Learning From Examples

- Improve performance based on real observations – not preprogrammed reactions
- Designers cannot account for all possible situations or a changing environment
- There may not be an obvious or feasible solution for some problems
  - Object segmentation
  - Sentiment analysis

Learning From Examples

Typically…

- Input: The relevant world is represented as a vector of features
  - Pixels in an image
  - Words in a sentence
- Output: some value
  - An item an image represents
  - A sentiment classification
  - Wait time

Types of Learning

- Supervised
  - Input/output pairs
  - Given **labeled** data, learn the features in the input that best predict the output
- Unsupervised
  - Find patterns in a set of unlabeled data
  - Typically cluster them into certain groups

Supervised Learning

- **Hypothesis** – a *function* that maps input to output
- **Hypothesis Space** - the set of all *possible* hypotheses
- **Learning** – a search through possible hypothesis
- **Data set** – labeled input/output
  - Training set – the set of data the learning algorithm uses to create a hypothesis
  - Test set – the set of examples that we use to determine how well the hypothesis generalizes

Supervised Learning

- Classification – output is from a finite set of values
  - Weather, sentiment,
  - Binary - either in a set or not in a set
- Regression – output is a continuous value
  - Temperature
  - Price
  - Wait time

Stationary Assumption

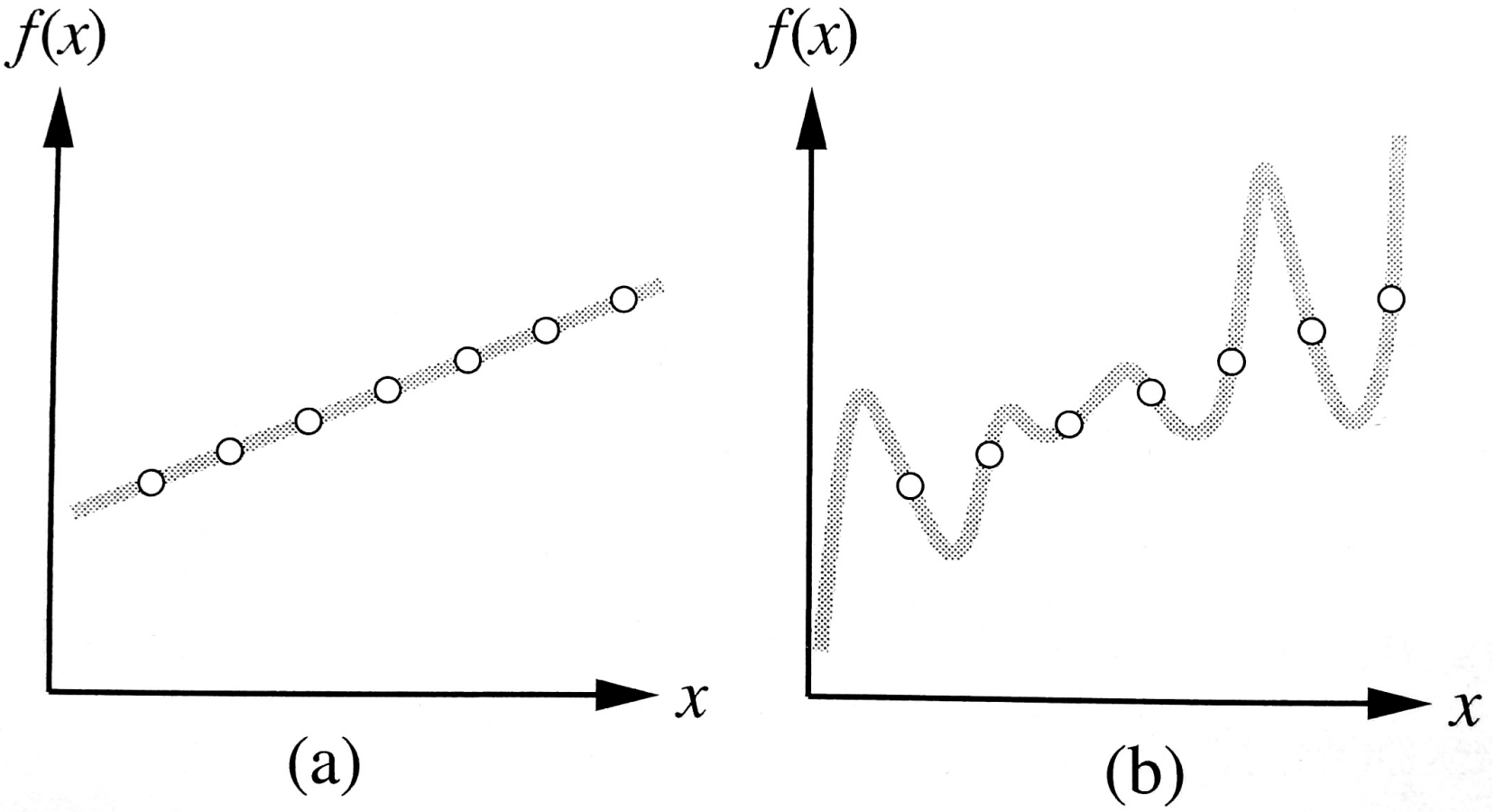
- The distribution we drew the training set from remains constant over time
- We assume that our training set examples are independent and drawn from an identical distribution (i.i.d)
- Each example is drawn from the same distribution

