CSE 473

Chapter 18

Decision Trees and Ensemble Learning

Recall: Learning Decision Trees

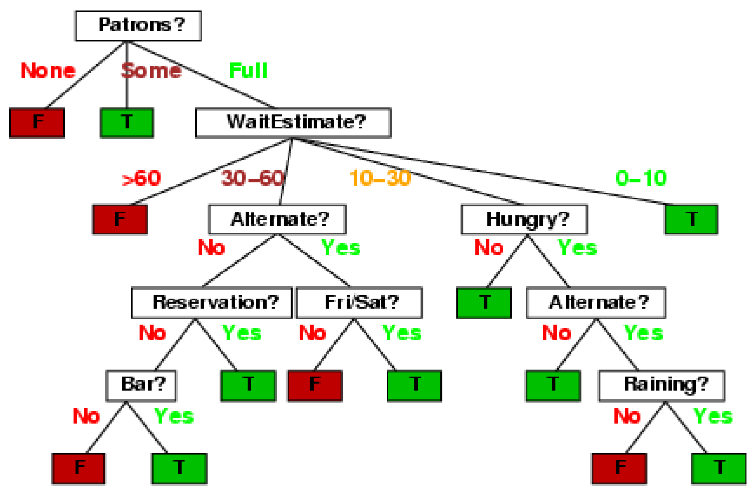
Example: When should I wait for a table at a restaurant?

Attributes (features) relevant to *Wait?* decision:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Example Decision tree

A decision tree for *Wait?* based on personal "rules of thumb" (this was used to generate input data):



Input Data for Learning

- Past examples where I did/did not wait for a table:

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

- Classification of examples is positive (T) or negative (F)

