

CS440/ECE448: Intro to Artificial Intelligence

**Lecture 28:**  
(the last one...)  
**Review session**

Prof. Julia Hockenmaier  
[juliahmr@illinois.edu](mailto:juliahmr@illinois.edu)

<http://cs.illinois.edu/fa11/cs440>



**Monday, May 2, 4 pm in 1404 Siebel Center**  
*Natural Language Applications Across Genres:  
From News to Novels*  
Prof. Kathleen McKeown, Columbia University

**Monday, May 2, 6 pm in 2405 Siebel Center**  
Attending Graduate School: A panel discussion

**Tuesday, May 3, 10 am 2405 Siebel Center**  
Machine Learning - Modern Times  
Dr Corinna Cortes (Head of Google Research, NY)

# **Class admin: Exams**

## **Final exam:**

**Friday, May 13, 7 pm in 1404**

## **Conflict exam:**

**Thursday, May 12, 10 am in 3401.**

## **4<sup>th</sup> credit hour presentation/Demo:**

**– Thursday, May 12, 2pm – 3pm**

# **Class admin:**

## **Office hours, review sessions**

**Last set of extra credit problem set office hours:**

- Monday, May 1, 1pm – 3pm, 3405
- Wednesday, May 3, 11am – 1pm, 3405

**Additional office hours:**

- Monday, May 9, 1pm – 3pm (Parisa Haghani)
- Tuesday, May 10, 2pm – 3pm, 3324
- Wednesday, May 11, 2pm – 3pm, 3324  
(Julia Hockenmaier)
- Thursday, May 13, 1pm – 3pm (Yonatan Bisk)

# Review

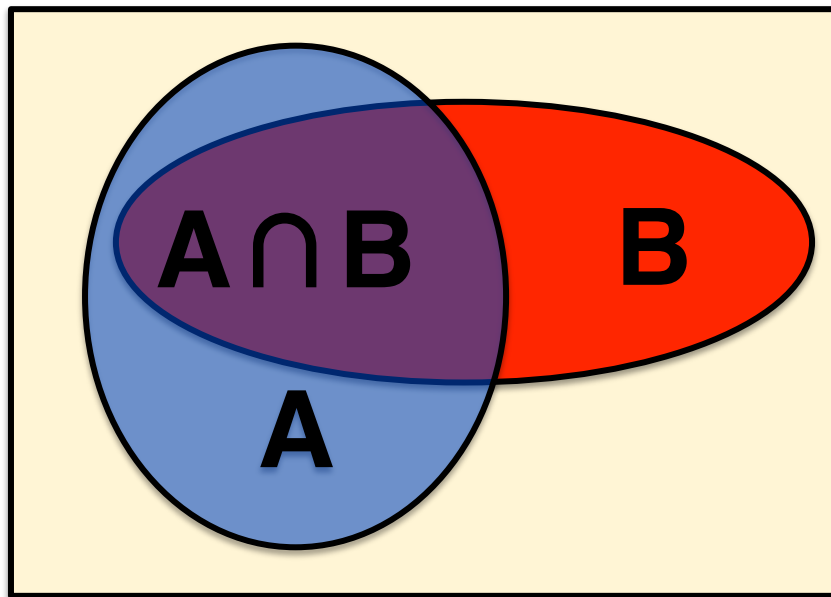
# Probability questions

1. How many parameters do you need to specify the full joint of  $P(A,B,C)$ ?
2. How can you compute  $P(A,B,C)$  using the chain rule?
3. How many parameters do you need
  - if you assume  $A, B, C$  are independent?
  - if you assume  $A$  and  $B$  are conditionally independent given  $C$ ?
4. How do you compute  $P(A)$  from the full joint  $P(A,B,C)$ ?

# Joint and conditional probabilities

Joint:  $P(A, B) = P(A | B)P(B)$   
(product rule)

Conditional:  $P(A | B) = P(A, B)/P(B)$



# Joint distributions $P(X_1 \dots X_i \dots X_n)$

How many parameters does  $P(A, B)$  have?

Answer:  $K_A \times K_B$  parameters

(here:  $K$  = number of outcomes)

In general: The joint distribution of  $n$  discrete random variables  $X_1 \dots X_i \dots X_n$  with  $K_i$  possible outcomes each has  $K_1 \times \dots \times K_i \times \dots \times K_n$  parameters.



# Chain rule

Extends the product rule to multiple variables:

$$\begin{aligned} P(X_1, \dots, X_n) = & P(X_1) \\ & \times P(X_2 \mid X_1) \\ & \times P(X_3 \mid X_{1..2}) \\ & \times \dots \\ & \times P(X_i \mid X_{1..i-1}) \\ & \times \dots \\ & \times P(X_n \mid X_{1..n-1}) \end{aligned}$$

# The parameters of a distribution

How many numbers do we need to specify a distribution?

**Bernoulli distribution:**

1 parameter (two, but one is implied)

**Categorical distribution:**

$N-1$  parameters ( $N$ , but one is implied)

**Joint distribution of  $N$  Bernoulli RVs:**

$2^N$  parameters

# Parameters of conditional distributions

How many **distributions** does  $P(A/B)$  stand for?

Answer:  $K_B$ ; one for each possible value of  $B$ .

How many parameters does  $P(A/B=b)$  have?

Answer:  $K_A$ ; one for each possible value of  $A$   
(minus one implied parameter)

So,  $P(A/B)$  has  $K_A \times K_B$  parameters

# Marginal distributions

If we only know the full joint  $P(X, Y)$ , we can still compute  $P(X)$ :

$$P(X) = \sum_y P(X, Y = y) = \sum_y P(X | Y = y) \times P(Y = y)$$

# Independence

Two random variables  $X$  and  $Y$  are **independent** if  $P(X,Y) = P(X)P(Y)$  and/or  $P(X|Y) = P(X)$

Two random variables  $X$  and  $Y$  are **conditionally independent** given  $Z$  if  $P(X,Y | Z) = P(X | Z)P(Y | Z)$

If we **assume**  $X, Y, Z$  are independent, we can **factor** the distribution:  $P(X,Y,Z) = P(X) \times P(Y) \times P(Z)$

How many parameters do we need to know to specify  $P(X,Y,Z)$  if we assume they are independent?

Only  $K_X + K_Y + K_Z$

# Independence assumptions

If we **assume**  $X, Y, Z$  are independent, we can **factor** the distribution:

$$P(X, Y, Z) = P(X) \times P(Y) \times P(Z)$$

How many parameters do we need to specify  $P(X, Y, Z)$  if we assume they are independent?

Only  $K_X + K_Y + K_Z$

# Bayes nets

1. What is a Bayes Net?
2. Can you draw one for  $P(A|B)P(B)P(C|B)$ ?
3. How do we use Bayes Nets for inference?
4. What is the advantage of variable elimination?

# Bayesian networks

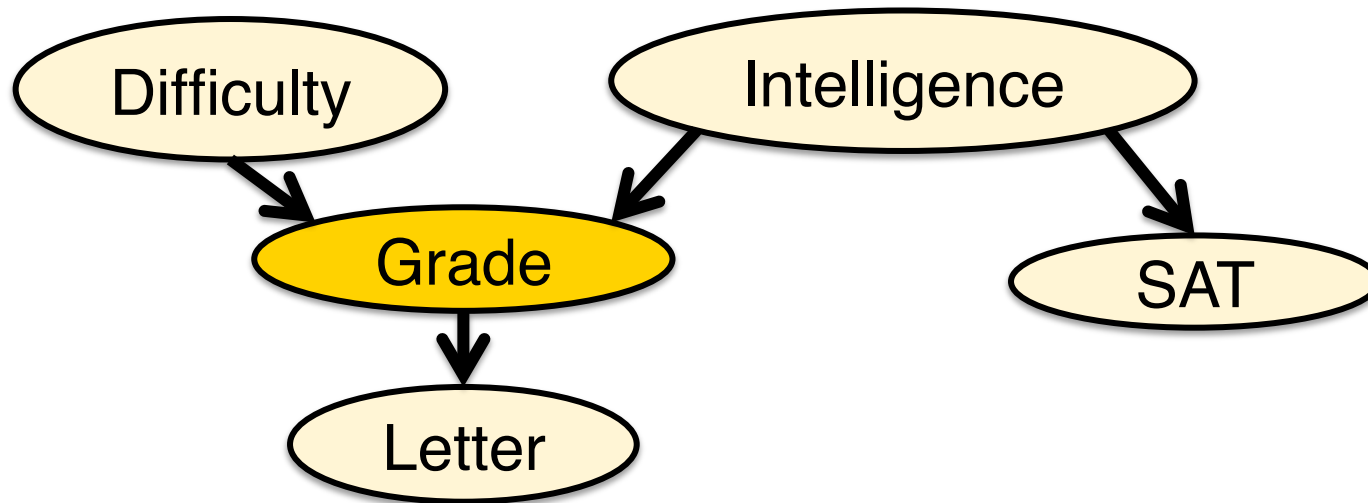
**Insight:** (Conditional) independence assumptions are essential for probabilistic modeling

**Bayes Net:** a directed graph which represents the joint distribution of a number of random variables in a directed graph

- Nodes = random variables
- Directed edges = dependencies



# Bayes Nets



*Difficulty* and *Intelligence* are **parents** of *Grade*.

