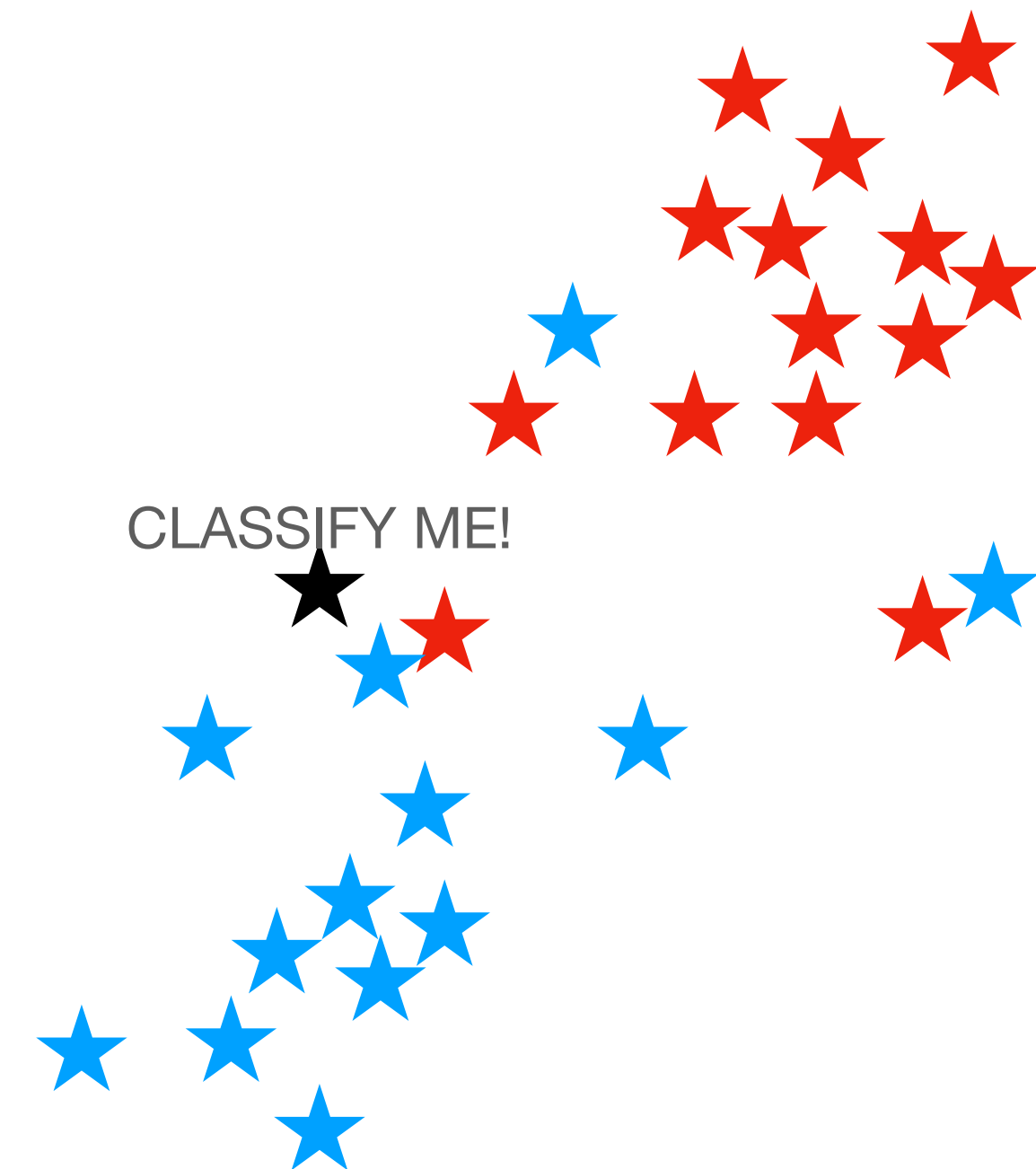


Lecture 6 pre-video

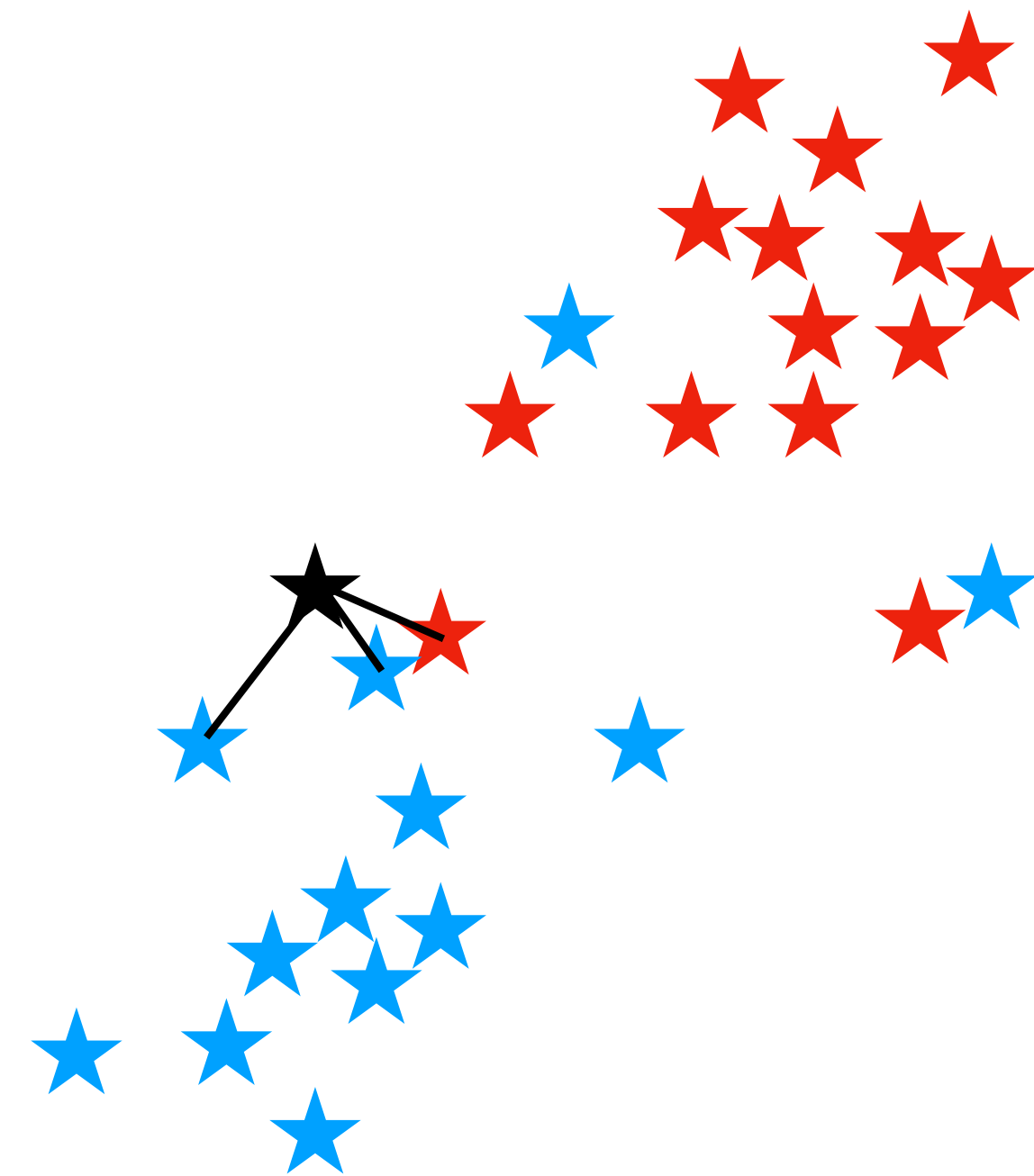
Nearest neighbors

Simple decision making

Nearest neighbors



Nearest neighbors



Nearest Neighbor Algorithm

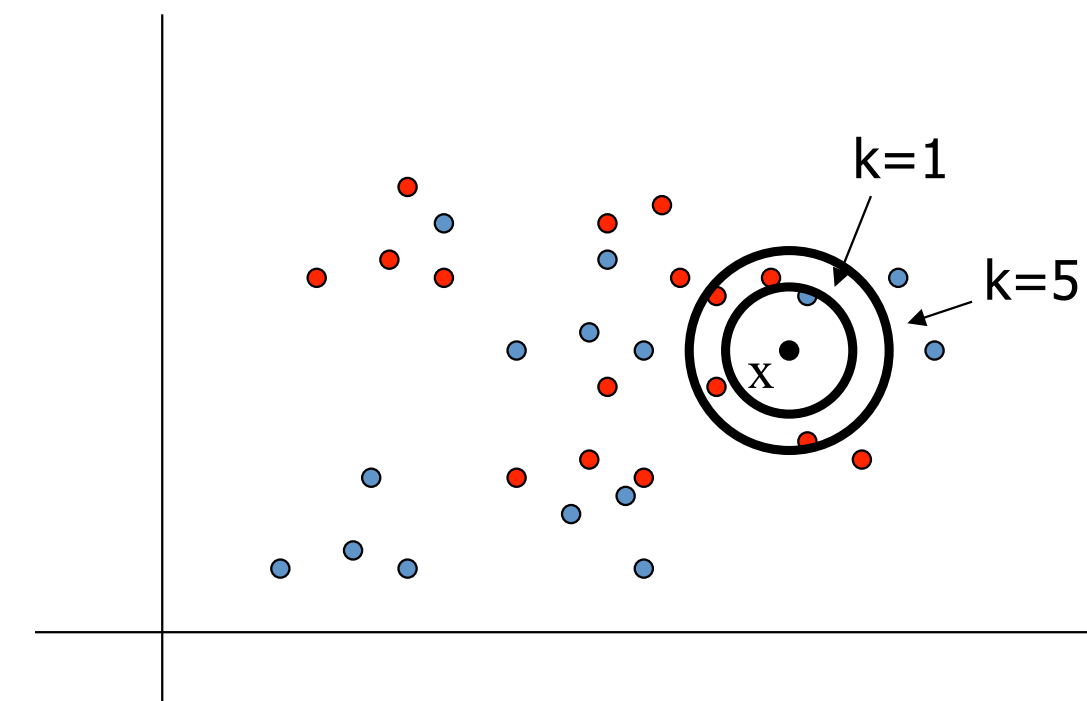
- Learning Algorithm:
 - Store training examples
- Prediction Algorithm:
 - To classify a new example \mathbf{x} by finding the training example (\mathbf{x}^i, y^i) that is *nearest* to \mathbf{x}
 - Guess the class $y = y^i$

Non parametric
Data based!
More prone to overfitting

Vs parametric
Model - based!
Prone to underfitting

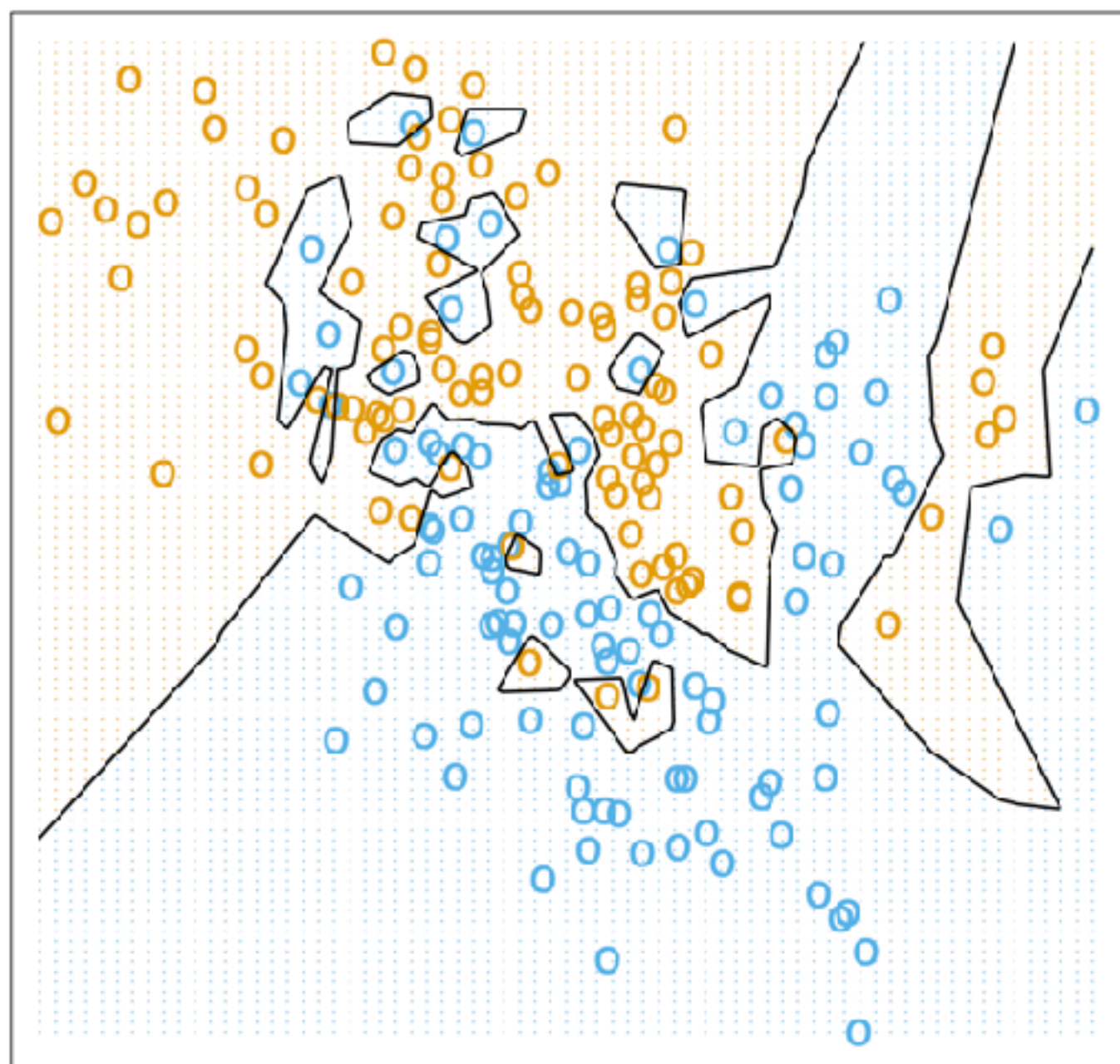
K-Nearest Neighbor Methods

- To classify a new input vector \mathbf{x} , examine the k -closest training data points to \mathbf{x} and assign the object to the most frequently occurring class

