Learning

BBM 405– Fundamentals of Artificial Intelligence

Pinar Duygulu

Hacettepe University

Slides are mostly adapted from AIMA and MIT Open Courseware

Learning

- It is often hard to articulate the knowledge we need to build AI systems
- Often, we don't even know it!
- Frequently, we can arrange to build systems that learn it themselves

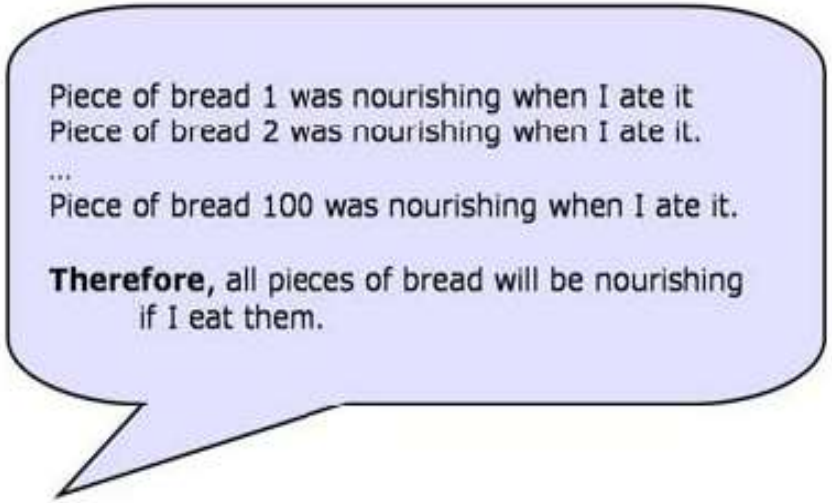
What is learning?

- memorizing something
- learning facts through observation and exploration
- improving motor and/or cognitive skills through practice
- organizing new knowledge into general, effective representations

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the task or tasks drawn from the same population more efficiently and more effectively the next time." -- Herb Simon

Induction

David Hume



Bertrand Russell

If asked why we believe the sun will rise tomorrow, we shall naturally answer, 'Because it has always risen every day.' We have a firm belief that it will rise in the future, because it has risen in the past.
The real question is: Do any number of cases of a law being fulfilled in the past afford evidence that it will be fulfilled in the future?
It has been argued that we have reason to know the future will resemble the past, because what was the future has constantly become the past, and has always been found to resemble the past, so that we really have experience of the future, namely of times which were formerly future, which we may call past futures. But such an argument really begs the very question at issue. We have experience of past futures, but not of future futures, and the question is: Will future futures resemble past futures?

http://www.ditext.com/russell/rus6.html

Kinds of Learning

- **Supervised learning**: given a set of example input/output pairs, find a rule that does a good job of predicting the output associated with a new input
- **Clustering**: given a set of examples, but no labeling of them, group the examples into "natural" clusters
- **Reinforcement** learning: an agent interacting with the world makes observations, takes actions, and is rewarded or punished; it should learn to choose actions in such a way as to obtain a lot of reward

Learning a function

Given a set of examples of input/output pairs, find a function that does a good job of expressing the relationship

- Pronunciation: function from letters to sounds
- Throw a ball: function from target locations to joint torques
- Read handwritten characters: function from collections of image pixels to letters
- Diagnose diseases: function from lab test results to disease categories

Aspects of function learning

- memory
- averaging
- generalization

Example Problem

When to drive the car? Depends on:

- temperature
- expected precipitation
- day of the week
- whether she needs to shop on the way home
- what she's wearing

Memory

temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
19	snow	mon	yes	casual	drive
65	none	tues	no	casual	walk
19	snow	mon	yes	casual	drive

Averaging

Dealing with noise in the data

temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk
80	none	sat	no	casual	drive
80	none	sat	no	casual	drive
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk

Sensor noise

Dealing with noise in the data

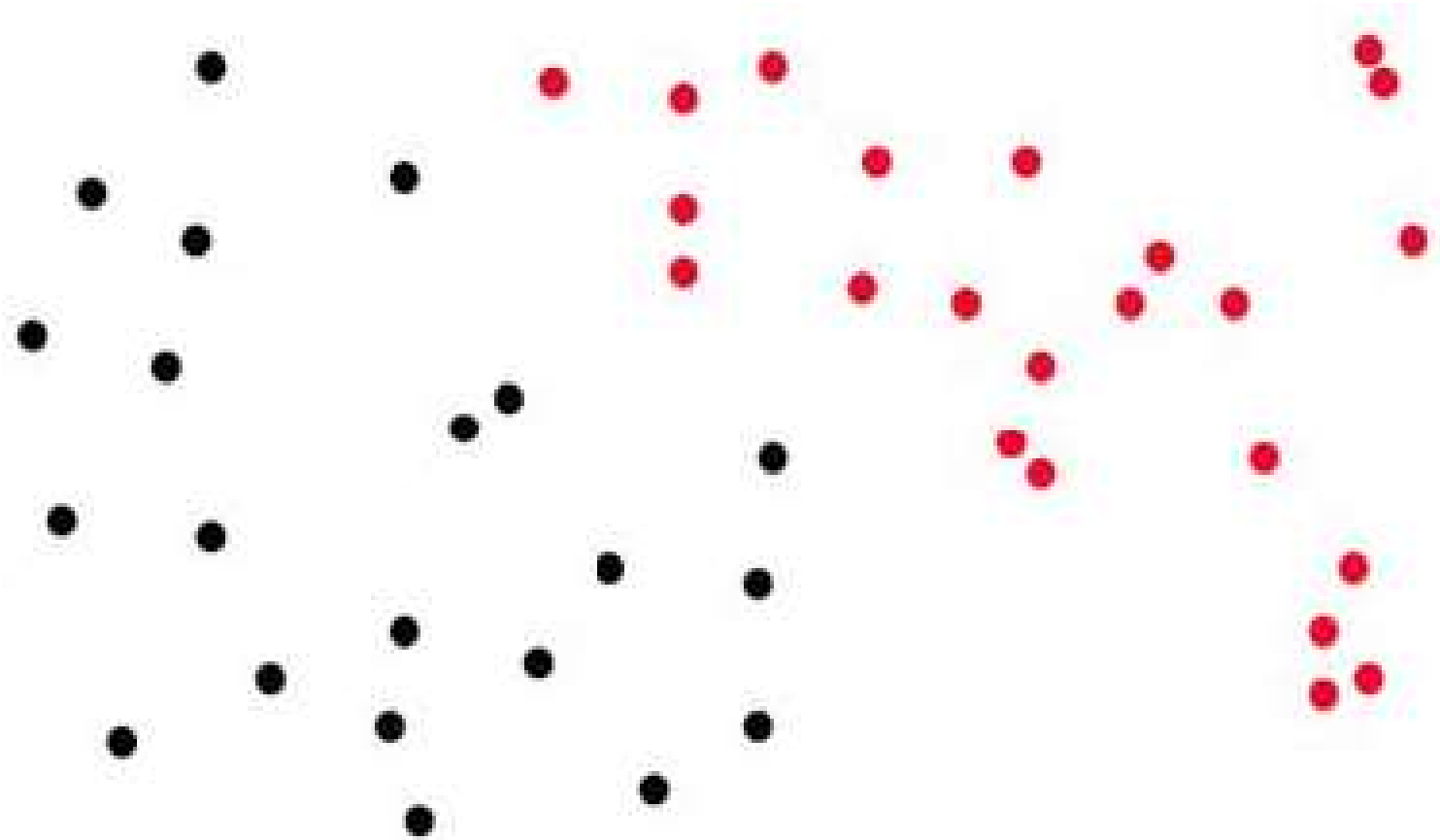
temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
82	none	sat	no	casual	walk
78	none	sat	no	casual	walk
21	none	sat	no	casual	drive
18	none	sat	no	casual	drive
19	none	sat	no	formal	drive
17	none	sat	no	casual	drive
20	none	sat	no	casual	drive

Generalization

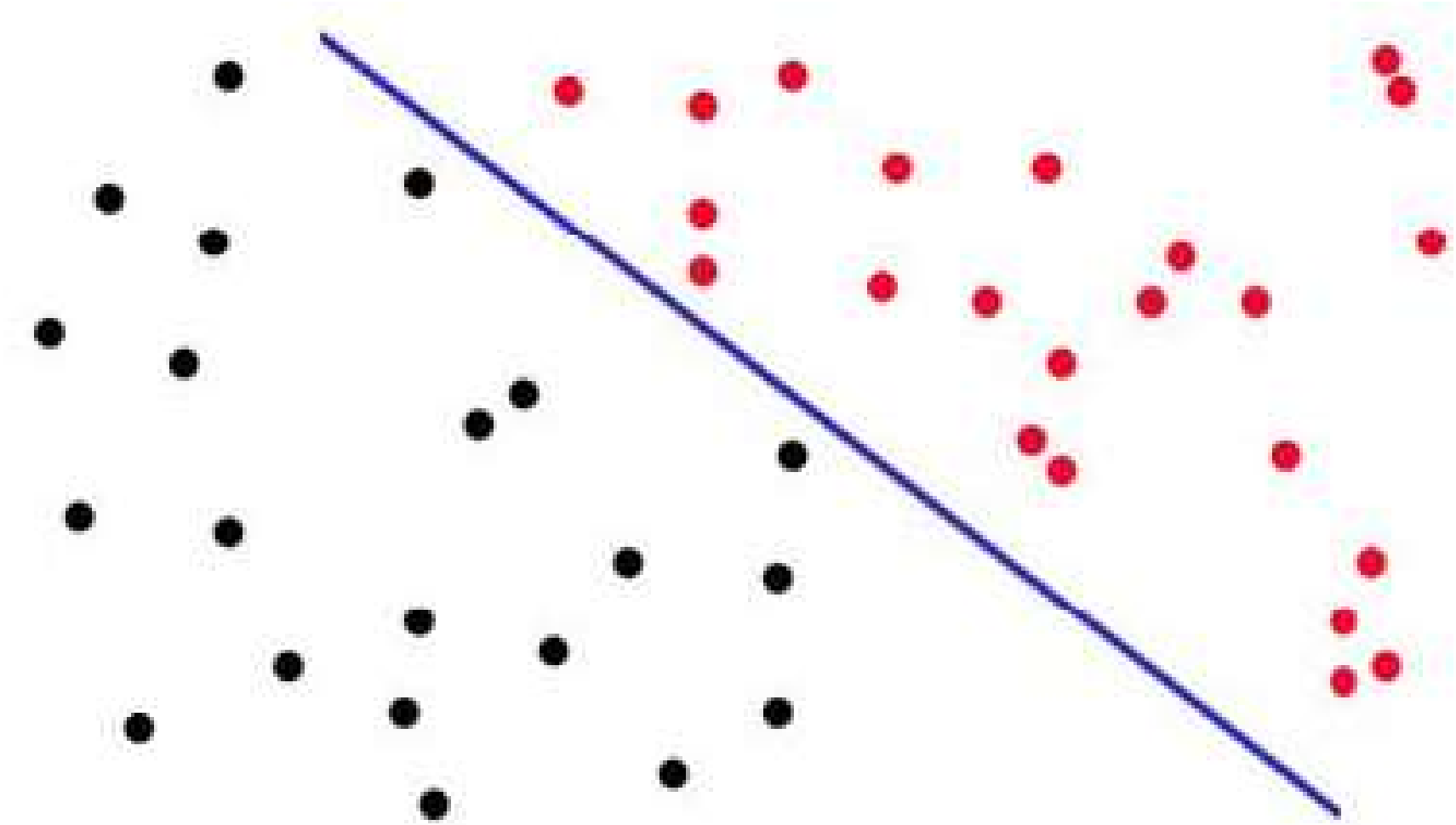
Dealing with previously unseen cases

temp	precip	day	shop	clothes	
71	none	fri	yes	formal	**drive**
36	none	sun	yes	casual	**walk**
62	rain	weds	no	casual	**walk**
93	none	mon	no	casual	**drive**
55	none	sat	no	formal	**drive**
80	none	sat	no	casual	**walk**
19	snow	mon	yes	casual	**drive**
65	none	tues	no	casual	**walk**
58	rain	mon	no	casual	

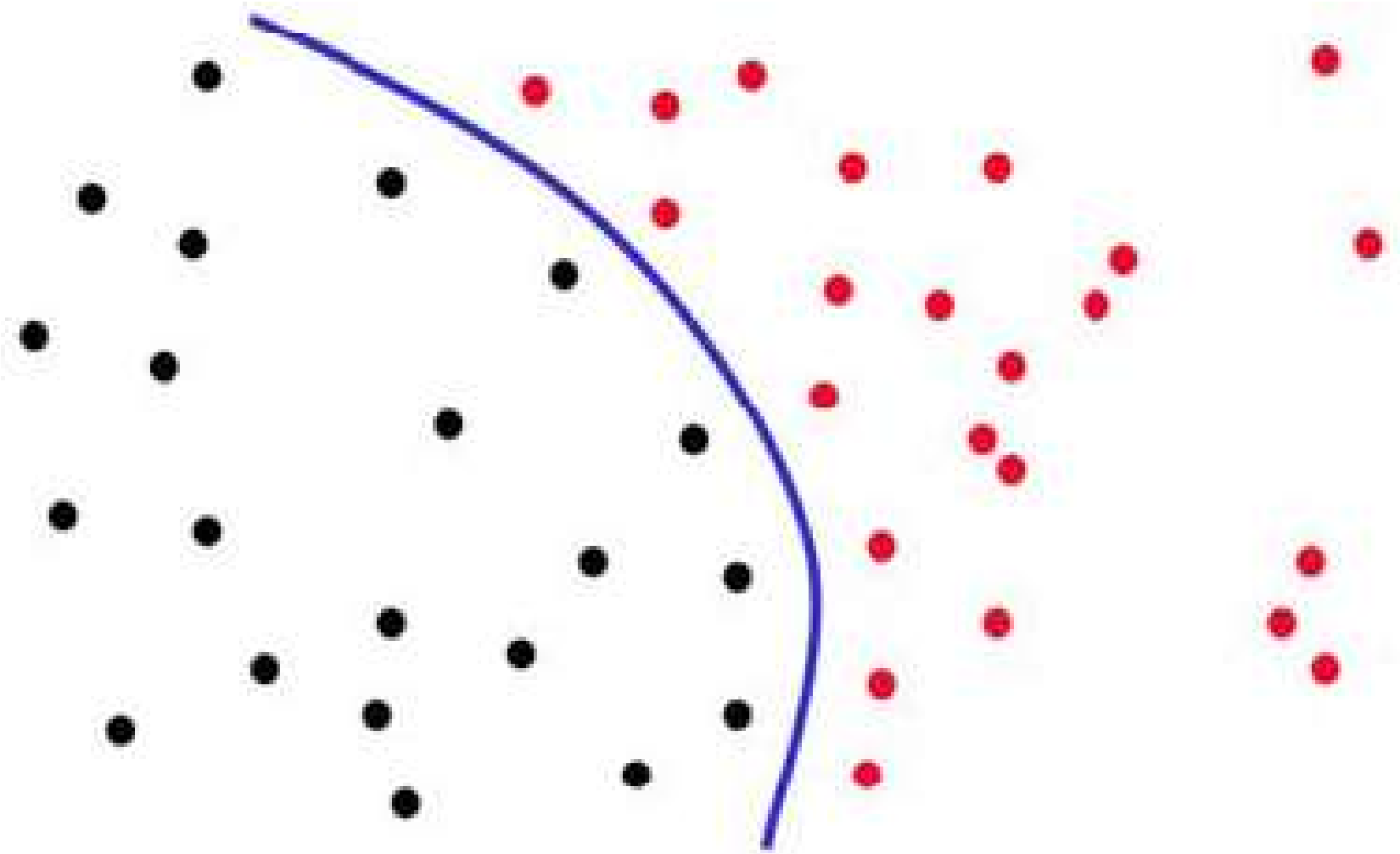
The red and the black



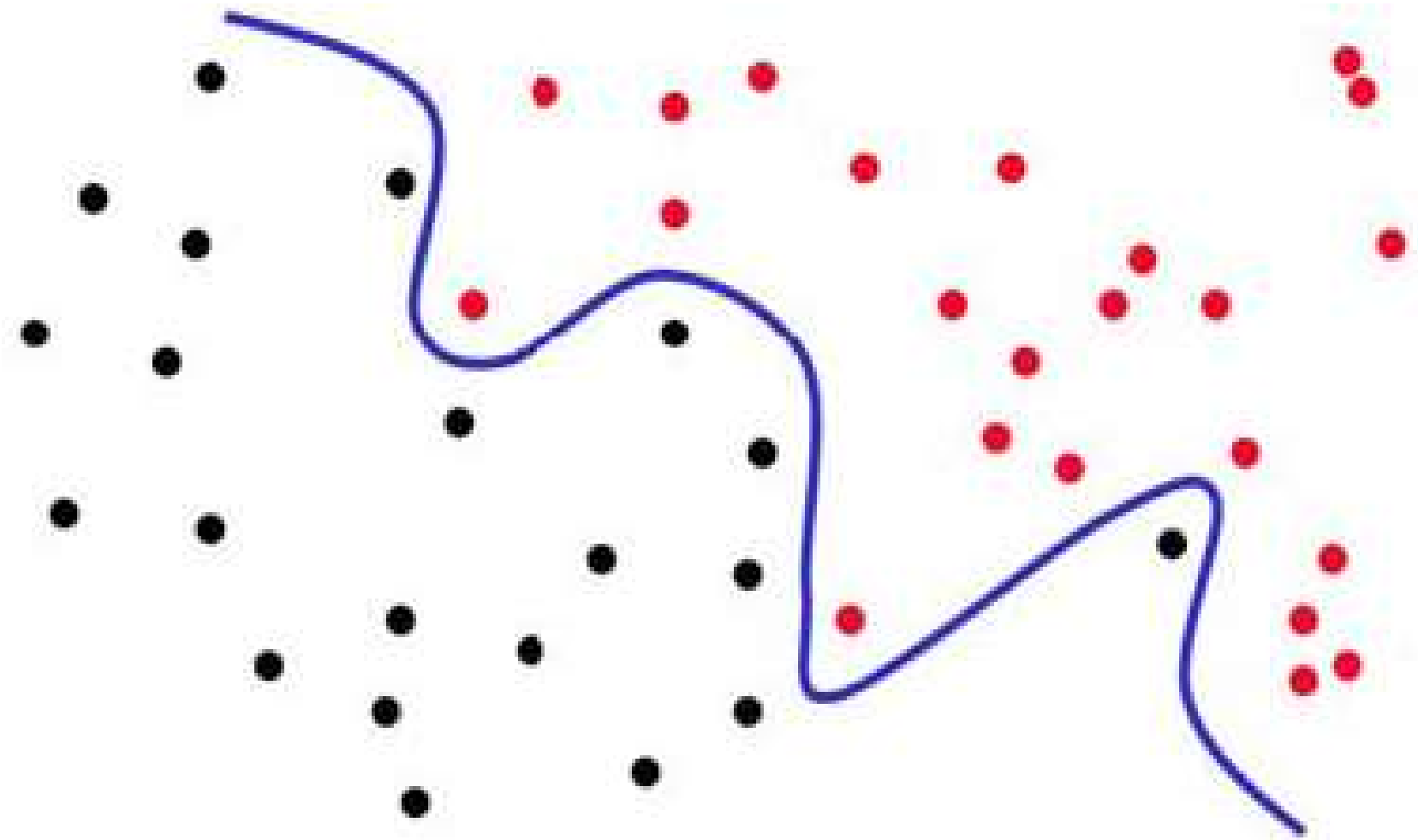
What is the right hypothesis?



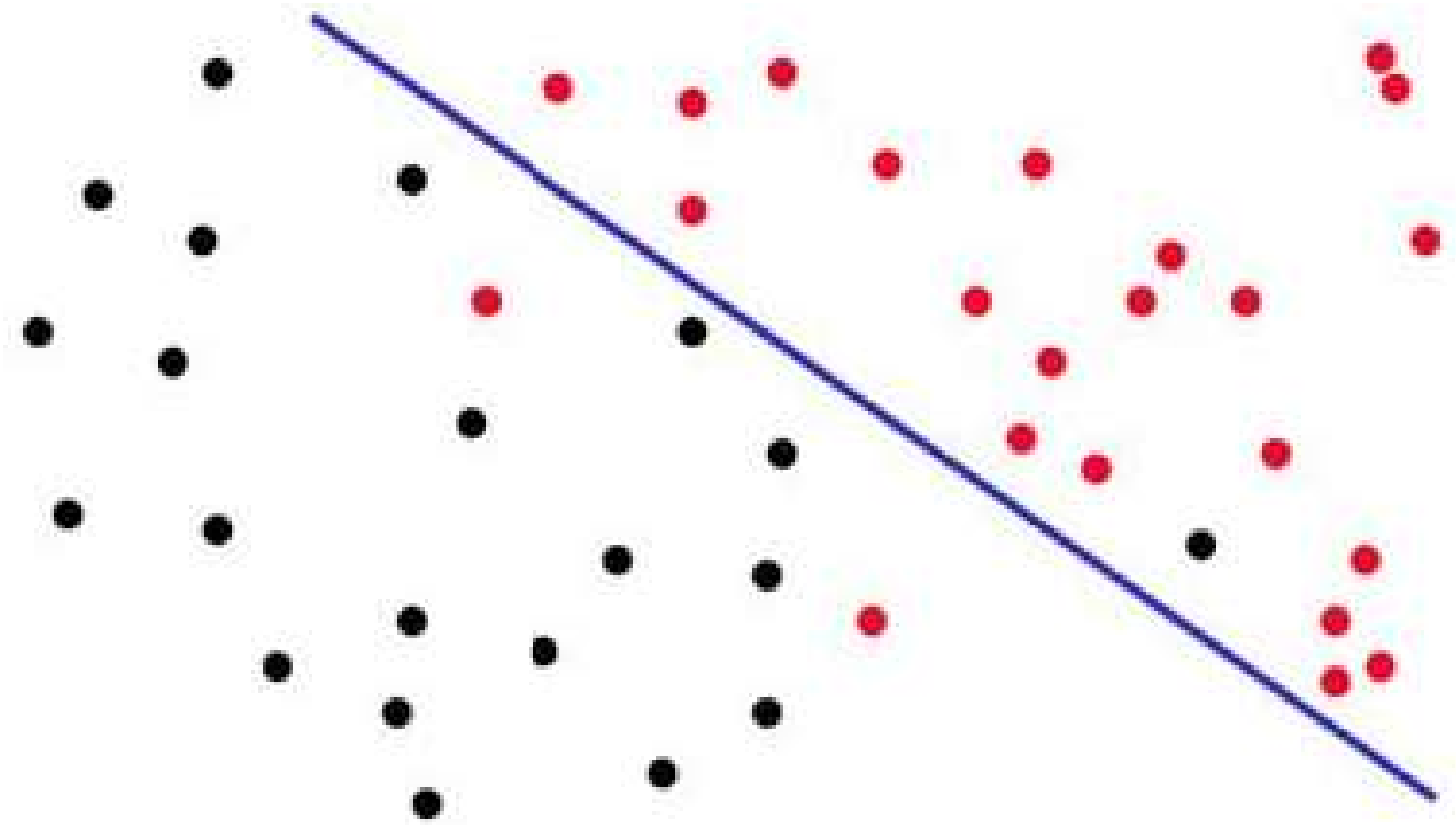
What is the right hypothesis?



What is the right hypothesis?



How about this?



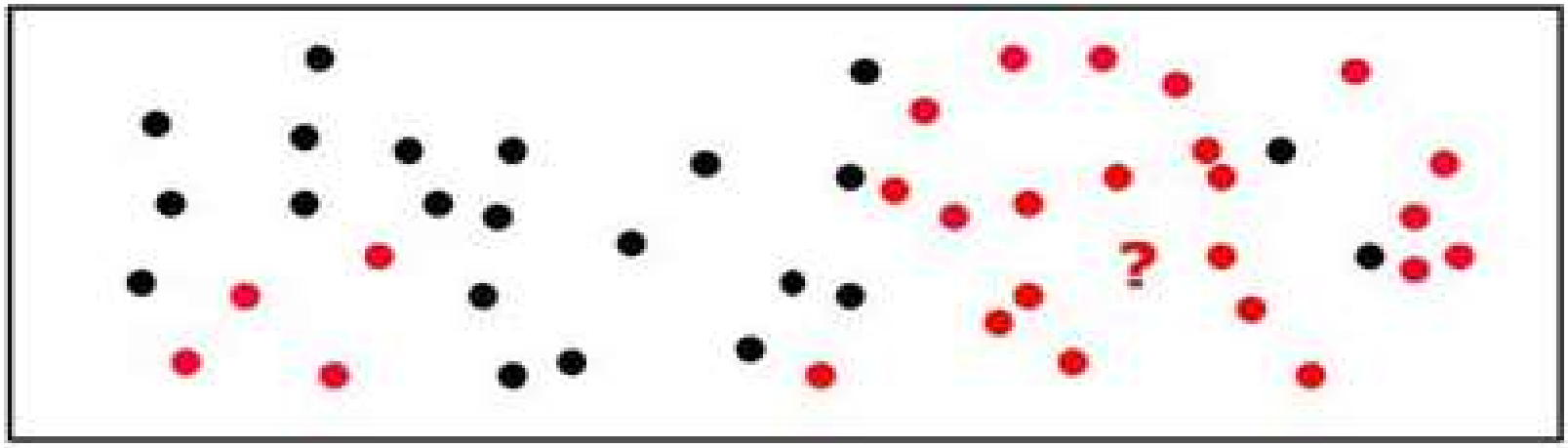
Variety of learning methods

Learning methods differ in terms of:

- the form of the hypothesis
- the way the computer finds a hypothesis given the data

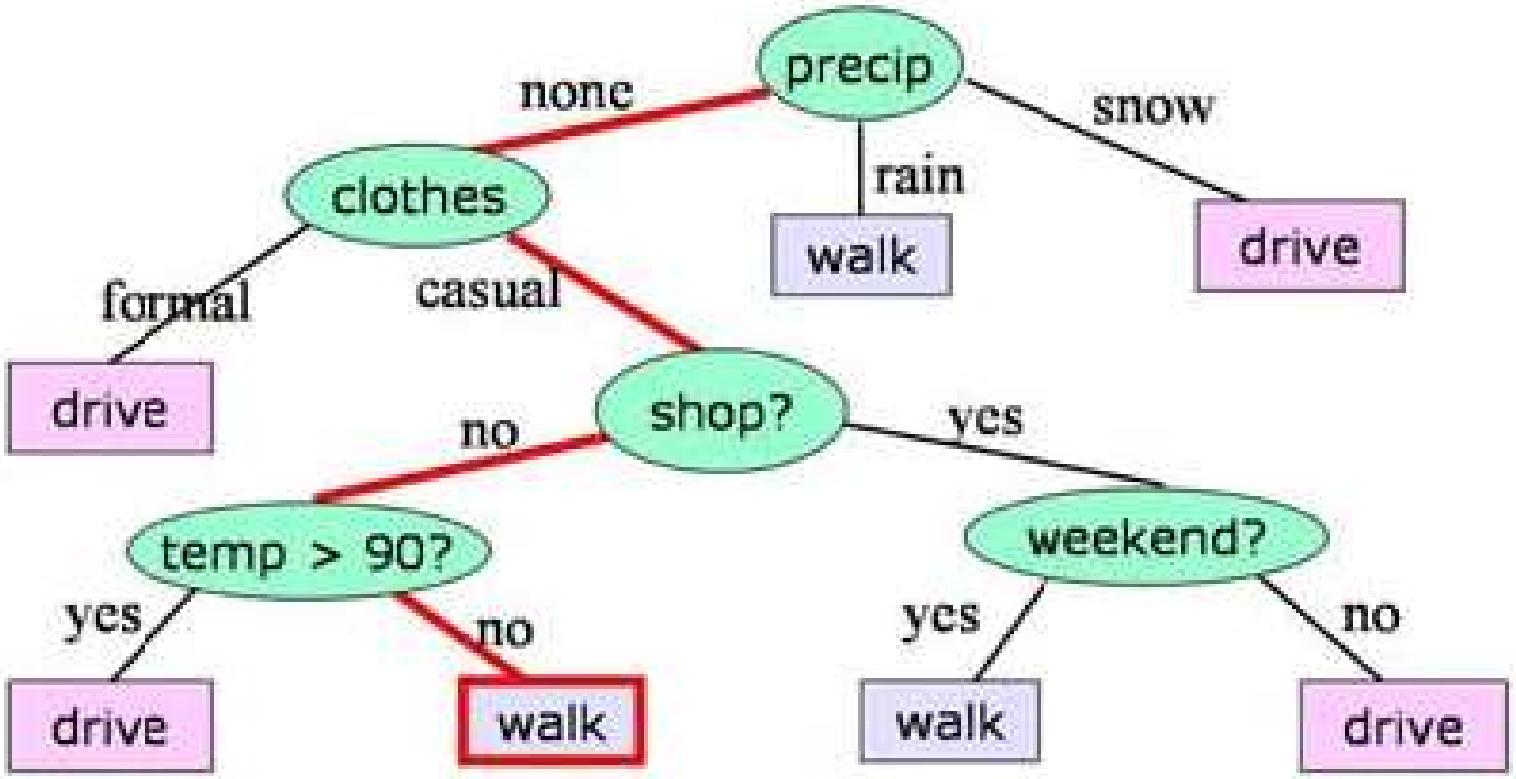
Nearest Neighbor

- Remember all your data
- When someone asks a question,
  - find the nearest old data point
  - return the answer associated with it



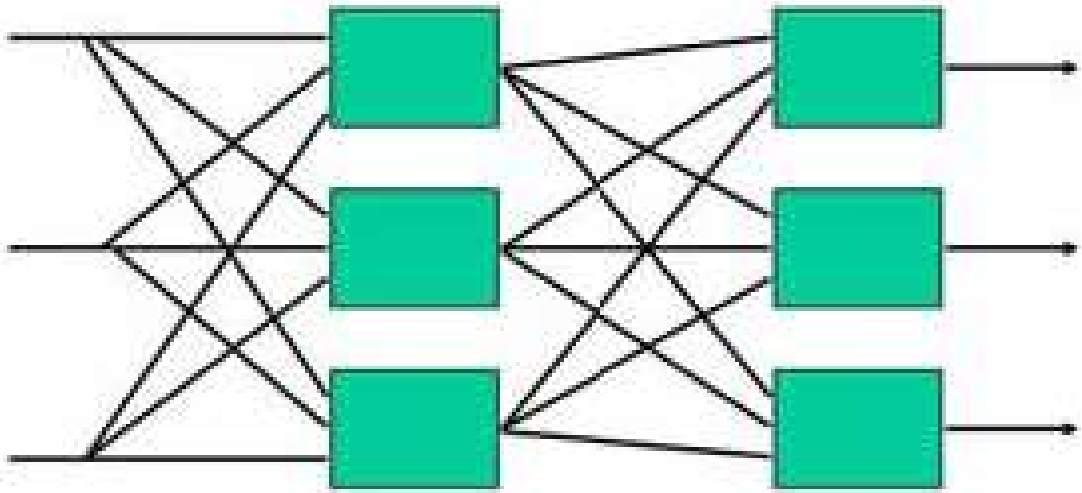
Decision trees

Use all the data to build a tree of questions with answers at the leaves



Neural Networks

- Represent hypotheses as combinations of simple computations
- Neurophysiologically plausible (sort of)



- Learning through weight adjustment

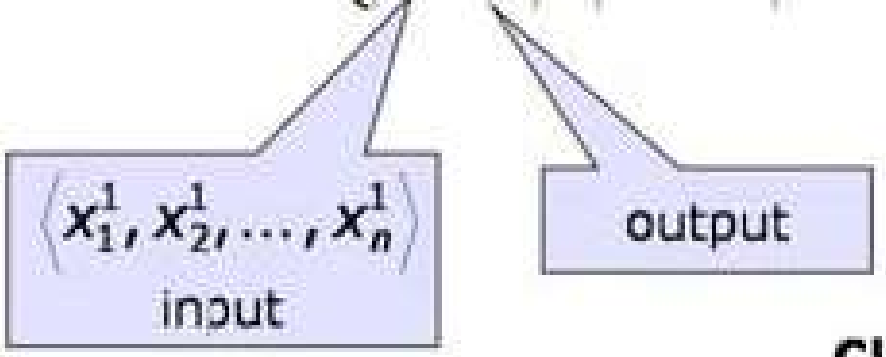
Machine learning successes

- assessing loan credit risk
- detecting credit card fraud
- cataloging astronomical images
- detecting and diagnosing manufacturing faults
- helping NBA coaches analyze performance
- personalizing news and web searches
- steering an autonomous car across the US

Supervised learning

- Given data (training set)

$$D = \left\{ \langle x^1, y^1 \rangle, \langle x^2, y^2 \rangle, \ldots \langle x^m, y^m \rangle \right\}$$



Classification: discrete Y

Regression: continuous Y

- Goal: find a hypothesis h in hypothesis class H that does a good job of mapping x to y

Best hypothesis

Hypothesis should

- do a good job of describing the data
  - ideally: $h(x^t) = y^t$
  - number of errors: $E(h,D)$

- not be too complex
  - measure: C(h)

Non sunt multiplicanda entia praeter necessitatem

William of Ockham

Minimize $E(h,D) + \alpha C(h)$

trade-off

Learning Conjunctions

- How can we find the hypothesis with the lowest value of $E(h, D) + \alpha C(h)$? Search!
- Boolean features and output
- H = conjunctions of features

f_1	f_2	f_3	f_4	y	
0	1	1	0	0	
1	0	1	1	1	
1	1	1	0	0	←
0	0	1	1	1	←
1	0	0	1	0	
0	1	1	1	1	←

$h = f_1 \wedge f_3$

$E(h, D) = 3$

$C(h) = 2$

- Set alpha so we're looking for smallest h with 0 error

Algorithm

- Could search in hypothesis space using tools we've already studied
- Instead, be greedy!
- Start with h = True
- All errors are on negative examples
- On each step, add conjunct that rules out most new negatives (without excluding positives)

Algorithm

```
N = negative examples in D
h = True
Loop until N is empty
   For every feature j that does not have value 0 on
     any positive examples
       n_j := number of examples in N
              for which f_j = 0
       j* := j for which n_j is maximized
       h := h ^ f_j*
       N := N - examples in N for which f_j = 0
   If no such feature found, fail
```

Start with N equal to all the negative examples and h = true
Then loop, adding conjuncts that rue out negative examples
until N is empty
Inside the loop consider every feature that would not rule out any positive examples

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N=\{x^1,x^3,x^5\}$, h = True

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N=\{x^1,x^3,x^5\}$, h = True
- $n_3 = 1$, $n_4 = 2$

Now, we consider all the the features that would not exclude any positive examples.
Those are features f3 and f4.
f3 would exclude 1 negative example;
f4 would exclude 2.
So we pick f4.

