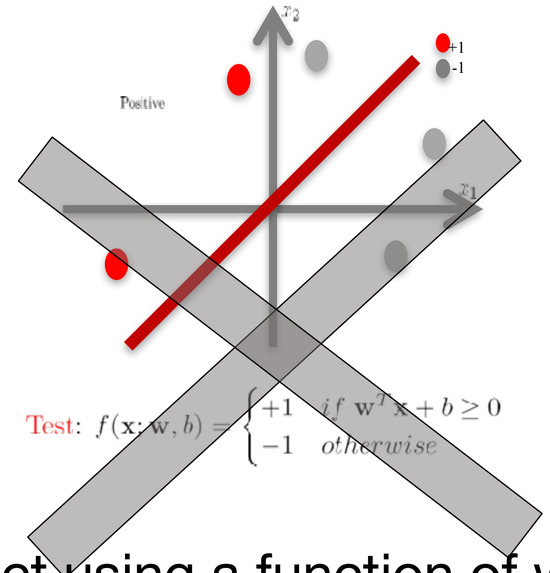
Minimize linear function of weights

Training: Minimize

$$\mathcal{L}(\mathbf{w}, b) = \sum_i max(0, -y_i(\mathbf{w}^T\mathbf{x}_i + b))$$



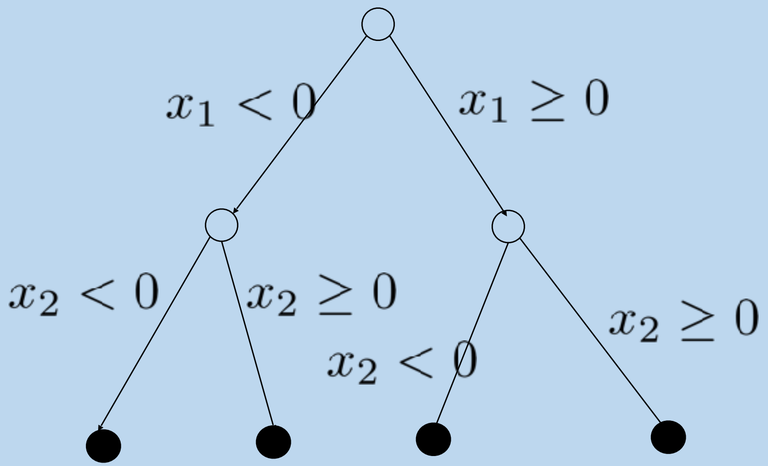
Testing



Test: $f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1 & otherwise \end{cases}$

Predict using a function of weights

Decision Tree



Training

Minimize a recursive objective function to decide where to split

Testing

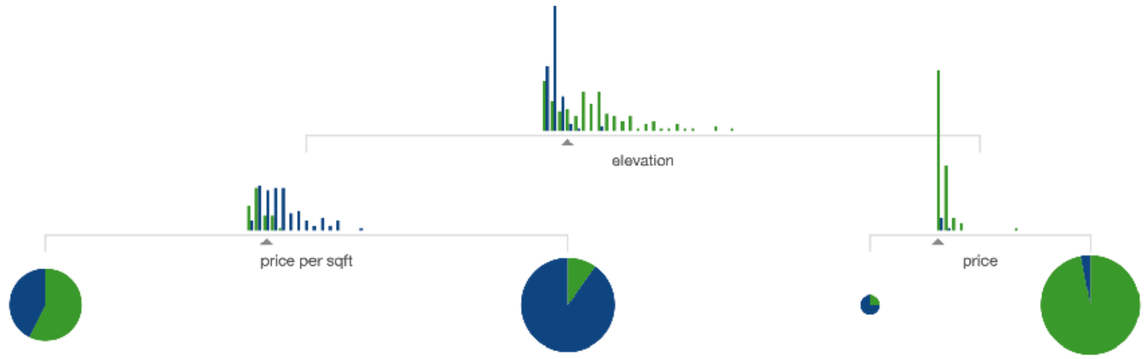
Prediction from following the tree and making a sequence of decisions until you end up at a leaf node

How to do this in polynomial time

- Learning the simplest (smallest) tree is NP-complete
  - Expand the exhaustive search of a decision stump into a tree that can arrange all stumps in any order to make a tree (no bueno)
- OK, forget that, let’s do greedy search instead

Generic Decision Tree

A recursive algorithm




```
def build_tree(S):
    # inputs: dataset S
    # outputs: a (sub)tree
    if stop_criteria(S): # typically S is pure OR len(S) < some value
        return leaf_node(S)
    else:
        # typically optimal_test() picks a
        # test `t_star` (e.g., a feature and a decision threshold)
        # that creates `partitions` of S such that
        # those partitions maximize an information metric
        # i.e., each partition is mostly just a single class
        t_star, partitions = optimal_test(S)

        children=[]
        for child_S in partitions:
            children.append( build_tree(child_S) )
        return (t_star + children)
```

Design choices for DT algos

- What kinds of attributes it can handle (discrete or continuous)
- How to choose an attribute to split (splitting criterion)
- When to stop building the tree (more else-ifs)
- What to do to the tree after its built (pruning)

