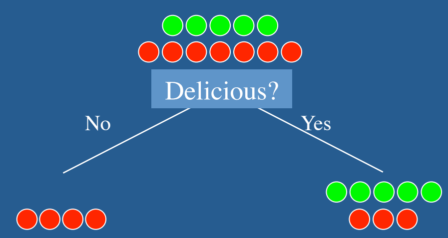
1



CLASSIFICATION

EECS 492 Lecture 22
April 5, 2011

Kalwar Update

2

Last Tournament Results

	0	1	2	3	4	5	6	7	8	Total
ebolson (0)	---	0.58	0.70	0.73	0.90	1.00	0.98	1.00	1.00	6.875
grwright (1)	0.43	---	0.60	0.88	0.83	0.88	1.00	1.00	1.00	6.600
ryjust (2)	0.30	0.40	---	0.88	0.90	0.83	0.93	1.00	1.00	6.225
DaffyDuck (3)	0.28	0.13	0.13	---	0.58	0.68	0.93	1.00	1.00	4.700
petnic (4)	0.10	0.18	0.10	0.43	---	0.65	0.88	0.98	1.00	4.300
attang (5)	0.00	0.13	0.18	0.33	0.35	---	0.93	1.00	1.00	3.900
DrunkenDuck (6)	0.03	0.00	0.07	0.08	0.13	0.07	---	0.95	0.93	2.250
saluber (7)	0.00	0.00	0.00	0.00	0.03	0.00	0.05	---	1.00	1.075
SittingDuck (8)	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.00	---	0.075

Classification and Regression

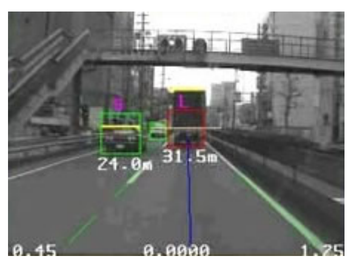
3

- We want to learn functions of the form:
  - y = f(x)
- Y is discrete valued:
  - Classification
- Y is continuous
  - Regression
- X can be one or more continuous or discrete values.
  - Often called "features"

Classification

4

- Estimate a discrete-valued quantity in terms of a number of features
  - Example: Car or Motorcycle?
    - Features:
      - Size in pixels
      - Aspect ratio
      - Average color
      - …



Regression

5

- Estimate a continuous-valued quantity in terms of a number of features
- Example: APPL stock price
  - Features:
    - Number of news articles about upcoming products
    - Last quarter's revenue
    - Cash on hand
    - Whether Steve Jobs is CEO
- Example: Movie rating predictions
  - Features:
    - How much did the user like other movies?
    - How much did other users like this movie?

Basics

6

- Training dataset
  - Data used to learn our model
- Test dataset
  - Data used to see how well we've learned f(x)
  - Why is this separate from training data?

A trip to the classification zoo

7

- kNN
- Decision Trees
- Boosting
- SVM
- Neural networks

K Nearest Neighbors

8

K Nearest Neighbors

- Given feature vector **x**, estimate y based on previously seen examples close to **x**
- K-Nearest Neighbors
  - Find k closest examples
    - Majority vote
  - Special data structures make nearest-neighbor lookups relatively fast. (How would you do it?)
- Very simple, effective, little parameter tuning
  - A good "first try" method

Nearest Neighbor: A problem

10

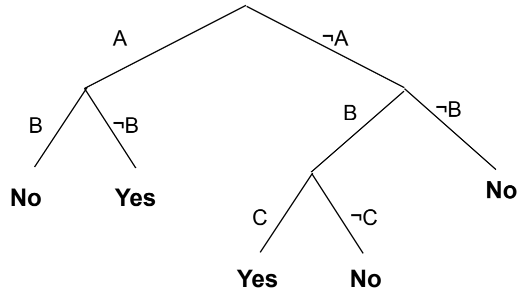
- Predict MPG given:
  - Feature 1: # of cylinders
  - Feature 2: Car mass (kg)
  - Distance = $(c_i-c_j)^2 + (m_i-m_j)^2$
- What happens?
  - # of cylinders doesn't matter much at all!
  - Scaling matters!
    - Normalization

Decision Trees

11

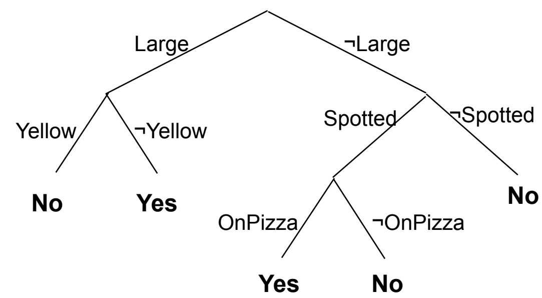
Decision Trees

- Classify attribute vectors into two or more classes



- Which boolean functions can we learn?

Edible Mushrooms?



(Large ∧ ¬Yellow) ∨ (¬Large ∧ Spotted ∧ OnPizza)

from Ginsberg, *Essentials of AI*

Building Decision Trees

- Given set of examples, derive consistent decision tree
- Idea: just include path for each positive example
  - What's wrong with this?
  - How can we do better?

