Security Analytics

Topic 7: Decision Trees

Purdue University
Prof. Ninghui Li
Based on slides by Raymond J. Mooney and Gavin Brown

Readings

- Principle of Data Mining
  - Chapter 10: Predictive Modeling for Classification
- Outline:
  - A bit of learning theory
  - Classification trees

Classification (Categorization)

- Given:
  - A description of an instance, $x \in X$, where X is the ***instance language*** or ***instance space***.
  - A fixed set of categories: $C=\{c_1, c_2,\ldots c_n\}$
- Determine:
  - The category of x: $c(x) \in C$, where $c(x)$ is a categorization function whose domain is X and whose range is C.
  - If $c(x)$ is a binary function $C=\{0,1\}$ ({true,false}, {positive, negative}) then it is called a ***concept***.

Learning for Categorization

- A training example is an instance $x \in X$, paired with its correct category $c(x)$: $\langle x, c(x) \rangle$ for an unknown categorization function, c.
- Given a set of training examples, D.
- Find a hypothesized categorization function, $h(x)$, such that:

$$\forall \langle x, c(x) \rangle \in D : h(x) = c(x)$$

Consistency

Sample Category Learning Problem

- Instance language: <size, color, shape>
  - size ∈ {small, medium, large}
  - color ∈ {red, blue, green}
  - shape ∈ {square, circle, triangle}
- *C* = {positive, negative}
- *D*:

Example	Size	Color	Shape	Category
1	small	red	circle	positive
2	large	red	circle	positive
3	small	red	triangle	negative
4	large	blue	circle	negative

Hypothesis Selection

- Many hypotheses are usually consistent with the training data.
  - red & circle
  - (small & circle) or (large & red)
  - (small & red & circle) or (large & red & circle)
  - not [(red & triangle) or (blue & circle)]
  - not [(small & red & triangle) or (large & blue & circle)]
- Restrict learned functions a priori to a given ***hypothesis space***, H, of functions $h(x)$ that can be considered as definitions of $c(x)$.

Generalization

- Hypotheses must generalize to correctly classify instances not in the training data.
- Simply memorizing training examples is a consistent hypothesis that does not generalize.
- *Occam's razor*:
  - "when you have two competing theories that make exactly the same predictions, the simpler one is the better."
  - Finding a *simple* hypothesis helps ensure generalization.

Ockham (Occam)’s Razor

- William of Ockham (1295-1349) was a Franciscan friar who applied the criteria to theology:
  - “Entities should not be multiplied beyond necessity” (Classical version but not an actual quote)
  - “The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single datum of experience.” (Einstein)
- Requires a precise definition of simplicity.
- Acts as a bias which assumes that nature itself is simple.
- Role of Occam’s razor in machine learning remains controversial.

Inductive Learning Hypothesis

- Any function that is found to approximate the target concept well on a sufficiently large set of training examples will also approximate the target function well on unobserved examples.
- Assumes that the training and test examples are drawn independently from the same underlying distribution.
- This is a fundamentally unprovable hypothesis unless additional assumptions are made about the target concept and the notion of “approximating the target function well on unobserved examples” is defined appropriately (cf. computational learning theory).

Inductive Bias

- Any means that a learning system uses to choose between two functions that are both consistent with the training data is called ***inductive bias***.

- Inductive bias can take two forms:
  - ***Language bias***: The language for representing concepts defines a hypothesis space that does not include all possible functions (e.g. conjunctive descriptions).
  - ***Search bias***: The language is expressive enough to represent all possible functions (e.g. disjunctive normal form) but the search algorithm embodies a preference for certain consistent functions over others (e.g. syntactic simplicity).

Unbiased Learning

- For instances described by *n* features each with *m* values, there are m^n instances. If these are to be classified into *c* categories, then there are $c^{m\text{^}n}$ possible classification functions.
  - For *n*=10, *m*=*c*=2, there are approx. $2^{2^{10}} = 3.4 \times 10^{308}$ possible functions, of which only $3^{10} = 59{,}049$ can be represented as conjunctions (an incredibly small percentage!)

- However, unbiased learning is futile since if we consider all possible functions then simply memorizing the data without any real generalization is as good an option as any.

Futility of Bias-Free Learning

- A learner that makes no *a priori* assumptions about the target concept has no rational basis for classifying any unseen instances.

- Inductive bias can also be defined as the assumptions that, when combined with the observed training data, logically entail the subsequent classification of unseen instances.
  - Training-data + inductive-bias $\rightarrow$ novel-classifications

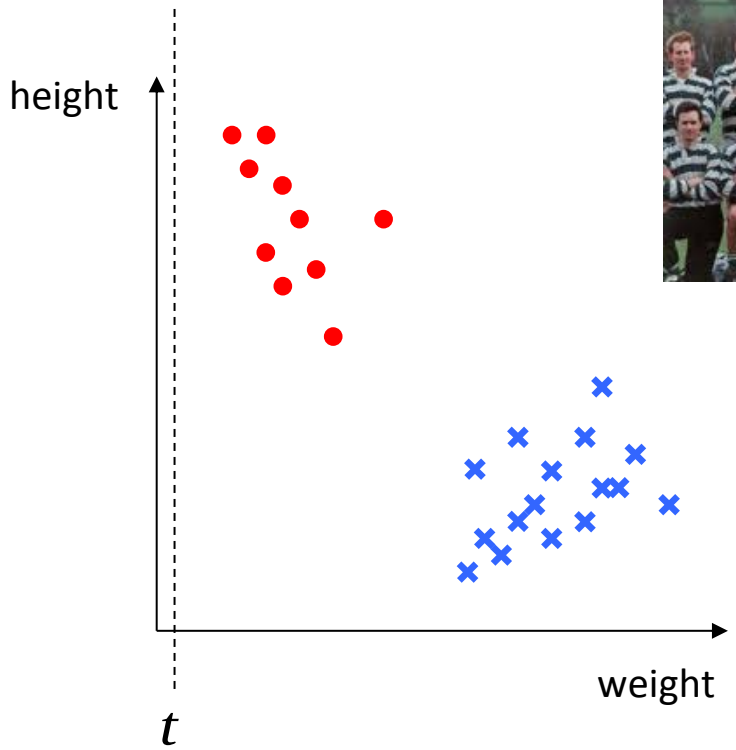
- The rote learner, which refuses to classify any instance unless it has seen it during training, is the least biased.

No Panacea

- No Free Lunch (NFL) Theorem (Wolpert, 1995)

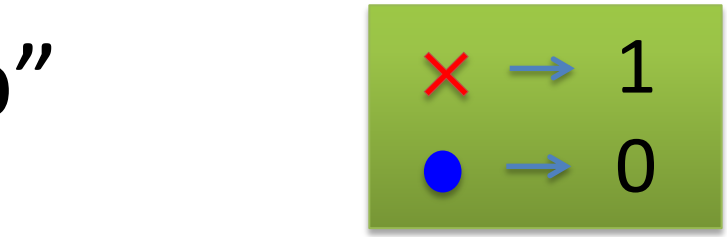
  Law of Conservation of Generalization Performance (Schaffer, 1994)
  - One can prove that improving generalization performance on unseen data for some tasks will always decrease performance on other tasks (which require different labels on the unseen instances).
  - Averaged across all possible target functions, no learner generalizes to unseen data any better than any other learner.
- There does not exist a learning method that is uniformly better than another for all problems.
- Given any two learning methods *A* and *B* and a training set, *D*, there always exists a target function for which *A* generalizes better (or at least as well) as *B*.
  - Train both methods on *D* to produce hypotheses h_A and h_B.
  - Construct a target function that labels all unseen instances according to the predictions of h_A.
  - Test h_A and h_B on any unseen test data for this target function and conclude that h_A is better.

