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 $c(x)$: $<x, c(x)>$ for an unknown categorization function, c.
- Given a set of training examples, D.
- Find a hypothesized categorization function, $h(x)$, such that:

$$\forall < x, c(x) > \in D : h(x) = c(x)$$

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 $c^{m\wedge n}$ possible classification functions.
  - For n=10, m=c=2, there are approx. $3.4\text{x}10^{38}$ possible functions, of which only 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.

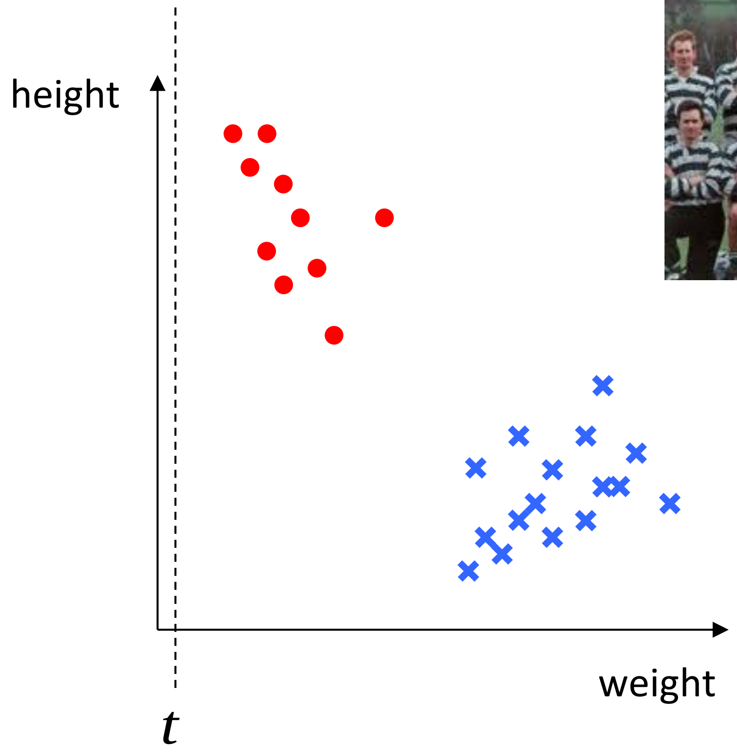
- Inductive bias can also be defined as the assumptions that, when combined with the observed training data, logically entail the subsequent classification of unseen instances.
  - Training-data + inductive-bias **|—** novel-classifications

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

No Panacea

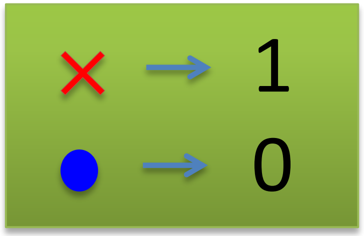
- No Free Lunch (NFL) Theorem (Wolpert, 1995)
  Law of Conservation of Generalization Performance (Schaffer, 1994)
  - One can prove that improving generalization performance on unseen data for some tasks will always decrease performance on other tasks (which require different labels on the unseen instances).
  - 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.

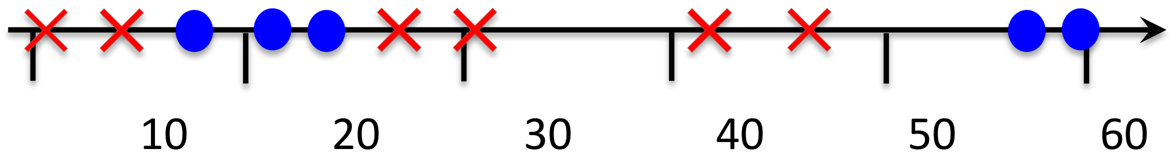
Threshold classifiers



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

Also known as "decision stump"

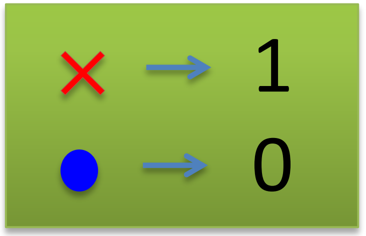


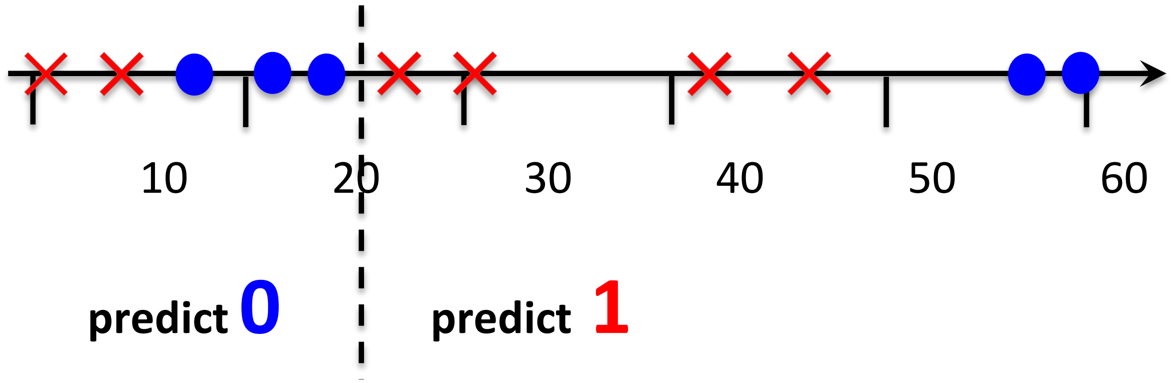


if $x > t$ then $\hat{y} = 1$ else $\hat{y} = 0$

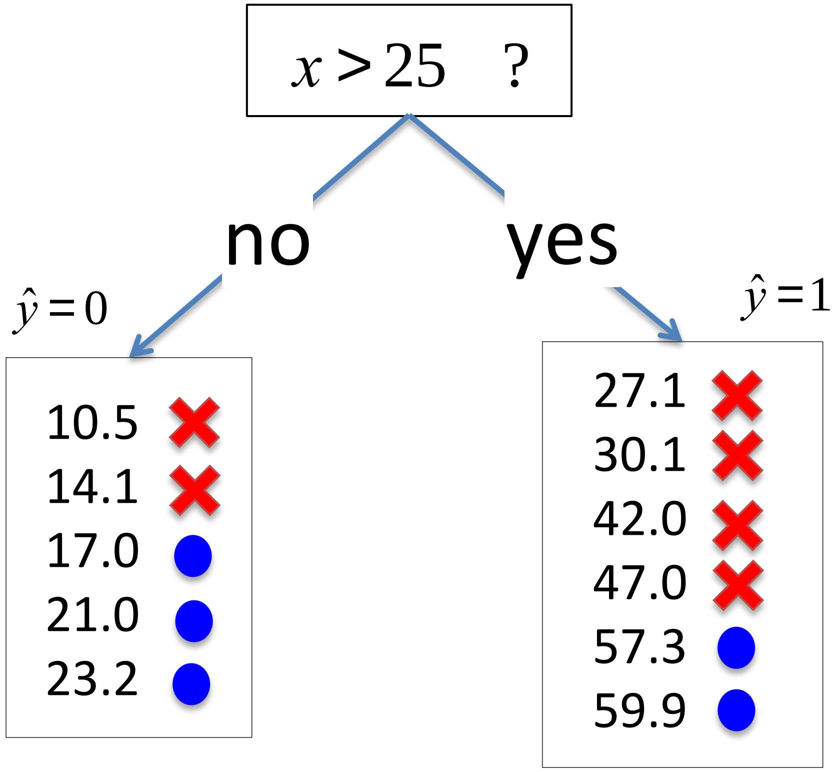
Q. Where is a good threshold?

Decision Stumps





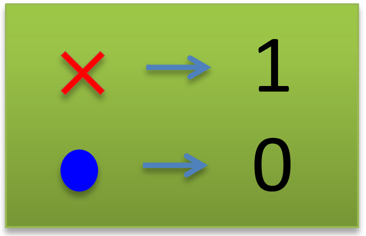
if $x > t$ then $\hat{y} = 1$ else $\hat{y} = 0$

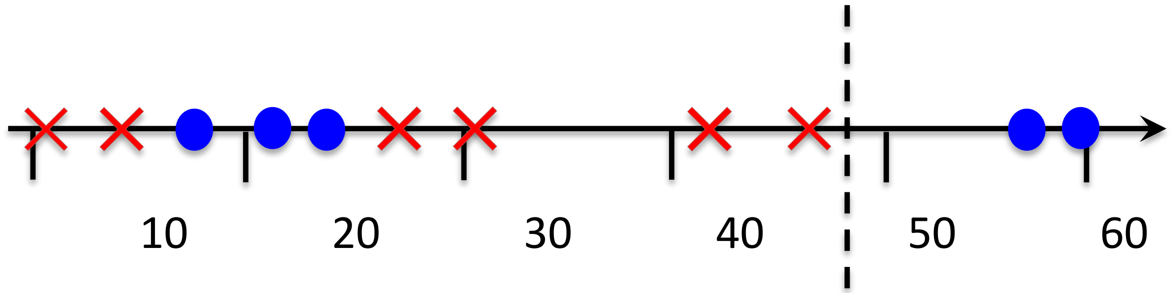


The stump "splits" the dataset.