Ockham's Razor

> *"Pluralitas non est ponenda sine neccesitate"*
> —William of Ockham, 14th century

- Plurality should not be posited without necessity

- Prefer the simplest consistent hypothesis
- Allows for generalization

Building Decision Tree

- Bad news
  - Finding smallest possible tree intractable
- Greedy approach
  - Starting from root (containing all examples)
  - Until stuck:
    - Pick a node in which not all examples are the same
      - (And at least one attribute is left)
    - Pick feature most effective in distinguishing among examples
    - Split node using feature.

Mushroom Instances

Pattern	Size	Color	OnPizza	Edible
Spotted	Large	Yellow	Yes	NO
Spotted	Large	Yellow	No	NO
Spotted	Large	NY	Yes	YES
Spotted	Large	NY	No	YES
Spotted	Small	Yellow	Yes	YES
Spotted	Small	Yellow	No	NO
Spotted	Small	NY	Yes	YES
Spotted	Small	NY	No	NO
No Spots	Large	Yellow	Yes	NO
No Spots	Large	Yellow	No	NO
No Spots	Large	NY	Yes	YES
No Spots	Large	NY	No	YES
No Spots	Small	Yellow	Yes	NO
No Spots	Small	Yellow	No	NO
No Spots	Small	NY	Yes	NO
No Spots	Small	NY	No	NO

Decision Tree Learning Algorithm

function DTL(*examples,attrs,default*) returns a *decision tree*

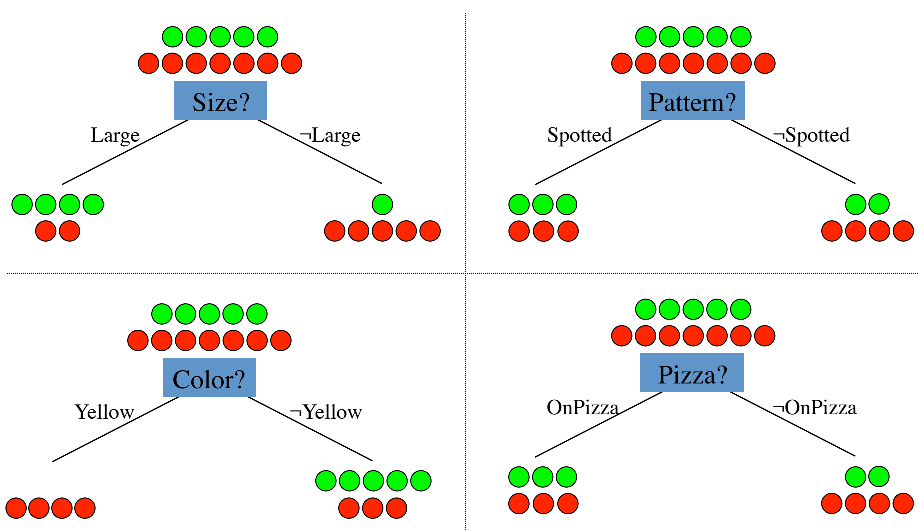
  if *examples* is empty **then**
    return *default*
  else
    if all *examples* have same classfcn **then**
      return classfcn
  else
    if *attrs* is empty **then**
      return Majority(*examples*)
  else
    best ← Choose-Attribute(*attrs,examples*)
    tree ← a new decision tree with root *best*
    for each value v_i of *best* **do**
      $examples_i$ ← {elements of *examples* with *best* = v_i}
      subtree ← DTL($examples_i$,*attrs–best*,Majority(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
  return *tree*

Key implementation detail

Choose-Attribute

- Best case?
  - Attribute fully resolves classification
- Worst case?
  - Attribute isn't correlated with classification
- Information gain
  - Measures discrimination value of attribute
  - Based on information-theoretic characterization of remaining uncertainty

Choose-Attribute in DTL



Measuring Information Value

- Consider binary event with probability *p*.
- How much information do we get from the outcome?
  - *p* = 1 or 0. Already knew it, **no new information**.
  - *p* = 1/2. Maximal information from event: **1 bit**.

$$I(p) = \log_2 \frac{1}{p(x)}$$

$$H(x) = E\left[\log_2 \frac{1}{p(x)}\right] = \sum_i p(x_i) \log_2 \frac{1}{p(x_i)}$$