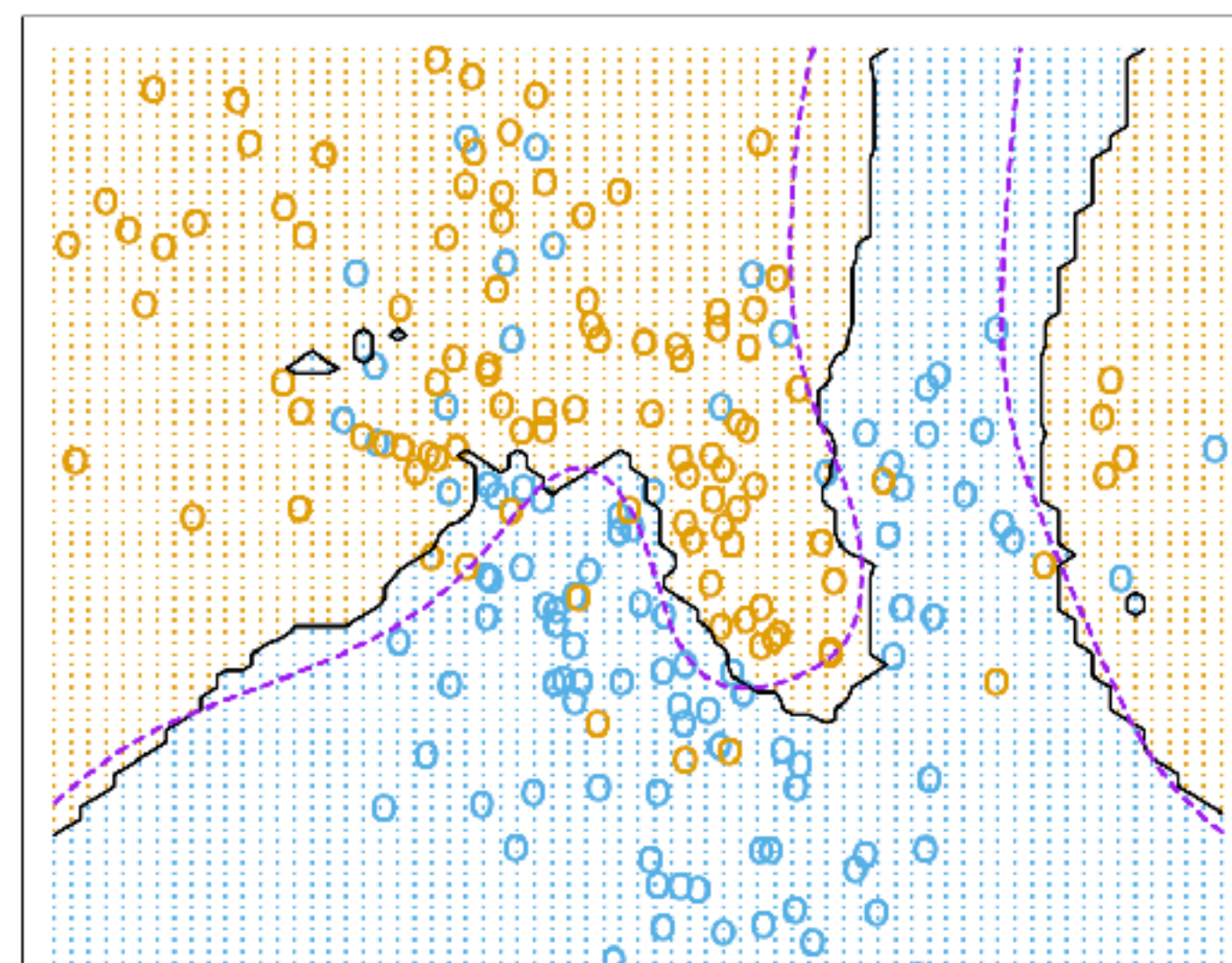


common values for k : 3, 5

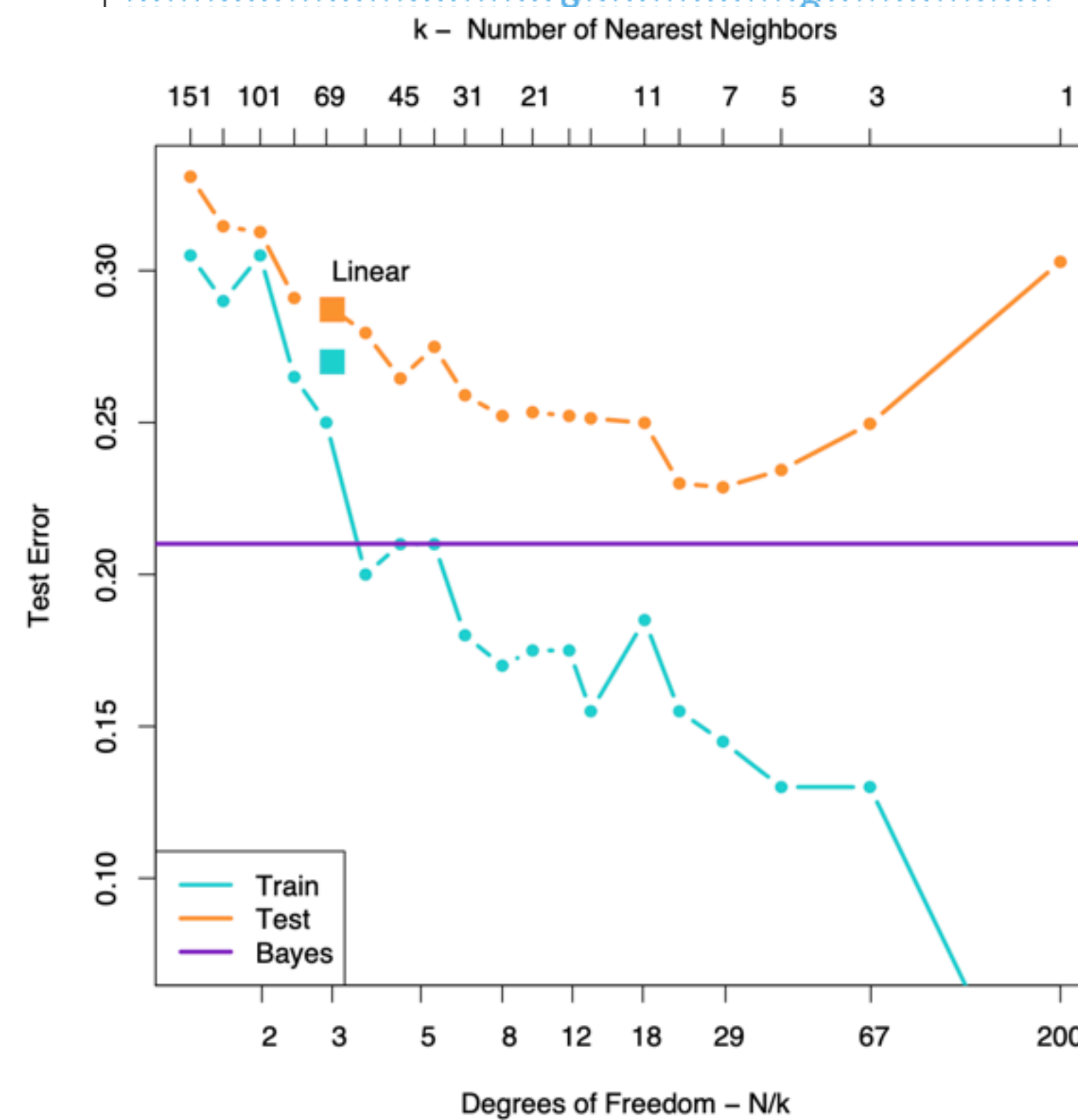
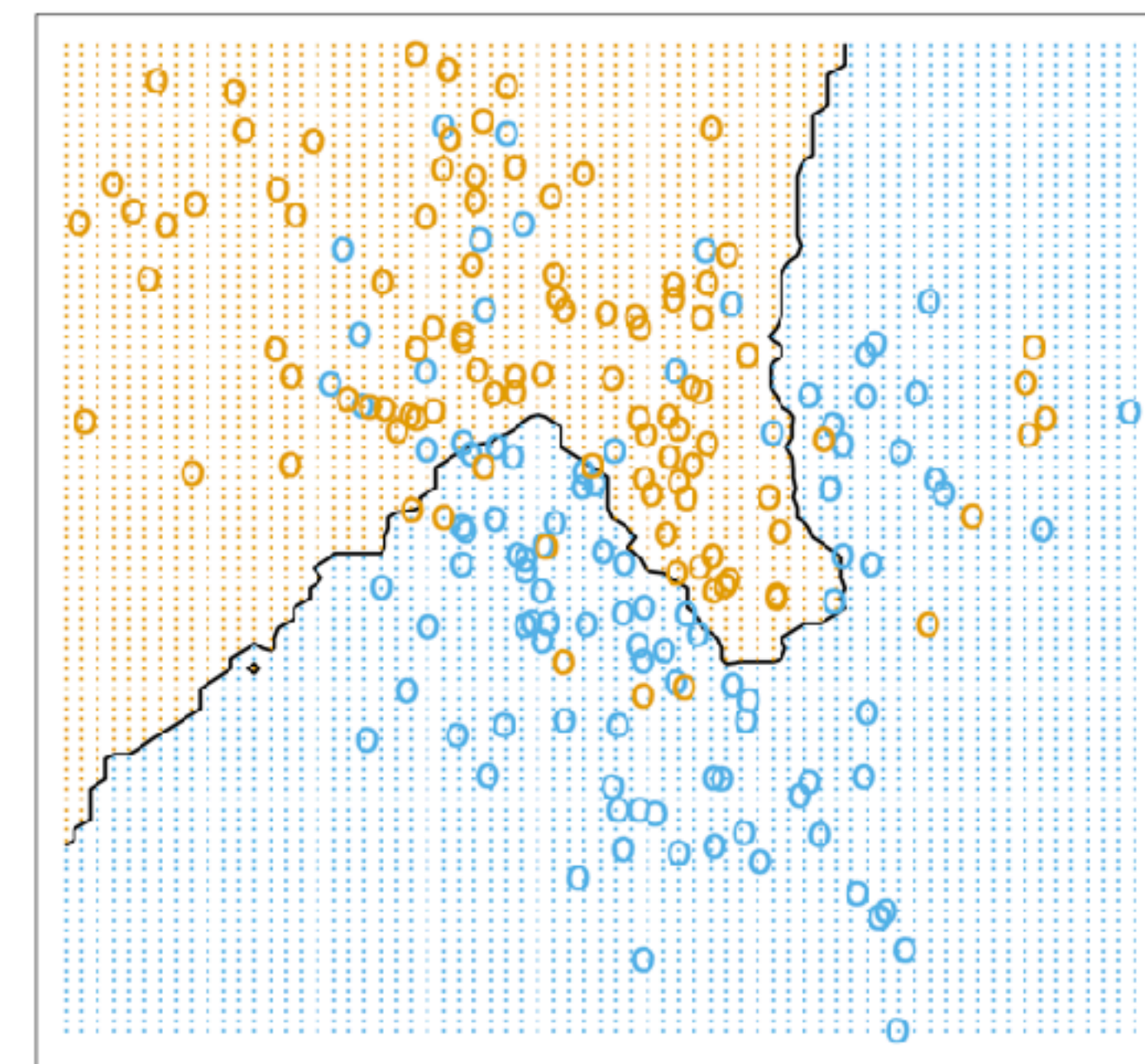
1-Nearest Neighbor Classifier



7-Nearest Neighbors



15-Nearest Neighbor Classifier



Nearest Neighbor

When to Consider

- Instance map to points in R^n
- Less than 20 attributes per instance
- Lots of training data

Advantages

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages

- Slow at query time
- Easily fooled by irrelevant attributes

Training
Time $O(1)$
Memory $O(n)$

Testing
Time $O(n)$
Memory $O(n)$

Issues

- Distance measure
 - Most common: Euclidean
- Choosing k
 - Increasing k reduces variance, increases bias
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.

A **metric** on a set X is a **function** (called *distance function* or simply **distance**)

$$d : X \times X \rightarrow [0, \infty),$$

where $[0, \infty)$ is the set of non-negative **real numbers** and for all $x, y, z \in X$, the following three axioms are satisfied:

1. $d(x, y) = 0 \Leftrightarrow x = y$ **identity of indiscernibles**
2. $d(x, y) = d(y, x)$ **symmetry**
3. $d(x, y) \leq d(x, z) + d(z, y)$ **subadditivity** or **triangle inequality**

We kinda skipped this back when we did error metrics... ML is full of metric spaces!

Common ML metrics:

Euclidean (L2) *

Manhattan (L1)

Minkowski (Lp norm for arbitrary p)

Mahalanobis (generalization of z-score)

Hamming (binary feature distance) *

* often used in k-NN implementations

Standardization

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.

The estimate for the standard deviation of x_k :

$$\hat{\sigma}_k = \left(\frac{1}{n} \sum_{i=1}^n (x_k^i - \bar{x}_k)^2 \right)^{\frac{1}{2}}$$

where \bar{x}_k is the sample mean:

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^n x_k^i$$

Weighted Euclidean distance

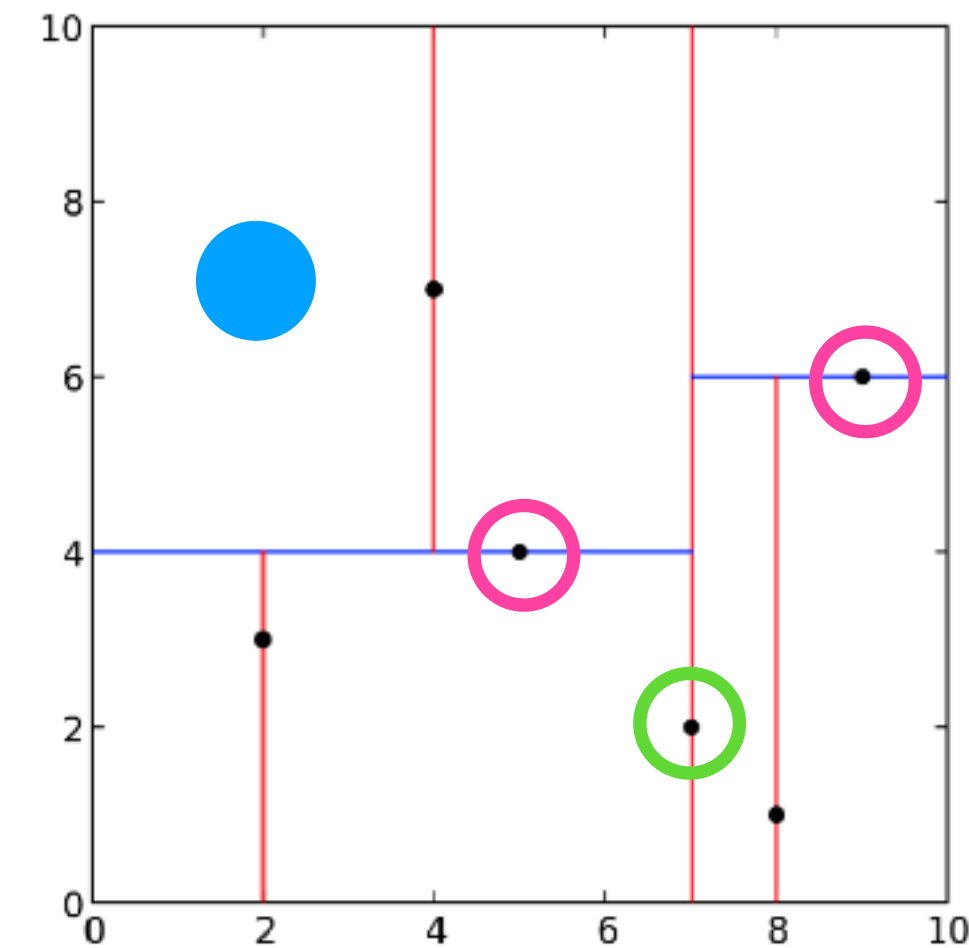
Finally, if we have some idea of the relative importance of each variable, we can weight them:

$$d_{WE}(i, j) = \left(\sum_{k=1}^p w_k (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}}$$

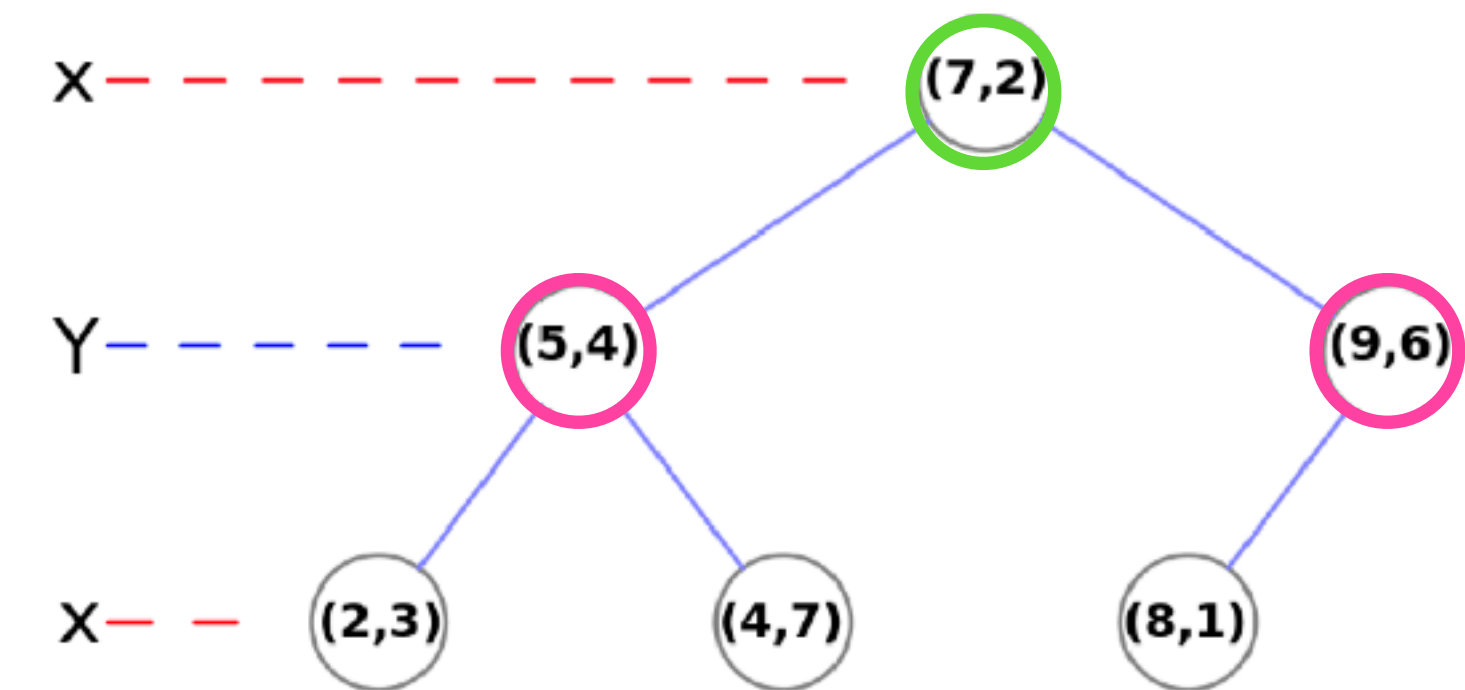
Nearest neighbor problem

- **Problem:** given sample $S = ((x_1, y_1), \dots, (x_m, y_m))$, find the nearest neighbor of test point x .
- general problem extensively studied in computer science.
- exact vs. approximate algorithms.
- dimensionality N crucial.
- better algorithms for small intrinsic dimension (e.g., limited doubling dimension).

[Slides from Mehryar Mohri]



KD-trees



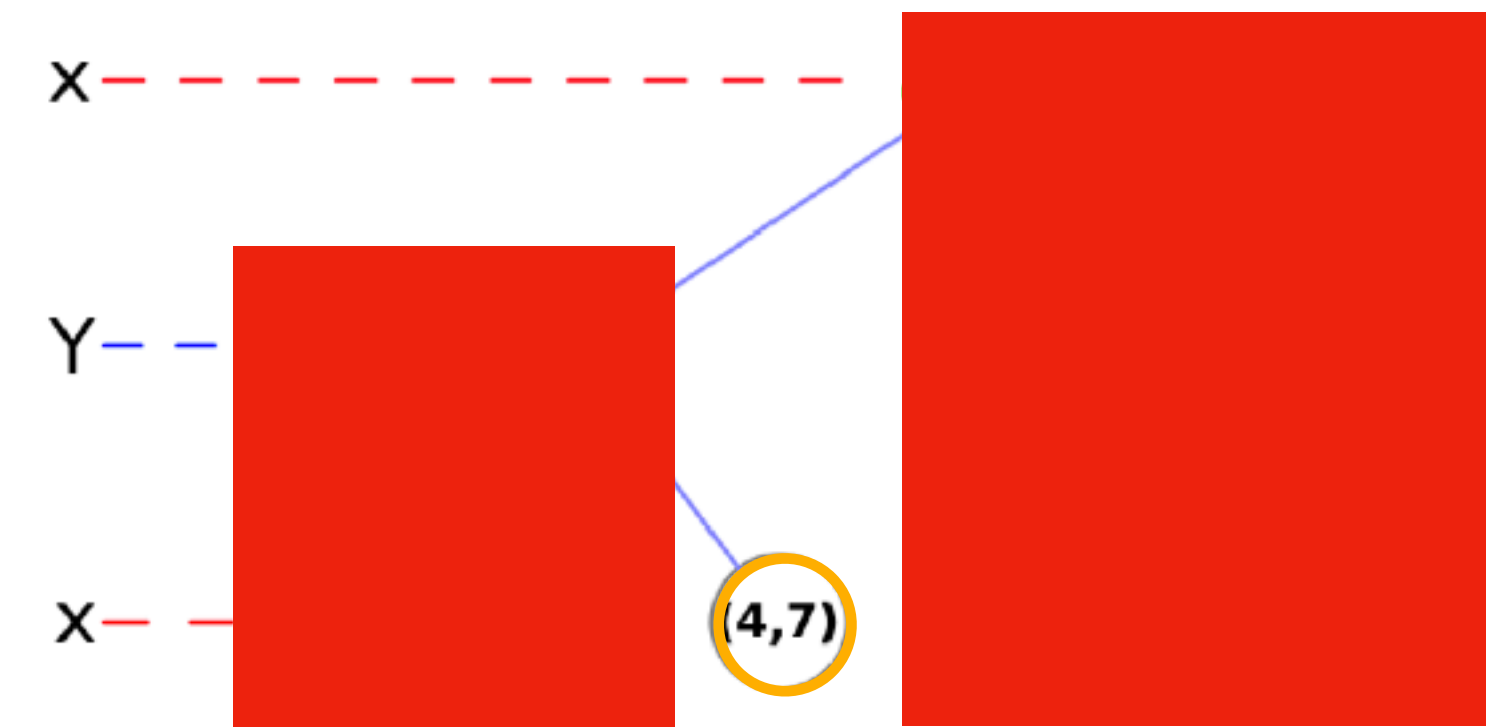
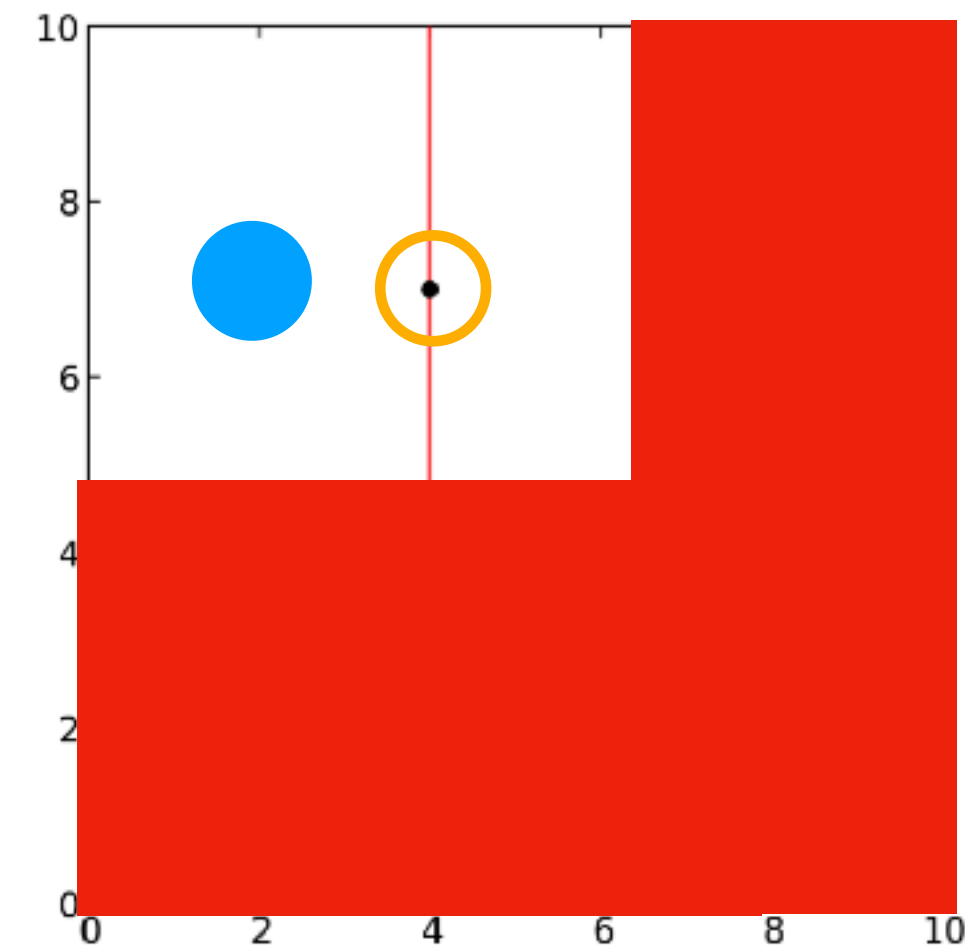
$O(n \log n)$ to build the tree!