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N=\{x^1,x^3,x^5\}$, h = True
- $n_3 = 1$, $n_4 = 2$
- $N = \{x^5\}$, $h = f_4$
- $n_3 = 1$, $n_4 = 0$

Now we remove the examples from N that are ruled out by f4 and add f4 to h.

Now, based on the new N, n3 = 1 and n4 = 0. So we pick f3.

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N=\{x^1,x^3,x^5\}$, $h = True$
- $n_3 = 1$, $n_4 = 2$
- $N = \{x^5\}$, $h = f_4$
- $n_3 = 1$, $n_4 = 0$
- $N = \{\}$, $h = f_4 \wedge f_3$

Because f3 rules out the last remaining negative example, we're done!

A harder Problem

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- We made one negative into a positive
- Only usable feature is f_3
- Can't add any more features to h
- We're stuck

- Best we can do when H is conjunctions
- Live with error or change H

Disjunctive Normal form

- Like the opposite of conjunctive normal form (but, for now, without negations of the atoms)

$$(A \wedge B \wedge C) \vee (D \wedge A) \vee E$$

- Think of each disjunct as narrowing in on a subset of the positive examples

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

$$(f_3 \wedge f_4) \vee (f_1 \wedge f_3)$$

Learning DNF

- Let H be DNF expressions
- C(h) : number of mentions of features

$$C((f_3 \wedge f_4) \vee (f_1 \wedge f_2)) = 4$$

- Really hard to search this space, so be greedy again!

- A conjunction **covers** an example if all of the features mentioned in the conjunction are true in the example

Algorithm

```
P = set of all positive examples
h = False
Loop until P is empty
    r = True
    N = set of all negative examples
    Loop until N is empty
        If all features are in r, fail
        Else, select a feature f_j to add to r
            r = r ^ f_j
            N = N - examples in n for which f_j = 0
    h := h v r
    Covered := examples in P covered by r
    If Covered is empty, fail
    Else P := P - Covered
end
```

The idea is that each disjunct will *cover* or account for some subset of the positive examples.

So in the outer loop, we make a conjunction that includes some positive examples and no negative examples,

and add it to our hypothesis. We keep doing that until no more positive examples remain to be covered.

Choosing a feature

Heuristic: $v_j = n_j^+ / \max(n_j^-, 0.001)$

n_j^+ = # not yet covered positive examples covered by $r \wedge f_j$

n_j^- = # *not* yet ruled out negative examples covered by $r \wedge f_j$

Choose feature with largest value of V_j

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- h = False, P = $\{x^2, x^3, x^4, x^6\}$
  - r = True, N = $\{x^1, x^5\}$
  - v_1=2/1, v_2=2/1, v_3=4/1, v_4=3/1
  - r = f_3, N=$\{x^1\}$
  - v_1=2/0, v_2=2/1, v_4=3/0
  - r=f_3^f_4, N={}
- h=f_3^f_4, P=$\{x^3\}$
  - r=True, N=$\{x^1, x^5\}$
  - v_1=1/1, v_2=1/1, v_3=1/1, v_4=0/1
  - r=f_1, N=$\{x^1\}$
  - v_2=1/0, v_3=1/0, v_4=0/1
  - r=f_1^f_2, N={}
- h=(f_3^f_4)v(f_1^f_2), P={}

How well does it work?

- We'd like to know how well our h will perform on new data (drawn from the same distribution as the training data)
- Performance of hypothesis on the training set is not indicative of its performance on new data
- Save some data as a **test set**; performance on h is a reasonable estimate of performance on new data

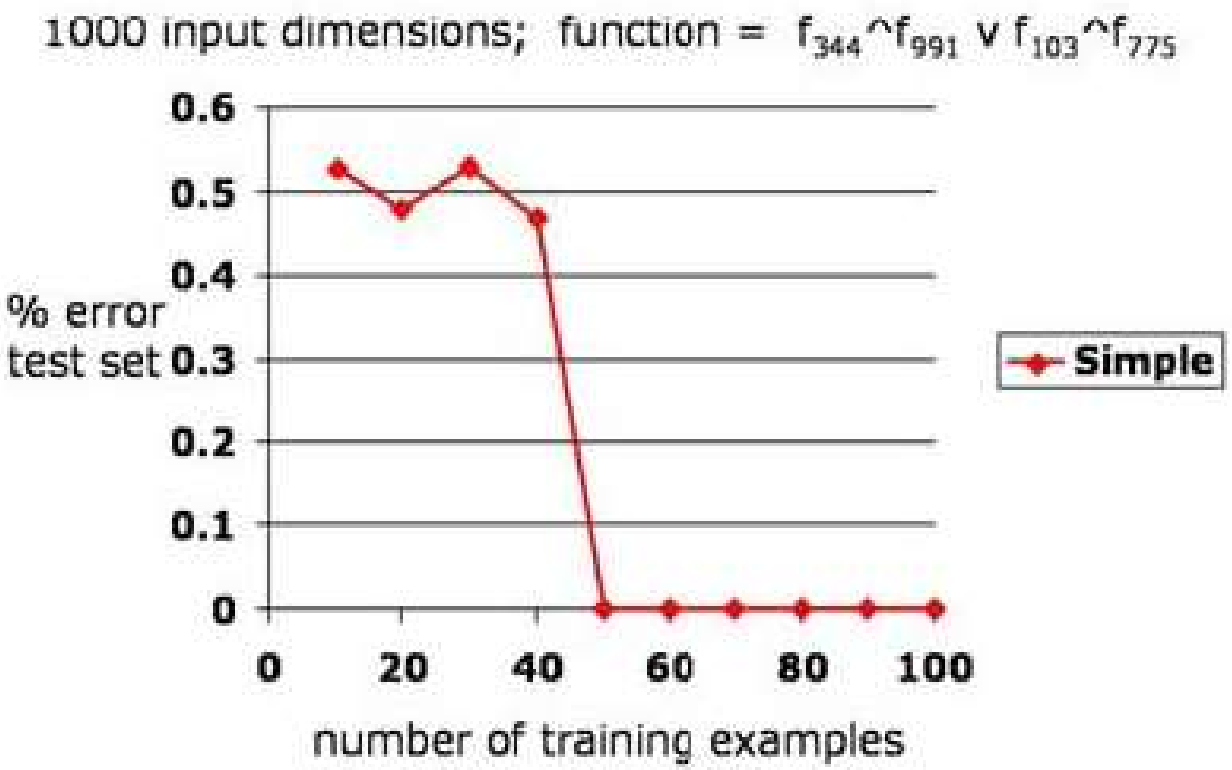
Cross validation

To evaluate performance of an algorithm as a whole (rather than a particular hypothesis):

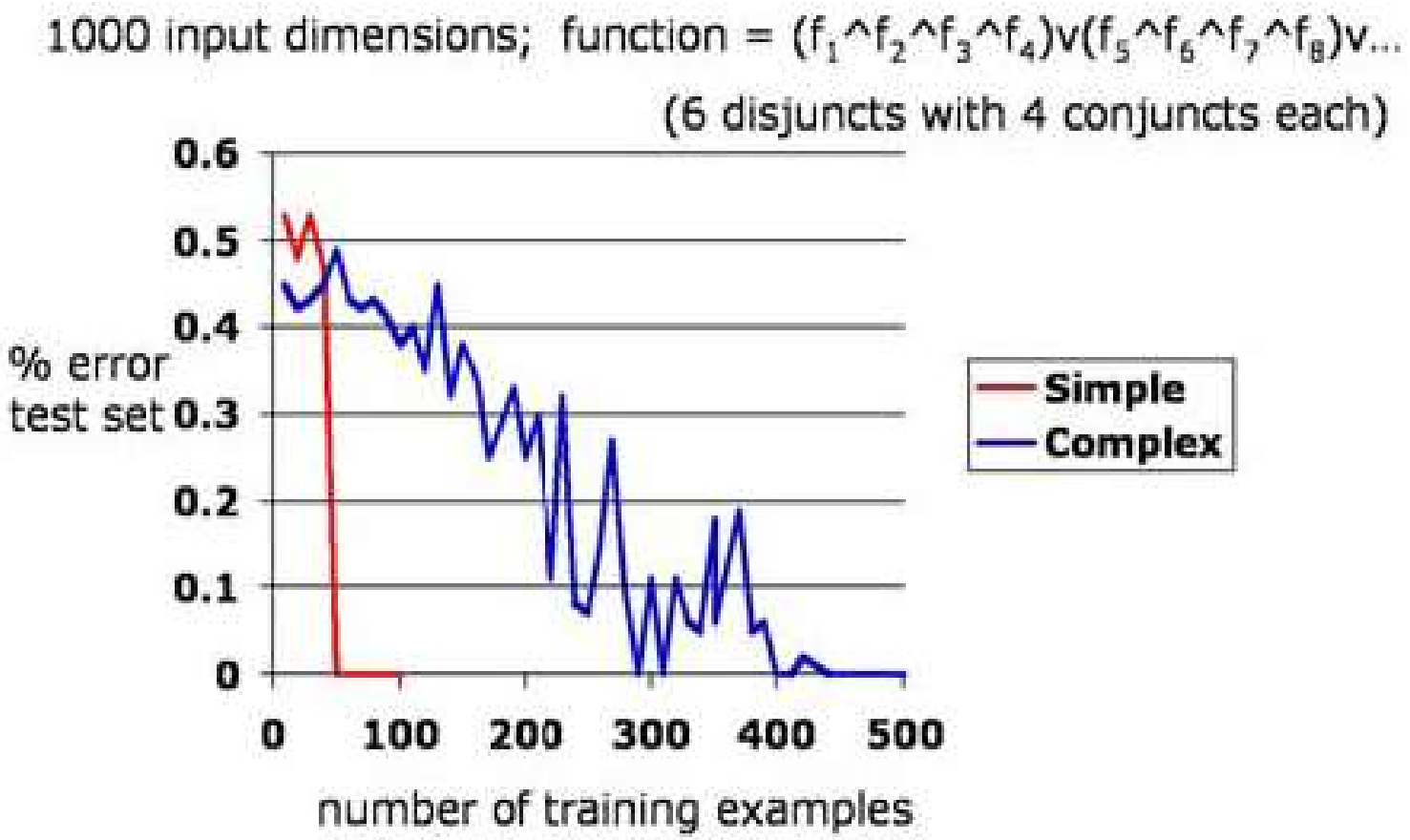
- Divide data into k subsets
- k different times
  - train on k-1 of the subsets
  - test on the held-out subset
- return average test score over all k tests

Useful for deciding which class of algorithms to use on a particular data set.

Learning curves



Learning curves



Simple Gifts

- At every point in the graph, we've found an h that gets the whole training set correct (call it apparently correct)
- When input dimensionality is high and size of training set is small, there are lots of apparently correct hypotheses
- Our algorithm tries to return (but doesn't always) the simplest apparently correct hypothesis
- So, when the target hypothesis is simple, we discover it quickly (the other simple hypotheses are ruled out quickly because there are so few)
- When the target hypothesis is complex, it's hard to rule out all of the (many) other competitors, so it takes more data to learn

Noisy data

- Have to accept non-zero error on training data
- Weaken DNF learning algorithm
  - Don't require the hypothesis to cover all positive examples
  - Don't require each rule to exclude all negative examples

Pseudo code: Noisy DNF Learning

```
P := the set of positive examples
h := False
np := epsilon * number of examples in P
nn := epsilon * number of examples in N
Loop until P has fewer than np elements or not progressing
    r = True
    N = the set of negative examples
    Repeat until N has fewer than nn elements or not progress
        Select a feature f_j to add to r
        r := r ^ f_j
        N := N - examples in N for which f_j = 0
    h := h v r
    P := P - elements in P covered by r
```

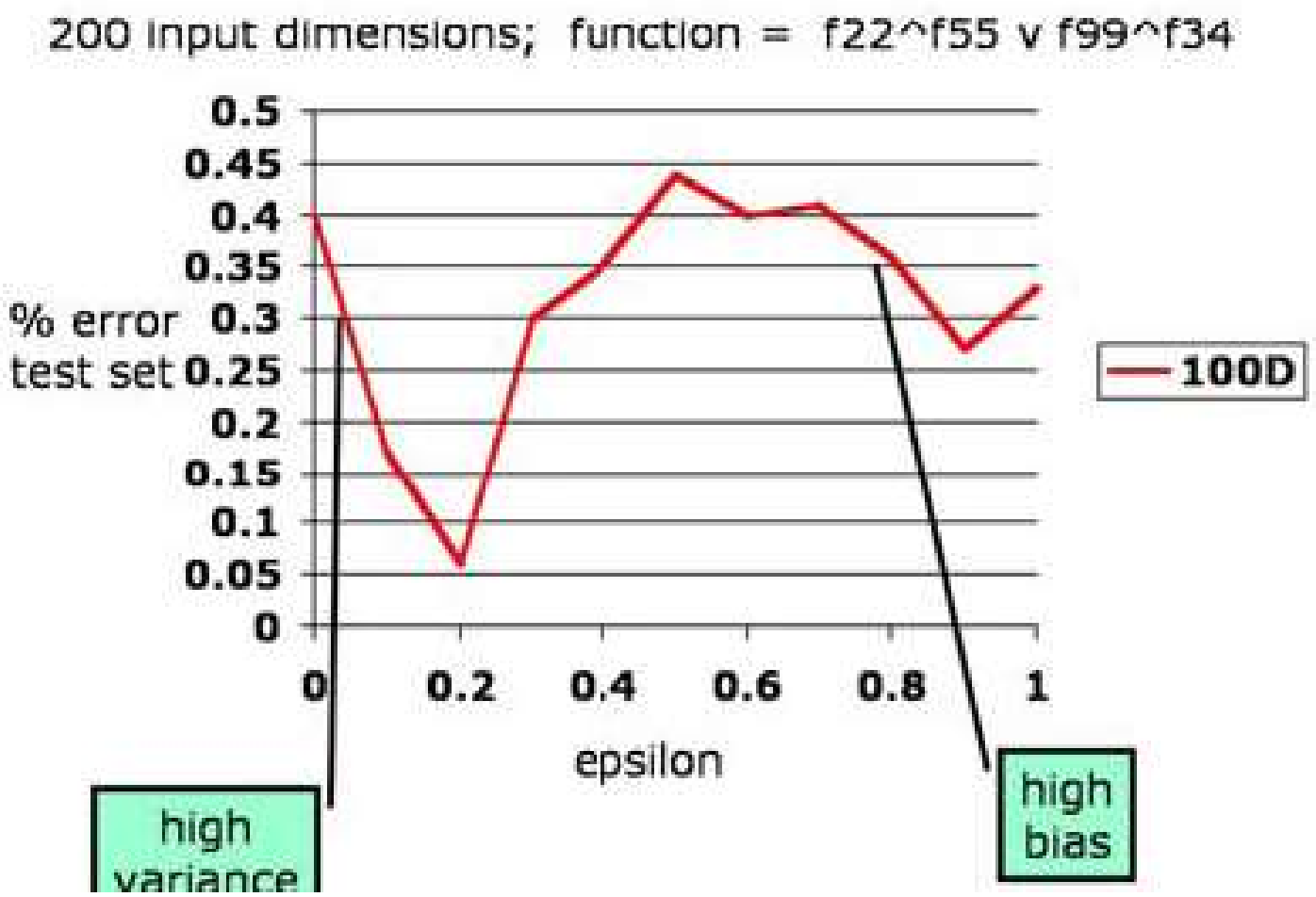
allow epsilon percentage error

Handle failure to progress

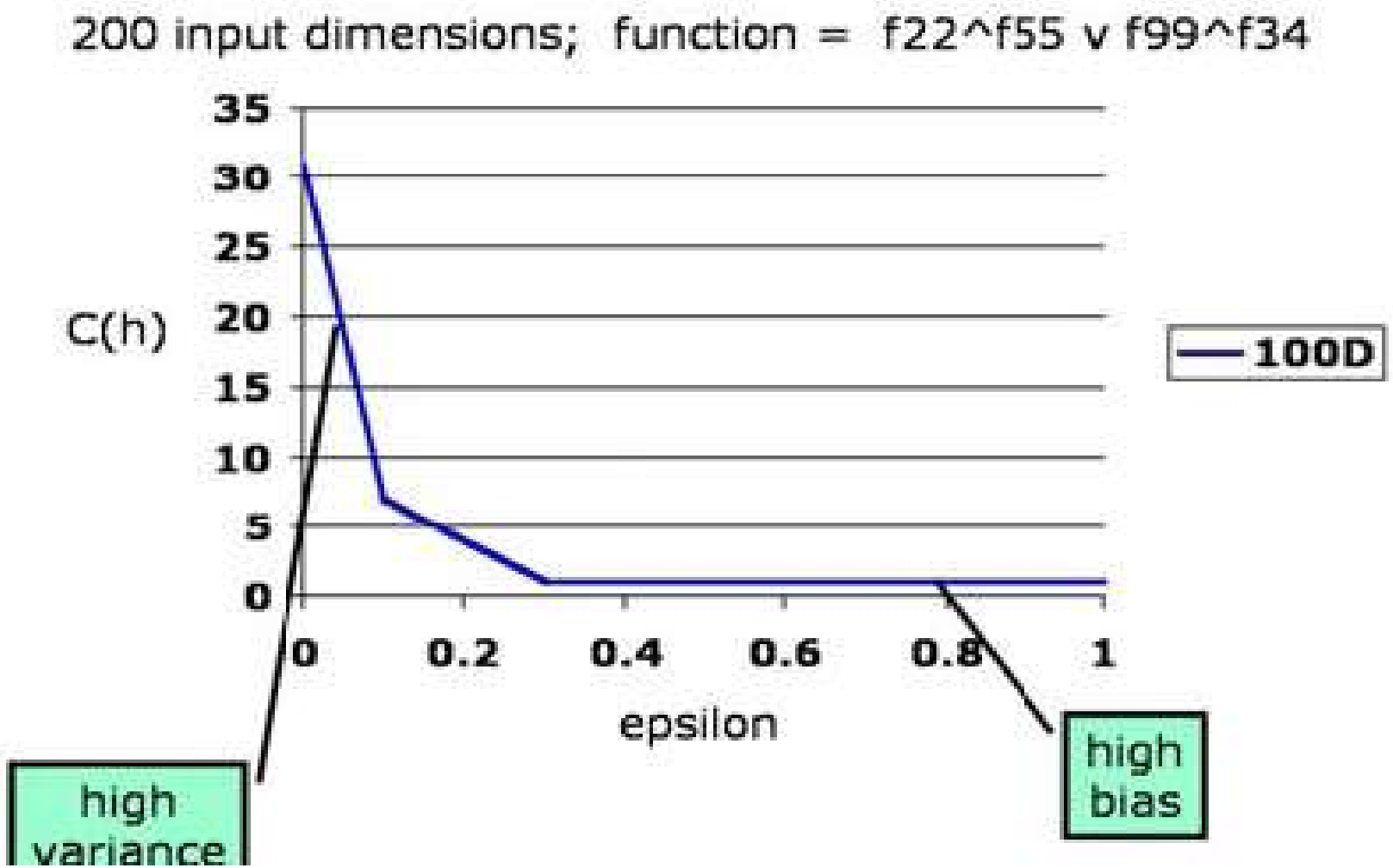
Epsilon is our data

- Parameter epsilon is the percentage error allowed
- The higher epsilon, the simpler and more error-prone the hypothesis
- If epsilon is small and the data is noisy, the algorithm may fail to find an acceptable hypothesis

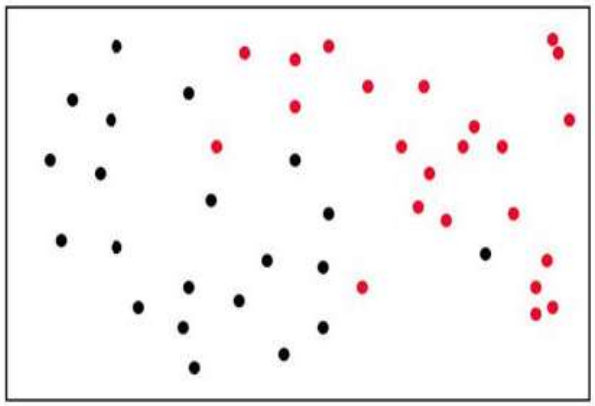
Overfitting curve

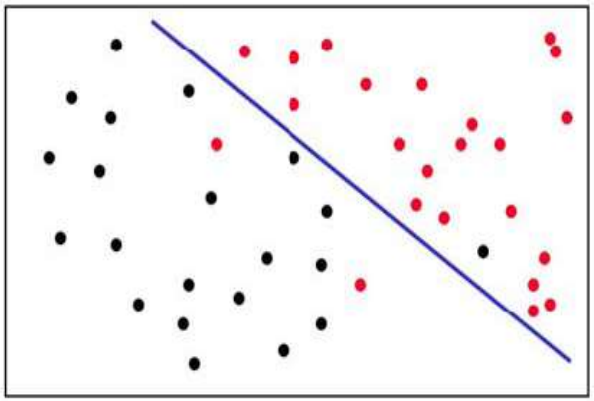


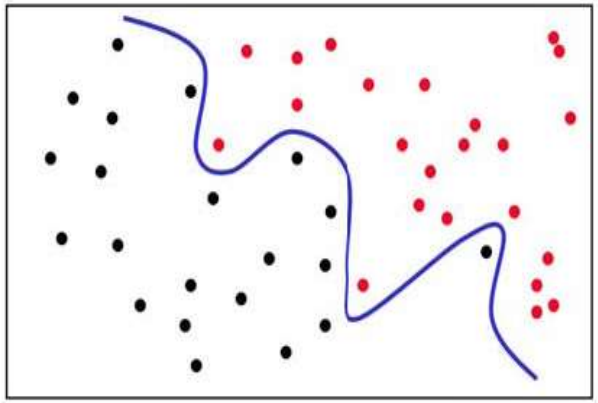
Hypothesis complexity

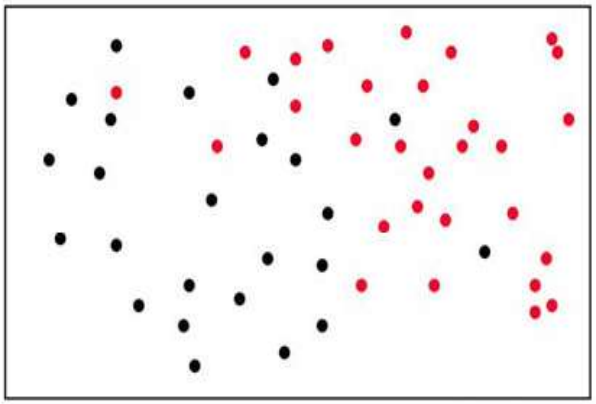


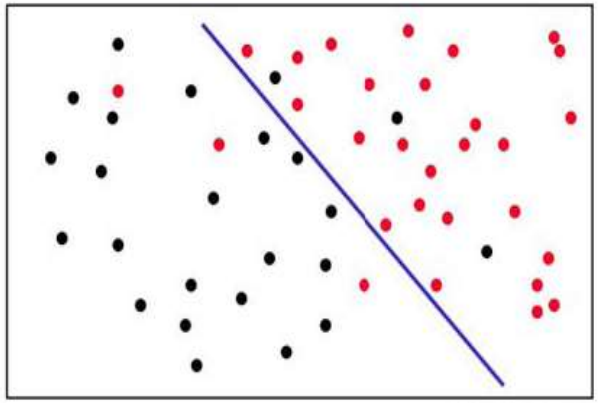
Bias vs variance

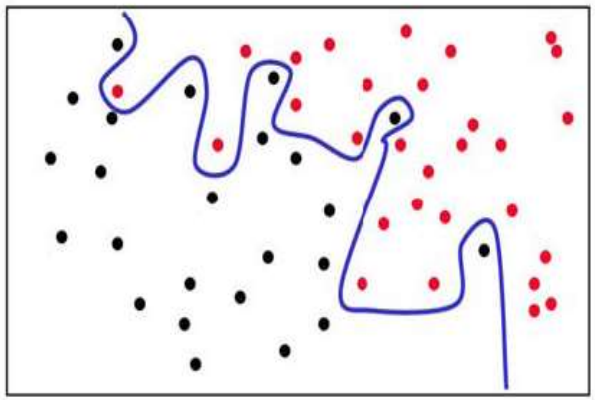


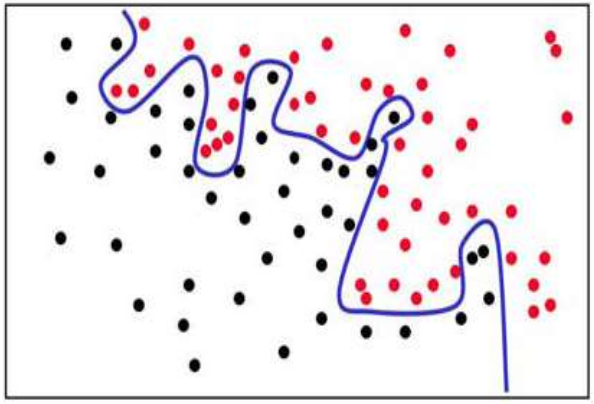












Picking epsilon

- Pick epsilon using cross validation
- Try multiple different values of epsilon
- See which gives the lowest cross-validation error
- Use that value of epsilon to learn a hypothesis from the whole training set
- Return that hypothesis as your best answer

Domains

- Congressional voting: given a congressperson's voting record (list of 1s and 0s), predict party
- Gene splice: predict the beginning of a coding section of the genome; input is vector of elements chosen from the set {ACGT}; encode each element with one bit (or possibly with 4)
- Spam filtering: encode every message as a vector of features, one per word; a feature is on if that word occurs in the message; predict whether or not the message is spam
- Marketing: predict whether a person will buy beer based on previous purchases; encode buying habits with a feature for all products, set to 1 if previously purchased

Congressional Voting

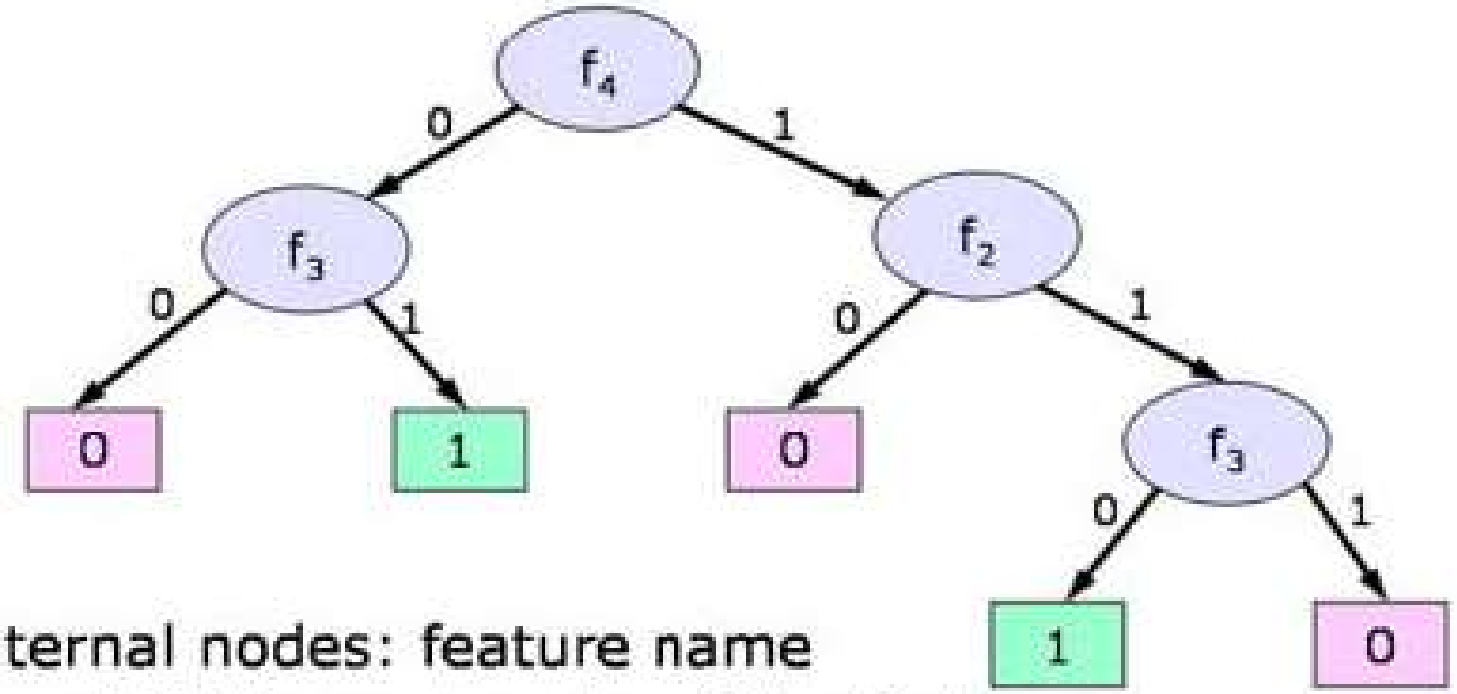
0. handicapped-infants
1. water-project-cost-sharing
2. adoption-of-the-budget-resolution
3. physician-fee-freeze
4. el-salvador-aid
5. religious-groups-in-schools
6. anti-satellite-test-ban
7. aid-to-nicaraguan-contras
8. mx-missile
9. immigration
10. synfuels-corporation-cutback
11. education-spending
12. superfund-right-to-sue
13. crime
14. duty-free-exports
15. export-administration-act-south-africa

232 data points

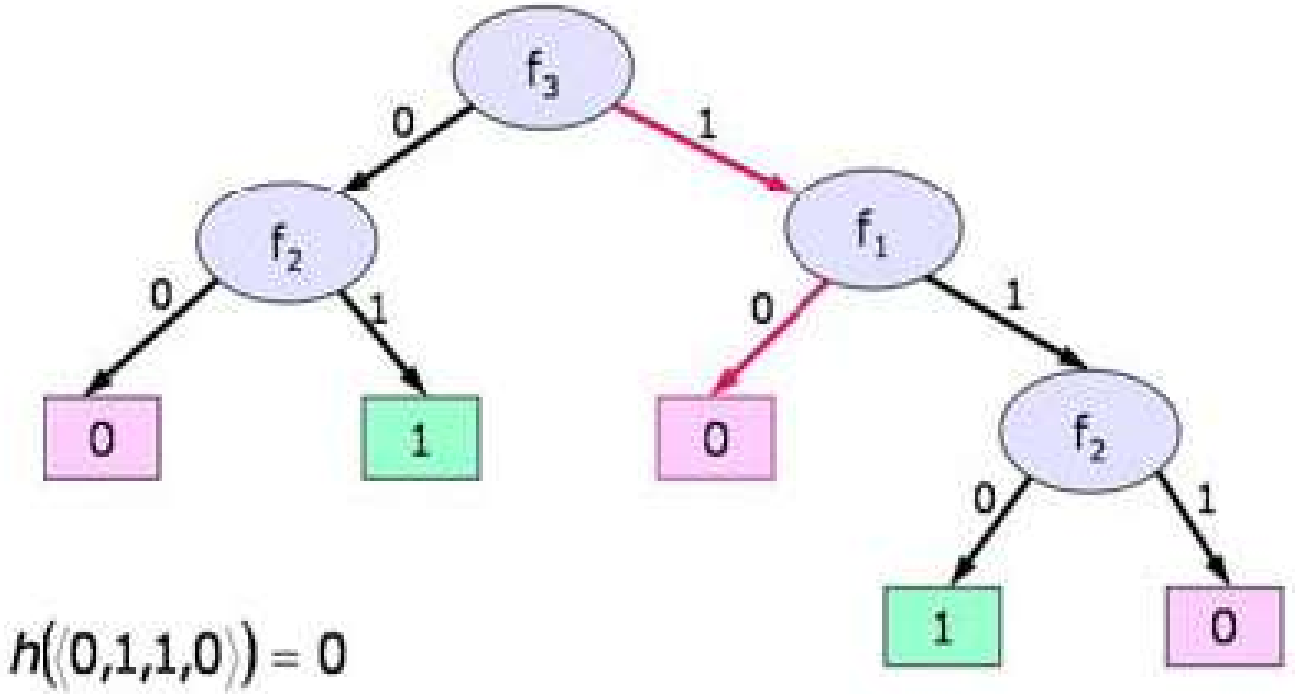
Decision Trees

- DNF learning algorithm is a bit cumbersome and inefficient. Also, the exact effect of the heuristic is unclear.
- Still assume binary inputs and output, but much more broadly applicable.

