Machine Learning

CS 486/686: Introduction to AI

Outline

- Introduction to Machine Learning
  - Components
  - Common Learning Tasks
  - Measuring Success
  - Bias
  - Learning as Search
- Supervised Learning
  - Basic Models
  - Decision Trees

Introduction

Learning is the ability to improve one’s behaviour based on experience

- The range of behaviours is expanded
  - The agent can do more

- The accuracy on tasks is improved
  - The agent can do things better

- The speed is improved
  - The agent can do things faster

What is Machine Learning

Definition (T Mitchell):

> A computer program is said to **learn** from **experience** E with respect to some class of **tasks** T and **performance measures** P, if its performance at tasks in T, as measured by P, improves with experience E.

Examples

- Handwriting recognition
  - Tasks: Recognize and classify handwritten letters and digits
  - Experience: Database of pre-classified letters and digits
  - Performance measure: Percent of letters/digits correctly classified
- Game playing problem
  - Tasks: Playing the game
  - Experience: Playing practice games against itself (self-play)
  - Performance measure: Percentage of games won against an opponent

Common Learning Tasks

Supervised Classification

- Given a set of pre-classified training examples, classify a new instance

Unsupervised Learning

- Find natural classes for examples

Reinforcement Learning

- Determine what to do based on rewards and punishments

Transfer Learning

- Learning from an expert

Active Learning

- Actively seek to learn

Feedback

Learning tasks can be defined by the feedback the learner receives

Supervised Learning:

- What has to be learned is specified for each example

Unsupervised Learning:

- No classifications are given. Learner has to discover categories and patterns in the data

Reinforcement Learning:

- Feedback occurs after taking a sequence of actions. Credit assignment problem

Representations

- The representation of what we are learning is crucial
  - It determines how the learning algorithm will work

- The richer the representation the more useful it is for subsequent problem solving

- The richer the representation, the more difficult it is to learn

Measuring Performance

- We will always have some sort of performance measure so as to judge the learning
- The measure of performance is not on how well the agent does on the training examples, but on how well it performs with new examples
- Example
  - Agent P claims the negative examples it has seen are the only negative examples. All other instances are positive.
  - Agent N claims the positive examples it has seen as the only positive one. All other instances are negative.
  - What will happen?

Bias

- A tendency to prefer one hypothesis over another is a **bias**
- Saying a hypothesis is better than N or P’s isn’t something that is obtained from the data
- To make any inductive process make predictions on unseen data, an agent must have a bias
- What is a good bias is an empirical question
  - Often prefer simpler hypothesis over complex (Ockham’s Razor)

Learning as Search

- Given a representation, data, and a bias, we now have a search problem
- Learning is search though the space of possible representations looking for the representation that best fits the data, given the bias
- Search spaces are usually too large for systematic search (instead use gradient descent, stochastic simulation,….)
- A learning problem is made up of a search space, an evaluation function, and a search method

Some Notes About Data

Data is not perfect:

- The features given are inadequate to predict classification
- There are examples with missing features
- Data is just incorrect (e.g. labeled incorrectly)
- It is incomplete
- …

Overfitting

- Finding patterns in the data where there is no actual pattern

Supervised Learning

Given

- A set of input features $X_1,\dots,X_n$
- A set of target features f(**X**) or $Y_1,\dots,Y_k$
- A set of training examples where the values for the input features and target features are given for each example
- A set of test examples, where only the values for the input features are given

Predict the values for the target features for the test examples

- Classification: Yi are discrete
- Regression: Yi are continuous

- **Very Important: keep training and test sets separate!!!**

Supervised Learning

Sky	**AirTemp**	**Humidity**	**Wind**	**Water**	**Forecast**	**EnjoySport**
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{Xi} \qquad \overset{\uparrow}{f(X)=Y}$$

Goal: Return a function h that approximates f(x)

h is the **hypothesis**

Supervised Learning

Let H be the set of all possible hypothesis given our chosen representation

Learning is search though H to find a “good” h

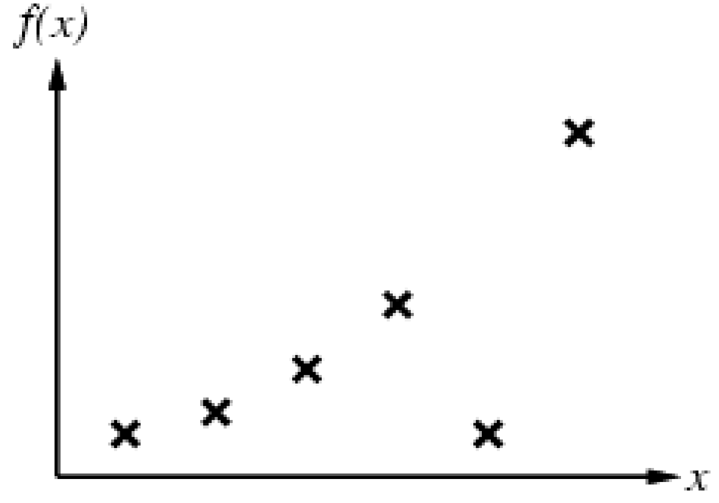
What does “good” mean?

- Usually that it **generalizes** well (i.e. performs well on unseen examples)

Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well ***over a sufficiently large set of training examples*** will also approximate the target function well over any unobserved examples