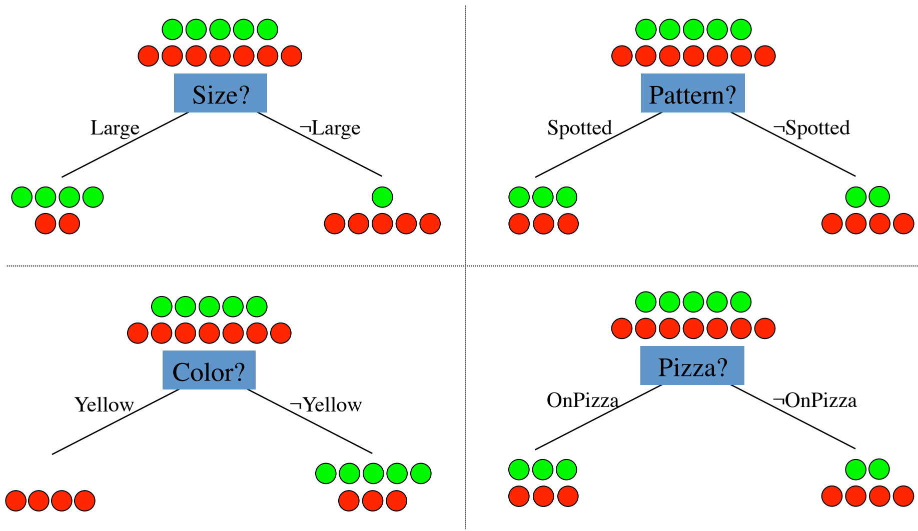
Information Gain

- Before observing attribute, suppose we have *p* positive examples, *n* negative.
  - H(x) = H(coin with prob p/(p+n)) -> lazy notation -> H(p/(p+n))
- After observing binary attribute, we have *four* categories
  - Attribute is true for pos/neg examples: p_t, n_t
  - Attribute if false for pos/neg examples: p_f, n_f

$$H(x|z) = \frac{p_t + n_t}{p+n} H\left(\frac{p_t}{p_t + n_t}\right) + \frac{p_f + n_f}{p+n} H\left(\frac{p_f}{p_f + n_f}\right)$$

- Strategy: pick an attribute that maximizes our information gain:
  - H(x) – H(x|z)

Choose-Attribute in DTL



Calculating Initial Information

Initially:

$I(5/12) = -5/12 \log_2 (5/12) - (7/12) \log_2 (7/12)$

$= -5/12(-1.263) - 7/12(-0.778)$

$= .980$

Fair amount of uncertainty!

Attribute Information Calculations

After observing "Large" (remainder):
(6/12) H(4/6) + (6/12) H(1/6) = .784
So Gain(Large) = .980 – .784 = .196

After observing "Spotted" (remainder):
(6/12) H(3/6) + (6/12) H(2/6) = .959
So Gain(Spotted) = .980 – .959 = .021

After observing "Yellow" (remainder):
(4/12) H(0) + (8/12) H(5/8) = .636
So Gain(Yellow) = .980 - .636 = .354

After observing "OnPizza" (remainder):
Same as Spotted.

So, split on Yellow: positive = NO, negative is 8 cases.

Remaining Mushroom Instances

Pattern	Size	Color	OnPizza	Edible
Spotted	Large	Yellow	Yes	NO
Spotted	Large	Yellow	No	NO
Spotted	Large	NY	Yes	YES
Spotted	Large	NY	No	YES
Spotted	Small	Yellow	Yes	YES
Spotted	Small	Yellow	No	NO
Spotted	Small	NY	Yes	YES
Spotted	Small	NY	No	NO
No Spots	Large	Yellow	Yes	NO
No Spots	Large	Yellow	No	NO
No Spots	Large	NY	Yes	YES
No Spots	Large	NY	No	YES
No Spots	Small	Yellow	Yes	NO
No Spots	Small	Yellow	No	NO
No Spots	Small	NY	Yes	NO
No Spots	Small	NY	No	NO

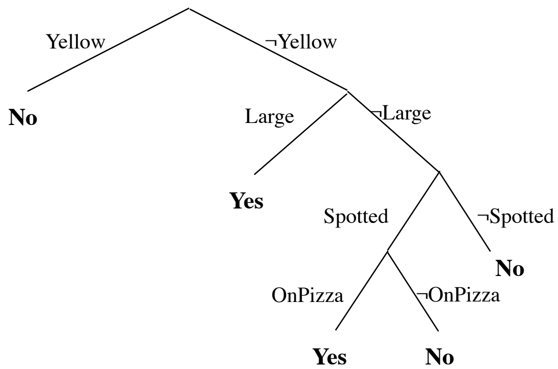
Measuring Information Value

Now, initially 5 positive and 3 negative examples, so:
  I(5/8) = .95443
After observing "Large" (remainder):
  (4/8) I(0) + (4/8) I(1/4) = .40564
  So Gain(Large) = .95443 - .40564 = .54879
After observing "Spotted" (remainder):
  (4/8) I(3/4) + (4/8) I(2/4) = .90564
  So Gain(Spotted) = .95443 - .90564 = .04879
After observing "OnPizza" (remainder):
  Same as Spotted.
So, split on Large: positive = Yes, negative is 4 cases.

Measuring Information Value

Now, initially 1 positive and 3 negative examples, so:
  I(1/4) = .81128
After observing "Spotted" (remainder):
  (2/4) I(1/2) + (2/4) I(0) = .5
  So Gain(Spotted) = .81128 - .5 = .31128
After observing "OnPizza" (remainder):
  Same as Spotted.
So, arbitrarily split on Spotted: positive = 2 cases, negative is No.