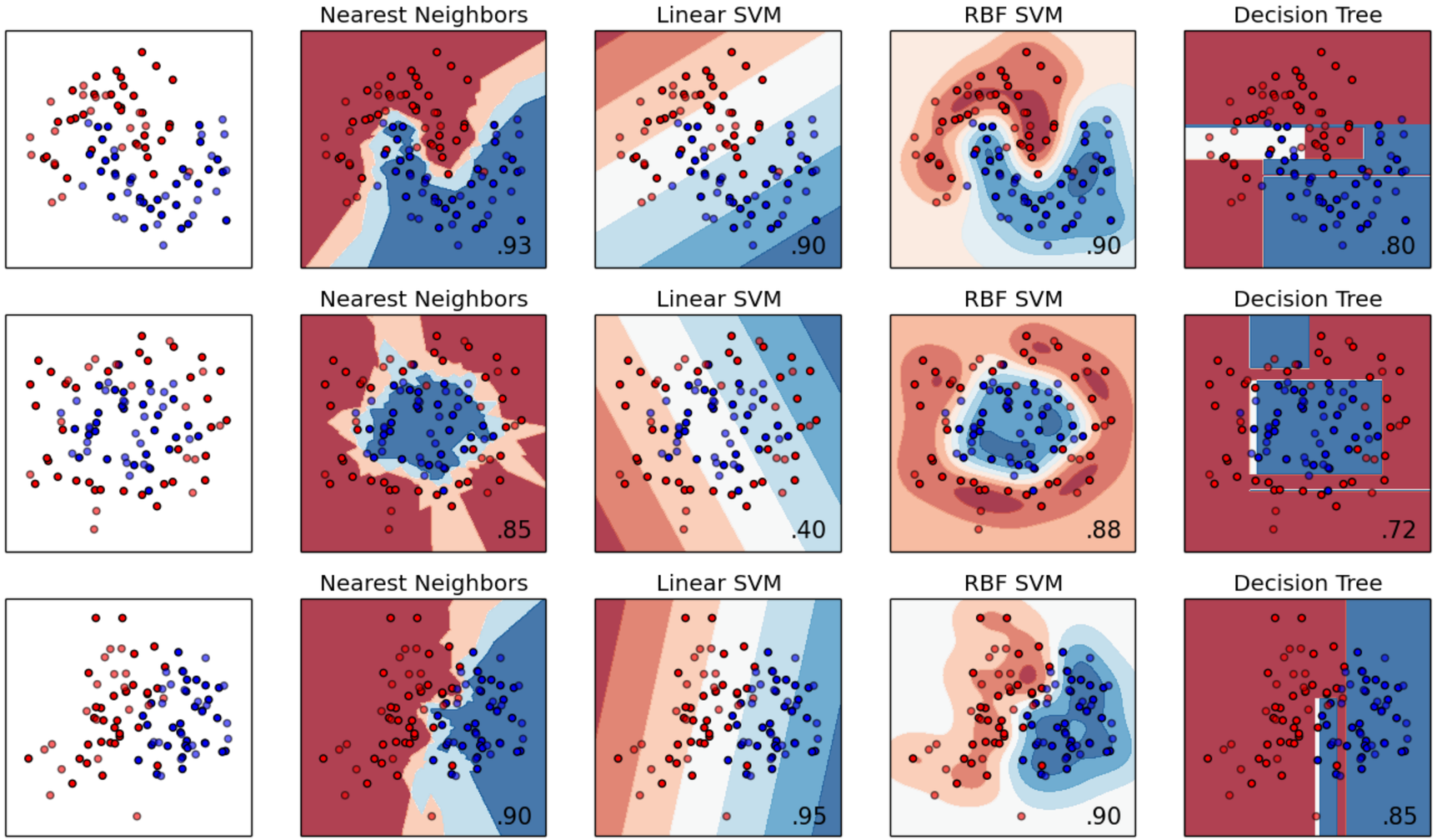
← DT forms of regularization

Time complexity of DT

- If the tree is balanced, it will be $\log_2 n$ deep therefore prediction is $O(\log_2 n)$
- Learning is $O(m \cdot n^2 \log n)$
  - The optimal binary split on continuous features is on the boundary between adjacent examples with different class labels
  - Therefore sorting the values of continuous features helps with determining a decision threshold efficiently.
  - Sorting has time complexity $O(n \log n)$. On m features, this becomes $O(m \cdot n \log n)$ If we have to re-sort on each split (dumb way, cache that!) add another $O(n)$