$$P(E_i) = P(E_{i-1}) = P(E_{i-2}) = \ldots$$

- Each example is drawn independently of one another

$$P(E_i | E_{i-1}, E_{i-2}, \ldots) = P(E_i)$$

Supervised Learning



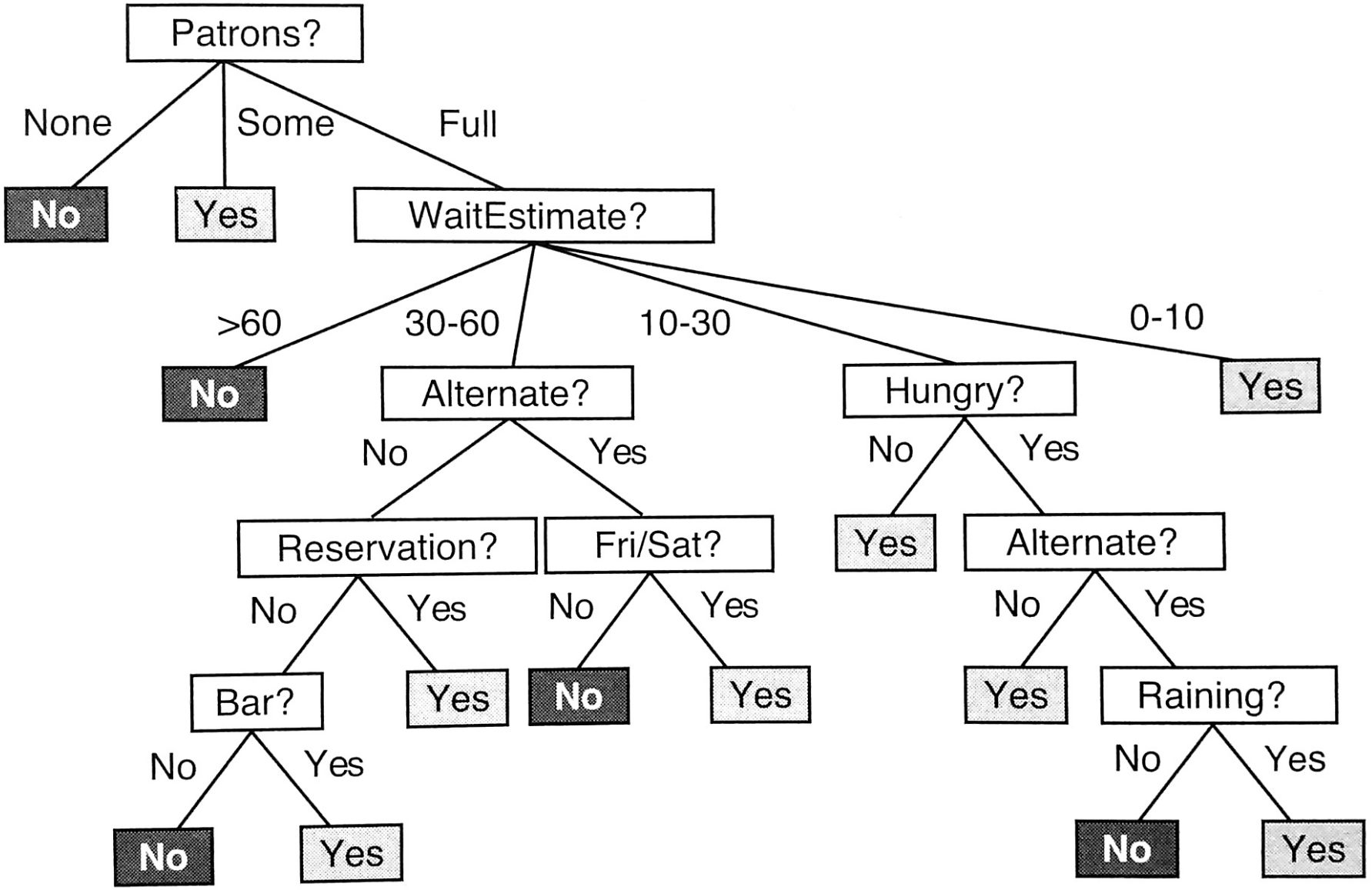
Decision Trees

- A tree where each node is an attribute and each edge is a possible value of that attribute
- **Goal predicate** – output, the value we are trying to determine
- Leaf nodes have decision value for the goal associated with them
- Maps a vector of attribute values to a single output value representing a decision

Decision Trees

Alternate: whether there is a suitable alternative restaurant nearby.
Bar: whether the restaurant has a comfortable bar area to wait in.
Fri/Sat: true on Fridays and Saturdays.
Hungry: whether we are hungry.
Patrons: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
Price: the restaurant's price range ($, $$, $$$).
Raining: whether it is raining outside.
Reservation: whether we made a reservation.
Type: the kind of restaurant (French, Italian, Thai, or burger).
WaitEstimate: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).

Decision Trees



Decision Trees

- Boolean decision tree – the goal and every attribute has exactly two values (true/false)
- A path from the root to a leaf node can be represented by a conjunction of predicates
- Tree is equivalent to propositional logic:
  - Goal ↔ $Path_1$ ∨ $Path_2$ ∨ …
  - Goal ↔ Disjunction of all paths leading to True
- Hypothesis Space = # of possible boolean *functions* of arity n
  - # of unique truth tables of n predicates and 1 expression (goal)
- $2^{(2^n)}$

Inducing From Examples

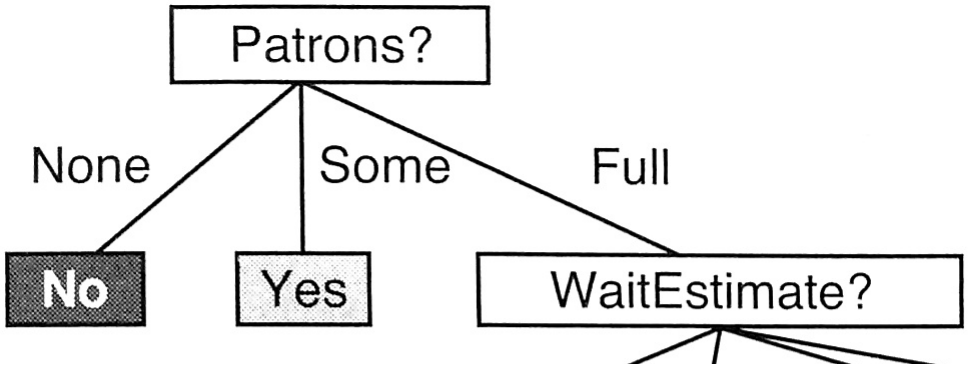
- Training set – set of pairs of attribute vectors (**x**) and goals values (y)
  - (**x**,y)
  - Positive Examples: All pairs where y is True
  - Negative Examples: All pairs where y is False
- Find a tree that is consistent with the training set (given **x** outputs the same y) while being as small as possible