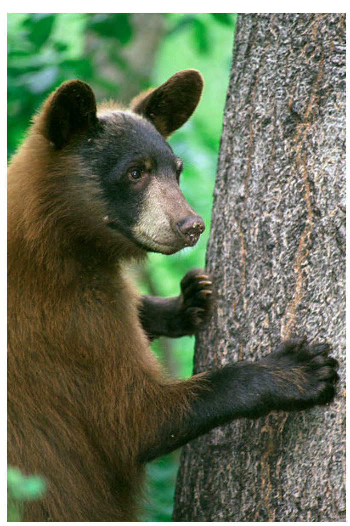
Induced Tree



$(\neg \text{Yellow} \wedge \text{Large}) \vee (\neg \text{Yellow} \wedge \neg \text{Large} \wedge \text{Spotted} \wedge \text{OnPizza})$

Boosting

30



Boosting

- Combine predictions from multiple hypotheses
  - May be produced by different learning algorithms
  - Or variations of same algorithm
- To the extent errors are *independent*, hypotheses are complementary
- Combination more likely to be right than any individual hypothesis

Simple Majority Voting

- Build M simple classifiers (e.g. M=5)
  - Suppose (optimistically) that each has an error rate P.
  - Ensemble is wrong only when three or more classifiers are wrong:
    - $P_M = (5C3)\ P^3\ (1\text{-}P)^2 + (5C4)\ P^4\ (1\text{-}P) + P^5$
  - Suppose P = 0.1. Estimate P_M.
- Why is independence assumption optimistic?

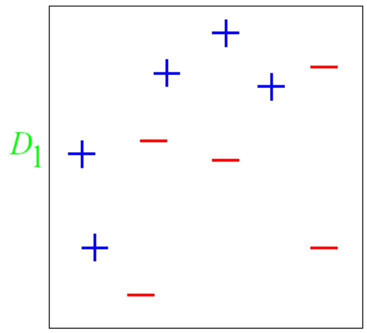
Boosting

- Requires: learning method operating over weighted training set.
  - Method attempts to minimize weighted error
  - E.g., decision stumps: decision trees with only one attribute test
- Approach
  - Modify weights over time to reward good performance over "difficult" instances
  - Combine hypotheses derived in each iteration

Boosting Algorithm

- $W(x)$ is the distribution of weights over the N training instances $\sum W(x_i)=1$
- Initially assign uniform weights $W_0(x) = 1/N$ for all x, step k=0
- At each iteration k :
  - Find hypothesis $H_k(x)$ with minimum error ε_k using weights $W_k(x)$
  - Compute $a_k = \frac{1}{2} \log \frac{1 - e_k}{e_k}$ ***What is the behavior of a_k?***
  - Update weights of every training example
    - Correctly labeled points: $W_{k+1} = W_k * \exp(-a_k)$
    - Incorrectly labeled points: $W_{k+1} = W_k * \exp(a_k)$
- $H_{FINAL}(x)$ =sign $[\sum \alpha_i H_i(x)]$

AdaBoost (Example)

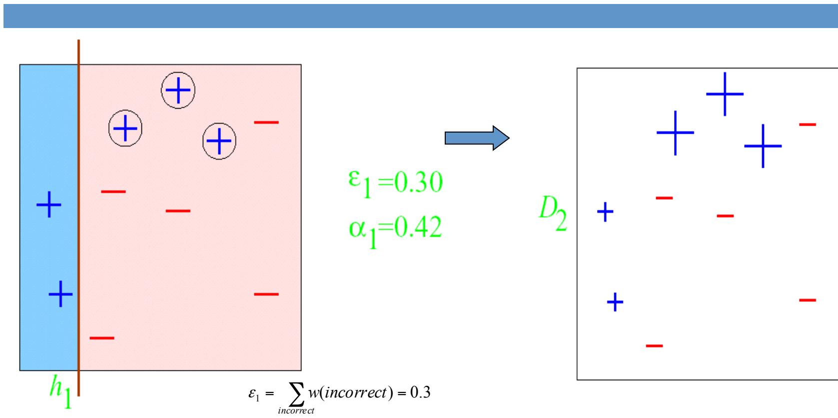


Can you find a reasonable decision stump?

Original Training set : Equal weights for all training samples

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

AdaBoost (Example) ROUND 1



$$\varepsilon_1 = \sum_{incorrect} w(incorrect) = 0.3$$

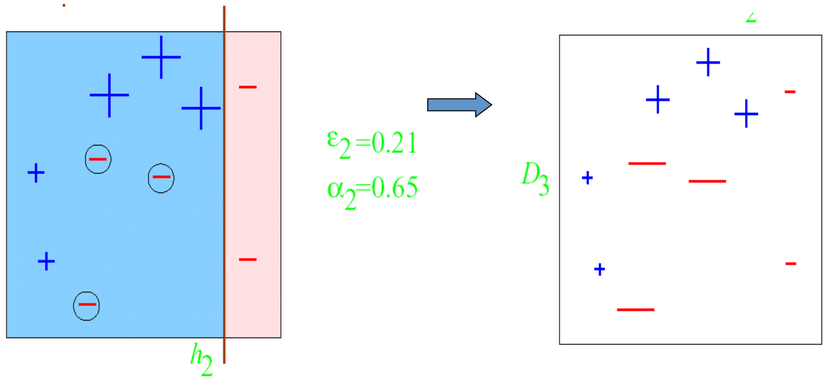
$$\alpha_1 = \frac{1}{2}\ln(\frac{1-\varepsilon_1}{\varepsilon_1}) = \frac{1}{2}\ln(\frac{.7}{.3}) = .42$$

$$w'_{correct} = C_N w_{correct} e^{-\alpha_1} = (1.091).1(.654653) = .0714$$

$$w'_{incorrect} = C_N w_{incorrect} e^{\alpha_1} = (1.091).1(1.527525) = .1667$$