Here we have 4 classification errors.

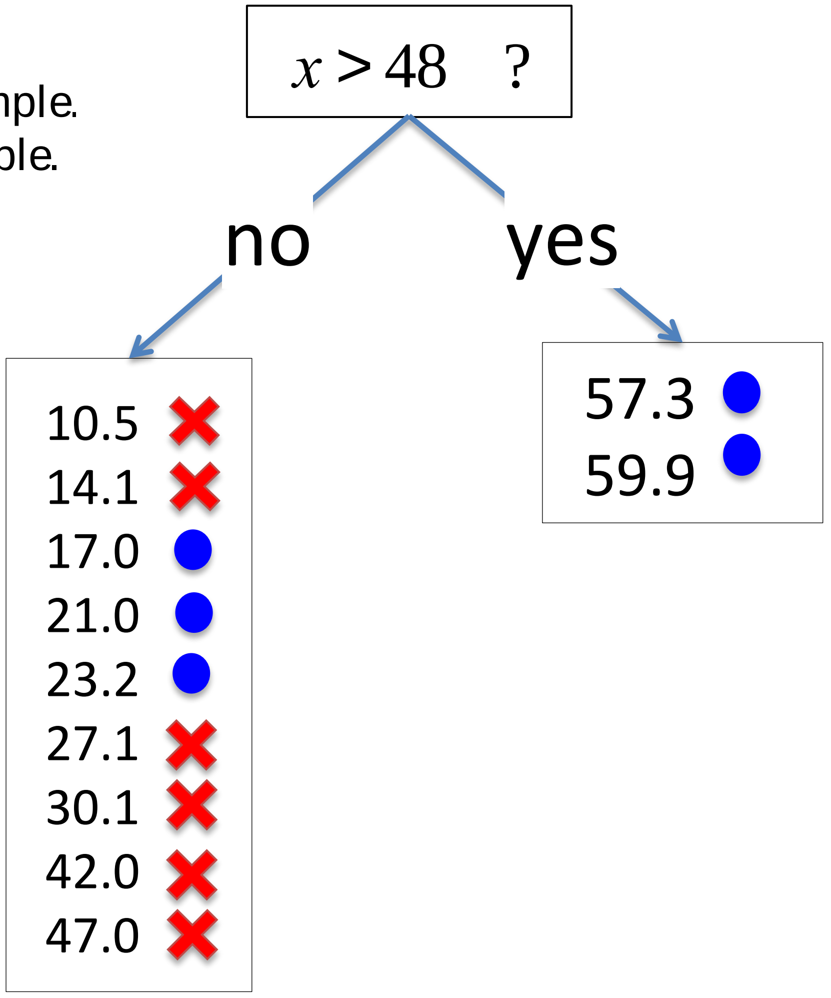
A modified stump





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 $\hat{y} = y_{right}$
else
    predict $\hat{y} = y_{left}$
endif
```



Here we have 3 classification errors.

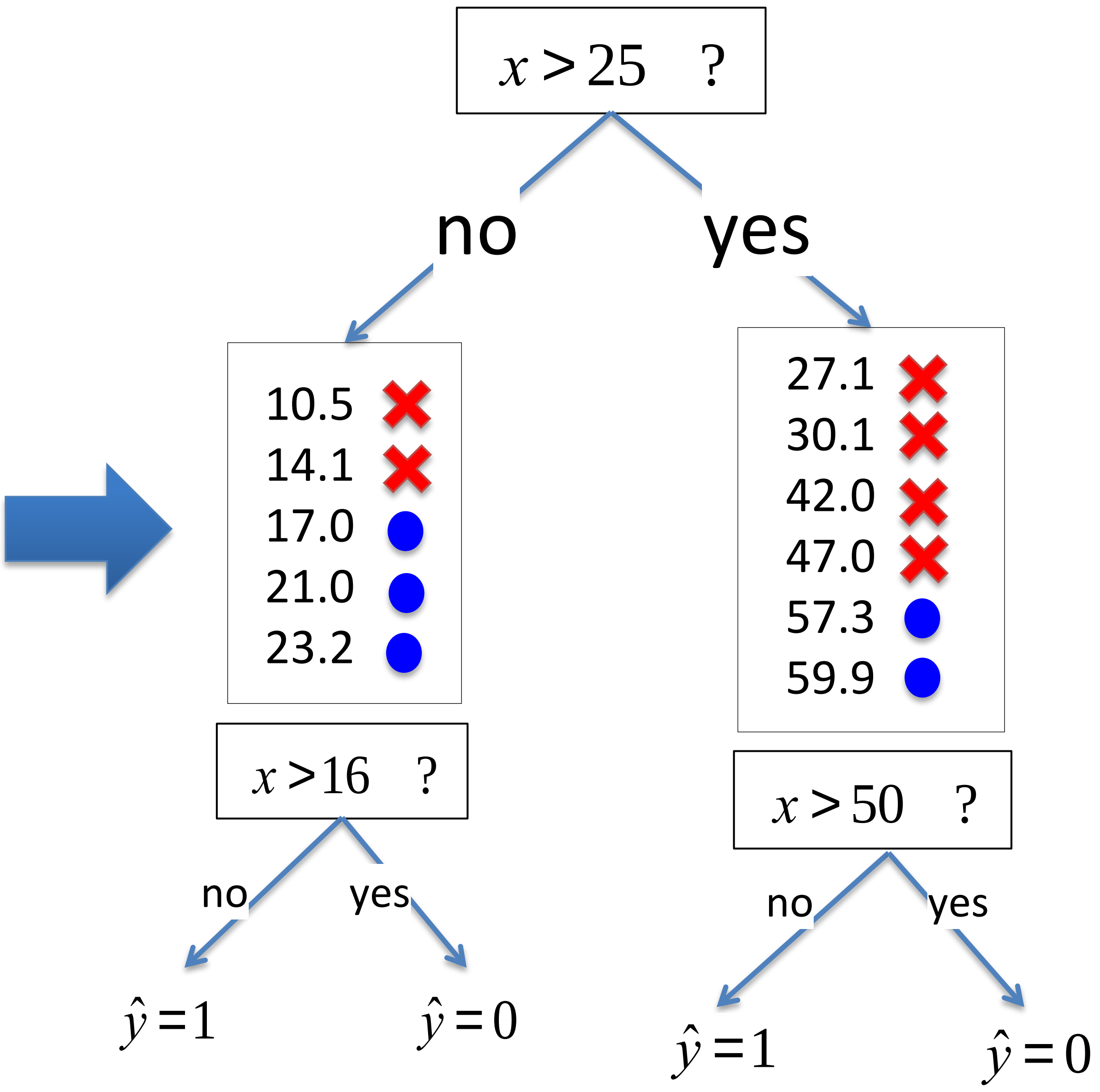
From Decision Stumps, to Decision Trees

- 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

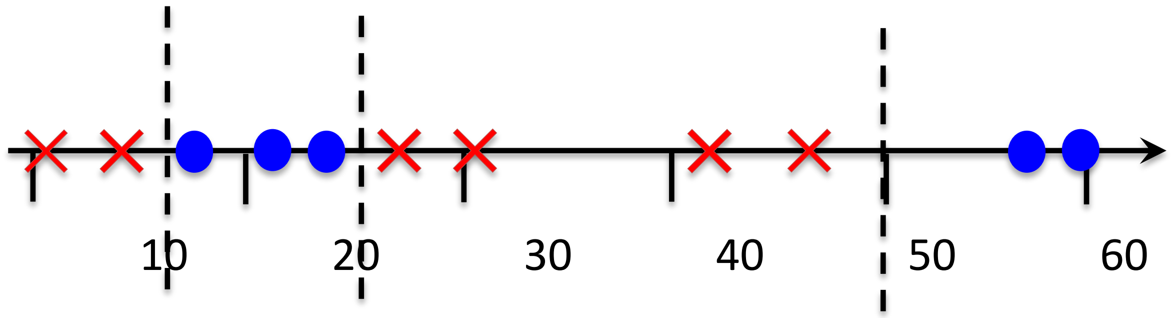
Recursion…

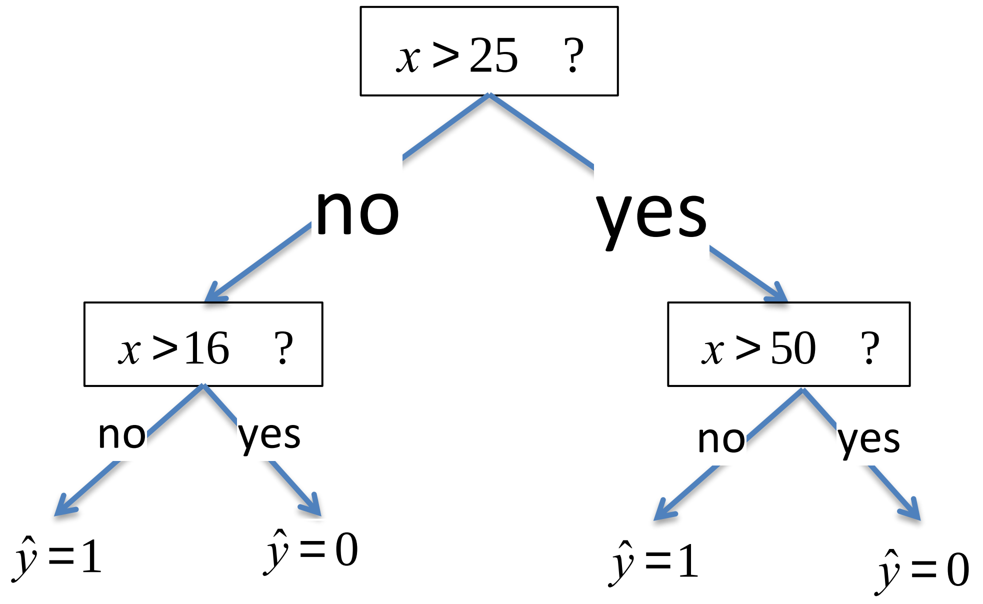
Just another dataset!

Build a stump!



Decision Trees = nested rules

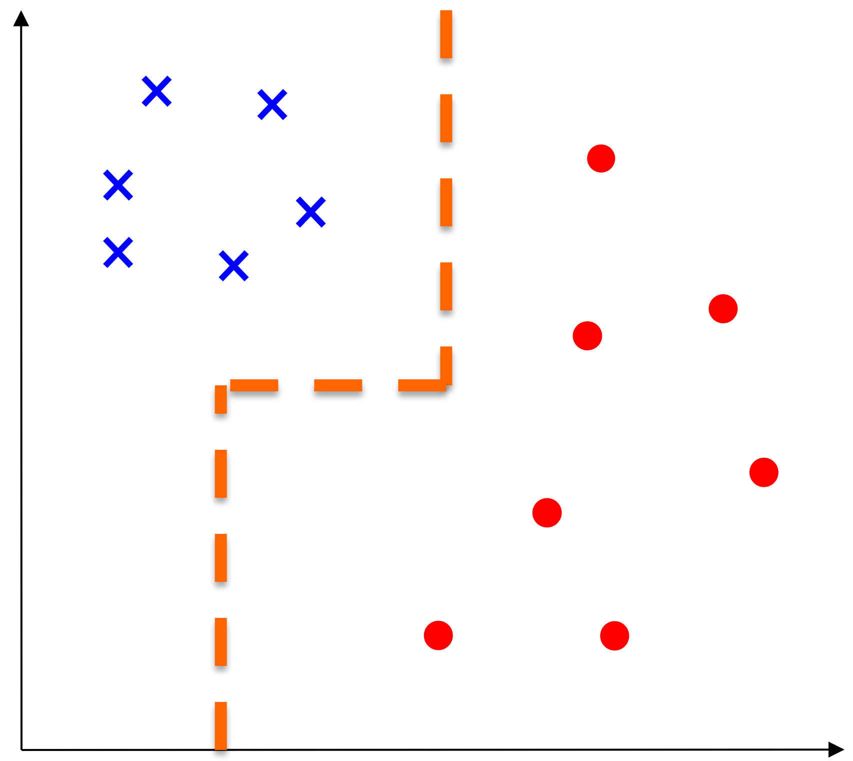