*Letter* is a (direct) **descendant** of *Grade*.

The parents and direct descendant of a node form its **Markov blanket**. Each node is conditionally independent of its non-descendants given its Markov blanket.

# The chain rule for BNs

In order to compute the joint probability of the random vars  $X_1 \dots X_n$  in a Bayes Net, we multiply the conditional probabilities of each R.V.  $X_i$  given its parents  $Pa(X_i)$ :

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i \mid Pa(X_i))$$

# Inference in Bayes Nets

We want to know the distribution of a set of **query variables** given some observed **event**.

*What is the probability of getting a strong letter if you are an intelligent student?*

An event is an assignment of values to a set of **evidence variables**. (*here: intelligence*)  
The other, **hidden variables** have to be marginalized out.

# Inference by enumeration

What is the joint probability  $P(b, j, m)$ ? a

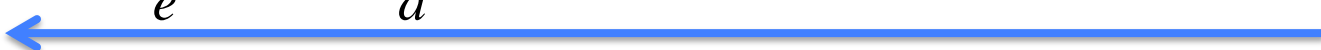
$$P(b, j, m) = \sum_e \sum_a P(b, j, m, e, a)$$

Enumeration is inefficient because it repeatedly evaluates the same terms.

# Variable elimination

We want to compute

$$P(b, j, m, a, e)$$

$$= P(b) \sum_e P(e) \sum_a P(a | b, e) P(j | a) P(m | a)$$


- We have ordered the terms so that when we sum over a variable, we sum only over terms that depend on this variable.
- Now we evaluate this equation in **right-to-left order** and **store intermediate results**

# Learning with Bayes Nets

1. How does Bayes Rule relate the posterior to the prior and the likelihood?
2. What is likelihood, and how do we compute it?
3. What is maximum likelihood estimation?
4. Do you know any other estimation techniques?

# Bayes Rule

$$P(h | D) = \frac{P(D | h)P(h)}{P(D)}$$

$P(h)$ : prior probability of hypothesis

$P(h | D)$ : posterior probability of hypothesis.

$P(D | h)$ : likelihood of data, given hypothesis

Prior  $\propto$  posterior  $\times$  likelihood

$$P(h | D) \propto P(D | h)P(h)$$

# The likelihood $P(D|h)$

We typically assume that each observation  $d_i$  is drawn “i.i.d.” - independently from the same (identical) distribution.

Therefore:

$$P(D | h) = \prod_i P(d_i | h)$$



# Three kinds of estimation techniques

Bayes optimal: Marginalize out the hypotheses

$$P(X | \mathbf{D}) = \sum_i P(X | h_i)P(h_i | \mathbf{D})$$

MAP (maximum a posteriori):

Pick the hypothesis with the highest posterior

$$h_{MAP} = \operatorname{argmax}_h P(h | \mathbf{D})$$

ML (maximum likelihood):

Pick the hypothesis that assigns highest likelihood

$$h_{ML} = \operatorname{argmax}_h P(\mathbf{D} | h)$$

# Maximum likelihood estimation

Given data  $\mathbf{D}$ , we want to find the parameters that maximize  $P(\mathbf{D} \mid \theta)$ .

We have a data set with  $N$  candies.  
 $c$  are cherry.  $l = (N-c)$ , are lime.

Parameter  $\theta$  = probability of cherry

Maximum likelihood estimate:  $\theta = c/N$

# Supervised learning

1. What is supervised learning?
2. Why do we evaluate a classifier on unseen test data?
3. Can you name reasons why we may not be able to learn a perfect classifier?
4. What does PAC learning theory tell us?

# Supervised learning

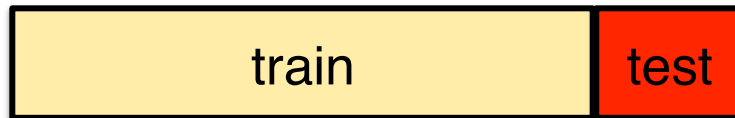
Given a set  $D$  of  $N$  items  $\mathbf{x}_i$ , each paired with an output value  $y_i = f(\mathbf{x}_i)$ , discover a function  $h(\mathbf{x})$  which approximates  $f(\mathbf{x})$

$$D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$$

Typically, the **input** values  $\mathbf{x}$  are (real-valued or boolean) **vectors**:  $\mathbf{x}_i \in R^n$  or  $\mathbf{x}_i \in \{0, 1\}^n$

The **output** values  $y$  are either boolean (*binary classification*), elements of a finite set (*multiclass classification*), or real (*regression*)

# Supervised learning



**Training:** find  $h(\mathbf{x})$

Given a training set  $D_{train}$  of items  $(\mathbf{x}_i, y_i = f(\mathbf{x}_i))$ , return a function  $h(\mathbf{x})$  which approximates  $f(\mathbf{x})$

**Testing:** how well does  $h(\mathbf{x})$  generalize?

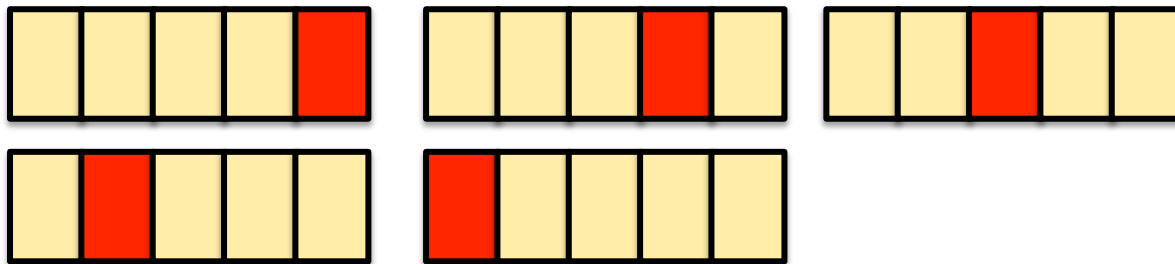
Given a test set  $D_{test}$  of items  $\mathbf{x}_i$  that is disjoint from  $D_{train}$ , evaluate how close  $h(\mathbf{x})$  is to  $f(\mathbf{x})$ .

- (classification) accuracy: pctg. of  $\mathbf{x}_i \in D_{test} : h(\mathbf{x}_i) = f(\mathbf{x}_i)$

# N-fold cross-validation

A better indication of how well  $h(x)$  generalizes:

- Split data into  $N$  equal-sized parts,
- Run and evaluate  $N$  experiments
- Report average accuracy, variance, etc.



# Example space and hypothesis space

## Example space:

The set of all possible examples  $\mathbf{x}$   
(this depends on our feature representation)

## Hypothesis space:

The set of all possible hypotheses  $h(\mathbf{x})$   
that a particular classifier can express.

# Generalization

We need to label **unseen** examples accurately.

But:

The training data is only a **very small sample** of the example space.

- We won't have seen all possible combinations of attribute values.