$$C_{Normalization} = 1/\sum_{examples} w' = 1.091$$

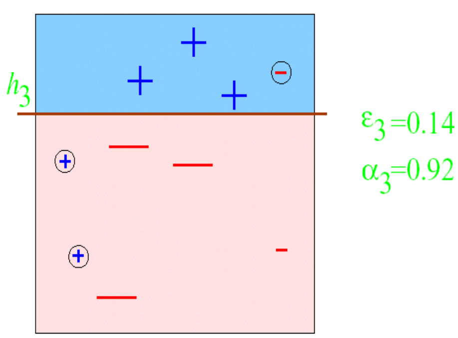
AdaBoost (Example) ROUND 2



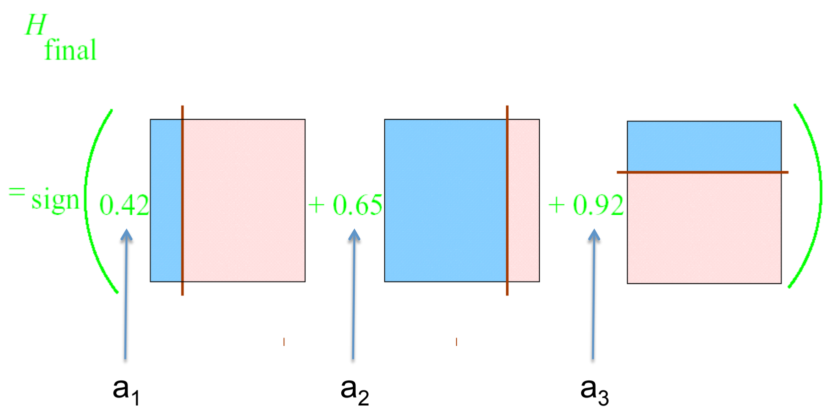
$$\varepsilon_2 = \sum_{incorrect} w(incorrect) = 0.21$$

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_2}{\varepsilon_2}\right) = \frac{1}{2}\ln\left(\frac{.79}{.21}\right) = .65$$

AdaBoost (Example) ROUND 3



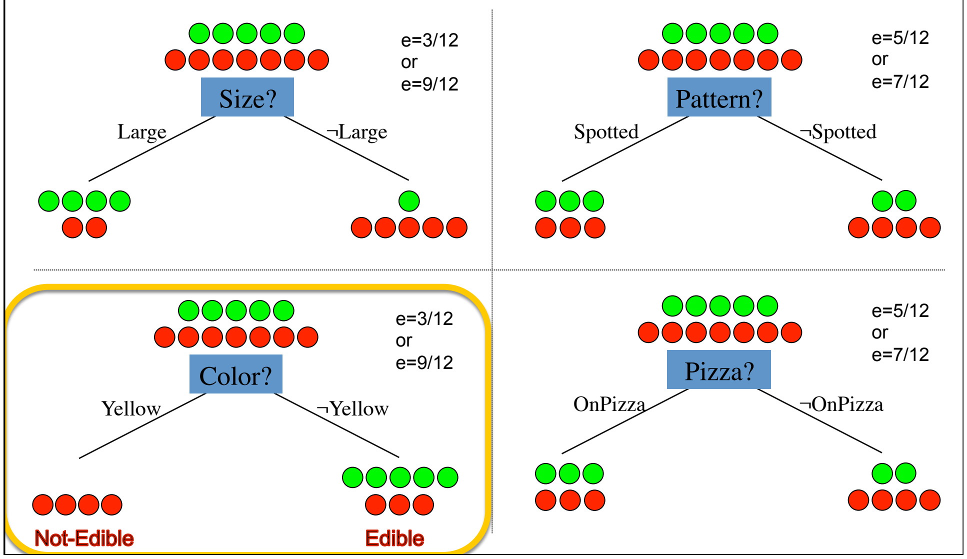
AdaBoost (Example)



Mushroom Instances

Pattern	Size	Color	OnPizza	Edible
S	L	Y	Y	No
S	L	N	Y	Yes
S	L	N	N	Yes
S	S	Y	N	No
S	S	N	Y	Yes
S	S	N	N	No
N	L	Y	N	No
N	L	N	Y	Yes
N	L	N	N	Yes
N	S	Y	Y	No
N	S	N	Y	No
N	S	N	N	No