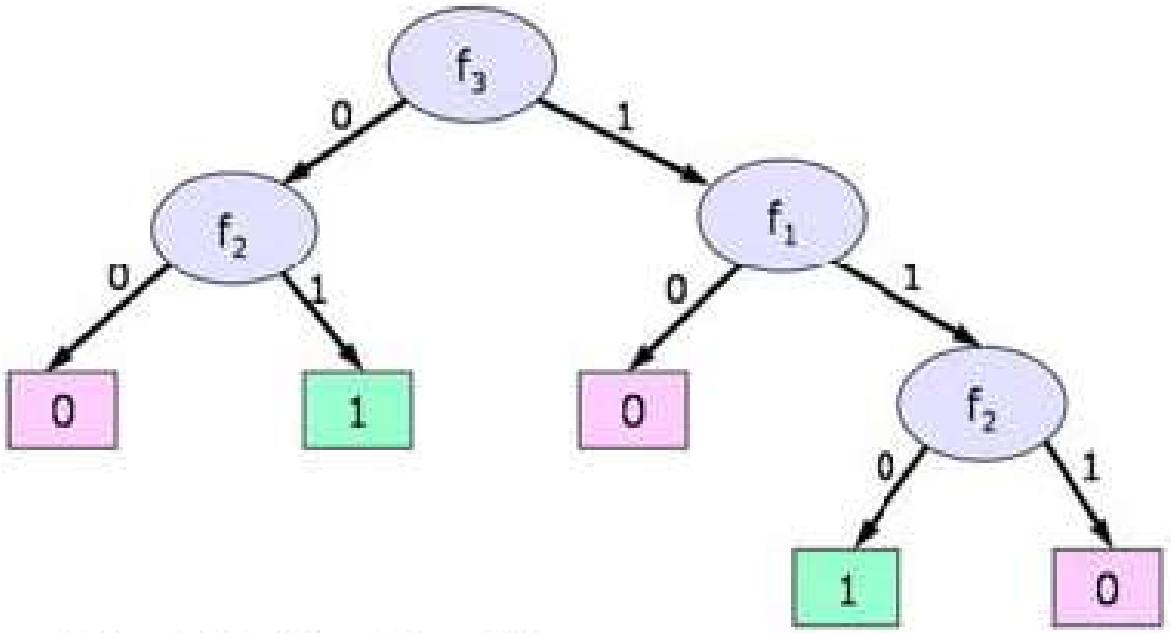
Hypothesis class



- Internal nodes: feature name
- One child for each value of the feature
- Leaf nodes: output



$h(0,1,1,0) = 0$

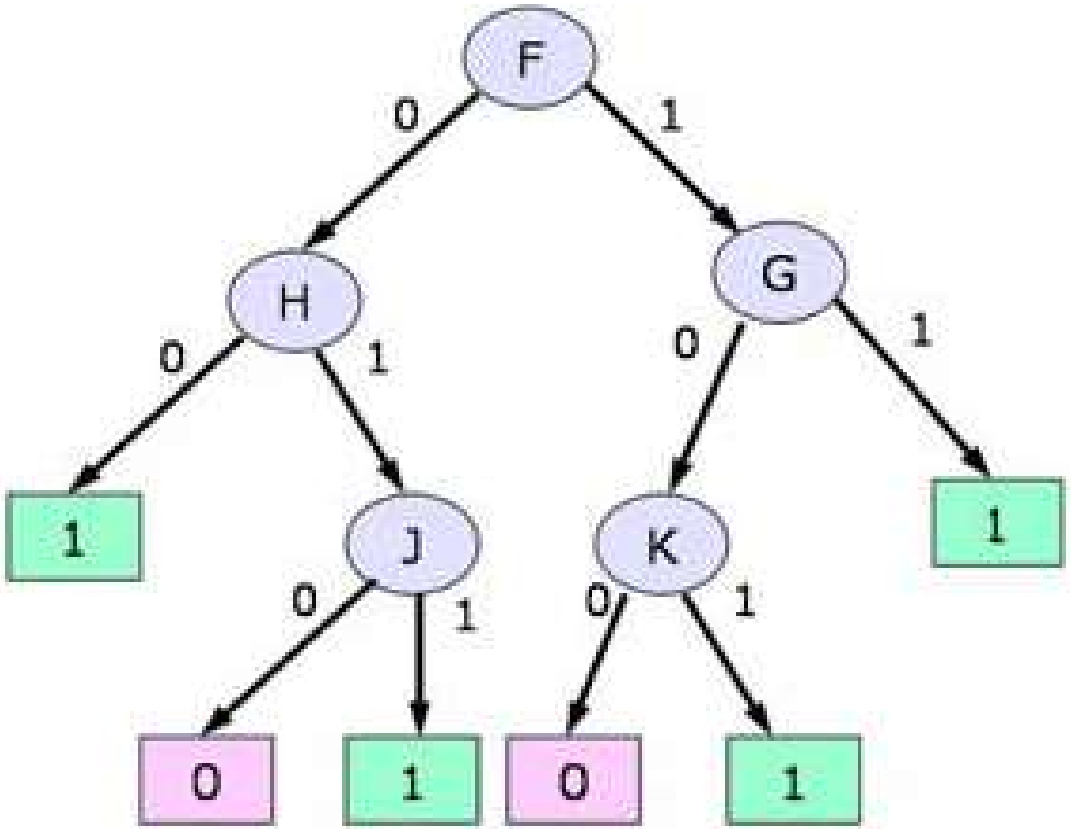


$$h = (\neg f_3 \wedge f_2) \vee (f_3 \wedge f_1 \wedge \neg f_2)$$

Tree Bias

- Both decision trees and DNF with negation can represent any Boolean function. So why bother with trees?
- Because we have a nice algorithm for growing trees that is consistent with a bias for simple trees (few nodes)
- Too hard to find the smallest good tree, so we'll be greedy again
- Have to watch out for overfitting

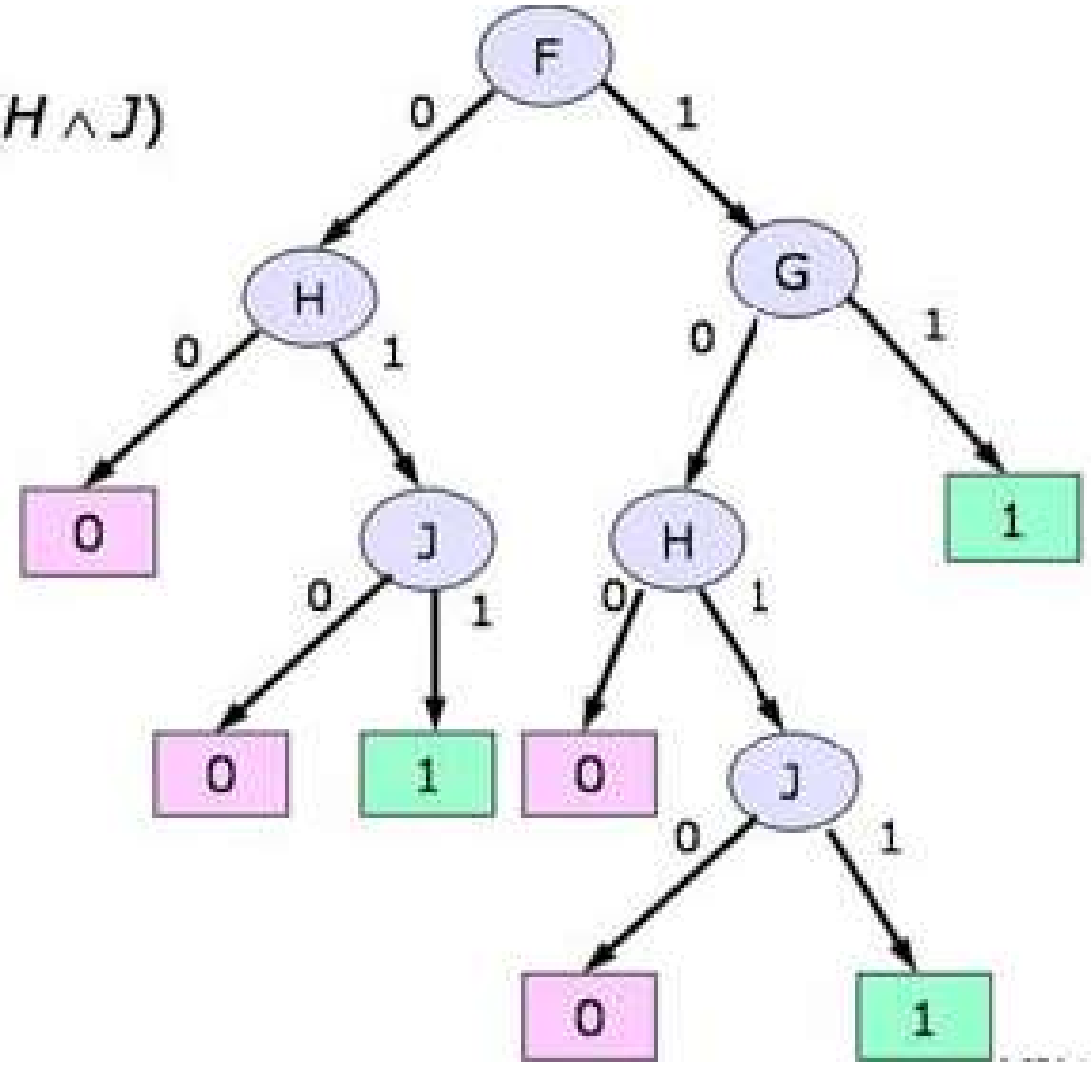
Trees vs DNF



$$(\neg F \wedge \neg H) \vee (\neg F \wedge H \wedge J) \vee (F \wedge \neg G \wedge K) \vee (F \wedge G)$$

Trees vs DNF

$(F \wedge G) \vee (H \wedge J)$



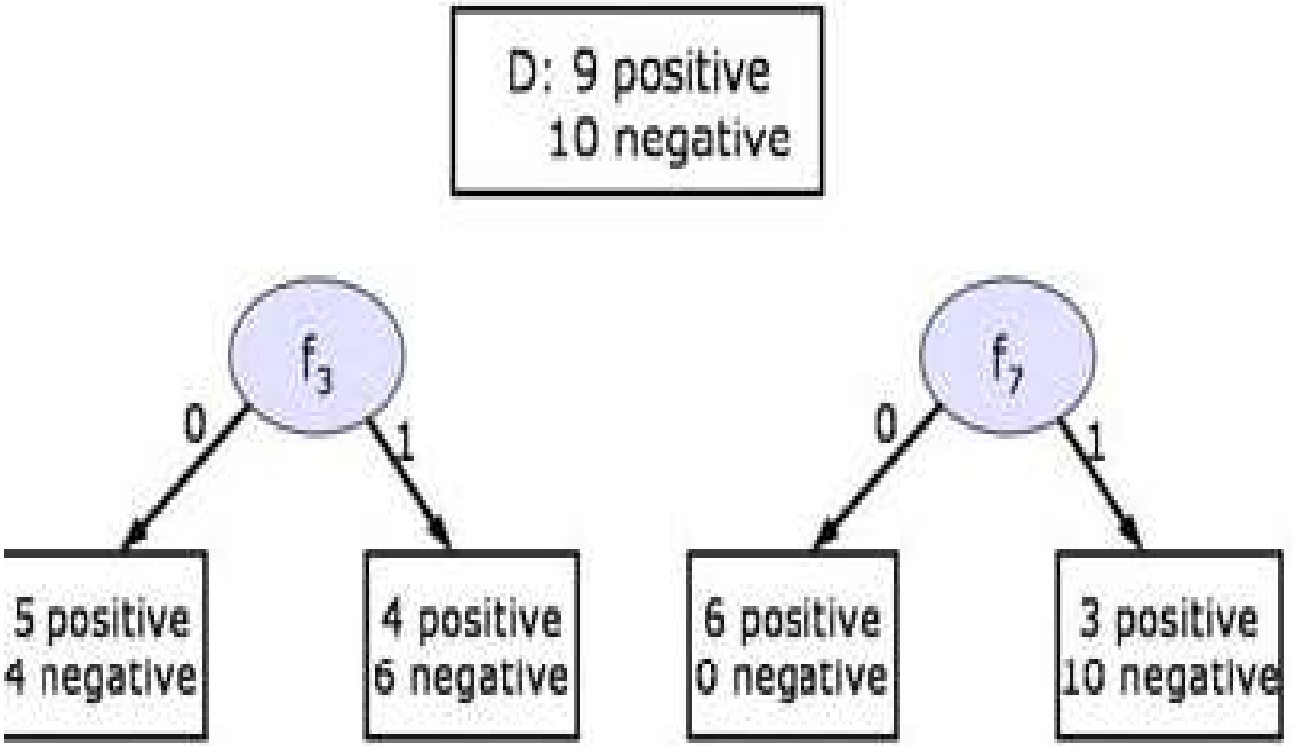
Algorithm

- Developed in parallel in AI by Quinlan and in statistics by Breiman, Friedman, Olshen and Stone

```
BuildTree (Data)
    if all elements of Data have the same y value, then
        MakeLeafNode(y)
    else
        feature := PickBestFeature(Data)
        MakeInternalNode(feature,
            BuildTree(SelectFalse(Data, feature)),
            BuildTree(SelectTrue(Data, feature)))
```

- Best feature minimizes average entropy of data in the children

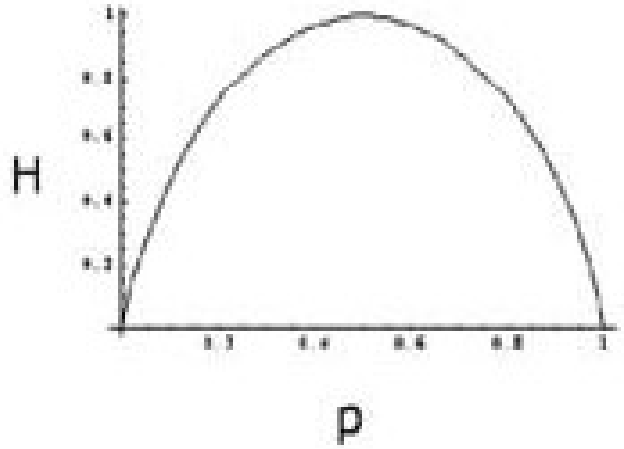
Let's split


D: 9 positive
10 negative
f_3
0
1
5 positive
4 negative
4 positive
6 negative
f_7
0
1
6 positive
0 negative
3 positive
10 negative

Entropy

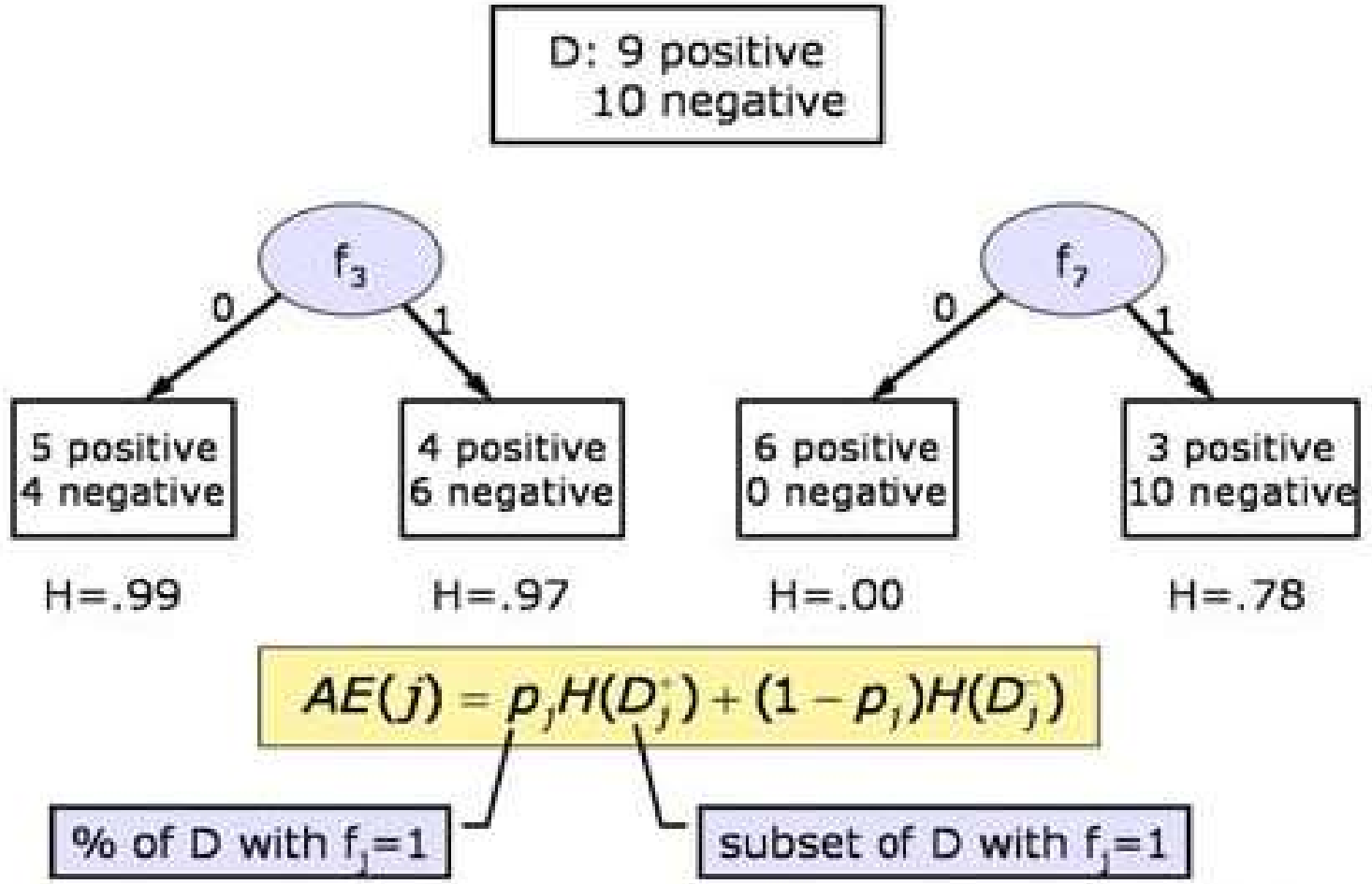
p : proportion of positive examples in a data set

$$H = -p \log_2 p - (1-p) \log_2 (1-p)$$

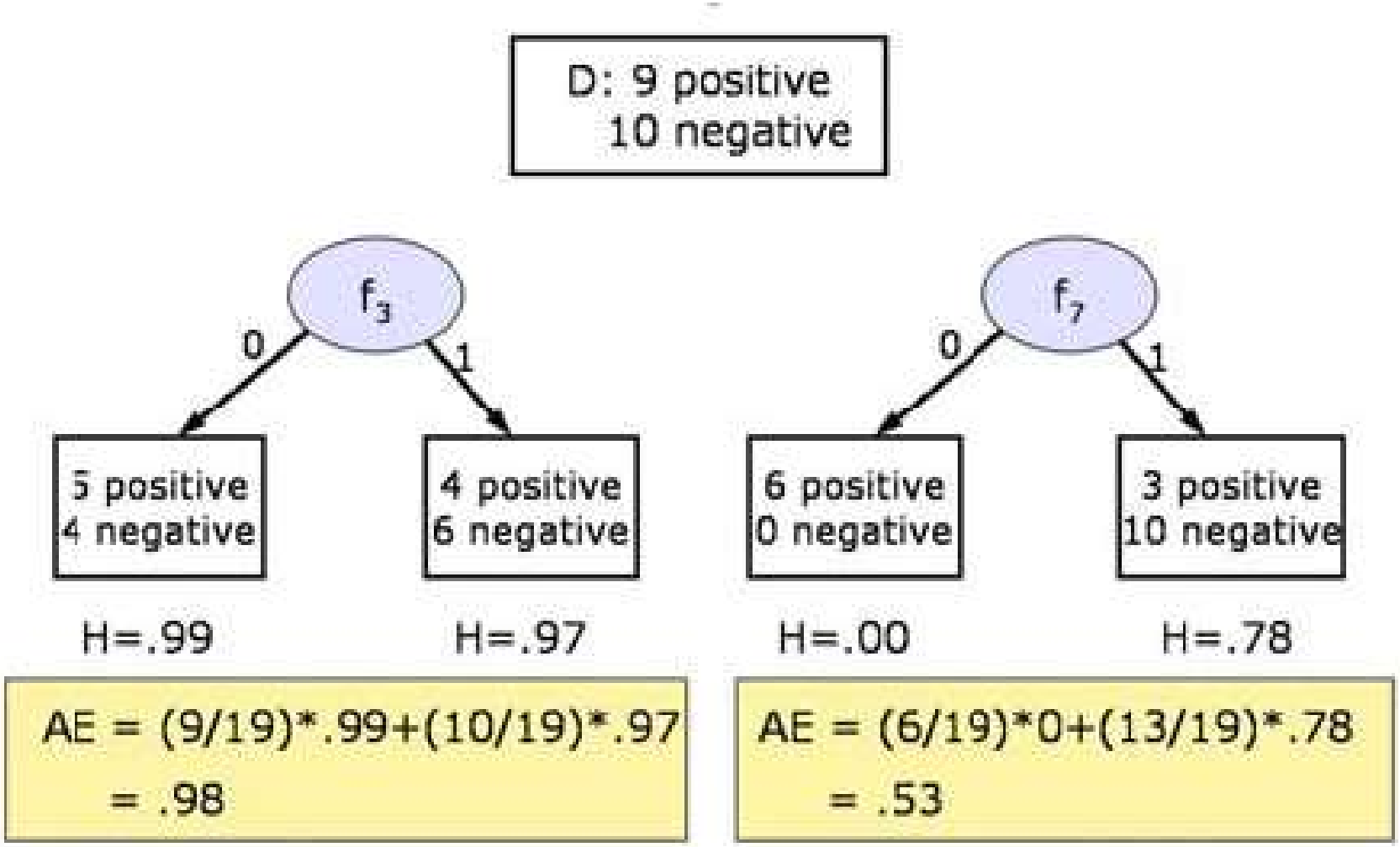


$$0 \log_2 0 = 0$$

Let's split


D: 9 positive
10 negative
f_3
0
1
5 positive
4 negative
4 positive
6 negative
H=.99
H=.97
f_7
0
1
6 positive
0 negative
3 positive
10 negative
H=.00
H=.78
$AE(j) = p_j H(D_j^+) + (1 - p_j) H(D_j^-)$
% of D with f_j=1
subset of D with f_j=1

Let's split


D: 9 positive
10 negative
f_3
0
1
5 positive
4 negative
4 positive
6 negative
H=.99
H=.97
AE = (9/19)*.99+(10/19)*.97
= .98
f_7
0
1
6 positive
0 negative
3 positive
10 negative
H=.00
H=.78
AE = (6/19)*0+(13/19)*.78
= .53

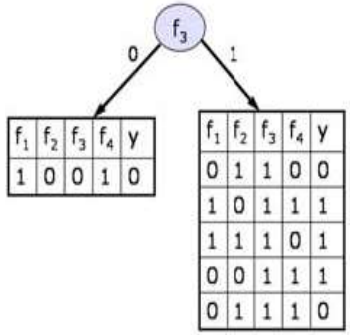
Stopping

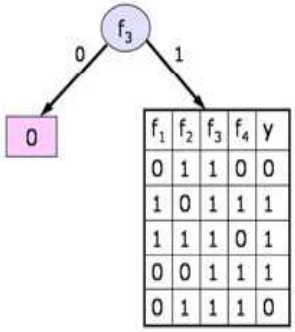
- Stop recursion if data contains only multiple instances of the same x with different y values
  - Make leaf node with output equal to the y value that occurs in the majority of the cases in the data
- Consider stopping to avoid overfitting when:
  - entropy of a data set is below some threshold
  - number elements in a data set is below threshold
  - best next split doesn't decrease average entropy (but this can get us into trouble)

Simulation

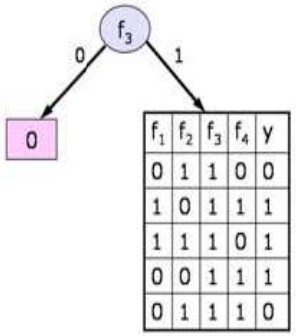
- H(D) = .92
- AE_1=.92, AE_2=.92, AE_3=.81, AE_4=1

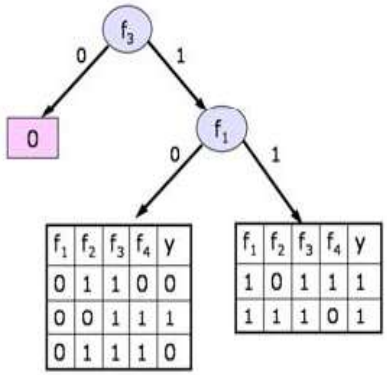
f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	0

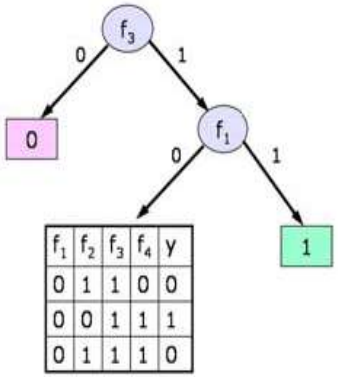


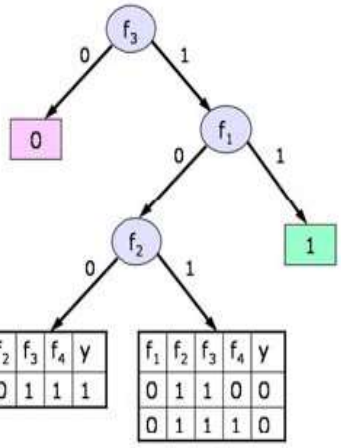


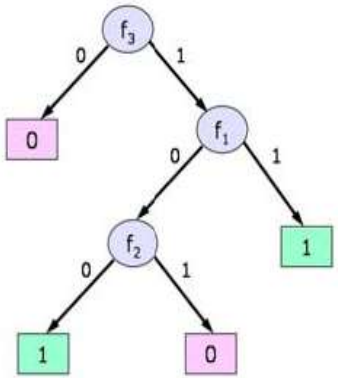
- AE_1=.55, AE_2=.55, AE_4=.95









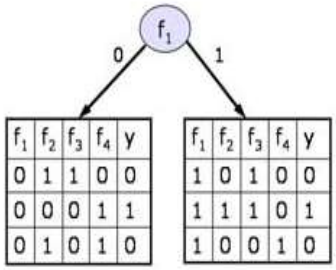


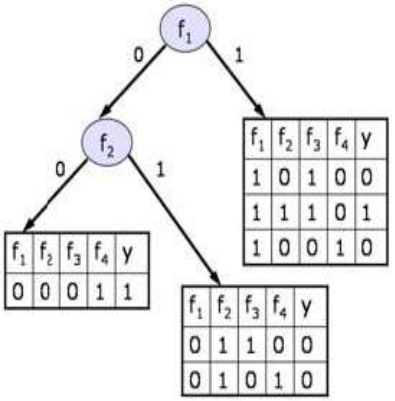
Exclusive OR

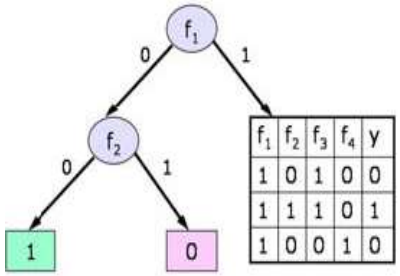
$(A \wedge \neg B) \vee (\neg A \wedge B)$