Nearest neighbor problem

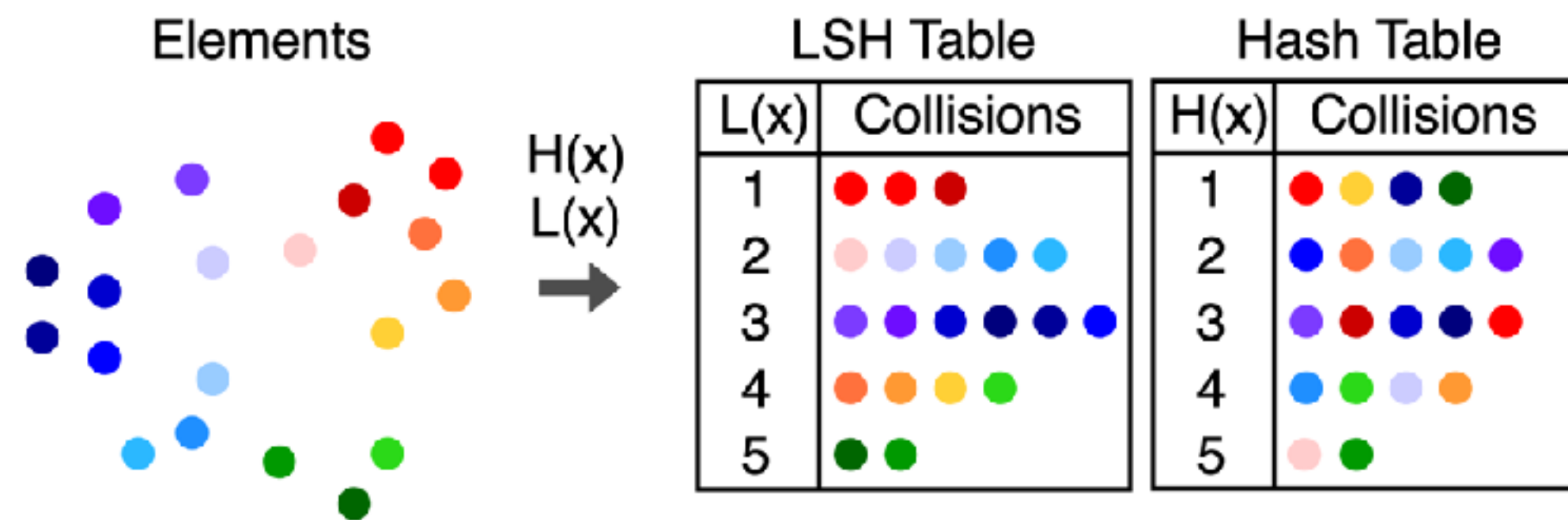
- **Problem:** given sample $S = ((x_1, y_1), \dots, (x_m, y_m))$, find the nearest neighbor of test point x .
- general problem extensively studied in computer science.
 - exact vs. approximate algorithms.
 - dimensionality N crucial.
 - better algorithms for small intrinsic dimension (e.g., limited doubling dimension).

[Slides from Mehryar Mohri]

KD-trees



Locally sensitive hashing

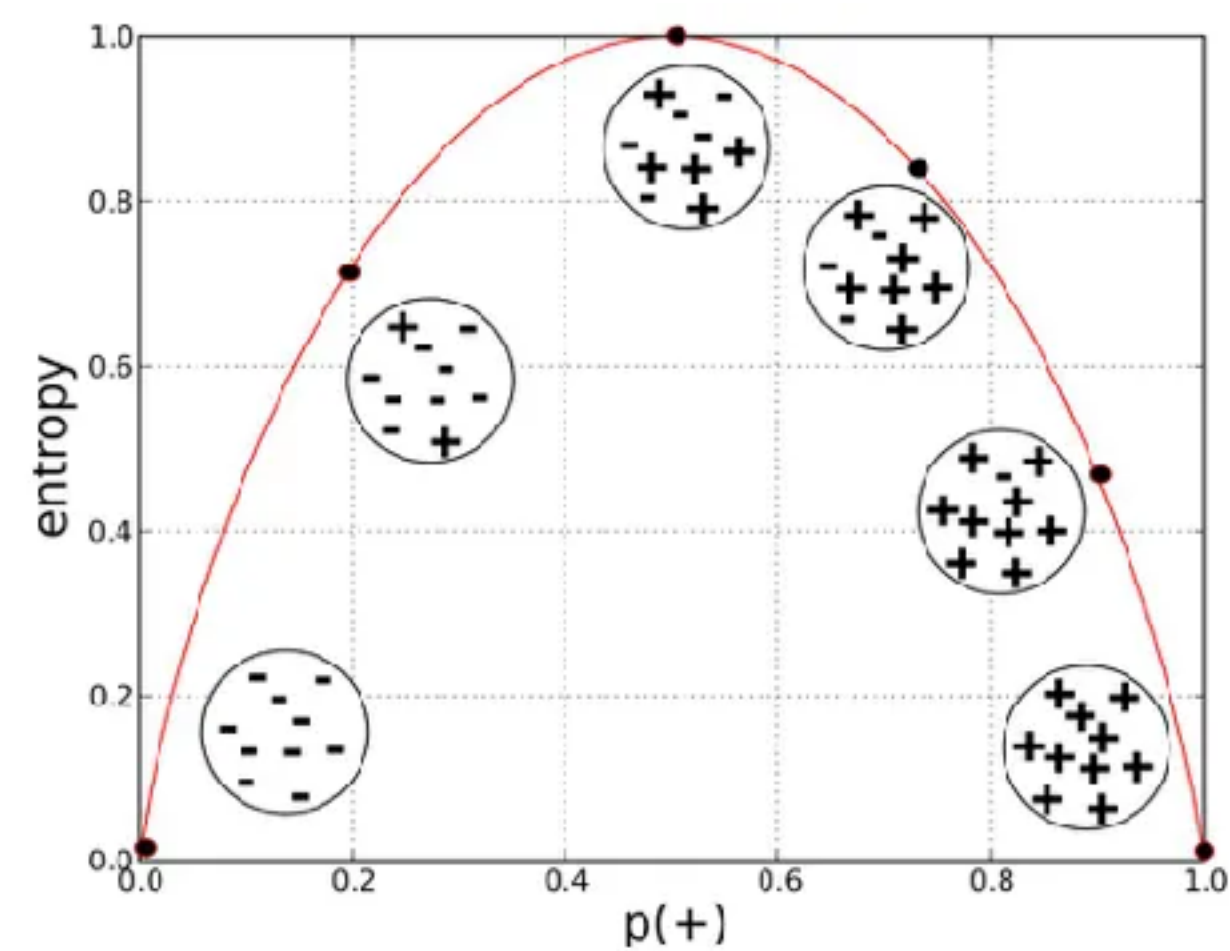
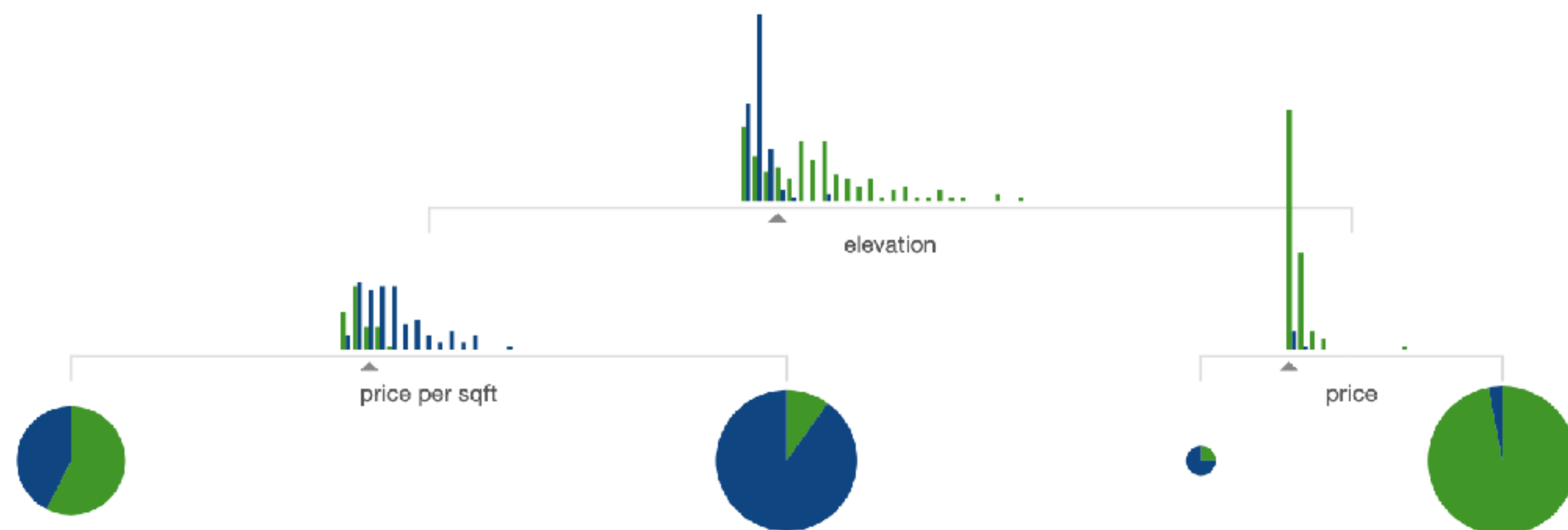


Slides & graphics from from David Sonntag and Ben Coleman

Decision trees II

Growing a tree

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



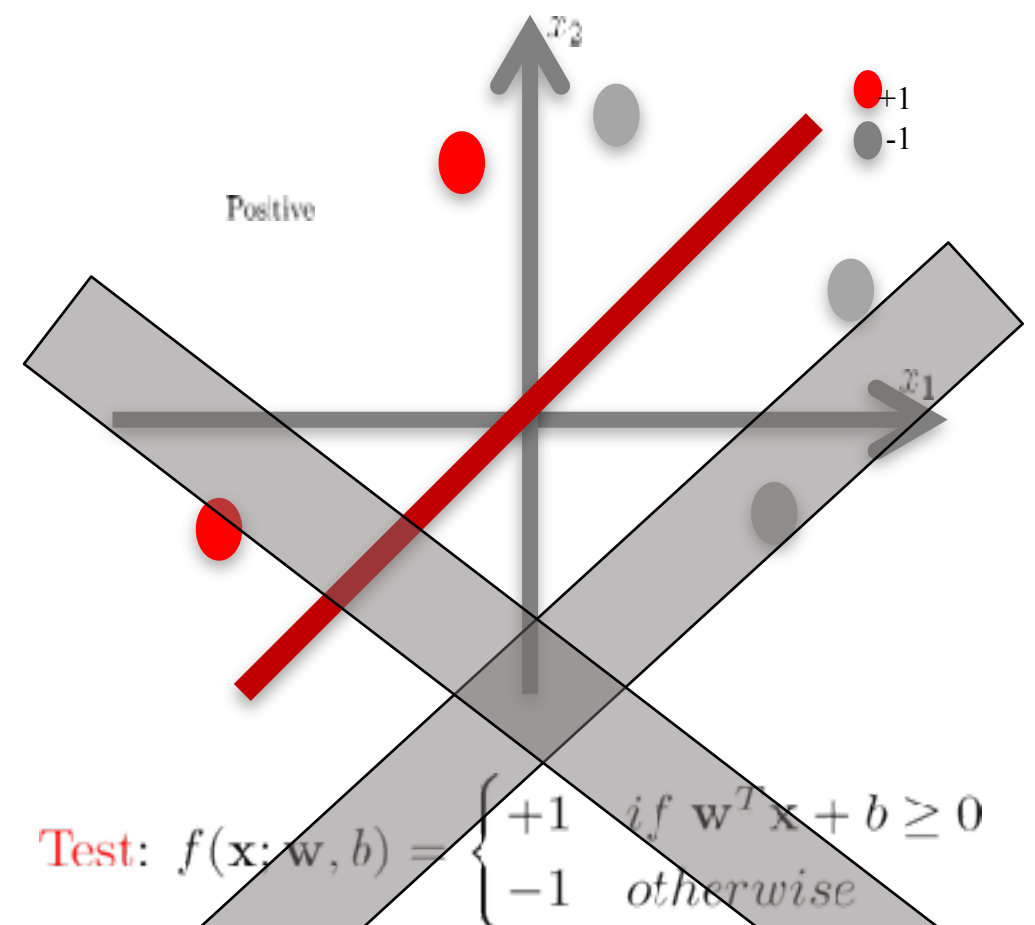
Training

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



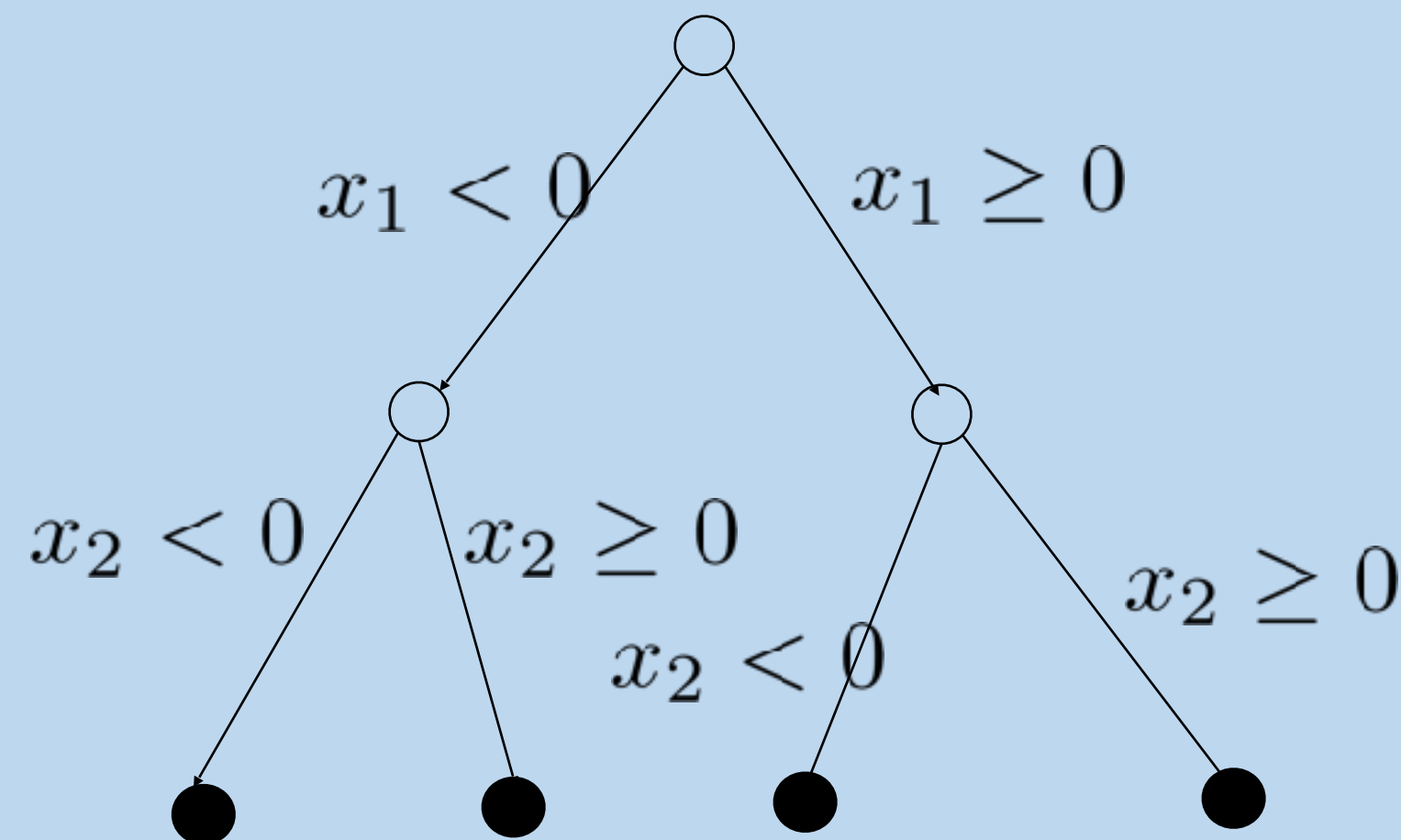
Predict using a function of weights

Very different than other
classifiers

Training

Recursive greedy search,
each step minimizes
information entropy to
decide where to split