Decision Tree Learning

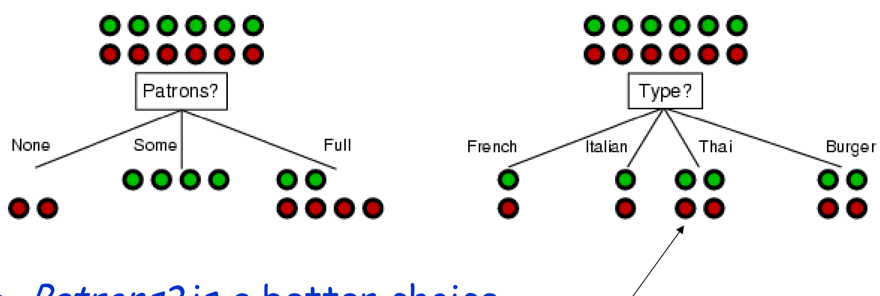
- *Aim: find a small tree consistent with training examples*
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
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
```


Choosing an attribute to split on

- Idea: a good attribute should reduce uncertainty

  E.g., splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice

To wait or not to wait is still at 50%.

How do we quantify uncertainty?

Using information theory to quantify uncertainty

- Entropy measures the amount of uncertainty in a probability distribution
- Entropy (or Information Content) of an answer to a question with possible answers $v_1, \ldots, v_n$:

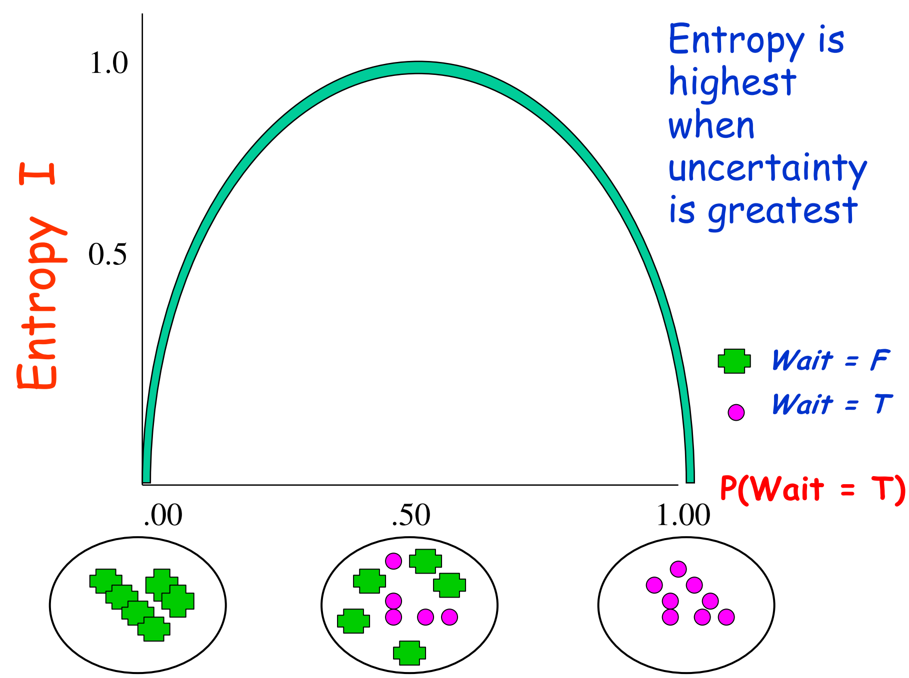
$$I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$$

Using information theory

- Imagine we have p examples with Wait = True (positive) and n examples with Wait = false (negative).
- Our best estimate of the probabilities of Wait = true or false is given by: $P(true) \approx p / p+n$
  $p(false) \approx n / p+n$
- Hence the entropy of Wait is given by:

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

Entropy is highest when uncertainty is greatest

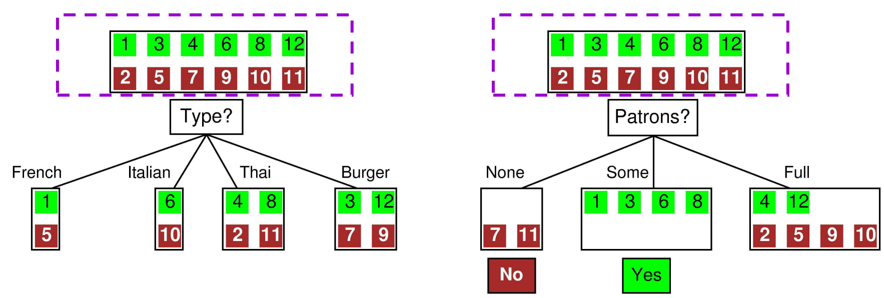


Choosing an attribute to split on

- Idea: a good attribute should reduce uncertainty and result in "gain in information"
- How much information do we gain if we disclose the value of some attribute?