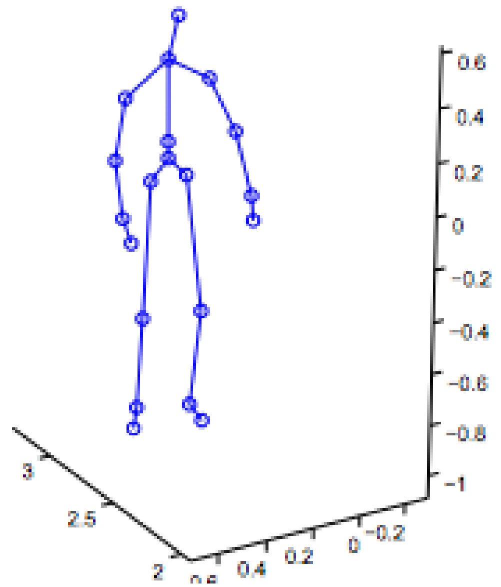
```
if x>25 then
   if x>50 then y=0 else y=1; endif
else
   if x>16 then y=0 else y=1; endif
endif
```

Trees build “orthogonal” decision boundaries.

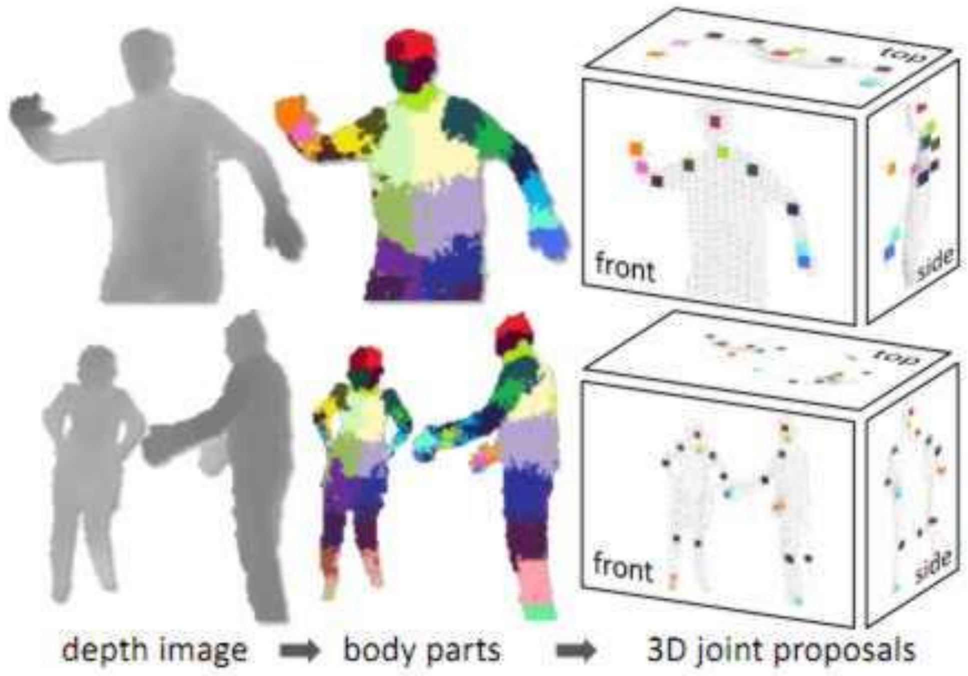
Boundary is piecewise, and at 90 degrees to feature axes.



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

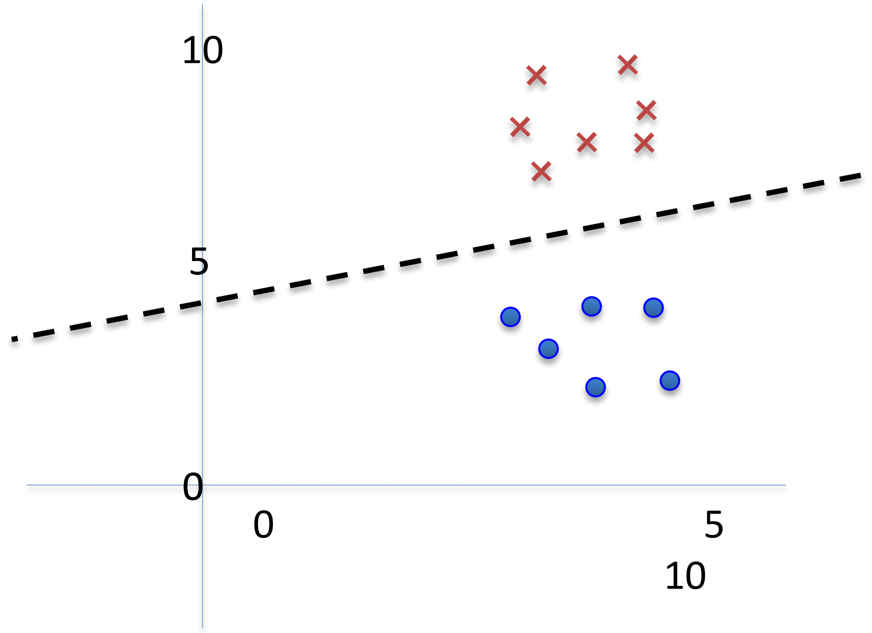


x,y,z-coordinates per joint, ~60 total
x,y,z-velocities per joint, ~60 total
joint angles (~35 total)
joint angular velocities (~35 total)

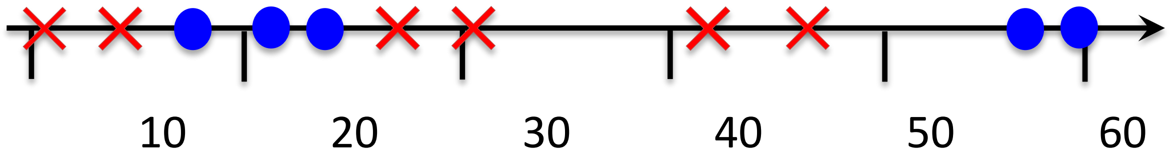