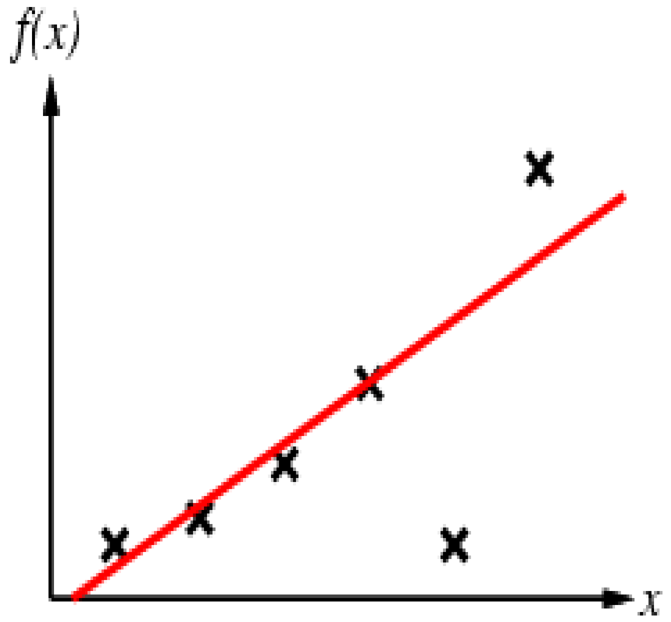
Inductive Learning



Construct/adjust h to agree with f on training set

- h is **consistent** if it agrees with f on all examples
- e.g. curve fitting

Inductive Learning

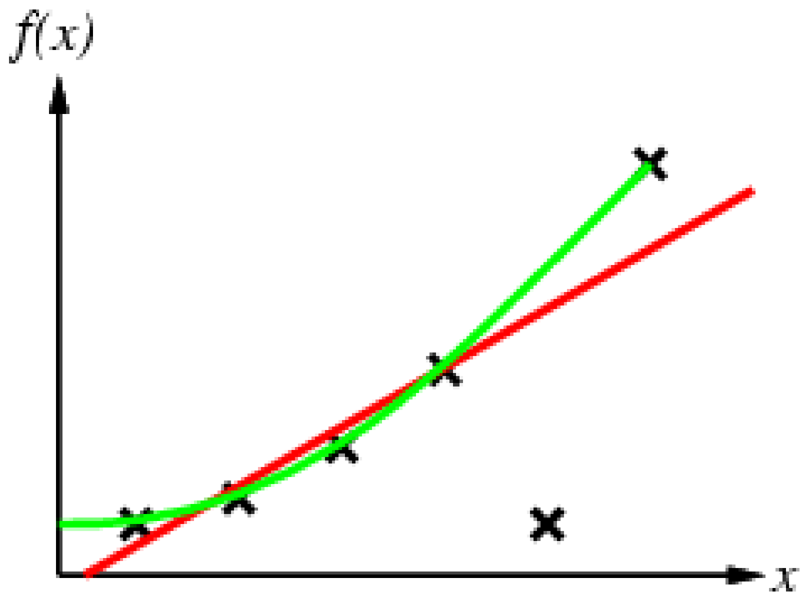


Construct/adjust h to agree with f on training set

h is **consistent** if it agrees with f on all examples

e.g. curve fitting

Inductive Learning

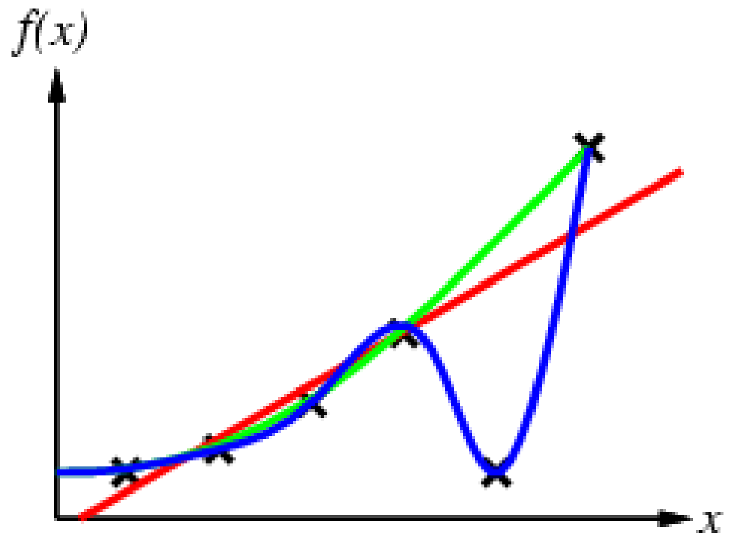


Construct/adjust h to agree with f on training set

h is **consistent** if it agrees with f on all examples

e.g. curve fitting

Inductive Learning

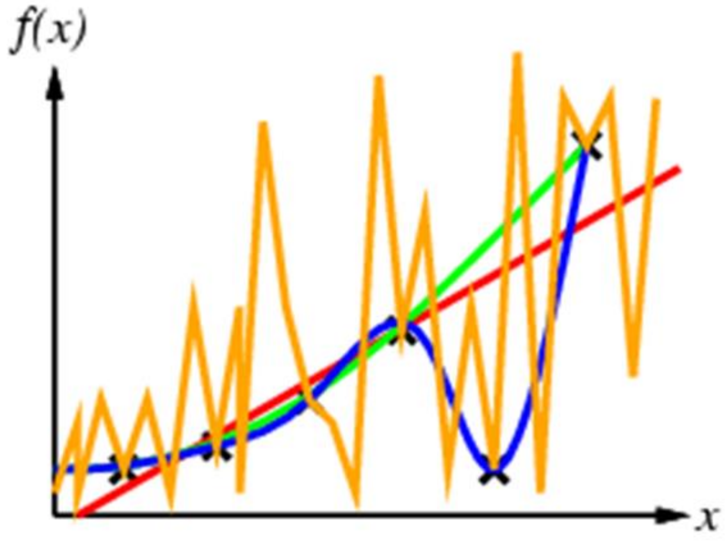


Construct/adjust h to agree with f on training set

h is **consistent** if it agrees with f on all examples

e.g. curve fitting

Inductive Learning



Bias (Ockham’s Razor): Prefer the simplest hypothesis consistent with the data

Evaluating Performance of a Supervised Learning Algorithm

- Suppose Y is a feature and e is an example
    - Y(e) is the true value of feature Y for example e
    - Y*(e) is the predicted value of feature Y for example e
- The error of the prediction is a measure of how close Y*(e) is to Y(e)
- There are many ways of measuring error
    - Absolute error, sum-of-squares error, worst-case error, cost-based error, likelihood, entropy,…

Receiver Operating Curve (ROC)

- Not all errors are equal!
  - Predict a patient has a disease when they do not
  - Predict a patient does not have a disease when they do

Predicted

Actual

	T	F
T	True Positive (TP)	False Negative (FN)
F	False Positive (FP)	True Negative (TN)

Receiver Operating Curve (ROC)

Predicted

Actual

	T	F
T	True Positive (TP)	False Negative (FN)
F	False Positive (FP)	True Negative (TN)

- Recall=Sensitivity = TP/(TP+FN)
- Specificity = TN/(TN+FP)
- Precision = TP/(TP+FP)
- F-measure = 2*Precision*Recall/(Precision + Recall)

Supervised Learning

Many supervised learning algorithms can be seen as being derived from

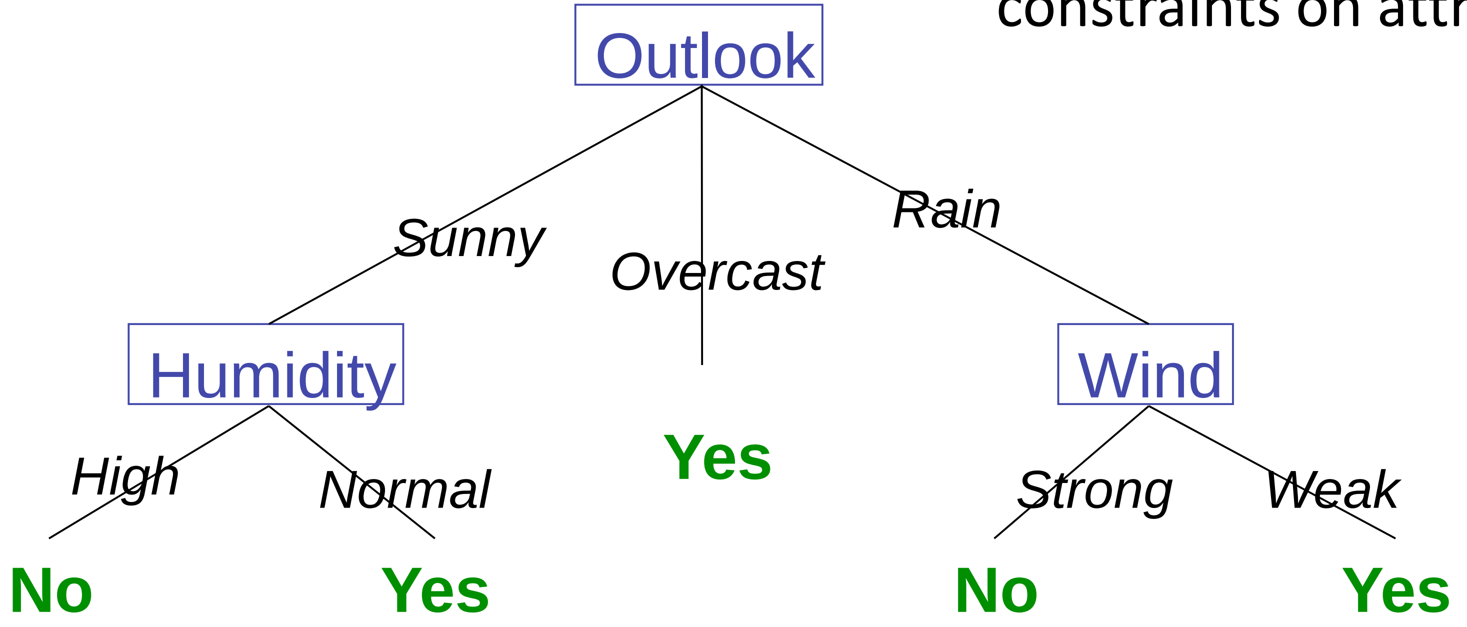
- **Decision Trees**
- Linear Classifiers
- Bayesian Classifiers (later in the semester)