Boosting on Features



Computing Weighting

Hypothesis is: Yellow=Not edible, ~Yellow=Edible

$\varepsilon 1 = \sum w(\text{incorrect}) = 1/12 + 1/12 + 1/12 = ¼$

$\alpha 1 = ½ \ln(3/1) = .55$

w'(correct) = Cn(1/12)($e^{-.55}$) = (.048)Cn

w'(incorrect) = Cn(1/12($e^{.55}$) = (.144)Cn

Cn normalizes so it is 1.1574

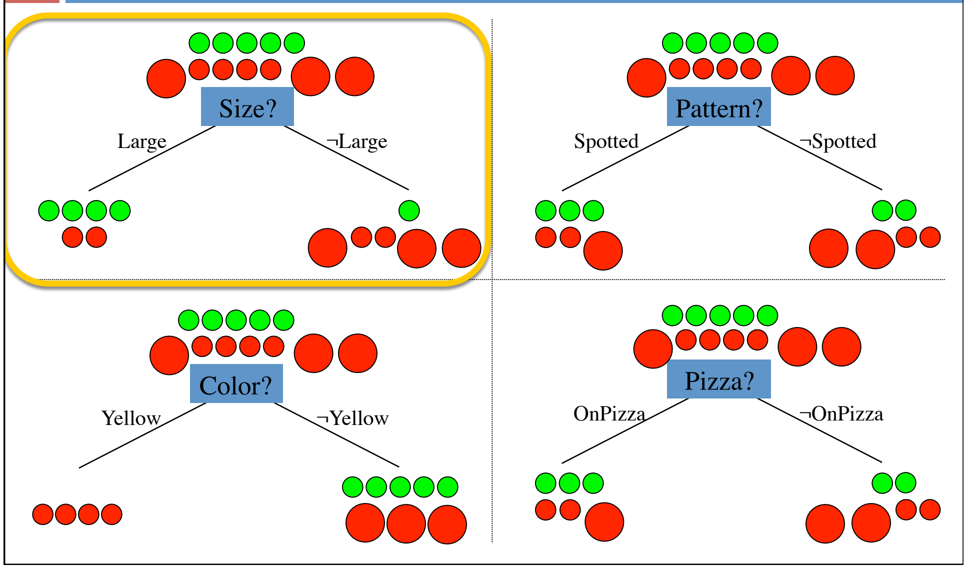
w'(correct) = .0555

w'(incorrect) = .1666

Mushroom Instances

Pattern	Size	Color	OnPizza	Edible	
S	L	Y	Y	No	.0555
S	L	N	Y	Yes	.0555
S	L	N	N	Yes	.0555
S	S	Y	N	No	.0555
S	S	N	Y	Yes	.0555
S	S	N	N	No	.1666
N	L	Y	N	No	.0555
N	L	N	Y	Yes	.0555
N	L	N	N	Yes	.0555
N	S	Y	Y	No	.0555
N	S	N	Y	No	.1666
N	S	N	N	No	.1666

Boosting on Features (step 2)



Computing Weighting

Hypothesis is: Large=Yes, ~Large=No

ε2 = ∑w(incorrect) = (2 * .0555) + (1 * .0555) = .1665

α2 = ½ ln(.8335/.1665) = .80

What does AdaBoost actually do?

46

- It's iteratively finding weights that minimize the exponential loss function [Collins 2002]

$$\sum_i e^{-y_i f_\lambda(x_i)}$$

where $f_\lambda(x) = \sum_t a_t h_t(x)$

- (Now those exponential re-weightings make a bit more sense!)
- Is that what we want?

Neural Network

47

Neural Networks

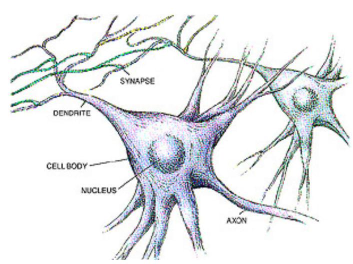
- A good world model often has several interacting processes
  - Bayes nets, for example
    - Inputs = Earthquake, Burglary
    - Outputs = John/Mary calls
  - Hidden nodes moderate influence between other nodes
    - Alarm
- Conceptual idea: perhaps hidden nodes are there, even if we don't know what they are
  - Can we assume the presence of hidden nodes and learn their behavior

Brain Inspiration

- It is hard to make a machine behave intelligently
- Approach: reverse engineering!
- Problem: we don't really know all about how brains work, either

Neurons

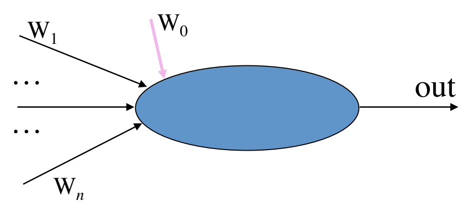
- Brains are made out of neurons.
- Lots of them (~10^{11})
  - Highly connected
  - Really slow (~1ms)



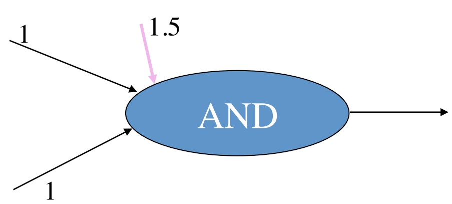
- Cartoon version
  - Neuron "fires" along axon given sufficient signal from dendrites

McCulloch-Pitts Model

- (1943) Neuron as threshold unit
- Output is one iff weighted sum of inputs exceeds threshold

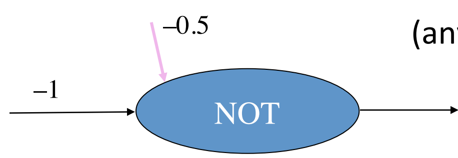


Representing Logical Fns

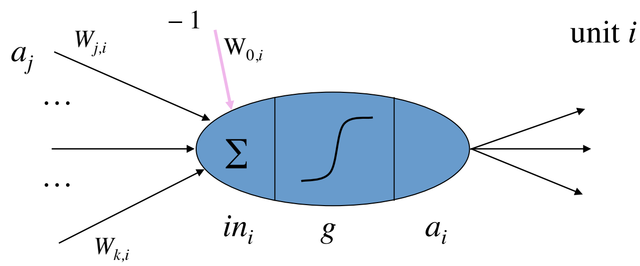


with AND and NOT, can represent *any* combinational circuit

(any boolean function).