- Answer:

uncertainty before – uncertainty after

Back at the Restaurant



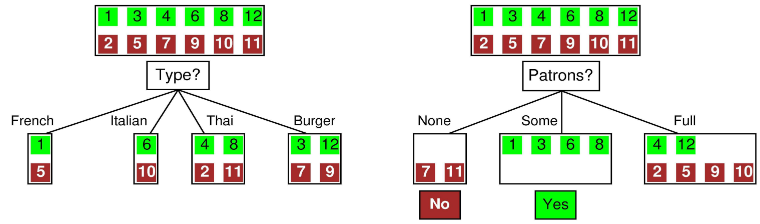
Before choosing an attribute:

Entropy = - 6/12 log(6/12) – 6/12 log(6/12)

= - log(1/2) = log(2) = 1 bit

There is "1 bit of information to be discovered"

Back at the Restaurant



If we choose Type: Go along branch "French": we have entropy = 1 bit; similarly for the others.

Information gain = 1-1 = 0 along any branch

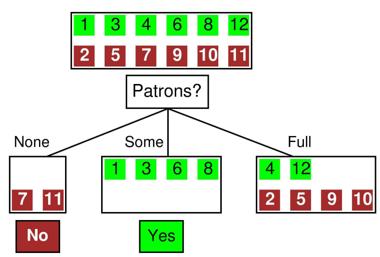
If we choose Patrons:
In branch "None" and "Some", entropy = 0
For "Full", entropy = -2/6 log(2/6)-4/6 log(4/6) = 0.92

Info gain = (1-0) or (1-0.64) bits > 0 in both cases

So choosing Patrons gains more information!

Entropy across branches

- How do we combine entropy of different branches?
- Answer: Computed average entropy
- Weight entropies according to probabilities of branches
  - 2/12 times we enter "None", so weight for "None" = 1/6
  - "Some" has weight: 4/12 = 1/3
  - "Full" has weight 6/12 = ½



$$AvgEntropy = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} Entropy\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

weight for each branch

entropy for each branch

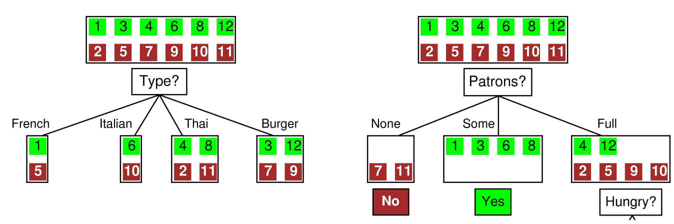
Information gain

- Information Gain (IG) or reduction in entropy from using attribute A:

IG(A) = Entropy before – AvgEntropy after choosing A