Decision Tree



Testing

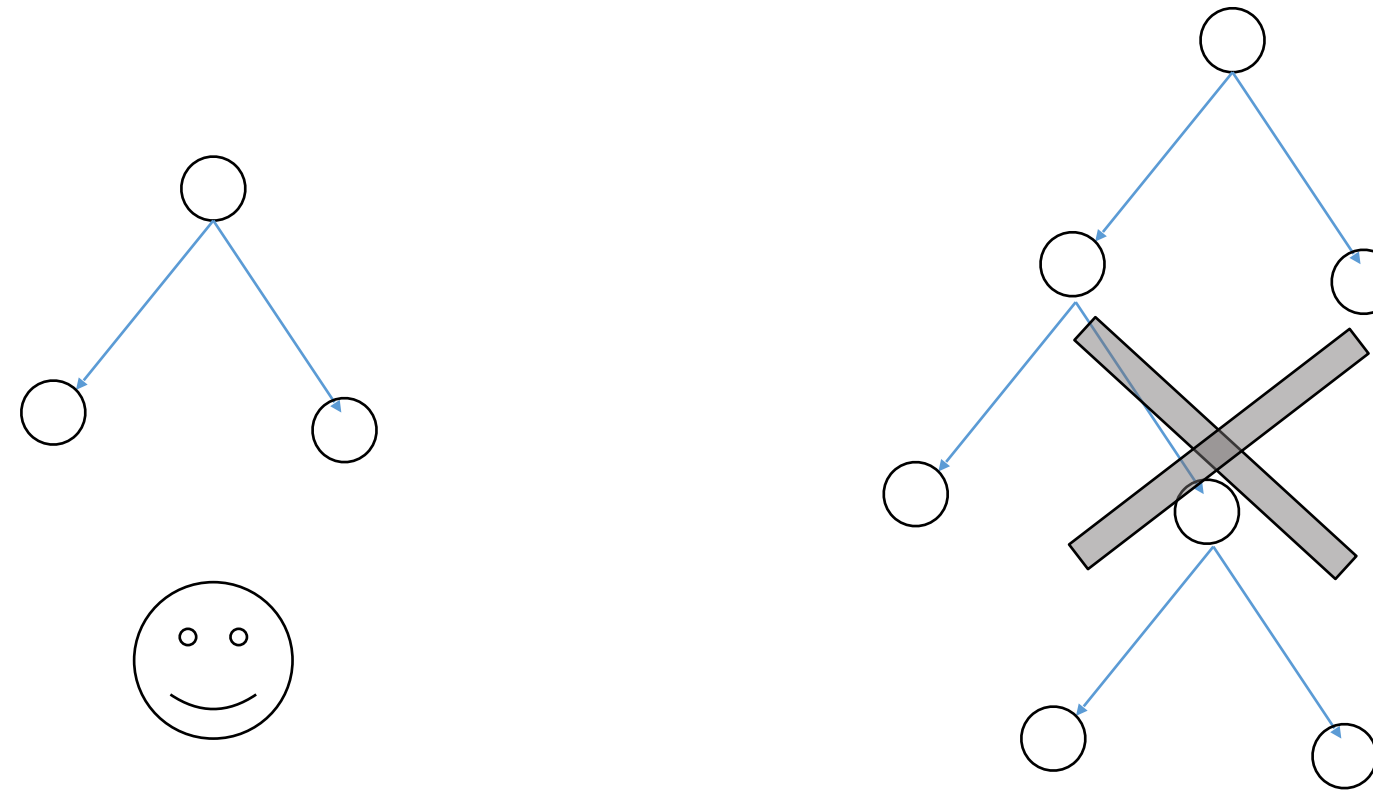
Prediction from following the
tree and making a sequence
of decisions until you end up
at a leaf node

A rule of thumb when
constructing a **decision tree**
classifier

What is a **good** decision tree classifier?

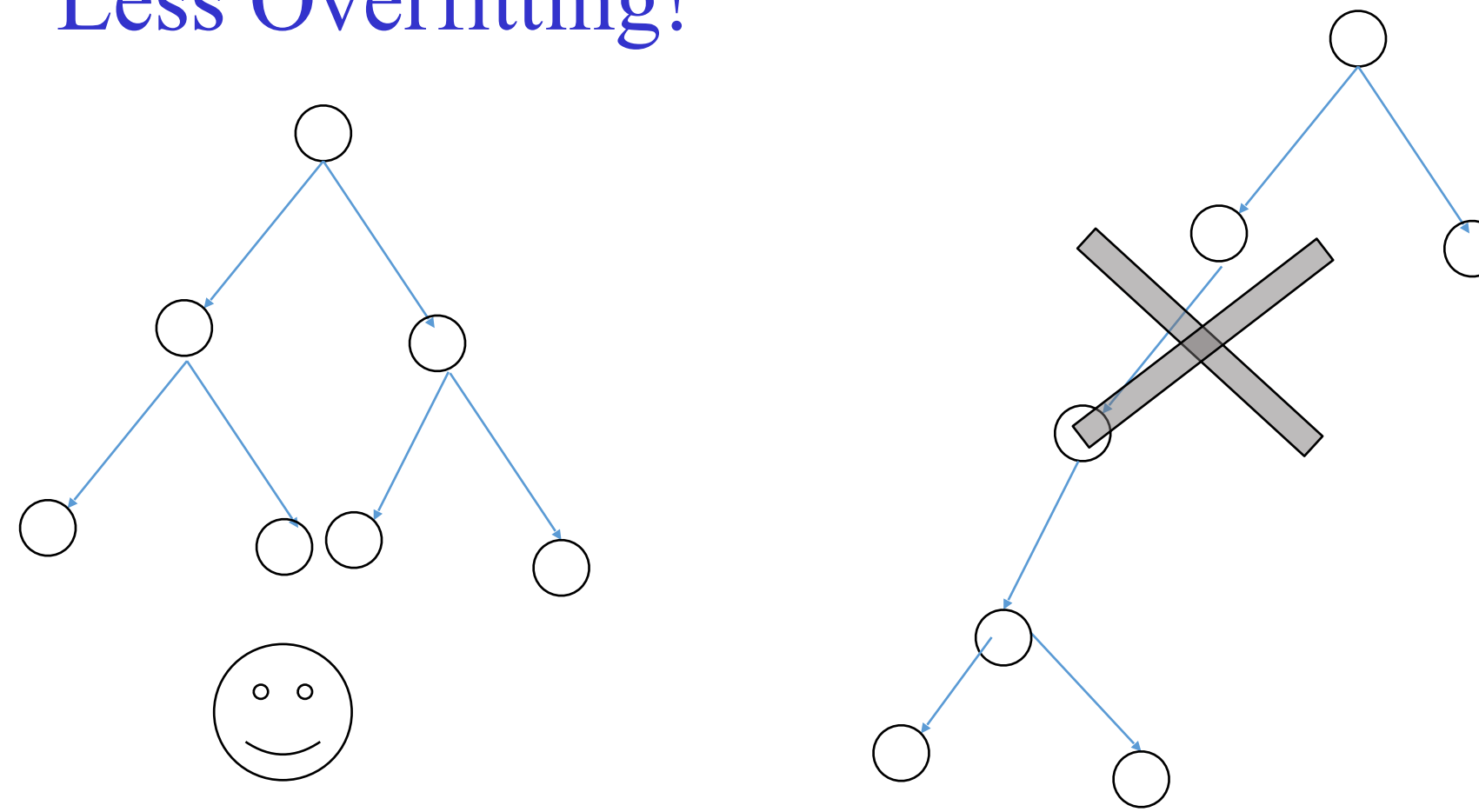
1. For the same training error, a **shallow** tree is more preferred than a **deep** tree.

Low Complexity!



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

Less Overfitting!

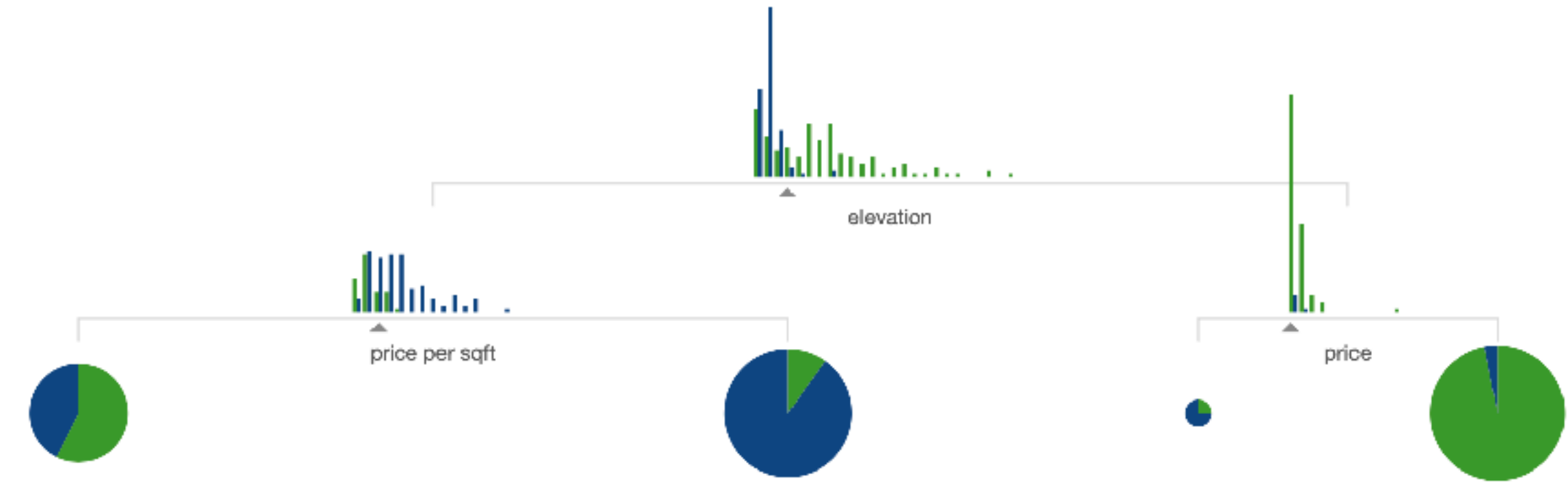


How to do this in polynomial time

- 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

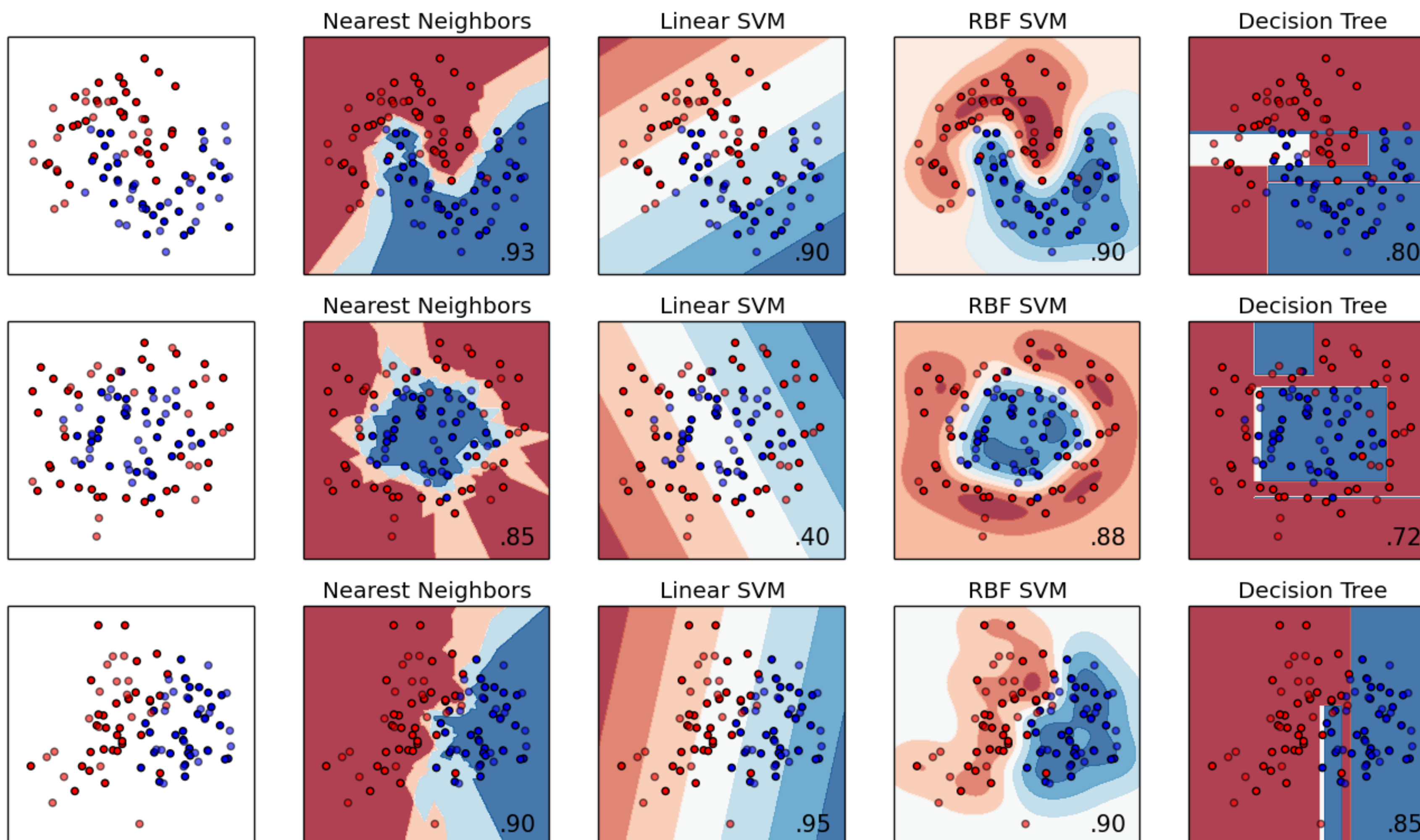
- What kinds of attributes it can handle (discrete or continuous)
 - How to choose an attribute to split (splitting criterion)
 - When to stop building the tree (more else-ifs)
 - What to do to the tree after its built (pruning)
- ← DT forms of regularization

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 $O(n)$

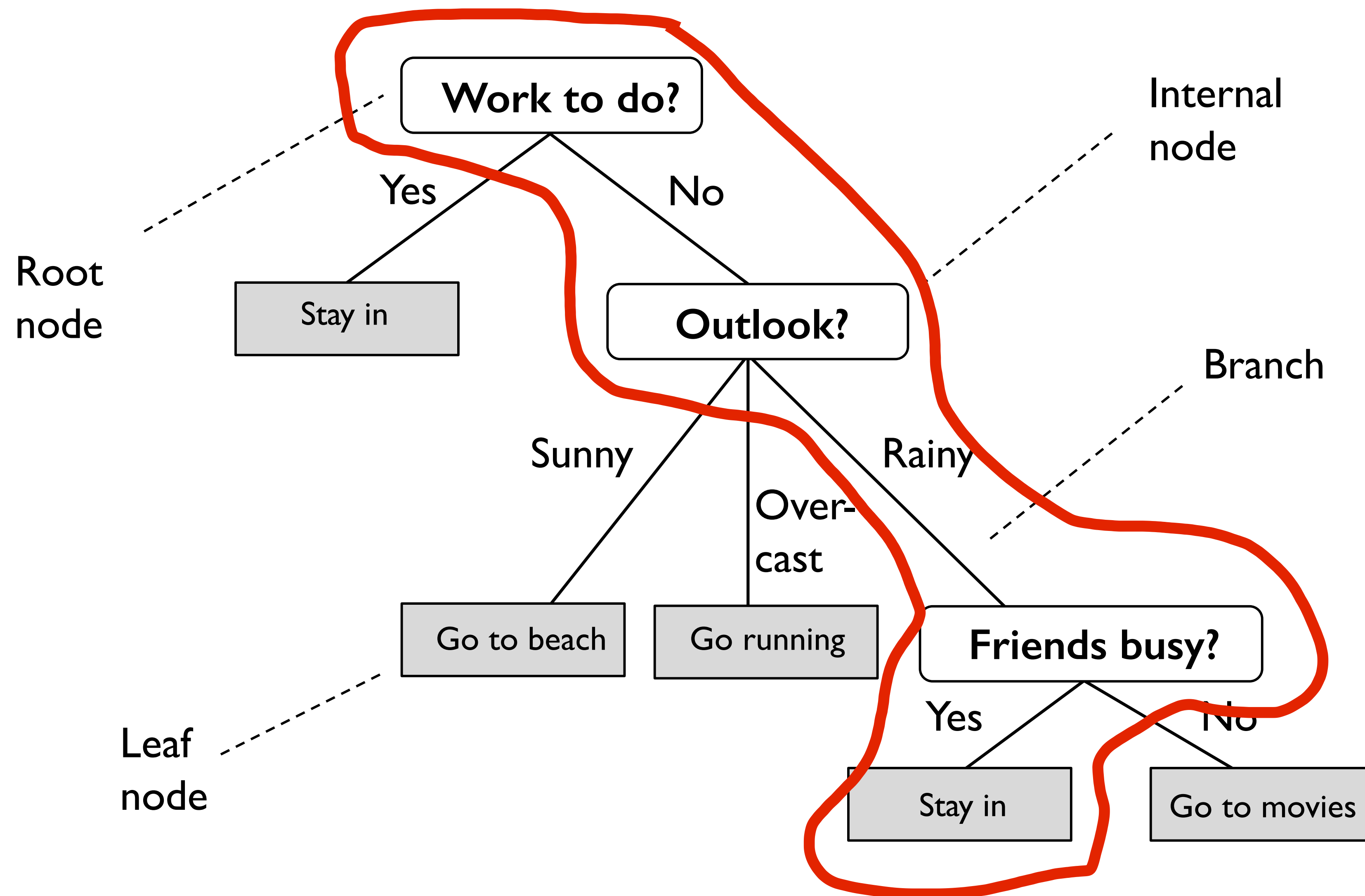
Decision trees can learn any boundary

... as long as its piecewise linear



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

$(\text{Work to do?} = \text{False}) \cap (\text{Outlook} = \text{Rainy?}) \cap (\text{Friends busy?} = \text{Yes}) \cup (\text{Work to do?} = \text{True})$