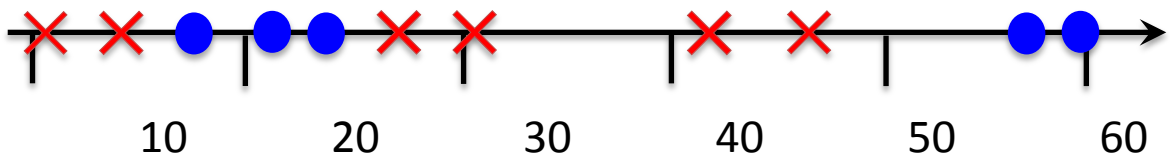
Threshold classifiers



$$\text{if } (weight > t) \text{ then"player"else "dancer"}$$

Also known as “decision stump”

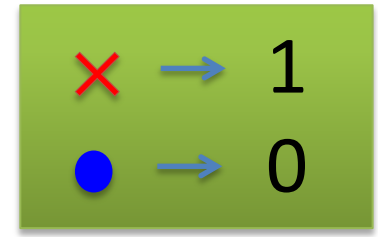


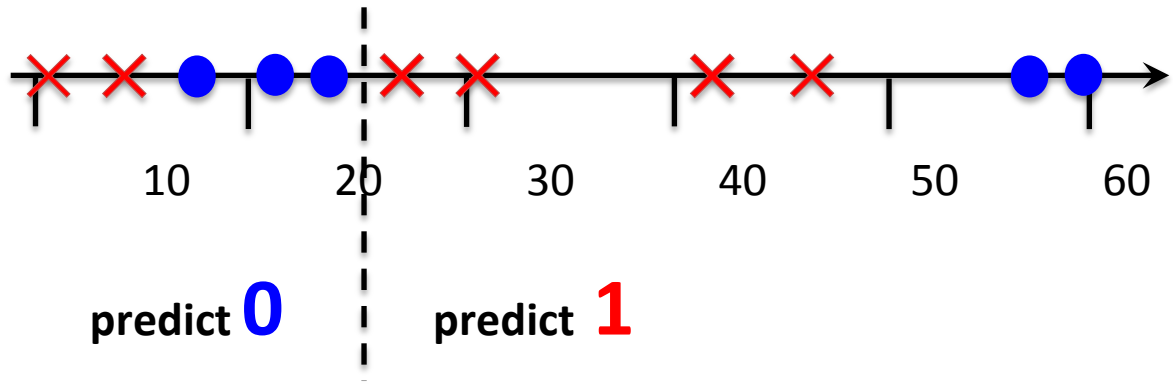


if $x > t$ then $\hat{y} = 1$ else $\hat{y} = 0$

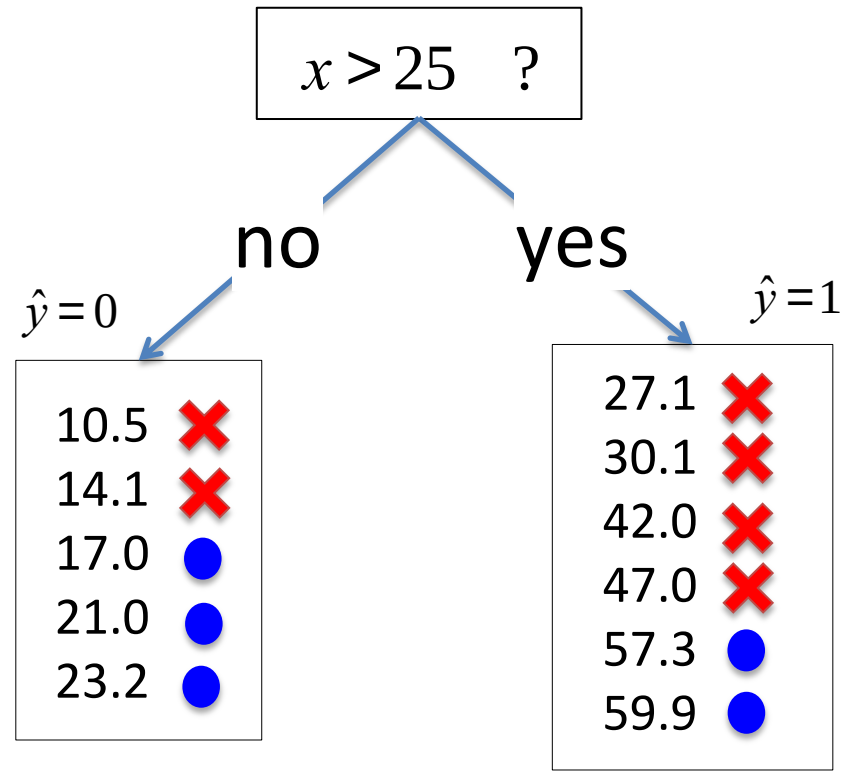
Q. Where is a good threshold?

Decision Stumps





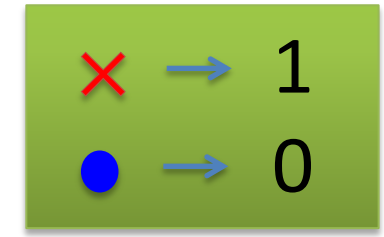
if $x > t$ then $\hat{y} = 1$ else $\hat{y} = 0$

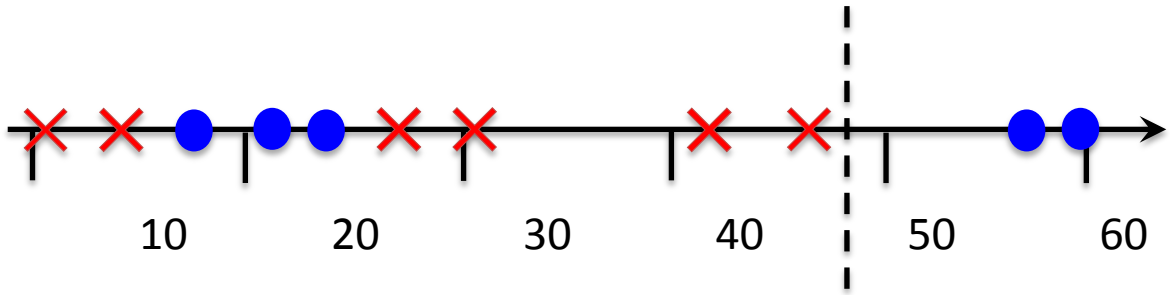


The stump "splits" the dataset.

Here we have 4 classification errors.

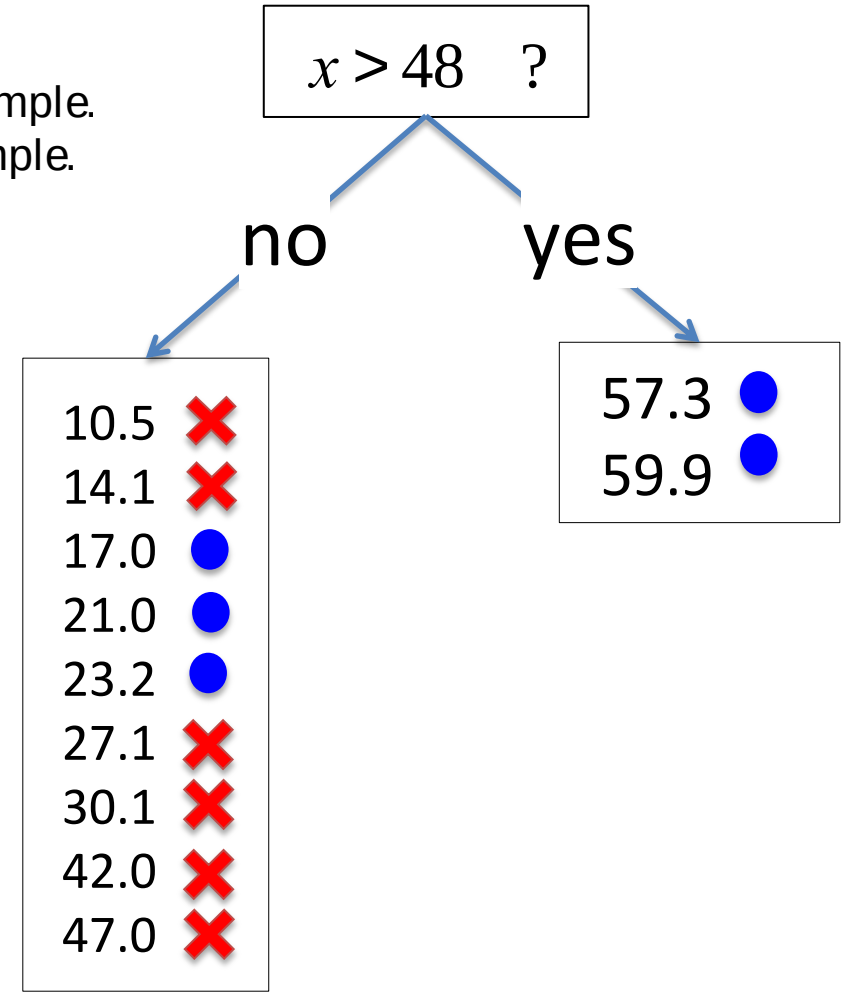
A modified stump





Set y_{right} to the most common label in the ($> t$) subsample.
Set y_{left} to the most common label in the ($< t$) subsample.

```
if x > t then
    predict ŷ = y_right
else
    predict ŷ = y_left
endif
```



Here we have 3 classification errors.