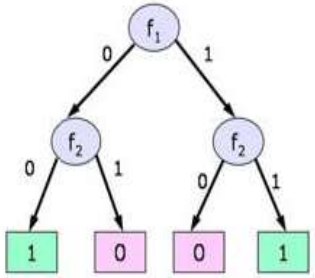
f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	0	0
1	1	1	0	1
0	0	0	1	1
1	0	0	1	0
0	1	0	1	0

- H(D) = .92
- AE_1=.92, AE_2=.92, AE_3=.92, AE_4=.92

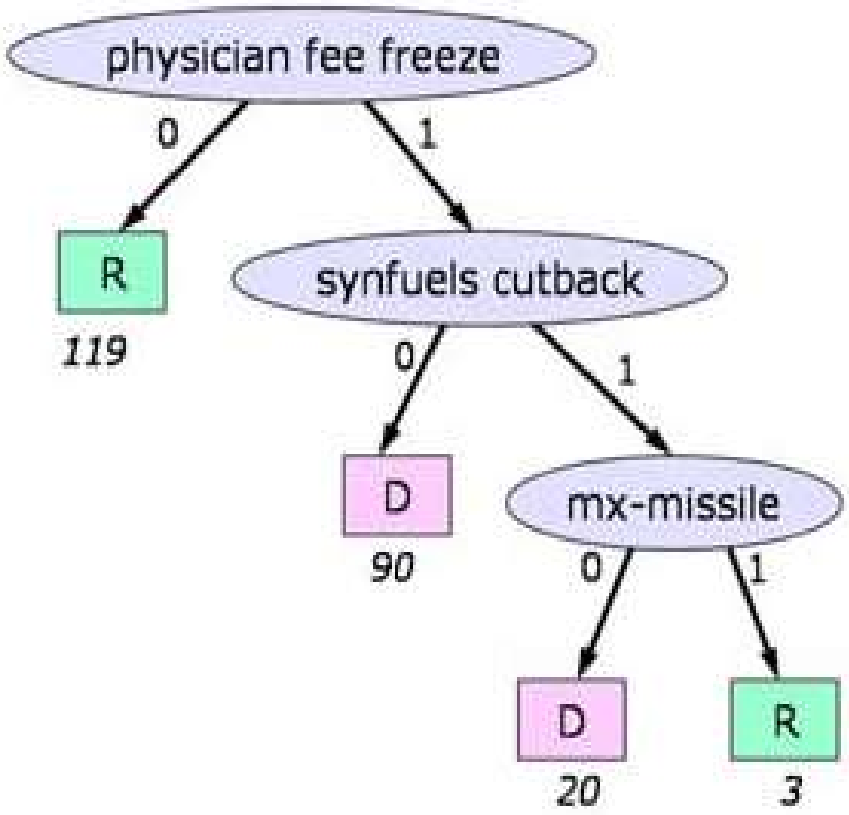








Congressional voting



min leaf size = 20

Naïve Bayes

- Founded on Bayes' rule for probabilistic inference
- Update probability of hypotheses based on evidence
- Choose hypothesis with the maximum probability after the evidence has been incorporated

- Algorithm is particularly useful for domains with **lots** of features

Example

f_1	f_2	f_3	f_4	y
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	0	1	1	1
0	0	0	0	1
1	0	0	1	0
1	1	0	1	0
1	0	0	0	0
1	1	0	1	0
1	0	1	1	0

- $R_1(1,1)=1/5$: fraction of all **positive** examples that have feature 1 **on**
- $R_1(0,1)=4/5$: fraction of all **positive** examples that have feature 1 **off**

f_1	f_2	f_3	f_4	y
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	0	1	1	1
0	0	0	0	1
1	0	0	1	0
1	1	0	1	0
1	0	0	0	0
1	1	0	1	0
1	0	1	1	0

- $R_1(1,1)=1/5$: fraction of all **positive** examples that have feature 1 **on**
- $R_1(0,1)=4/5$: fraction of all **positive** examples that have feature 1 **off**
- $R_1(1,0)=5/5$: fraction of all **negative** examples that have feature 1 **on**
- $R_1(0,0)=0/5$: fraction of all **negative** examples that have feature 1 **off**

f_1	f_2	f_3	f_4	y
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	0	1	1	1
0	0	0	0	1
1	0	0	1	0
1	1	0	1	0
1	0	0	0	0
1	1	0	1	0
1	0	1	1	0

$R_1(1,1)=1/5 \quad R_1(0,1)=4/5$
$R_1(1,0)=5/5 \quad R_1(0,0)=0/5$

$R_2(1,1)=1/5 \quad R_2(0,1)=4/5$
$R_2(1,0)=2/5 \quad R_2(0,0)=3/5$

$R_3(1,1)=4/5 \quad R_3(0,1)=1/5$
$R_3(1,0)=1/5 \quad R_3(0,0)=4/5$

$R_4(1,1)=2/5 \quad R_4(0,1)=3/5$
$R_4(1,0)=4/5 \quad R_4(0,0)=1/5$

Prediction

- $R_1(1,1)=1/5$ $R_1(0,1)=4/5$
- $R_1(1,0)=5/5$ $R_1(0,0)=0/5$
- $R_2(1,1)=1/5$ $R_2(0,1)=4/5$
- $R_2(1,0)=2/5$ $R_2(0,0)=3/5$
- $R_3(1,1)=4/5$ $R_3(0,1)=1/5$
- $R_3(1,0)=1/5$ $R_3(0,0)=4/5$
- $R_4(1,1)=2/5$ $R_4(0,1)=3/5$
- $R_4(1,0)=4/5$ $R_4(0,0)=1/5$

P

- New x = <0,0,1,1>
- $S(1) = R_1(0,1)*R_2(0,1)*R_3(1,1)*R_4(1,1) = .205$
- $S(0) = R_1(0,0)*R_2(0,0)*R_3(1,0)*R_4(1,0) = 0$
- S(1) > S(0), so predict class 1

Learning Algorithm

- Estimate from the data, for all j:

$$R_j(1,1) = \frac{\#(x_j^i = 1 \wedge y^i = 1)}{\#(y^i = 1)}$$

$$R_j(0,1) = 1 - R_j(1,1)$$

$$R_j(1,0) = \frac{\#(x_j^i = 1 \wedge y^i = 0)}{\#(y^i = 0)}$$

$$R_j(0,0) = 1 - R_j(1,0)$$

Prediction Algorithm

- Given a new x,

$$\log S(1) = \sum_j \begin{cases} \log R_j(1,1) & \text{if } x_j = 1 \\ \log R_j(0,1) & \text{otherwise} \end{cases}$$

$$\log S(0) = \sum_j \begin{cases} \log R_j(1,0) & \text{if } x_j = 1 \\ \log R_j(0,0) & \text{otherwise} \end{cases}$$

- Output 1 if $\log S(1) > \log S(0)$

Better to add logs than to multiply small probabilities

Laplace Correction

- Avoid getting 0 or 1 as an answer:

$$R_j(1,1) = \frac{\#(x_j^i = 1 \wedge y^i = 1) + \mathbf{1}}{\#(y^i = 1) + \mathbf{2}}$$

$$R_j(0,1) = 1 - R_j(1,1)$$

$$R_j(1,0) = \frac{\#(x_j^i = 1 \wedge y^i = 0) + \mathbf{1}}{\#(y^i = 0) + \mathbf{2}}$$

$$R_j(0,0) = 1 - R_j(1,0)$$

Example with correction

f_1	f_2	f_3	f_4	y
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	0	1	1	1
0	0	0	0	1
1	0	0	1	0
1	1	0	1	0
1	0	0	0	0
1	1	0	1	0
1	0	1	1	0

- $R_1(1,1)=2/7$ $R_1(0,1)=5/7$
- $R_1(1,0)=6/7$ **$R_1(0,0)=1/7$**
- $R_2(1,1)=2/7$ $R_2(0,1)=5/7$
- $R_2(1,0)=3/7$ $R_2(0,0)=4/7$
- $R_3(1,1)=5/7$ $R_3(0,1)=2/7$
- $R_3(1,0)=2/7$ $R_3(0,0)=5/7$
- $R_4(1,1)=3/7$ $R_4(0,1)=4/7$
- $R_4(1,0)=5/7$ $R_4(0,0)=2/7$

Prediction with correction

- $R_1(1,1)=2/7$ $R_1(0,1)=5/7$
- $R_1(1,0)=6/7$ $R_1(0,0)=1/7$
- $R_2(1,1)=2/7$ $R_2(0,1)=5/7$
- $R_2(1,0)=3/7$ $R_2(0,0)=4/7$
- $R_3(1,1)=5/7$ $R_3(0,1)=2/7$
- $R_3(1,0)=2/7$ $R_3(0,0)=5/7$
- $R_4(1,1)=3/7$ $R_4(0,1)=4/7$
- $R_4(1,0)=5/7$ $R_4(0,0)=2/7$

- New x = <0,0,1,1>
- $S(1) = R_1(0,1)*R_2(0,1)*R_3(1,1)*R_4(1,1) = .156$
- $S(0) = R_1(0,0)*R_2(0,0)*R_3(1,0)*R_4(1,0) = .017$
- S(1) > S(0), so predict class 1

Hypothesis space

- Output 1 if

$$\prod_j \alpha_j x_j + (1-\alpha_j)(1-x_j) > \prod_j \beta_j x_j + (1-\beta_j)(1-x_j)$$

- Depends on parameters $\alpha_1 \ldots \alpha_n, \beta_1 \ldots \beta_n$ (which we set to be the R_j values)

- Our method of computing parameters doesn't minimize training set error, but it's fast!

- Weight of feature j's "vote" in favor of output 1:

$$\log \frac{\alpha_j}{1-\alpha_j} - \log \frac{\beta_j}{1-\beta_j}$$

Exclusive OR

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	0	0
1	0	0	1	0
0	1	0	1	0
1	1	1	0	1
0	0	0	1	1

- $R_1(1,1)=2/4$ $R_1(0,1)=2/4$
- $R_1(1,0)=3/6$ $R_1(0,0)=3/6$
- $R_2(1,1)=2/4$ $R_2(0,1)=2/4$
- $R_2(1,0)=3/6$ $R_2(0,0)=3/6$
- $R_3(1,1)=2/4$ $R_3(0,1)=2/4$
- $R_3(1,0)=3/6$ $R_3(0,0)=3/6$
- $R_4(1,1)=2/4$ $R_4(0,1)=2/4$
- $R_4(1,0)=3/6$ $R_4(0,0)=3/6$

- For any new x
- $S(1) = .5 * .5 * .5 * .5 = .0625$
- $S(0) = .5 * .5 * .5 * .5 = .0625$
- We're indifferent between classes

Probabilistic Inference

- Think of features and output as random variables
- Learn $\Pr(Y = 1 | f_1, \ldots, f_n)$
- Given new example, compute probability it has value 1
- Generate answer 1 if that value is > 0.5, else 0

- Concentrate on estimating this distribution from data

$$\Pr(Y = 1 | f_1, \ldots, f_n)$$

Bayes' rule

- Generically:

$$\Pr(A \mid B) = \Pr(B \mid A)\frac{\Pr(A)}{\Pr(B)}$$

- Specifically:

$$\Pr(Y = 1 \mid f_1 \ldots f_n) = \Pr(f_1 \ldots f_n \mid Y = 1)\frac{\Pr(Y = 1)}{\Pr(f_1 \ldots f_n)}$$

independent of Y

prior

- Concentrate on:

$$\Pr(f_1 \ldots f_n \mid Y = 1)$$

Why is Bayes Naive

- Make a big independence assumption

$$\Pr(f_1 \ldots f_n \mid Y = 1) = \prod_j \Pr(f_j \mid Y = 1)$$

Learning Algorithm

- Estimate from the data, for all j:

$$R(f_j = 1 \mid Y = 1) = \frac{\#(x_j^i = 1 \wedge y^i = 1)}{\#(y^i = 1)}$$

$$R(f_j = 0 \mid Y = 1) = 1 - R(f_j = 1 \mid Y = 1)$$

$$R(f_j = 1 \mid Y = 0) = \frac{\#(x_j^i = 1 \wedge y^i = 0)}{\#(y^i = 0)}$$

$$R(f_j = 0 \mid Y = 0) = 1 - R(f_j = 1 \mid Y = 0)$$

Prediction Algorithm

- Given a new x,

$$S(x_1 \ldots x_n \mid Y=1) = \prod_j \begin{cases} R(f_j = 1 \mid Y=1) & \text{if } x_j = 1 \\ R(f_j = 0 \mid Y=1) & \text{otherwise} \end{cases}$$

$$S(x_1 \ldots x_n \mid Y=0) = \prod_j \begin{cases} R(f_j = 1 \mid Y=0) & \text{if } x_j = 1 \\ R(f_j = 0 \mid Y=0) & \text{otherwise} \end{cases}$$

- Output 1 if

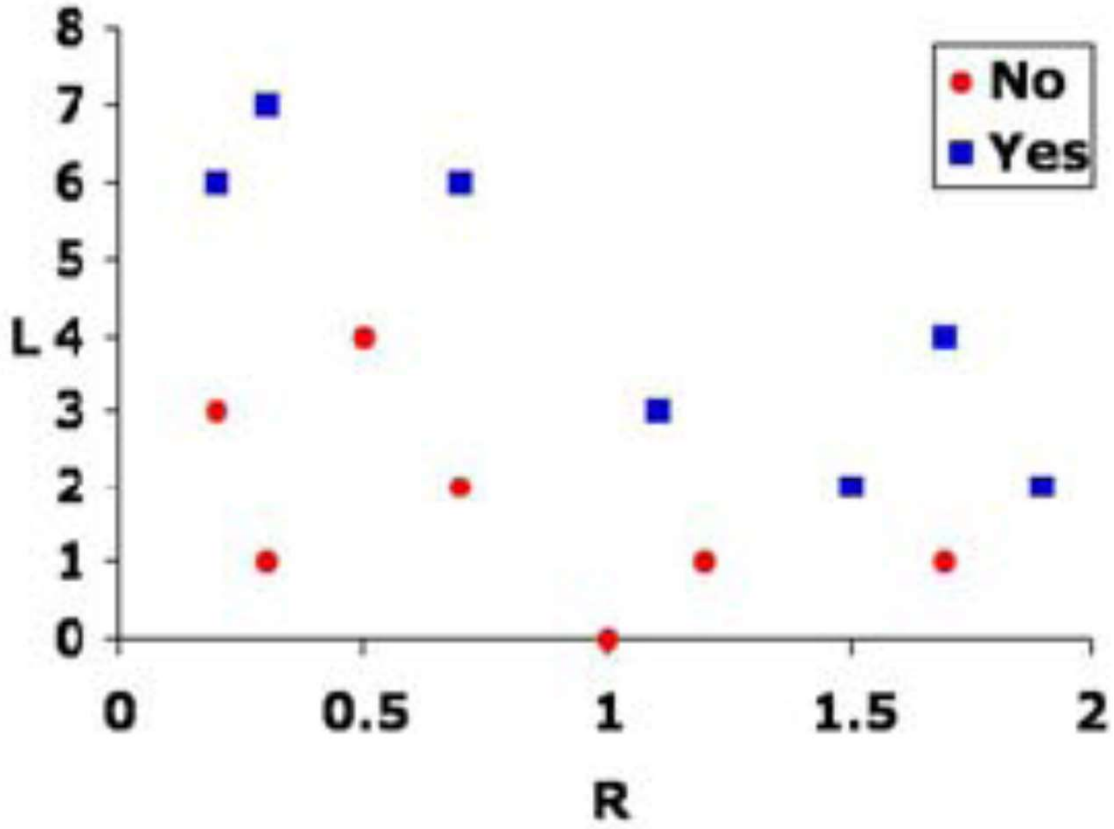
$$S(x_1 \ldots x_n \mid Y=1) > S(x_1 \ldots x_n \mid Y=0)$$

Feature Spaces

- Features can be much more complex
- Drawn from bigger discrete set
  - If set is unordered (4 different makes of cars, for example), use binary attributes to encode the values (1000, 0100, 0010, 0001)
  - If set is ordered, treat as real-valued
- Real-valued: bias that inputs whose features have "nearby" values ought to have "nearby" outputs

Predicting Bankruptcy

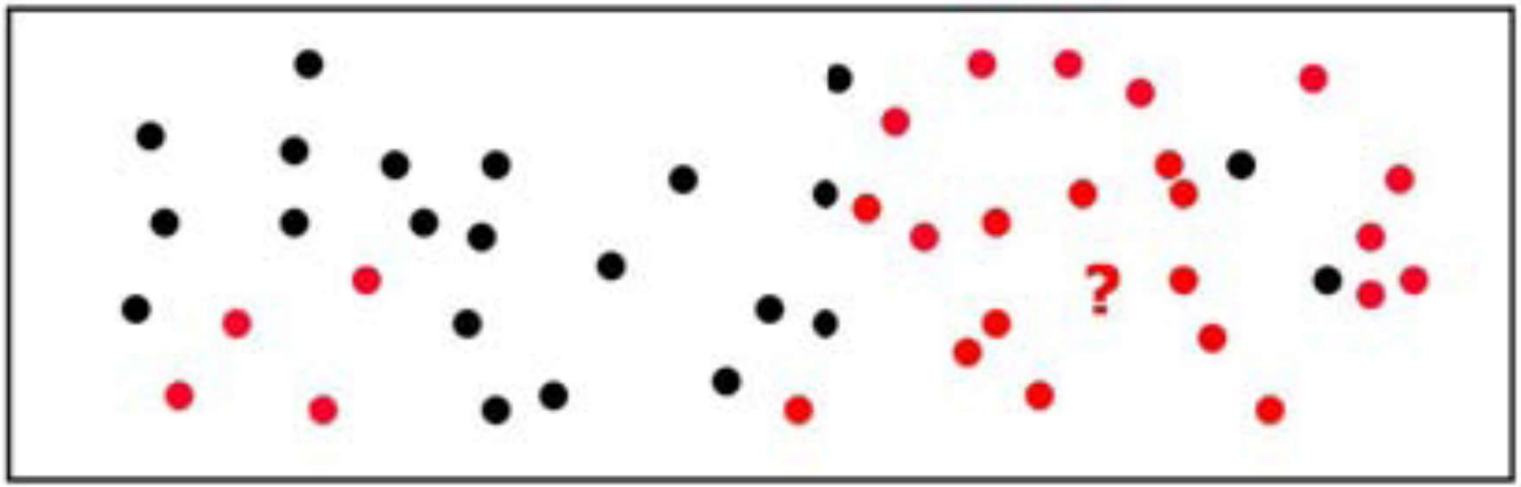
L	R	B
3	0.2	No
1	0.3	No
4	0.5	No
2	0.7	No
0	1.0	No
1	1.2	No
1	1.7	No
6	0.2	Yes
7	0.3	Yes
6	0.7	Yes
3	1.1	Yes
2	1.5	Yes
4	1.7	Yes
2	1.9	Yes



L: #late payments / year
R: expenses / income

Nearest neighbor

- Remember all your data
- When someone asks a question,
  - find the nearest old data point
  - return the answer associated with it



What do we mean by "nearest"?

- Need a distance function on inputs
- Typically use Euclidean distance (length of a straight line between the points)

$$D(x^l, x^k) = \sqrt{\sum_j (x_j^l - x_j^k)^2}$$

- Distance between character strings might be number of edits required to turn one into the other

Scaling

- What if we're trying to predict a car's gas mileage?
  - f_1 = weight in pounds
  - f_2 = number of cylinders
- Any effect of f_2 will be completely lost because of the relative scales
- So, re-scale the inputs to have mean 0 and variance 1:

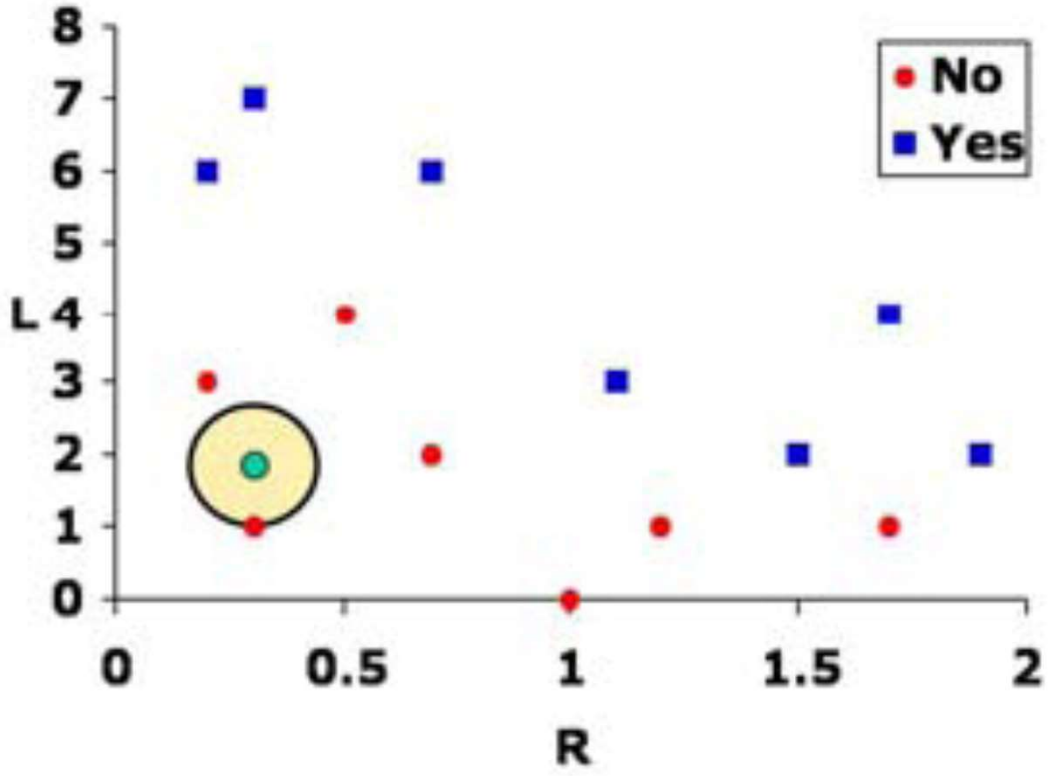
$$x' = \frac{x - \bar{x}}{\sigma_x}$$

$\bar{x}$: average

σ_x: standard deviation

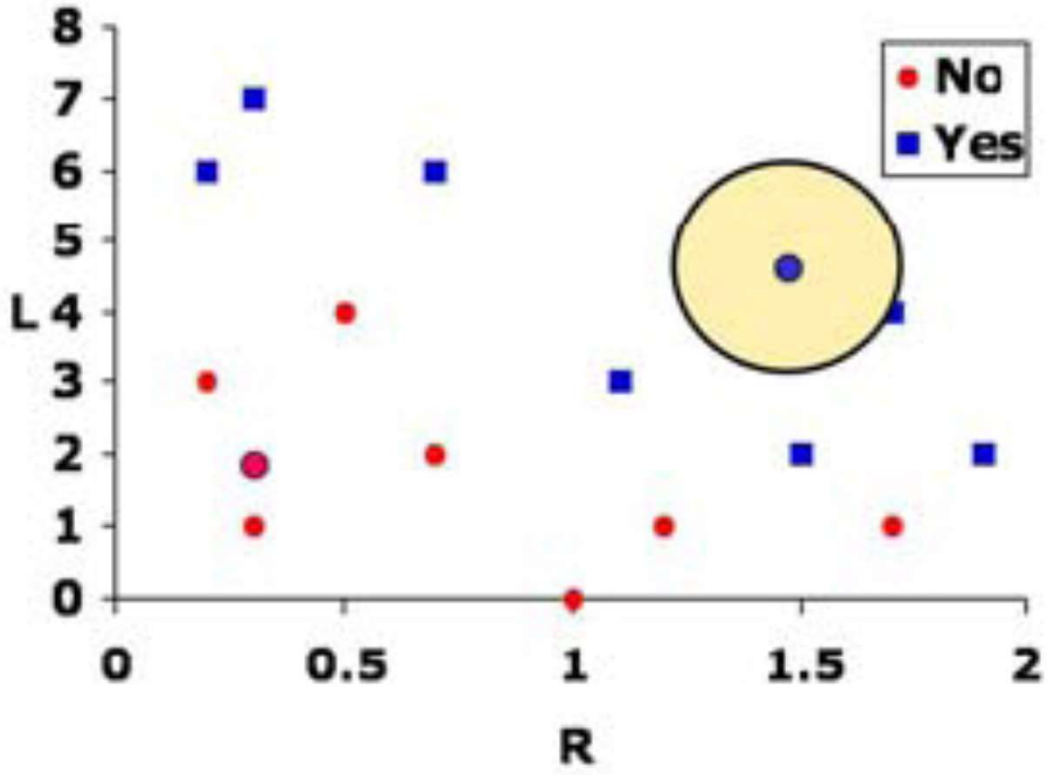
- Or, build knowledge in by scaling features differently
- Or use cross-validation to choose scales

Predicting Bankruptcy



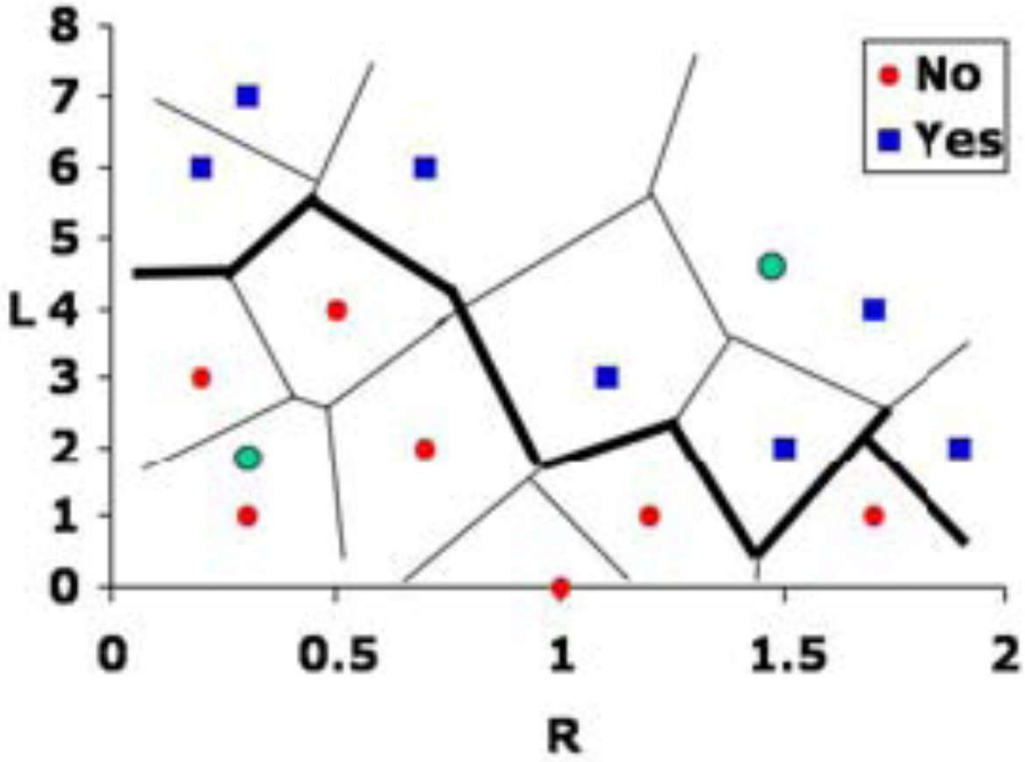
$$D(x^i, x^k) = \sqrt{\sum_j (L^i - L^k)^2 + (5R^i - 5R^k)^2}$$

Predicting Bankruptcy



$$D(x^i, x^k) = \sqrt{\sum_j (L^i - L^k)^2 + (5R^i - 5R^k)^2}$$

Hypothesis

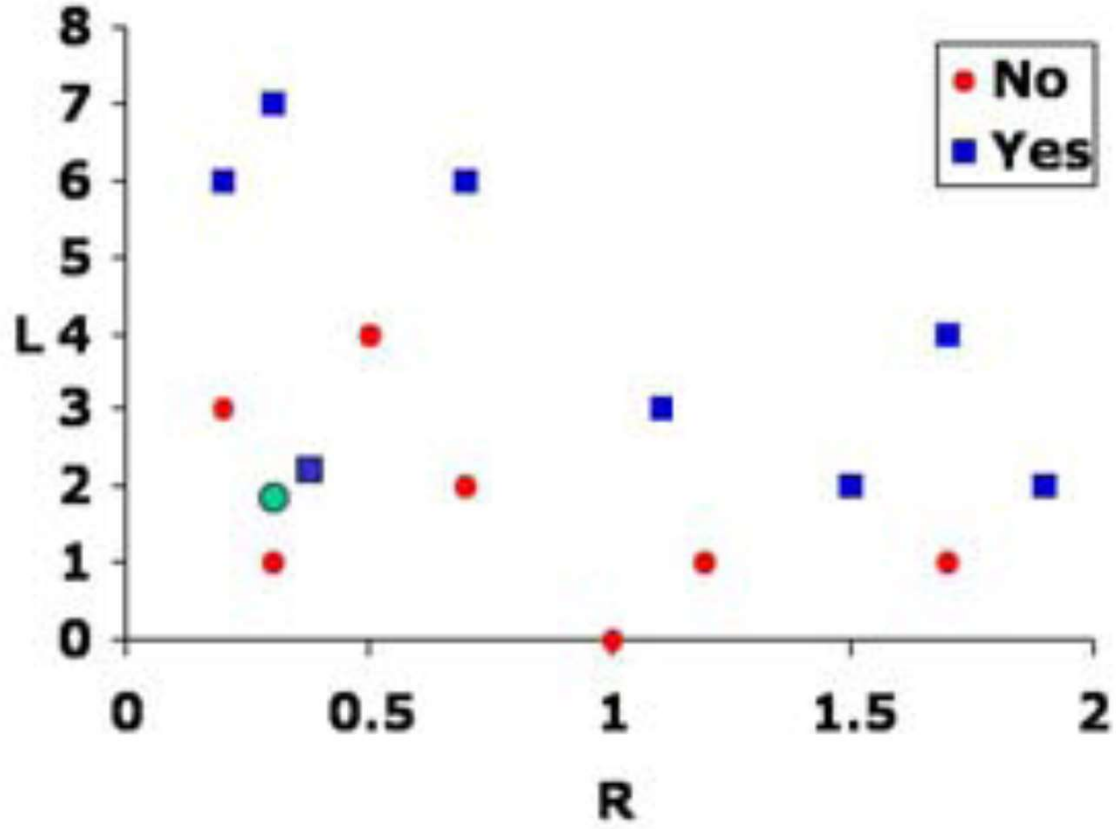


$$D(x^i, x^k) = \sqrt{\sum_j (L^i - L^k)^2 + (5R^i - 5R^k)^2}$$

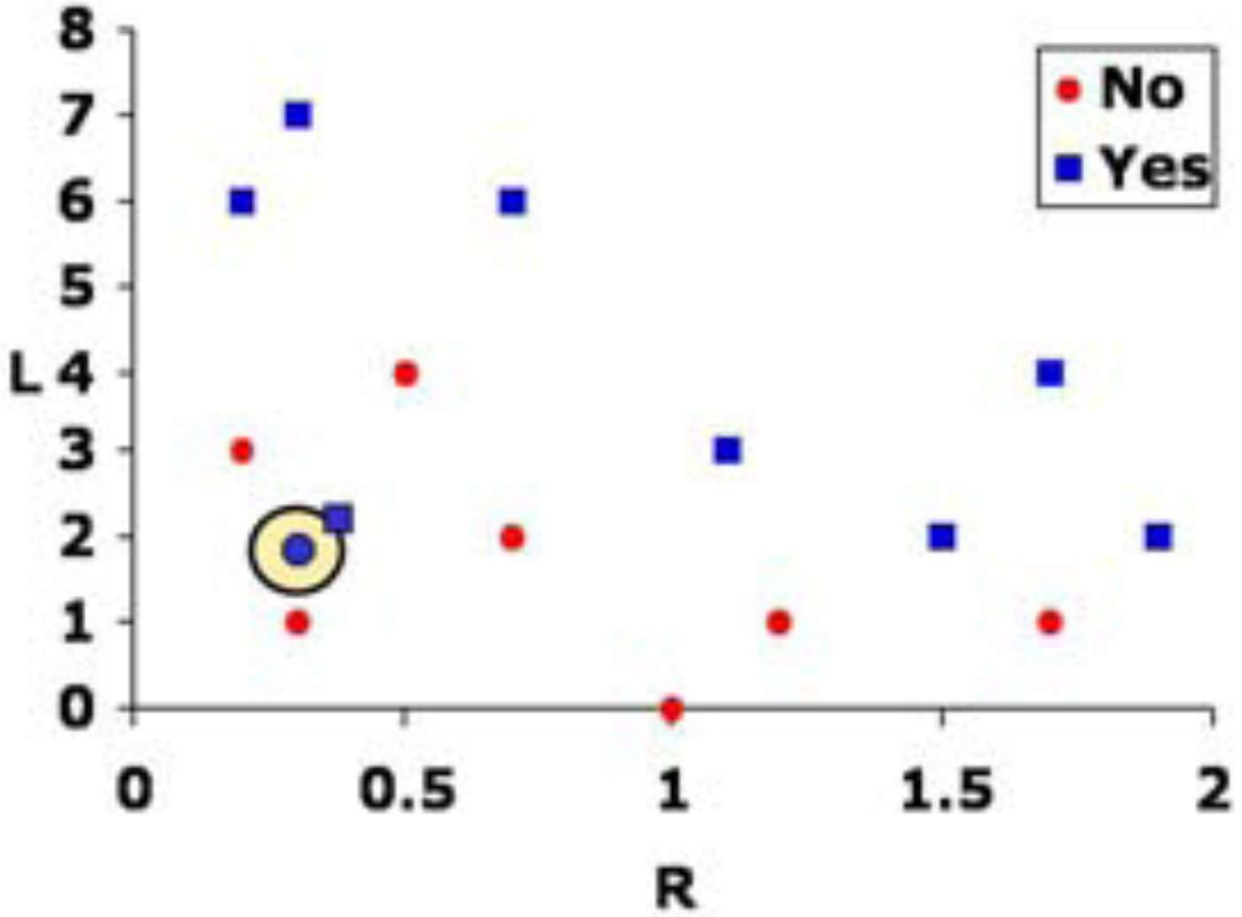
Time and space

- Learning is fast
- Lookup takes about m*n computations
  - storing data in a clever data structure (KD-tree) reduces this, on average, to log(m)*n
- Memory can fill up with all that data
  - delete points that are far away from the boundary