Inducing From Examples

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

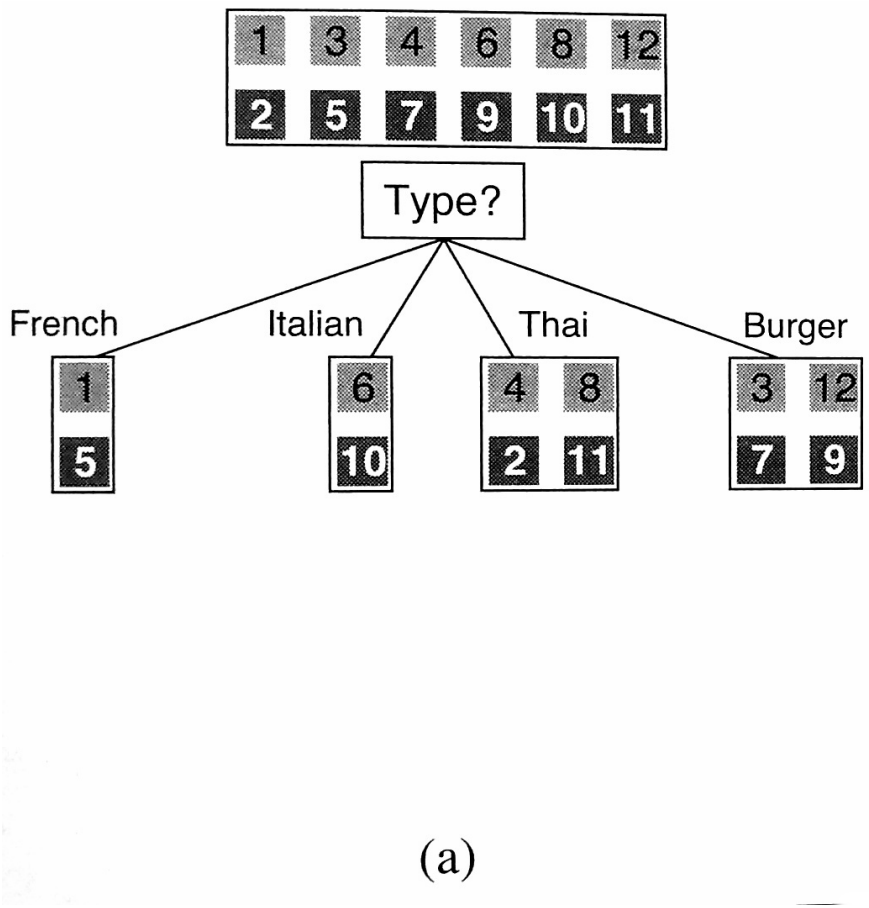
Inducing From Examples

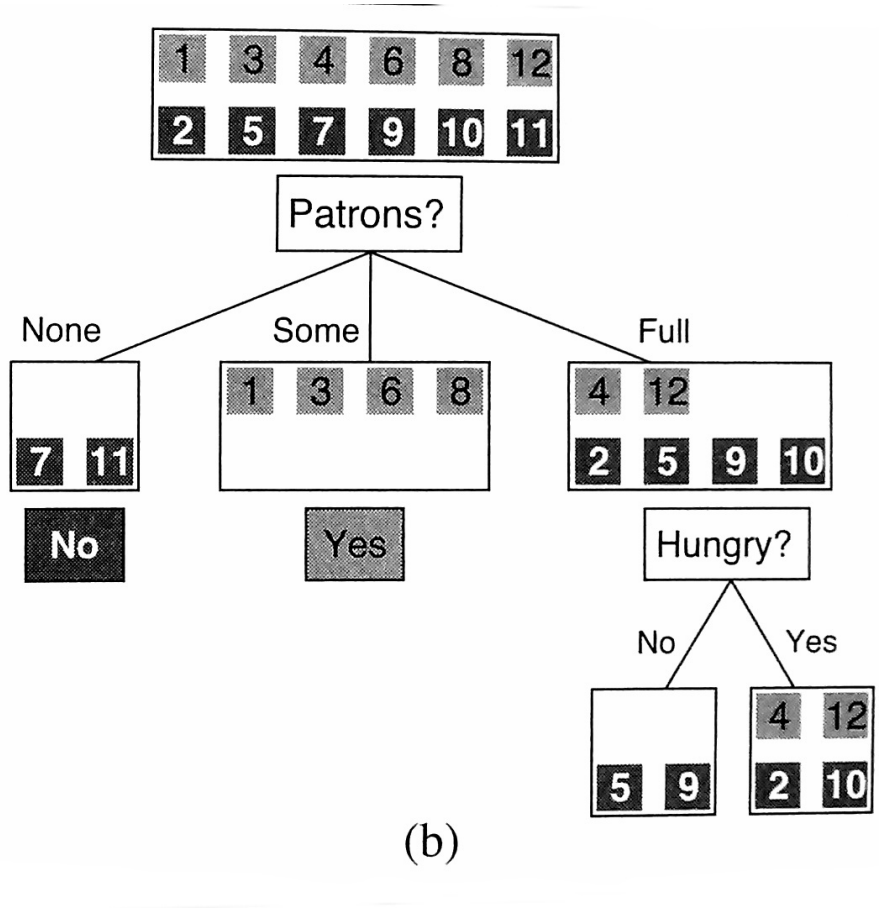
- Searching 2^(2^n) functions is infeasible
- Using a simple heuristic* to cut corners we can induce a “pretty good” tree that is consistent but not necessarily the smallest
- If we could find an attribute that best predicts the decision then we can divide an conquer



* colloquial; a broader sense than when we used it in A*

Inducing From Examples





Inducing From Examples

Given the training set and a set of attributes, A

- If the examples are all the same value, we answer that value
- Find the most informative attribute, *a*, recurse on the training set split up by each of the possible values of *a* and A-{a}
- Edge cases:
  - If there set of attributes is empty and the training set is not all one label, return which ever answer has the majority
    - **Noise**
  - If there are no examples left then the training set hasn't seen this scenario before. Return which ever answer of has the majority in the recursive step immediately preceding this

Inducing From Examples

function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$)
a tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
$A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
$tree \leftarrow$ a new decision tree with root test A
for each value v_k of A **do**
$exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
$subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
return $tree$

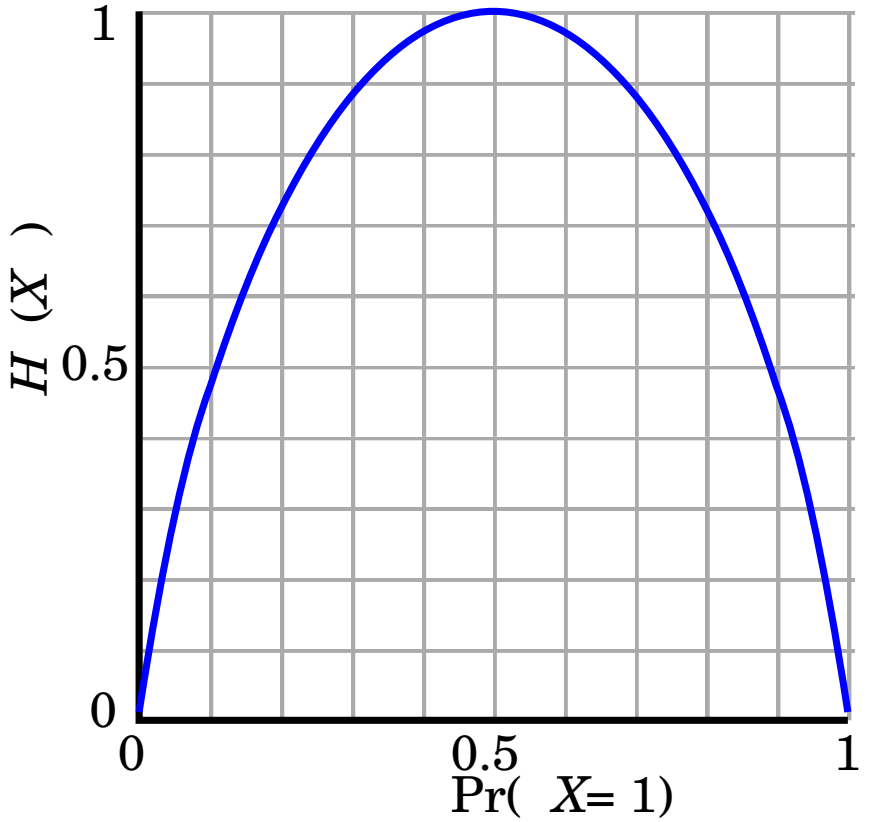
Measuring Importance

- How do we choose the most informative attribute that reduces the complexity of our problem
- Ideally we can find an attribute that splits the training data into sets of examples with the same label
- We need a way to determine Patrons better reduces the problem than Type

Information and Entropy

- In information theory, entropy measure the amount of uncertainty a random variable in bits
- H(V) – the entropy of an random variable, V
- A random variable provides the most information we we are least sure of its value
- H(Fair_Coin) = 1
- P(Loaded=H)=.99

