

# Binary Classification

Wei Xu

(many slides from Greg Durrett and Vivek Srikumar)

# Administrivia

---

- ▶ Readings on course website: [https://cocoxu.github.io/CS7650\\_fall2021/](https://cocoxu.github.io/CS7650_fall2021/)
- ▶ Problem set 1 is released on GradeScope, due on August 30.
- ▶ TA Office hours:
  - ▶ Tuesday 3-4pm (Nathan), Thursday 4-5pm (Mounica), Friday 10-11am (Sarah)

# Coursework Plan

---

- ▶ 3 Programming Projects (40%; fairly substantial implementation effort)
  - ▶ Text classification
  - ▶ Named entity recognition (BiLSTM-CNN-CRF)
  - ▶ Neural chatbot (Seq2Seq with attention)
- ▶ 2 written assignments (20%) + midterm exam (15%)
- ▶ Mostly math problems related to ML / NLP
- ▶ Final project (20%; details on course website, will discuss later)

There will be a lot of math and programming!

# Assignments

---

- ▶ 3 Programming Assignments (40% grade)
- ▶ Implementation-oriented
- ▶ ~2 weeks per assignment, 3 “slip days” for automatic extensions

These projects require understanding of the concepts, ability to write performant code, and ability to think about how to debug complex systems. **They are challenging, so start early!**

# Final Project

---

- ▶ Final project (20% grade)
  - ▶ Groups of 3-4 preferred, 1 is possible.
  - ▶ Good idea to talk to run your project idea by me in office hours or email.
  - ▶ 4 page report + final project presentation.

# Outline of the Course

ML and structured prediction for NLP

Neural Networks semantics

Applications:  
MT, IE,  
summarization,  
dialogue, etc.

Date	Topic	Projects	Problem Sets	
8/23/2021	Course Overview			
8/25/2021	Machine Learning		PS1 Out	
8/30/2021	Machine Learning			
9/1/2021	Neural Networks in NLP	Proj. 1 Out	PS1 Due	
<b>9/6/2021</b>	<b>No class - Labor Day Holiday</b>			
9/8/2021	Neural Networks in NLP			
9/13/2021	Sequence Labeling			
9/15/2021	The Viterbi Algorithm	Proj. 1 Due	PS2 Out	
9/20/2021	Conditional Random Fields			
9/22/2021	Word Embeddings			
9/27/2021	Recurrent Neural Networks	Proj. 2 Out	PS2 Due	
9/29/2021	Convolutional Neural Networks			
10/4/2021	Neural CRF			
10/6/2021	Machine Translation			
<b>10/11/2021</b>	<b>No class - Fall Break</b>			
10/13/2021	Encoder-Decoder networks	Proj. 2 Due	Midterm Out	Final Projects Kickoff
10/18/2021	Neural Machine Translation			
10/20/2021	Machine Translation			
10/25/2021	Dialogue	Proj. 3 Out	Midterm Due	
10/27/2021	Dialogue			
11/1/2021	Pertrained Transformers			
11/3/2021	Pertrained Transformers			
11/8/2021	Semantics	Proj. 3 Due		
11/10/2021	Information Extraction			
11/15/2021	Information Extraction			
11/17/2021	Reading Comprehension			
11/22/2021	Reading Comprehension			
<b>11/24/2021</b>	<b>No class - Thanksgiving Break</b>			
11/29/2021	Wrap-up / Ethics			

tentative plan  
(subject to change)

# This and next Lecture

---

- ▶ Linear classification fundamentals
- ▶ Naive Bayes, maximum likelihood in generative models
- ▶ Three discriminative models: logistic regression, perceptron, SVM
  - ▶ Different motivations but very similar update rules / inference!

# Classification

# Classification

---

- ▶ Datapoint  $x$  with label  $y \in \{0, 1\}$
- ▶ Embed datapoint in a feature space  $f(x) \in \mathbb{R}^n$   
but in this lecture  $f(x)$  and  $x$  are interchangeable