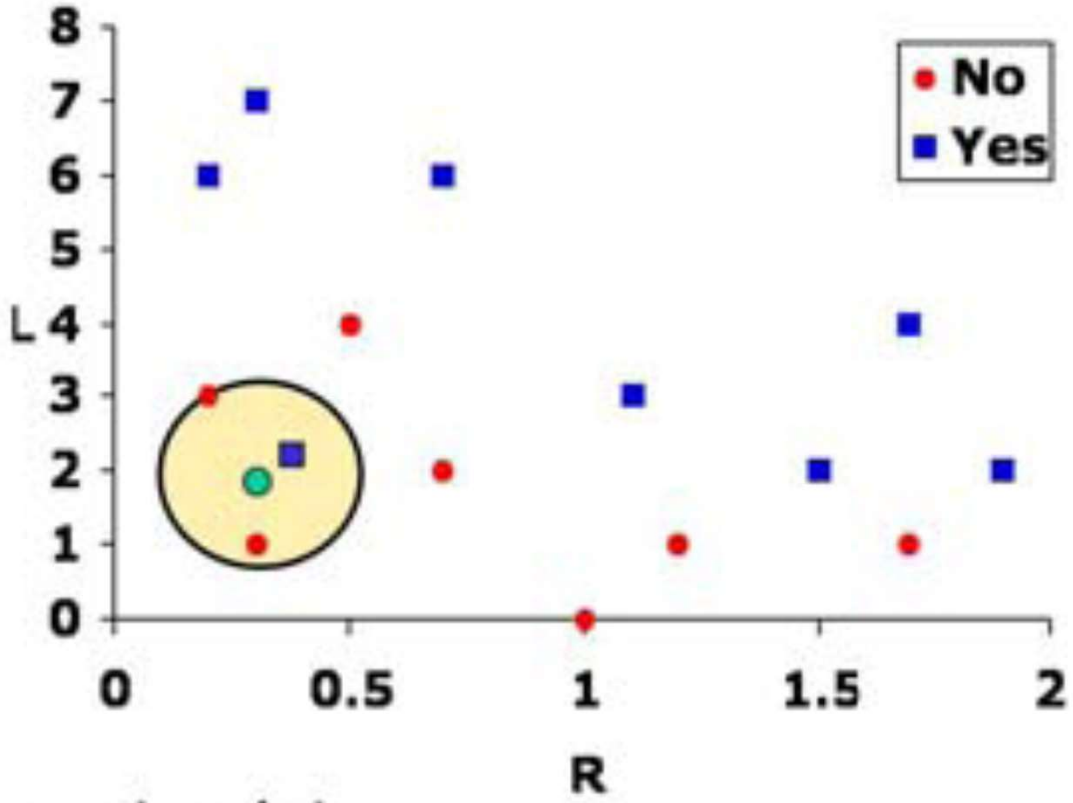
Noise



Noise



K-nearest neighbor



- Find the k nearest points
- Predict output according to the majority
- Choose k using cross-validation

Curse of dimensionality

- Nearest neighbor is great in low dimensions (up to about 6)
- As n increases, things get weird:
  - In high dimensions, almost all points are far away from one another
  - They're almost all near the boundaries
- Imagine sprinkling data points uniformly within a 10-dimensional unit cube
  - To capture 10% of the points, you'd need a cube with sides of length .63!
- Cure: feature selection or more global models

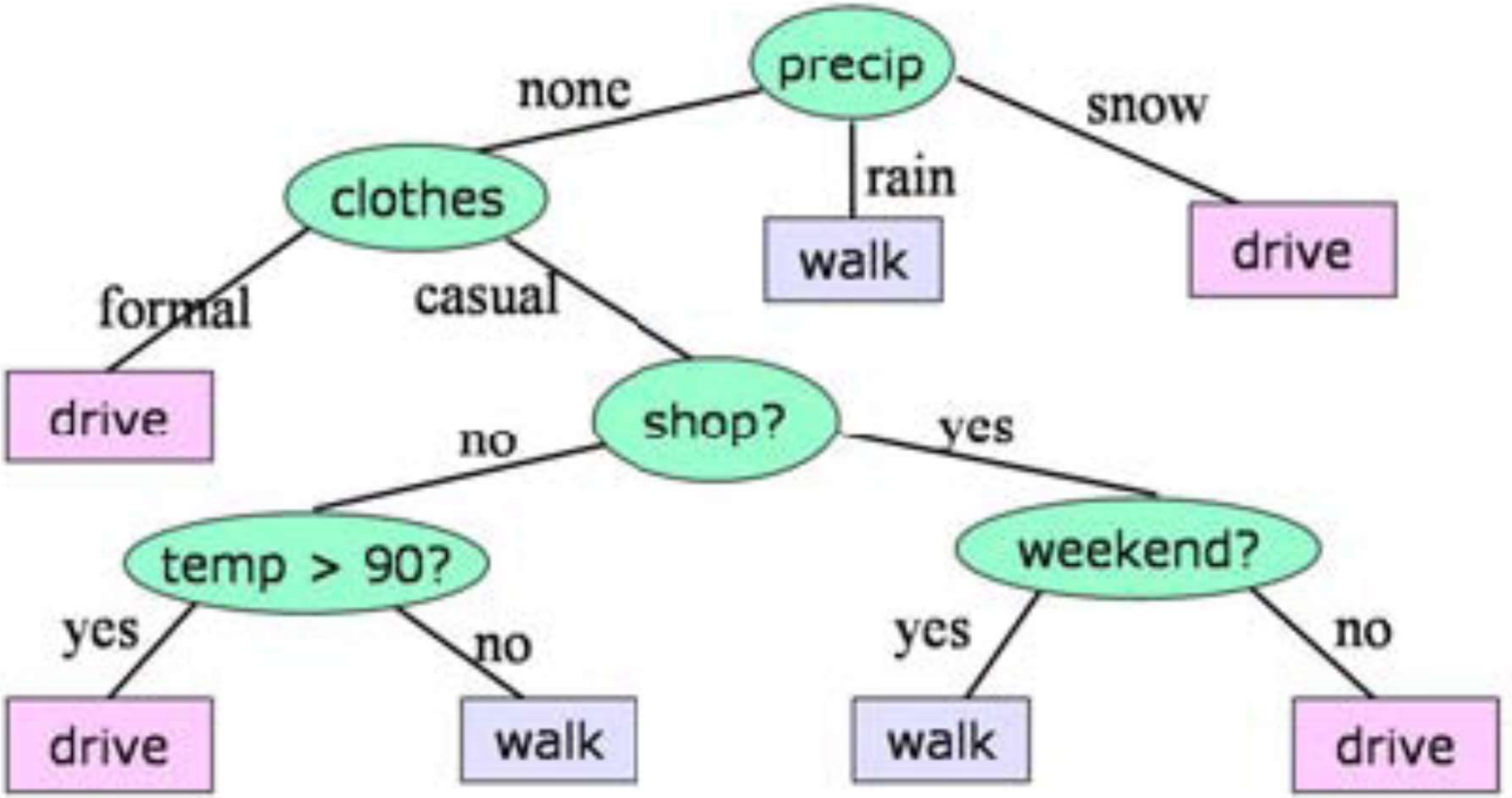
Test domains

- Heart Disease: predict whether a person has significant narrowing of the arteries, based on tests
  - 26 features
  - 297 data points

- Auto MPG: predict whether a car gets more than 22 miles per gallon, based on attributes of car
  - 12 features
  - 385 data points

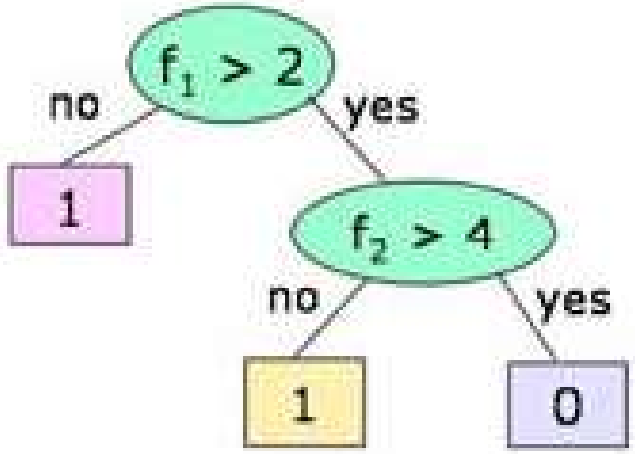
Decision trees

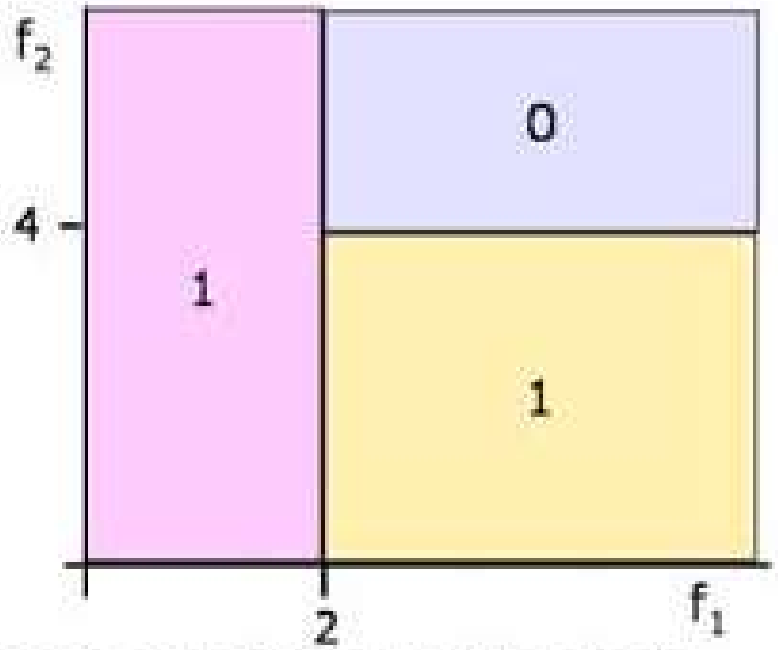
Use all the data to build a tree of questions with answers at the leaves



Numerical attributes

- Tests in nodes can be of the form x_j > constant
- Divides the space into axis-aligned rectangles

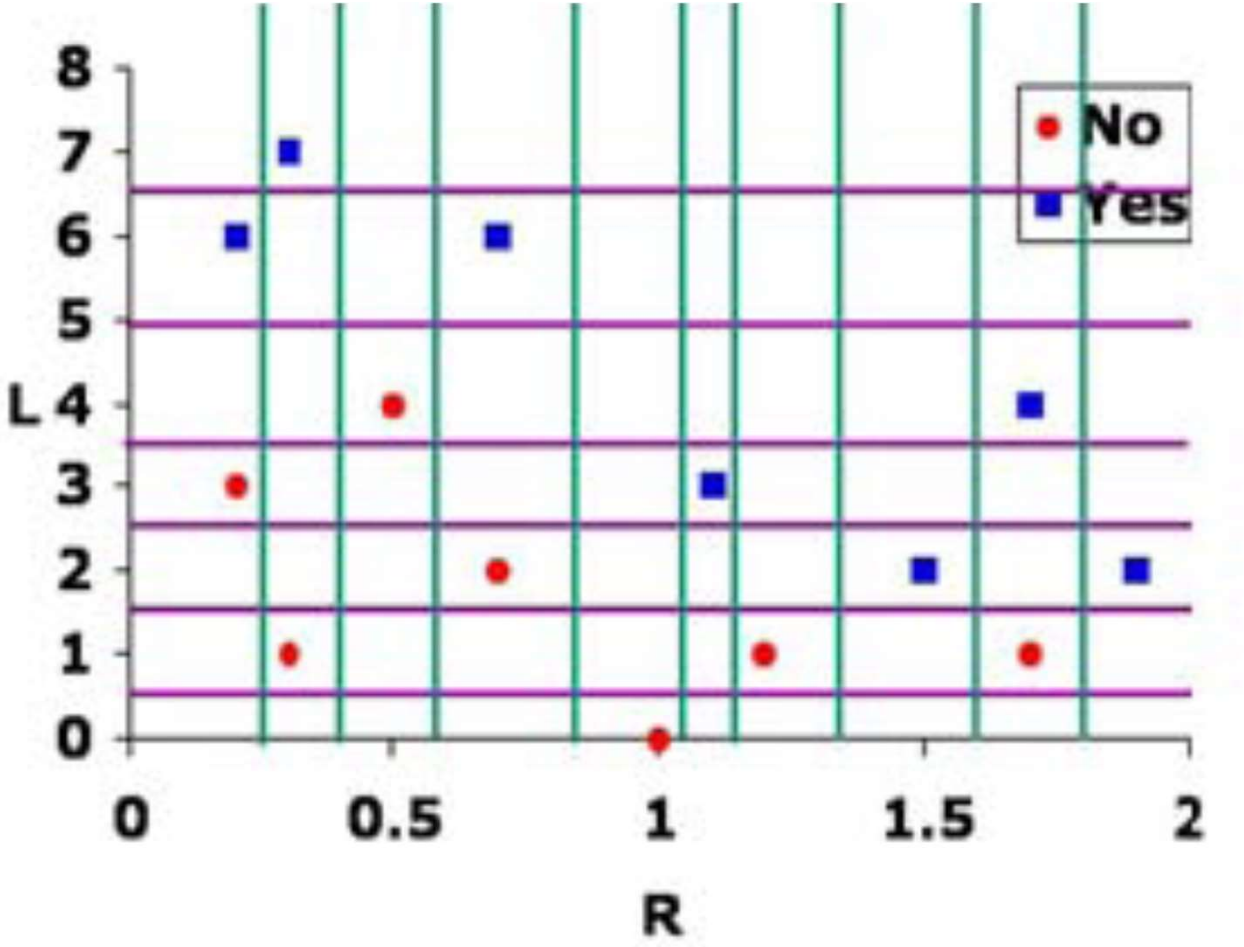




- Non-axis aligned hypotheses can be smaller but hard to find

Considering splits

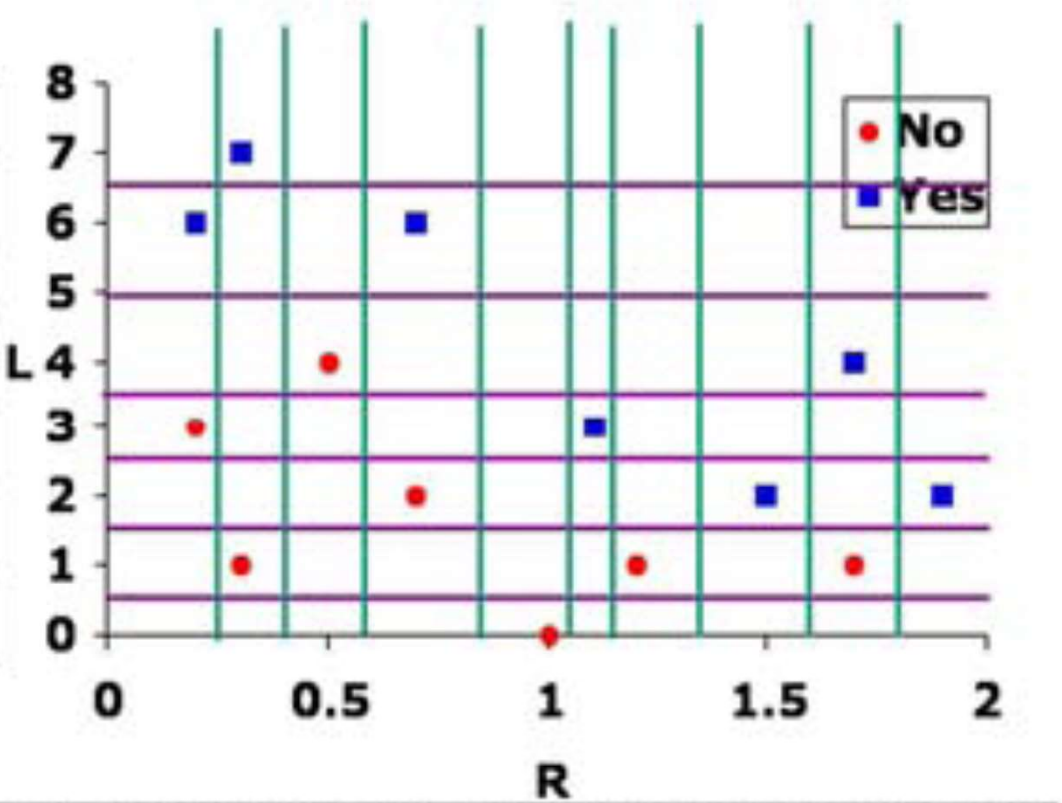
- Consider a split between each point in each dimension



Considering splits

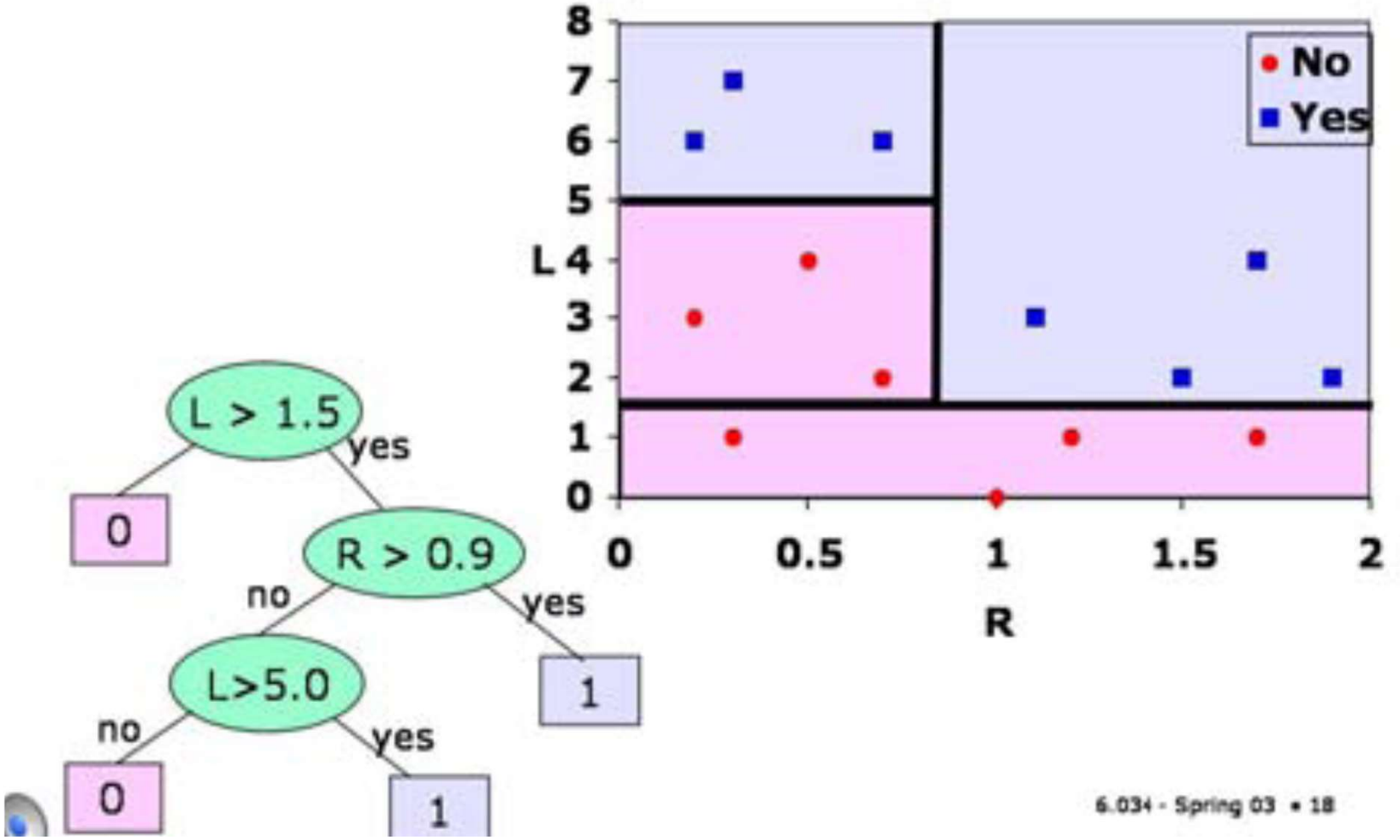
- Choose split that minimizes average entropy of child nodes

L<y	NL	PL	NR	PR	AE
6.5	7	6	0	1	0.93
5.0	7	4	0	3	0.74
3.5	6	3	1	4	0.85
2.5	5	2	2	5	0.86
1.5	4	0	3	7	0.63
0.5	1	0	6	7	0.93
	# neg to left	# pos to left	# neg to right	# pos to right	

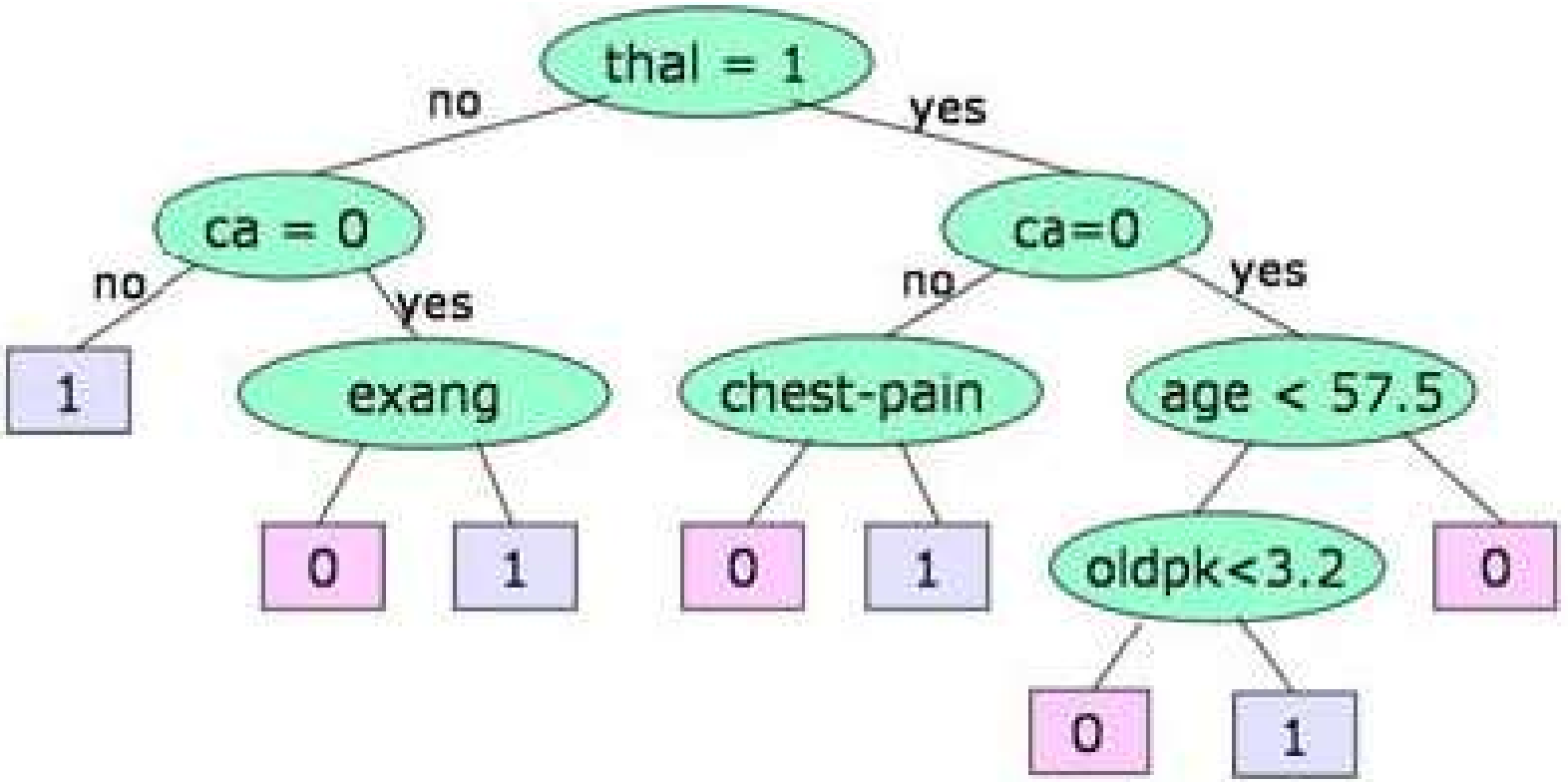


R<x	0.25	0.40	0.60	0.85	1.05	1.15	1.35	1.60	1.80
AE	1.00	1.00	0.98	0.98	0.94	0.98	0.92	0.98	0.92

Bankruptcy example

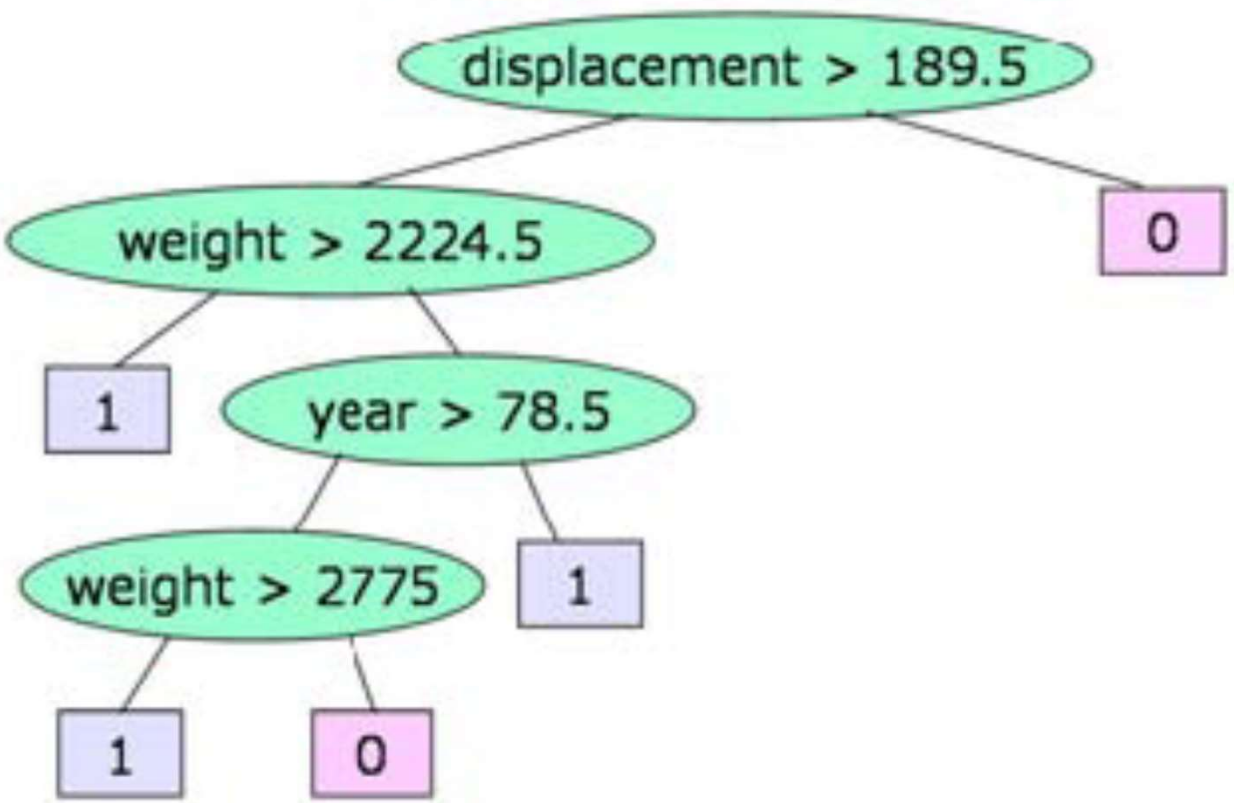


Heart disease

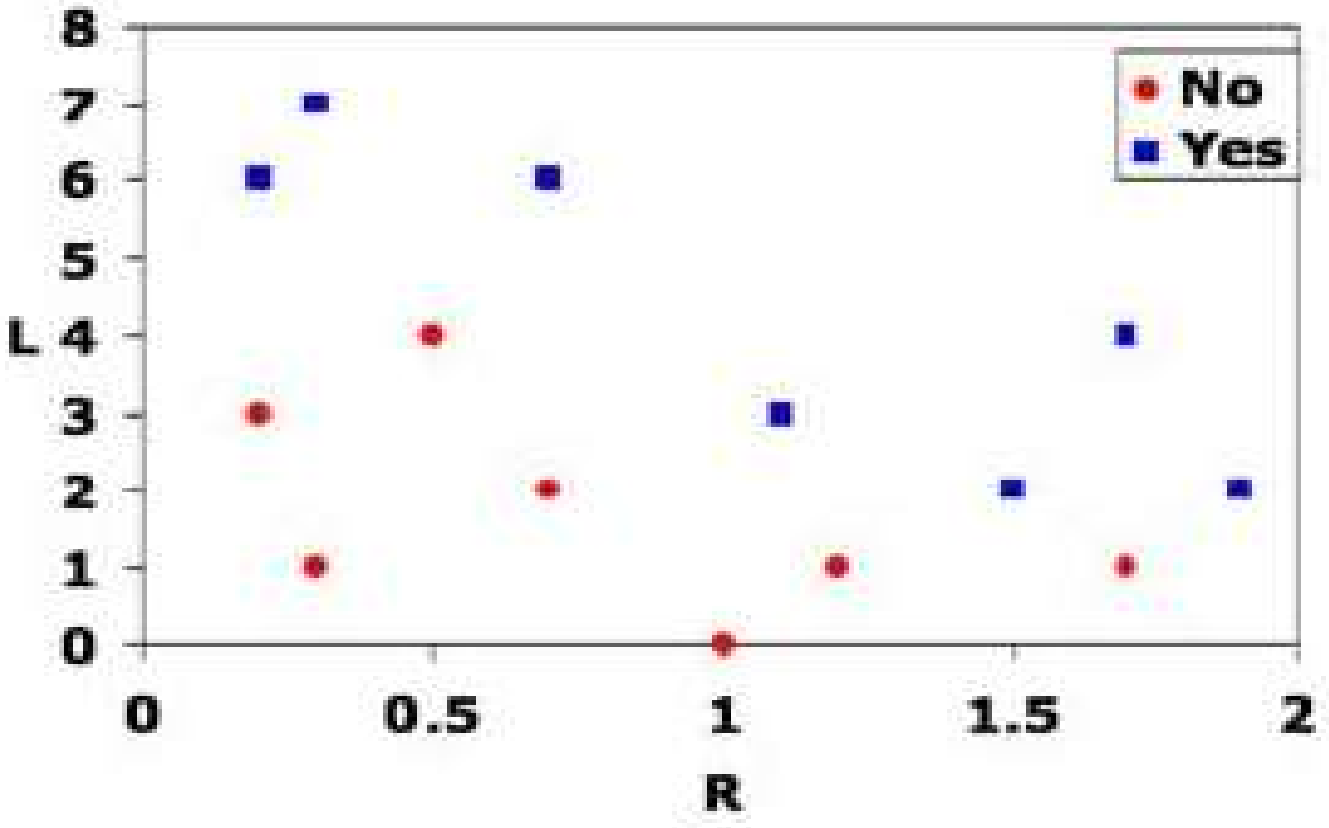


thal = 1: normal exercise thallium scintigraphy test
ca = 0: no vessels colored by fluoroscopy
exang: exercise induced angina
oldpk: feature of cardiogram