The training data may be **noisy**

- Some items may have incorrect attributes or labels

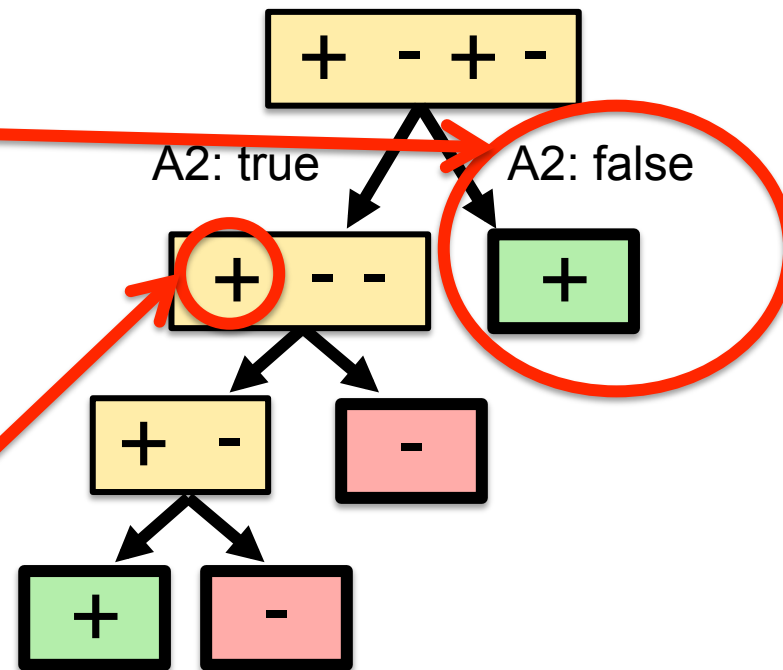


# The effect of noise

If the training data are noisy, it may introduce incorrect splits.

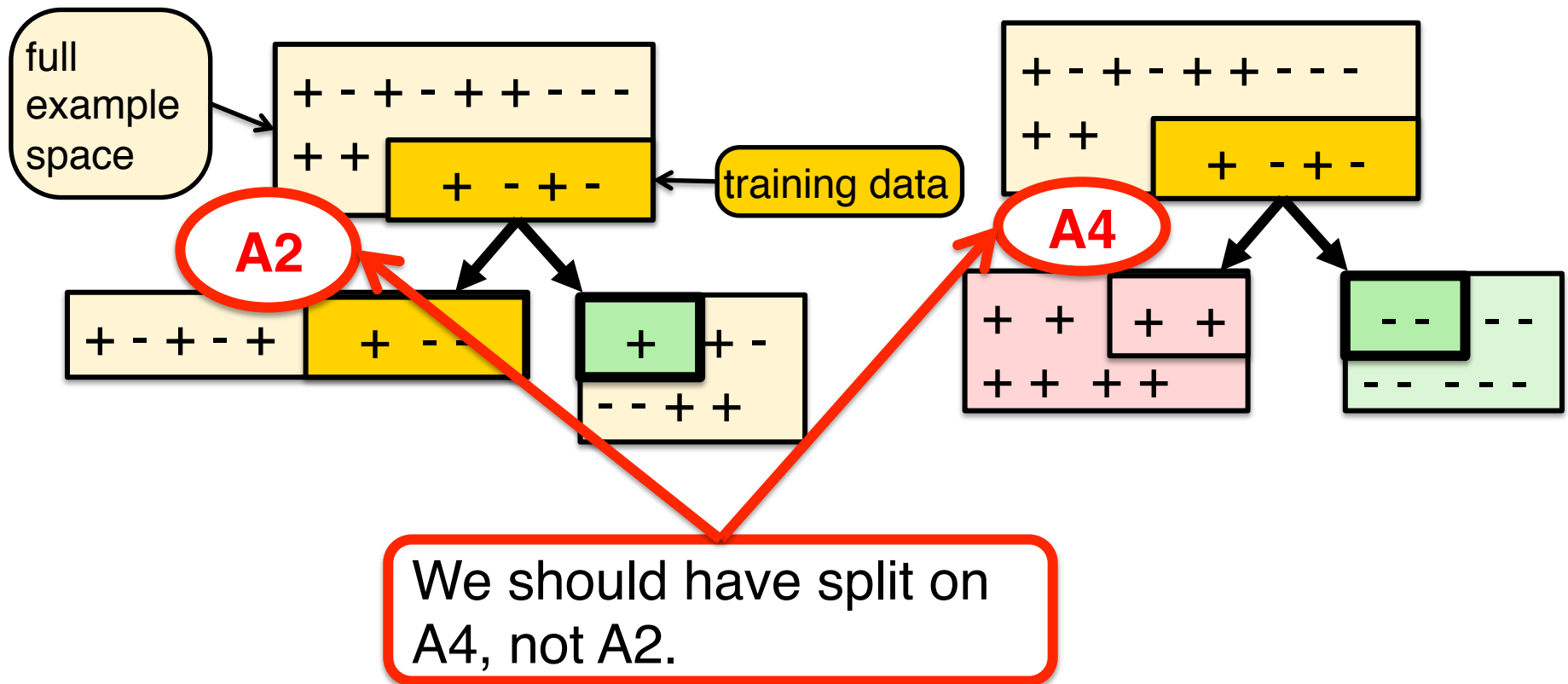
If this *false* **value** should have been *true*, we wouldn't split on *A2*.

If this + **label** should have been -, we wouldn't have to split any further.



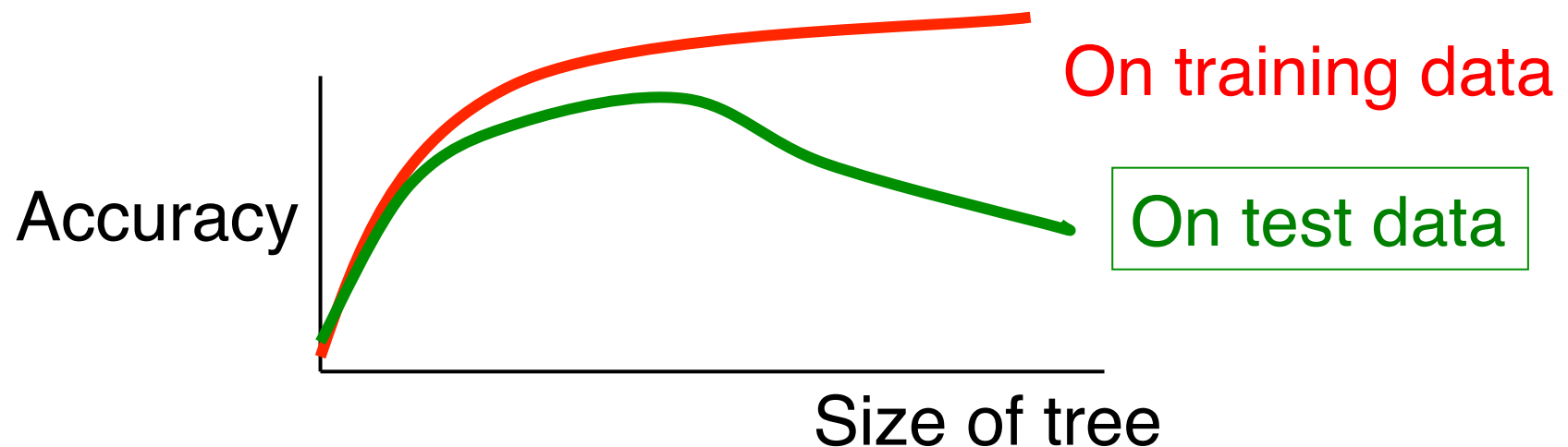
# The effect of incomplete data

If the training data are incomplete, we may miss important generalizations.



# Overfitting

The decision tree might **overfit** the particularities of the training data.



# Bias-variance tradeoff

**Bias:** What kind of hypotheses do we allow?

We want rich enough hypotheses to capture the target function  $f(\mathbf{x})$

**Variance:** How much does our learned hypothesis change if we resample the training data?

Rich hypotheses (e.g. large decision trees) need more data (which we may not have)

# Reducing variance: bagging

Create a new training set by sampling (with replacement)  $N$  items from the original data set.

Repeat this  $K$  times to get  $K$  training sets.  
( $K$  is an odd number, e.g. 3, 5, ...)