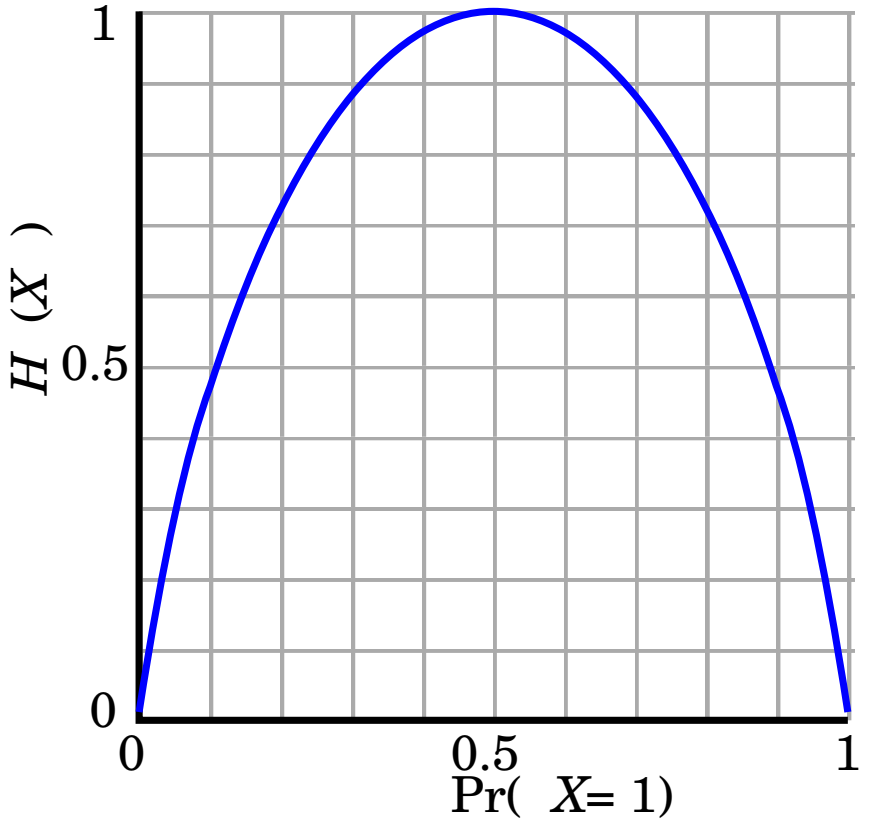
  H(Loaded) = .08



Information and Entropy

- In information theory, entropy measure the amount of uncertainty a random variable in bits
- H(V) – the entropy of an random variable, V
- A random variable provides the most information we we are least sure of its value
- H(Fair_Coin) = 1
- P(Loaded=H)=.99

  H(Loaded) = .08



Information and Entropy

In general

- $H(V)=\sum_K P(v_k)\log_2 \frac{1}{P(v_k)}=-\sum_K P(v_k)\log_2 P(v_k)$
- P(F=apple) = .5
- P(F=orange) = .25
- P(F=banana) = .25
- 1.5

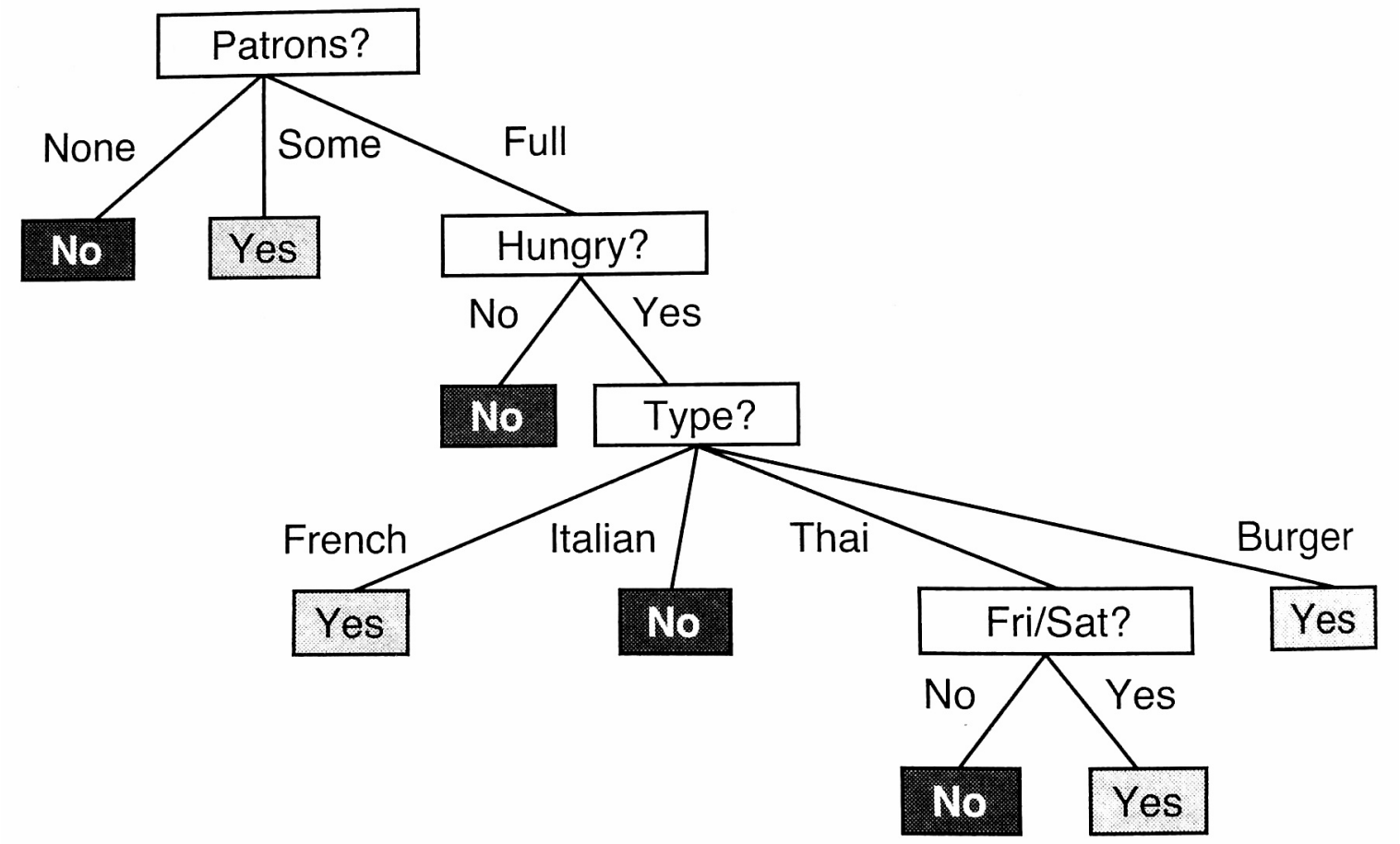
For boolean random variables, q = P(V =True)