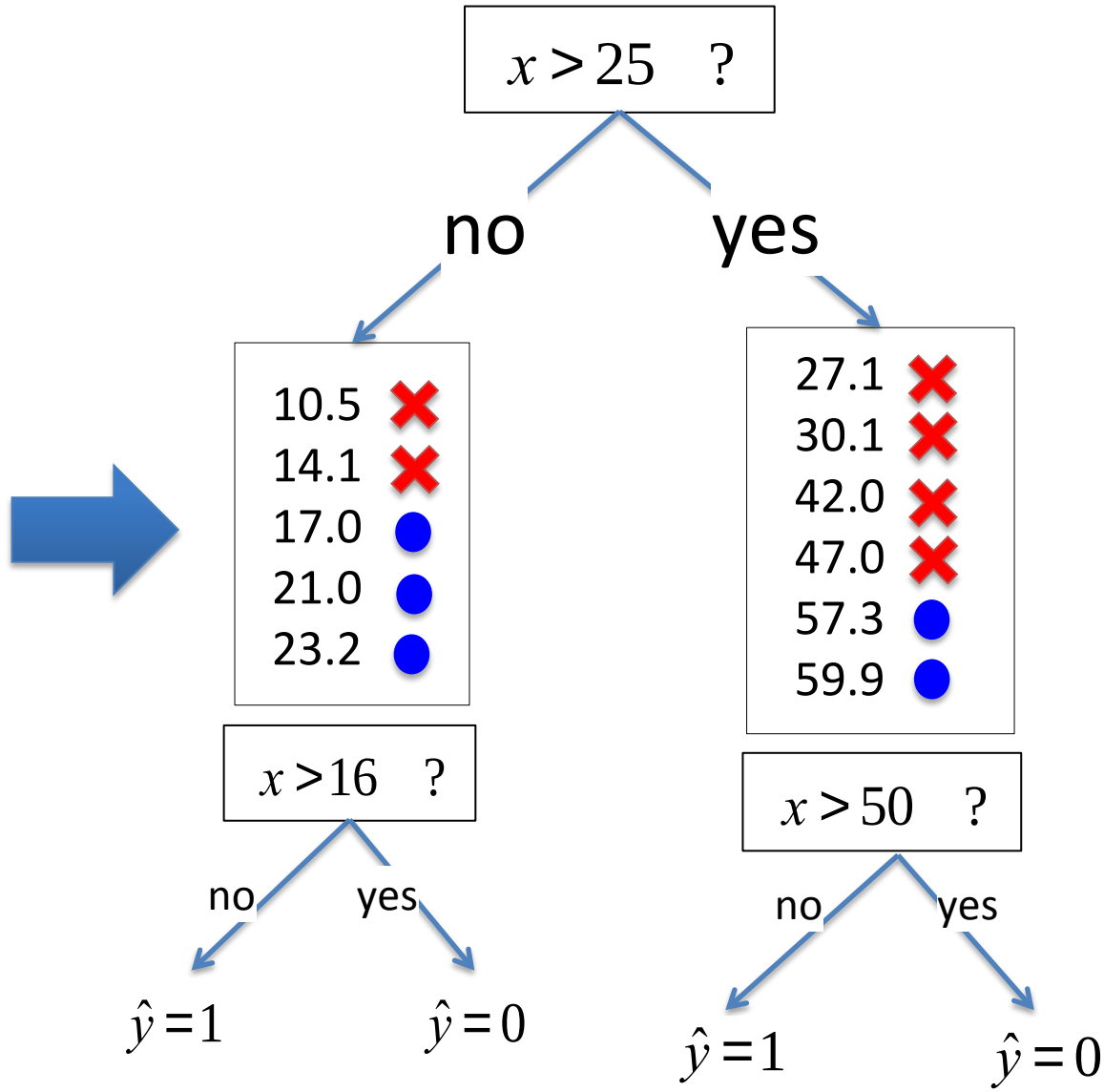
From Decision Stumps, to Decision Trees

- New type of **non-linear model**
- Copes naturally with continuous and **categorical data**
- **Fast** to both train and test (highly parallelizable)
- Generates a set of **interpretable** rules

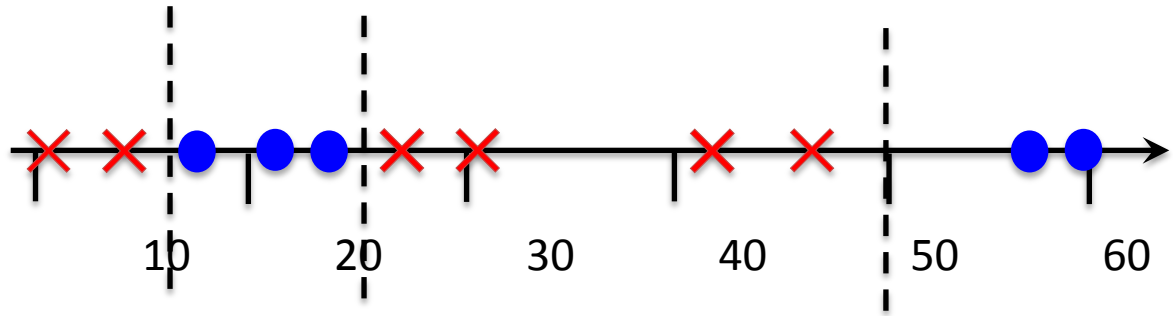
Recursion…

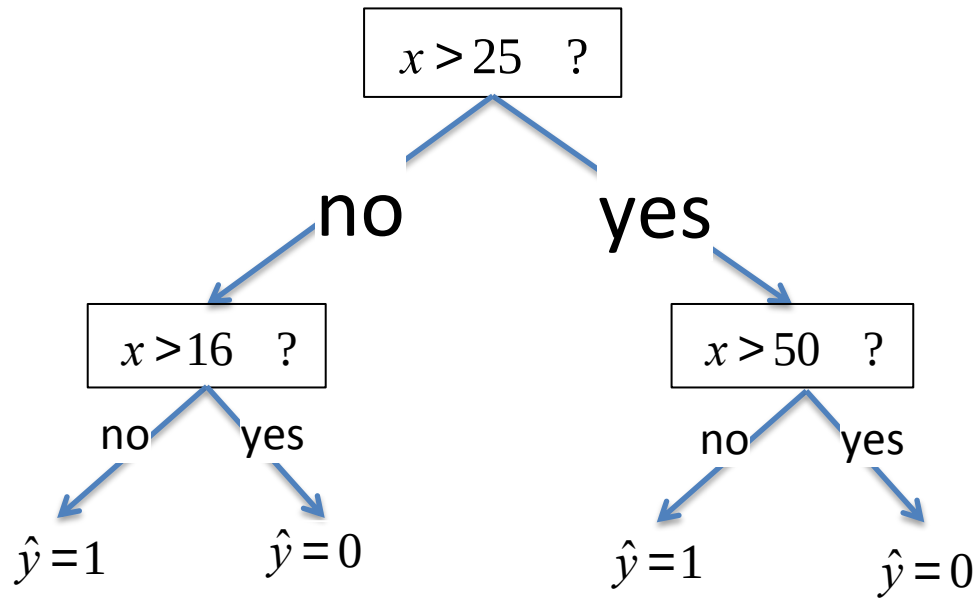
Just another dataset!