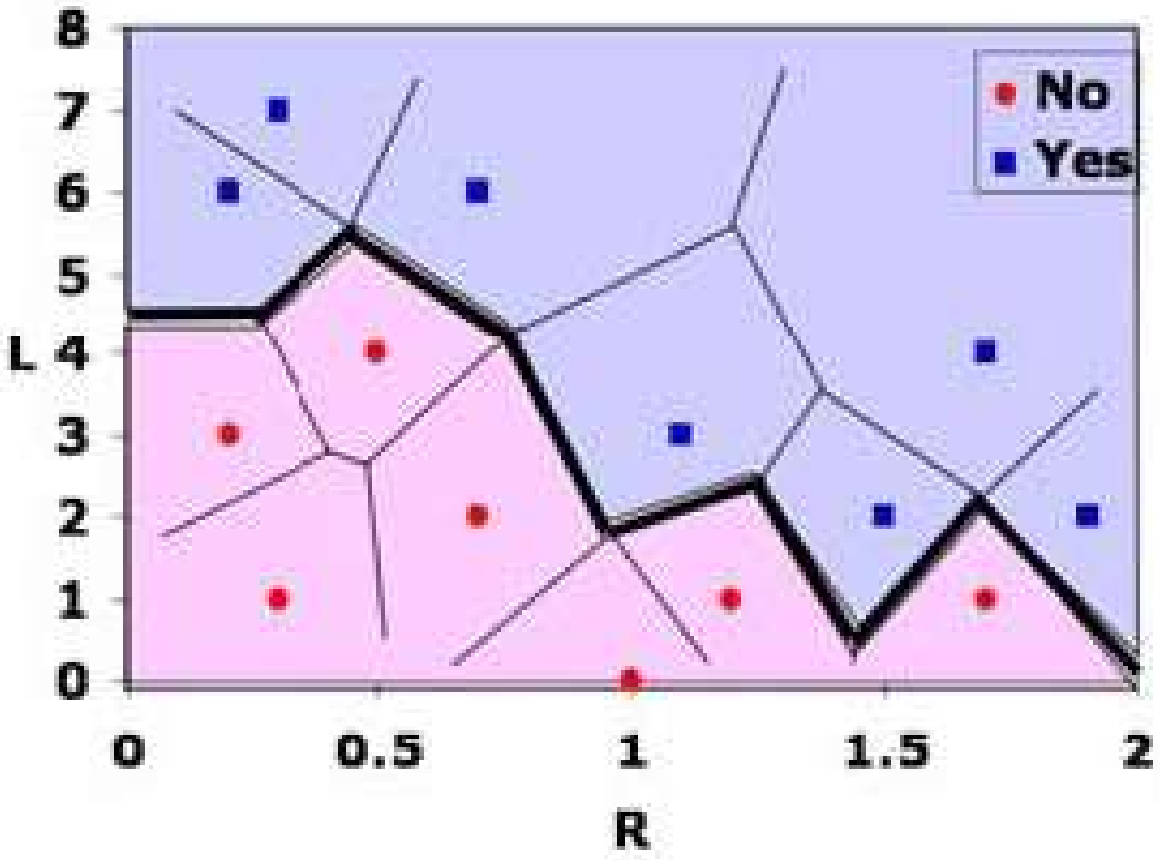
More than 22 MPG?



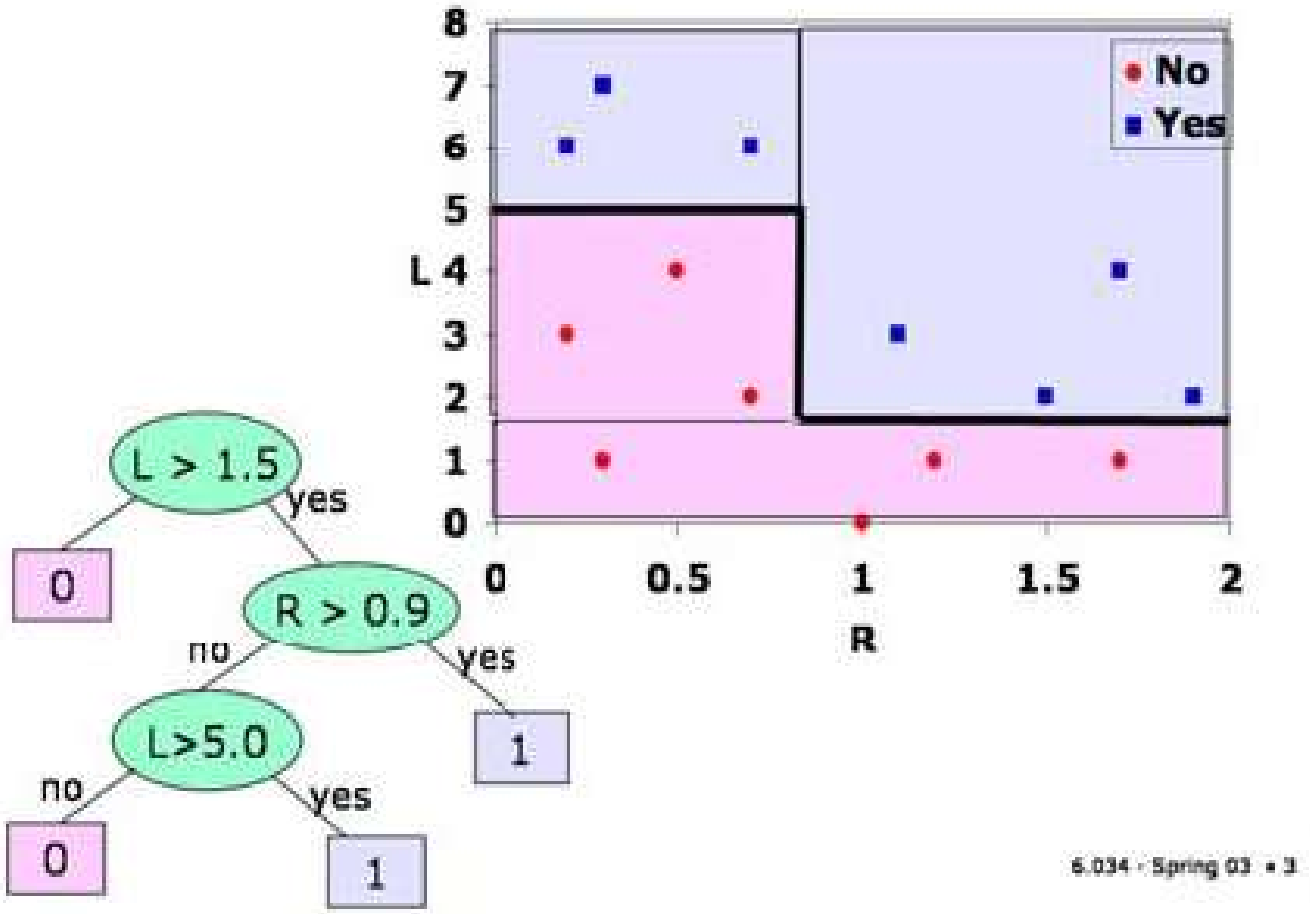
Bankruptcy example



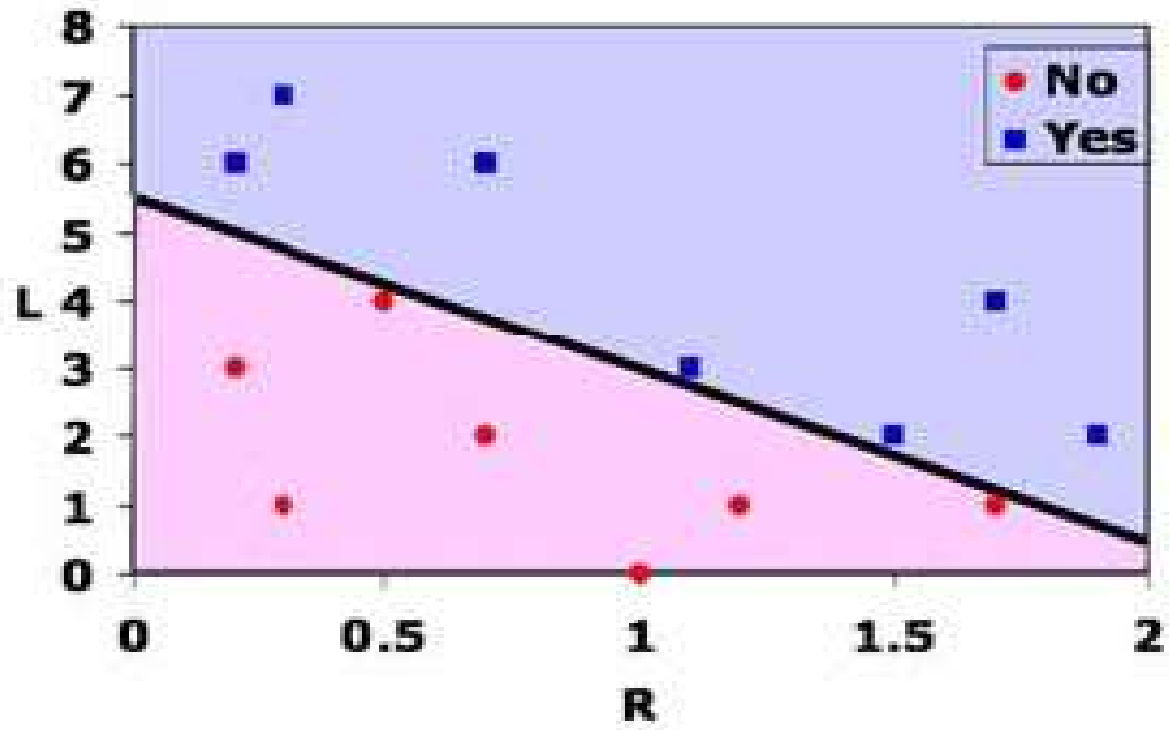
1-Nearest Neighbor hypothesis



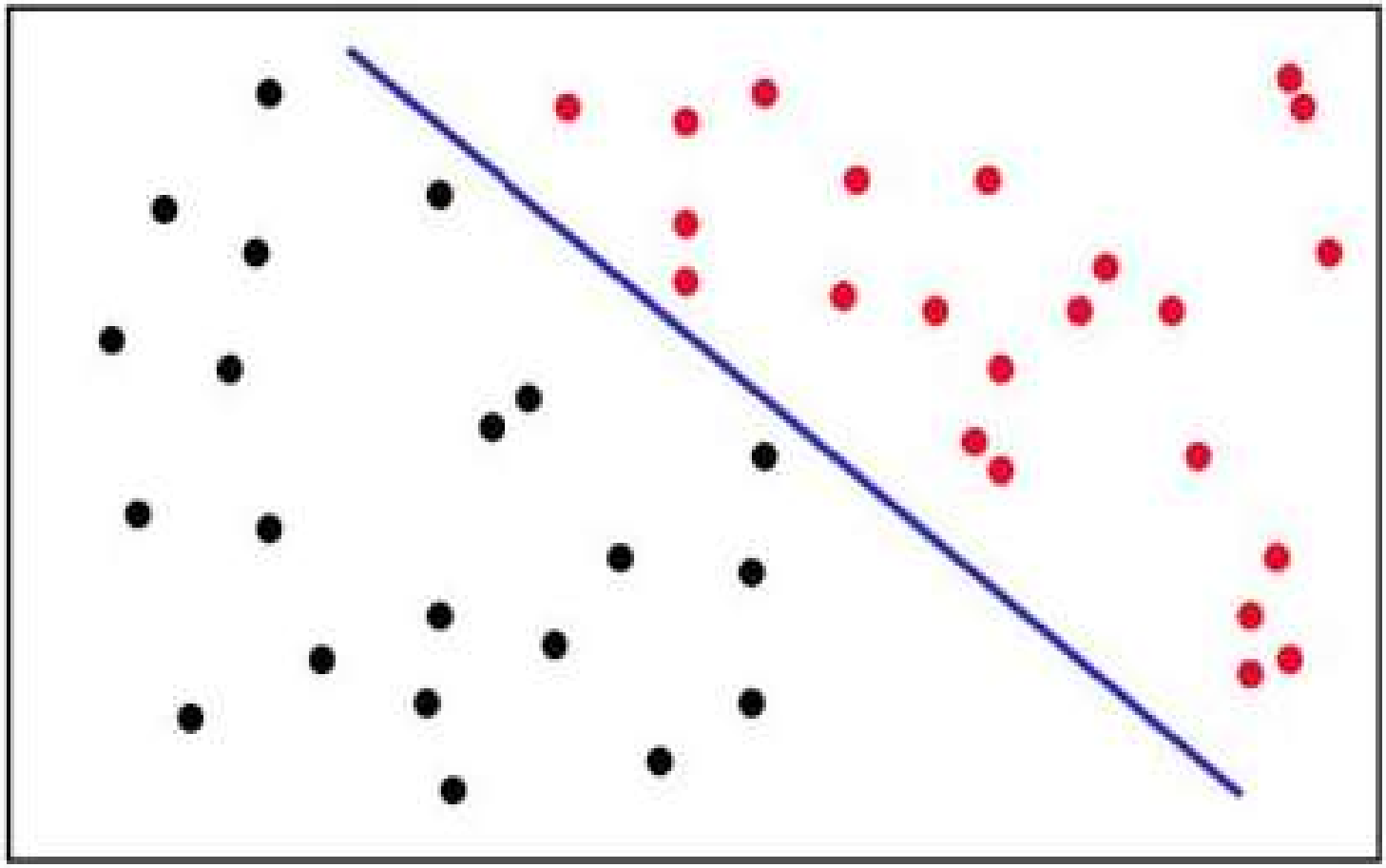
Decision tree hypothesis



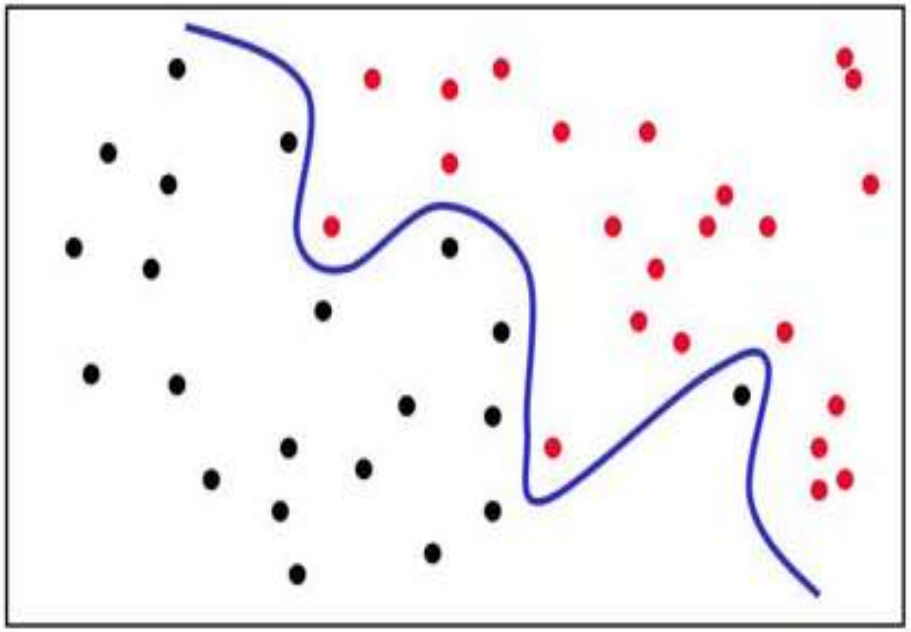
Linear hypothesis

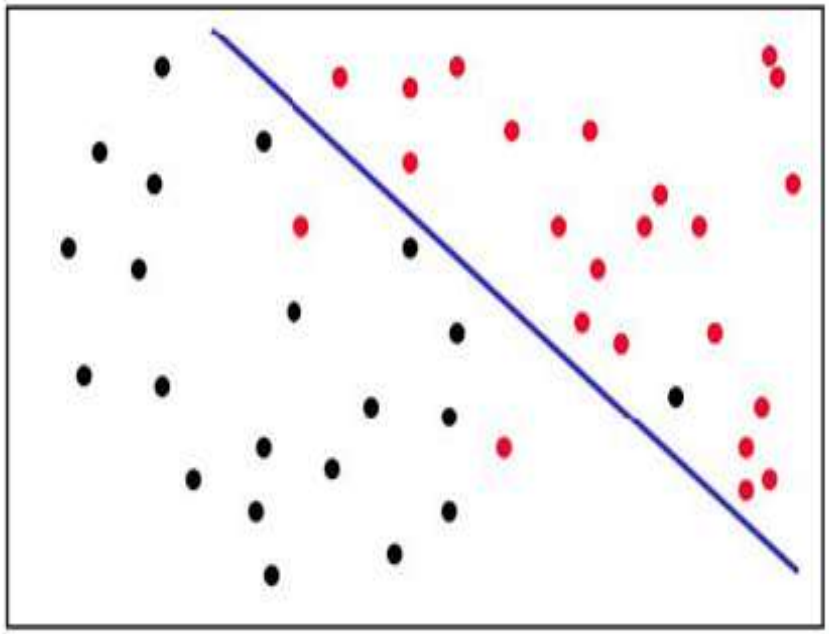


Linearly separable



Not linearly separable



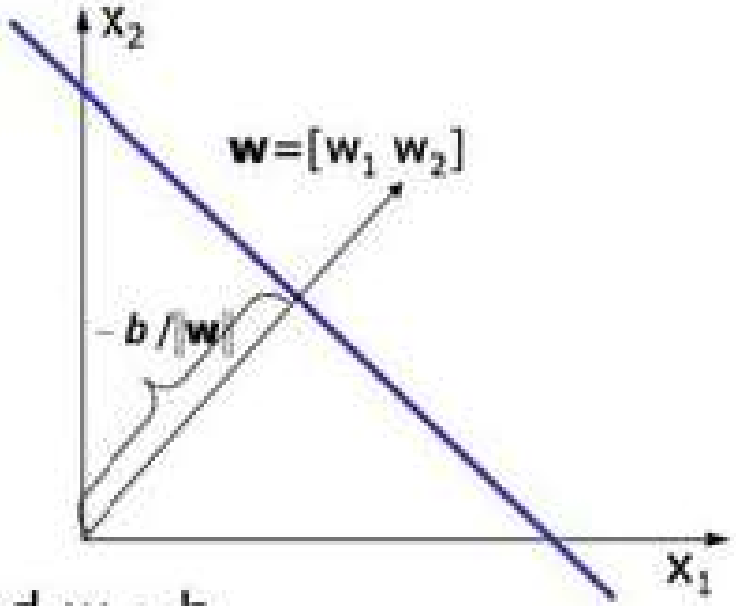


Linear hypothesis class

- Equation of a hyperplane in the feature space

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

$$\sum_{j=1}^{n} w_j x_j + b = 0$$



- **w**, b are to be learned

- A useful trick: let $x_0=1$ and $w_0=b$

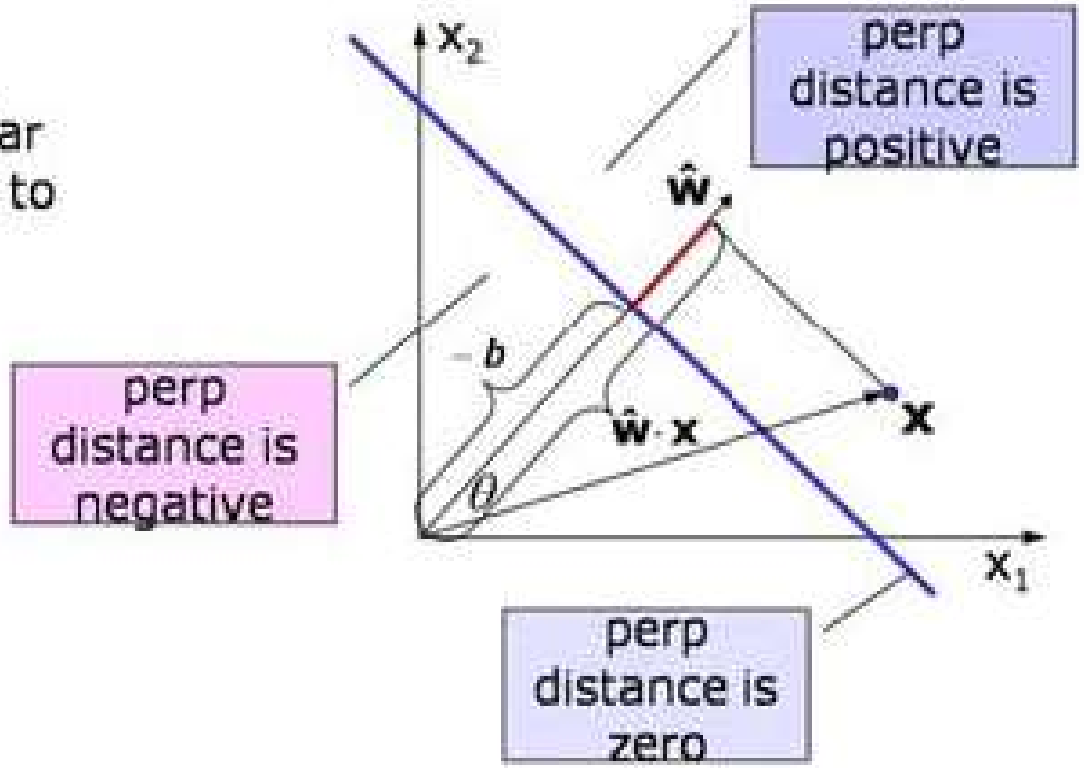
$$\overline{\mathbf{w}} \cdot \overline{\mathbf{x}} = 0$$

$$\sum_{j=0}^{n} w_j x_j = 0$$

Hyperplane geometry

$\hat{\mathbf{w}} \cdot \mathbf{x} + b$

signed perpendicular distance of point **x** to hyperplane.



recall: $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\| \cos\theta$

$$h(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x} + b) \equiv sign(\overline{\mathbf{w}} \cdot \overline{\mathbf{x}})$$

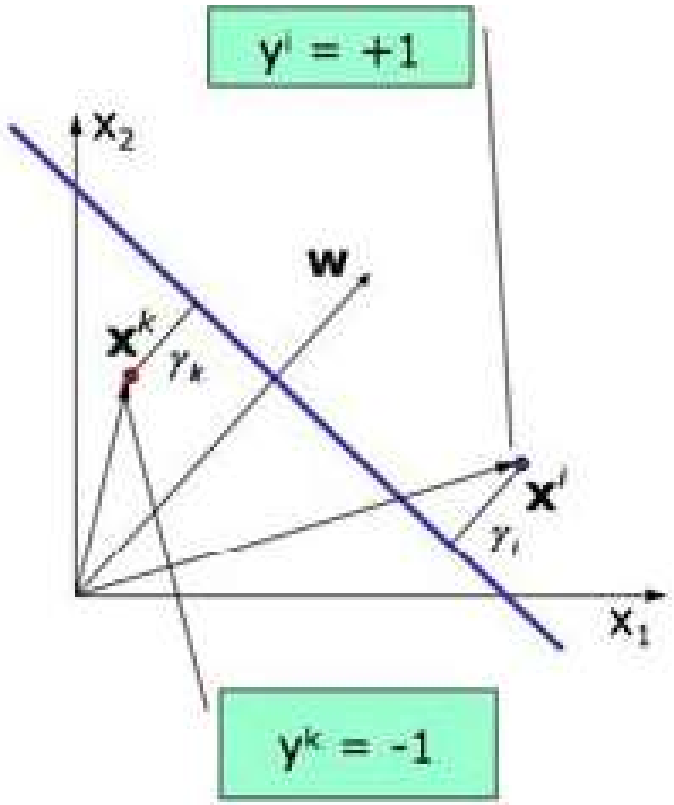
Margin:

$\gamma_i \equiv y^i(\mathbf{w} \cdot \mathbf{x}^i + b) \equiv y^i \mathbf{w} \cdot \mathbf{x}^i$

proportional to perpendicular distance of point $\mathbf{x}^i$ to hyperplane.

$\gamma_i > 0$: point is correctly classified (sign of distance = y^i)

$\gamma_i < 0$: point is incorrectly classified (sign of distance ≠ y^i)



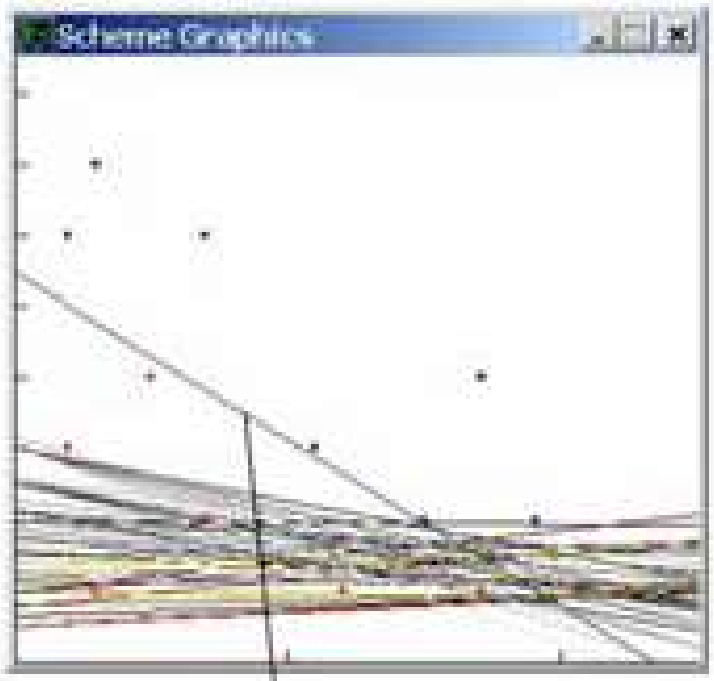
Perceptron algorithm

Rosenblatt, 1956

- **Pick initial weight vector (including b), e.g. [0 … 0]**
- **Repeat until all points correctly classified**
  - **Repeat for each point**
    - **Calculate margin ($y^i \overline{wx}^i$) for point i**
    - **If margin > 0, point is correctly classified**
    - **Else change weights to increase margin; change in weight proportional to $y^i \overline{x}^i$**

- Note that, if $y^i = 1$
  if $x_j^i > 0$ then w_j increased (increases margin)
  if $x_j^i < 0$ then w_j decreased (increases margin)
- And, similarly for $y^i = -1$
- Guaranteed to find separating hyperplane if one exists
- Otherwise, data are not linearly separable, loops forever

Bankruptcy example- 49 iterations



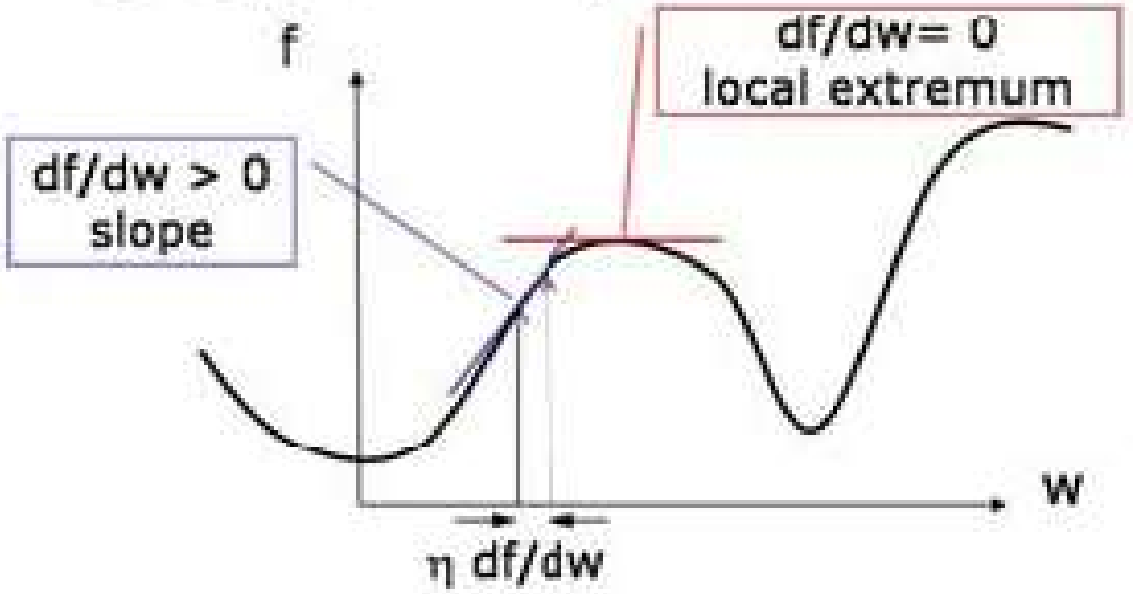
rate $\eta = 0.1$

Initial Guess:
w=[0.0 0.0 0.0]

Final Answer:
w=[-2.2 0.94 0.4]

Gradient Ascent

- Why pick $y^l \bar{\mathbf{x}}^l$ as increment to weights?
- To maximize scalar function of one variable f(w)
  - Pick initial w
  - Change w to w + η df/dw ($\eta > 0$, small)
  - until f stops changing (df/dw $\approx$ 0)