```
if x>25 then
   if x>50 then y=0 else y=1; endi
else
   if x>16 then y=0 else y=1; endi
endif
```

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

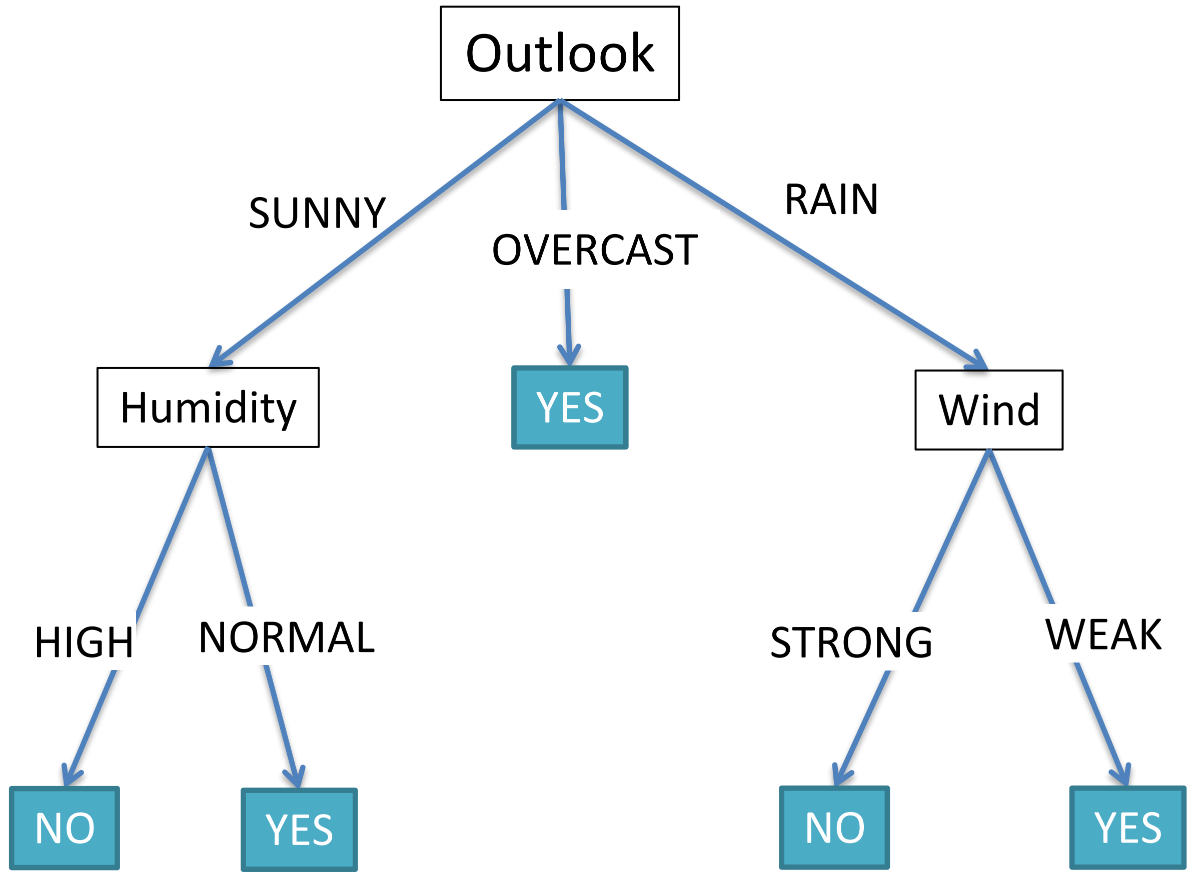


x_1,	x_2,	y (label)
98.79,	157.59,	1
93.64,	138.79,	1
42.89,	171.89,	0
...		
...		
87.91,	142.65,	1
97.92,	162.12,	1
47.63,	182.26,	0
92.72,	154.50,	1



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")

```
function buildTree( subsample, depth )

    //BASE CASE:
    if ($depth == 0$) OR (all examples have same label) then
        return most common label in the subsample
    end if

    //RECURSIVE CASE:
    for each feature do
        Try splitting the data (i.e. build a decision stump)
        Calculate the cost for this stump
    end for
    Pick feature with minimum cost

    Find left/right subsamples
    Add left branch ← buildTree( $leftSubSample, depth - 1$ )
    Add right branch ← buildTree( $rightSubSample, depth - 1$ )