▶ Linear decision rule:  $w^\top f(x) + b > 0$

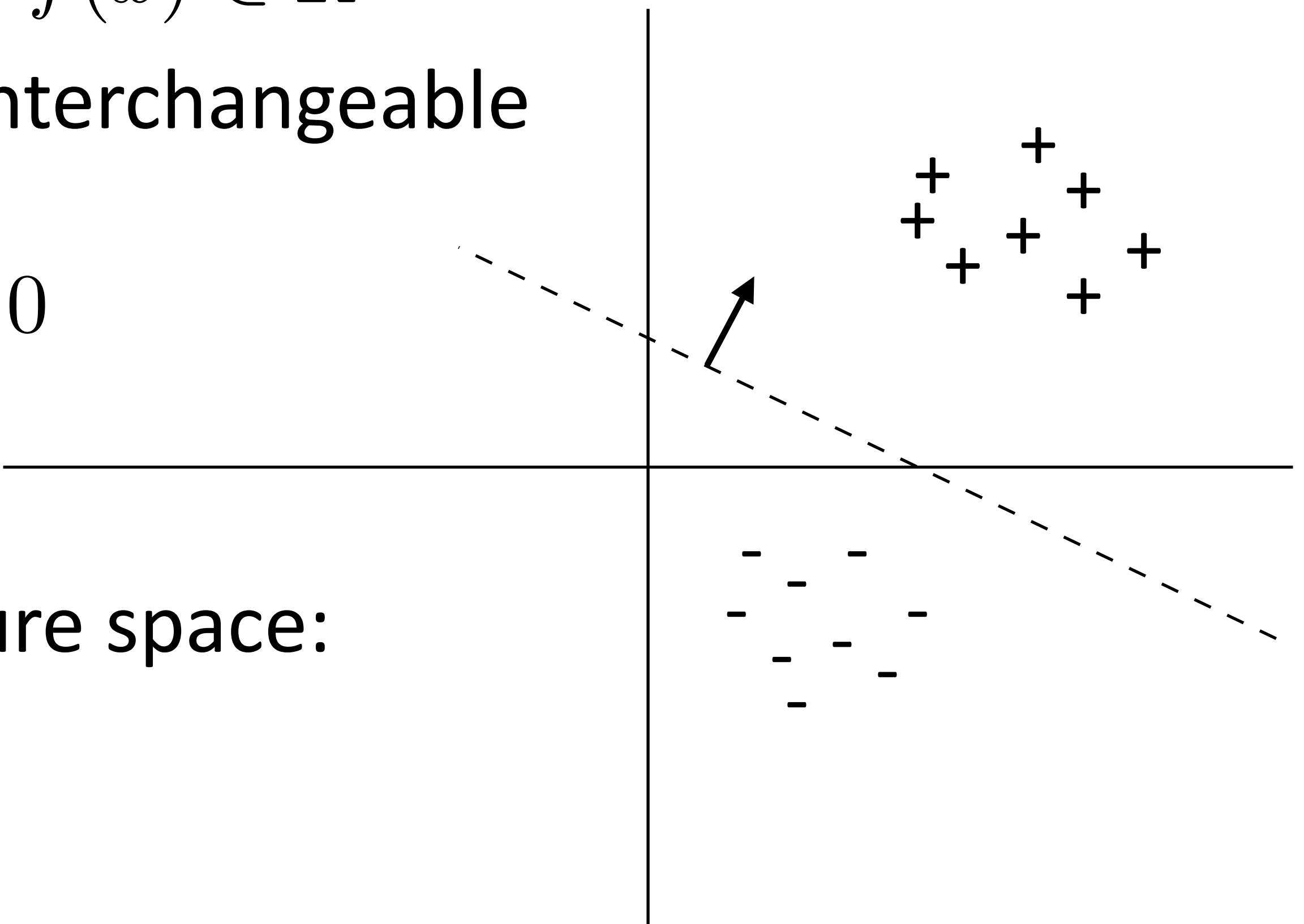
$$w^\top f(x) > 0$$

▶ Can delete bias if we augment feature space:

$$f(x) = [0.5, 1.6, 0.3]$$



$$[0.5, 1.6, 0.3, 1]$$



# Classification

- ▶ Datapoint  $x$  with label  $y \in \{0, 1\}$
- ▶ Embed datapoint in a feature space  $f(x) \in \mathbb{R}^n$   
but in this lecture  $f(x)$  and  $x$  are interchangeable

▶ Linear decision rule:  $w^\top f(x) + b > 0$

$$\underline{w^\top f(x) > 0}$$

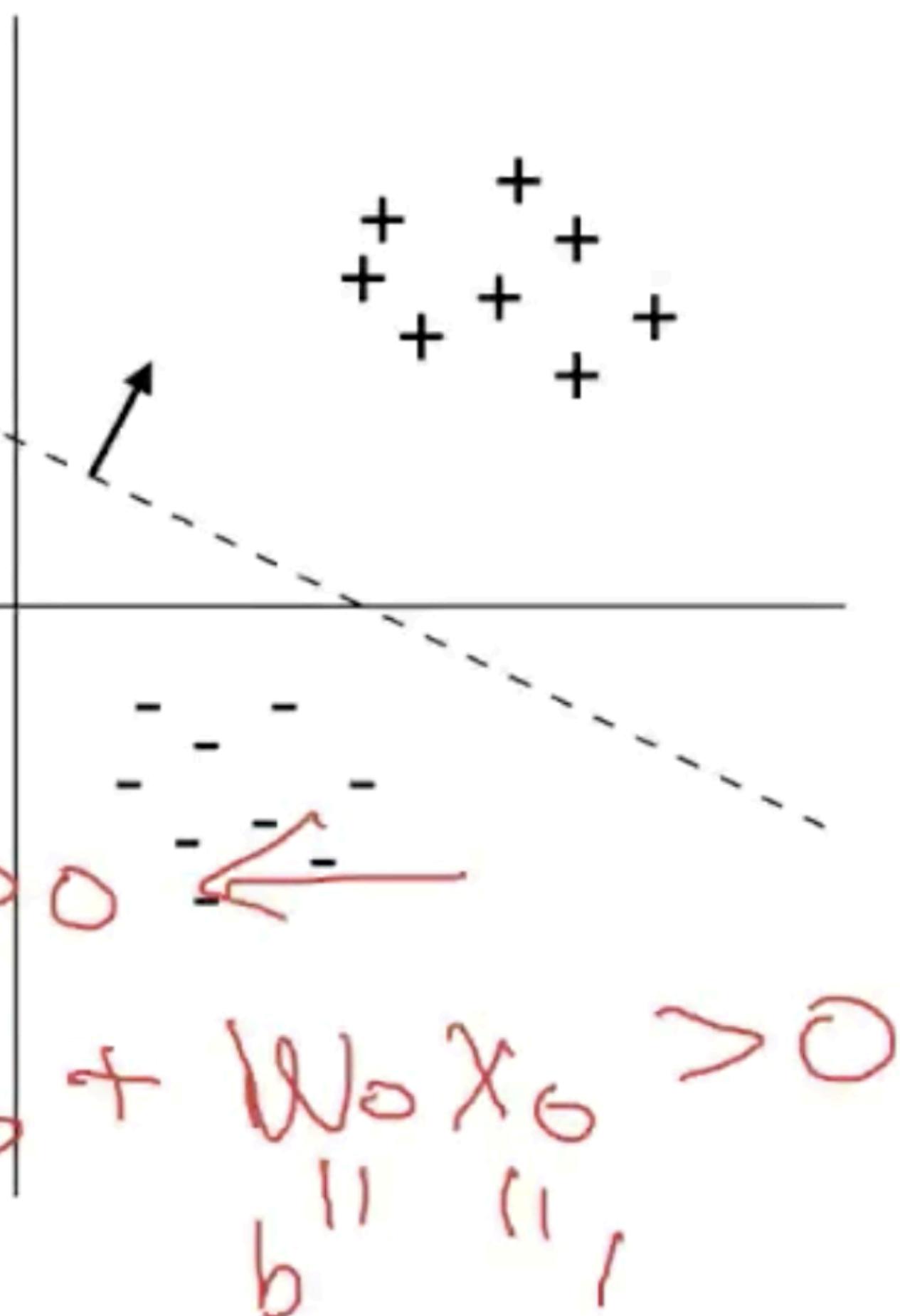
▶ Can delete bias if we augment feature space:

$$f(x) = [x_1, x_2, x_3] \quad w_1 x_1 + w_2 x_2 + w_3 x_3 + \underline{b} > 0$$

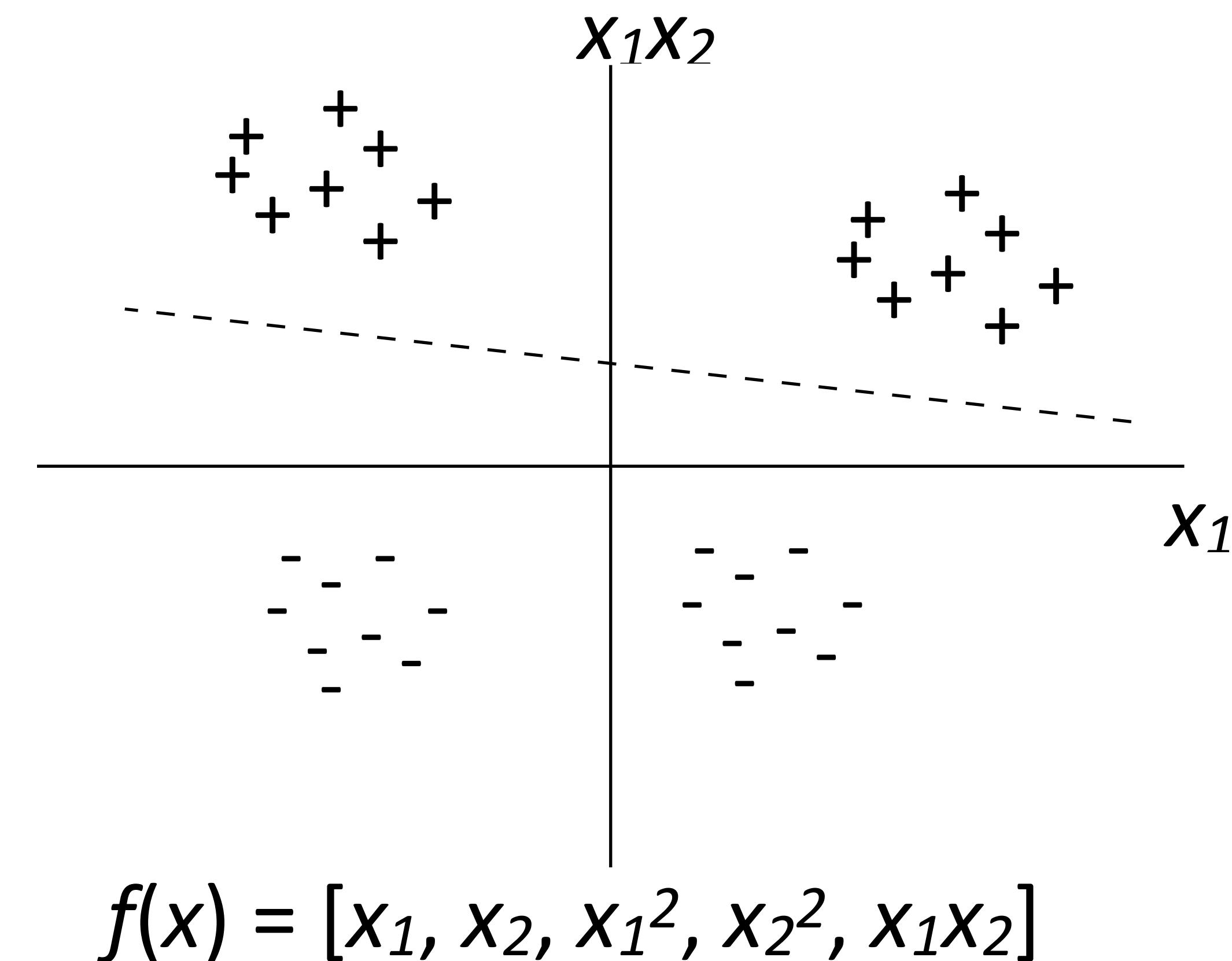
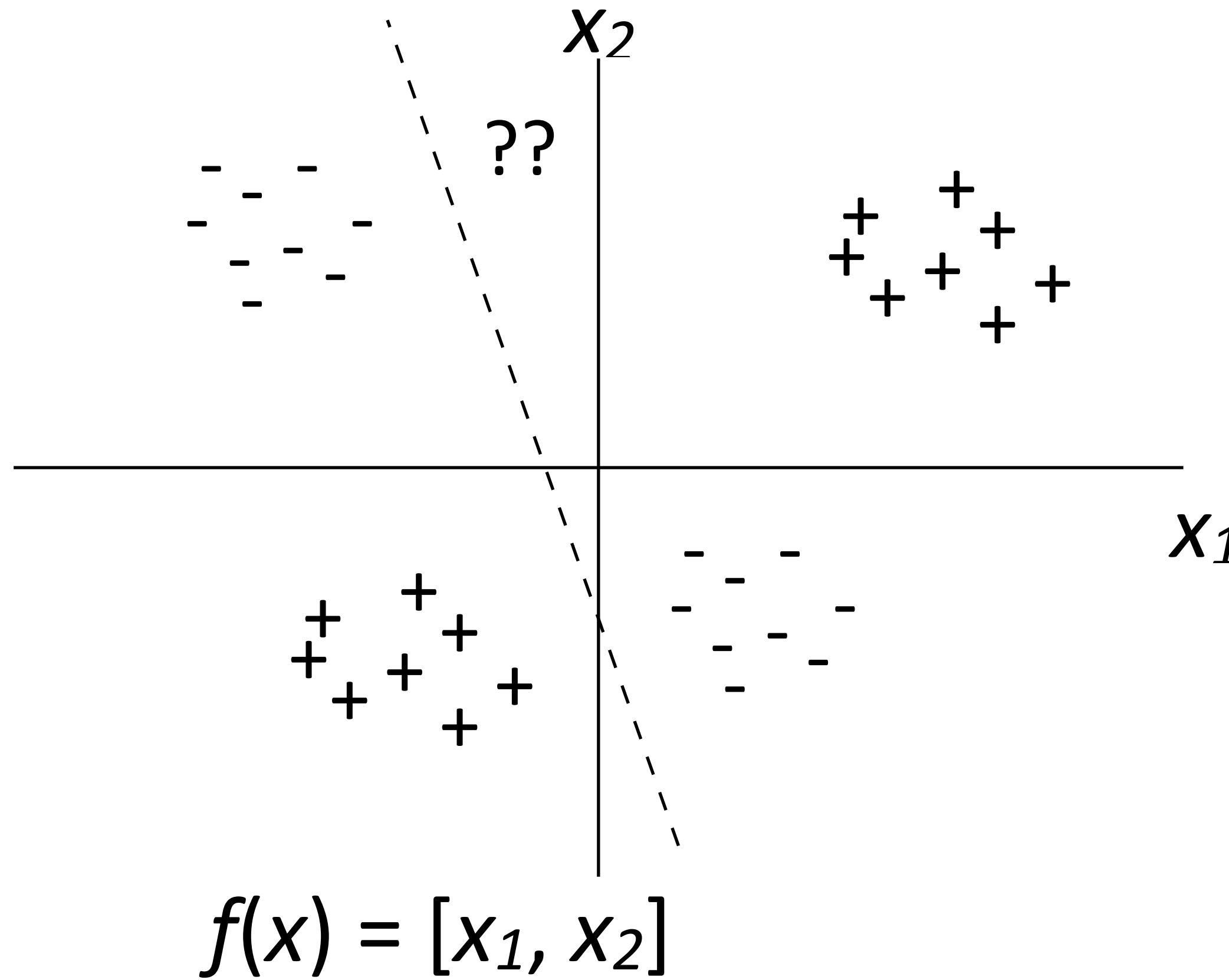
$$\downarrow \quad [0.5, 1.6, 0.3, 1]$$