Decision Trees

- Decision trees classify instances by sorting them down the tree from root to leaf
  - Nodes correspond with a test of some attribute
  - Each branch corresponds to some value an attribute can take
- Classification algorithm
  - Start at root, test attribute specified by root
  - Move down the branch corresponding to value of the attribute
  - Continue until you reach leaf (classification)

Decision Trees

Note: Decision trees represent disjunctions of conjunctions of constraints on attribute values



An instance

<Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong>

Classification: No

Decision Tree Representation

Decision trees are fully expressive within the class of propositional languages

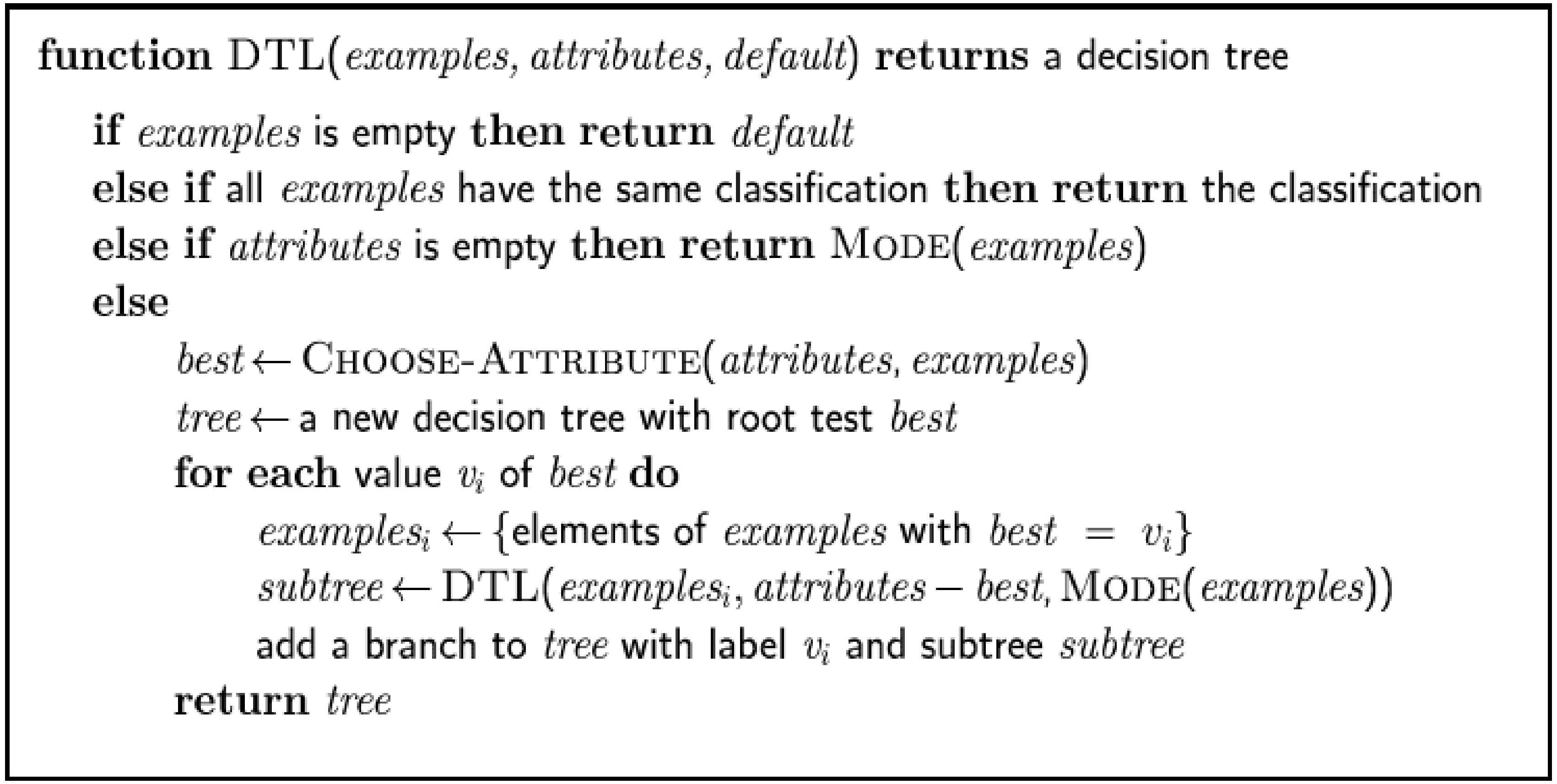
Any Boolean function can be written as a decision tree

No representation is efficient for all functions

Inducing a Decision Tree

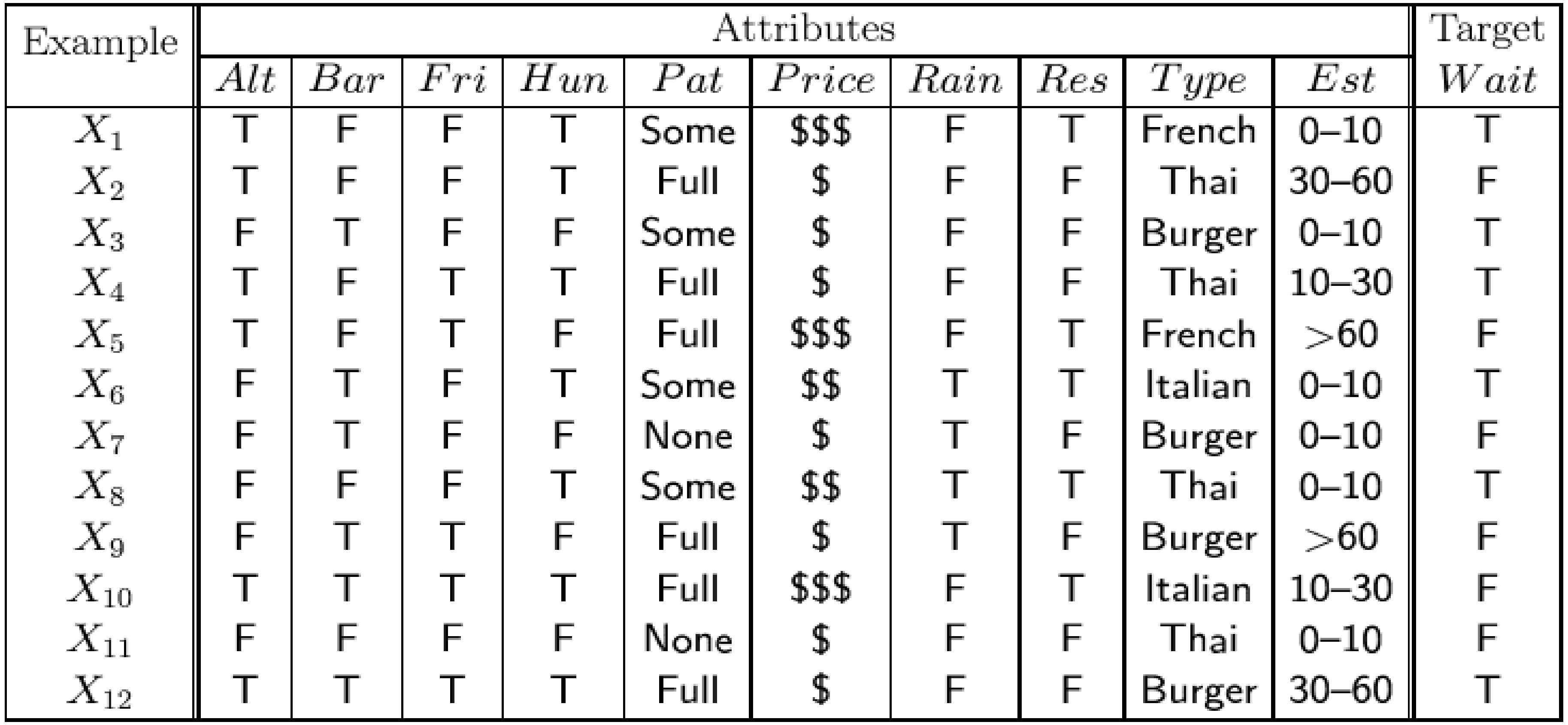
Aim: Find a small tree consistent with the training examples