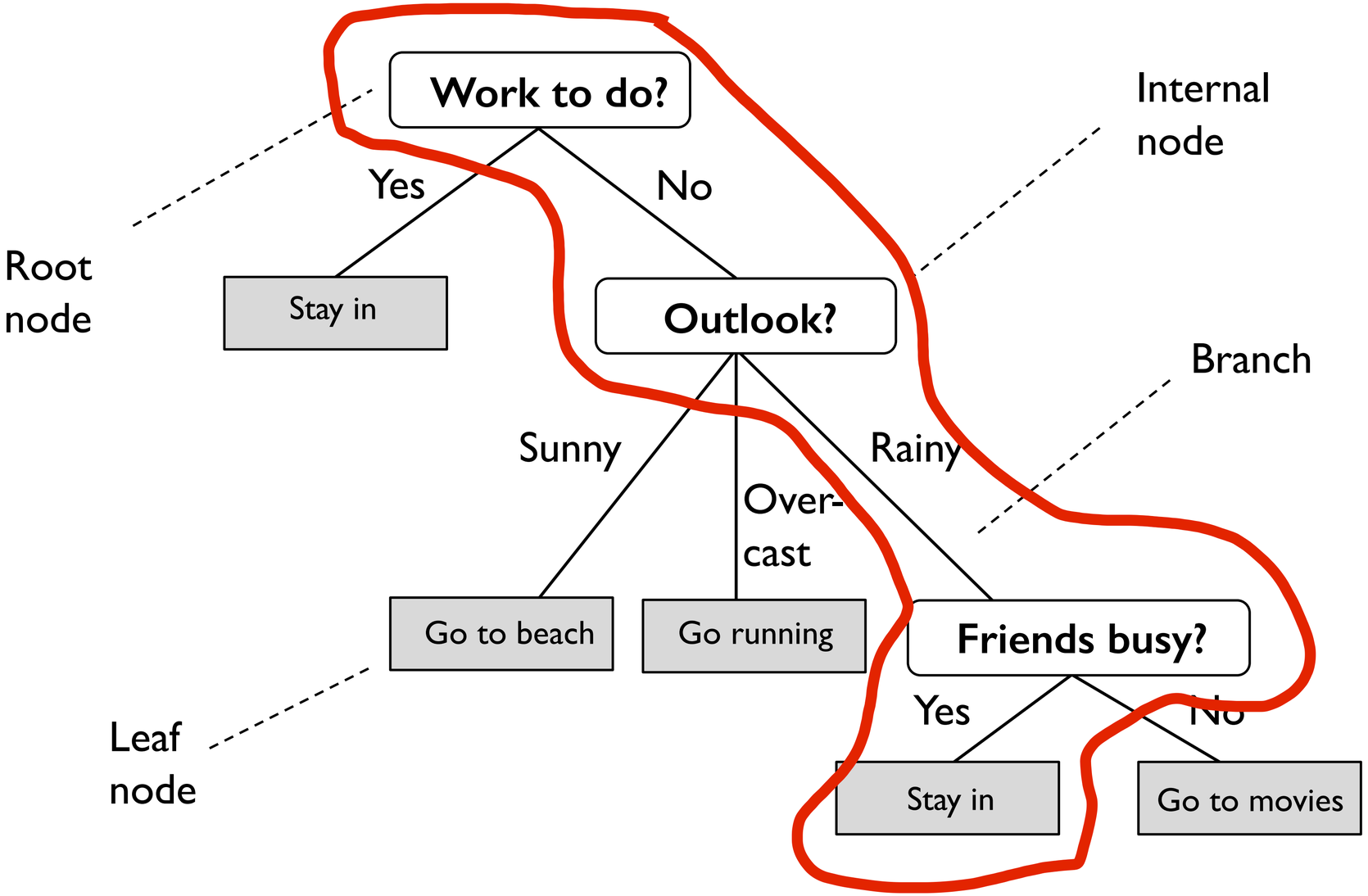
Decision trees can learn any boundary

… as long as its piecewise linear



Each leaf of a tree corresponds to an if-then rule

(Work to do? = False)∩(Outlook = Rainy?)∩(Friends busy? = Yes)∪(Work to do? = True)

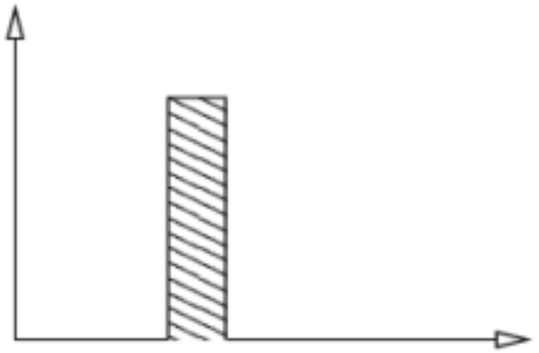


Decision trees and rules

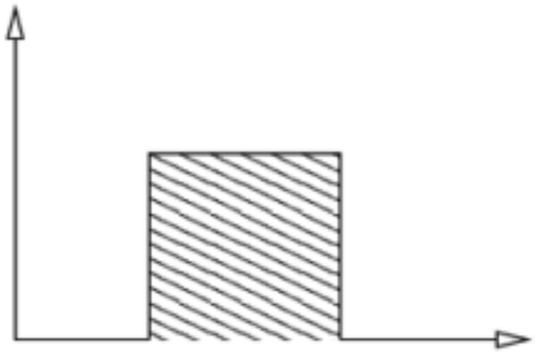
- Can go from DT to rules
- Can’t always go from rules to DT (where’s the root go?)
- Evaluating a ruleset is O(n_rules), evaluating a tree is O(tree depth)
- Rule sets could have multiple answers, DT’s only one
- Rule sets more expressive, more prone to overfitting

Training a tree

Entropy is uncertainty



A.



B.

C.

Which distribution has the lowest entropy (uncertainty)?

A.

B.

C.

Entropy

General measure for knowing the underlying uncertainty of a random variable.

Discrete random variable:

$$H(X) = -\sum_i P(X = x_i) \log P(X = x_i)$$

Continuous random variable:

$$H(X) = -\int p(x) \log p(x) dx$$

0.5	**0.5**	$H(X) = -(0.5 \times \log 0.5 + 0.5 \times \log 0.5) \approx 0.30$
0.9	**0.1**	$H(X) = -(0.9 \times \log 0.9 + 0.1 \times \log 0.1) \approx 0.14$
0.1	**0.9**	$H(X) = -(0.9 \times \log 0.9 + 0.1 \times \log 0.1) \approx 0.14$

ID3 -- Iterative Dichotomizer 3

- one of the earlier/earliest decision tree algorithms
- Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106.
- cannot handle numeric features
- no pruning, prone to overfitting
- short and wide trees (compared to CART)
- maximizing information gain/minimizing entropy
- discrete features, binary and multi-category features

After S is partitioned into subsets S_1, S_2, ..., S_t by a test B, the information gained is then