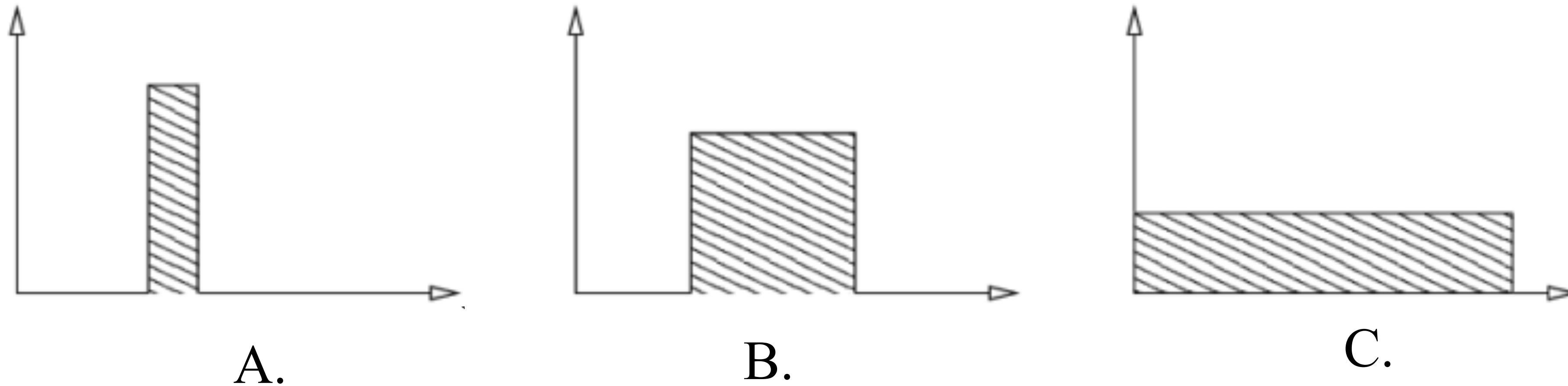


Decision trees and rules

- 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 $O(\text{tree depth})$
- Rule sets could have multiple answers, DT's only one
- Rule sets more expressive, more prone to overfitting

Training a tree

Entropy is uncertainty



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, \dots, S_t by a test B , the information gained is then

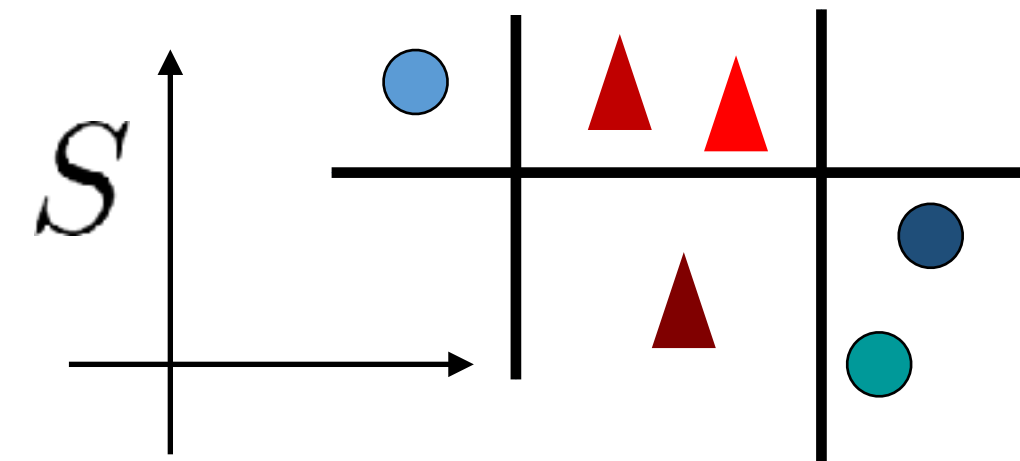
$$G(S, B) = I(S) - \sum_{i=1}^t \frac{|S_i|}{|S|} I(S_i). \quad (\text{C5.1.3.1})$$

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

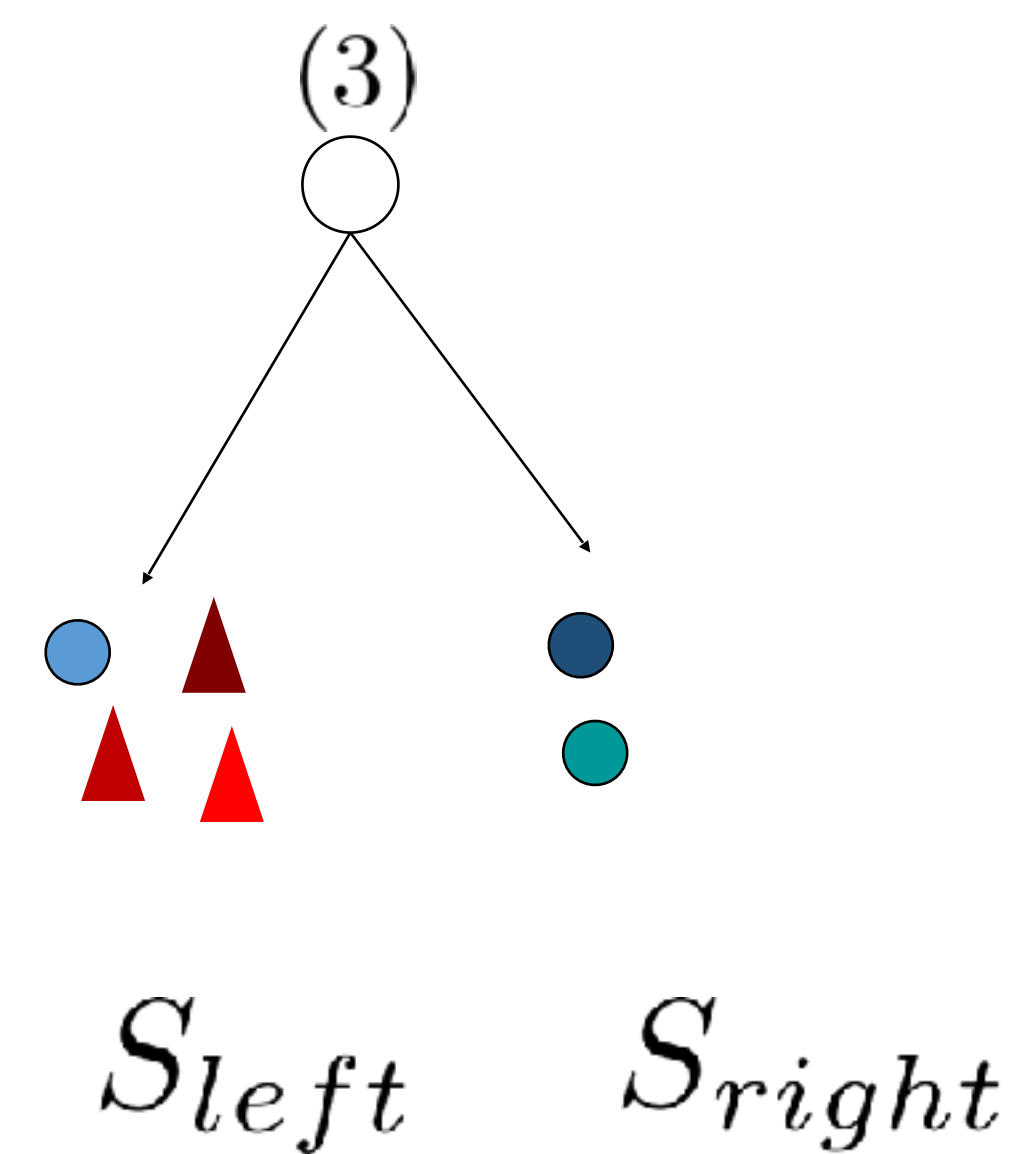
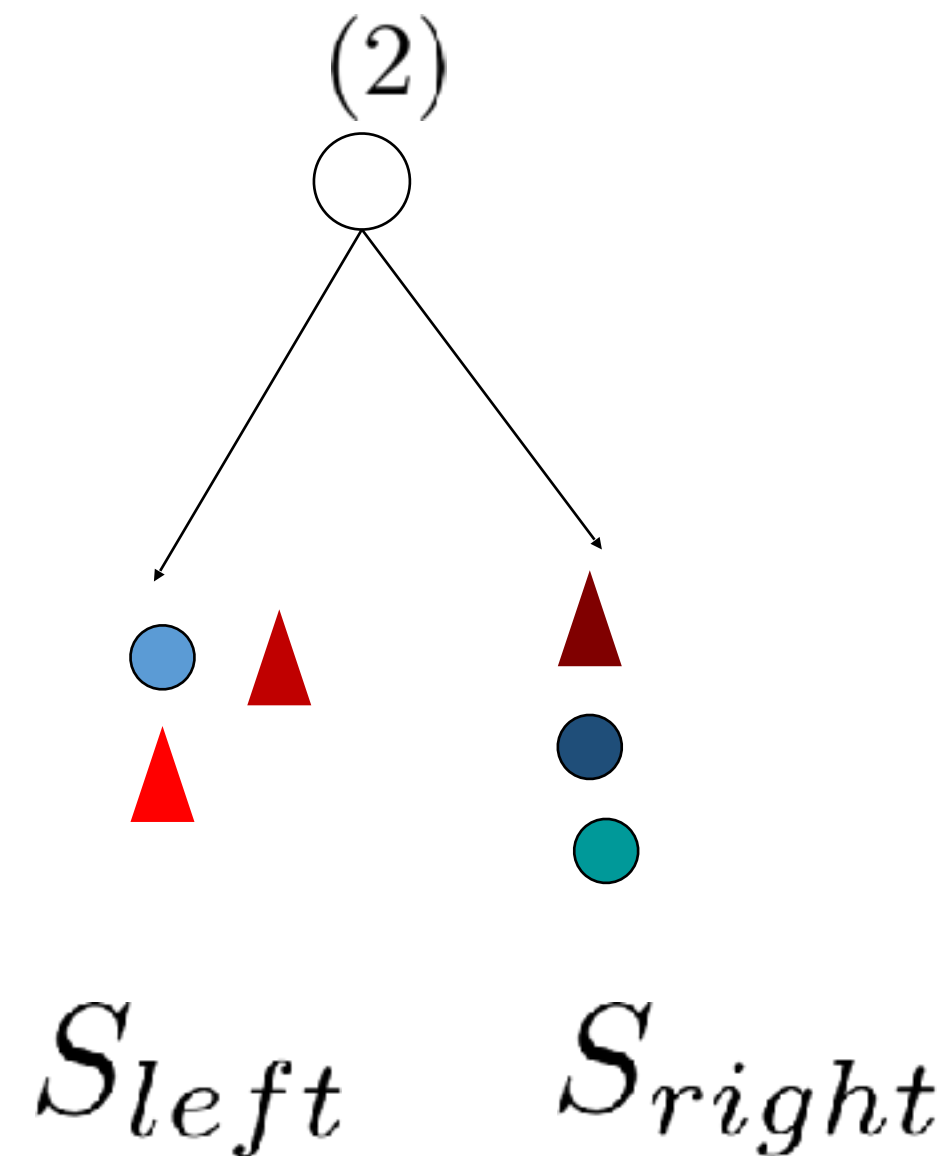
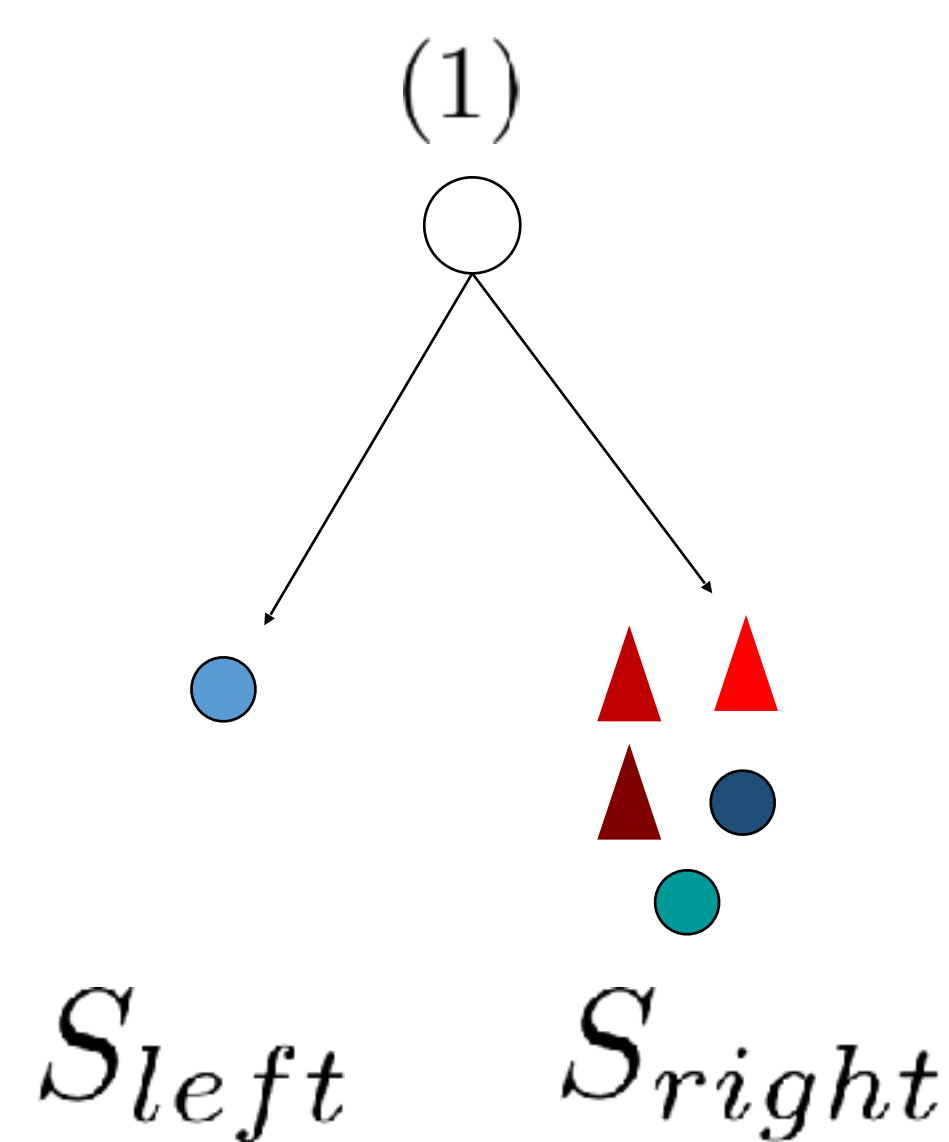
Tree construction (J. Quinlan)

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?