    return $tree$

end function
```

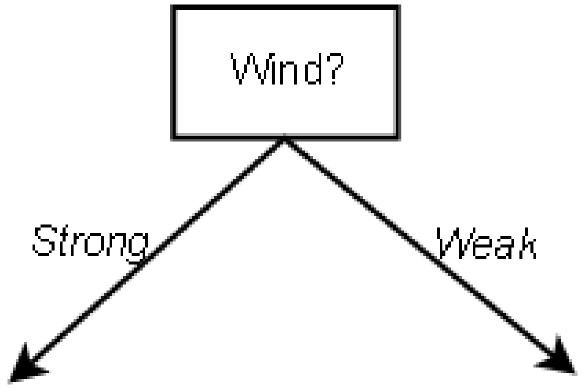
The Tennis Problem

	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

	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

3 examples say yes, 3 say no.

6 examples say yes, 2 examples say no.

Thinking in Probabilities...

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

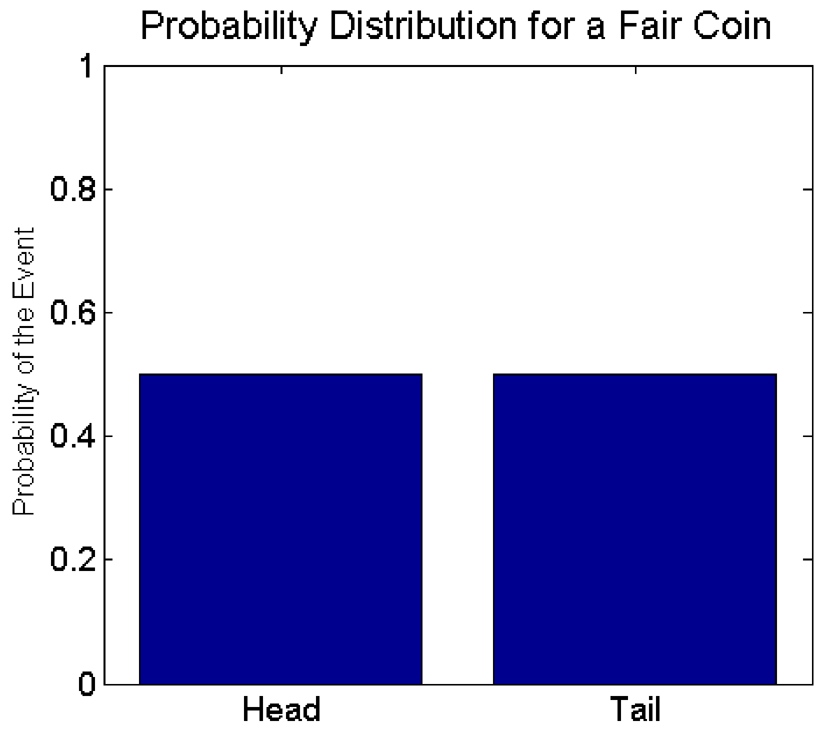
On the left branch : 3 'yes', 3 'no', ... $p('yes') = \frac{3}{6} = 0.5$

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

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

The “Information” in a feature

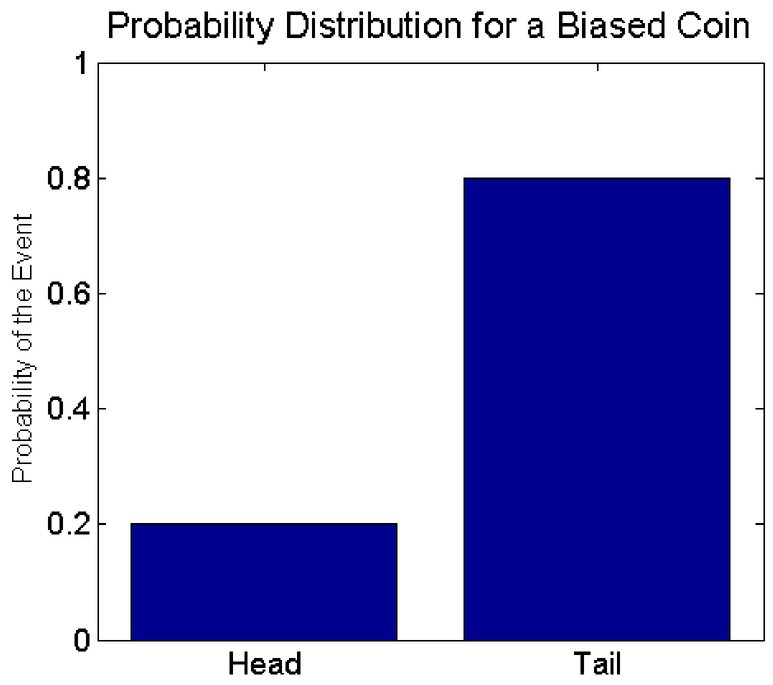
More uncertainty = less information



$H(X) = 1$

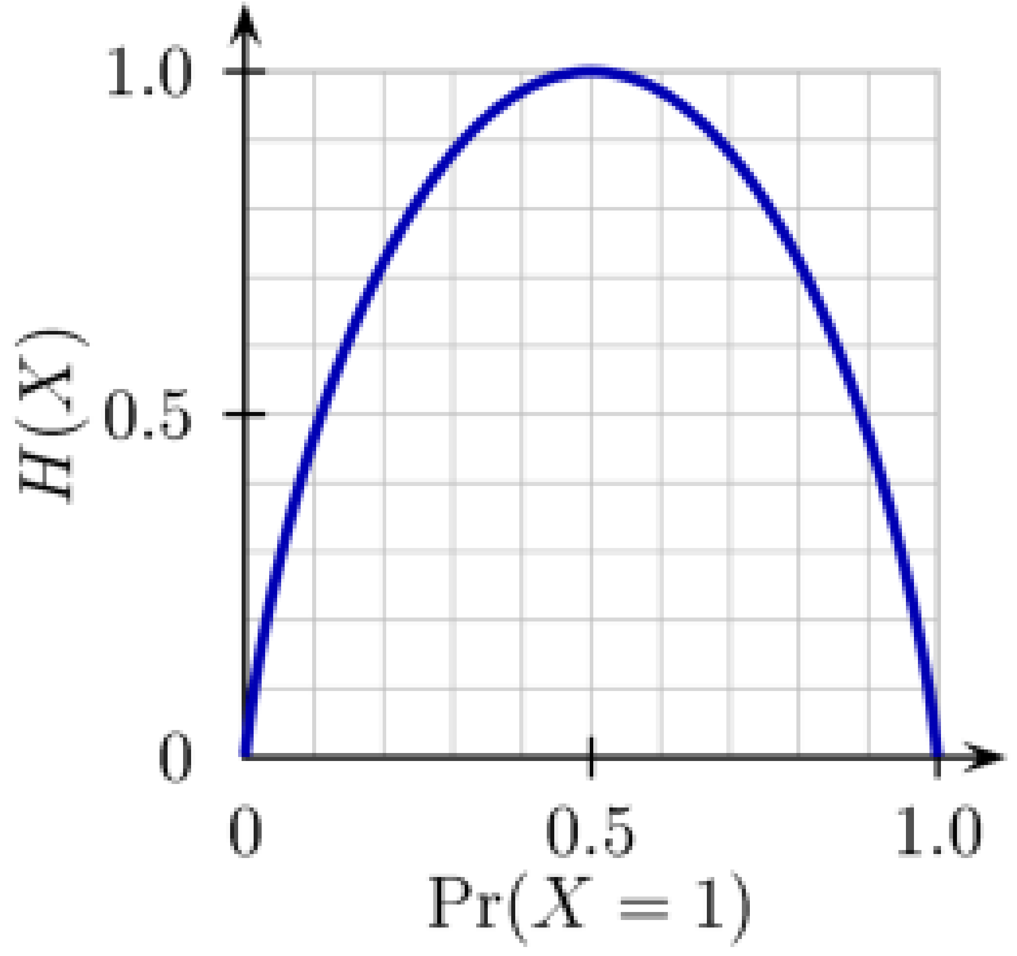
The “Information” in a feature

Less uncertainty = more information



H(X) = 0.72193

Entropy



$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

$$\begin{aligned} H(X) &= -\Big(p(head)\log p(head) + p(tail)\log p(tail)\Big) \\ &= -\Big(0.5\log 0.5 + 0.5\log 0.5\Big) \\ &= -\Big((-0.5) + (-0.5)\Big) = 1 \end{aligned}$$

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.

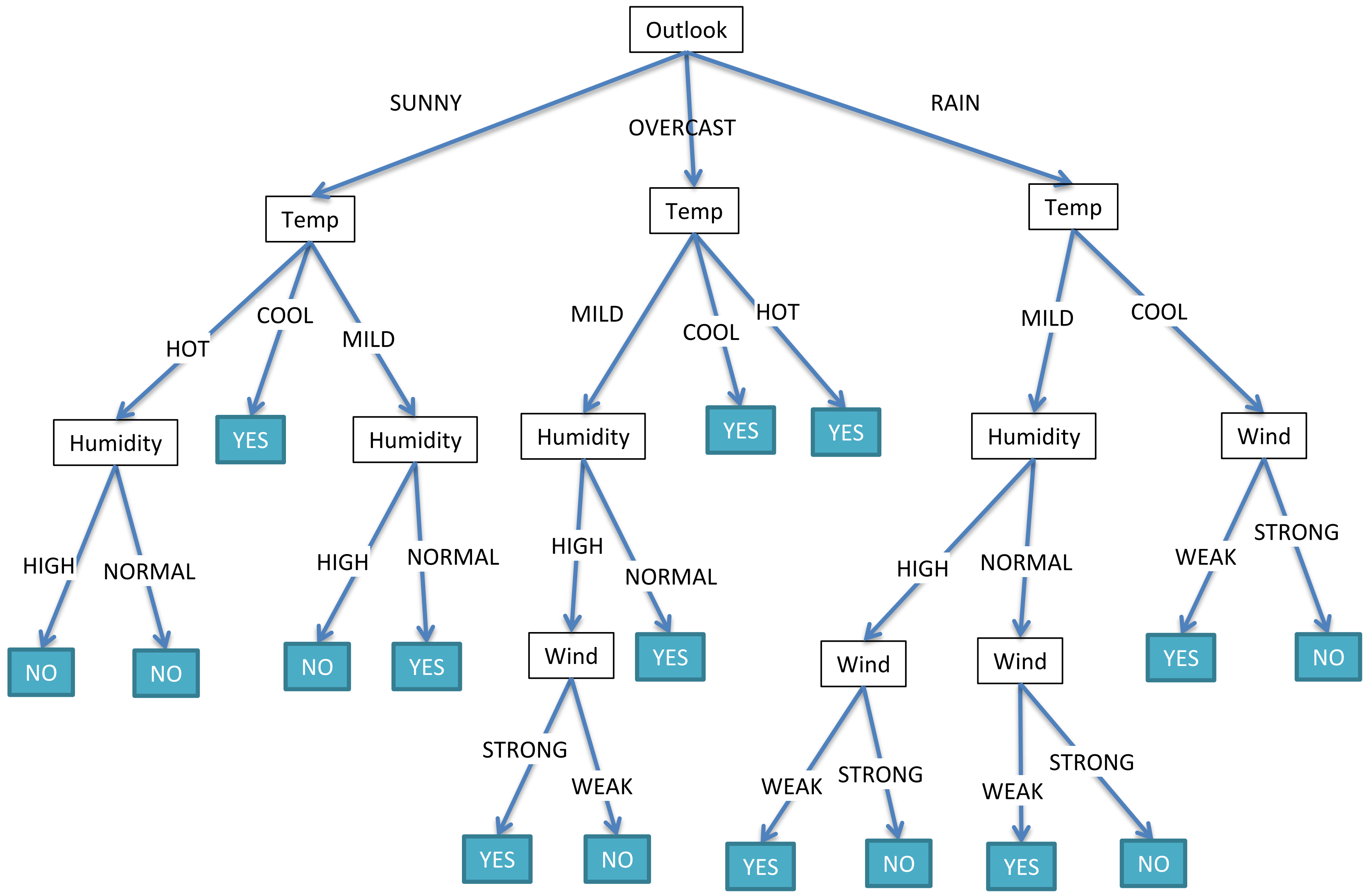
$H(T|W = strong)$ is the entropy of the data on the left branch.
$H(T|W = weak)$ is the entropy of the data on the right branch.

$H(T|W)$ 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")