Train one classifier on each of the  $K$  training sets

Testing: take the majority vote of these  $K$  classifiers

# PAC learning

(PAC = probably approximately correct)

**Original question:** How many training examples do we need to find the correct hypothesis  $h$  ?

**A simpler task:** Find an approximately correct hypothesis  $h'$  that makes no more than  $\epsilon$  errors.

**A simpler question:** How many training examples do we need to find with high probability an approximately correct hypothesis  $h'$  ?

# PAC learning

(PAC = probably approximately correct)

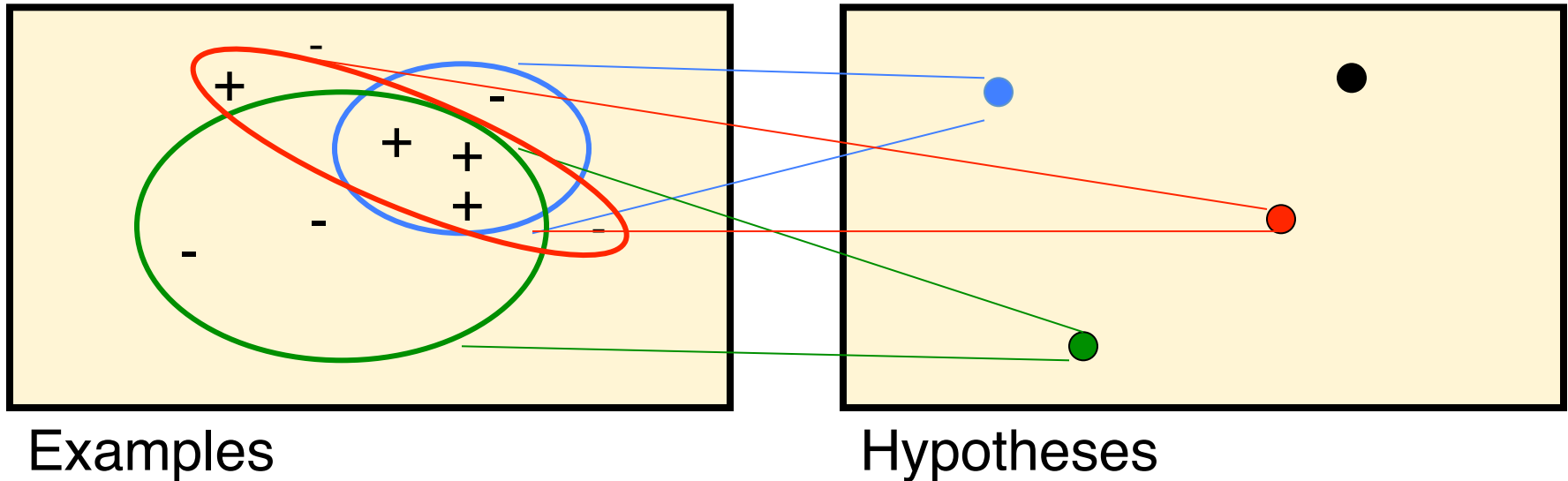
## Assumptions:

- The hypothesis space  $H$  is finite
- One hypothesis  $h \in H$  generates the labels
- Training data is sampled i.i.d according to an unknown distribution  $D$

## Question:

How many training examples do we need to see to find  $h$ ?

# Hypotheses as partitioning functions $h_i: X \rightarrow \{+,-\}$



Given a set of  $N$  labeled examples, is there a consistent hypothesis?

If we change the labels, is there still a consistent hypothesis?

What is the largest  $N$  for which the answer is always “yes”?

This is the **Vapnik-Chervonenkis dimension** of the hypothesis space **VC(H)**



# PAC learning

(PAC = probably approximately correct)

How many examples do we need to find with high probability an approximately correct hypothesis  $h'$ ?

How many examples do we need to find with probability  $p > 1 - \delta$  a hypothesis  $h'$  that has accuracy  $> 1 - \epsilon$ ?

$$N \geq \frac{1}{\epsilon} \left( \ln \frac{1}{\delta} + \ln |H| \right)$$

# Capacity & VC Dimension

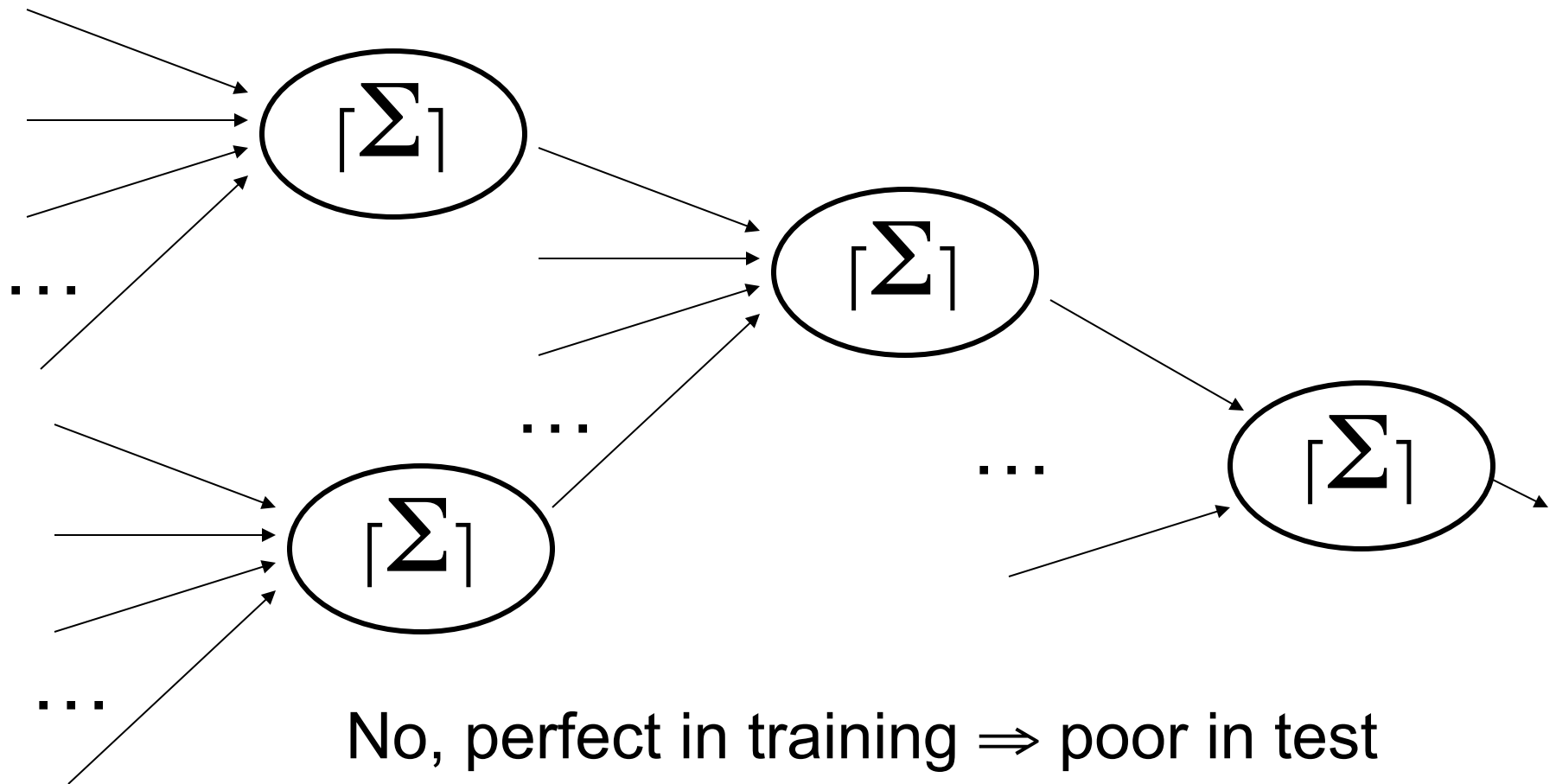
$VC(H)$  is the cardinality of the largest set of examples *shattered* by  $H$

An example set is shattered by a hypothesis set *iff* every label assignment of the examples is consistent with some element of  $H$