Build a stump!



Decision Trees = nested rules

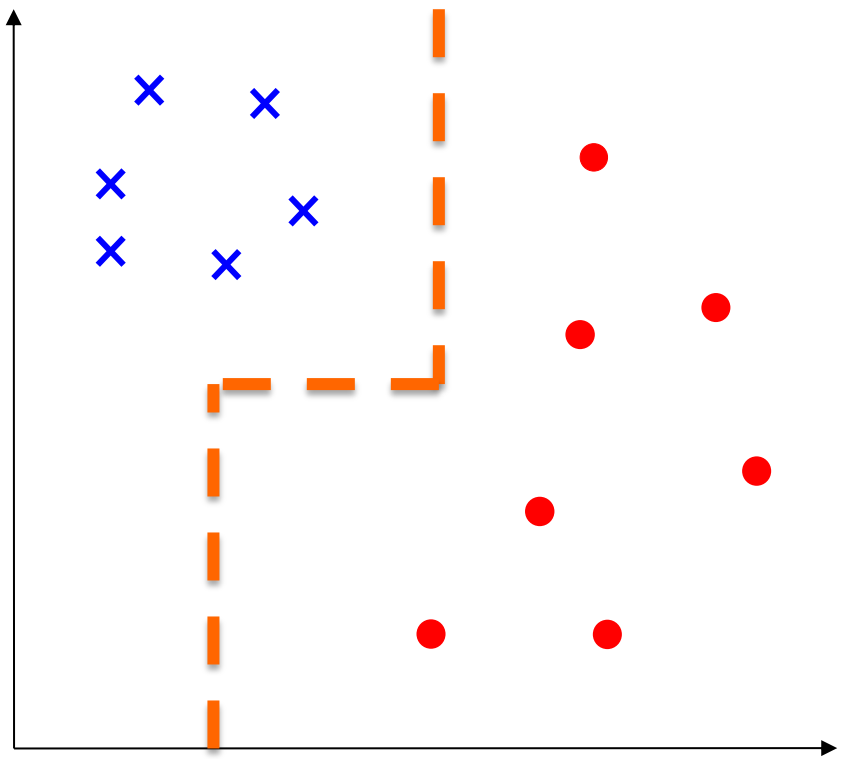




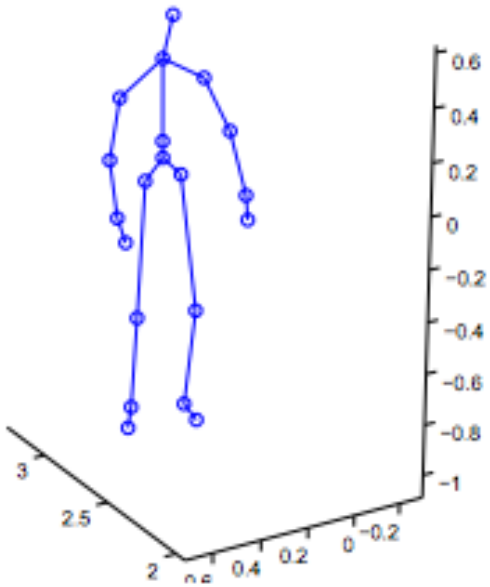

```
if x>25 then
   if x>50 then y=0 else y=1; endif
else
   if x>16 then y=0 else y=1; endif
endif
```

Trees build “orthogonal” decision boundaries.

Boundary is piecewise, and at 90 degrees to feature axes.

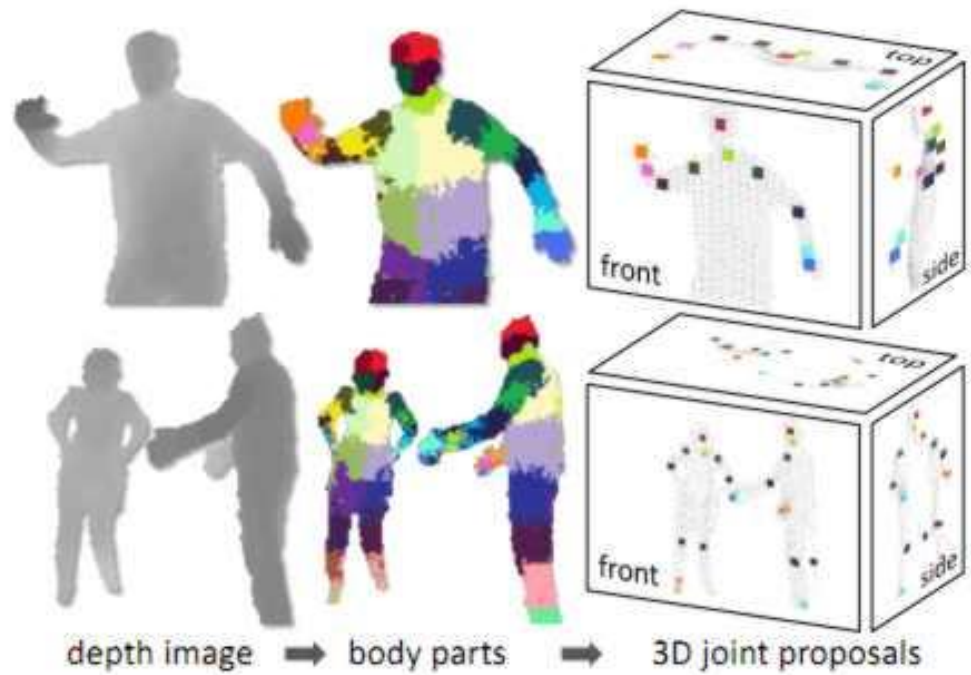


Decision trees can be seen as nested rules.
Nested rules are FAST, and highly parallelizable.

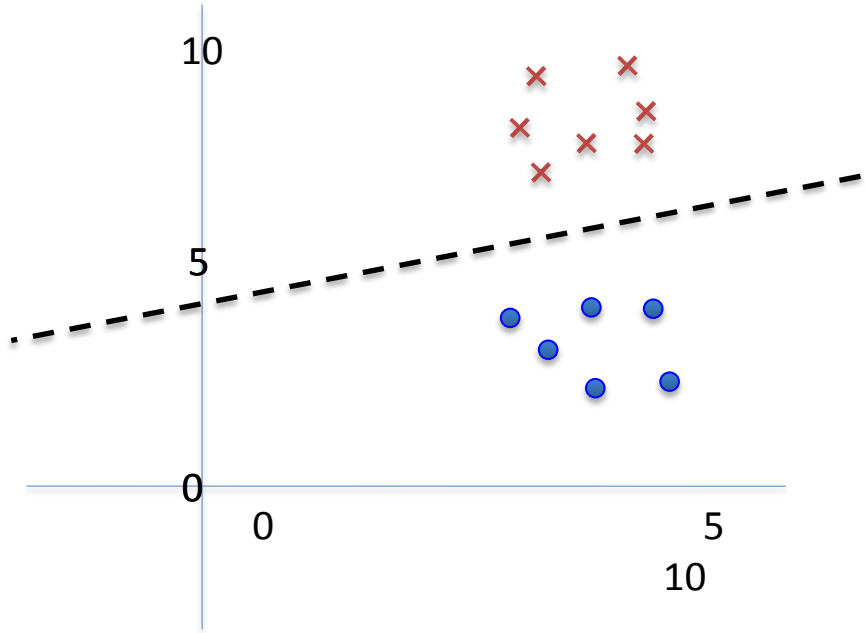


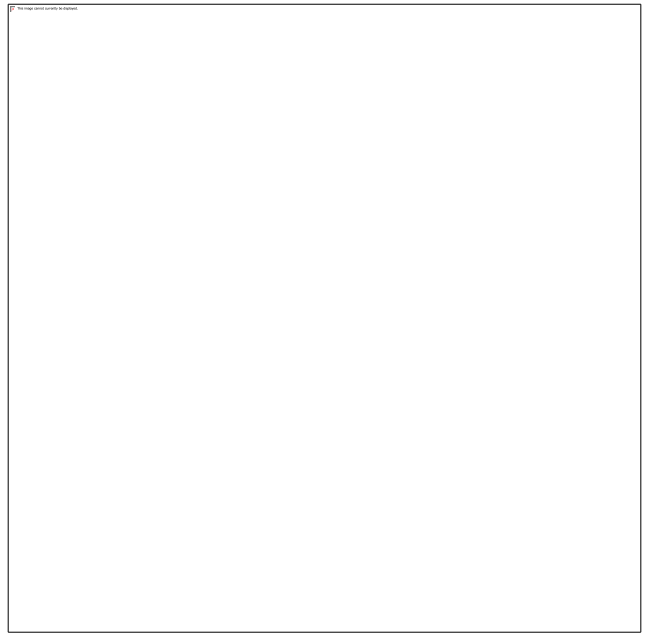
x,y,z-coordinates per joint, ~60 total
x,y,z-velocities per joint, ~60 total
joint angles (~35 total)
joint angular velocities (~35 total)