$$(w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 x_0 + \underline{b''}) > 0$$

$$\sim w^\top f(x) > 0$$



# Linear functions are powerful!



- ▶ “Kernel trick” does this for “free,” but is too expensive to use in NLP applications, training is  $O(n^2)$  instead of  $O(n \cdot (\text{num feats}))$

# Classification: Sentiment Analysis

---

*this movie was great! would watch again*

Positive

*that film was awful, I'll never watch again*

Negative

- ▶ Surface cues can basically tell you what's going on here: presence or absence of certain words (*great, awful*)
- ▶ Steps to classification:
  - ▶ Turn examples like this into feature vectors
  - ▶ Pick a model / learning algorithm
  - ▶ Train weights on data to get our classifier

# Feature Representation

---

*this movie was great! would watch again* Positive

- ▶ Convert this example to a vector using *bag-of-words features*

[contains *the*] [contains *a*] [contains *was*] [contains *movie*] [contains *film*] ...  
position 0 position 1 position 2 position 3 position 4

$$f(x) = [0 \quad 0 \quad 1 \quad 1 \quad 0 \quad \dots]$$

- ▶ Very large vector space (size of vocabulary), sparse features
- ▶ Requires *indexing* the features (mapping them to axes)

# What are features?

---

- ▶ Don't have to be just *bag-of-words*

$$f(x) = \begin{pmatrix} \text{count}(\text{"boring"}) \\ \text{count}(\text{"not boring"}) \\ \text{length of document} \\ \text{author of document} \\ \vdots \end{pmatrix}$$

- ▶ More sophisticated feature mappings possible (tf-idf), as well as lots of other features: character n-grams, parts of speech, lemmas, ...

# Naive Bayes

# Naive Bayes

---

- ▶ Data point  $x = (x_1, \dots, x_n)$ , label  $y \in \{0, 1\}$
- ▶ Formulate a probabilistic model that places a distribution  $P(x, y)$
- ▶ Compute  $P(y|x)$ , predict  $\text{argmax}_y P(y|x)$  to classify

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)}$$

Bayes' Rule

# Naive Bayes

- ▶ Data point  $x = (x_1, \dots, x_n)$ , label  $y \in \{0, 1\}$
- ▶ Formulate a probabilistic model that places a distribution  $P(x, y)$
- ▶ Compute  $P(y|x)$ , predict  $\underbrace{\operatorname{argmax}_y P(y|x)}$  to classify

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)}$$

Bayes' Rule

$$\begin{aligned} P(y=1|x) &= 0.8 \\ P(y=0|x) &= 0.2 \\ \max P(y|x) &= 0.8 \\ \operatorname{argmax}_y P(y|x) &= 1 \end{aligned}$$

# Naive Bayes

- ▶ Data point  $x = (x_1, \dots, x_n)$ , label  $y \in \{0, 1\}$
- ▶ Formulate a probabilistic model that places a distribution  $P(x, y)$
- ▶ Compute  $P(y|x)$ , predict  $\text{argmax}_y P(y|x)$  to classify

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)}$$

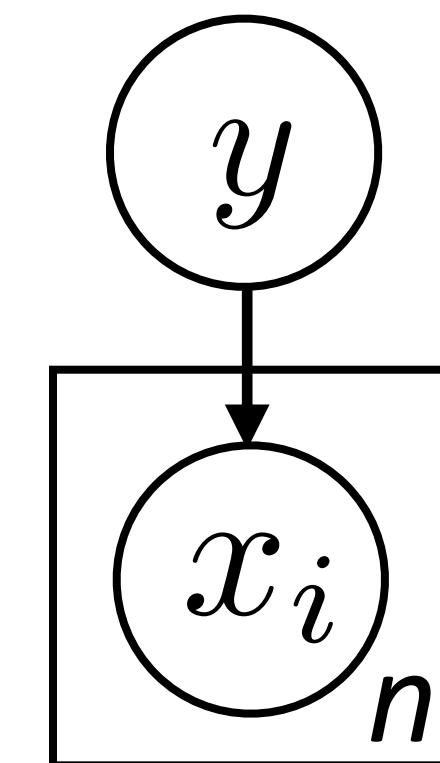
$$\propto P(y)P(x|y)$$

$$= P(y) \prod_{i=1}^n P(x_i|y)$$

Bayes' Rule

constant: irrelevant  
for finding the max

“Naive” assumption:  
conditional independence



$$\text{argmax}_y P(y|x) = \text{argmax}_y \log P(y|x) = \text{argmax}_y \left[ \log P(y) + \sum_{i=1}^n \log P(x_i|y) \right]$$

# Why the log?

---

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)} = P(y) \prod_{i=1}^n P(x_i|y)$$

- ▶ Multiplying together lots of probabilities
- ▶ Probabilities are numbers between 0 and 1

Q: What could go wrong here?