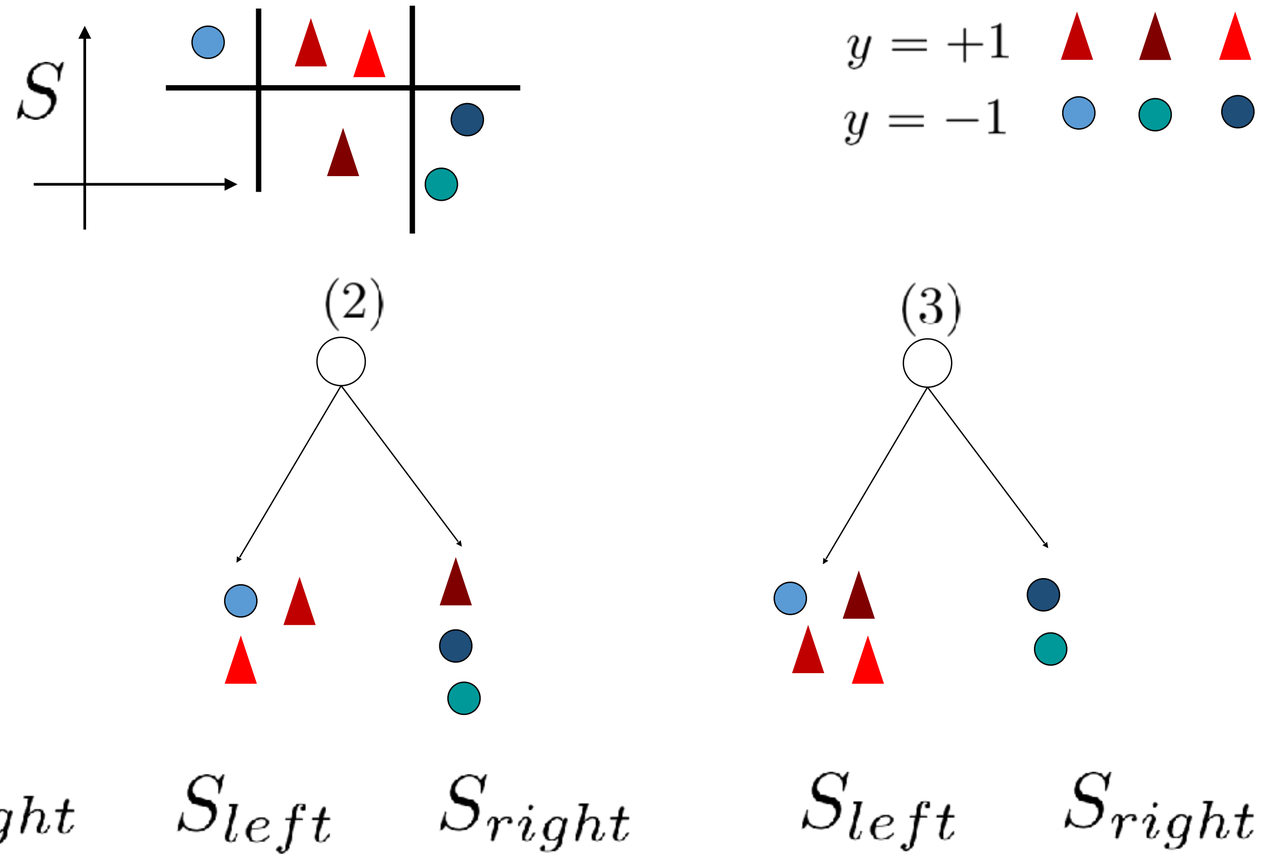
$$G(S, B) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I(S_i). \qquad \text{(C5.1.3.1)}$$

The gain criterion chooses the test B that maximizes $G(S, B)$.

Tree construction (J. Quinlan)

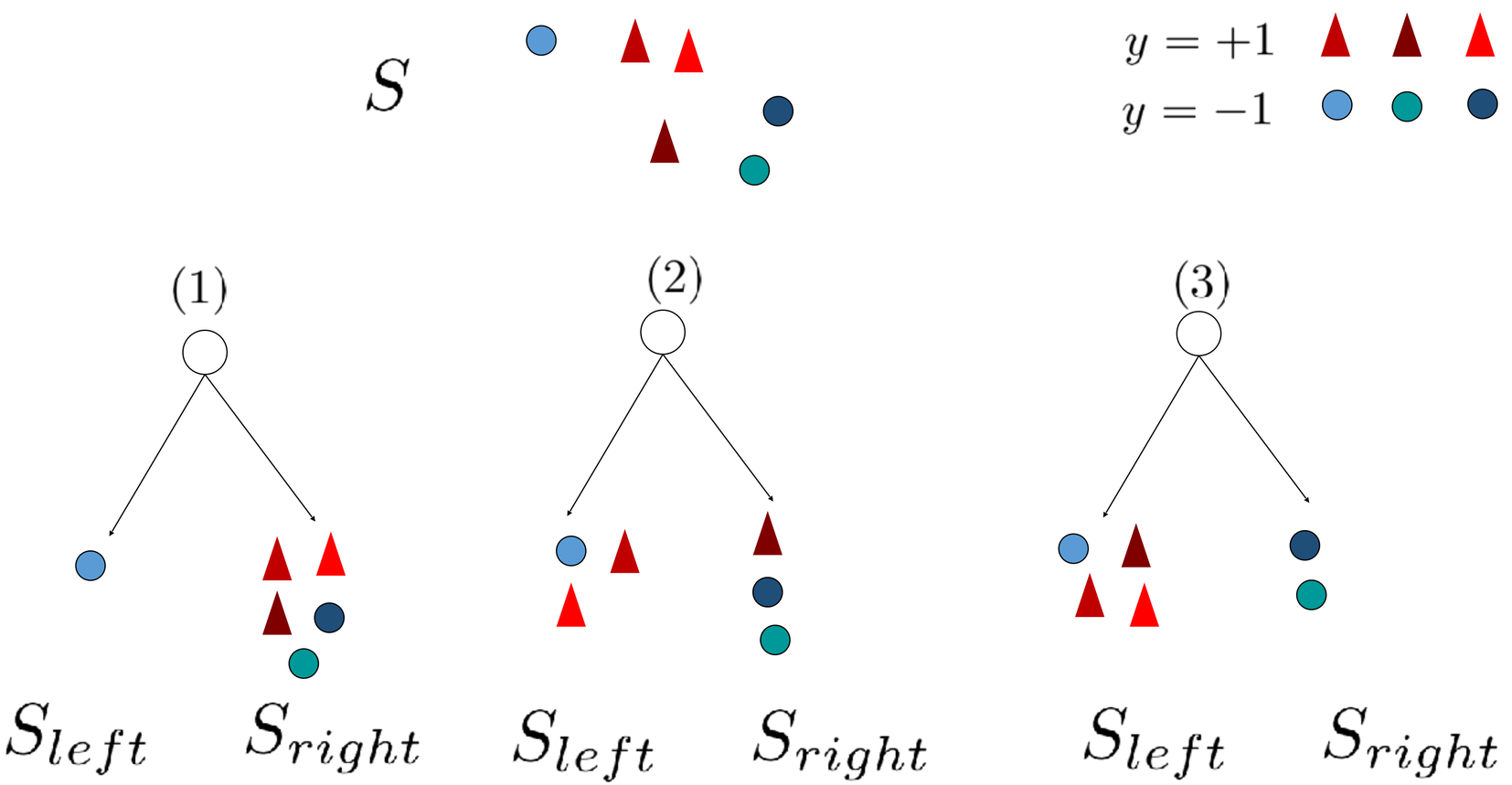
Recursively construct a tree, selecting a feature and a threshold value that maximizes the gain at each recursion.