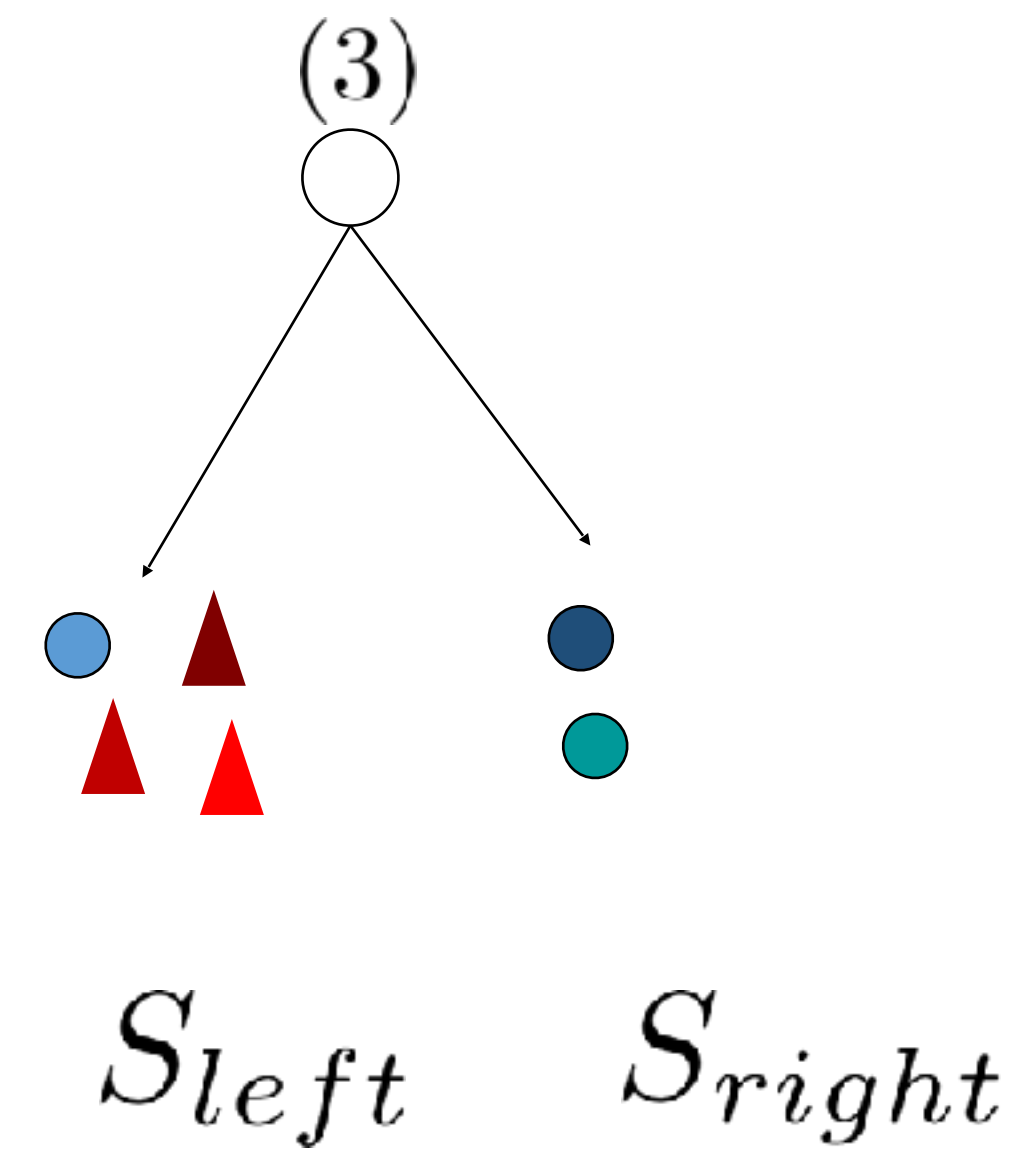
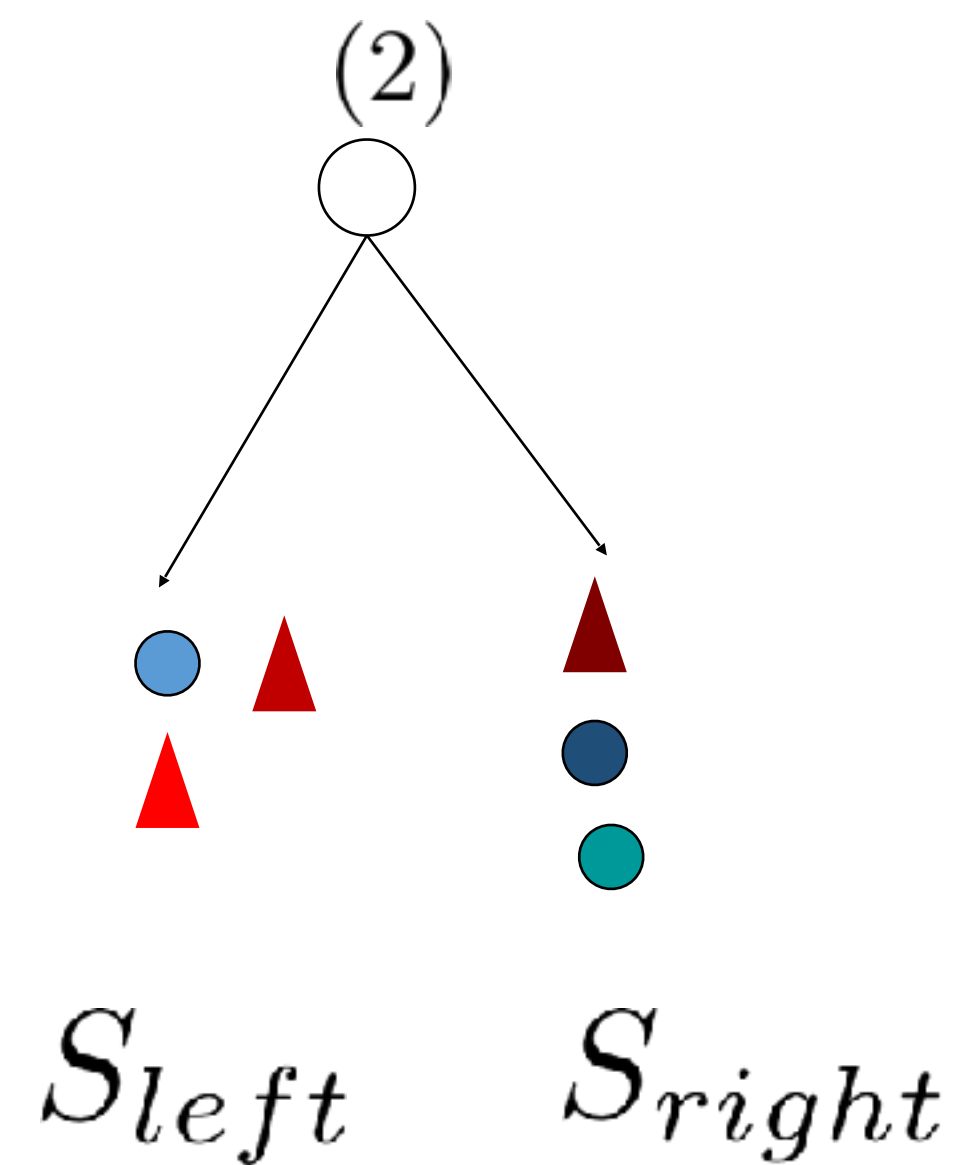
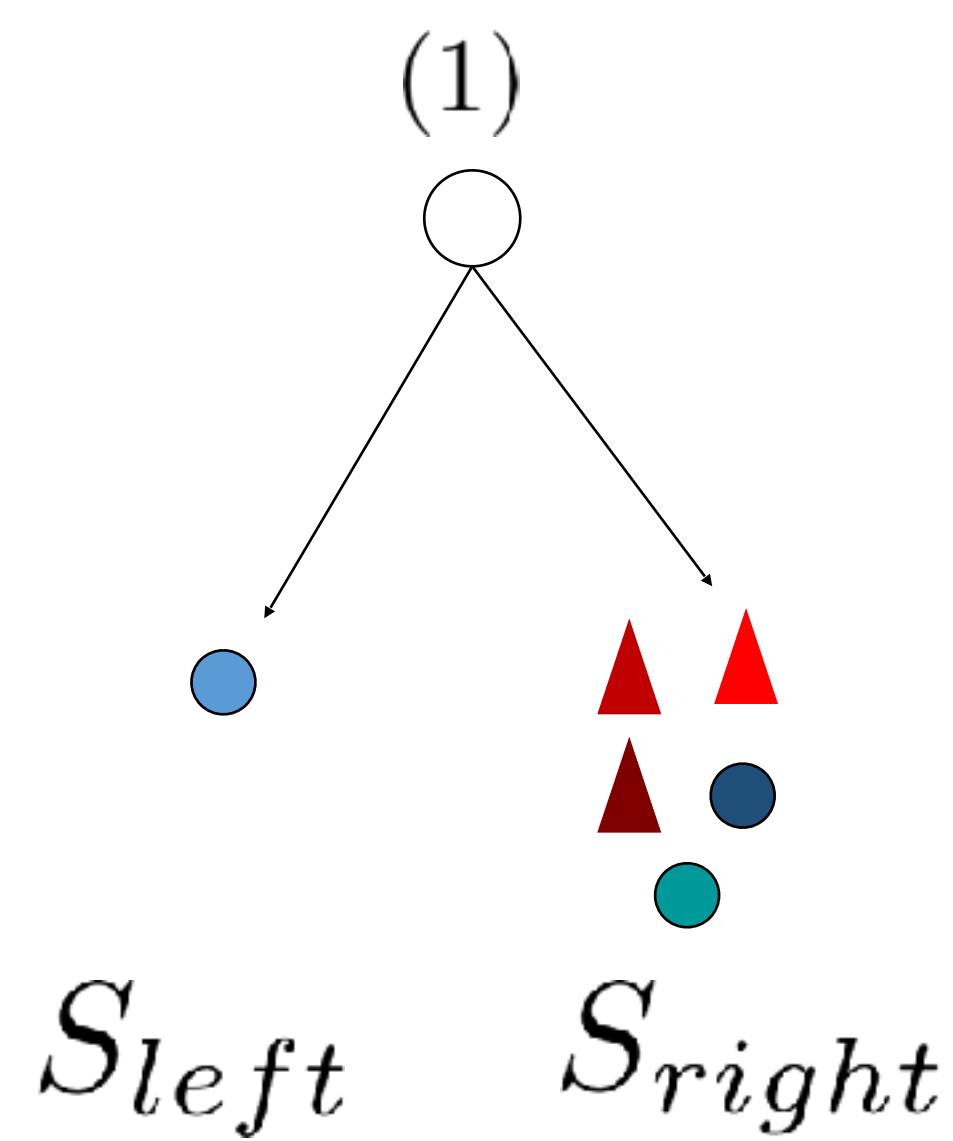
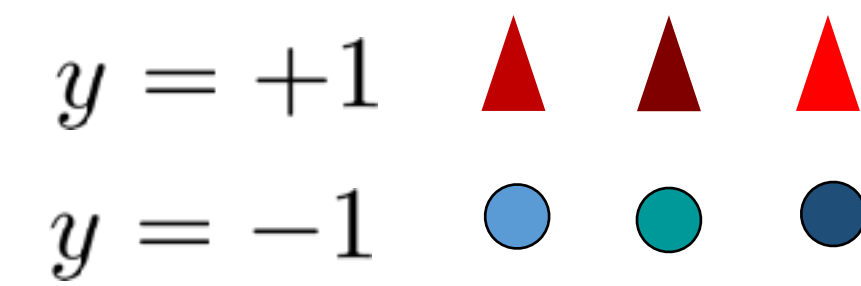
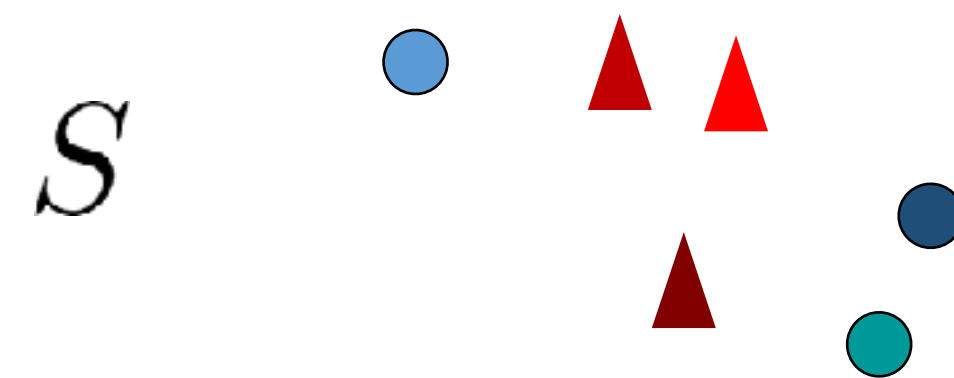


$y = +1$ ▲ ▲ ▲
 $y = -1$ ● ● ●



ID3 tree construction

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



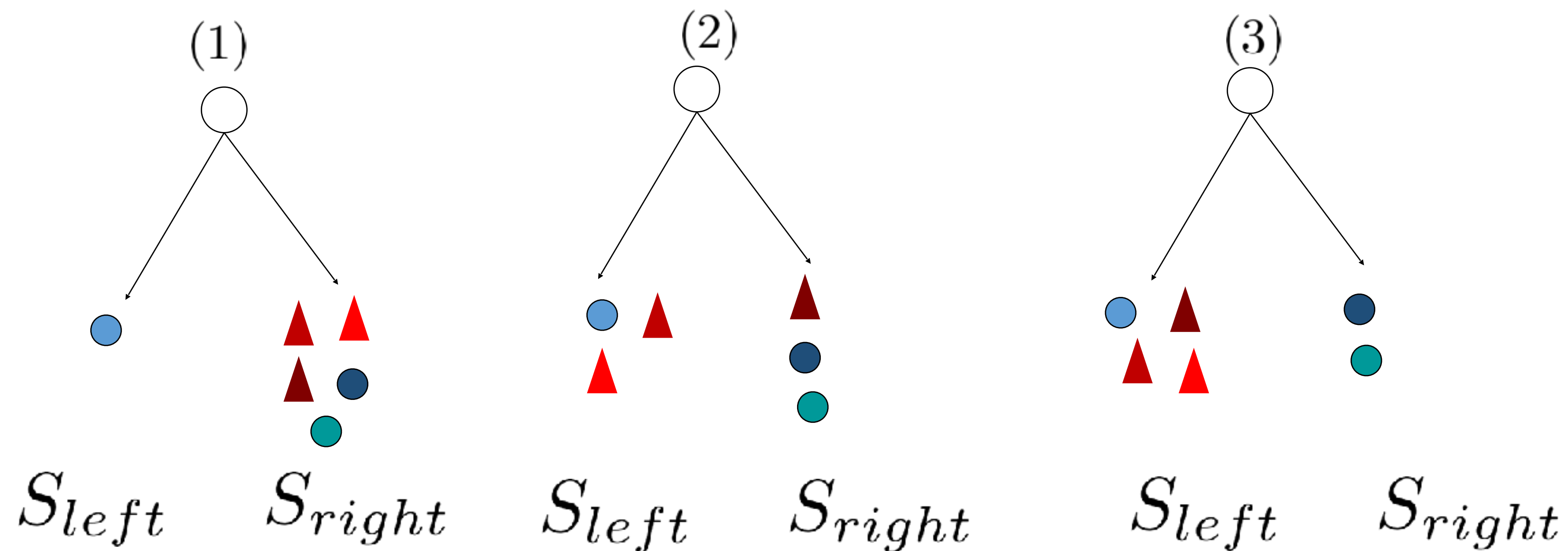
ID3 tree construction

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

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


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

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



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

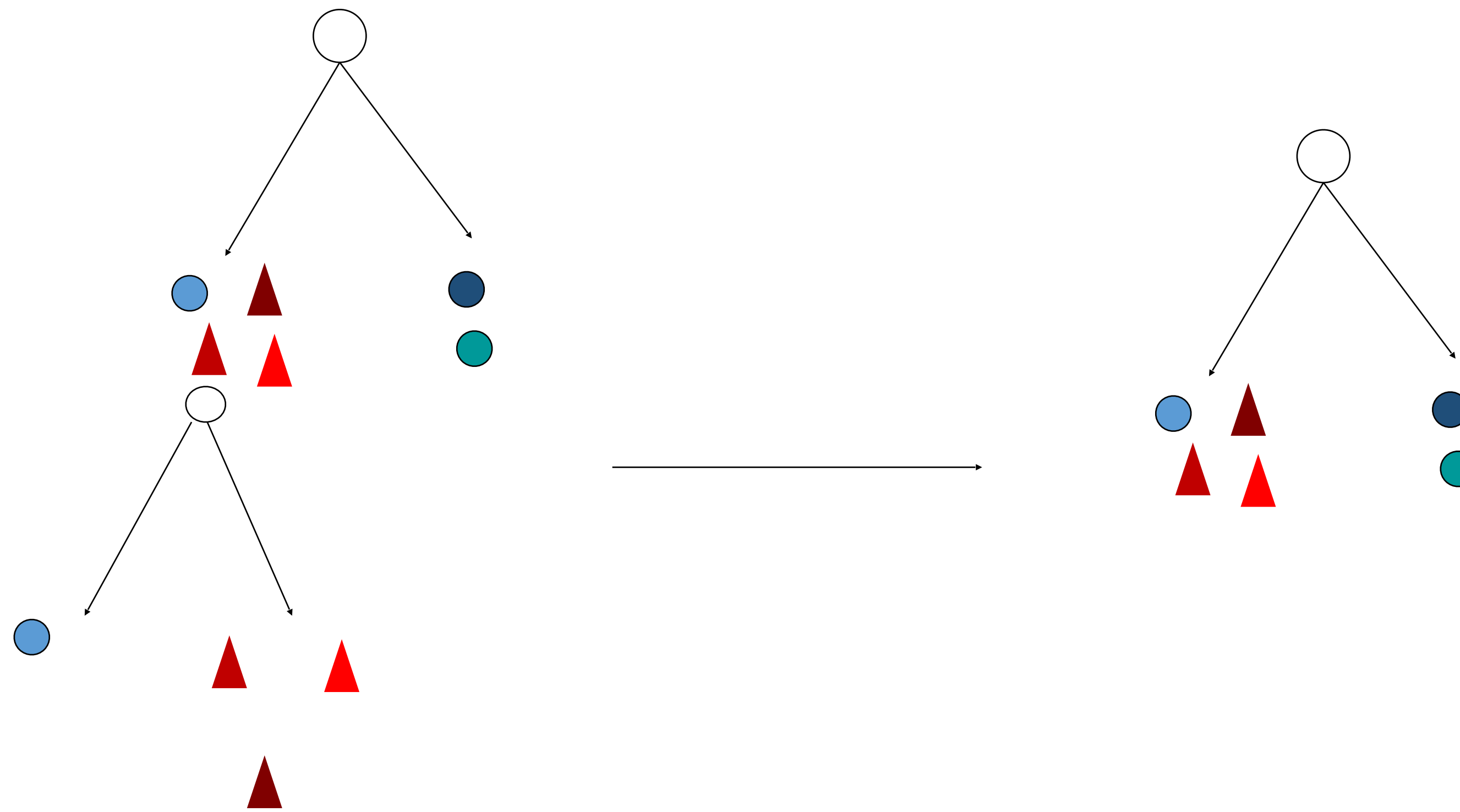
Removing retrospectively some of the tree structure using selected accuracy criteria.

Cost complexity pruning C4.5

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

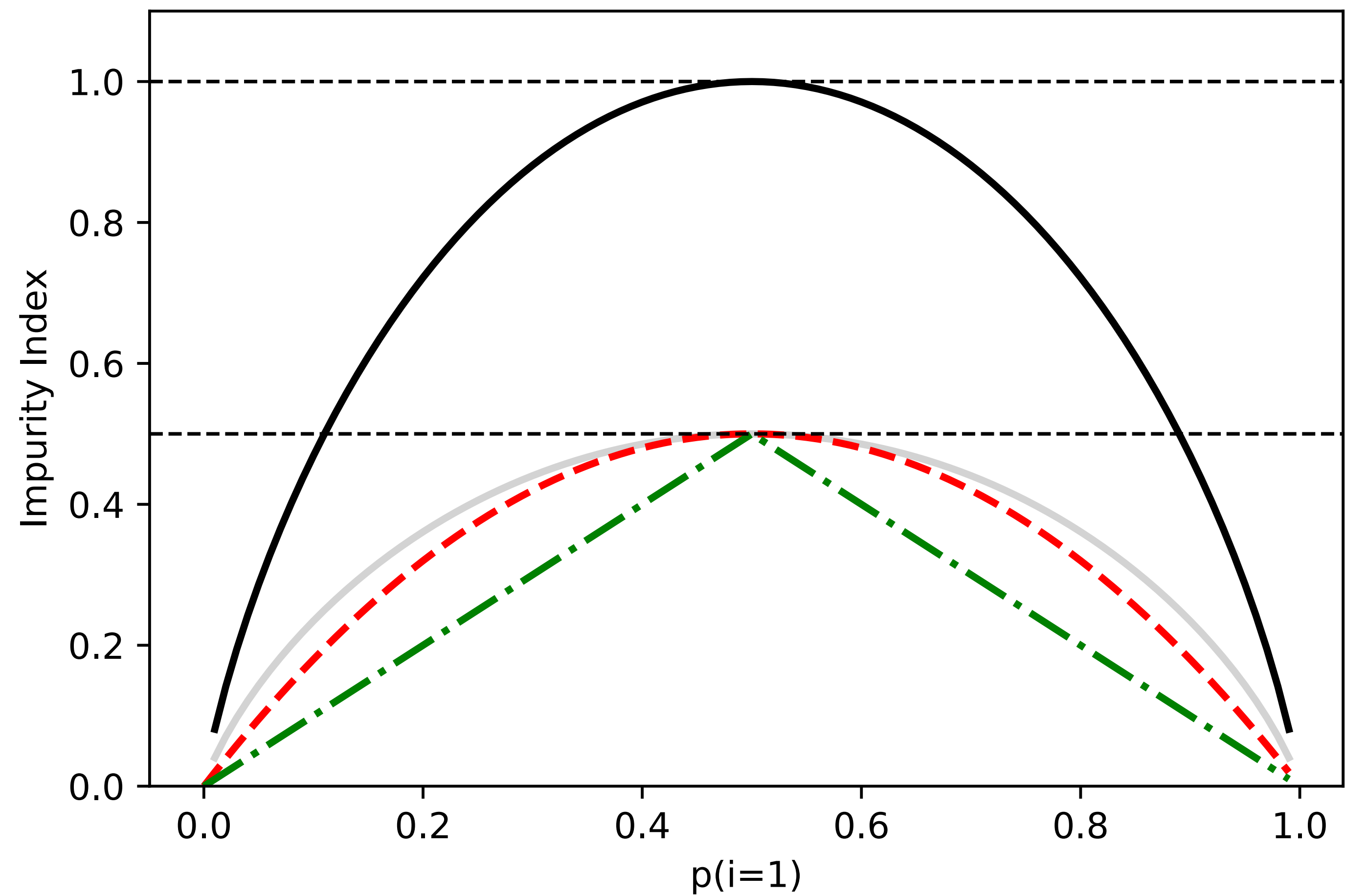
where $|T|$ is the number of terminal nodes in T and $R(T)$ is traditionally defined as the total misclassification rate of the terminal nodes. Alternatively, scikit-learn uses the total sample weighted impurity of the terminal nodes for $R(T)$. As shown above, the impurity of a node depends on the criterion. 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, $R_\alpha(T_t) = R_\alpha(t)$ or $\alpha_{eff}(t) = \frac{R(t) - R(T_t)}{|T| - 1}$. 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.