```
function BUILDTREE( subsample, depth )

    //BASE CASE:
    if ($depth == 0$) OR (all examples have same label) then
        return most common label in the subsample
    end if

    //RECURSIVE CASE:
    for each feature do
        Try splitting the data (i.e. build a decision stump)
        Calculate gain for this stump
    end for
    Pick feature with ~~minimum cost~~ maximum information gain

    Find left/right subsamples
    Add left branch ← BUILDTREE( $leftSubSample$, $depth - 1$ )
    Add right branch ← BUILDTREE( $rightSubSample$, $depth - 1$ )

    return $tree$

end function
```


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

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.
- Methods developed for handling noisy training data (both class and feature noise).
- Methods developed for handling missing feature values.

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.
- There are a variety of heuristics for picking a good test, a popular one is based on information gain that originated with the ID3 system of Quinlan (1979).

Entropy

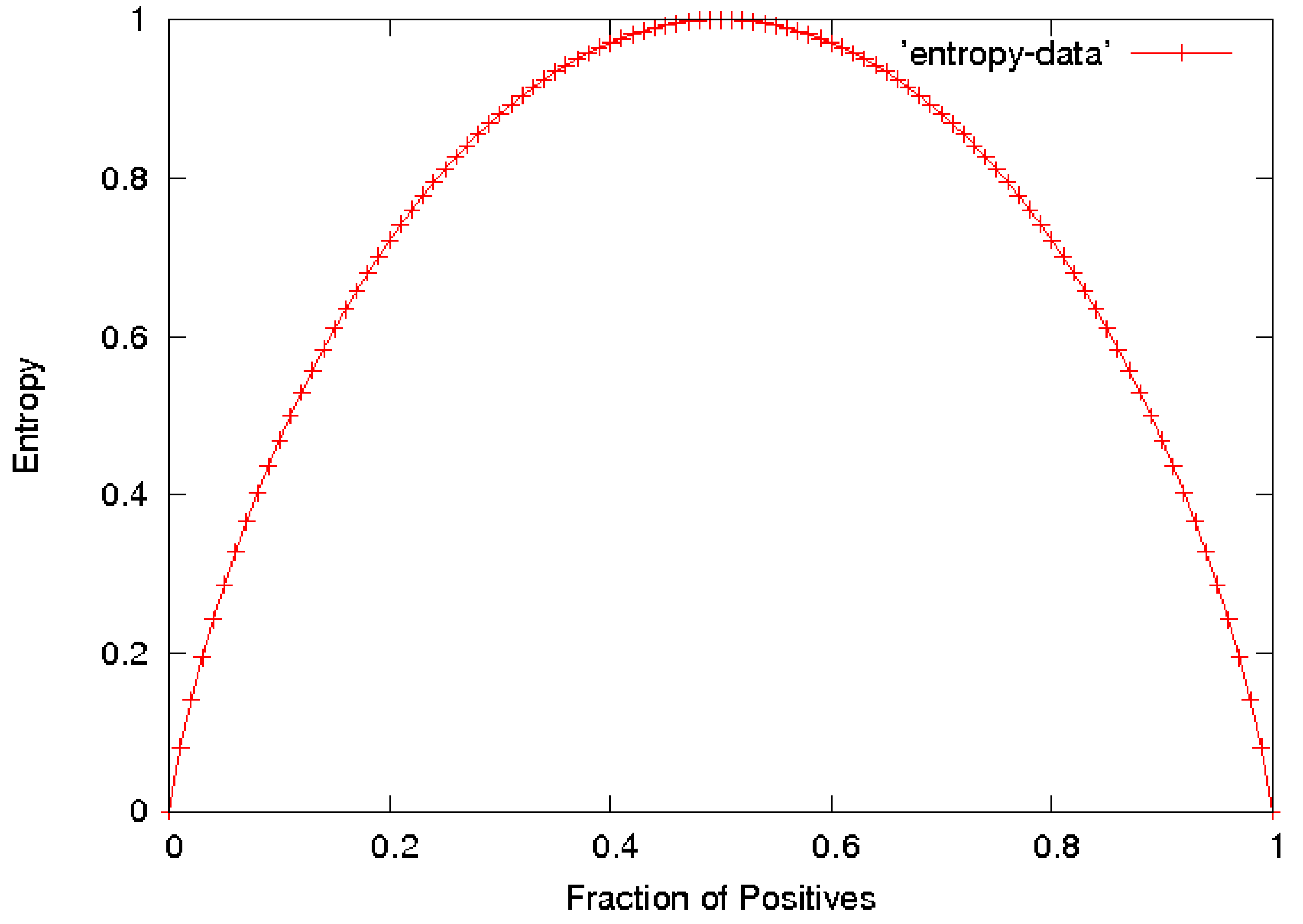
- Entropy (disorder, impurity) of a set of examples, S, relative to a binary classification is:

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

  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·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:

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

Entropy Plot for Binary Classification



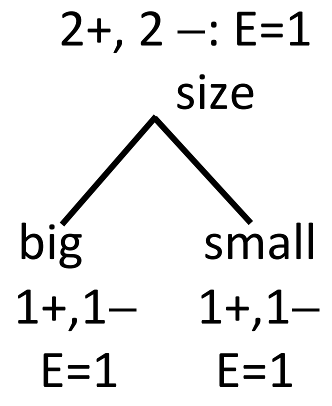
Information Gain

- The information gain of a feature *F* is the expected reduction in entropy resulting from splitting on this feature.

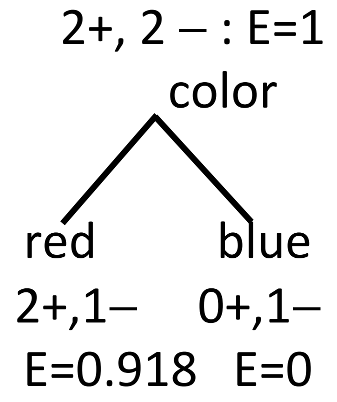
$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where S_v is the subset of *S* having value *v* for feature *F*.

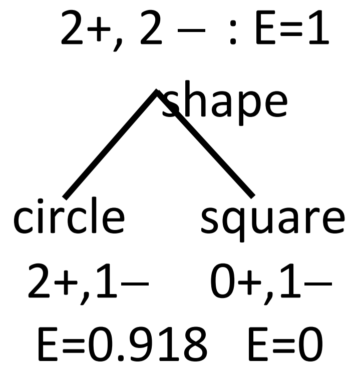
- Entropy of each resulting subset weighted by its relative size.
- Example:
  - <big, red, circle>: + <small, red, circle>: +
  - <small, red, square>: – <big, blue, circle>: –



Gain=1–(0.5·1 + 0.5·1) = 0



Gain=1–(0.75·0.918 + 0.25·0) = 0.311



Gain=1–(0.75·0.918 + 0.25·0) = 0.311

Hypothesis Space Search

- Performs ***batch learning*** that processes all training instances at once rather than ***incremental learning*** that updates a hypothesis after each example.