- $B(q)=-(q\log_2 q+(1-q)\log_2(1-q))$
- P(Rain=True) = .2
- H(Rain) = .7219

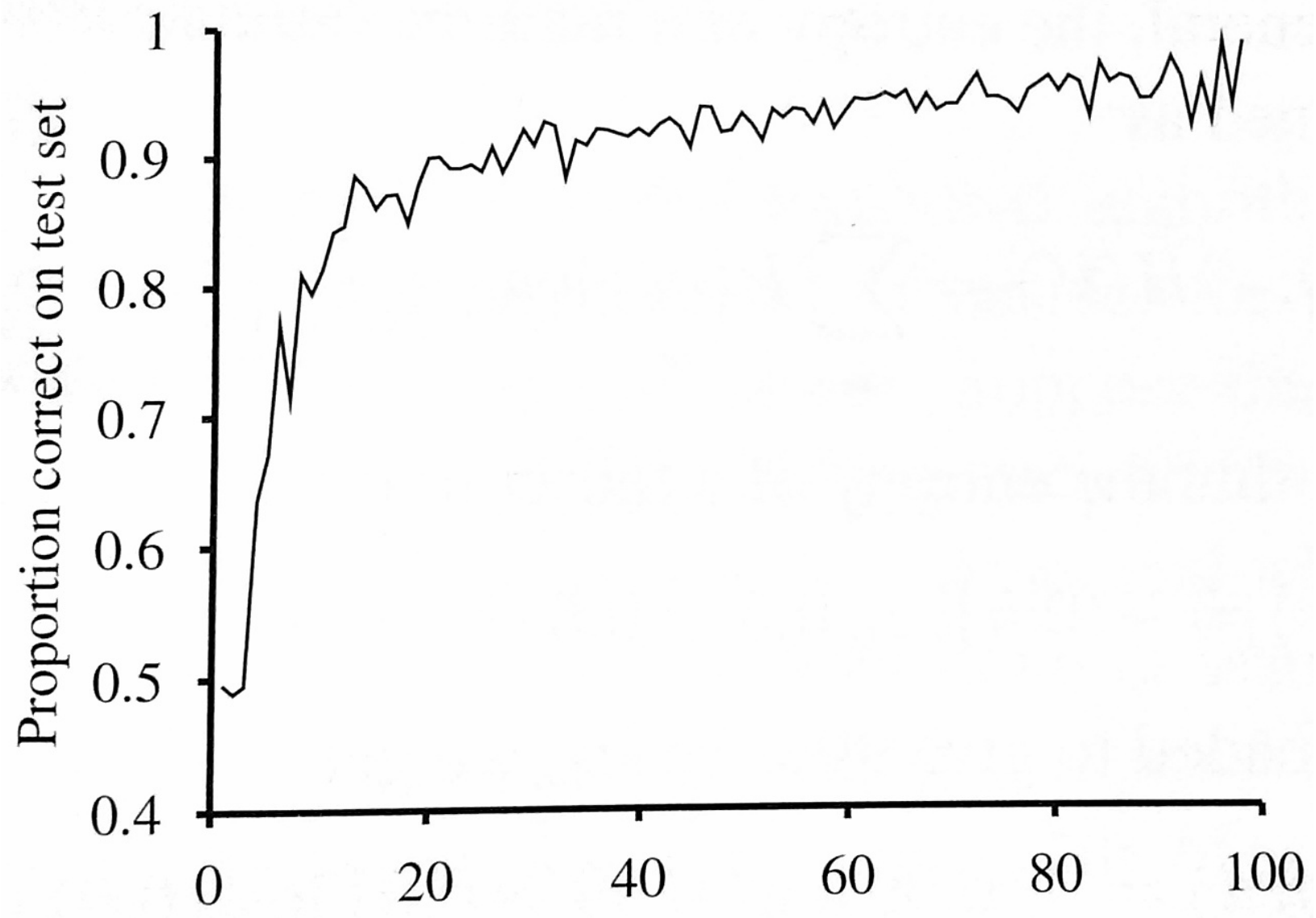
Inducing From Examples

- H(ExampleSet) = B(p/(p+n))
- Remaining entropy after splitting on an attribute is a sum of each branches new example sets scaled by their size relative to the current set
- $$Remainder(A) = \sum_{k \in A_{values}} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$$
- The importance of the attribute is measured by the information gained after splitting on A
- $$Gain(A) = B\left(\frac{p}{p+n}\right) - Remainder(A)$$

Inducing From Examples



Learning Curves



Inducing From Examples

Example						Goal
	Alt	*Bar*	*Hun*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*Yes*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*No*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$

Pruning

- **Overfitting** – learning patterns that emerge by chance in the training set that are not encountered in actual problem space
- Pruning – remove the nodes that only have leaf node children if the information we gain from them is below some threshold

Significance

- **Null hypothesis** (H_0)- assume there is not an underlying pattern we can detect
- Significance testing : determine if the performance of the learned hypothesis (H_1) is not statistically different enough from the performance of H_0
- Given some evaluation set,S, that H_1 performs better on than H_0

  What is the likelihood that we picked that set at random given that H_0 is true

  - If the probability is low we reject H_0 and accept H_1

Pruning

- We prune nodes that do not do statistically better than H_0
- H_0 -The attribute is irrelevant; as the example set grows the information gain tends to 0
- We need to compare the expected number of positive and negative values to the actual number we get by splitting with an attribute
- If they do not differ *significantly* then we can prune that node

Pruning

- Expected values:

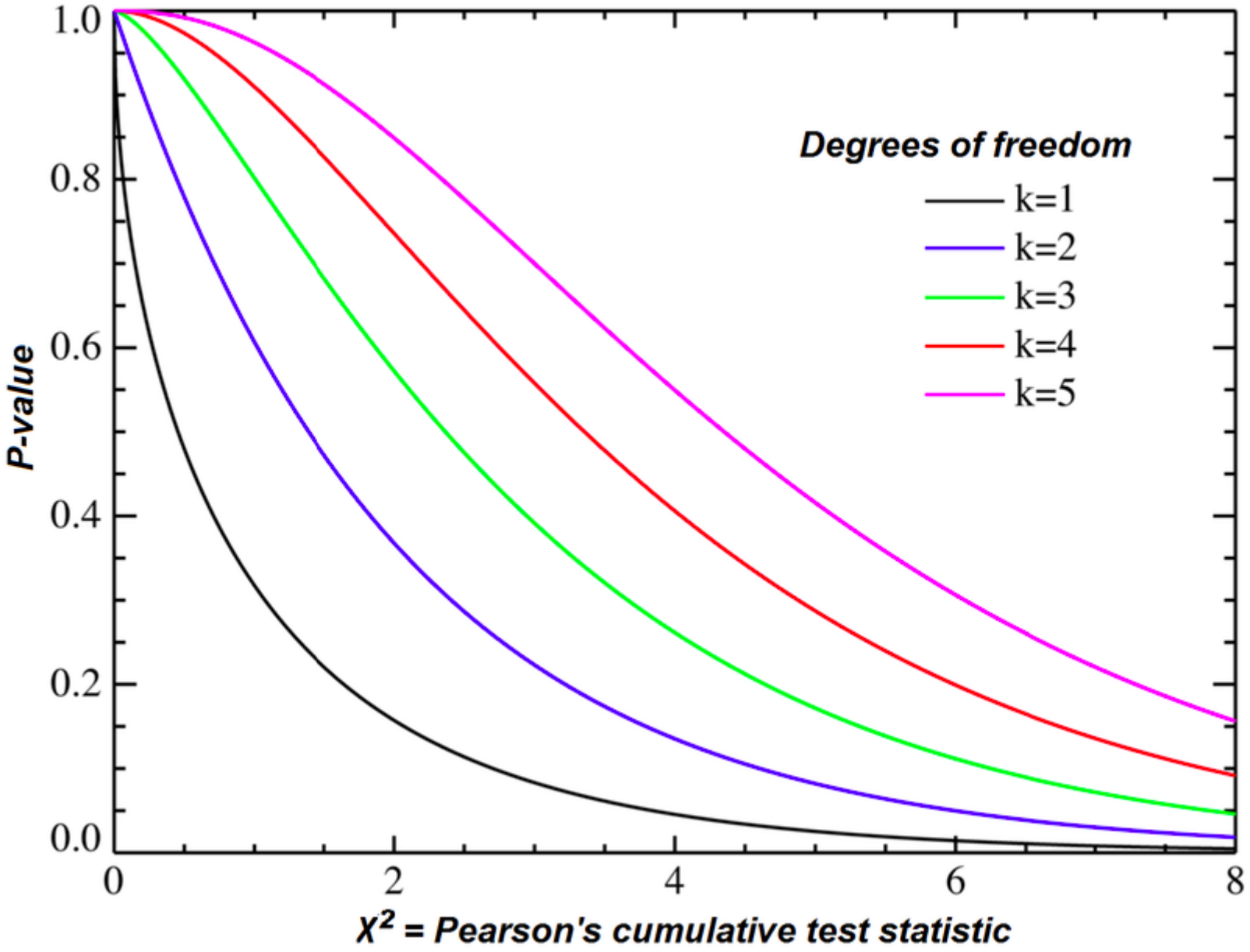
$$\hat{p}_k = p \times \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \times \frac{p_k + n_k}{p + n}$$