Idea: (recursively) choose "most significant" attribute as root of (sub)tree



```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value $v_i$ of best do
            $examples_i$ ← {elements of examples with best = $v_i$}
            subtree ← DTL($examples_i$, attributes − best, MODE(examples))
            add a branch to tree with label $v_i$ and subtree subtree
        return tree
```

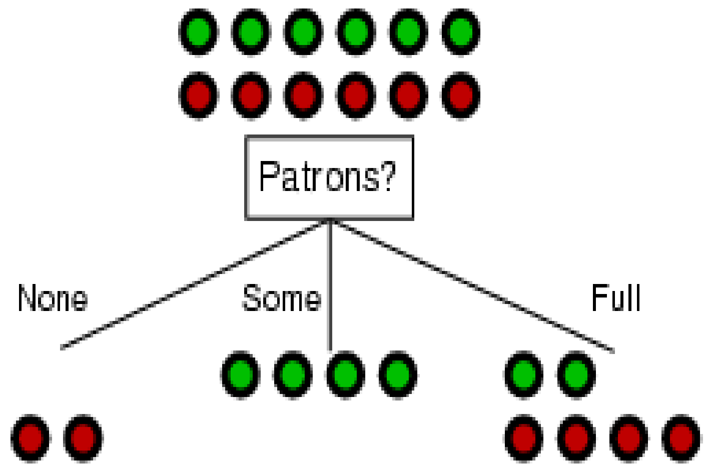
Example: Restaurant

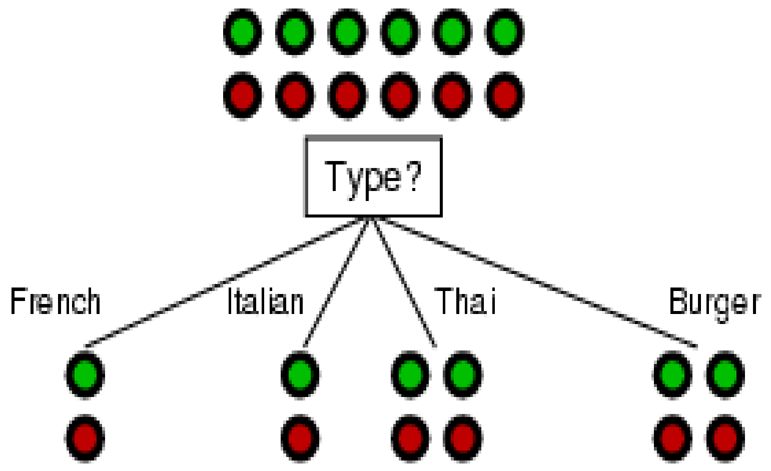


Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Choosing an Attribute

- There are different ways of selecting attributes, but generally a "good attribute" splits the training examples appropriately





Using Information Theory

Information content (Entropy):

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} (-P(V_i) \log_2 P(V_i))$$

For a training set containing p positive examples and n negative examples

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Information Gain

Chosen attribute A divides the training set E into subsets $E_1,\ldots,E_v$ according to their values for A, where A has v distinct values

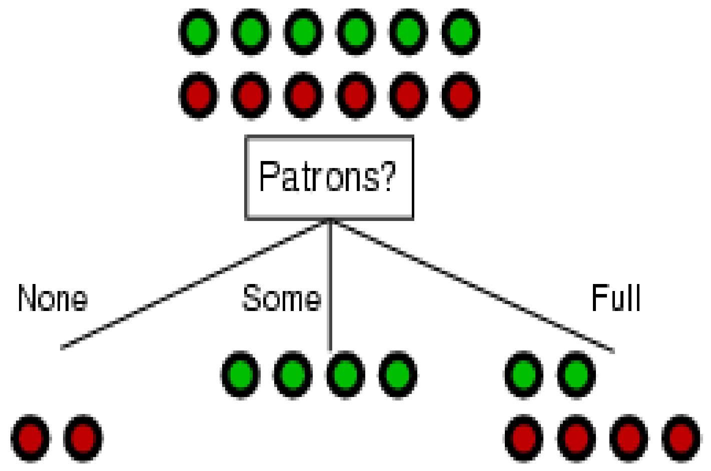
$$remainder(A) = \sum_{i=1}^{v} \frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

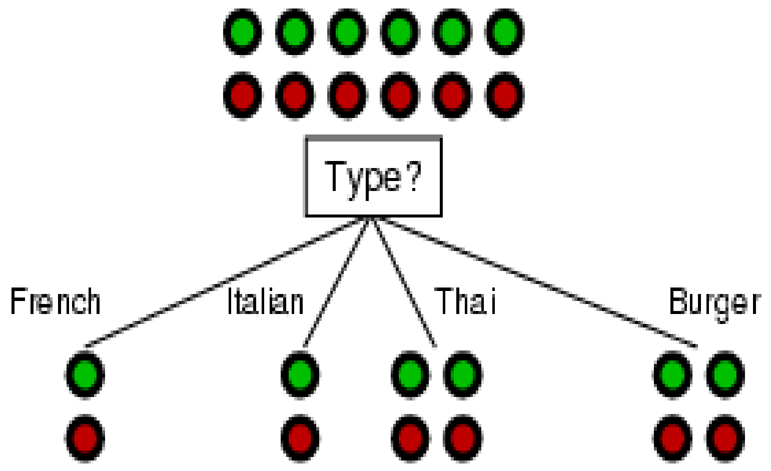
Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(A)$$

Choosing an Attribute

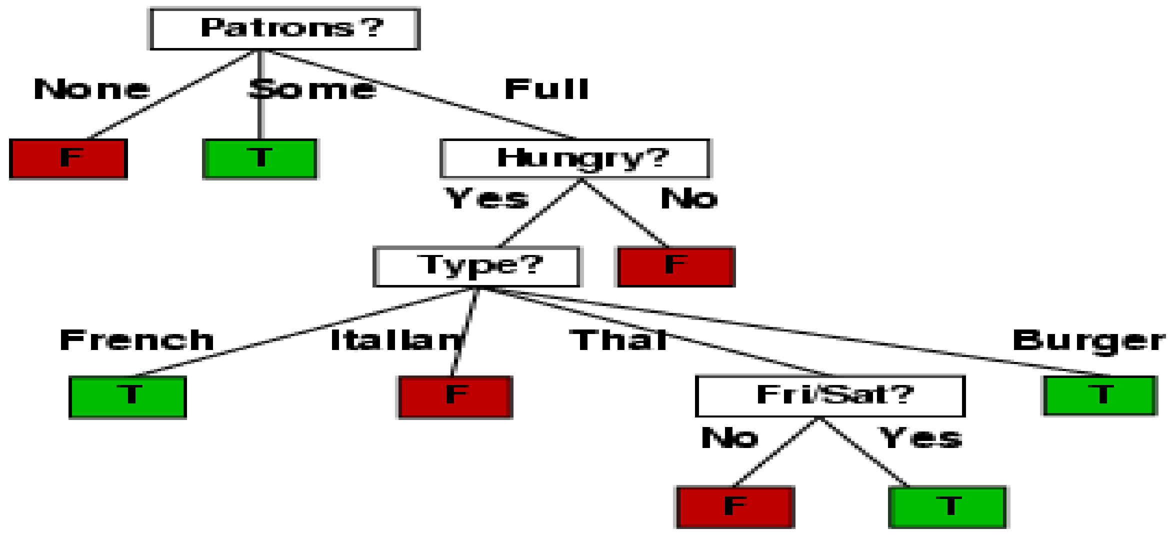
- There are different ways of selecting attributes, but generally a “good attribute” splits the training examples appropriately