Slightly Generalized Model

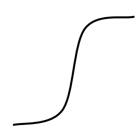


- $in_i = \Sigma_j W_{j,i} a_j$ — input fn
- g — activation fn
- $a_i = g(in_i)$ — output

Activation Functions

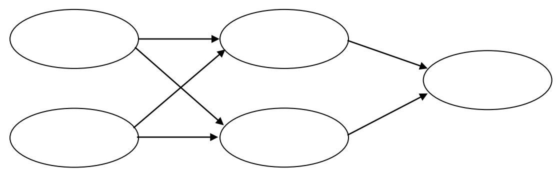
- Step function
  - g(x) = 1 iff x > 0, else 0.

- Sigmoid
  - $g(x) = 1/(1 + e^{-x})$



Neural Networks

- Collection of units, connected together



Recurrent: cycles allowed
Feedforward: no cycles
  Layered: can partition into strata

Perceptrons

- (Rosenblatt, 1950s)
- Set of units in a single feedforward layer
  - (inputs connected directly to outputs)

$$out = Step_0\left(\sum_j W_j x_j\right) = Step_0(\mathbf{W} \cdot \mathbf{x})$$

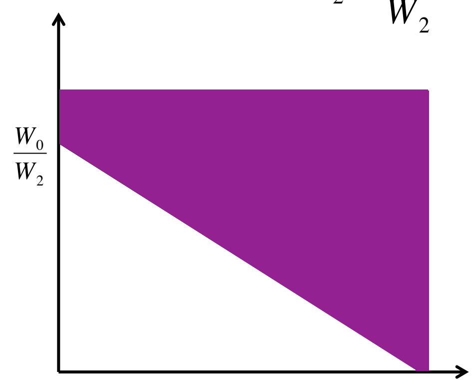
Output is 1 iff: $\mathbf{W} \cdot \mathbf{x} \geq 0$
For two inputs:

$$W_1 x_1 + W_2 x_2 \geq W_0$$

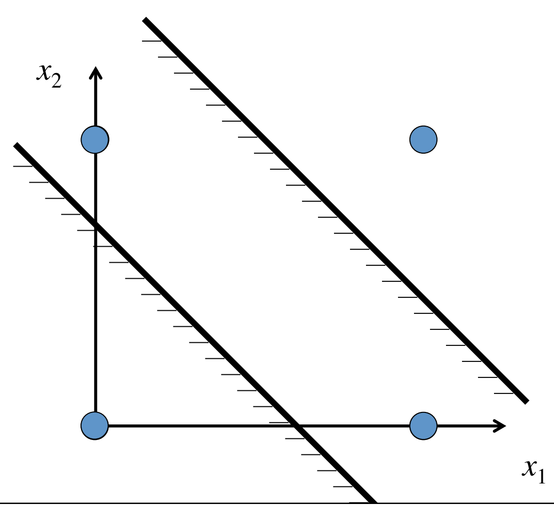
$$x_2 \geq \frac{W_0}{W_2} - \frac{W_1}{W_2} x_1$$

Perceptron Boundaries

$$x_2 \geq \frac{W_0}{W_2} - \frac{W_1}{W_2} x_1$$



Linear Separability



x_1 AND x_2

x_1 OR x_2

x_1 IFF x_2

Perceptron Limitations

- Can't learn functions that aren't linearly separable
- But, we *can* learn some "hard" functions easily!

Perceptron Learning

- Suppose we have weights $\mathbf{w}$
- Observe $\mathbf{x}_i$, y_i
- What is the error?

  $e = y_i - g(\mathbf{w} \cdot \mathbf{x}_i)$
- Squared error:

  $e^2 = (y_i - g(\mathbf{w} \cdot \mathbf{x}_i))^2$

Perceptron Learning

- Squared error: $e^2 = (y_i - g(\mathbf{w} \cdot \mathbf{x}_i))^2$
- How do we minimize the squared error?
  - We can adjust **w**'s:
  - $de^2/dw_i =$
- Adjusting w_i in the *opposite* direction will reduce e^2

  $$w_j' = w_j - \delta e^2/\delta w_j \quad (????)$$

- How big a step should we take?

Perceptron Learning

- How big a step should we take?
  - Could we compute how big a step would reduce the error to zero?
  - Do we really want to fit *this* training example?
- Learning rate: α

  $$w_j' = w_j + \alpha\, \delta e^2/\delta w_j$$

- Yes, but what should α be?

Learning Rate

- What should α be?
  - Hard to pick… must tune.
- Stochastic Gradient Descent
  - Learning rate *schedule*
  - Fancier strategies, e.g. search then converge

Perceptron Learning Wrap-Up

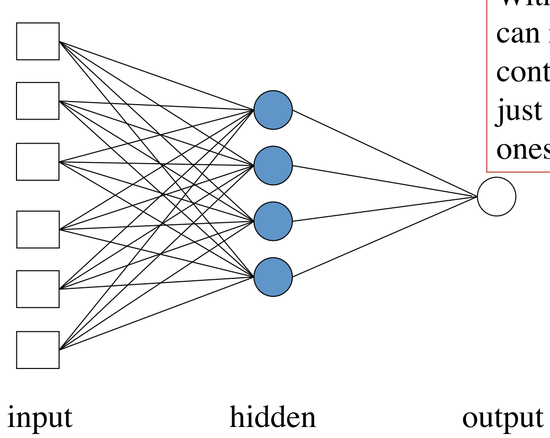
- Repeat
  - Pick an example x_i, y_i
  - Compute error: $e = y_i - g(\mathbf{w} \cdot \mathbf{x}_i)$
  - For each input j:

    $w_j' = w_j + \alpha\, \delta e^2 / \delta w_j$
- Hill Climbing – iterative improvement
  - Given small enough a, it will converge.
- A bit of terminology:
  - Epoch: do an update for every example

Limitations

- Many (most?) interesting functions not linearly separable
  - From late 1960s, interest in perceptrons waned
- Can get around expressive limitations with multilayer networks

Multilayer Networks



With enough hidden units, can represent *any* continuous function, not just linearly separable ones.

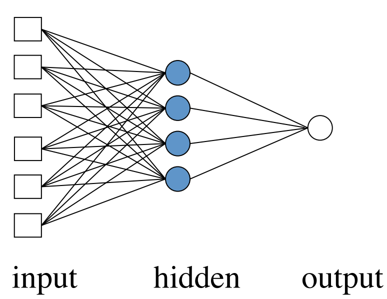
Learning Multilayer Networks

- More difficult, because we do not know what hidden units should represent.
- Multiple weights between every input and output.
- Credit (blame) assignment problem.
- (Re)discovery of backpropagation in 1980s led to resurgent interest in neural networks.

Back-propagation

68

- Basic idea:
  - Compute effect of every weight on output.
    - Work backwards from output to input
    - Similar to chain rule.
  - If output is wrong value, move weights in –gradient direction.



Backpropagation Updates

- For output unit

$$W_{j,i} \leftarrow W_{j,i} + \alpha\, a_j\, Err_i\, g'(in_i) = W_{j,i} + \alpha\, a_j \Delta_i$$

- For hidden units
  - need way to take share of blame for output error among its successors
  - make it proportional to weight

$$\Delta_j = Err_j\, g'(in_j) = g'(in_j) \sum_i W_{j,i} \Delta_i$$

$$W_{k,j} \leftarrow W_{k,j} + \alpha\, a_k \Delta_j$$

Backpropagation

For each example:

- Forward pass
  - Compute activation level for each unit
- Backward pass
  - Compute error and Δ values for output layer
  - Update weights to output layer, pass back Δ values to previous layer
  - For each node in previous layer, use Δ values from succeeding layer to compute Δ values for itself, update incoming weights, pass back Δ values to its preceding layer…

Backpropagation Analysis

- A form of hill-climbing (gradient descent), just like perceptron algorithm
- No convergence guarantees,
  - due to local minima
  - ridges also slow convergence
- General problem of finding consistent weights is NP-complete
- Performance dependent on network structure
  - Need sufficient hidden nodes to express target
  - Too many leads to overfitting, slow training

Neural Networks

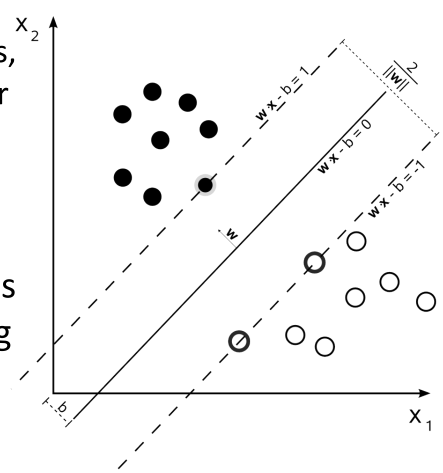
- Appealing due to brain analogy
- Other advantages
  - Simplicity, expressiveness,
  - Ability to handle noise
- Disadvantages
  - Opaque: cannot be used in some applications due to regulatory constraints!
  - Black art of designing structures and tuning parameters
- Ultimately, one of many forms of nonlinear regression

SVMs

73

Support Vector Machines

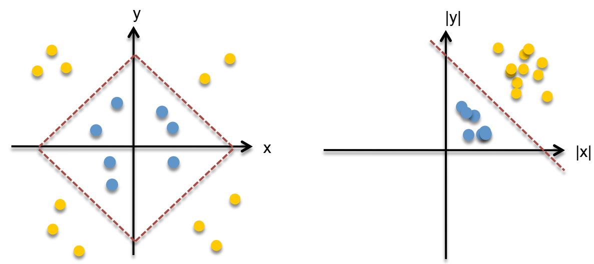
- All about separability:
  - Given a bunch of features, find the (linear) separator that maximizes the *margin*.
- This can be formulated as a quadratic programming problem



SVMs: Features

75

- The key is to find features that make the data linearly separable



- When viewed from the original space, these features can be complex looking.

SVMs: Kernel Trick

76

- Where do we get the "right" features?
  - In higher dimensions, data tends to become linearly separable, even if the features aren't particularly clever.
- Idea: generate features *from our data*
  - E.g., compute the dot product of every point x_i with respect to x_{17}
  - In fact, let's make *every* point its own feature
- Linear separators can be efficiently computed for features of this form
  - "Kernel Trick"
  - We won't worry about mechanics

Next Time

77

- Learning Theory
  - Why does any of this work?
- Statistical Learning

Review questions

78