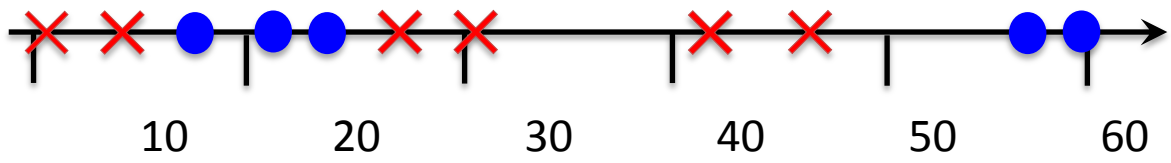
```
if x>25 then
    if x>50 then y=0 else y=1; endi
else
    if x>16 then y=0 else y=1; endi
endif
```



We've been assuming **continuous** variables!

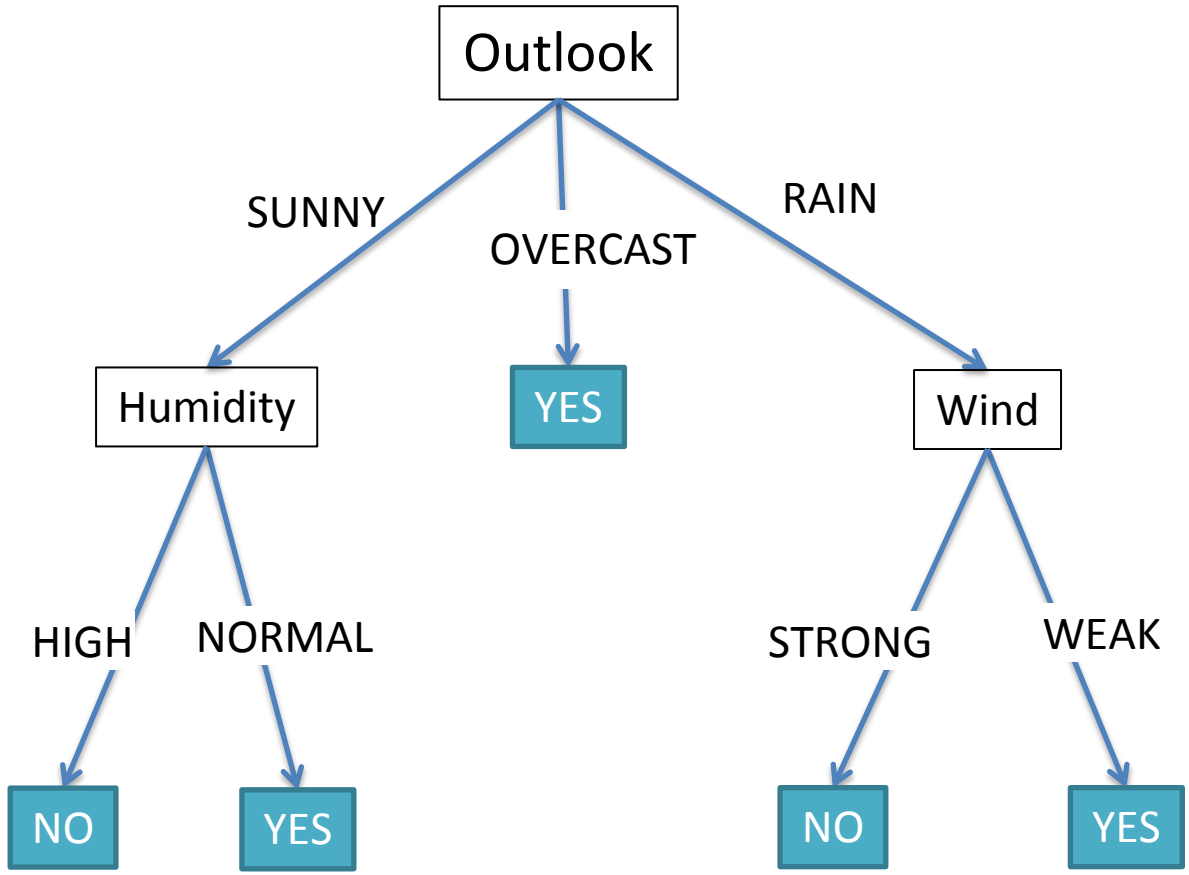






The Tennis Problem

	Outlook	Temperature	Humidity	Wind	Play Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No




```
if ( Outlook==sunny AND Humidity==high )      then NO
if ( Outlook==sunny AND Humidity==normal )    then YES
if ( Outlook==overcast )                      then YES
if ( Outlook==rain AND Wind==strong )         then NO
if ( Outlook==rain AND Wind==weak )           then YES
```

Decision Tree Learning Algorithm (sometimes called "ID3")

```
1: function buildTree( subsample, depth )
2:
3:     //BASE CASE:
4:     if (depth == 0) OR (all examples have same label) then
5:         return most common label in the subsample
6:     end if
7:
8:     //RECURSIVE CASE:
9:     for each feature do
10:        Try splitting the data (i.e. build a decision stump)
11:        Calculate the cost for this stump
12:    end for
13:    Pick feature with minimum cost
14:
15:    Find left/right subsamples
16:    Add left branch ← buildTree( leftSubSample, depth − 1 )
17:    Add right branch ← buildTree( rightSubSample, depth − 1 )
18:
19:    return tree
20:
21: end function
```

Picking a Good Split Feature

- Goal is to have the resulting tree be as small as possible, per Occam's razor.
- Finding a minimal decision tree (nodes, leaves, or depth) is an NP-hard optimization problem.
- Top-down divide-and-conquer method does a greedy search for a simple tree but does not guarantee to find the smallest.
  - General lesson in ML: "Greed is good."
- Want to pick a feature that creates subsets of examples that are relatively "pure" in a single class so they are "closer" to being leaf nodes.

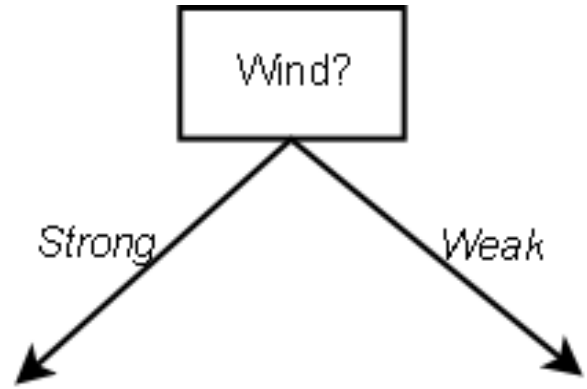
The Tennis Problem

	Outlook	Temperature	Humidity	Wind	Play Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Note: 9 examples say “YES”, while 5 say “NO”.

Partitioning the data…

	Outlook	Temperature	Humidity	Wind	Play Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



	Outlook	Temp	Humid	Wind	Play?
2	Sunny	Hot	High	Strong	No
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
14	Rain	Mild	High	Strong	No

3 examples say yes, 3 say no.

	Outlook	Temp	Humid	Wind	Play?
1	Sunny	Hot	High	Weak	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
13	Overcast	Hot	Normal	Weak	Yes

6 examples say yes, 2 examples say no.

Thinking in Probabilities...