- Performs hill-climbing (greedy search) that may only find a locally-optimal solution. Guaranteed to find a tree consistent with any conflict-free training set (i.e. identical feature vectors always assigned the same class), but not necessarily the simplest tree.
- Finds a single discrete hypothesis, so there is no way to provide confidences or create useful queries.

Bias in Decision-Tree Induction

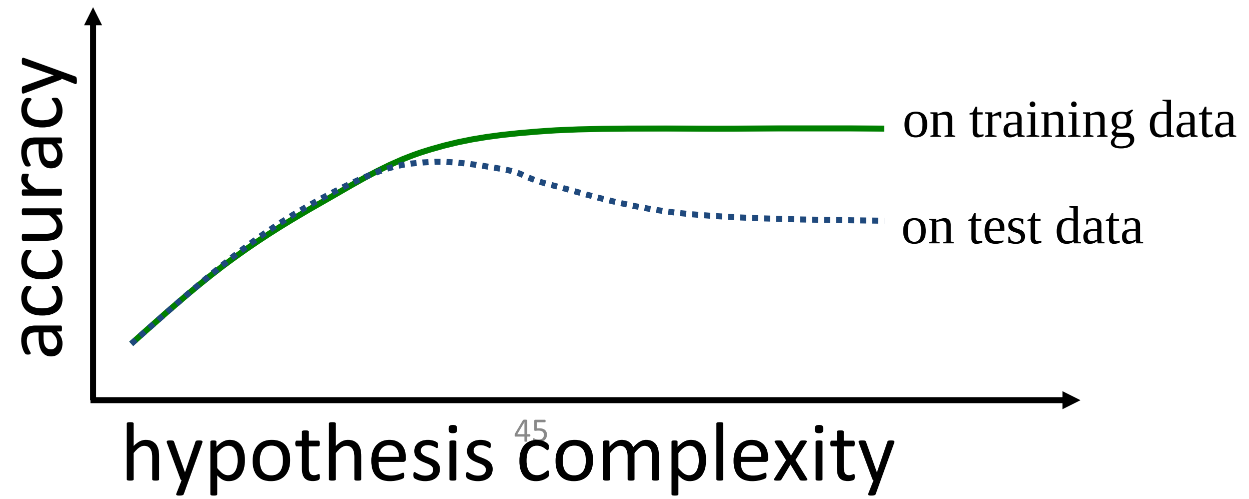
- Information-gain gives a bias for trees with minimal depth.
- Implements a search (preference) bias instead of a language (restriction) bias.

History of Decision-Tree Research

- Hunt and colleagues use exhaustive search decision-tree methods (CLS) to model human concept learning in the 1960's.
- In the late 70's, Quinlan developed ID3 with the information gain heuristic to learn expert systems from examples.
- Simulataneously, Breiman and Friedman and colleagues develop CART (Classification and Regression Trees), similar to ID3.
- In the 1980's a variety of improvements are introduced to handle noise, continuous features, missing features, and improved splitting criteria. Various expert-system development tools results.
- Quinlan's updated decision-tree package (C4.5) released in 1993.
- Weka includes Java version of C4.5 called J48.

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.

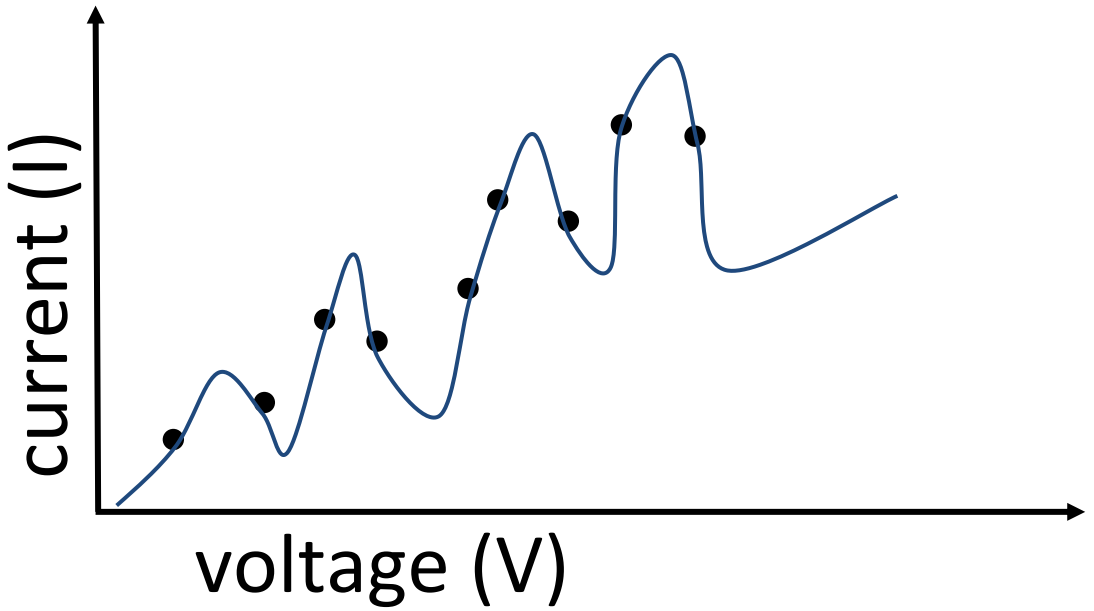


Overfitting Example

Testing Ohms Law: V = IR   (I = (1/R)V)

Experimentally measure 10 points

Fit a curve to the Resulting data.

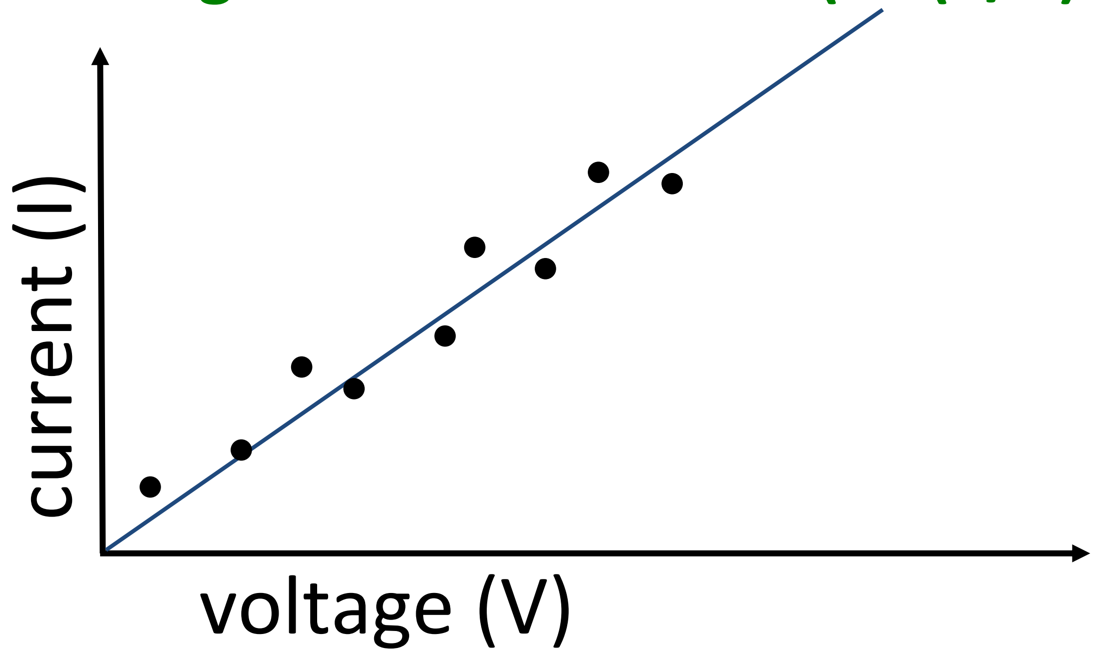


Perfect fit to training data with an 9th degree polynomial (can fit *n* points exactly with an *n*-1 degree polynomial)

Ohm was wrong, we have found a more accurate function!

Overfitting Example

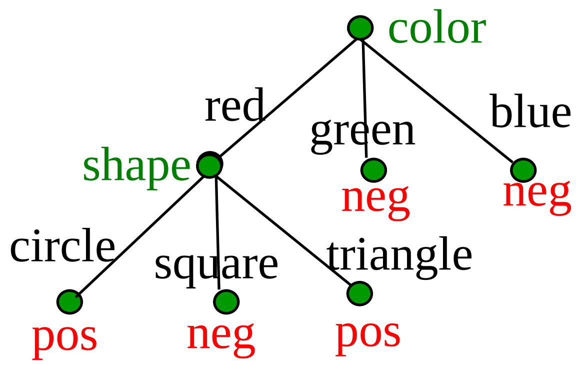
Testing Ohms Law: $V = IR$ ($I = (1/R)V$)



Better generalization with a linear function that fits training data less accurately.

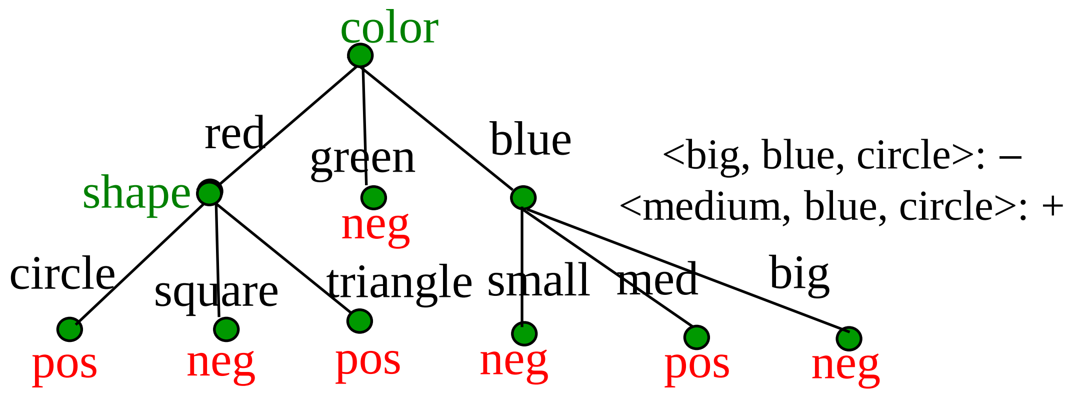
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



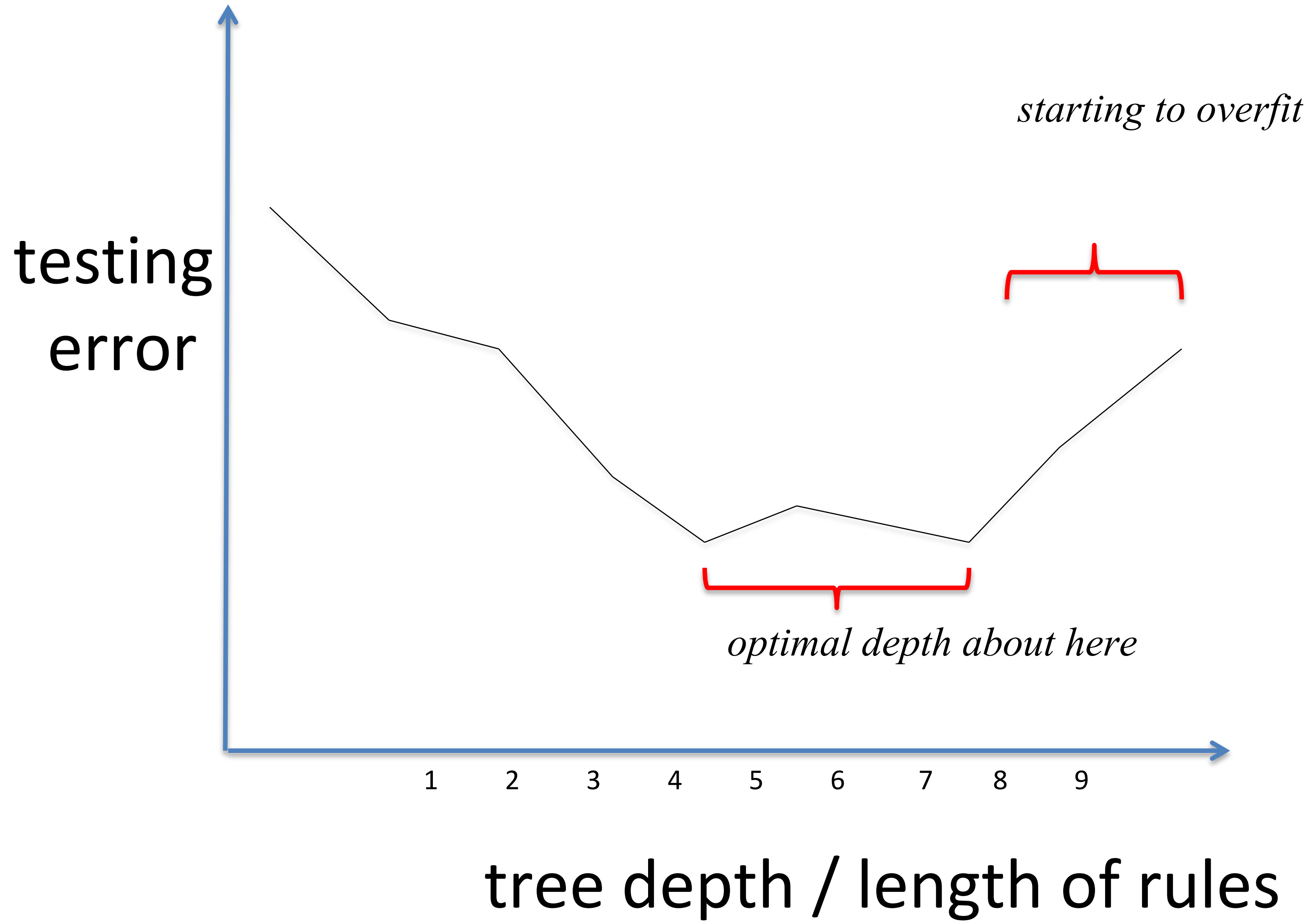
Overfitting Noise in Decision Trees

- Category or feature noise can easily cause overfitting.
  - Add noisy instance <medium, blue, circle>: pos (but really neg)



- Noise can also cause different instances of the same feature vector to have different classes. Impossible to fit this data and must label leaf with the majority class.