Decision Tree Example

Decision tree learned from 12 examples



Substantially simpler than “true” tree

A more complex hypothesis isn’t justified by the small amount of data

Assessing Performance

A learning algorithm is **good** if it produces a hypothesis that does a good job of predicting classifications of unseen examples

There are theoretical guarantees (learning theory)

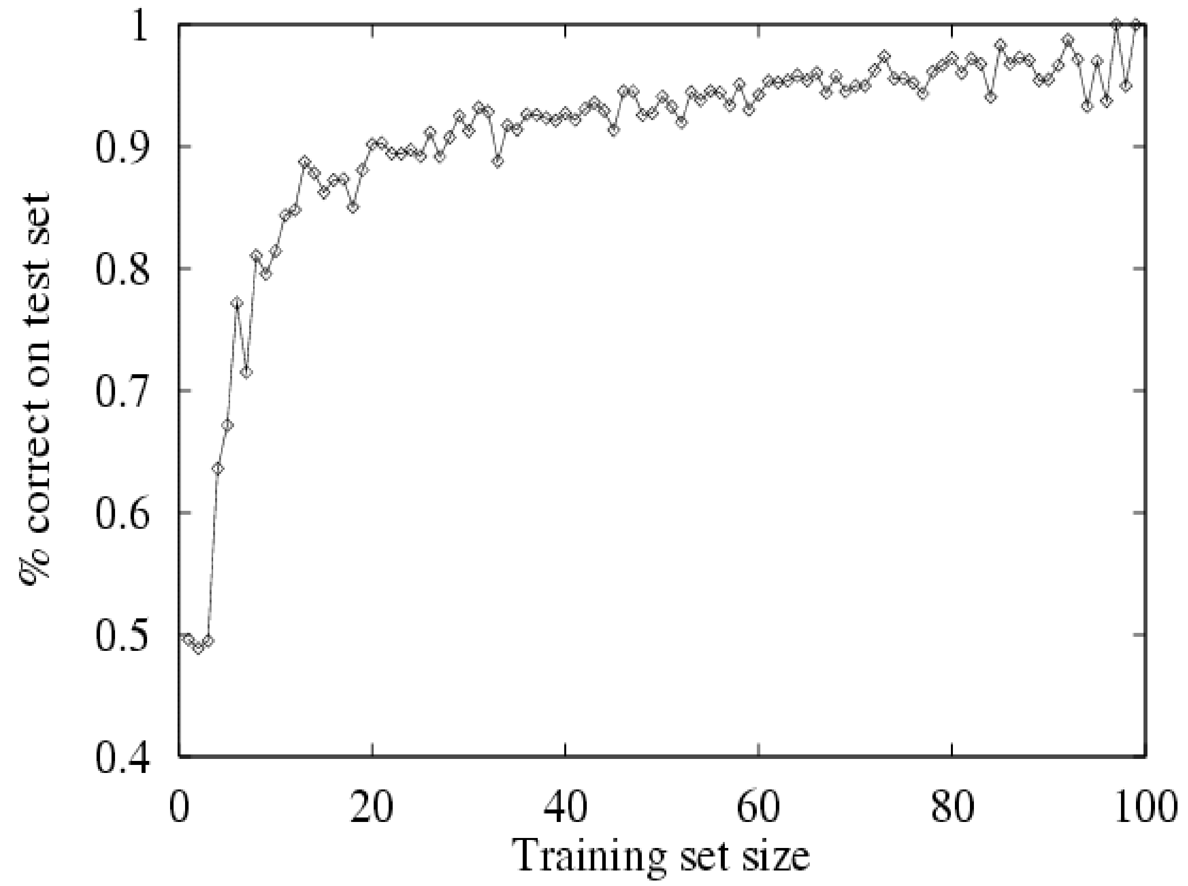
Can also test this

Assessing Performance

Test set

- Collect a large set of examples
- Divide them into 2 disjoint sets (training set and test set)
- Apply learning algorithm to training set to get h
- Measure percentage of examples in the test set that are correctly classified by h

Learning Curve



As the training set grows, accuracy increases

No Peeking at the Test Set

A learning algorithm should not be allowed to see the test set data before the hypothesis is tested on it

No Peeking!!

Every time you want to compare performance of a hypothesis on a test set ***you should use a new test set***!

Overfitting

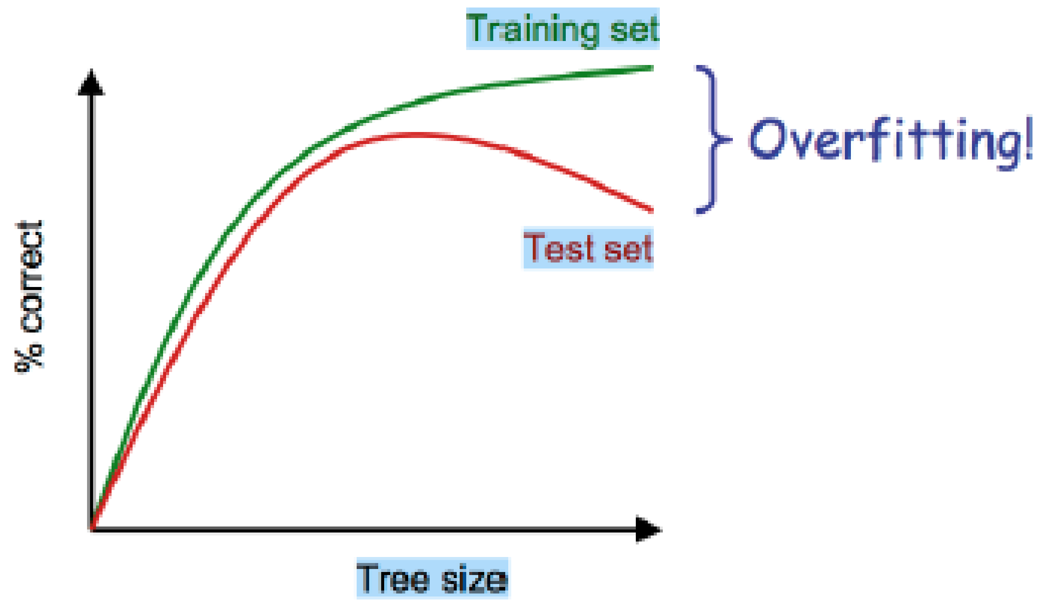
Why might a consistent hypothesis have a high error rate on a test set?

Given a hypothesis space H, a hypothesis h in H is said to ***overfit*** *the training data if there exists some alternative hypothesis h’ in H such that h has smaller error than h’ on the training examples, but h’ has smaller error than h over the entire distribution of instances*

h in H overfits if there exists h’ in H such that $\text{error}_{Tr}(h)<\text{error}_{Tr}(h')$ but $\text{error}_{Te}(h')<\text{error}_{Te}(h)$

Overfitting

- Overfitting has been found to decrease accuracy of decision trees by 10-25%



Overfitting

Test errors caused by

- **Bias**: the error due to the algorithm finding an imperfect model
  - Representation bias: model is too simple
  - Search bias: not enough search
- **Variance**: error due to lack of data
- **Noise**: error due to data depending on features not modeled or because the process generating data was inherently stochastic
- **Bias-Variance Trade-Off**:
  - Complicated model, not enough data (low bias, high variance)
  - Simple model, lots of data (high bias, low variance)