So which split should we use first?



ID3 tree construction

$$\arg\max_{B} G(S, B) = H(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} H(S_i)$$



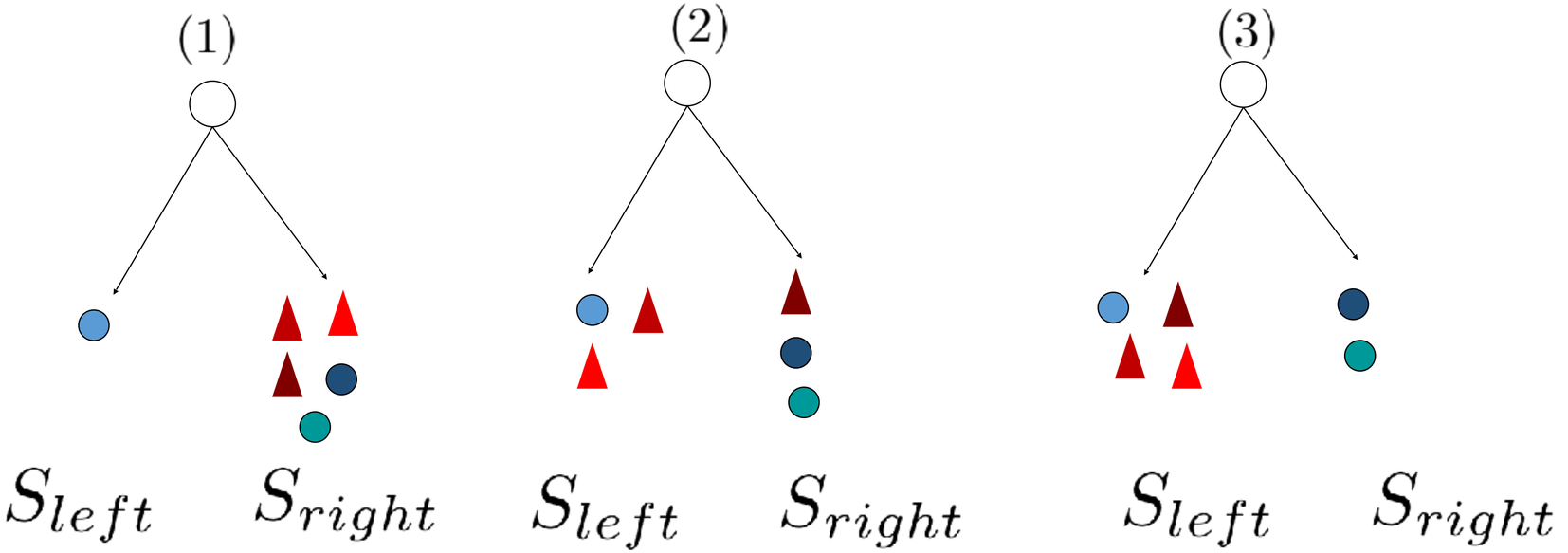
ID3 tree construction

$$\arg\max_{B} G(S,B) = H(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} H(S_i)$$

(1) $G(S_{left}) = -\frac{1}{6}(1\log_2 1 + 0\log_2 0) = 0, \quad G(S_{right}) = -\frac{5}{6}(0.4\log_2 0.4 + 0.6\log_2 0.6) = 0.81, \quad \therefore G(S,B) = H(S) - 0.81$

(2) $G(S_{left}) = -\frac{3}{6}(0.33\log_2 0.33 + 0.67\log_2 0.67) = 0.46, \quad G(S_{right}) = -\frac{3}{6}(0.67\log_2 0.67 + 0.33\log_2 0.33) = 0.46, \quad \therefore G(S,B) = H(S) - 0.92$