- Deviation

$$\Delta = \sum_{k \in A_{values}} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

- χ^2(Chi-Squred) test:
  - Deviation – [above]
  - Degree of freedom - #possible values – 1
  - p-value – the likelihood at which we are willing to reject H_0

Pruning

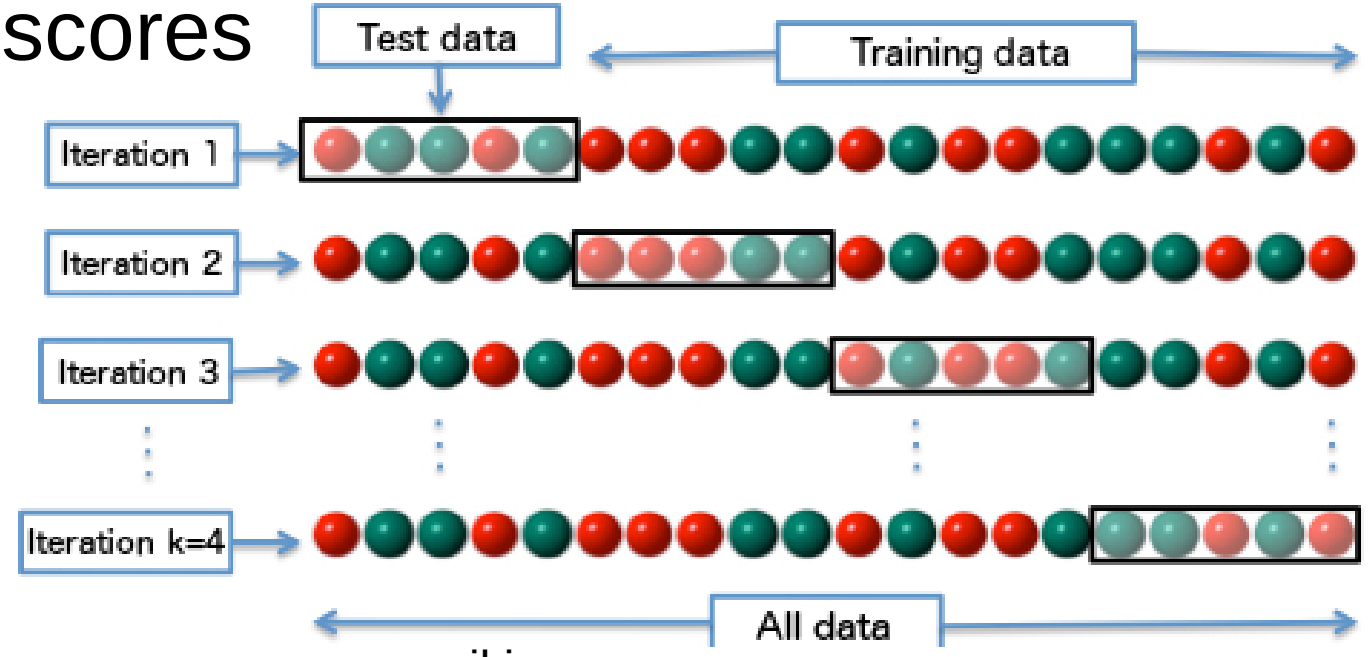

1.0
0.8
0.6
0.4
0.2
0.0
P-value
0
2
4
6
8
χ² = Pearson's cumulative test statistic
Degrees of freedom
k=1
k=2
k=3
k=4
k=5

Evaluation

- Split set of examples into a training set and a (smaller) test set
- After training, evaluate the hypothesis, h(), on the test set
- Given an observation (**x**,y), determine if h(**x**)=y
- **Error rate:** # of mistakes/ |test set|
- **Loss Function:** errors *can* have different weights
  - L(y',y) → Real#
  - h(**x**) = y'

K-fold Cross Validation

- Reserve 1/k of the available data for the testing set
- Run training a model and evaluate (as before)
- Repeat k times with different test set
- Average test scores



wikicommons

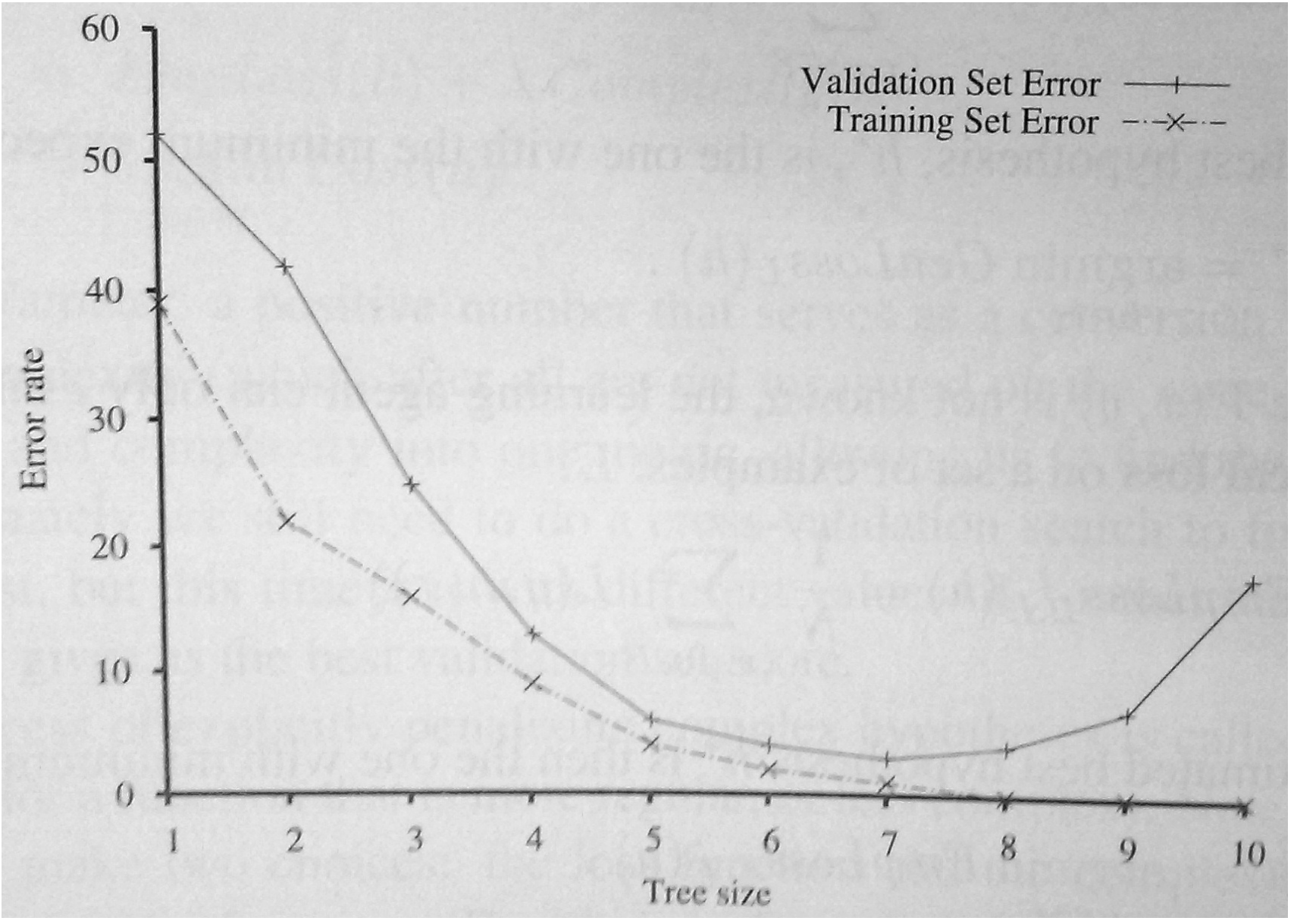
Model Selection

- Optimization – finding the best hypothesis (what we've been doing) in a hypothesis space
- Model selection – finding an appropriate hypothesis space for your problem