# Why the log?

---

- ▶ Problem — floating point underflow

s	exponent	significand
---	----------	-------------

1    11 bits                        52 bits

Largest =       $1.111\dots \times 2^{+1023}$

Smallest =      $1.000\dots \times 2^{-1024}$

- ▶ Solution: working with probabilities in log space

x	log(x)
0.0000001	-16.118095651
0.000001	-13.815511
0.00001	-11.512925
0.0001	-9.210340
0.001	-6.907755
0.01	-4.605170
0.1	-2.302585

# Maximum Likelihood Estimation

---

- ▶ Data points  $(x_j, y_j)$  provided ( $j$  indexes over examples)
- ▶ Find values of  $P(y)$ ,  $P(x_i|y)$  that maximize data likelihood (generative):

$$\prod_{j=1}^m P(y_j, x_j) = \prod_{j=1}^m P(y_j) \left[ \prod_{i=1}^n P(x_{ji}|y_j) \right]$$

data points (j)    features (i)    *i*th feature of *j*th example

# Maximum Likelihood Estimation

---

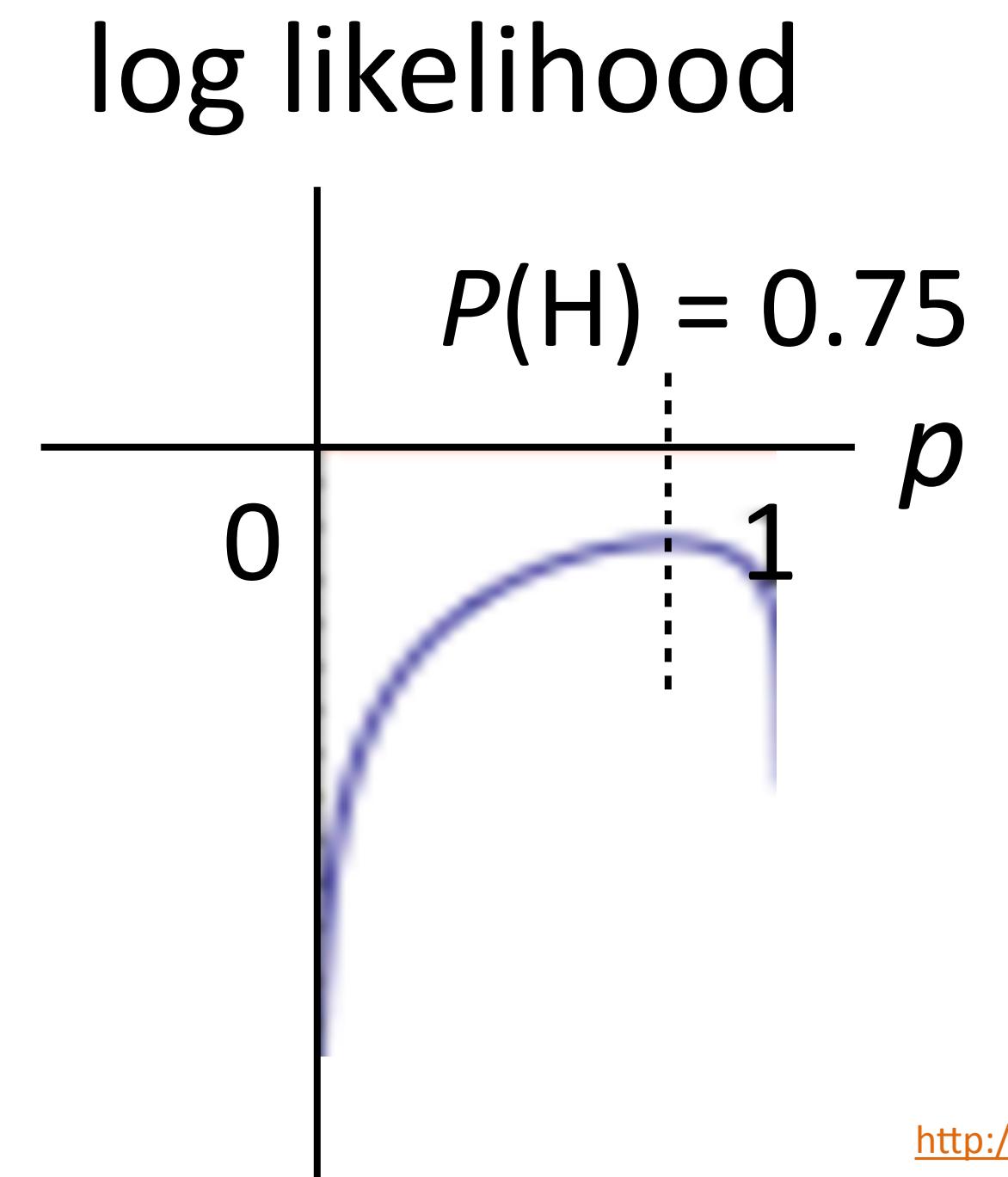
- ▶ Imagine a coin flip which is heads with probability  $p$



- ▶ Observe (H, H, H, T) and maximize likelihood:  $\prod_{j=1}^m P(y_j) = p^3(1 - p)$

- ▶ Easier: maximize *log* likelihood

$$\sum_{j=1}^m \log P(y_j) = 3 \log p + \log(1 - p)$$



# Maximum Likelihood Estimation

- Imagine a coin flip which is heads with probability  $p$

- Observe (H, H, H, T) and maximize likelihood:

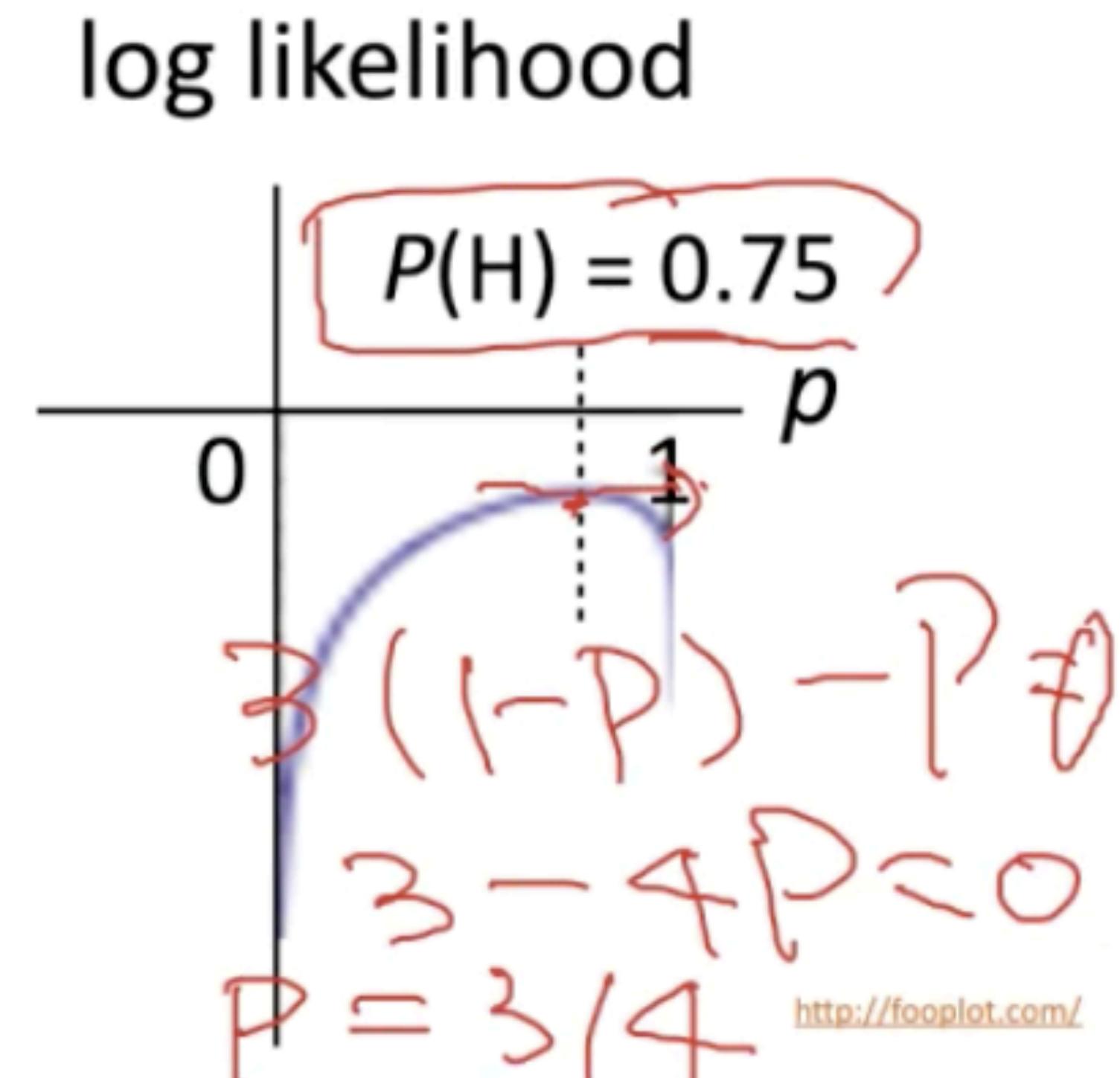
$$P(H, H, H, T) = p^3(1-p)$$

$$\prod_{j=1}^m P(y_j) = p^3(1-p)$$

- Easier: maximize *log likelihood*

$$\sum_{j=1}^m \log P(y_j) = 3 \log p + \log(1-p)$$

$$\frac{\partial \ell(p)}{\partial p} = 3 \cdot \frac{1}{p} + \frac{1}{1-p} (-1) = 0$$



# Maximum Likelihood Estimation

- Imagine a coin flip which is heads with probability  $p$

- Observe (H, H, H, T) and maximize likelihood:

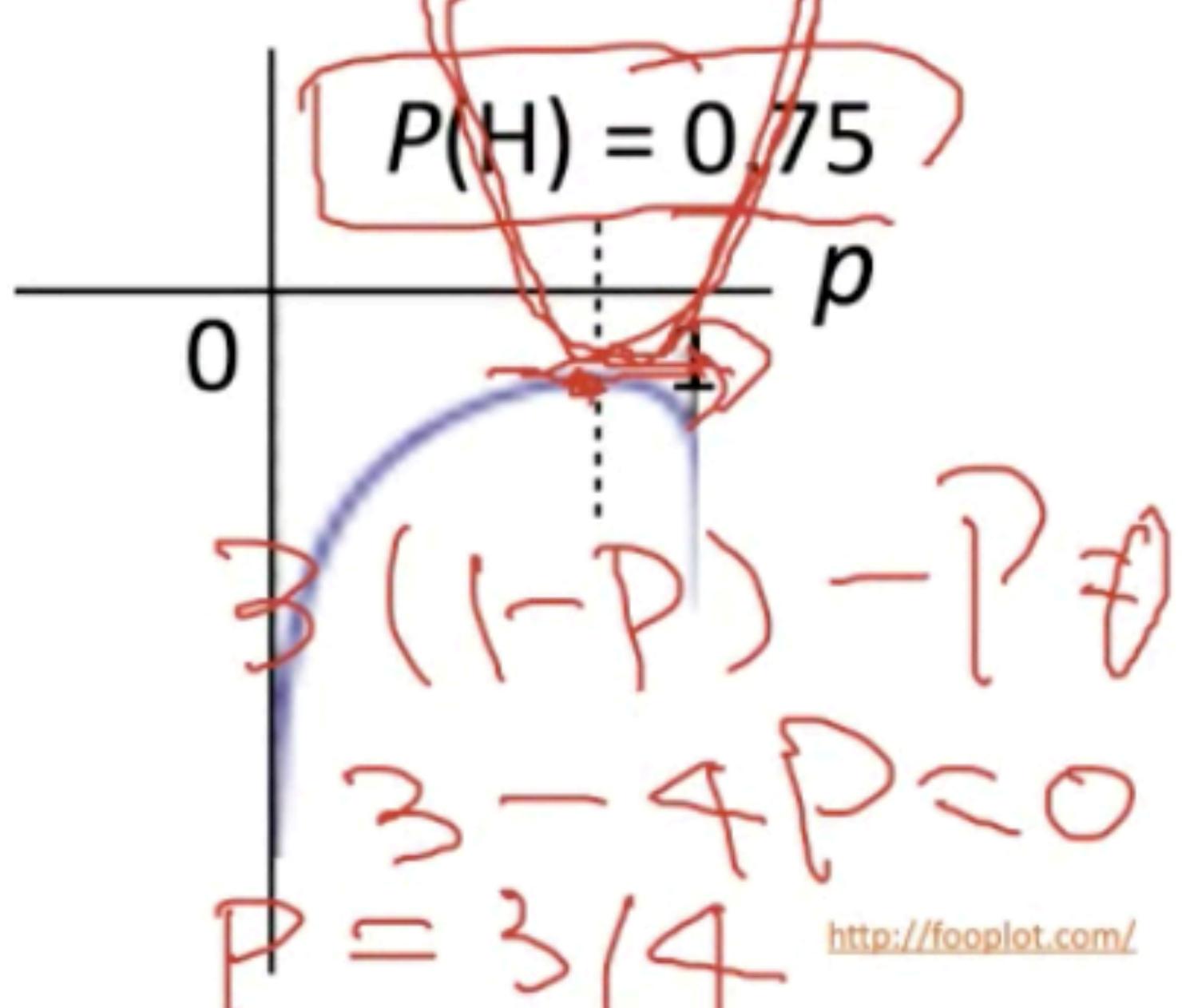
$$P(H, H, H, T) = p^3(1-p)$$

- Easier: maximize *log* likelihood

$$\sum_{j=1}^m \log P(y_j) = 3 \log p + \log(1-p)$$

$$\frac{\partial \ell(p)}{\partial p} = 3 \cdot \frac{1}{p} + \frac{1}{1-p} (-1) = 0$$

$$\begin{aligned} L(p) &= 3 \cdot p^2 + (1-p)^2 \\ \ell(p) &= 3 \log p + \log(1-p) \\ \frac{\partial \ell(p)}{\partial p} &= \frac{3}{p} - \frac{1}{1-p} = 0 \\ 3p - 1 &= 0 \\ p &= \frac{1}{3} \end{aligned}$$



# Maximum Likelihood Estimation

- ▶ Imagine a coin flip which is heads with probability  $p$

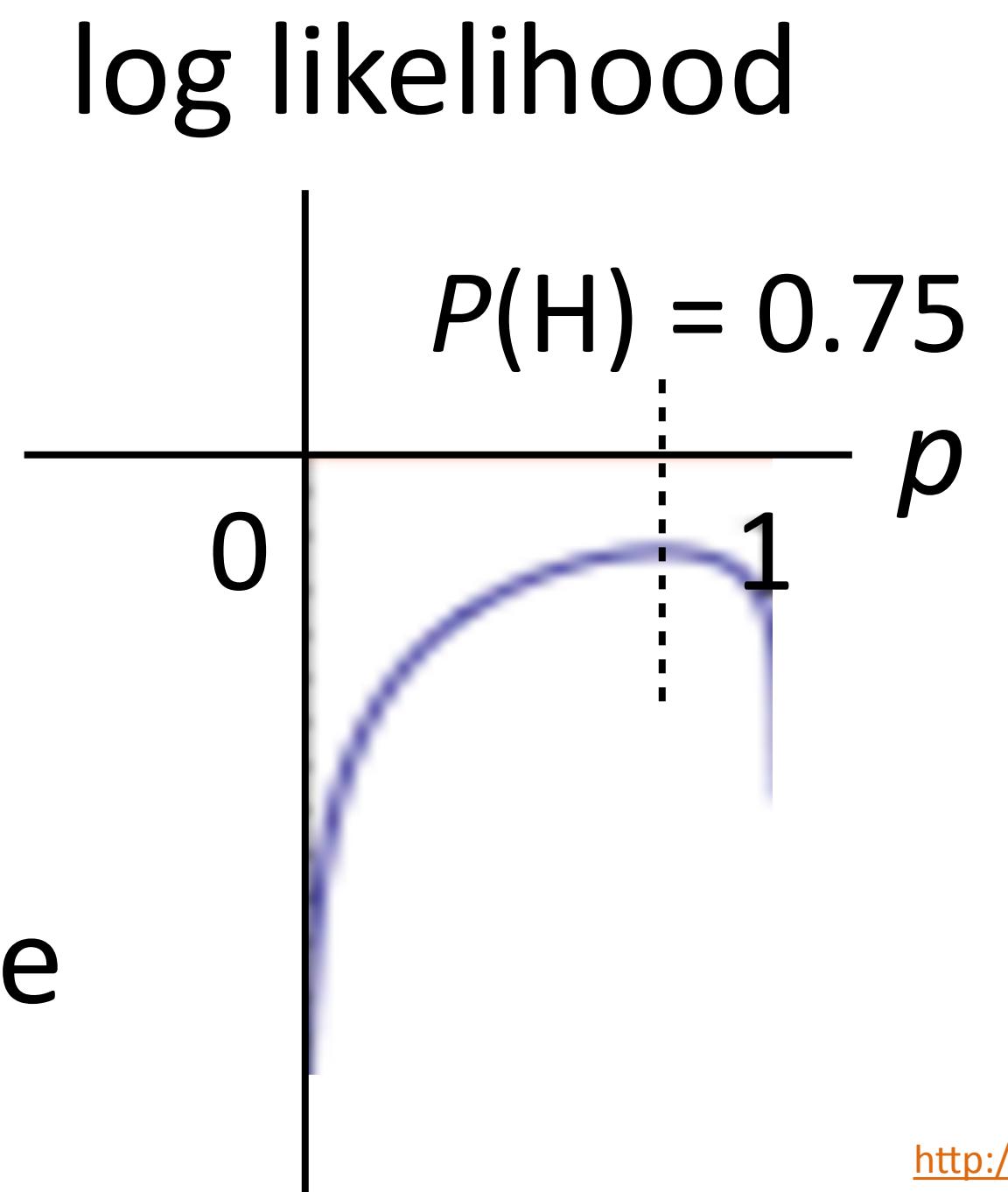


- ▶ Observe (H, H, H, T) and maximize likelihood:  $\prod_{j=1}^m P(y_j) = p^3(1 - p)$