Avoiding Overfitting

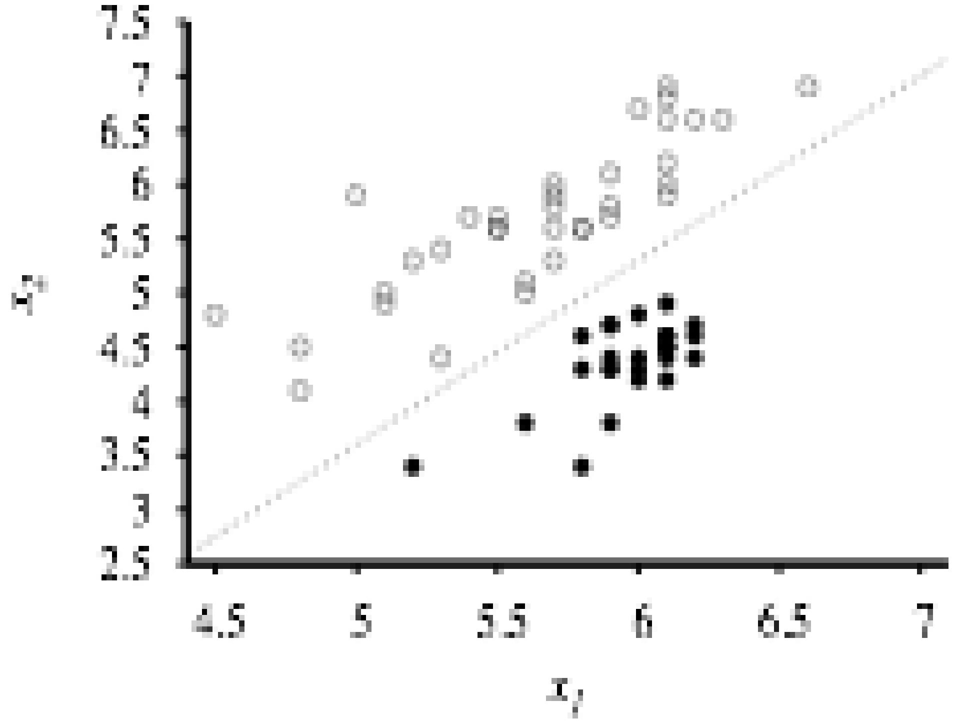
- Regularization: Prefer small decision trees over large ones so add a complexity penalty to the stopping criteria (stop early)
- Pseudocounts: Add some data based on prior knowledge
- Cross validation

Cross Validation

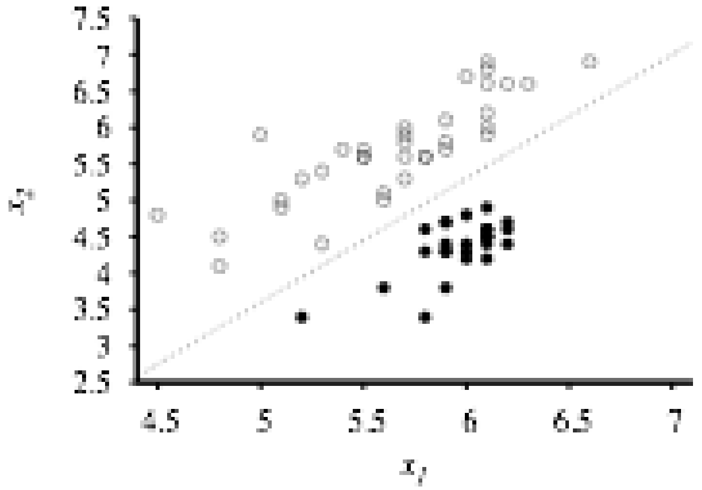
- Split your **training set** into a training and a validation set
- Use the validation set as a "***pretend***" test set
- Optimize the hypothesis/classifier/etc to perform well on the validation set, not the training set
- Can do this multiple times with different validation sets
  - K-fold validation, leave-one-out validation
- When measuring actual performance, report performance on **test set**

Classification with Linear Thresholds

Imagine you have data of the form (**x**,f(**x**)) where x in R^n and f(**x**) in {0,1}



Linear Threshold Classifiers



$$\mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2$$

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Linear Threshold Classifiers

Learning Problem: Find the weights **w** such that h$_{\mathbf{w}}$ is a good classifier

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x_1 + w_2 x_2$$

Learning Problem: Find the weights **w** to minimize the loss function.

$$\text{Loss}(h_{\mathbf{w}}) = L_2(y, h_{\mathbf{w}}(\mathbf{x})) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2$$

Gradient Descent

```
**w** ← any point in parameter space
Loop until convergence do
    For each $w_i$ in **w** do
        $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \mathrm{Loss}(\mathbf{w})$
```

α is the step size or learning rate. It can be a fixed constant or can decrease over time as the learning progresses

Update Rule (Perceptron Update Rule)

When updating weights

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i$$

Intuition:

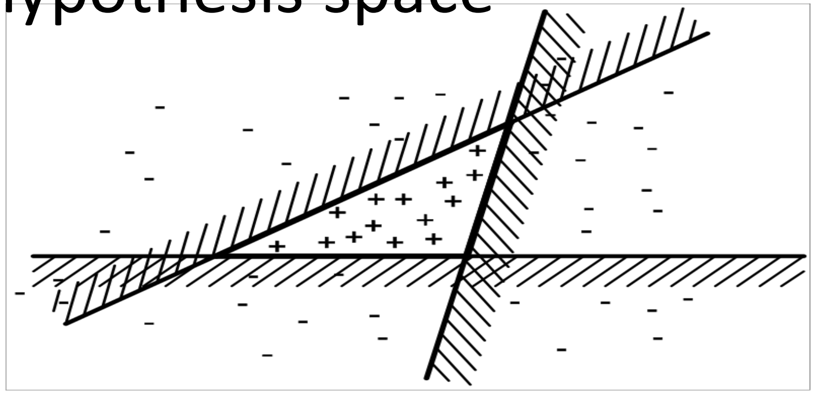
Ensembles

- So far we have discussed learning as though it followed a single general approach
    - Choose a single hypothesis from the hypothesis spae
    - Use it to make predictions/classifications

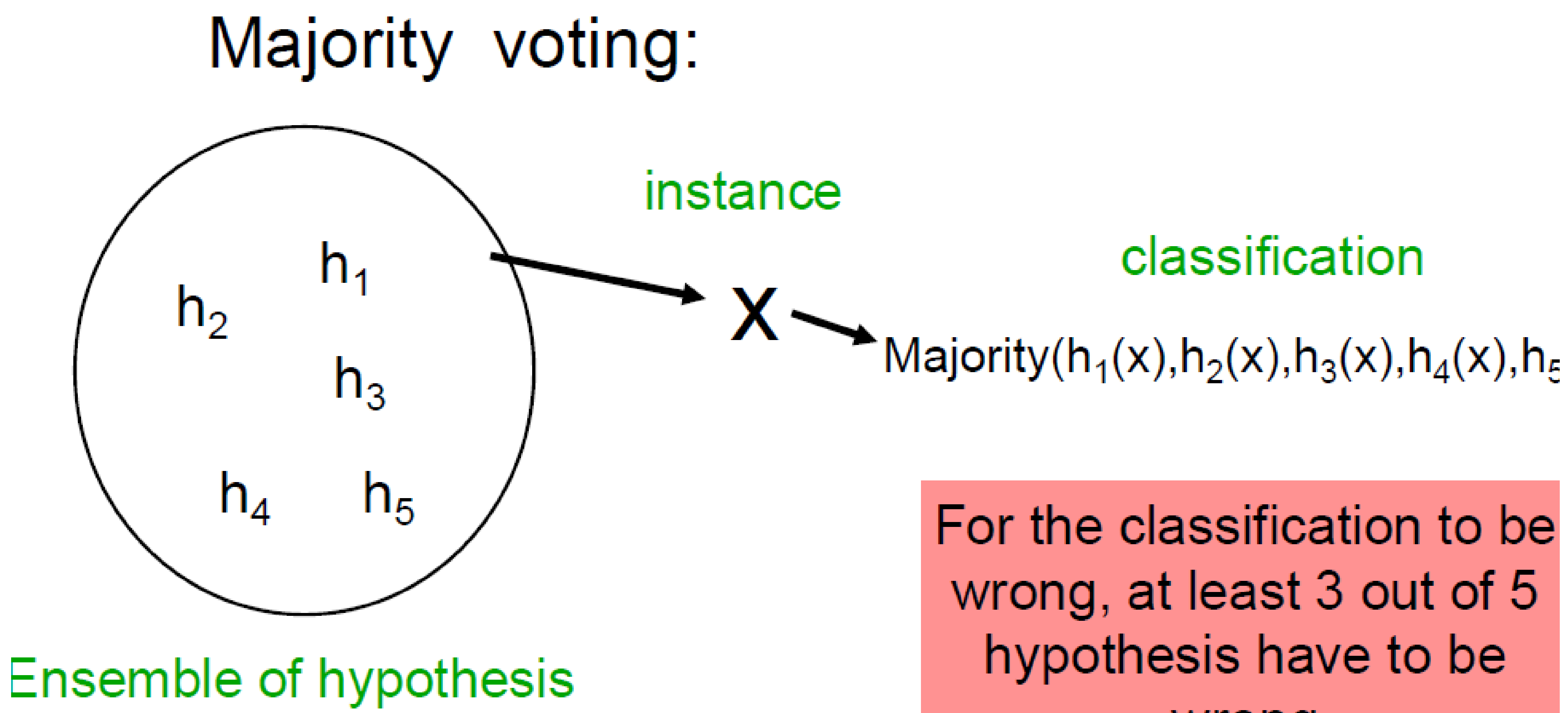
- What happens if we want to use many hypothesis and combine their predictions?