VC is most common but there are other measures of *capacity*

**With enough units, an ANN can learn any assignment of training labels**

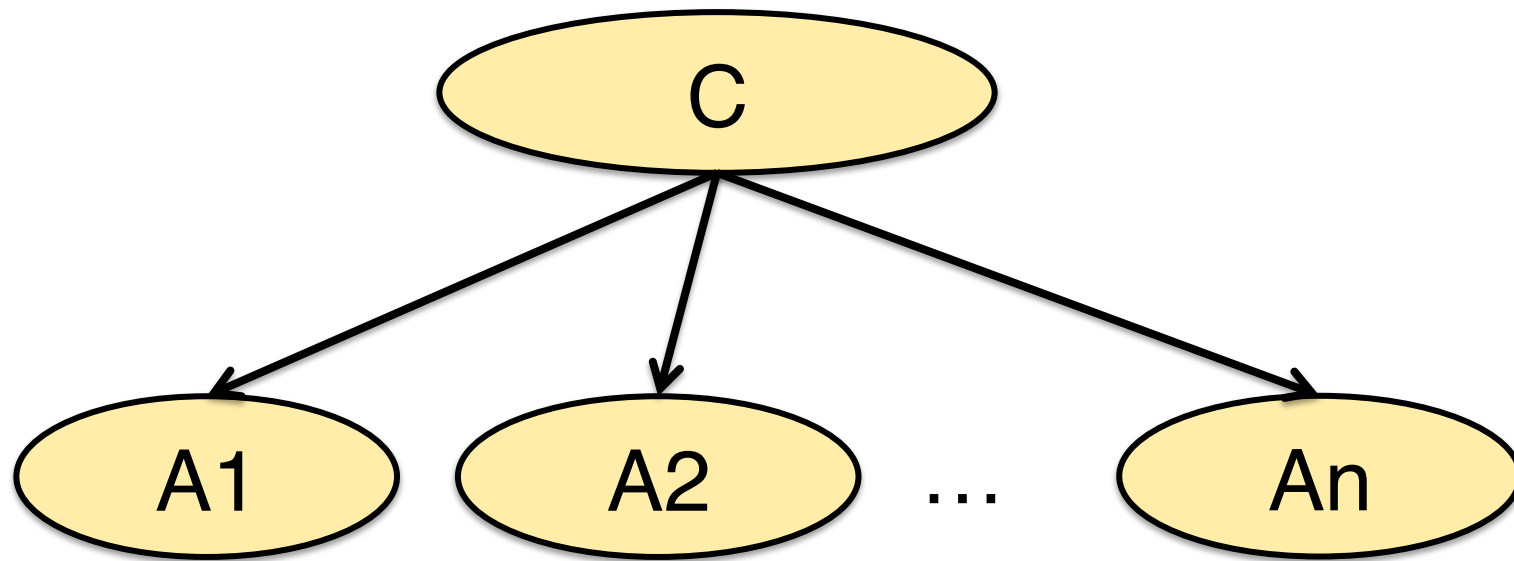
Is this a good thing?



# Classifiers

1. What is a Naïve Bayes classifier, and how do we learn it?
2. What is a decision tree classifier, and how do we learn it?
3. What is a perceptron classifier, and how do we learn it?
4. What is an SVM classifier?

# The Naïve Bayes classifier



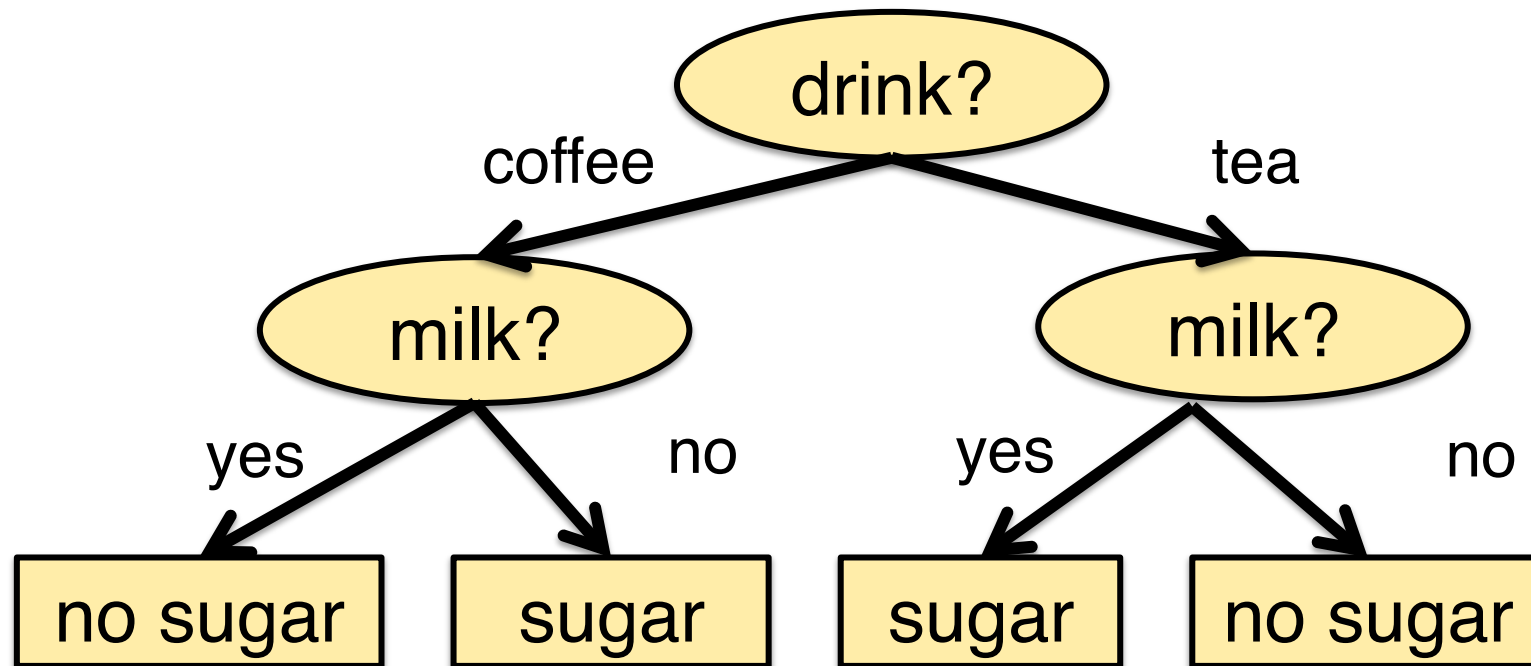
# Naïve Bayes

$$\begin{aligned}\operatorname{argmax}_C P(C | A_1 \dots A_n) &= \\ &= \operatorname{argmax}_C P(A_1 \dots A_n | C) P(C) \\ &= \operatorname{argmax}_C \prod_j P(A_j | C) P(C)\end{aligned}$$

We need to estimate:

- the multinomial  $P(C)$
- for each attribute  $A_j$  and class  $c$   $P(A_j | c)$

# Decision trees



# Expressiveness of decision trees

Consider **binary classification** ( $y = \text{true}, \text{false}$ ) where the items have Boolean attributes.

In the decision tree, each **path** from the root to a leaf node is a **conjunction of propositions**.

The **goal** ( $y = \text{true}$ ) corresponds to a **disjunction of such conjunctions** (=all the paths from the root to a *true* leaf)



# How can we find a good decision tree?

With  $n$  Boolean attributes, there are  $2^{2^n}$  possible decision trees.

We cannot enumerate all trees.

We will need to do a greedy (local) search.

Tests (splits) need to be informative, i.e. after a split we need to be more certain about which label to assign to the items that meet the test.

# Entropy $H(S)$

The entropy  $H(S)$  of a random variable  $S$  measures the uncertainty associated with  $S$ .

It also corresponds to the **average number of bits** required to specify  $S$ .

$$H(S) = - \sum_{i=1}^N P(s_i) \log_2 P(s_i)$$

# Information Gain

How much information are we gaining by splitting node  $S$  on attribute  $A$  with values  $V(A)$ ?