Before the split : 9 'yes', 5 'no', ... $p('yes') = \frac{9}{14} \approx 0.64$

On the left branch : 3 'yes', 3 'no', ... $p('yes') = \frac{3}{6} = 0.5$

On the right branch : 6 'yes', 2 'no', ... $p('yes') = \frac{6}{8} = 0.75$

Remember... $p('no') = 1 - p('yes')$

Entropy

- Entropy of a set of examples, S, relative to a binary classification is:

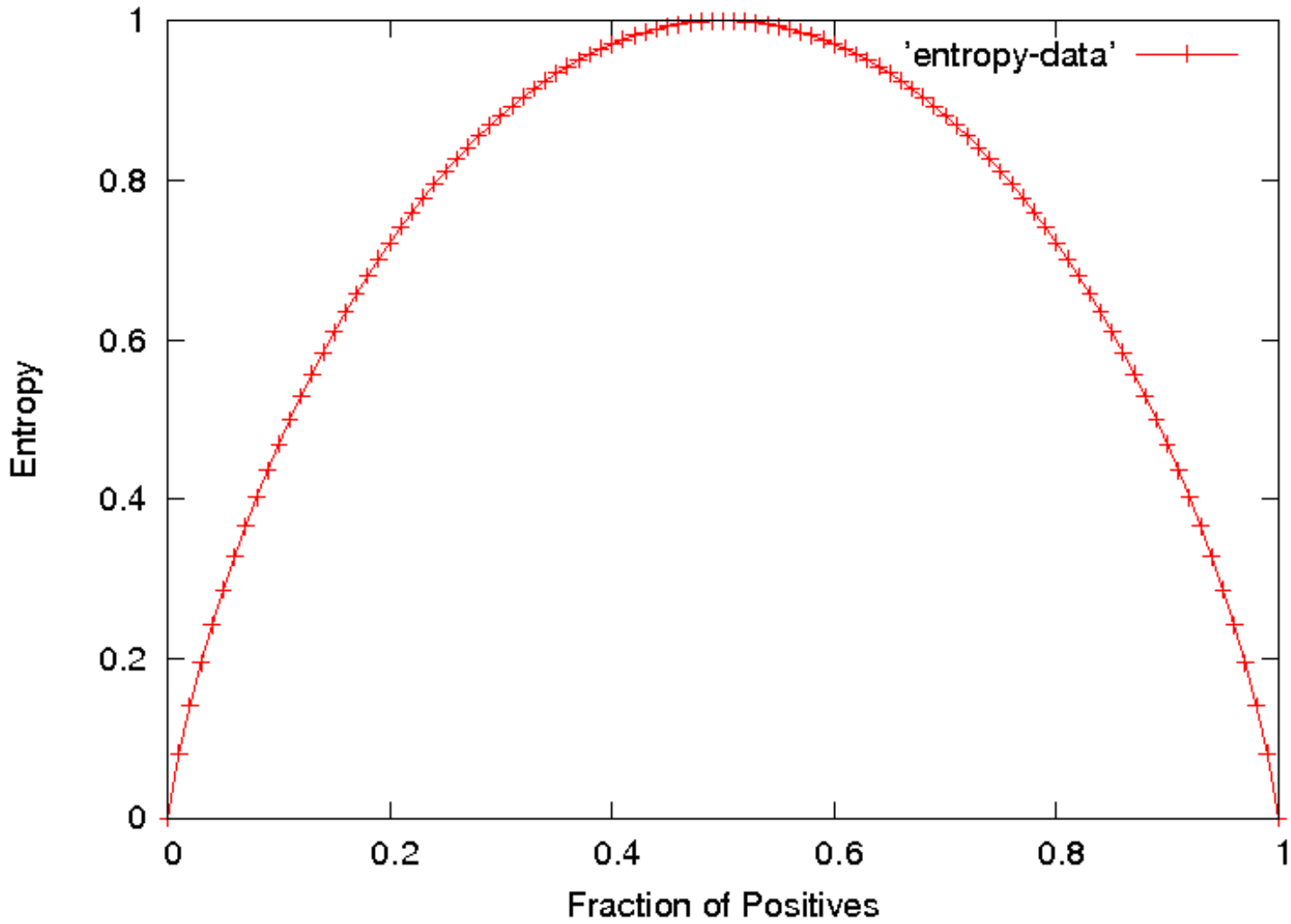
$$Entropy(S) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$

  where p_1 is the fraction of positive examples in S and p_0 is the fraction of negatives.
- If all examples are in one category, entropy is zero (we define $0 \cdot \log(0)=0$)
- If examples are equally mixed ($p_1=p_0=0.5$), entropy is a maximum of 1.
- Entropy can be viewed as the number of bits required on average to encode the class of an example in *S* where data compression (e.g. Huffman coding) is used to give shorter codes to more likely cases.
- For multi-class problems with c categories, entropy generalizes to:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

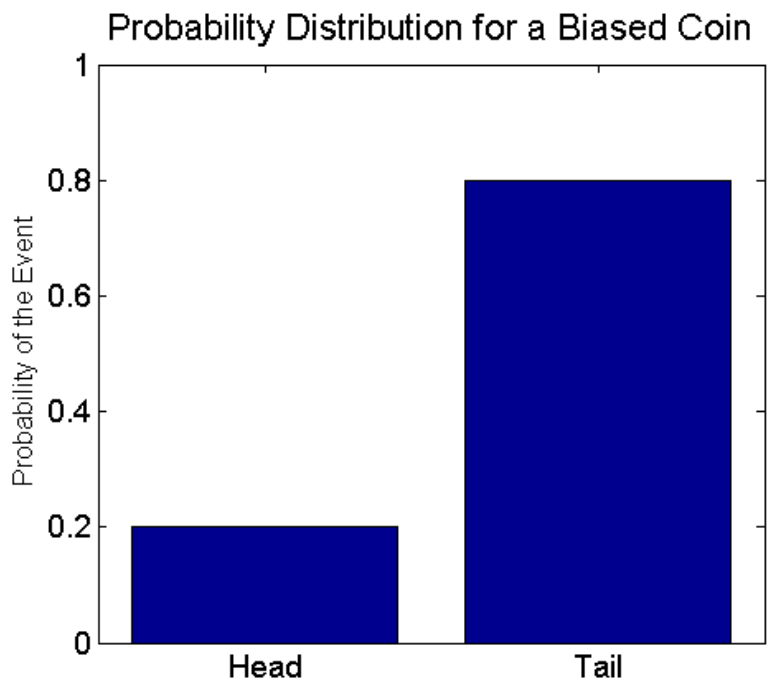
We often write H(S) for *Entropy(S).*

Entropy Plot for Binary Classification



The “Information” in a feature

Less uncertainty = more information



H(X) = 0.72193

Calculating Entropy

The variable of interest is "T" (for tennis), taking on 'yes' or 'no' values. Before the split : 9 'yes', 5 'no', ...
$p('yes') = \frac{9}{14} \approx 0.64$

In the whole dataset, the entropy is:

$$\begin{aligned} H(T) &= -\sum_i p(x_i) \log p(x_i) \\ &= -\left\{ \frac{5}{14} log \frac{5}{14} + \frac{9}{14} log \frac{9}{14} \right\} = 0.94029 \end{aligned}$$

$H(T)$ is the entropy **before** we split.

Information Gain, also known as "Mutual Information"

$H(T)$ is the entropy before we split.

$H(T|W = \textit{strong})$ is the entropy of the data on the left branch.
$H(T|W = \textit{weak})$ is the entropy of the data on the right branch.