- Choose the attribute with the largest IG

Information gain in our example



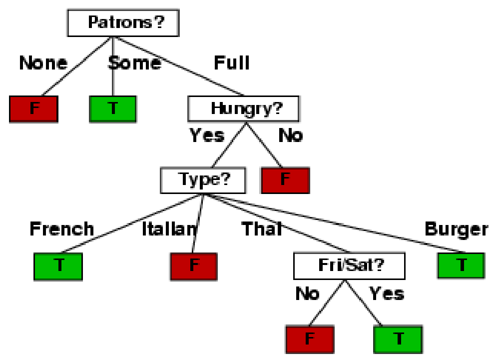
$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes
⇒ Chosen by the DTL algorithm as the root

Should I stay or should I go? Learned Decision Tree

- Decision tree learned from the 12 examples:

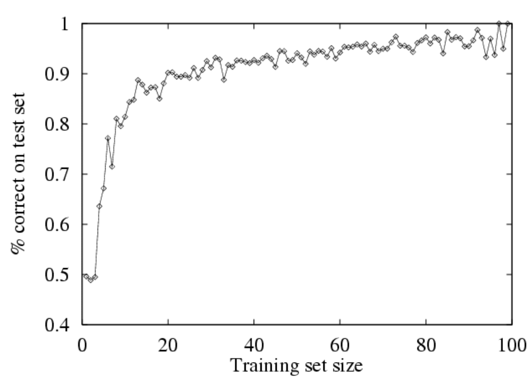


- Substantially simpler than other tree
  more complex hypothesis not justified by small amount of data

Performance Measurement

How do we know that the learned tree $h \approx f$?
Answer: Try h on a new test set of examples

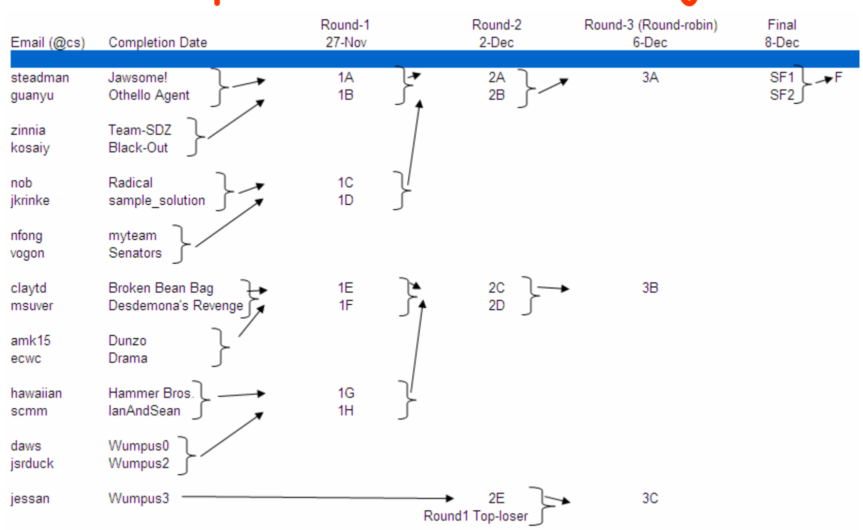
Learning curve = % correct on test set as a function of training set size



Ensemble Learning

- Sometimes each learning technique yields a different hypothesis (or function)
- But no perfect hypothesis…
- Could we combine several imperfect hypotheses to get a better hypothesis?

Example 1: Othello Project

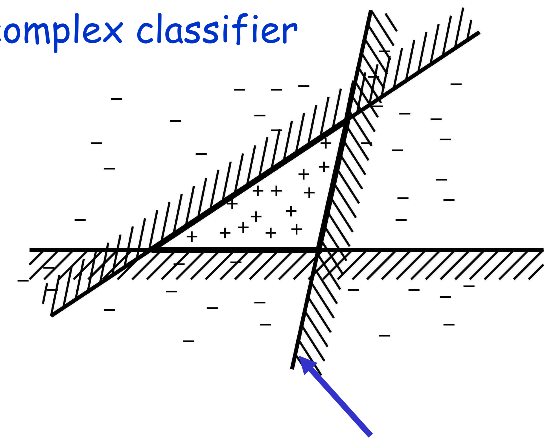


Many brains better than one?

Example 2

Combining 3 linear classifiers

⇒ More complex classifier



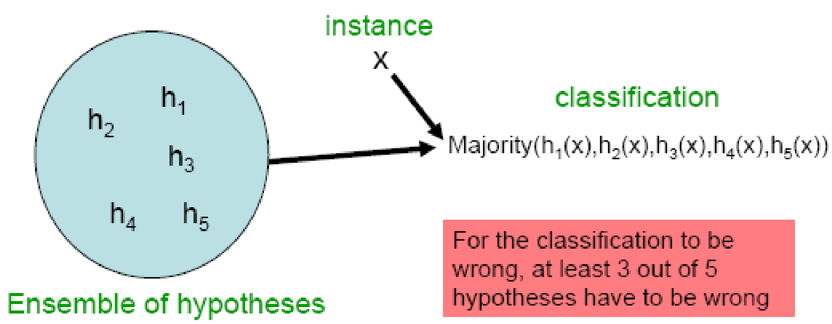
this line is one simple classifier saying that everything to the left is + and everything to the right is -

Ensemble Learning: Motivation

- Analogies:
  - Elections combine voters' choices to pick a good candidate (hopefully)
  - Committees combine experts' opinions to make better decisions