Information required before the split:

$$H(S_{\text{parent}})$$

Information required after the split:

$$\sum_{i \in V(A)} P(S_{\text{child}_i}) H(S_{\text{child}_i})$$

$$\text{Gain}(S_{\text{parent}}, A) = H(S_{\text{parent}}) - \sum_{i \in V(A)} H(S_{\text{child}_i}) \frac{|S_{\text{child}_i}|}{|S_{\text{parent}}|}$$

# Pruning a decision tree

1. Train a decision tree on training data  
(keep a part of training data as unseen validation data)

2. Prune from the leaves:

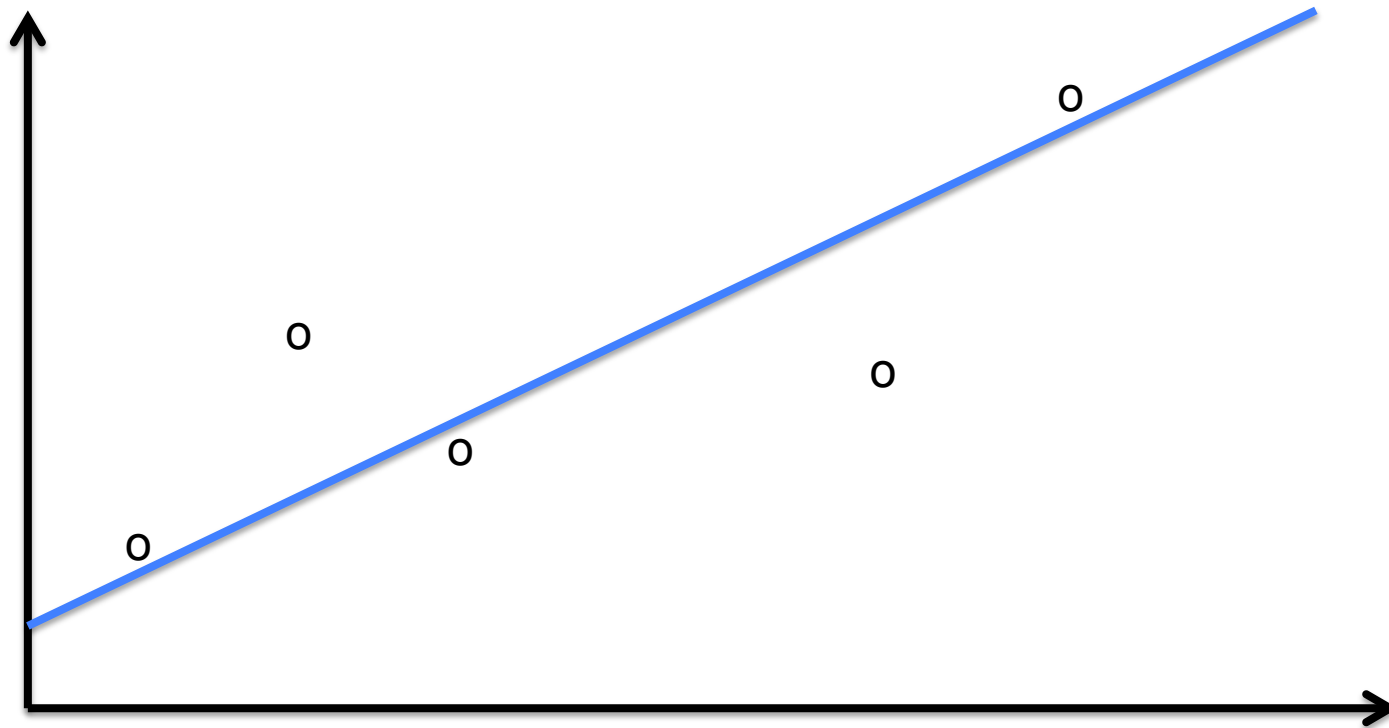
Simplest method:

Replace (prune) each non-leaf node whose children are all leaves with its majority label.

Keep this change if the accuracy on validation set does not degrade.

# Linear regression

Given some data  $\{(x,y)\dots\}$ , with  $x, y \in \mathbb{R}$ ,  
find a function  $f(x) = w_1x + w_0$  such that  $f(x) \approx y$ .



# Squared Loss

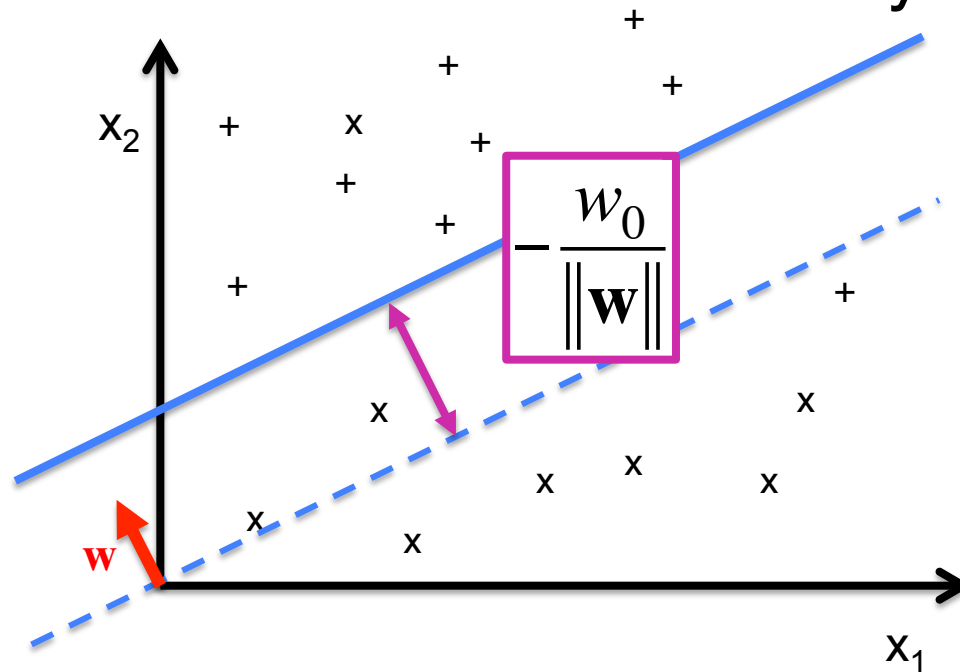
We want to find a weight vector  $\mathbf{w}$  which minimizes the loss (error) on the training data  $\{(x_1, y_1) \dots (x_N, y_N)\}$

$$\begin{aligned} L(\mathbf{w}) &= \sum_{i=1}^N L_2(f_{\mathbf{w}}(x_i), y_i) \\ &= \sum_{i=1}^N (y_i - f_{\mathbf{w}}(x_i))^2 \end{aligned}$$

# Linear classifiers

The **weight vector  $\mathbf{w}$**  defines the **orientation** of the decision boundary.

The **bias term  $w_0$**  defines the perpendicular **distance** of the decision boundary to the origin.



# Learning the weights (Perceptron algorithm)

Iterative solution:

- Start with initial weight vector  $\mathbf{w}$ .
- For each example  $(\mathbf{x}, y)$  update weights  $\mathbf{w}$  until  $\mathbf{w}$  has converged (does not change significantly anymore)

Perceptron update rule ('online'):

- For each example  $(\mathbf{x}, y)$  update each weight  $w_i$ :  
$$w_i := w_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x}))x_i$$
- $\alpha$  decays over time  $t$  ( $t = \text{\#examples}$ ) e.g  $\alpha = n/(n+t)$