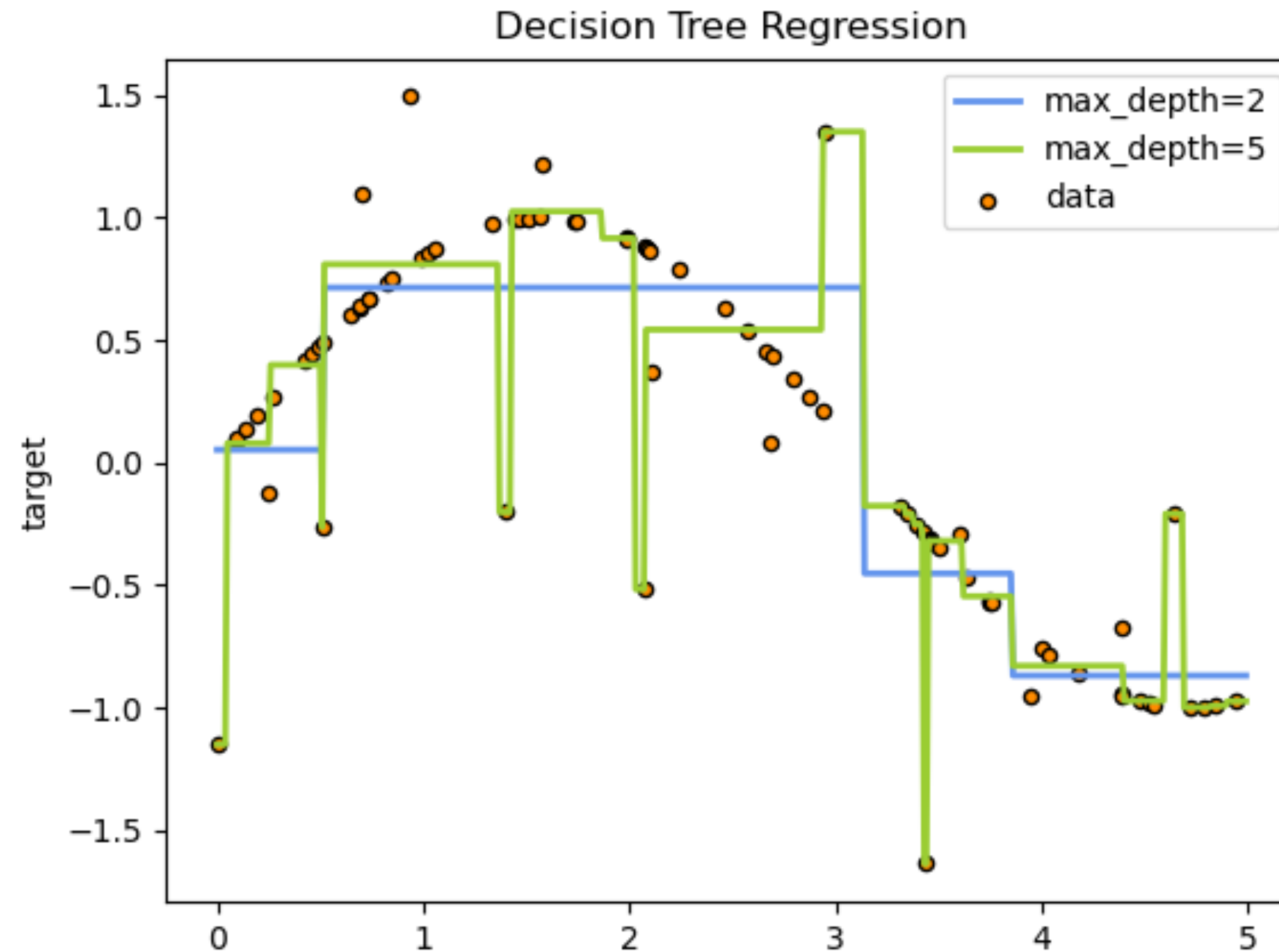
CART

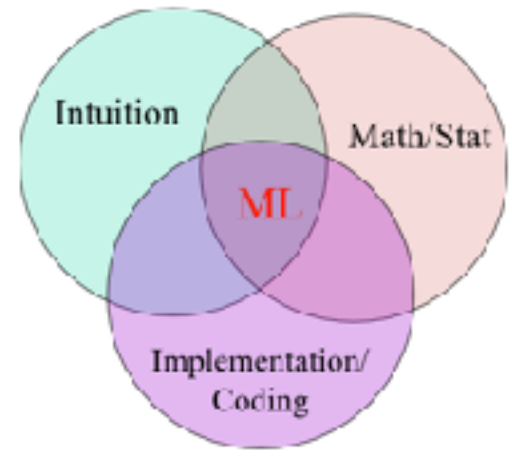
- Breiman, L. (1984). *Classification and regression trees*. Belmont, Calif: Wadsworth International Group.
- continuous and discrete features
- strictly binary splits (taller trees than ID3, C4.5)
- binary splits can generate better trees than C4.5, but tend to be larger and harder to interpret; k-attributes 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



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]

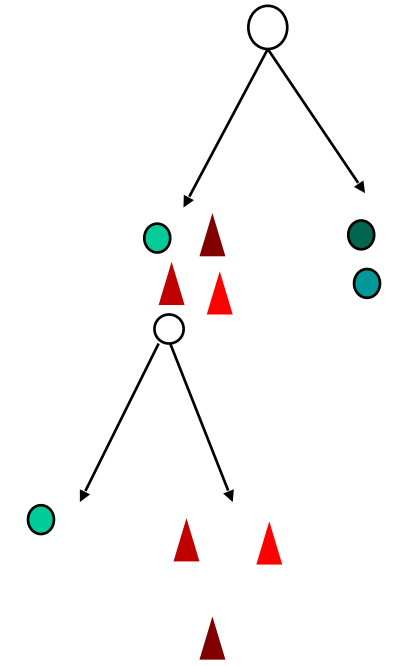
A decision tree classifier.

Read more in the [User Guide](#).

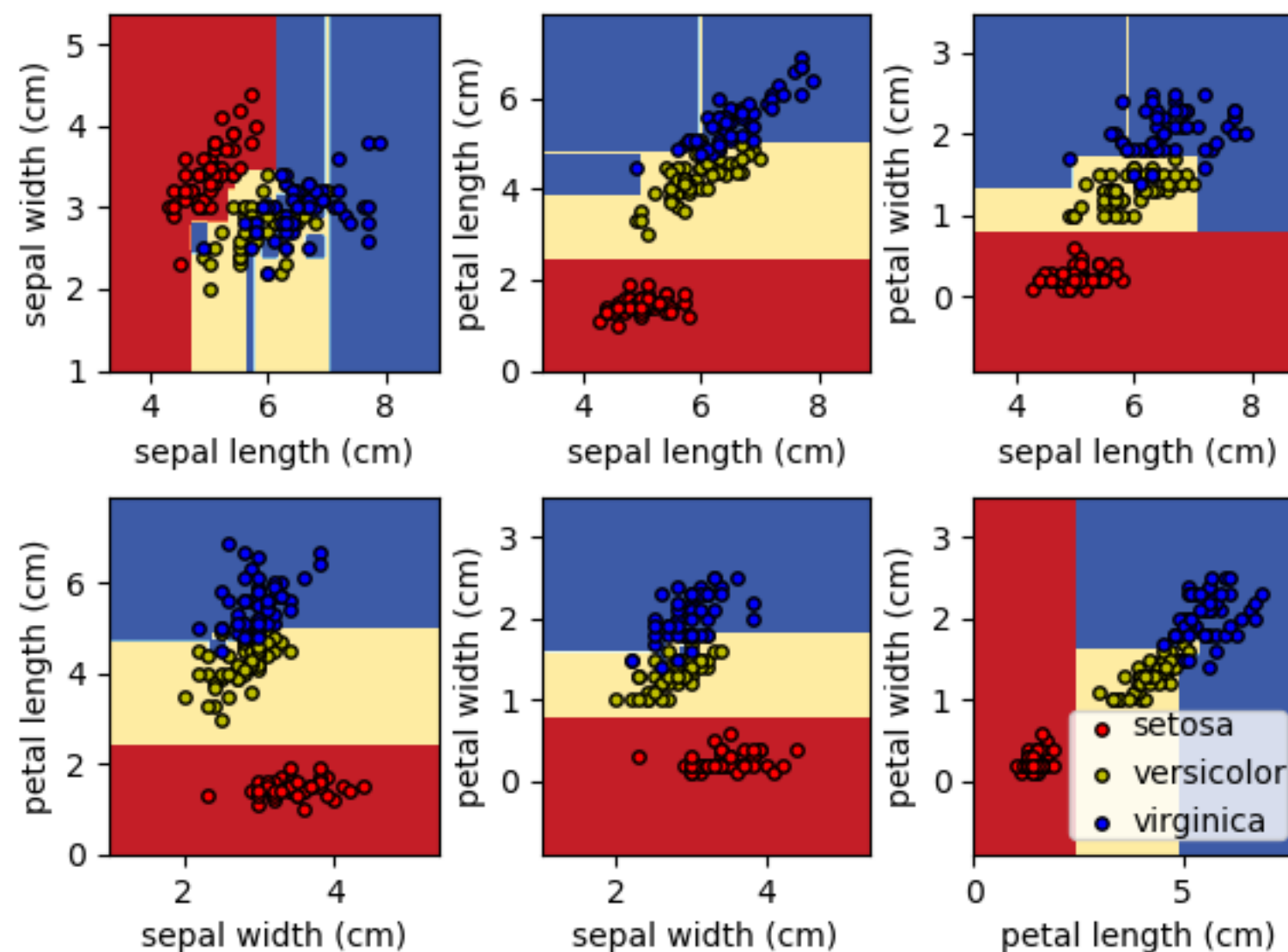
Parameters: `criterion : {"gini", "entropy"}, default="gini"`

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**.



Decision surface of a decision tree using paired features



Math:

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

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

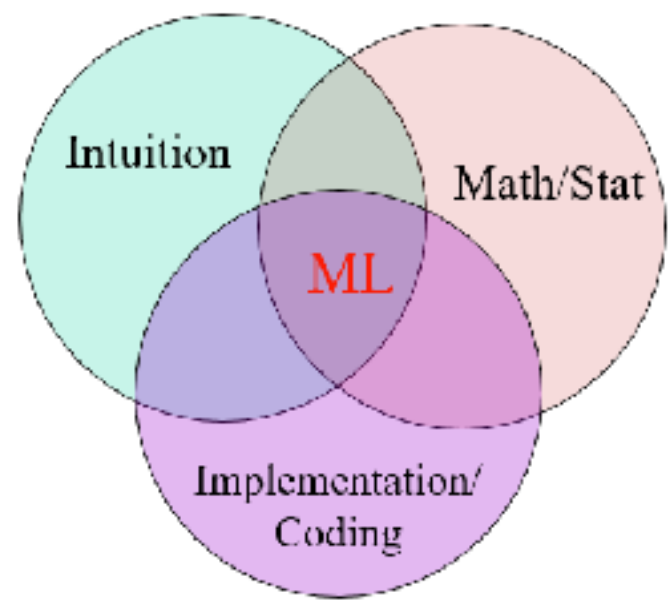
Decision trees

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



Implementation

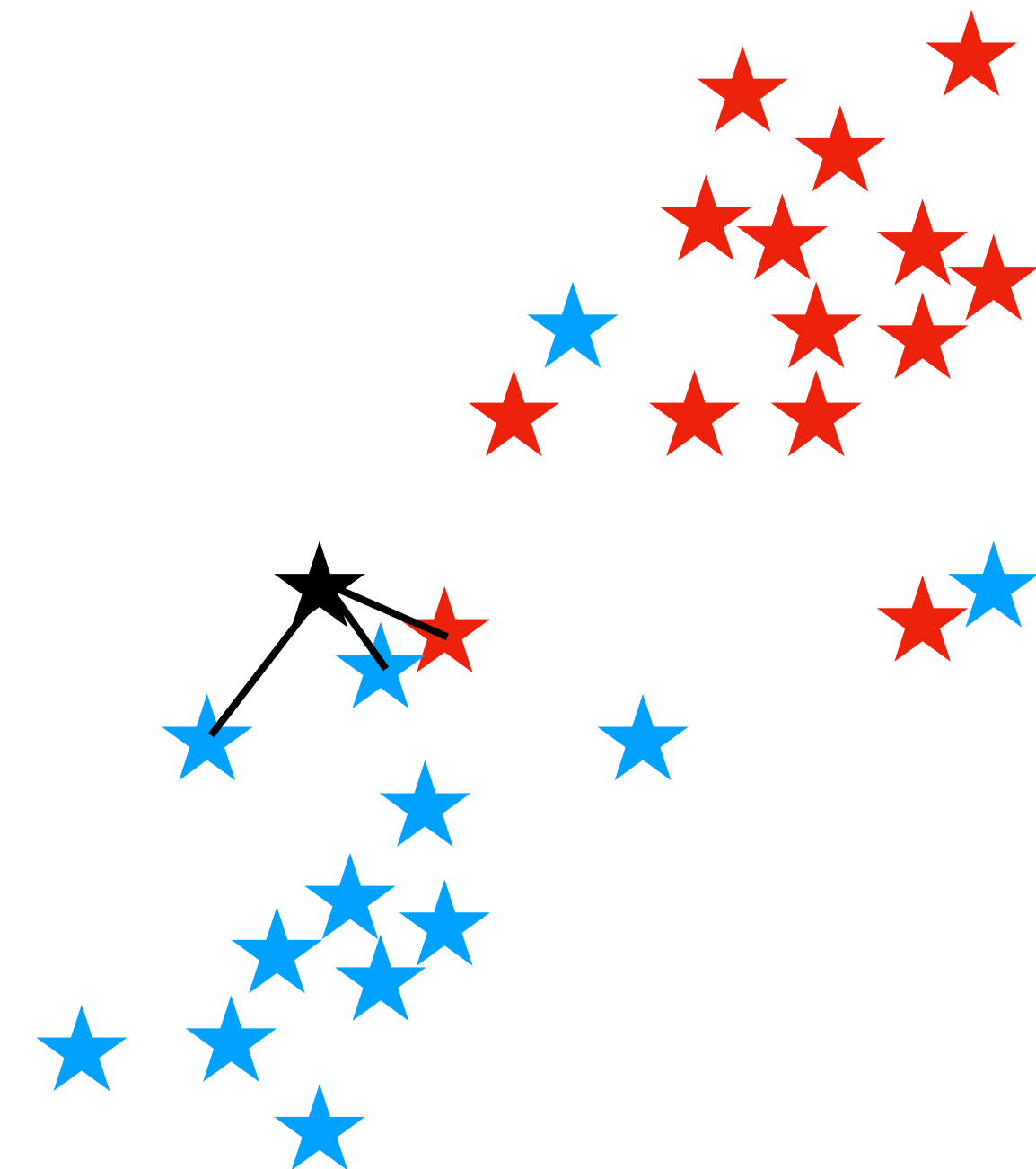
Decision trees and nearest neighbors



https://github.com/COGS118A/demo_notebooks/blob/main/lecture_08_trees_and_neighbors.ipynb