  - Students working together on Othello project
- Intuitions:
  - Individuals make mistakes but the "majority" may be less likely to (true for Othello? We shall see…)
  - Individuals often have partial knowledge; a committee can pool expertise to make better decisions

Technique 1: Bagging

- Combine hypotheses via majority voting



Bagging: Analysis

- Assumptions:
  - Each h_i makes error with probability p
  - The hypotheses are independent
- Majority voting of n hypotheses:
  - k hypotheses make an error: $\binom{n}{k} p^k(1-p)^{n-k}$
  - Majority makes an error: $\Sigma_{k>n/2} \binom{n}{k} p^k(1-p)^{n-k}$
  - With n=5, p=0.1 → err(majority) < 0.01

Error probability went down from 0.1 to 0.01!

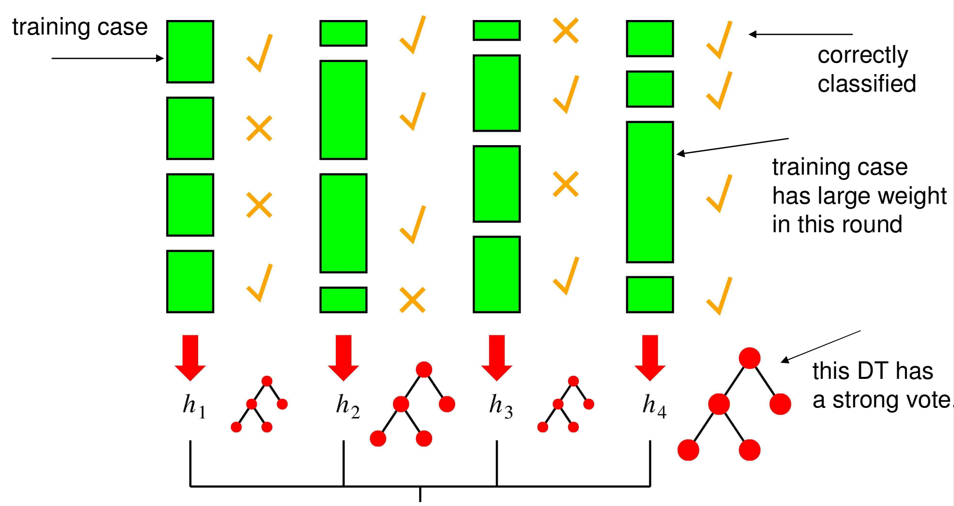
Weighted Majority Voting

- In practice, hypotheses rarely independent
- Some hypotheses have less errors than others $\Rightarrow$ all votes are not equal!
- Idea: Let's take a weighted majority

Technique 2: Boosting

- Most popular ensemble learning technique
  - Computes a weighted majority of hypotheses
  - Can "boost" performance of a "weak learner"
- Operates on a weighted training set
  - Each training example (instance) has a "weight"
  - Learning algorithm takes weight of input into account
- Idea: when an input is misclassified by a hypothesis, increase its weight so that the next hypothesis is more likely to classify it correctly

Boosting Example with Decision Trees (DTs)