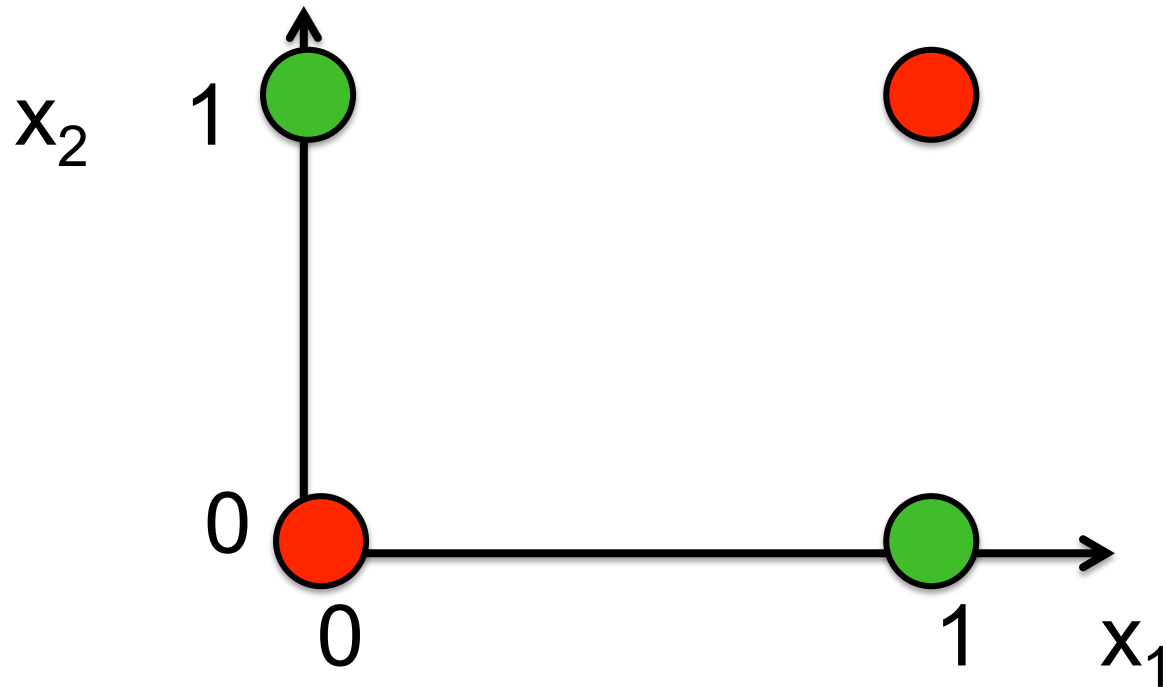


# Perceptron algorithm

Given training data  $\{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^j, y^j), \dots, (\mathbf{x}^N, y^N)\}$

- Start with initial weight vector  $\mathbf{w}$
- **Online update:** Update  $\mathbf{w}$  for each  $(\mathbf{x}^j, y^j)$   
$$w_i := w_i + \alpha (y^j - h_{\mathbf{w}}(\mathbf{x}^j)) x_i^j$$
- **Batch update:** Go through entire data set before updating  $\mathbf{w}$   
$$\Delta w_i = \sum_j (\alpha (y^j - h_{\mathbf{w}}(\mathbf{x})) x_i^j) \quad w_i := w_i + \Delta w_i$$
- The learning rate  $\alpha$  decays over time

# Boolean XOR



XOR is not linearly separable

# From perceptrons to neural networks

Each unit computes a weighted sum of its inputs:  $in_j = \sum_j w_{ij} a_{ij}$

and applies an **activation function** to this input

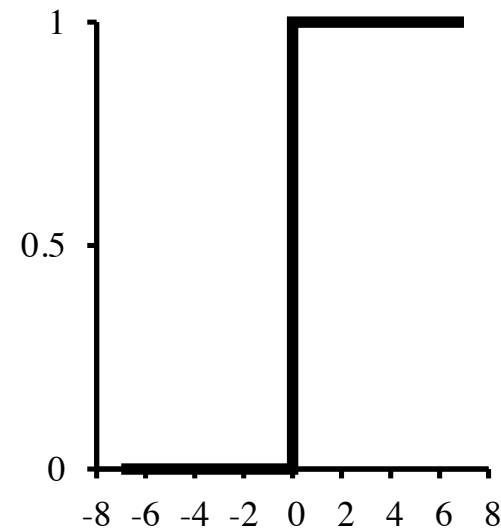
$$a_j = g(in_j) = g(\sum_j w_{ij} a_{ij})$$

**activation function:** linear threshold or sigmoid threshold

# The perceptron threshold

The perceptron uses a hard threshold function:  
 $h_{\mathbf{w}}(\mathbf{x})$ : if  $f(\mathbf{x}) = \mathbf{w}\mathbf{x} > 0$  return  $y = 1$ , else return  $y = 0$

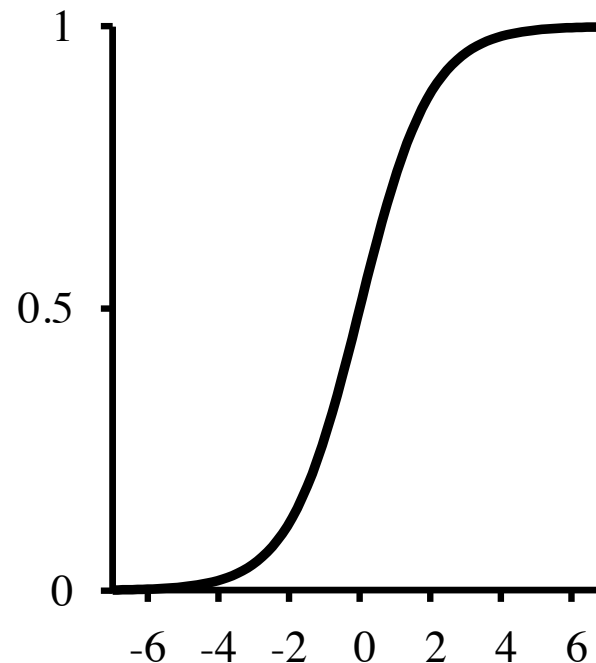
This is a non-differentiable function, so we cannot use gradient descent.



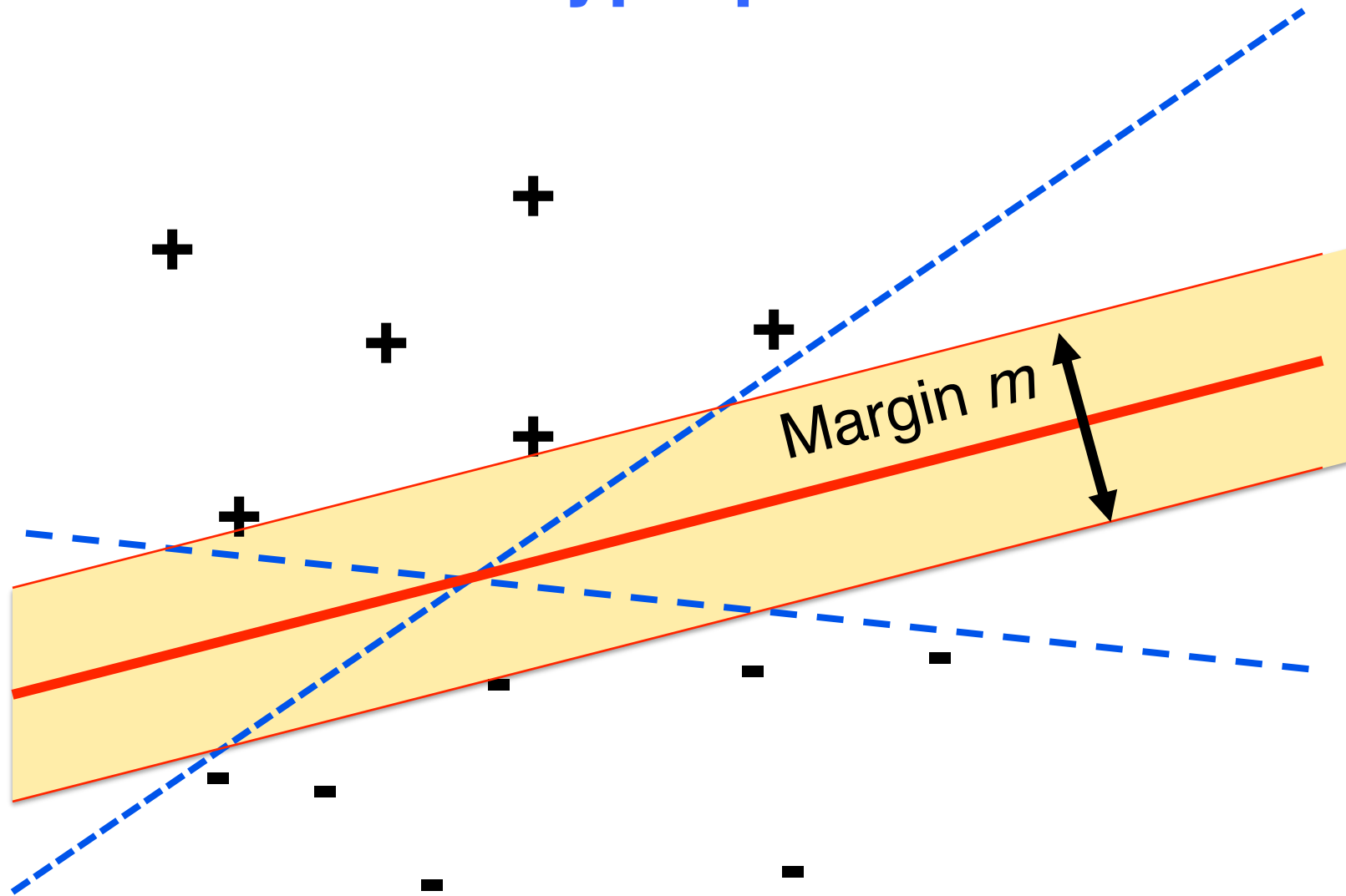
# The sigmoid threshold

The logistic (sigmoid) function is differentiable.  
We can also think of it as a probability