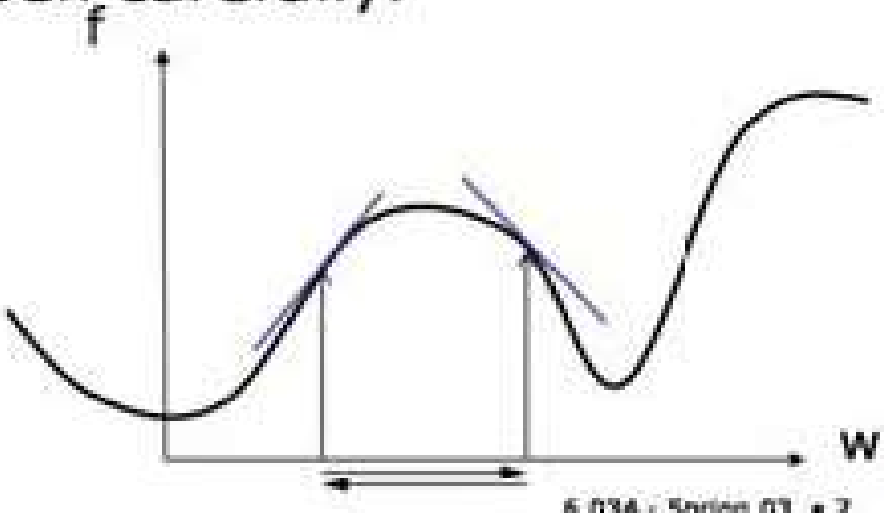


Gradient ascent/descent

- To maximize f(**w**)
  - Pick initial **w**
  - Change **w** to **w** + $\eta \nabla_{\mathbf{w}} f$ ($\eta > 0$, small)
  - until f stops changing ($\nabla_{\mathbf{w}} f \approx 0$)

$$\nabla_{\mathbf{w}} f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n} \right]$$

- Finds local maximum; global maximum if function is globally convex
- Rate (η) has to be chosen carefully.
  - Too small – slow convergence
  - Too big – oscillation



Perceptron training via gradient descent

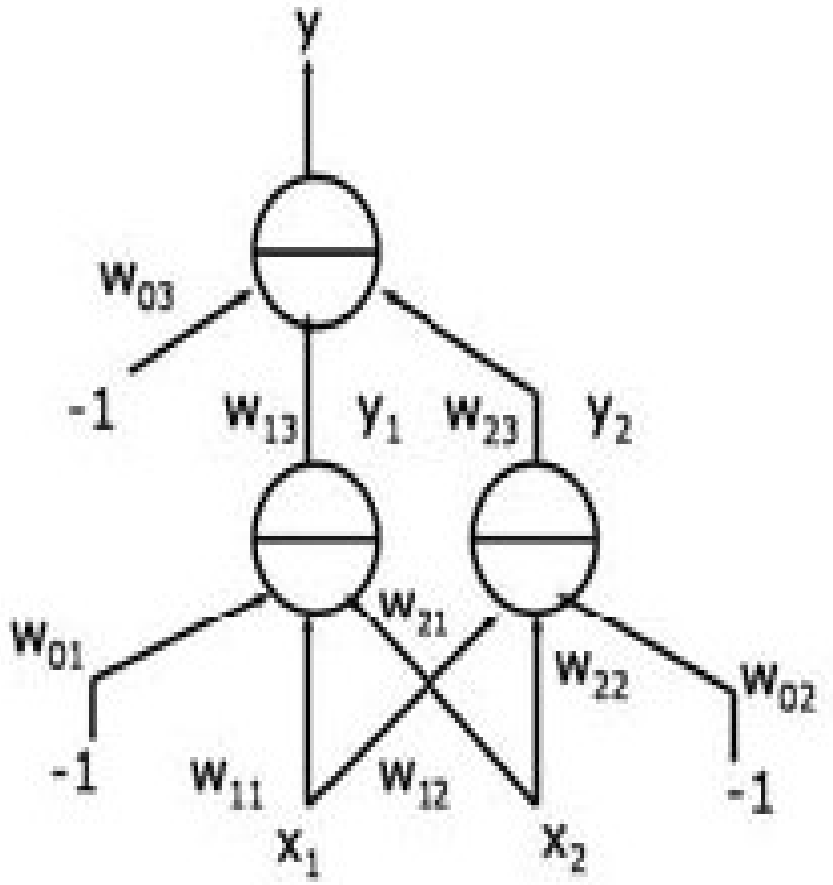
- Maximize sum of margins of misclassified points

$$f(\mathbf{w}) = \sum_{i \text{ misclassified}} y^i \overline{\mathbf{w}\mathbf{x}}^i$$

$$\nabla_{\mathbf{w}} f = \sum_{i \text{ misclassified}} y^i \overline{\mathbf{x}}^i$$

- Off-line training: Compute gradient as sum over all training points.
- On-line training: Approximate gradient by one of the terms in the sum: $y^i \overline{\mathbf{x}}^i$

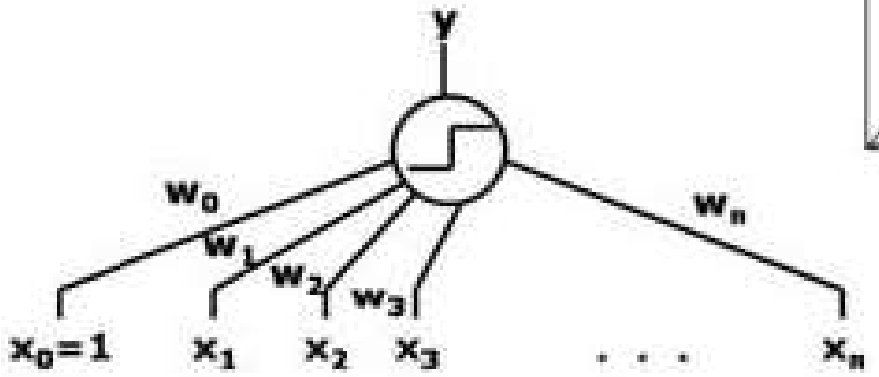
Artificial Neural Networks (Feedforward Nets)

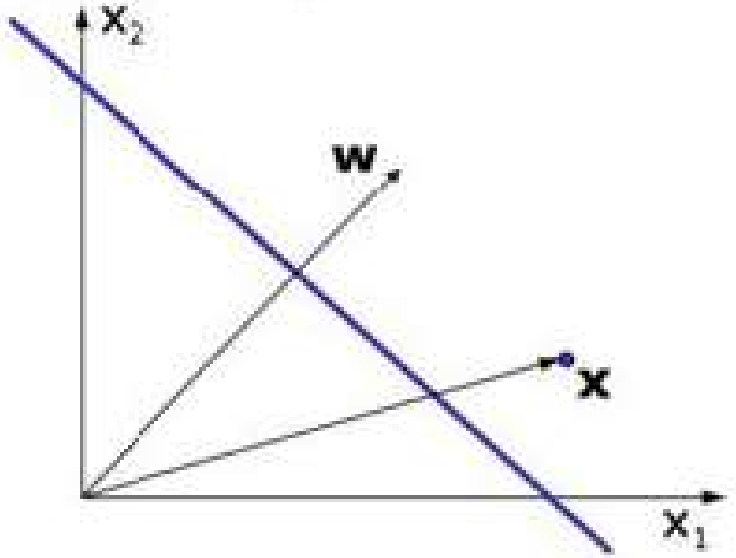


Single Perceptron Unit

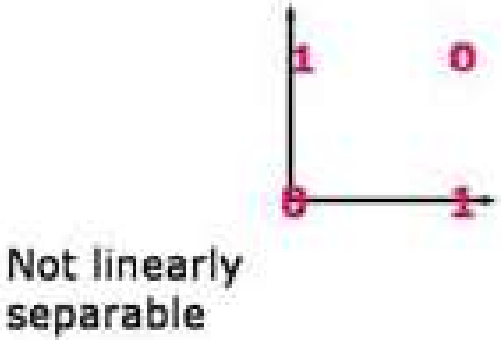
$$h(\mathbf{x}) = \theta(\mathbf{w} \cdot \mathbf{x} + b) \equiv \theta(\overline{\mathbf{w}} \cdot \overline{\mathbf{x}})$$

$$\theta(z) = \begin{cases} 1 & z \geq 0 \\ 0 & else \end{cases}$$

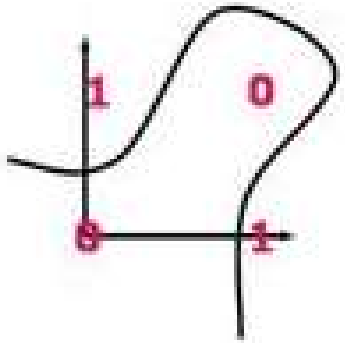


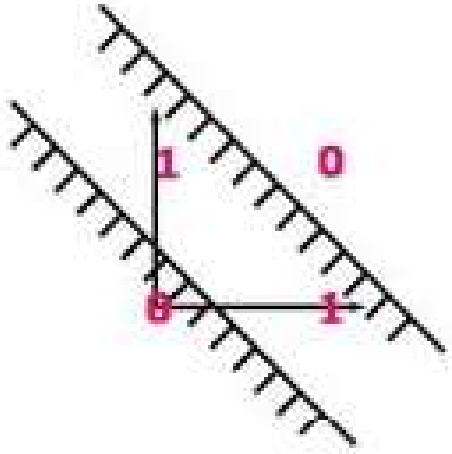


Beyond linear separability

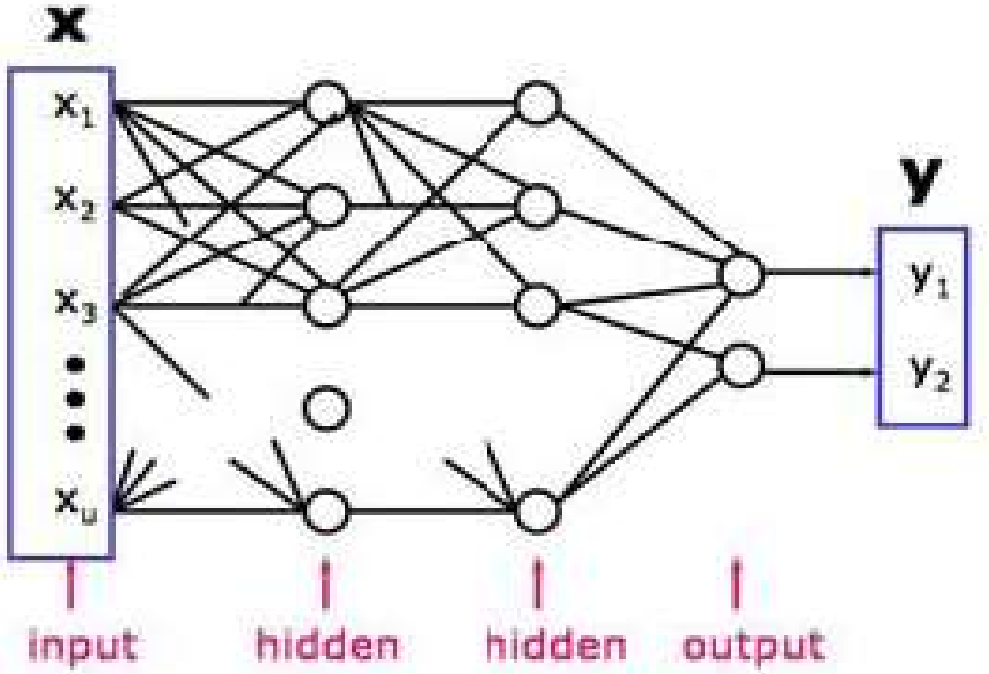


Not linearly separable



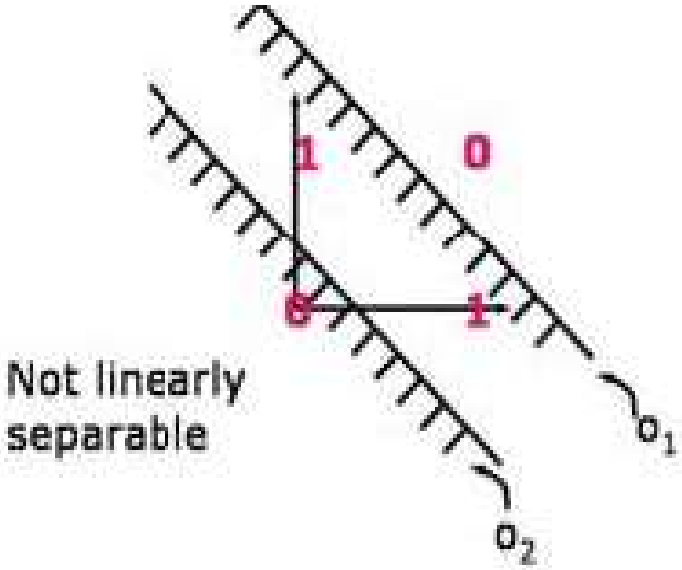


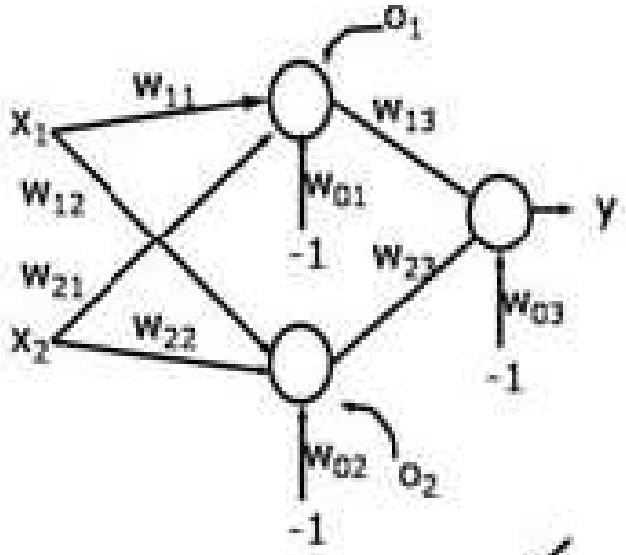
Multi-layer perceptron



- More powerful than single layer.
- Lower layers transform the input problem into more tractable (linearly separable) problems for subsequent layers.

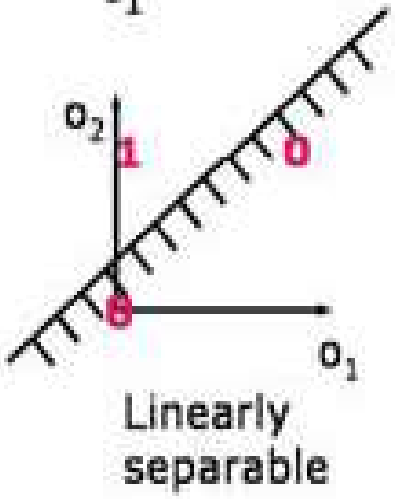
Multilayer perceptron





$w_{01} = 3/2$	$w_{11} = w_{12} = 1$
$w_{02} = 1/2$	$w_{21} = w_{22} = 1$
$w_{03} = 1/2$	$w_{31} = -1, w_{32} = 1$

x_1	x_2	o_1	o_2	y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



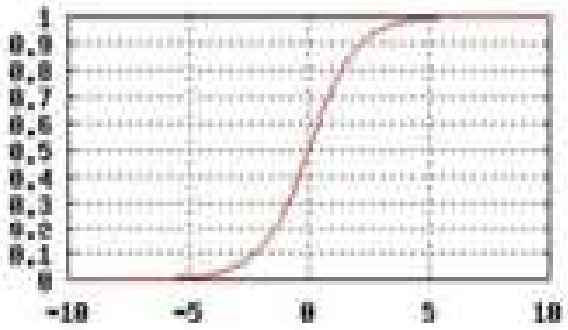
Multilayer perceptron learning

- Any set of training points can be separated by a three-layer perceptron network.
- "Almost any" set of points separable by two-layer perceptron network.
- But, no efficient learning rule is known.

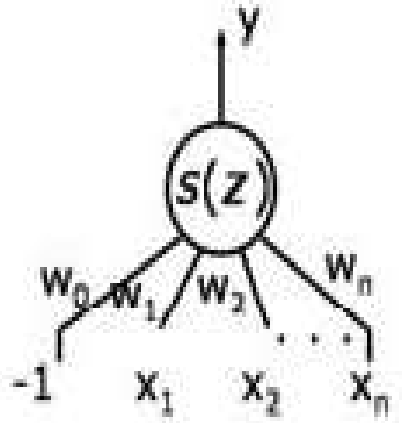
- Could we use gradient ascent/descent?
- We would need smoothness: small change in weights produces small change in output.
- Threshold function is not smooth.

- Use a smooth threshold function!

Sigmoid unit



$$z = \sum_{i}^{n} w_i x_i \quad s(z) = \frac{1}{1 + e^{-z}}$$



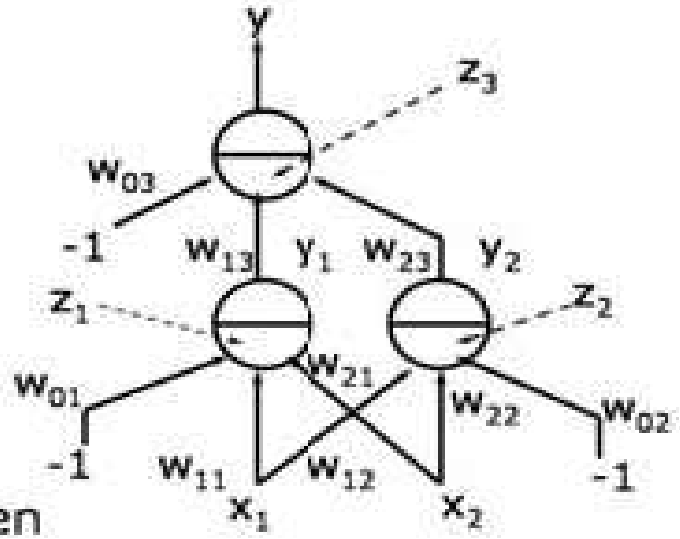
Training

$y(\mathbf{x}, \mathbf{w})$

w is a vector of weights

x is a vector of inputs

y' is desired output:



Error over the training set for a given weight vector:

$$E = \frac{1}{2} \sum_i (y(\mathbf{x}', \mathbf{w}) - y')^2$$

Our goal is to find weight vector that minimizes error

$$y = s(\overbrace{w_{13} s(\overbrace{w_{11} x_1 + w_{21} x_2 - w_{01}}^{z_1}) + w_{23} s(\overbrace{w_{12} x_1 + w_{22} x_2 - w_{02}}^{z_2}) - w_{03}}^{z_3})$$

Gradient descent

$$E = \frac{1}{2}\sum_i (y(\mathbf{x}^i, \mathbf{w}) - y^i)^2$$

Error on training set

$$\nabla_{\mathbf{w}} E = \sum_i (y(\mathbf{x}^i, \mathbf{w}) - y^i)\nabla_{\mathbf{w}} y(\mathbf{x}^i, \mathbf{w})$$

Gradient of Error

$$\nabla_{\mathbf{w}} y = \left[\frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n}\right]$$

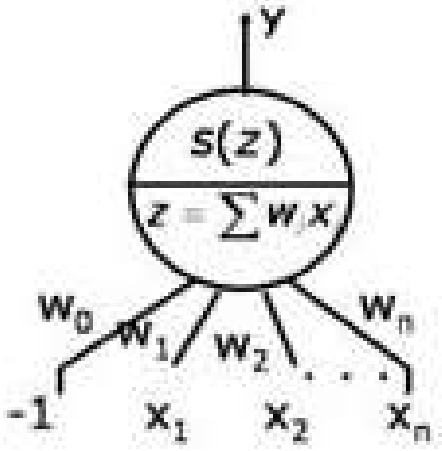
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E$$

Gradient Descent

Gradient descent – single unit

$$\nabla_w y = \left[\frac{\partial y}{\partial w_1}, \cdots, \frac{\partial y}{\partial w_n}\right]$$

$$z = \sum_i^n w_i x_i \quad y = s(z) = \frac{1}{1+e^{-z}}$$



$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z} x_i$$

$$w_i \leftarrow w_i - \eta(y - y^m)\frac{\partial s(z)}{\partial z} x_i$$

$$\delta \equiv \frac{\partial E}{\partial z} = (y - y^m)\frac{\partial s(z)}{\partial z}$$

$$\Delta w_i = -\eta \delta x_i$$

Delta Rule

Derivative of the sigmoid

$$s(z) = \frac{1}{1+e^{-z}}$$

$$\frac{ds(z)}{dz} = \frac{d}{dz}\left[(1+e^{-z})^{-1}\right]$$

$$= [-(1+e^{-z})^{-2}][-e^{-z}]$$

$$= \left[\frac{1}{1+e^{-z}}\right]\left[\frac{e^{-z}}{1+e^{-z}}\right]$$

$$= s(z)(1-s(z))$$

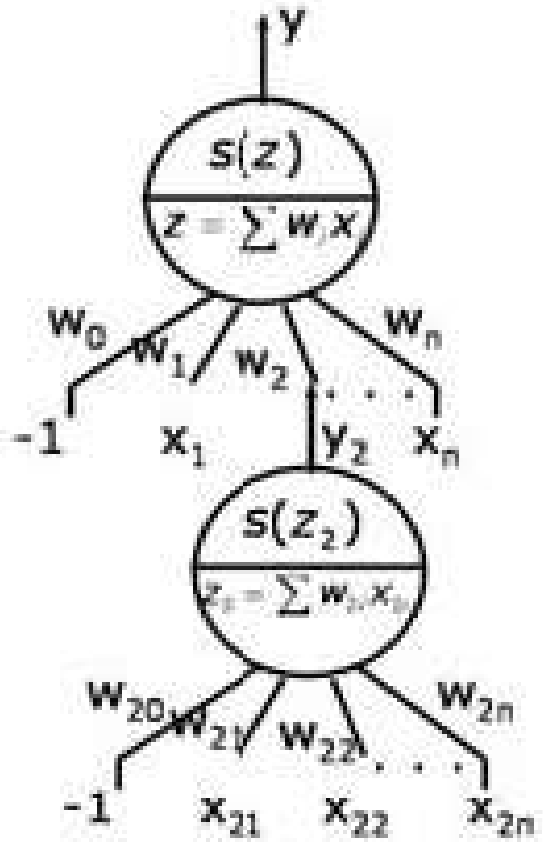
Gradient of unit output

$$\nabla_w y = \left[\frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n}\right]$$

$$z = \sum_i^n w_i x_i \qquad y = s(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_i} = \boxed{\frac{\partial s(z)}{\partial z} x_i}$$

$$\frac{\partial y}{\partial w_{ji}} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_{ji}} = \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_{ji}} = \boxed{\frac{\partial s(z)}{\partial z} w_j \frac{\partial y_j}{\partial w_{ji}}}$$



Base Case

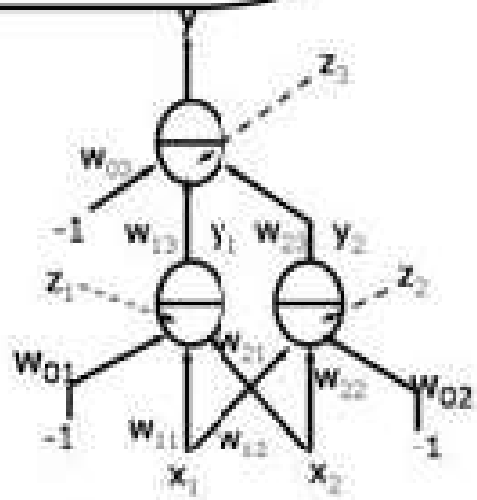
Recursion

Gradient of error

$$E = \frac{1}{2}\sum_i (y(\mathbf{x}', \mathbf{w}) - y')^2$$

$$y = s(\overbrace{w_{13}\overbrace{s(w_{11}x_1 + w_{21}x_2 - w_{01})}^{z_1} + w_{23}\overbrace{s(w_{12}x_1 + w_{22}x_2 - w_{02})}^{z_2} - w_{03}}^{z_3})$$

$$\frac{\partial E}{\partial w_j} = (y - y')\frac{\partial y}{\partial w_j}$$



$$\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = \frac{\partial y}{\partial z_3}y_1$$

$$\frac{\partial y}{\partial w_{11}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{11}} = \frac{\partial y}{\partial z_3}\left(w_{13}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}\right) = \frac{\partial y}{\partial z_3}\left(w_{13}\frac{\partial y_1}{\partial z_1}x_1\right)$$

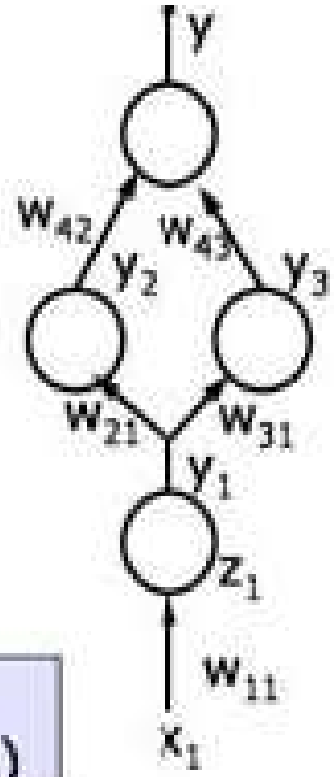
Gradient of Unit Output

$$\frac{\partial y}{\partial w_{11}} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_{11}}$$

$$= \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_{11}}$$

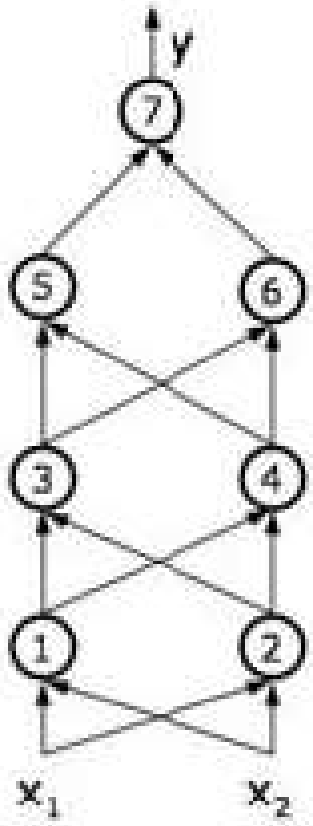
$$= \frac{\partial s(z)}{\partial z}\left(w_{42}\frac{\partial y_2}{\partial w_{11}} + w_{43}\frac{\partial y_3}{\partial w_{11}}\right)$$

Recursion (more general)



A change in w_{11} affects the error via a change in y_1, which affects y_2 **and** y_3

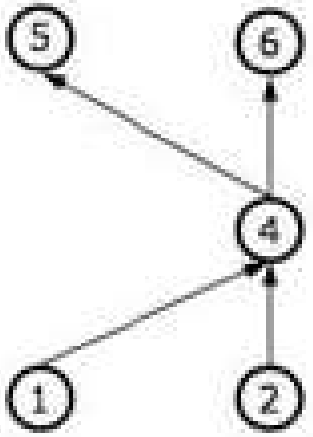
Generalized delta rule



$$\delta_j = \frac{\partial E}{\partial z_j}$$

$$\delta_j = \frac{ds(z_j)}{dz_j} \sum_k \delta_k w_{j \to k}$$

$$\Delta w_{i \to j} = -\eta \delta_j y_i$$



$$\delta_4 = \frac{ds(z_4)}{dz_4}\left(\delta_5 w_{4 \to 5} + \delta_6 w_{4 \to 6}\right)$$

$$\Delta w_{1 \to 4} = -\eta \delta_4 y_1$$

$$\Delta w_{2 \to 4} = -\eta \delta_4 y_2$$

Backpropagation

An efficient method of implementing gradient descent for neural networks

$$w_{i\to j} = w_{i\to j} - \eta\delta_j y_i$$ **Descent rule**

$$\delta_j = \frac{ds(z_j)}{dz_j}\sum_k \delta_k w_{j\to k}$$ **Backprop rule**

y_i is x_i for input layer

1. Initialize weights to small random values
2. Choose a random sample input feature vector
3. Compute total input (z_j) and output (y_j) for each unit (forward prop)
4. Compute δ_n for output layer
$$\delta_n = \frac{ds(z_n)}{dz_n}(y_n - y_n^m) = y_n(1-y_n)(y_n - y_n^m)$$
5. Compute δ_j for all preceding layers by backprop rule
6. Compute weight change by descent rule (repeat for all weights)

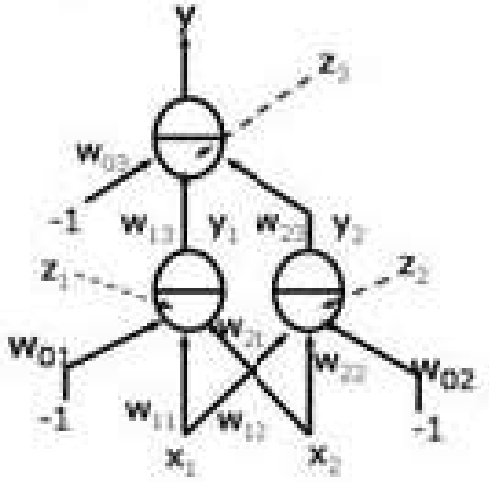
Backpropagation example

First do forward propagation:
Compute z_i and y_i given x_k, w_{ij}

$$\delta_3 = y(1-y)(y-y^m)$$

$$\delta_2 = y_2(1-y_2)\delta_3 w_{23}$$

$$\delta_1 = y_1(1-y_1)\delta_3 w_{13}$$



$$w_{03} = w_{03} - r\delta_3(-1)$$

$$w_{02} = w_{02} - r\delta_2(-1)$$

$$w_{01} = w_{01} - r\delta_1(-1)$$

$$w_{13} = w_{13} - \eta\delta_3 y_1 \qquad w_{23} = w_{23} - \eta\delta_3 y_2$$

$$w_{12} = w_{12} - \eta\delta_2 x_1 \qquad w_{22} = w_{22} - \eta\delta_2 x_2$$

$$w_{11} = w_{11} - \eta\delta_1 x_1 \qquad w_{21} = w_{21} - \eta\delta_1 x_2$$

Compare to the direct derivation earlier

Note that all computations are local!

Training neural nets

Given: Data set, desired outputs and a neural net with m weights. Find a setting for the weights that will give good predictive performance on new data. Estimate expected performance on new data.

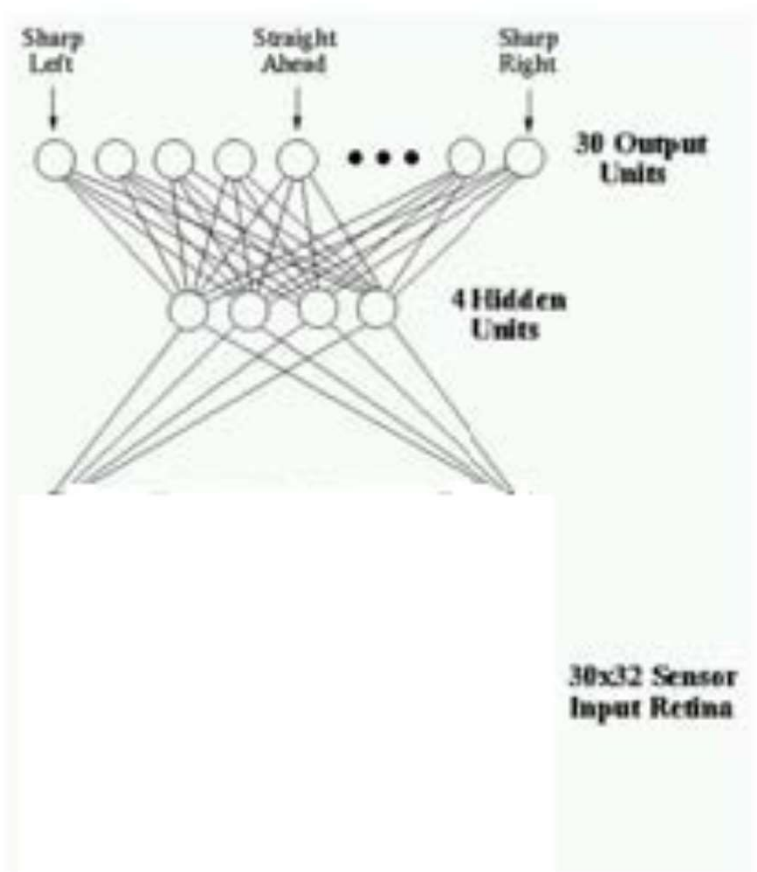
1. Split data set (randomly) into three subsets:
   - Training set – used for picking weights
   - Validation set – used to stop training
   - Test set – used to evaluate performance
2. Pick random, small weights as initial values
3. Perform iterative minimization of error over training set.
4. Stop when error on validation set reaches a minimum (to avoid overfitting).
5. Repeat training (from step 2) several times (avoid local minima)
6. Use best weights to compute error on test set, which is estimate of performance on new data. Do not repeat training to improve this.