Output of h_{final} is weighted majority of outputs of $h_1,\ldots,h_4$

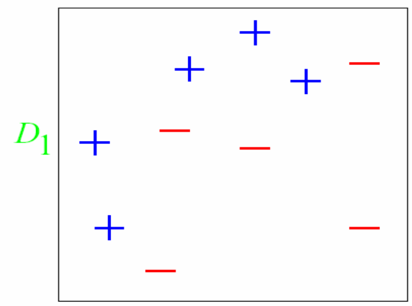
AdaBoost Algorithm

(Adaptive Boosting)

- $w_j \leftarrow 1/N \quad \forall_j$
- For m=1 to M do
  - $h_m \leftarrow$ learn(dataset,w)
  - err $\leftarrow$ 0
  - For each (x_j,y_j) in dataset do
    - If $h_m(x_j) \neq y_j$ then err $\leftarrow$ err + w_j
  - For each (x_j,y_j) in dataset do
    - If $h_m(x_j) = y_j$ then $w_j \leftarrow w_j$ err / (1-err)
  - w $\leftarrow$ normalize(w)
  - $z_m \leftarrow$ log [(1-err) / err]
- Return *weighted-majority(h,z)*

w: vector of N instance weights
z: vector of M hypoth. weights

AdaBoost Example

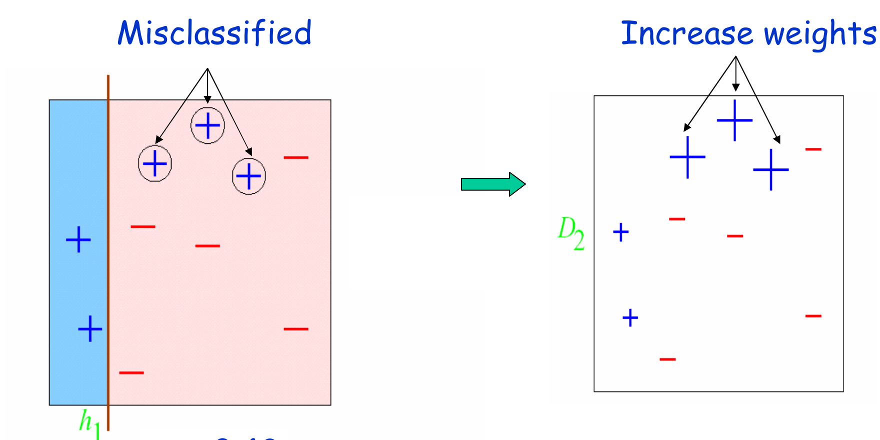


Original training set D_1 : Equal weights to all training inputs
Goal: In round t, learn classifier h_t that minimizes error with respect to weighted training set
h_t maps input to True (+1) or False (-1) $h_t : X \rightarrow \{-1,+1\}$

Taken from "**A Tutorial on Boosting" by** Yoav Freund and Rob Schapire

AdaBoost Example

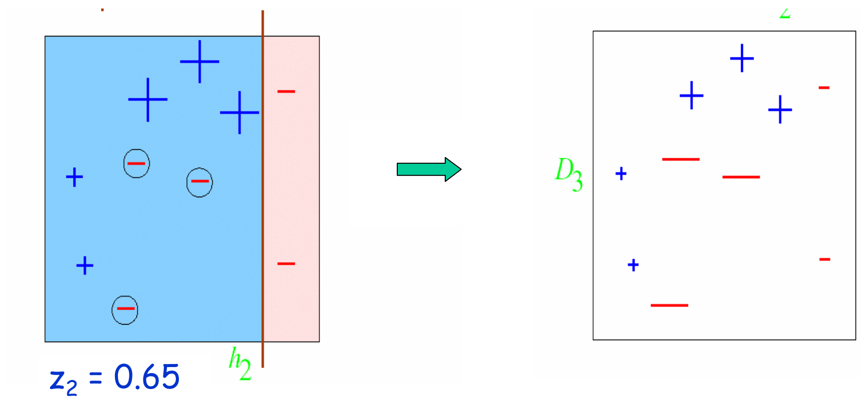
ROUND 1



$z_1 = 0.42$

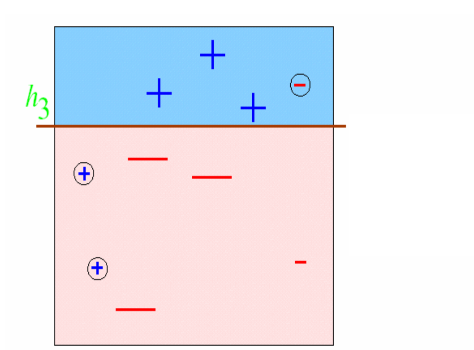
AdaBoost Example

ROUND 2



AdaBoost Example

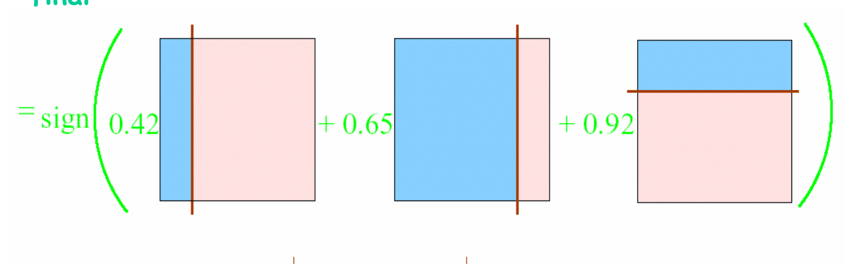
ROUND 3



$z_3 = 0.92$

AdaBoost Example

h_{final}



sign(x) = +1 if x > 0 and -1 otherwise

Next Time

- Classification using:
  - Nearest Neighbors
  - Neural Networks