  For instance:

  - Restrict decision tree to a certain depth or number of nodes
  - Only consider polynomials of a certain degree

Model Selection



Regularization

- It is possible to overfit with multiple variants
  - some features (dimensions) are irrelevant but our example set happen to contain a pattern
- Regularize – minimize the empirical loss + the complexity of the hypothesis
  - Cost(h) = EmpLoss(h) + λComplexity(h)
  - λ – a positive parameter that balances the weight of the complexity against the loss

Complexity

Typically for linear functions …

- $Complexity(h_{\boldsymbol{w}}) = L_q(\boldsymbol{w}) = \sum_i |w_i|^q$
- The complexity grows as a function of the size of the weights
- At q = 1, many weights set to 0 are preferred
- **Sparse model –** models with most of their weights set to 0
- Reduce the # of examples exhibiting a certain feature required to learn a good model at the risk oversimplifying

Ensemble Learning

- 2 binary classification hypotheses for the same distribution, h and h’
- Each with a probability, *p*, of misclassifying some example
- Assume there error is independent
  - $P(h(x) = y \mid h'(x) = y) = P(h_i(x) = y)$
- The likelihood that they both get the answer wrong is lower than their individual probabilities
  - $P(h(x) != y \wedge h'(x) != y) < P(h(x) != y)$
- We could construct K hypotheses (an **ensemble**) and determine our classification by a majority vote

Ensemble Learning

- $h(x) \rightarrow \{1,-1\}$
- For now: $h(x) = (h_i(x) + \ldots h_1(x))$
- The hypothesis space for h(x) is all the possible sets of all hypotheses in the original problem space

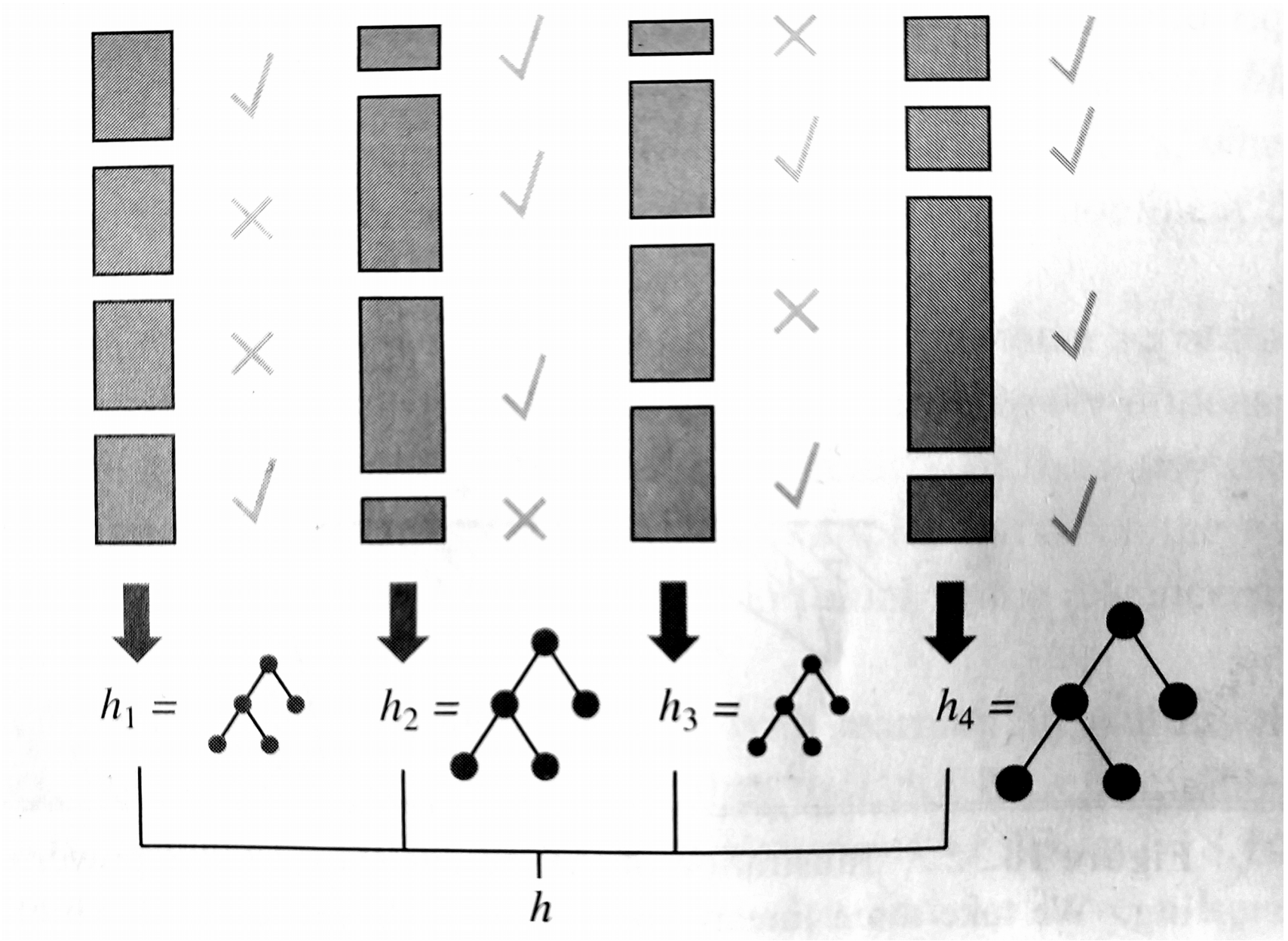
Boosting

- **Weighted training set** – set of examples with positive weights that determine how important they are to learning a hypothesis
- Train h_1 with every weight=1
- After training
  - decrease the weight for every example h_1 correctly classifies
  - Increase the weight for examples h_1 gets wrong
- Train h_2 with the new weighted training set