Ensembles

- Analogies
  - Elections, committees
- Intuition
  - Individuals may make mistakes, but the majority may be less likely to make a mistake
  - Individuals have partial information but committees can pool their expertise
- Using ensembles can also enlarge the hypothesis space



Ensembles: Bagging



For the classification to be wrong, at least 3 out of 5 hypothesis have to be wrong

Ensembles: Bagging

- Assumptions:
  - Each h_i makes an error with probability p
  - Hypothesis are independent

- Majority voting of n hypothesis
  - Probability k made an error?
  - Probability that the majority made an error?

Bagging Example: Random Forests

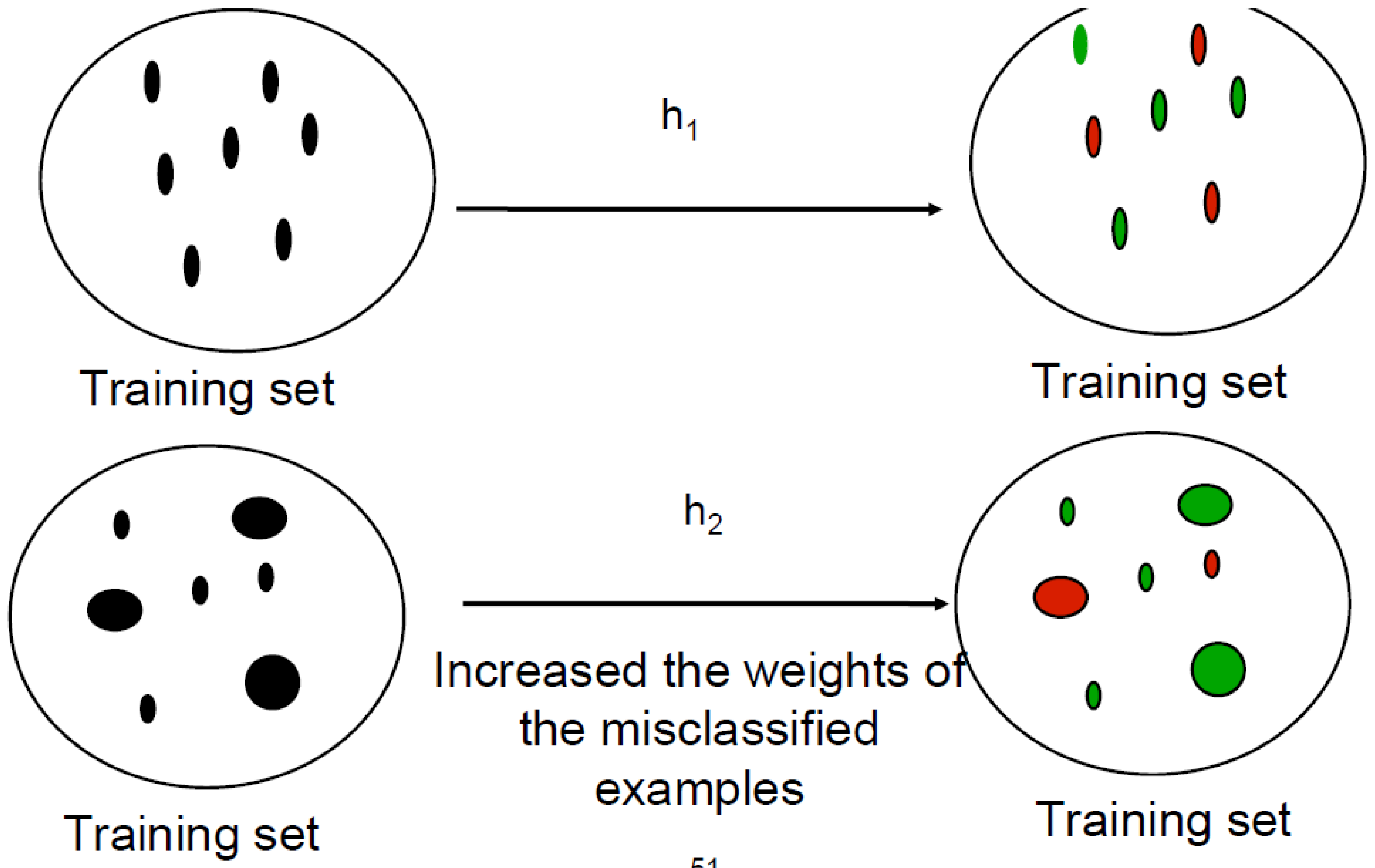
- Do K times
    - Randomly sample (with replacement) subsets of your training data
    - Randomly sample subsets of features (feature bagging)
    - Learn a decision tree using the subset of features

Classify using a majority vote of the k trees in your forest.