(3) $G(S_{left}) = -\frac{4}{6}(0.25\log_2 0.25 + 0.75\log_2 0.75) = 0.54, \quad G(S_{right}) = -\frac{2}{6}(0\log_2 0 + 1\log_2 1) = 0, \quad \therefore G(S,B) = H(S) - 0.54$



C4.5

- continuous and discrete features
- Ross Quinlan 1993, Quinlan, J. R. (1993). C4. 5: Programming for machine learning. *Morgan Kauffmann*, *38*, 48.
- continuous is very expensive, because must consider all possible ranges
- handles missing attributes (ignores them in gain compute)
- post-pruning (bottom-up pruning)
- Gain Ratio $G(S, B)/P(S, B)$

$$G(S, B) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{|S|} I(S_i).$$

$$P(S, B) = -\sum_{i=1}^{t} \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right).$$

Pruning decision trees

- Discarding one or more subtrees and replacing them with leaves simplify decision tree and that is the main task in decision tree pruning:
  - Prepruning
  - Postpruning
- C4.5 follows a postpruning approach (*pessimistic pruning)*.

Prepruning

Deciding not to divide a set of samples any further under some conditions. The stopping criterion is usually based on some statistical test, such as the χ^2-test.

Postpruning

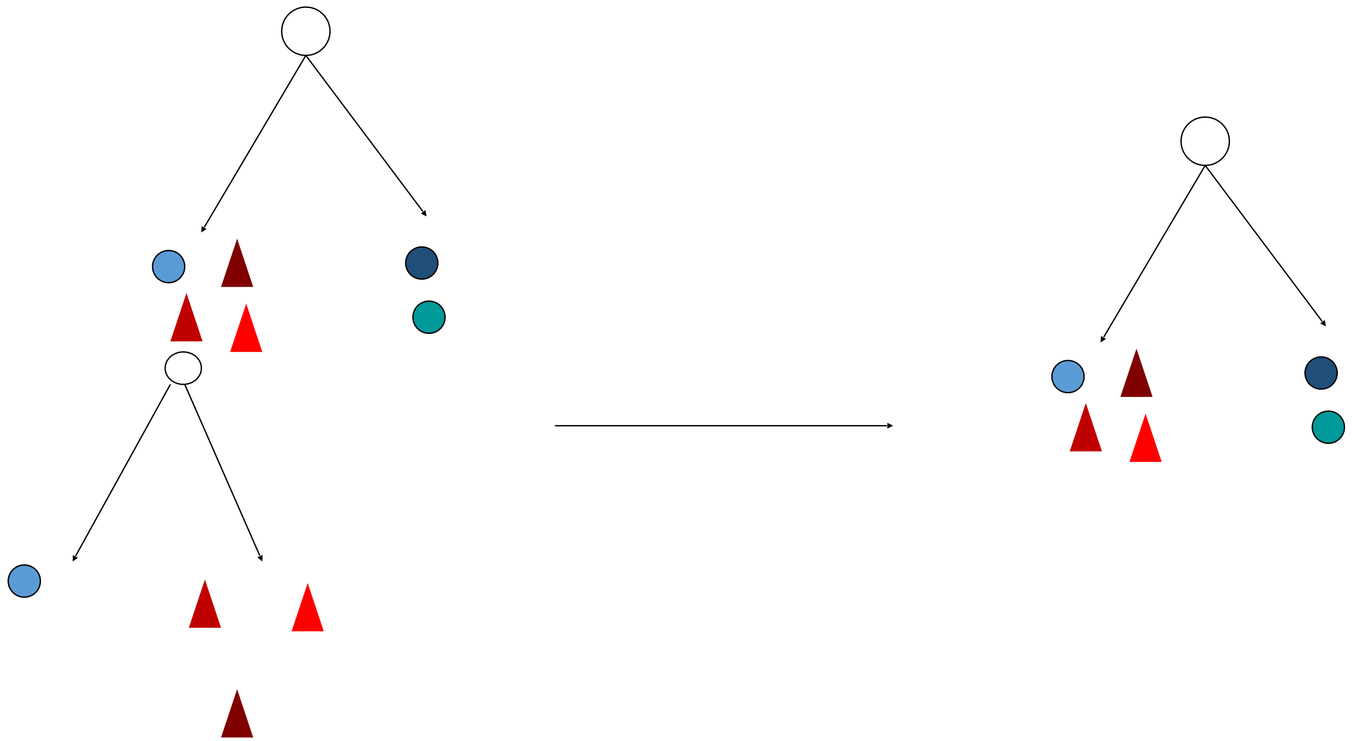
Removing retrospectively some of the tree structure using selected accuracy criteria.