https://github.com/COGS118A/demo_notebooks.git

Nearest neighbors



Training: store the data

Time $O(1)$

Memory $O(n)$

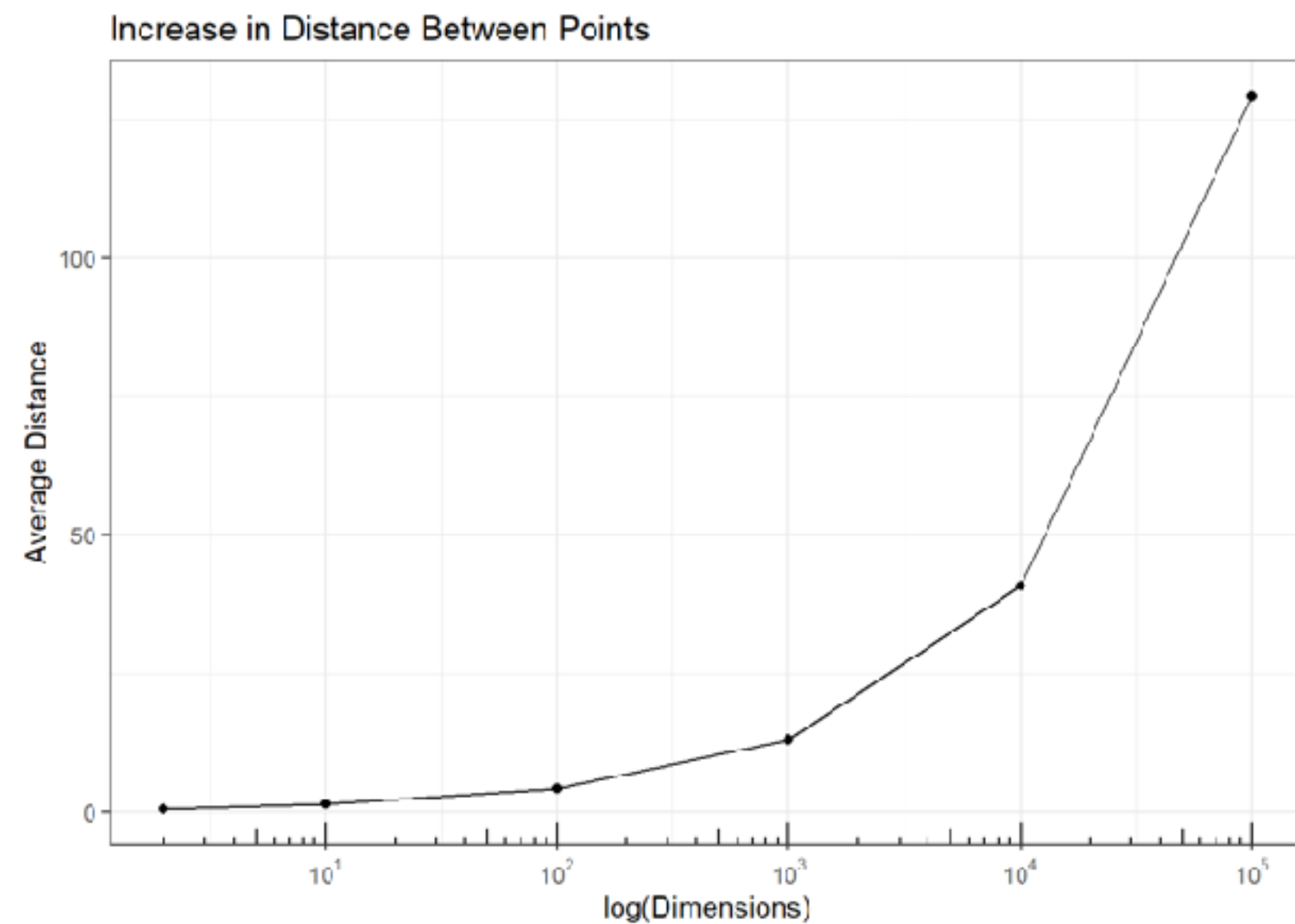
Testing: find the k nearest

Time $O(n)$

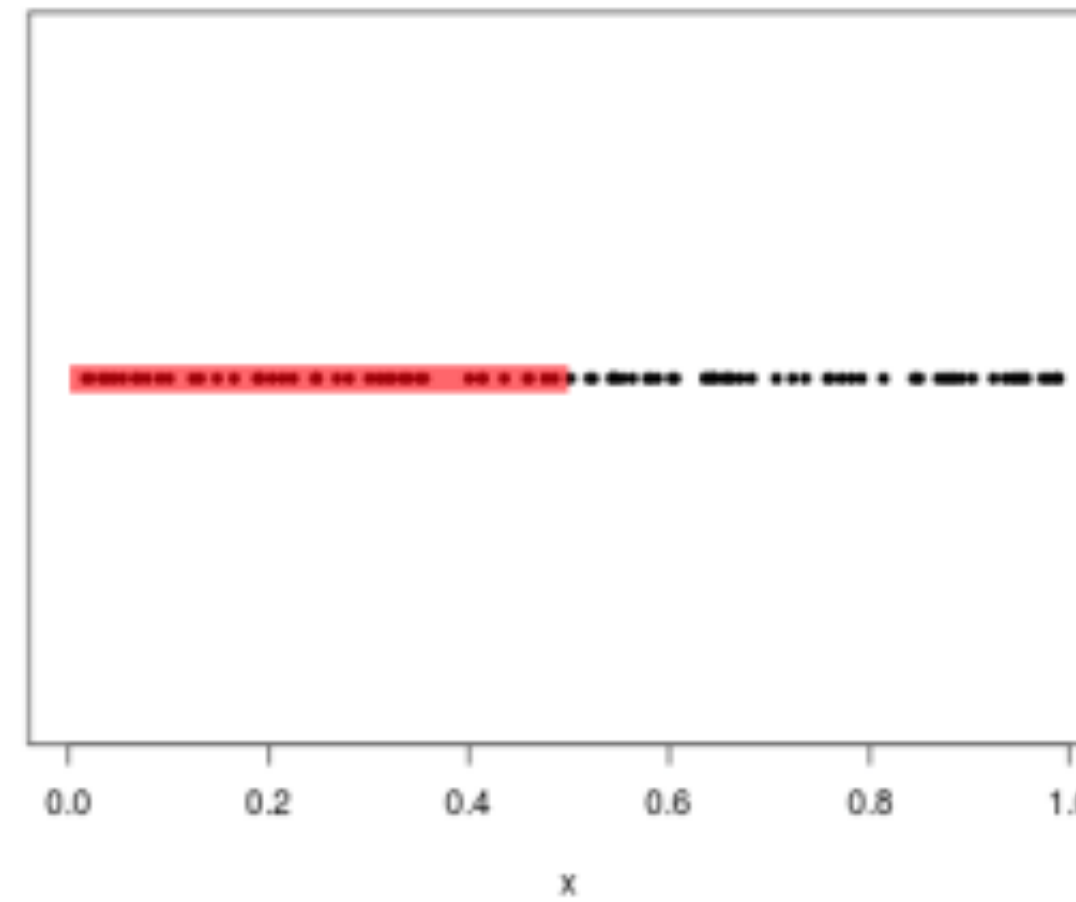
Memory $O(n)$

The Curse of Dimensionality

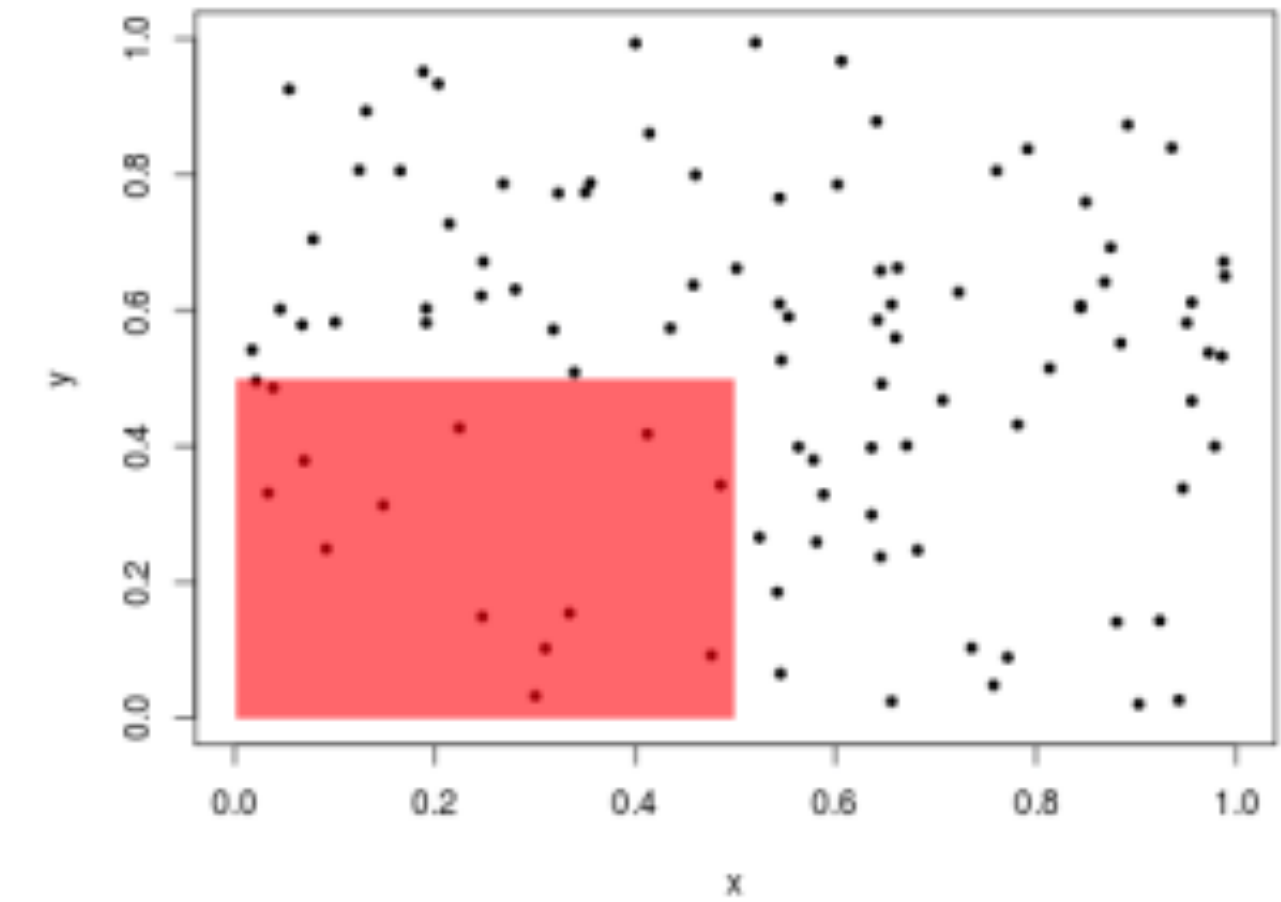
- Nearest neighbor breaks down in high-dimensional spaces because the “neighborhood” becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm.
- Suppose our query point is at the origin.
 - 1D –
 - On a one dimensional line, we must go a distance of $5/5000 = 0.001$ on average to capture the 5 nearest neighbors
 - 2D –
 - In two dimensions, we must go $\sqrt{0.001}$ to get a square that contains 0.001 of the volume
 - D –
 - In D dimensions, we must go $(0.001)^{1/D}$



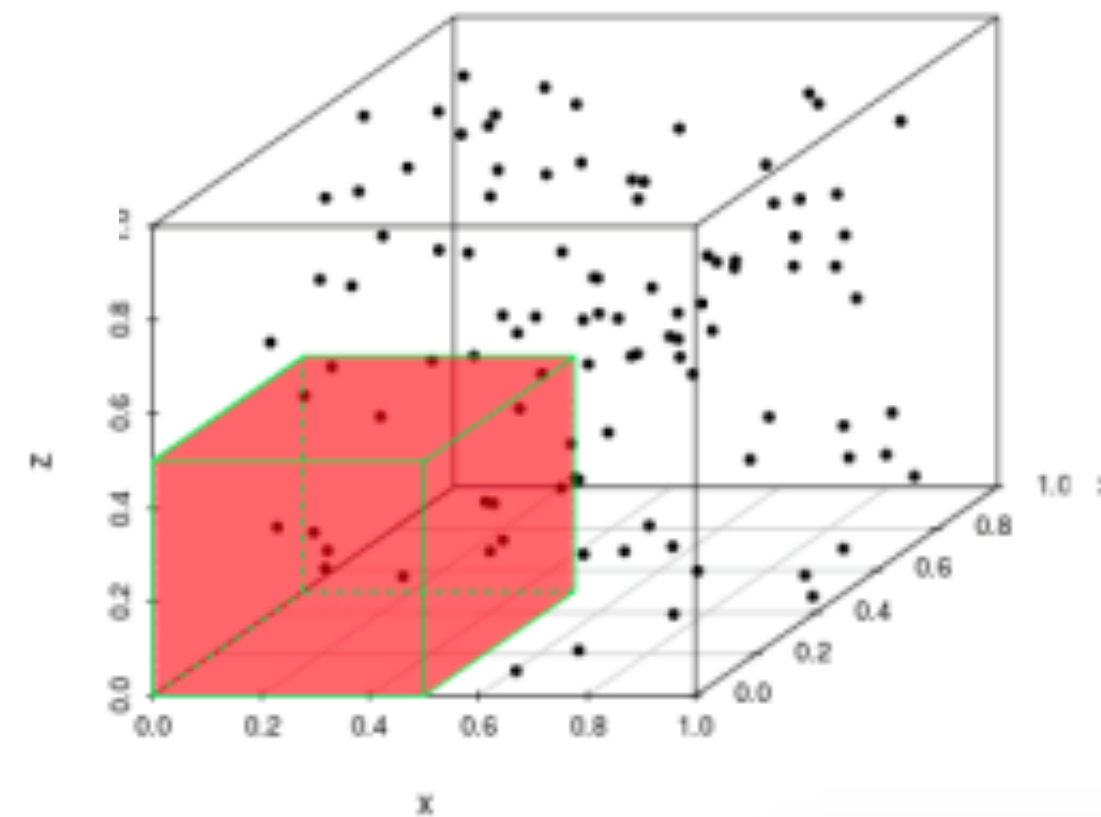
1-D: 42% of data captured.



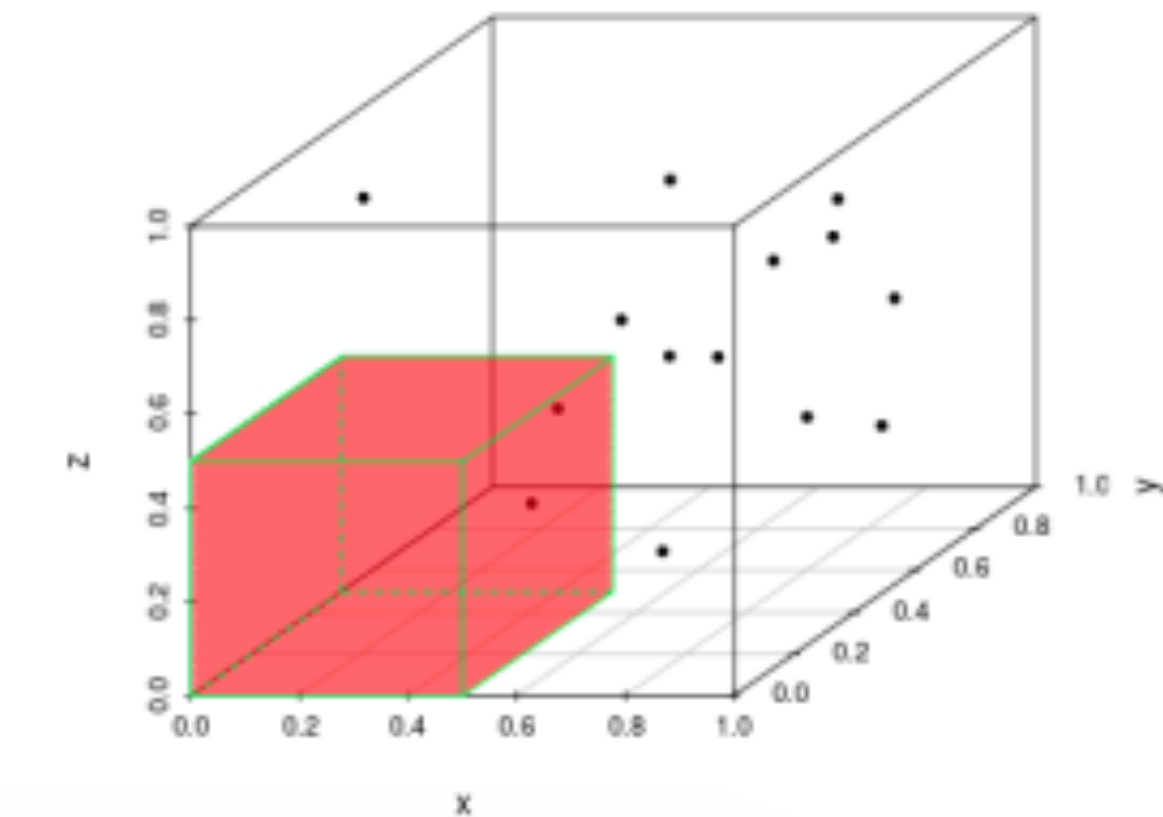
2-D: 14% of data captured.



3-D: 7% of data captured.

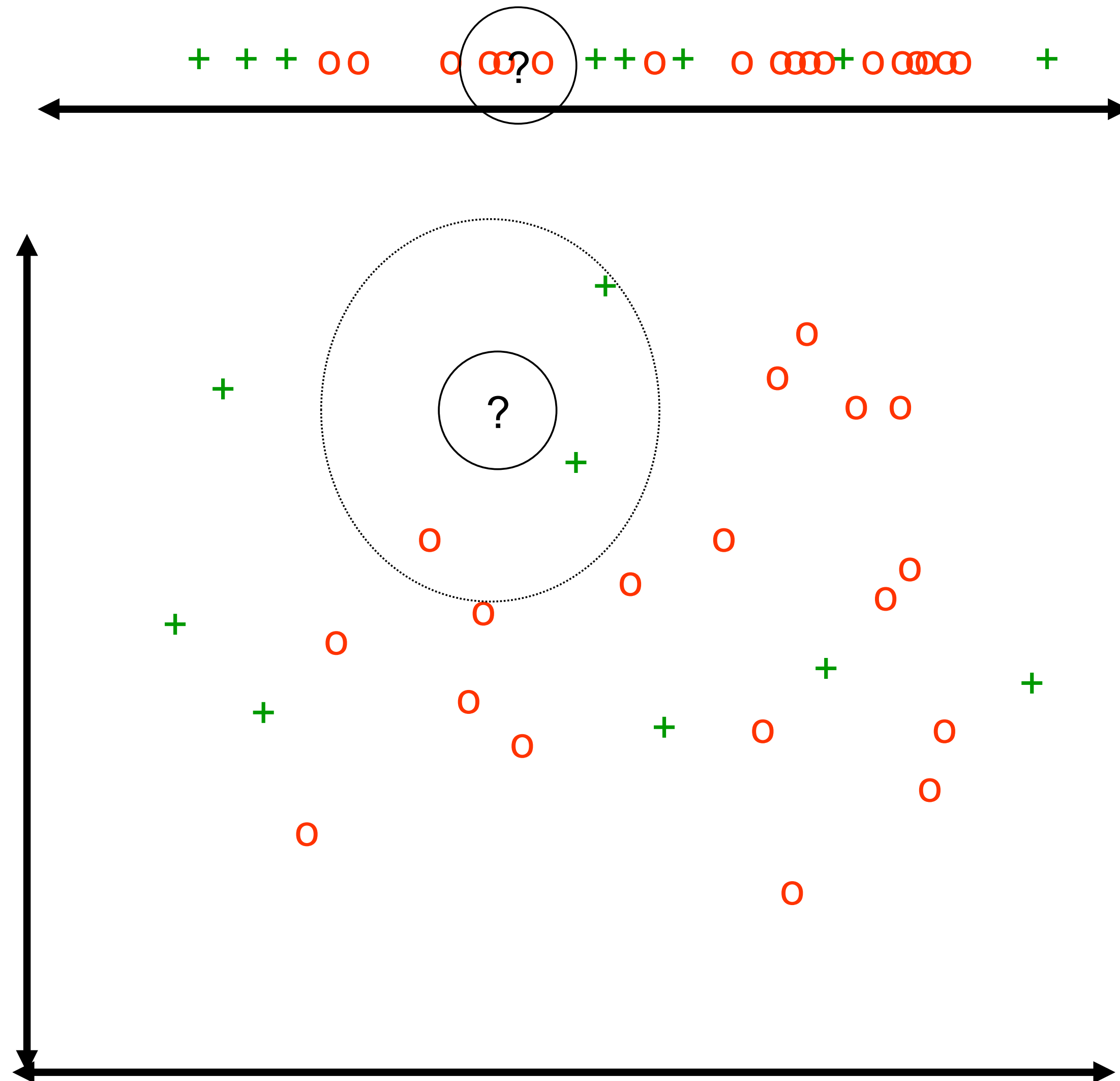


4-D: 3% of data captured.



t = 0

k-NN and irrelevant dimensions



**The curse of
dimensionality strikes!**