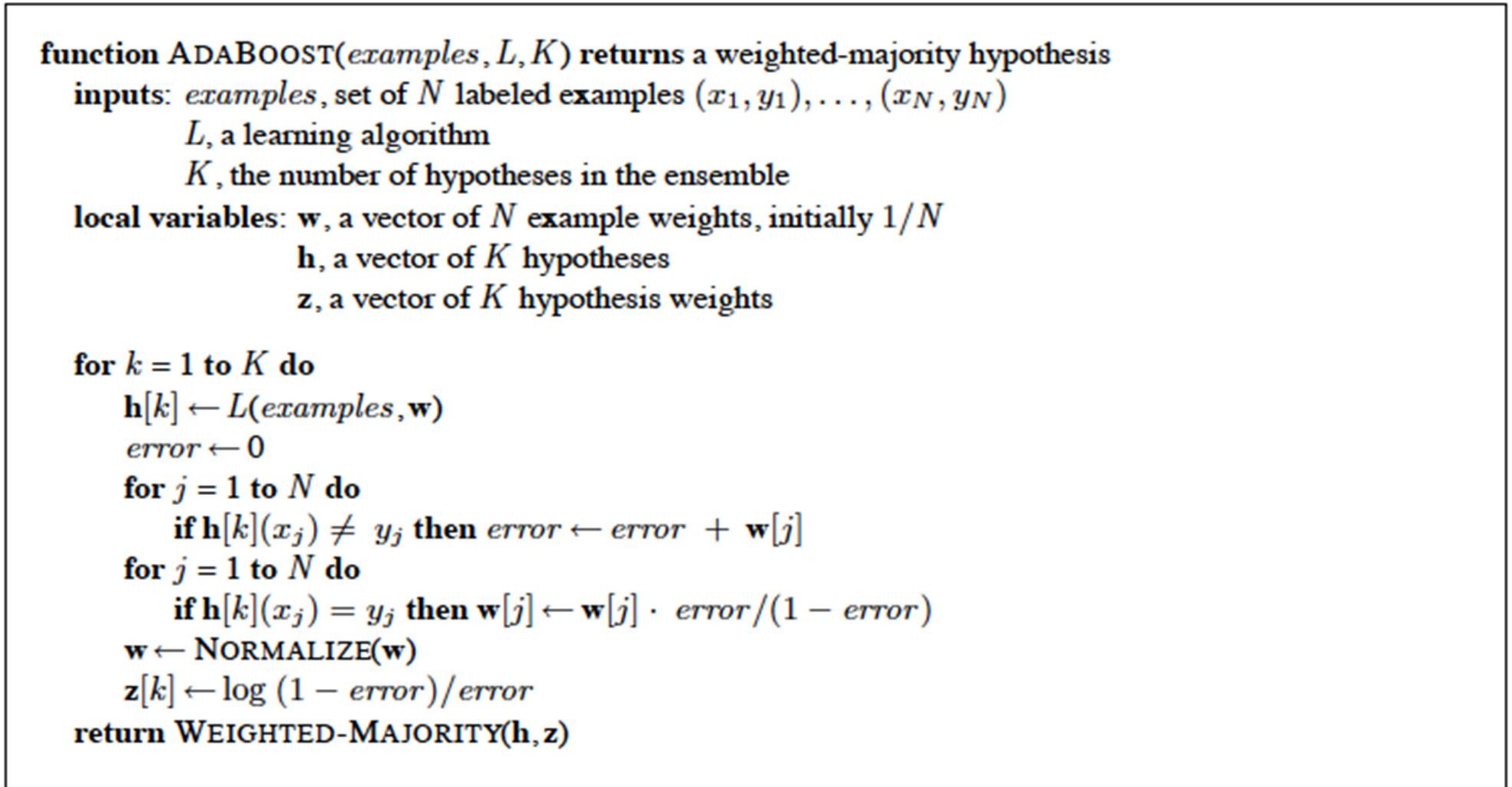
Ensembles: Boosting

- In practice hypothesis are rarely independent and some are more error-prone than others
- Weight majority:
  - Decrease weights of correlated hypothesis
  - Increase weights of good hypothesis
- Boosting

Ensembles: Boosting



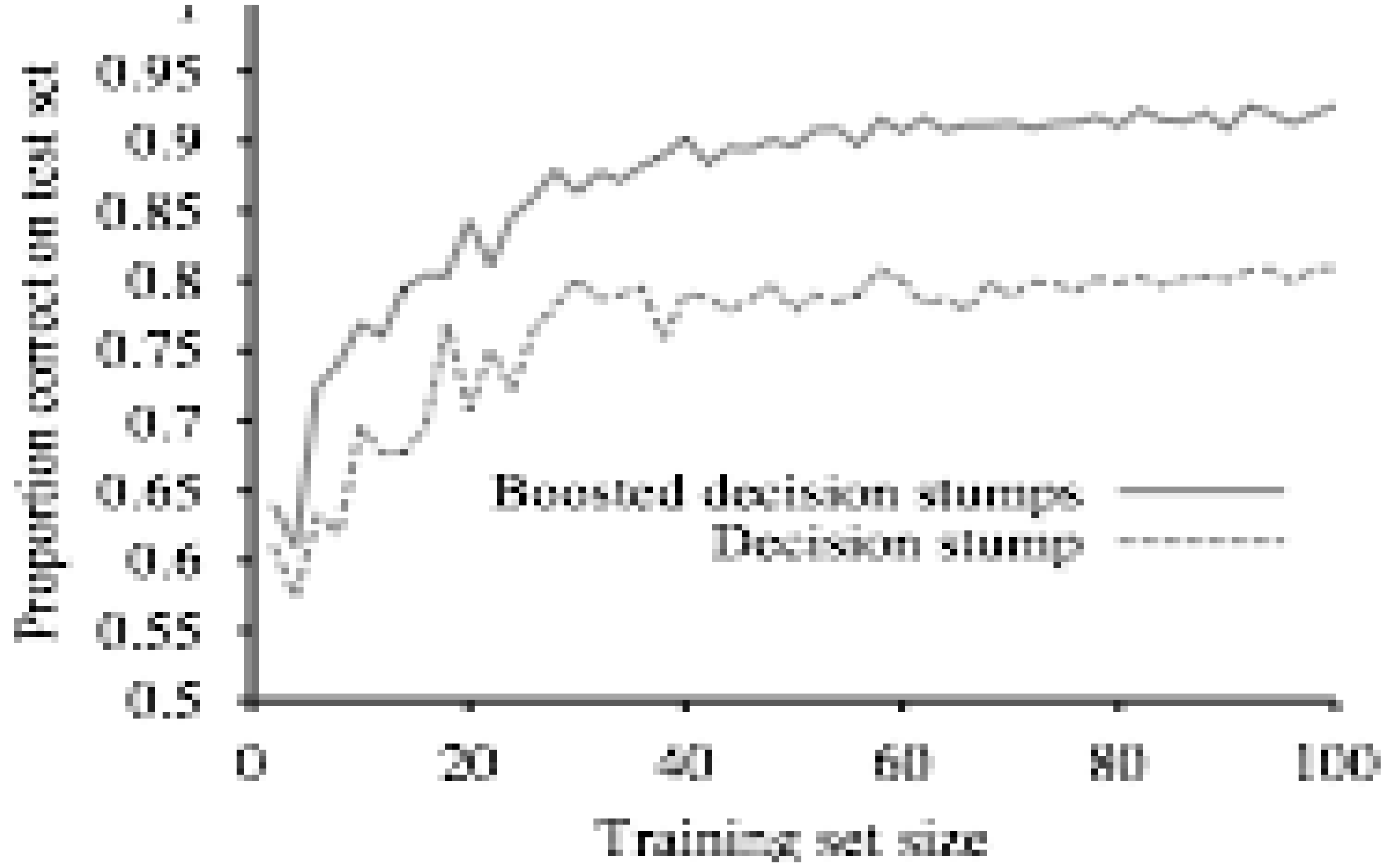
Ensembles: AdaBoost



```
function ADABOOST($examples$, $L$, $K$) returns a weighted-majority hypothesis
  inputs: $examples$, set of $N$ labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
          $L$, a learning algorithm
          $K$, the number of hypotheses in the ensemble
  local variables: $\mathbf{w}$, a vector of $N$ example weights, initially $1/N$
                   $\mathbf{h}$, a vector of $K$ hypotheses
                   $\mathbf{z}$, a vector of $K$ hypothesis weights

  for $k = 1$ to $K$ do
      $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
      $error \leftarrow 0$
      for $j = 1$ to $N$ do
          if $\mathbf{h}[k](x_j) \neq y_j$ then $error \leftarrow error + \mathbf{w}[j]$
      for $j = 1$ to $N$ do
          if $\mathbf{h}[k](x_j) = y_j$ then $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
      $\mathbf{w} \leftarrow$ NORMALIZE($\mathbf{w}$)
      $\mathbf{z}[k] \leftarrow \log\,(1 - error)/error$
  return WEIGHTED-MAJORITY($\mathbf{h}$, $\mathbf{z}$)
```

Ensembles: Boosting



K=5

Ensembles: Boosting

- Many variations of boosting (AdaBoost is a specific boosting algorithm)
  - Takes a weak learning L (classifies slightly better than just random guessing) and returns a hypothesis that classifies training data with 100% accuracy

Robert Schapire and Yoav Freund
Kanellakis Award for 2004