Cost complexity pruning C4.5

$$R_\alpha(T) = R(T) + \alpha|T|$$

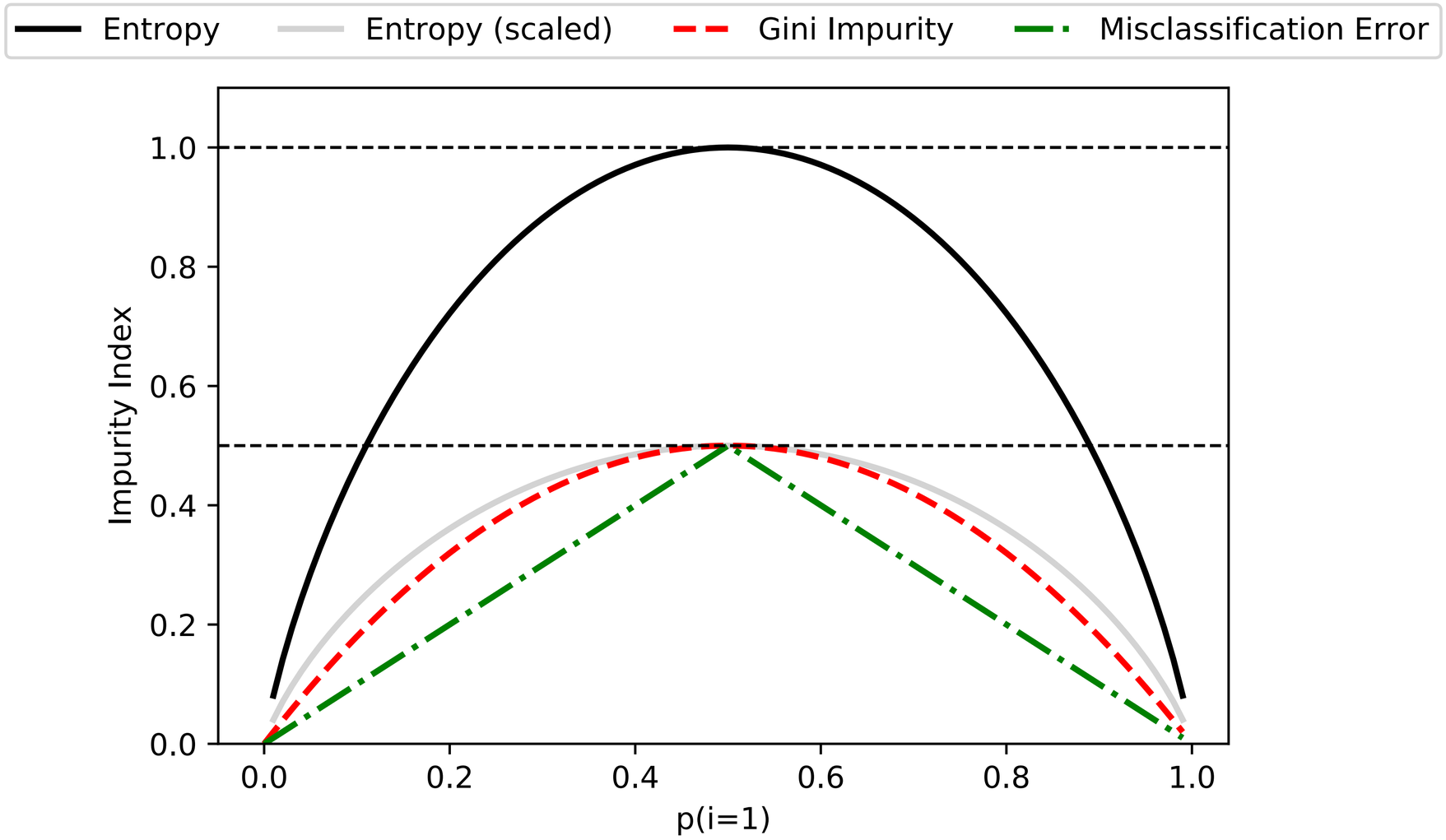
where $|T|$ is the number of terminal nodes in T and $R(T)$ is traditionally defined as the total misclassification rate of the terminal nodes. Alternatively, scikit-learn uses the total sample weighted impurity of the terminal nodes for $R(T)$. As shown above, the impurity of a node depends on the criterion. Minimal cost-complexity pruning finds the subtree of T that minimizes $R_\alpha(T)$.

The cost complexity measure of a single node is $R_\alpha(t) = R(t) + \alpha$. The branch, T_t, is defined to be a tree where node t is its root. In general, the impurity of a node is greater than the sum of impurities of its terminal nodes, $R(T_t) < R(t)$. However, the cost complexity measure of a node, t, and its branch, T_t, can be equal depending on α. We define the effective α of a node to be the value where they are equal, $R_\alpha(T_t) = R_\alpha(t)$ o $\alpha_{eff}(t) = \frac{R(t) - R(T_t)}{|T| - 1}$. A non-terminal node with the smallest value of α_{eff} is the weakest link and will be pruned. This process stops when the pruned tree's minimal α_{eff} is greater than the `ccp_alpha` parameter.