$$h_{\mathbf{w}}(x) = \frac{1}{1 + e^{-\mathbf{w}\mathbf{x}}}$$



# What's the Best Separating Hyperplane?



# Maximum margin classifier

We want to find the classifier whose decision boundary is **furthest away from any data point**. (this classifier has the **largest margin**).

This additional requirement (*bias*) reduces the *variance* (i.e. reduces overfitting).

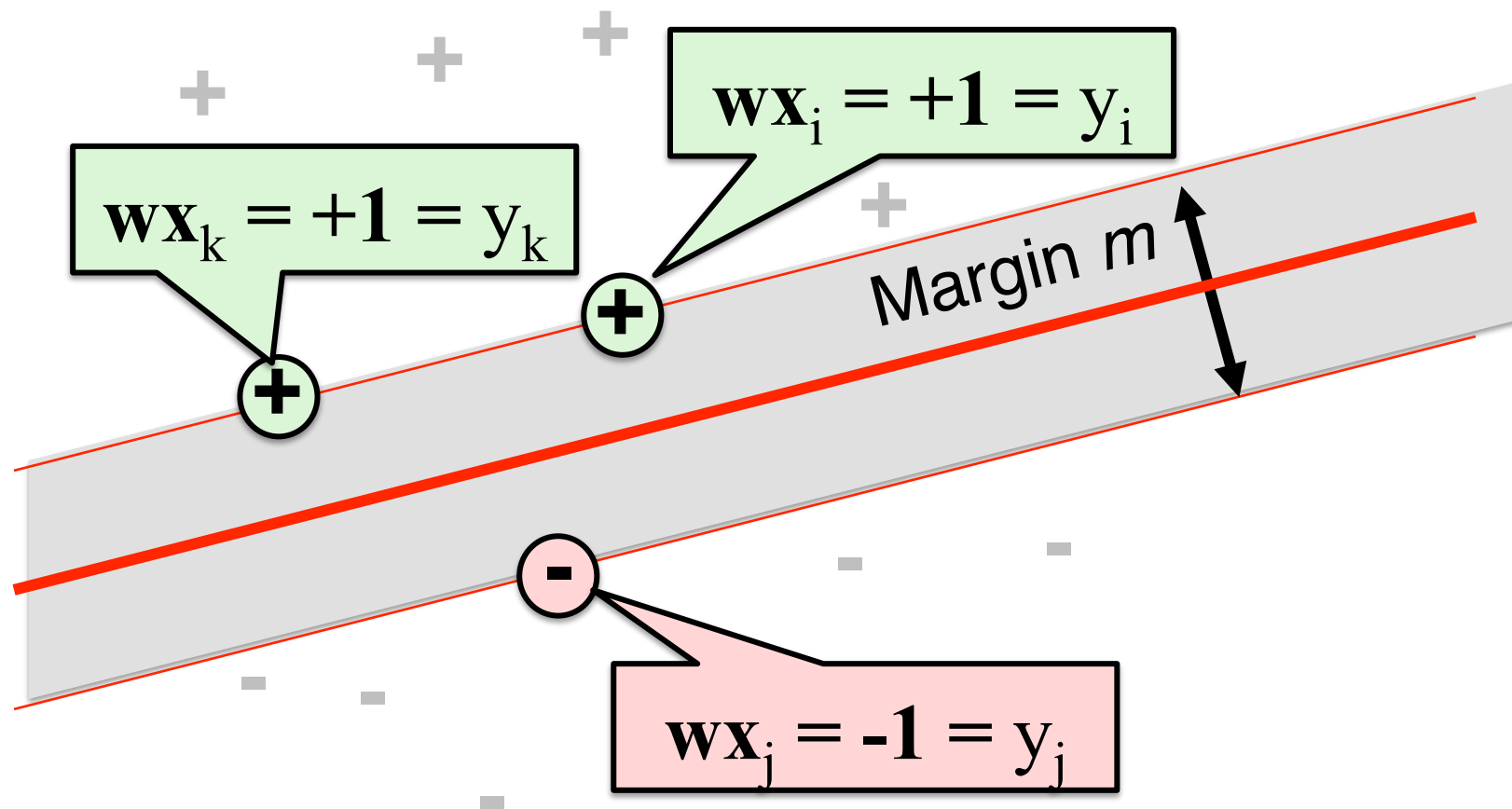
# The maximum margin decision boundary...

... is defined by two parallel hyperplanes:

- one that goes through the **positive** data points ( $y_j = +1$ ) that are closest to the decision boundary, and
- one that goes through the **negative** data points ( $y_j = -1$ ) that are closest to the decision boundary.



# Support vectors



# Support vectors

We can express the separating hyperplane in terms of the data points  $\mathbf{x}_j$  that are closest to the decision boundary.

These data points are called the **support vectors**.

# The primal representation

The data items  $\mathbf{x} = (x_1 \dots x_n)$  have  $n$  features

The weight vector  $\mathbf{w} = (w_1 \dots w_n)$  has  $n$  elements

Learning:

Find a weight  $w_j$  for each feature  $x_j$

Classification:

Evaluate  $\mathbf{w}\mathbf{x}$

# The dual representation

Equivalently, we can represent  $\mathbf{w}$  as a linear combination of the items in the training data:

$$\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$$

**Learning:**

Find a weight  $\alpha_j$  ( $\geq 0$ ) for each data point  $\mathbf{x}_j$   
This requires computing the inner product  $\mathbf{x}_i \mathbf{x}_j = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$   
between all data items  $\mathbf{x}_i, \mathbf{x}_j$

**Support vectors**

= the set of data points  $\mathbf{x}_j$  with non-zero weights  $\alpha_j$

# Classifying test data

In the primal:

Compute inner product between weight vector and test item

$$\mathbf{w}\mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle$$

In the dual:

Compute inner product between support vectors and test item

$$\mathbf{w}\mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle = \langle \sum_j \alpha_j x_j, \mathbf{x} \rangle = \sum_j \alpha_j \langle x_j, \mathbf{x} \rangle$$

# The kernel trick

$N$  (independent) data points will always be linearly separable in  $N-1$  dimensions.

Insight 1: if we map each  $\mathbf{x}$  to a point  $F(\mathbf{x})$  in a higher-dimensional feature space, the data become linearly separable

Insight 2: in the dual, we compute  $\langle F(\mathbf{x}_i) F(\mathbf{x}_j) \rangle$ . This can often be computed directly as a 'kernel' function  $K(\mathbf{x}_i, \mathbf{x}_j)$

# The kernel trick

What is  $K(\mathbf{x}, \mathbf{y})$  ?

Anything we want; often polynomial kernels:

$(\mathbf{x} \cdot \mathbf{y})^d$  Homogeneous polynomials

$(\mathbf{x} \cdot \mathbf{y} + 1)^d$  Complete polynomials

Condition: the kernel matrix  $\mathbf{K}$  with  $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$  is positive semi-definite

In the dual, we now compute

$$\sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \mathbf{x})$$

**Good luck  
and thanks!**