  Repeat for K hypotheses

Boosting

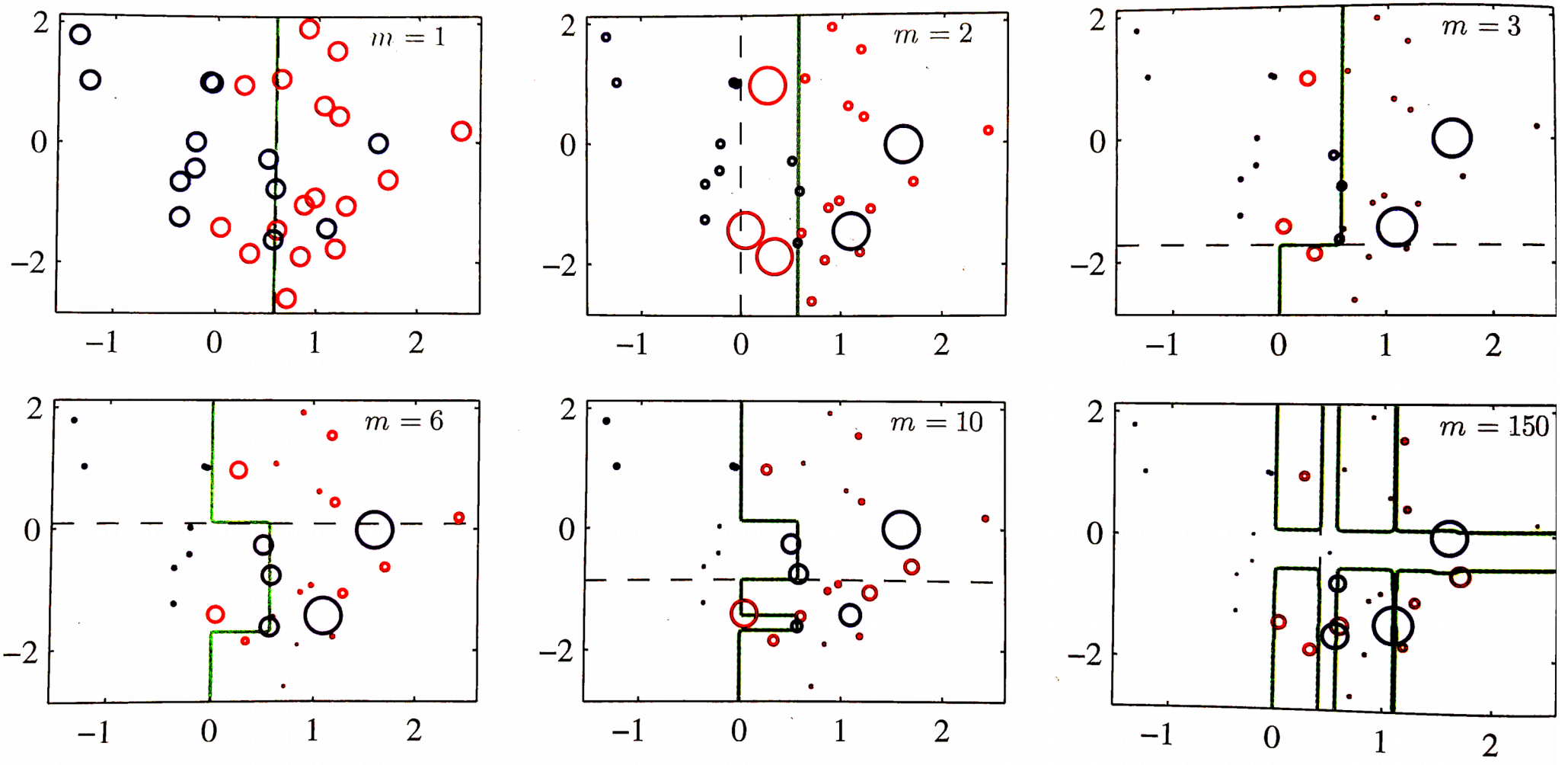
- The final hypothesis is a weighted majority combination of the K hypotheses we just learned
- w_i is determined by how well h_i performs on the training set
- In general a booting method takes a set of examples, an learning algorithm (*L*), and the # of hypotheses to learn (K)


$h_1 =$
$h_2 =$
$h_3 =$
$h_4 =$
h

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: **w**, a vector of N example weights, initially $1/N$
    h, a vector of K hypotheses
    z, a vector of K hypothesis weights

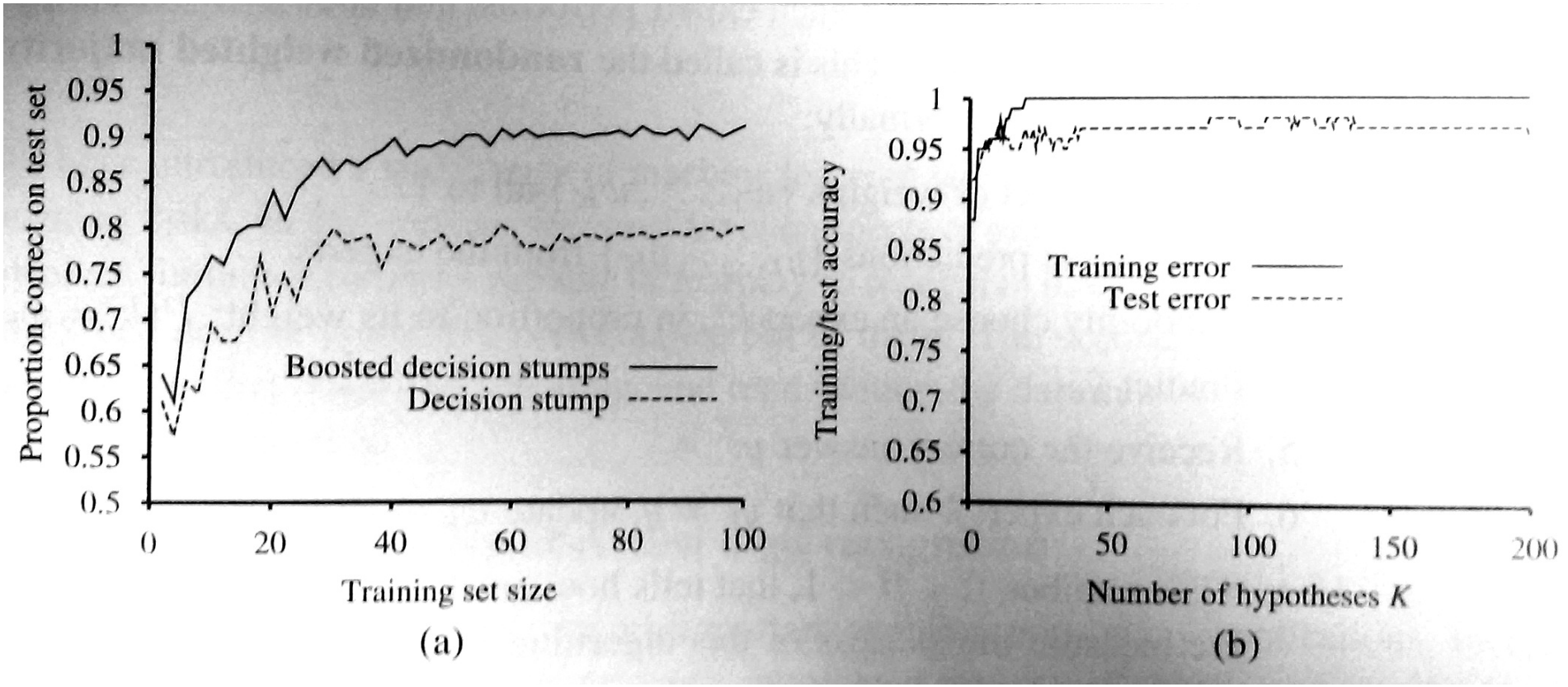
  for $k = 1$ **to** K **do**
    $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) = y_j$ **then**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



Pattern Recognition and Machine Learning – Bishop

AdaBoost

- Specifies that L can be a **weak learning** algorithm
- Weak learning generates hypotheses barely better than the base line
  - e.g. in boolean decisions with P(True)=.5; L gets (.5+ϵ) of the examples correct


Proportion correct on test set
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0
20
40
60
80
100
Training set size
Boosted decision stumps
Decision stump
(a)
Training/test accuracy
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0
50
100
150
200
Number of hypotheses K
Training error
Test error
(b)