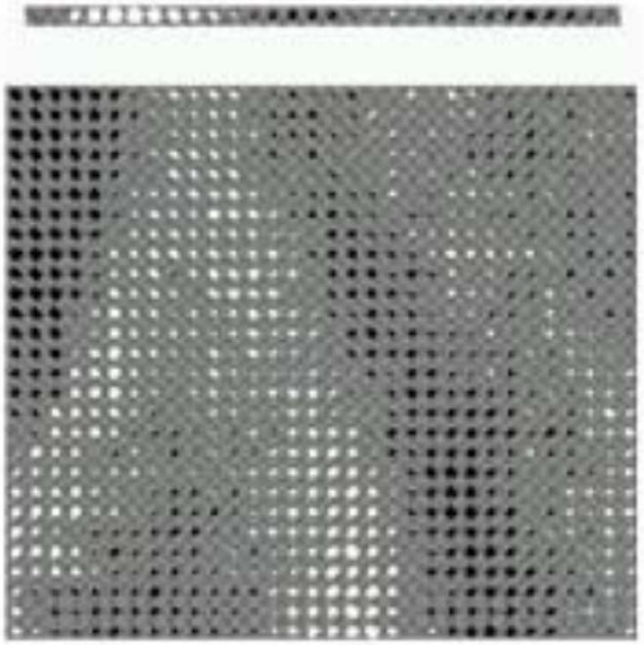
Can use cross-validation if data set is too small to divide into three subsets.

Applications

ALVINN steers on highways

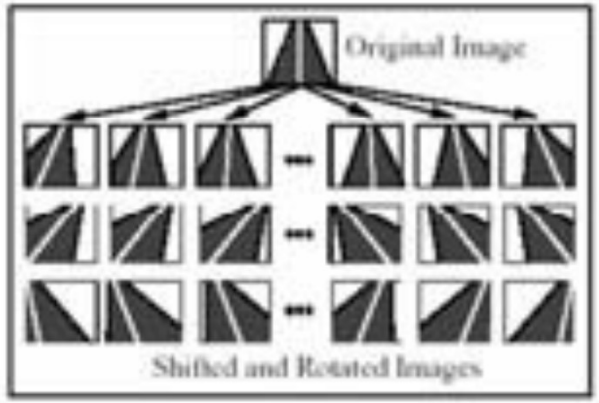
http://www.ri.cmu.edu/projects/project_160.html



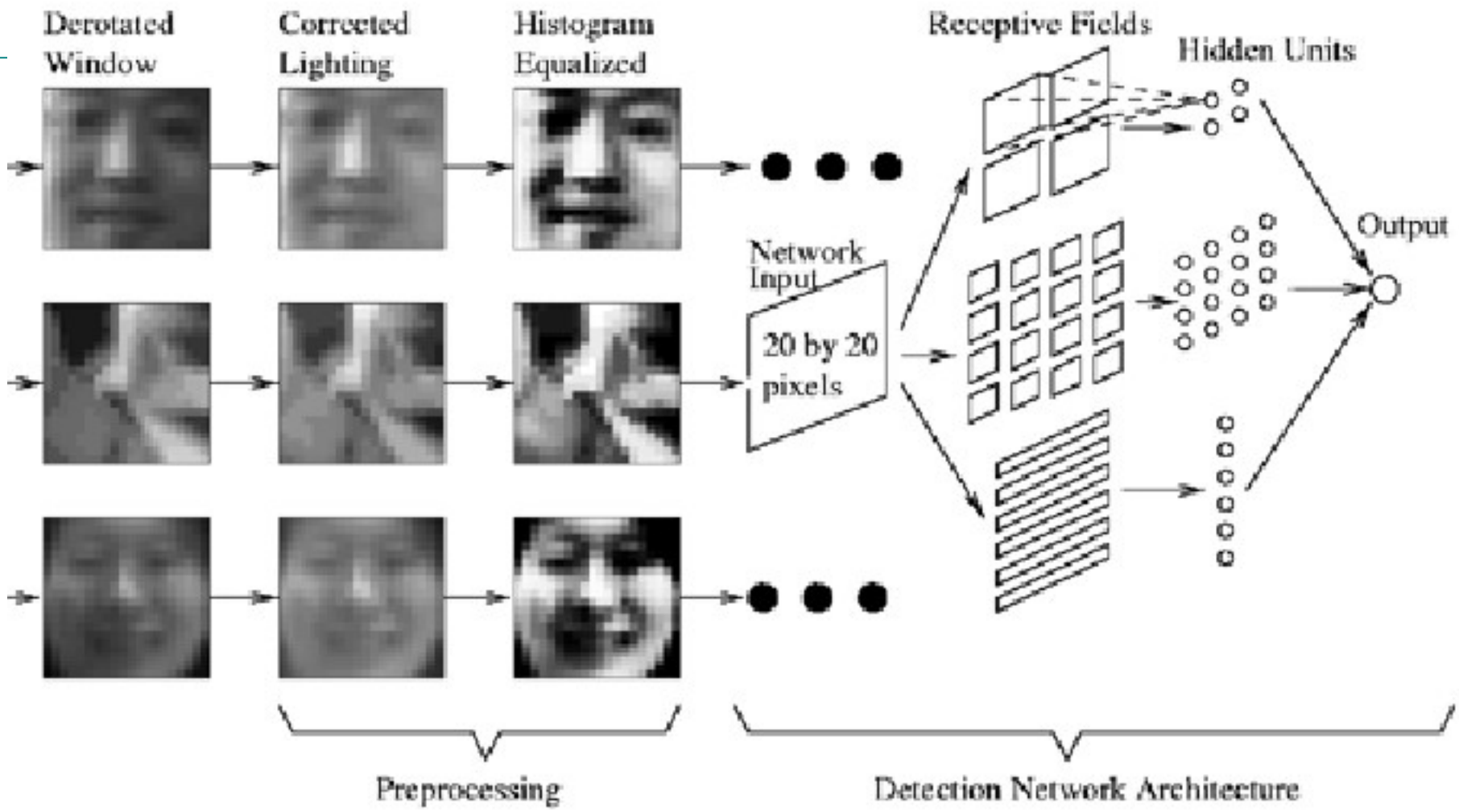


Applications

ALVINN steers on highways
http://www.ri.cmu.edu/projects/project_160.html

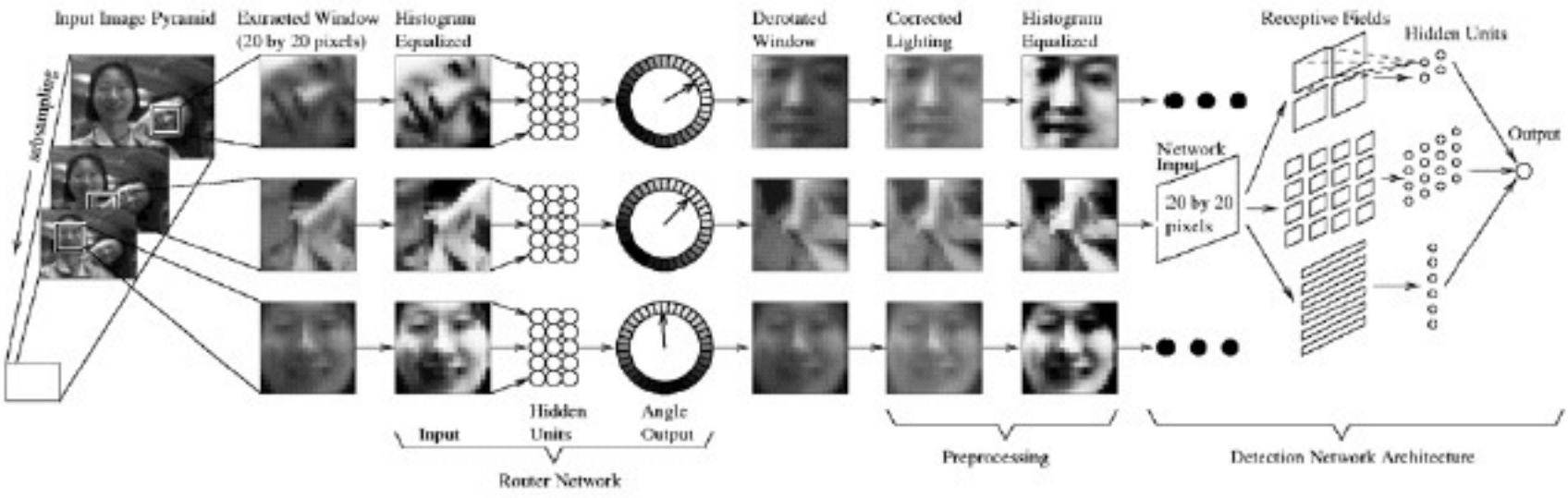


- Problem: Getting enough diversity in training set
- Answer: Transform sensor image and steering direction



The vertical face-finding part of Rowley, Baluja and Kanade's system

Figure from "Rotation invariant neural-network based face detection," H.A. Rowley, S. Baluja and T. Kanade, Proc. Computer Vision and Pattern Recognition, 1998, copyright 1998, IEEE



Architecture of the complete system: they use another neural net to estimate orientation of the face, then rectify it. They search over scales to find bigger/smaller faces.

Figure from "Rotation invariant neural-network based face detection," H.A. Rowley, S. Baluja and T. Kanade, Proc. Computer Vision and Pattern Recognition, 1998, copyright 1998, IEEE

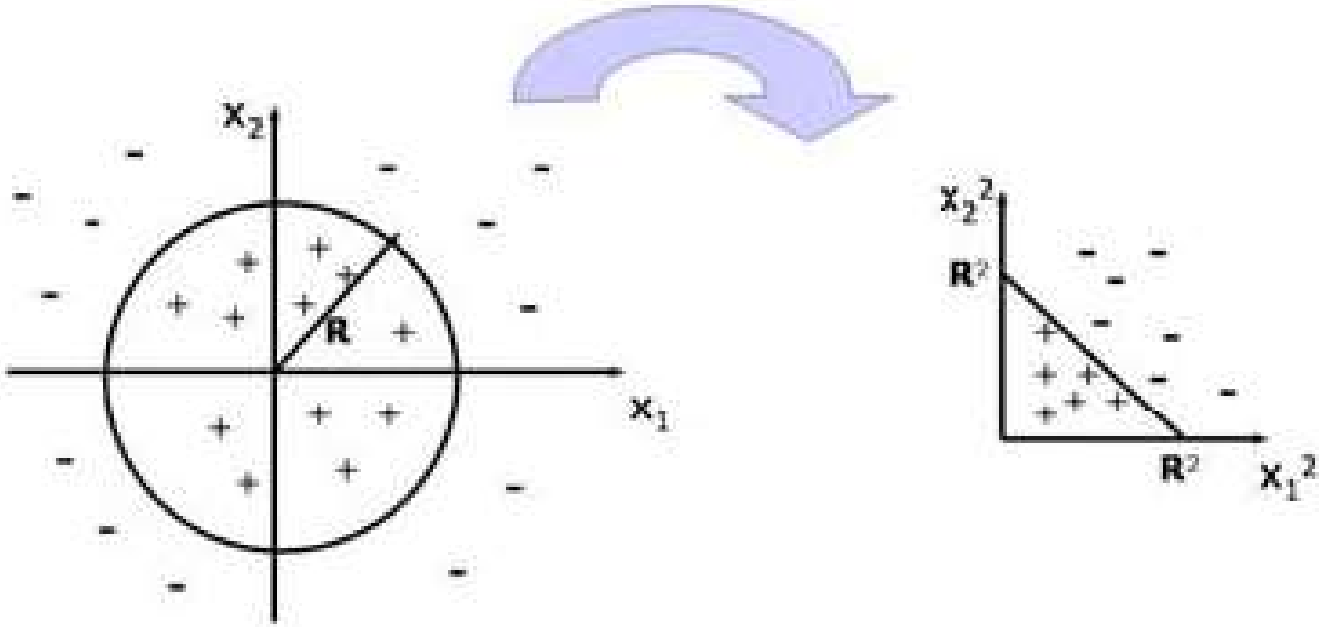
Adapted from David Forsyth, UC Berkeley

Figure from “Rotation invariant neural-network based face detection,” H.A. Rowley, S. Baluja and T. Kanade, Proc. Computer Vision and Pattern Recognition, 1998, copyright 1998, IEEE

Adapted from David Forsyth, UC Berkeley

Limitations

Isn't a linear classifier very limiting?

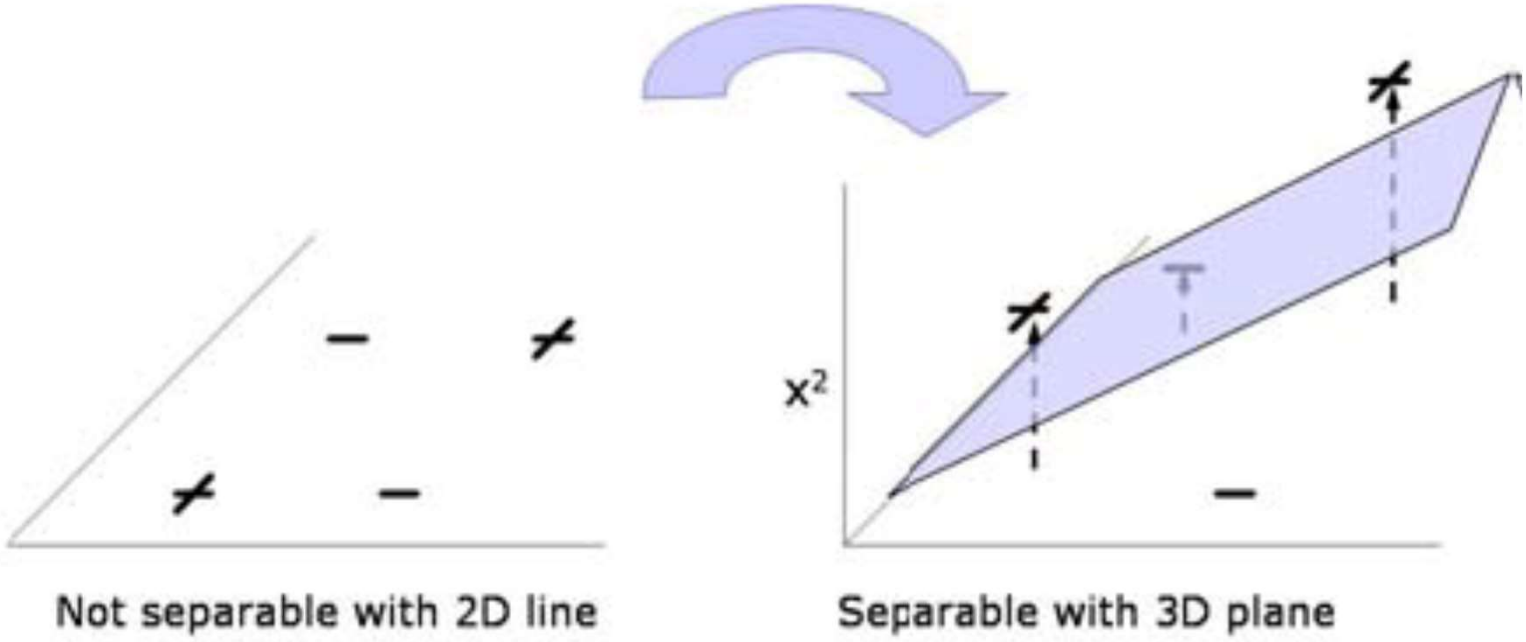


not linearly separable

linearly separable using squared value of features.

Important: Linear separator in transformed feature space maps into non-linear separator in original feature space

Not separable?
Try a higher dimensional space!



Not separable with 2D line

Separable with 3D plane

What you need

- To get into the new feature space, you use $\Phi(\mathbf{x}^i)$
- The transformation can be to a higher-dimensional feature space and may be non-linear in the feature values.
- Recall that SVM's only use dot products of the data, so
- To optimize classifier, you need $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k)$
- To run classifier, you need $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{u})$
- So, all you need is a way to compute dot products in transformed space as a function of vectors in original space!

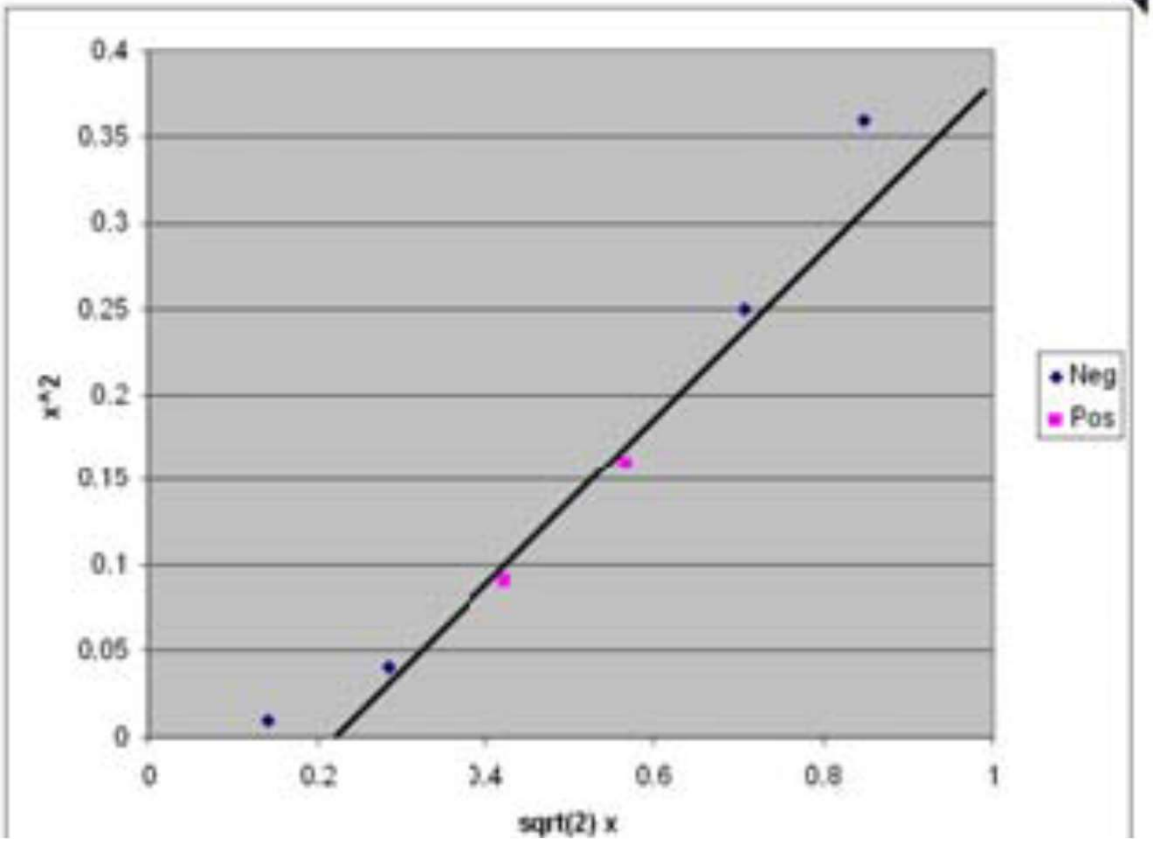
The "Kernel Trick"

- If dot products can be efficiently computed by
  $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k) = K(\mathbf{x}^i, \mathbf{x}^k)$
- Then, all you need is a function on low-dim inputs
  $K(\mathbf{x}^i, \mathbf{x}^k)$
- You don't need ever to construct high-dimensional
  $\Phi(\mathbf{x}^i)$

Polynomial Kernel Example
(one feature)

Not separable

$\Phi(x) = (x^2, \sqrt{2}x, 1)$



Separable

$$\Phi(x) \cdot \Phi(z)$$
$$= x^2z^2 + 2xz + 1$$
$$= (1 + xz)^2$$

- No change (linear kernel)

$$\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k) = K(\mathbf{x}^i, \mathbf{x}^k) = \mathbf{x}^i \cdot \mathbf{x}^k$$

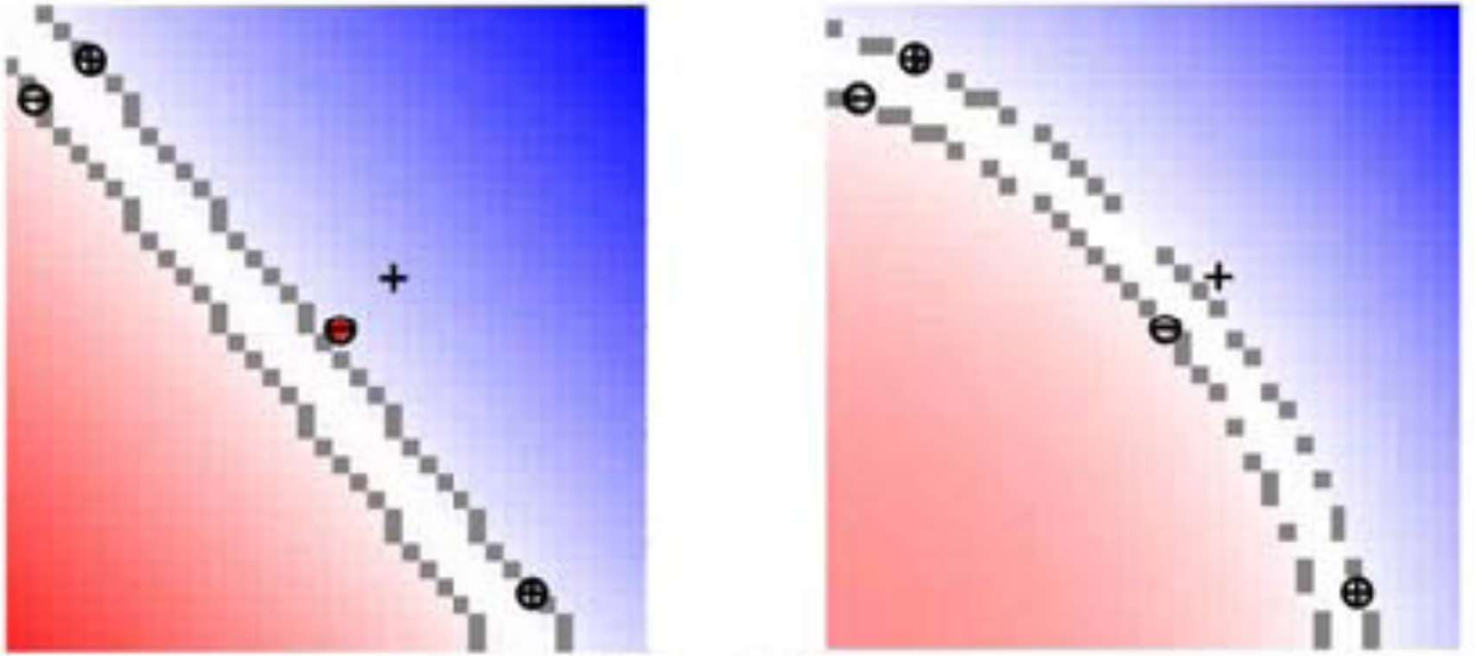
- Polynomial kernel (n^{th} order)

$$K(\mathbf{x}^i, \mathbf{x}^k) = (1 + \mathbf{x}^i \cdot \mathbf{x}^k)^n$$

- Radial basis kernel (σ is standard deviation)

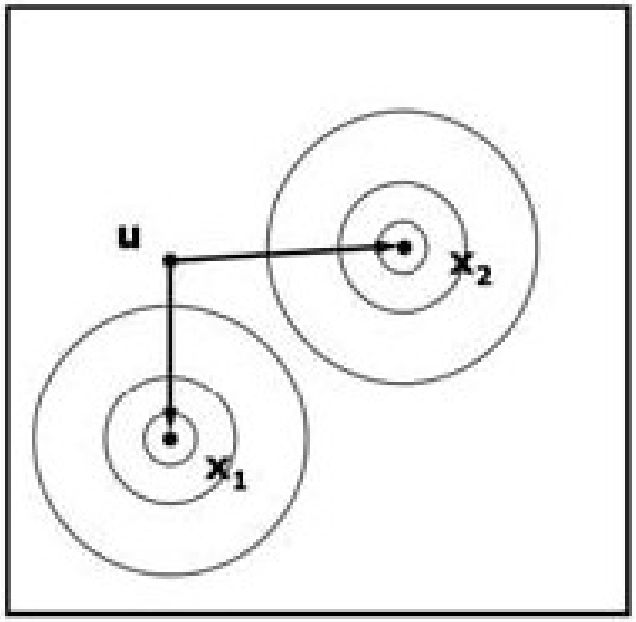
$$K(\mathbf{x}^i, \mathbf{x}^k) = e^{\frac{-\left|\mathbf{x}^i - \mathbf{x}^k\right|^2}{2\sigma^2}} = e^{\frac{-(\mathbf{x}^i - \mathbf{x}^k)\cdot(\mathbf{x}^i - \mathbf{x}^k)}{2\sigma^2}}$$

Polynomial Kernel



Images by Patrick Winston

- Classifier based on sum of Gaussian bumps with standard deviation σ, centered on support vectors.



$$h(\mathbf{u}) = sign[h'(\mathbf{u})]$$

$$h'(\mathbf{u}) = \sum_{i=1}^{k} \alpha_i y^i K(\mathbf{x}^i, \mathbf{u}) + b$$

$$K(\mathbf{x}', \mathbf{u}) = e^{\frac{-|\mathbf{x}' - \mathbf{u}|^2}{2\sigma^2}}$$

Radial-basis kernel

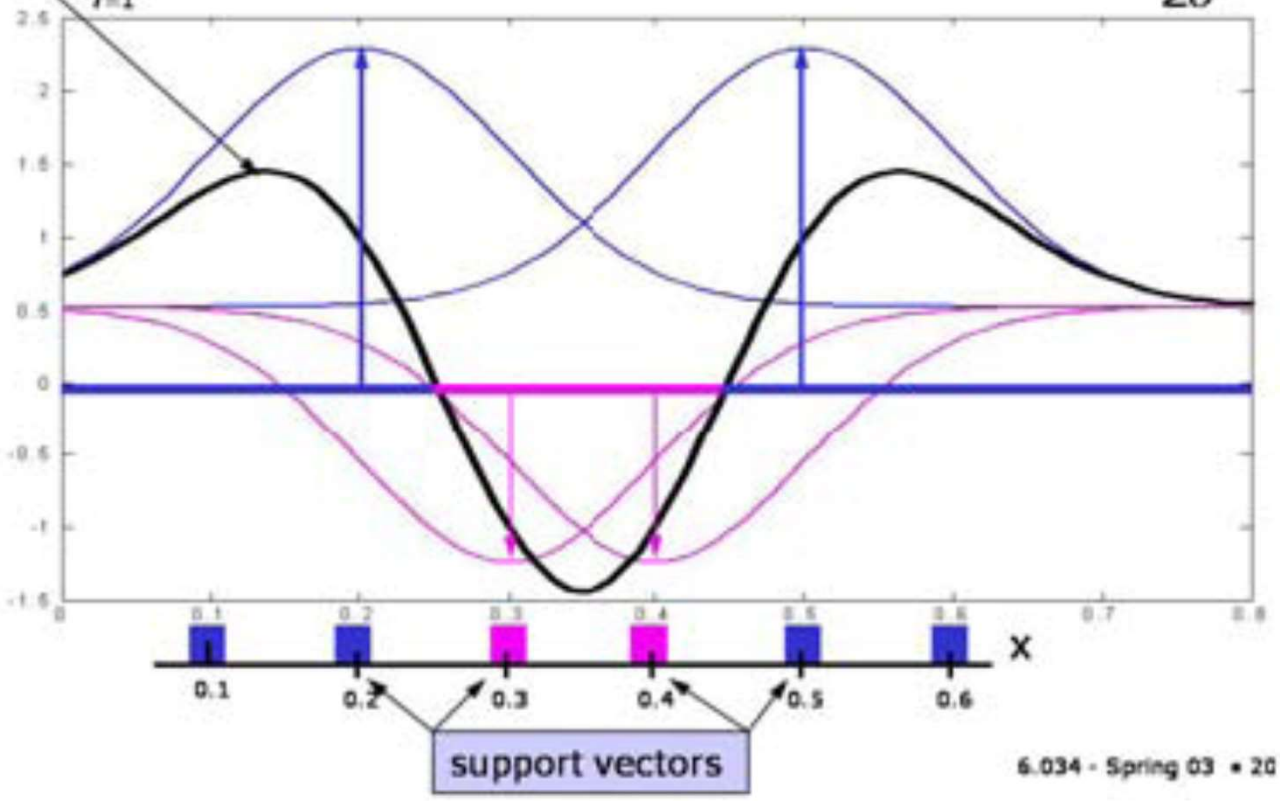
$\alpha_1 = 1.76 \quad \alpha_2 = -1.76$
$\alpha_3 = 1.76 \quad \alpha_4 = -1.76$

$b = 0.525$

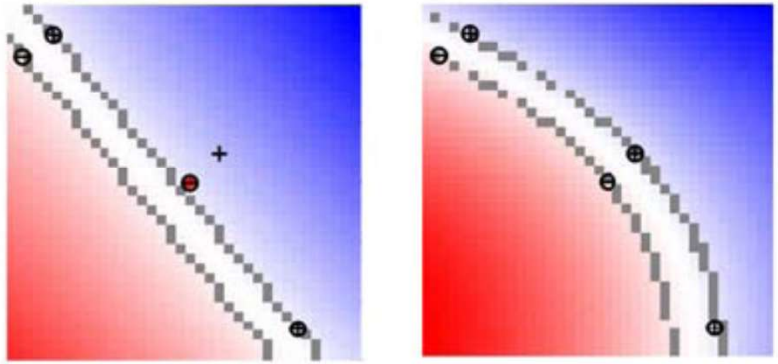
$\sigma = 0.1$

$$h'(\mathbf{u}) = \sum_{i=1}^{4} \alpha_i y^i K(\mathbf{x}^i, \mathbf{u}) + b$$

$$K(\mathbf{x}^i, \mathbf{u}) = e^{\frac{-|\mathbf{x}^i - \mathbf{u}|^2}{2\sigma^2}}$$



Radial-basis kernel (large σ)



Images by Patrick Winston

Another radial-basis example (small σ)

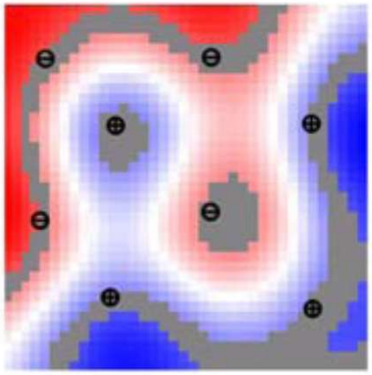


Image by Patrick Winston