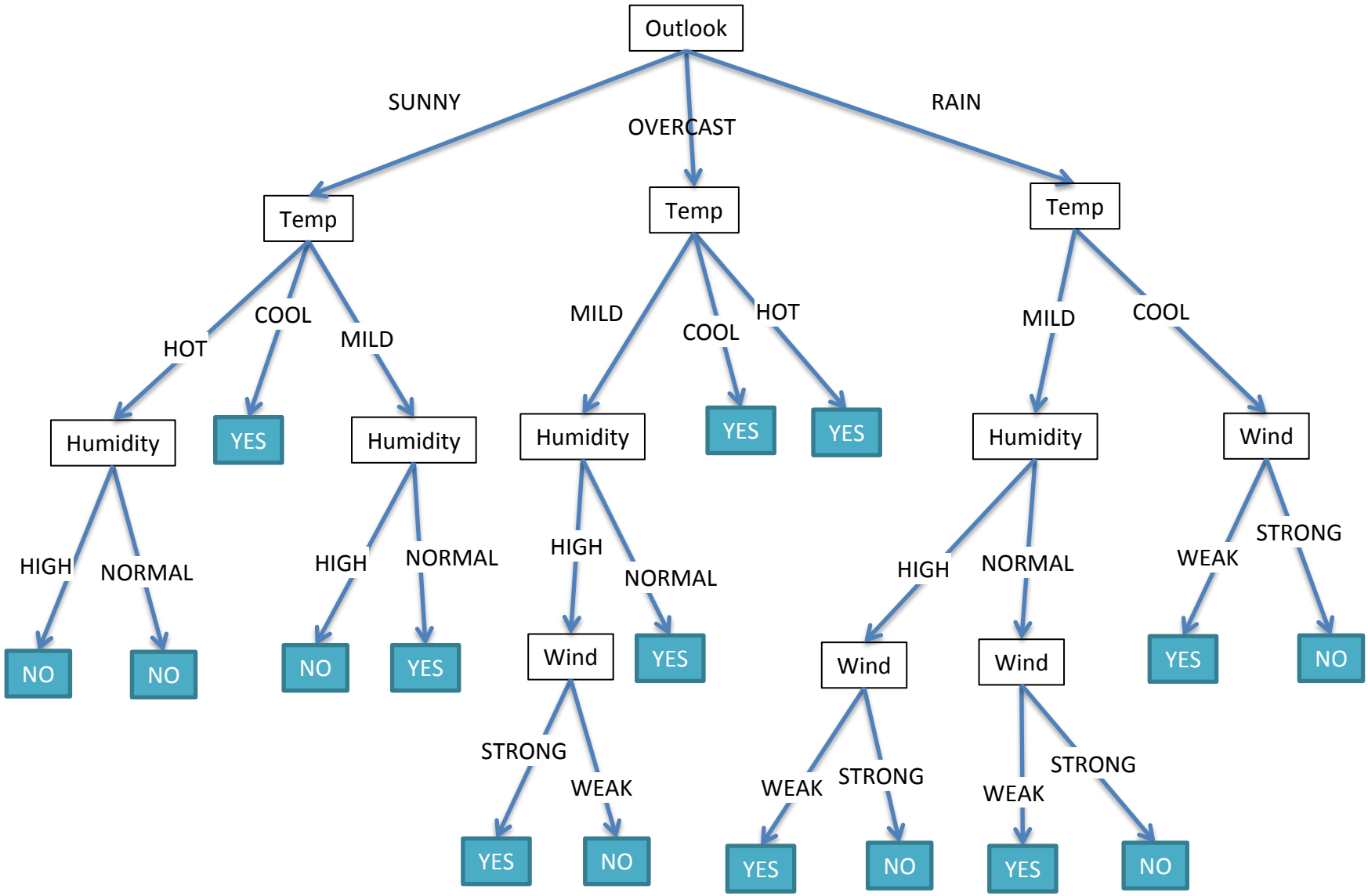
$H(T|W)$ is the weighted average of the two.

Choose the feature with maximum value of $H(T) - H(T|W)$.

Decision Tree Learning Algorithm (sometimes called "ID3")

```
1:  function BUILDTREE( subsample, depth )
2:
3:      //BASE CASE:
4:      if (depth == 0) OR (all examples have same label) then
5:          return most common label in the subsample
6:      end if
7:
8:      //RECURSIVE CASE:
9:      for each feature do
10:         Try splitting the data (i.e. build a decision stump)
11:         Calculate gain for this stump
12:     end for
13:     Pick feature with ~~minimum cost~~
14:
15:     Find left/right subsamples
16:     Add left branch ← BUILDTREE( leftSubSample, depth − 1 )
17:     Add right branch ← BUILDTREE( rightSubSample, depth − 1 )
18:
19:     return tree
20:
21: end function
```

maximum information gain


Outlook
SUNNY
OVERCAST
RAIN
Temp
Temp
Temp
HOT
COOL
MILD
MILD
COOL
HOT
MILD
COOL
Humidity
YES
Humidity
Humidity
YES
YES
Humidity
Wind
HIGH
NORMAL
HIGH
NORMAL
HIGH
NORMAL
HIGH
NORMAL
WEAK
STRONG
NO
NO
NO
YES
Wind
YES
Wind
Wind
YES
NO
STRONG
WEAK
WEAK
STRONG
WEAK
STRONG
YES
NO
YES
NO
YES
NO

Gini impurity (not to be confused wity Gini coefficient in economics)

- Gini impurity of a set of examples, S, relative to a binary classification is: $p_1 p_0 + p_0 p_1 = 1 - {p_1}^2 - {p_0}^2$

  where p_1 is the fraction of positive examples and p_0 is the fraction of negatives.
- If all examples are in one category, Gini impurity is zero
- If examples are equally mixed (p_1=p_0=0.5), Gini impurity is a maximum of 0.5.
- When : $p_1 = 0.8$, Gini impurity is 0.32
- For multi-class problems with c categories, Gini inpurity generalizes to: $1 - \sum {p_i}^2$
- Gini impurity measures how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.
- Used in the CART (classification and regression tree) algorithm

Bias in Decision-Tree Induction

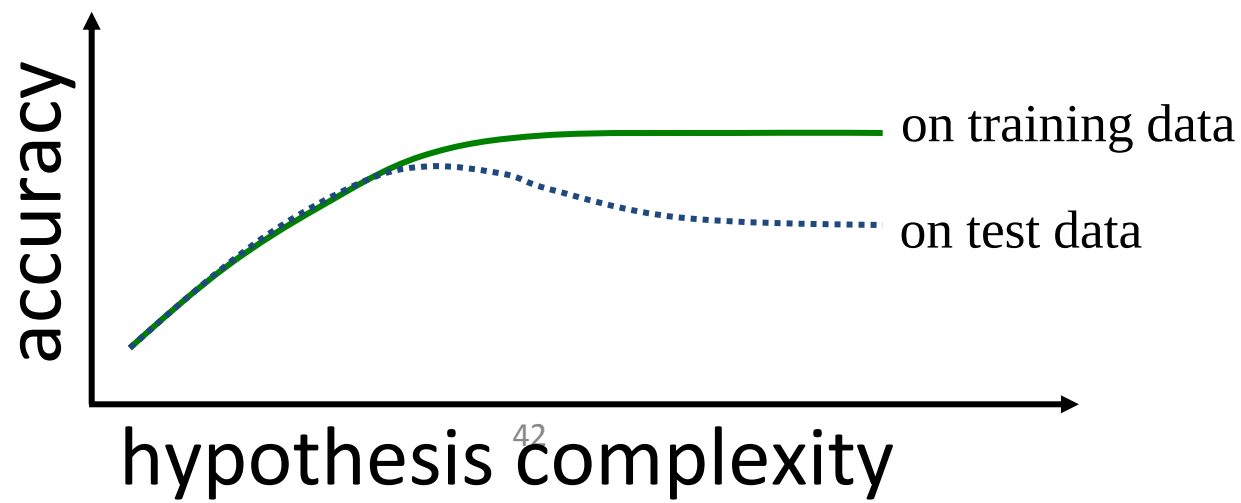
- Using either information-gain or Gini impurity gives a bias for trees with smaller depths.
- This is a search (preference) bias instead of a language (restriction) bias.