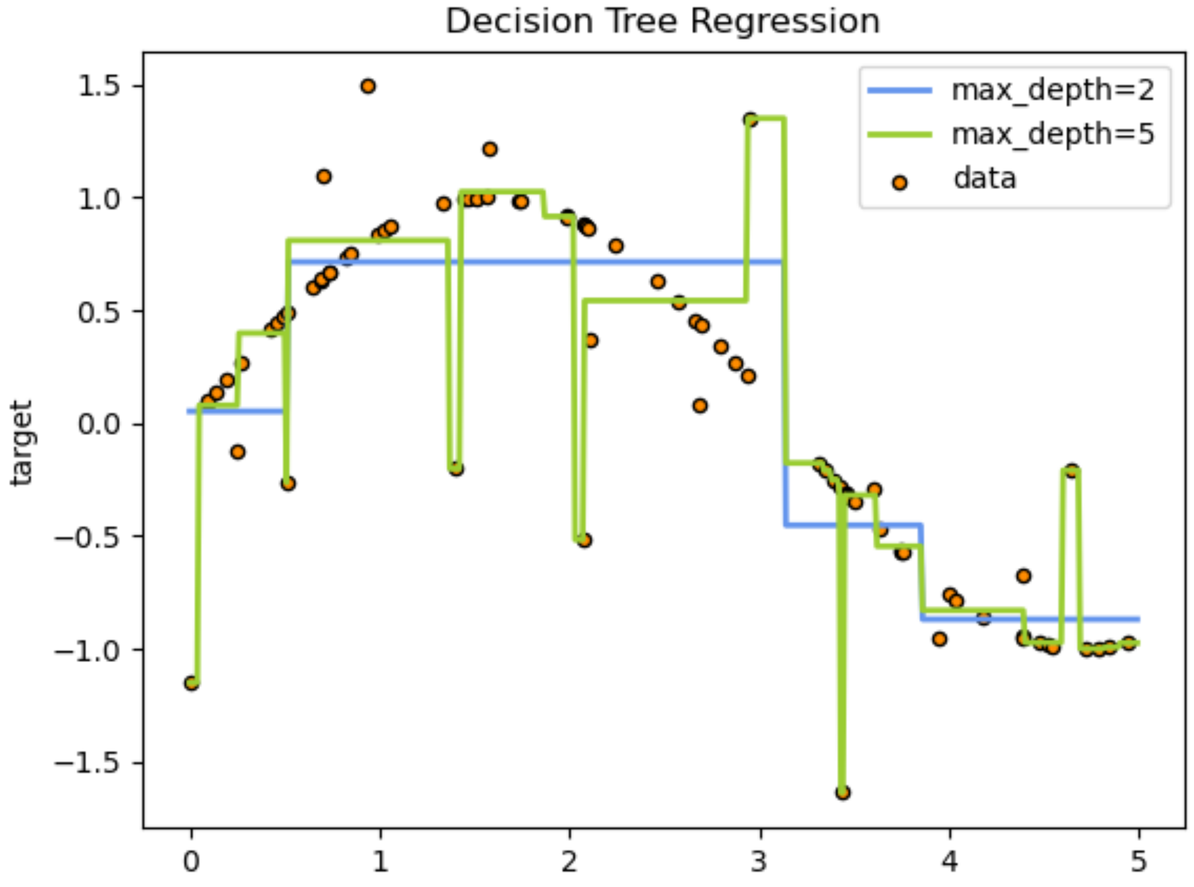
CART

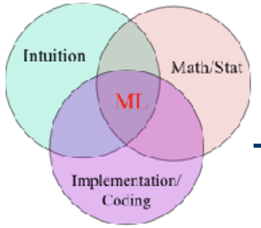
- Breiman, L. (1984). *Classification and regression trees*. Belmont, Calif: Wadsworth International Group.
- continuous and discrete features
- strictly binary splits (taller trees than ID3, C4.5)
- binary splits can generate better trees than C4.5, but tend to be larger and harder to interpret; k-attributes has a ways to create a binary partitioning
- variance reduction in regression trees
- Gini impurity, $G = \sum_{i=1}^{C} p(i) * (1 - p(i))$
- cost complexity pruning


Entropy
Entropy (scaled)
Gini Impurity
Misclassification Error
Impurity Index
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
p(i=1)

CART Regression

Use MSE as the impurity measurement





Recap: Decision Tree

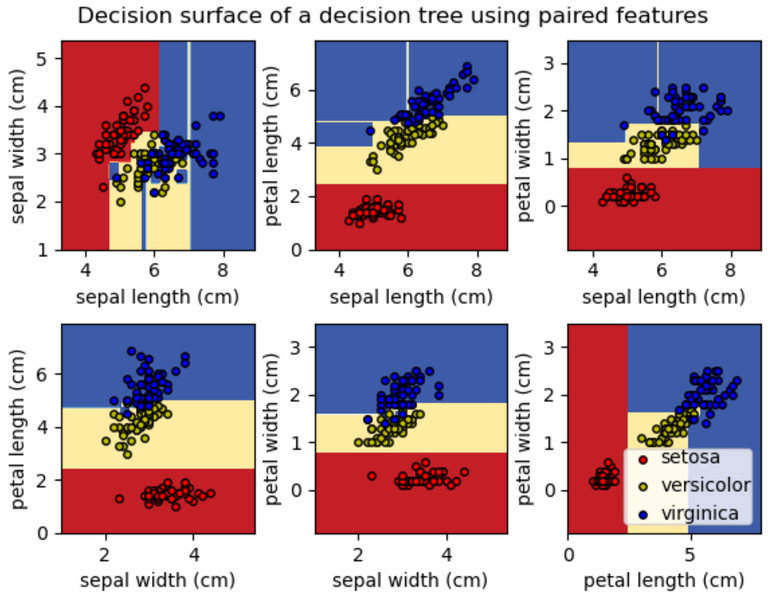
sklearn.tree.DecisionTreeClassifier

class sklearn.tree.**DecisionTreeClassifier**(*, *criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *class_weight=None*, *presort='deprecated'*, *ccp_alpha=0.0*) [source]

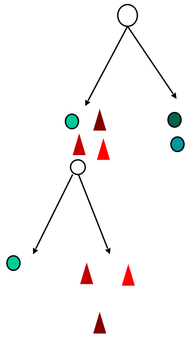
A decision tree classifier.

Read more in the User Guide.

Parameters: **criterion : *{"gini", "entropy"}, default="gini"***
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.



- Decision tree classifier is one of the most widely used classifiers in machine learning.
- It is a non-parametric model that can grow deep.
- Its key spirit is about divide-and-conquer.
- It has a nice balance between model complexity and classification power.



Math:

$$f^* = \arg max_f \quad gain(S^{(f)}_{left}) + gain(S^{(f)}_{right}) - gain(S)$$

$$gain(S) = -|S| \times Entropy(Y_S)$$

Decision trees

Advantages

- Interpretable
- Non-parametric method
- Able to fit arbitrary decision boundaries (not just linear!)
- Don’t need to scale features to match each other
- Can be combined with techniques to make it better (like bagging and boosting)

Disadvantages

- Easy to overfit
- Needs some kind of pruning and tree-growth limits to avoid overfitting
- For regression trees, the output is bounded by the limits of the training samples