  - <big, red, circle>: neg (but really pos)
- Conflicting examples can also arise if the features are incomplete and inadequate to determine the class or if the target concept is non-deterministic.

Overfitting….



Overfitting Prevention (Pruning) Methods

- Two basic approaches for decision trees
  - Prepruning: Stop growing tree as some point during top-down construction when there is no longer sufficient data to make reliable decisions.
  - Postpruning: Grow the full tree, then remove subtrees that do not have sufficient evidence.
- Label leaf resulting from pruning with the majority class of the remaining data, or a class probability distribution.
- Method for determining which subtrees to prune:
  - Cross-validation: Reserve some training data as a hold-out set (***validation set***, ***tuning set***) to evaluate utility of subtrees.
  - 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.
  - Minimum description length (MDL): Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions resulting from pruning.

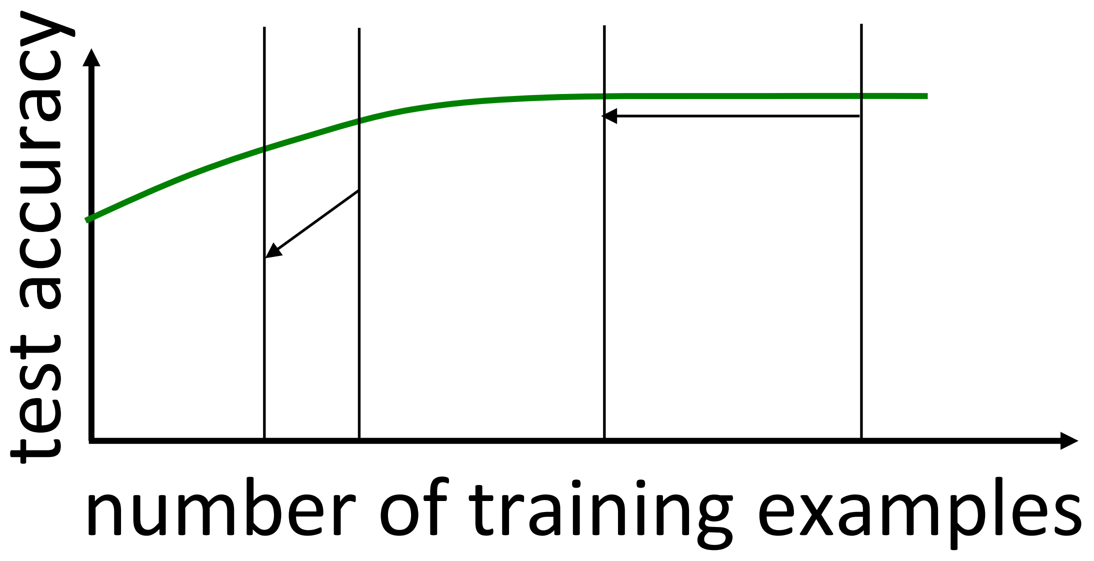
Reduced Error Pruning

- A post-pruning, cross-validation approach.

```
Partition training data in “grow” and “validation” sets.
Build a complete tree from the “grow” 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:



Cross-Validating without Losing Training Data

- If the algorithm is modified to grow trees breadth-first rather than depth-first, we can stop growing after reaching any specified tree complexity.
- First, run several trials of reduced error-pruning using different random splits of grow and validation sets.
- Record the complexity of the pruned tree learned in each trial. Let *C* be the average pruned-tree complexity.
- Grow a final tree breadth-first from all the training data but stop when the complexity reaches *C*.
- Similar cross-validation approach can be used to set arbitrary algorithm parameters in general.

Additional Decision Tree Issues

- Better splitting criteria
  - Information gain prefers features with many values.
- Continuous features
- Predicting a real-valued function (regression trees)
- Missing feature values
- Features with costs
- Misclassification costs
- Incremental learning
  - ID4
  - ID5
- Mining large databases that do not fit in main memory