Properties of Decision Tree Learning

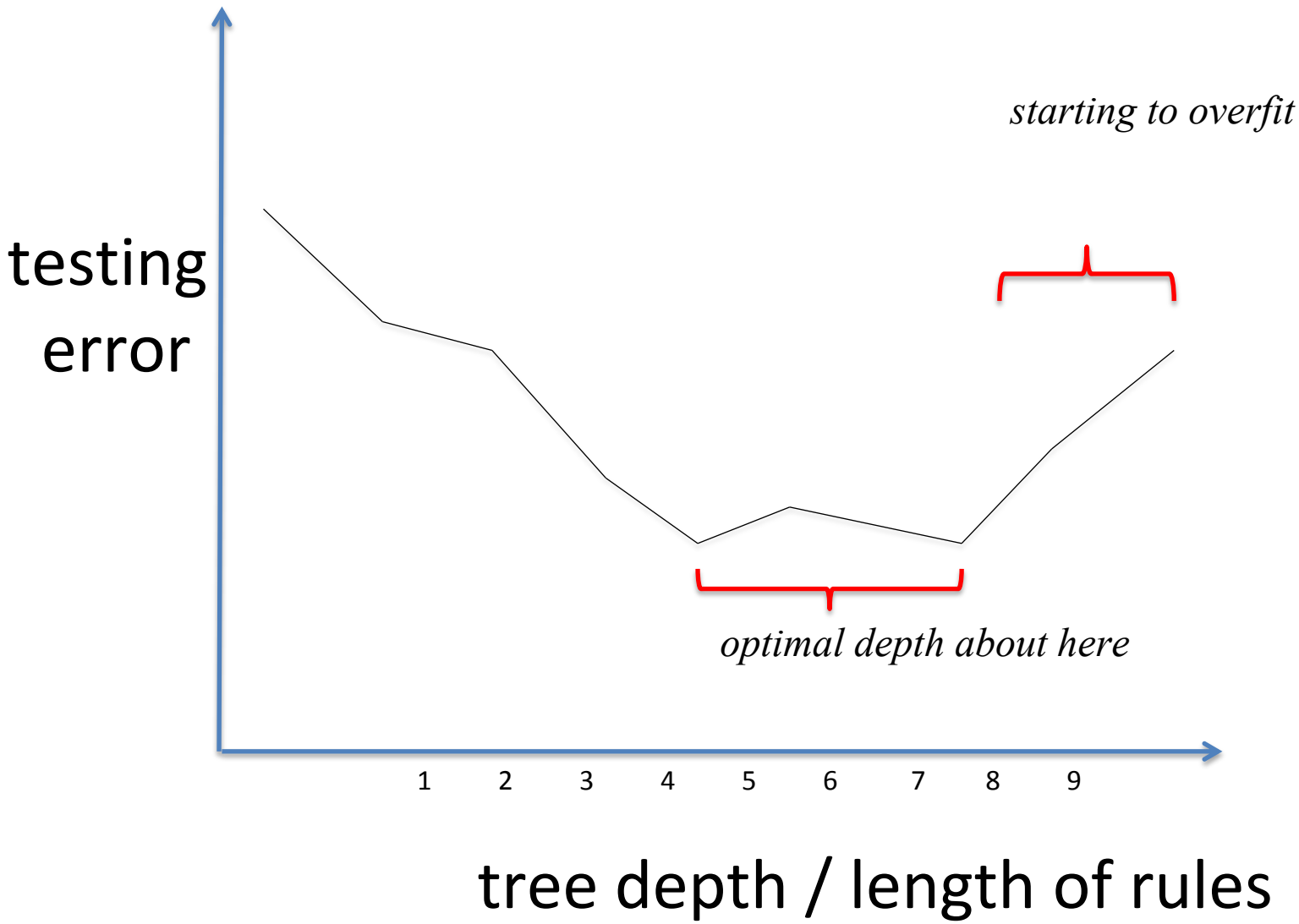
- Continuous (real-valued) features can be handled by allowing nodes to split a real valued feature into two ranges based on a threshold (e.g. length < 3 and length ≥3)
- Classification trees have discrete class labels at the leaves, ***regression trees*** allow real-valued outputs at the leaves.
- Algorithms for finding consistent trees are efficient for processing large amounts of training data for data mining tasks.

Overfitting

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization to unseen data.
  - There may be noise in the training data that the tree is erroneously fitting.
  - The algorithm may be making poor decisions towards the leaves of the tree that are based on very little data and may not reflect reliable trends.
- A hypothesis, *h*, is said to overfit the training data is there exists another hypothesis which, *h´*, such that *h* has less error than *h´* on the training data but greater error on independent test data.



Overfitting in Decision Trees; Depth is main parameter



Overfitting Prevention (Pruning) Methods

- Two basic approaches for pruning decision trees
  - Prepruning: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
    - Limiting depth to be under a threshold
    - Statistical test: Use a statistical test on the training data to determine if any observed regularity can be dismisses as likely due to random chance.
  - Postpruning: Grow the full tree, then remove subtrees.
    - Could use Cross-validation: Reserve some training data as a hold-out set (***validation set***, ***tuning set***) to evaluate utility of subtrees.

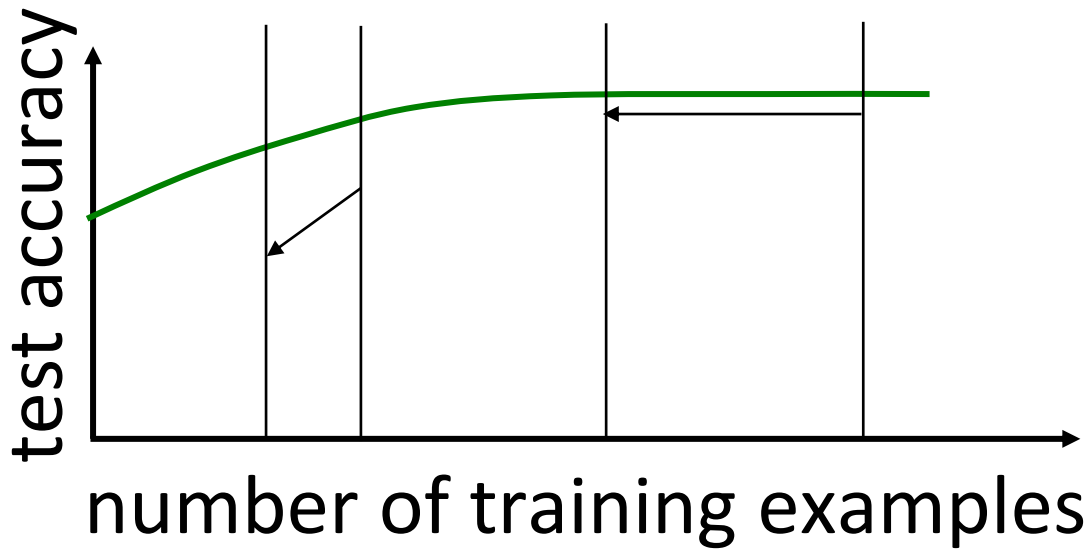
Reduced Error Pruning

- A post-pruning, cross-validation approach.

Partition training data in “train” and “validation” sets.
Build a complete tree from the “train” data.
Until accuracy on validation set decreases do:
    For each non-leaf node, n, in the tree do:
        Temporarily prune the subtree below n and replace it with a leaf labeled with the current majority class at that node.
        Measure and record the accuracy of the pruned tree on the validation set.
    Permanently prune the node that results in the greatest increase in accuracy on the validation set.

Issues with Reduced Error Pruning

- The problem with this approach is that it potentially "wastes" training data on the validation set.
- Severity of this problem depends where we are on the learning curve:



Different Flavors of Decision Trees

- ID3, or alternative Dichotomizer, was the first of three Decision Tree implementations developed by Ross Quinlan (Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106.) Only categorical predictors and no pruning.

- C4.5, Quinlan's next iteration. The new features (versus ID3) are: (i) accepts both continuous and discrete features; (ii) handles incomplete data points; (iii) solves over-fitting problem by (very clever) bottom-up technique usually known as "pruning"; and (iv) different weights can be applied the features that comprise the training data.

Different flavors

- C5.0, The most significant feature unique to C5.0 is a scheme for deriving rule sets. After a tree is grown, the splitting rules that define the terminal nodes can sometimes be simplified: that is, one or more condition can be dropped without changing the subset of observations that fall in the node.

- CART or Classification And Regression Trees is often used as a generic acronym for the term Decision Tree, though it apparently has a more specific meaning. In sum, the CART implementation is very similar to C4.5. Weka includes Java version of C4.5 called J48.