- ▶ Easier: maximize *log* likelihood

$$\sum_{j=1}^m \log P(y_j) = 3 \log p + \log(1 - p)$$

- ▶ Maximum likelihood parameters for binomial/multinomial = read counts off of the data + normalize



# Maximum Likelihood Estimation

---

- ▶ Data points  $(x_j, y_j)$  provided ( $j$  indexes over examples)
- ▶ Find values of  $P(y)$ ,  $P(x_i|y)$  that maximize data likelihood (generative):

$$\prod_{j=1}^m P(y_j, x_j) = \prod_{j=1}^m P(y_j) \left[ \prod_{i=1}^n P(x_{ji}|y_j) \right]$$

data points (j)    features (i)     $i$ th feature of  $j$ th example

- ▶ Equivalent to maximizing logarithm of data likelihood:

$$\sum_{j=1}^m \log P(y_j, x_j) = \sum_{j=1}^m \left[ \log P(y_j) + \sum_{i=1}^n \log P(x_{ji}|y_j) \right]$$

# Maximum Likelihood for Naive Bayes

this movie was great! would watch again

+

I liked it well enough for an action flick

+

I expected a great film and left happy  
brilliant directing and stunning visuals

+

that film was awful, I'll never watch again

-

I didn't really like that movie

-

dry and a bit distasteful, it misses the mark

-

great potential but ended up being a flop

-

$$P(+) = \frac{1}{2}$$

$$P(-) = \frac{1}{2}$$

$$P(\text{great}|+) = \frac{1}{2}$$

$$P(\text{great}|-) = \frac{1}{4}$$

$$P(y|x) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

$$\text{it was great} \longrightarrow P(y|x) \propto \begin{bmatrix} P(+)P(\text{great}|+) \\ P(-)P(\text{great}|-) \end{bmatrix} = \begin{bmatrix} 1/4 \\ 1/8 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1/3 \end{bmatrix}$$

prior

word likelihood

# Naive Bayes: Learning

---

$$P(y|x) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

- ▶ Learning = estimate the parameters of the model
  - ▶ Prior probability —  $P(+)$  and  $P(-)$ :
    - ▶ fraction of + (or -) documents among all documents
  - ▶ Word likelihood —  $P(\text{word}_i | +)$  and  $P(\text{word}_i | -)$ :
    - ▶ number of + (or -) documents  $\text{word}_i$  is observed, divide by the total number of documents of + (or -) documents

This is for Bernoulli (binary features) document model!

# Zero Probability Problem

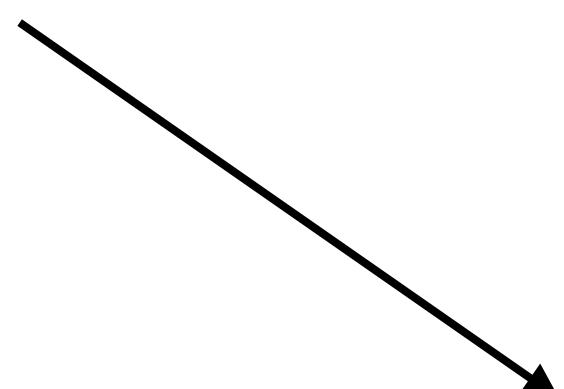
---

- ▶ What if we have seen no training document with the word “fantastic” and classified in the topic positive?
- ▶ Laplace (add-1) Smoothing
  - ▶ Word likelihood —  $P(\text{word}_i | +)$  and  $P(\text{word}_i | -)$ :
    - ▶ frequency of  $\text{word}_i$  is observed **plus 1**, divide by ...

# Naive Bayes

---

- ▶ Bernoulli document model:
  - ▶ A document is represented by binary features
  - ▶ Feature value be 1 if the corresponding word is represent in the document and 0 if not
- ▶ Multinomial document model:
  - ▶ A document is represented by integer elements
  - ▶ Feature value is the frequency of that word in the document
  - ▶ See textbook and lecture note by Hiroshi Shimodaira linked below for more details



# Naive Bayes

## Text Classification using Naive Bayes

Hiroshi Shimodaira\*

10 February 2015

Text classification is the task of classifying documents by their content: that is, by the words of which they are comprised. Perhaps the best-known current text classification problem is email *spam filtering*: classifying email messages into spam and non-spam (ham).

### 1 Document models

Text classifiers often don't use any kind of deep representation about language: often a document is represented as a *bag of words*. (A bag is like a set that allows repeating elements.) This is an extremely simple representation: it only knows which words are included in the document (and how many times each word occurs), and throws away the word order!

Consider a document  $D$ , whose class is given by  $C$ . In the case of email spam filtering there are two classes  $C = S$  (spam) and  $C = H$  (ham). We classify  $D$  as the class which has the highest posterior probability  $P(C|D)$ , which can be re-expressed using Bayes' Theorem:

$$P(C|D) = \frac{P(D|C) P(C)}{P(D)} \propto P(D|C) P(C). \quad (1)$$

We shall look at two probabilistic models of documents, both of which represent documents as a bag of words, using the Naive Bayes assumption. Both models represent documents using feature vectors whose components correspond to word types. If we have a vocabulary  $V$ , containing  $|V|$  word types, then the feature vector dimension  $d = |V|$ .

**Bernoulli document model:** a document is represented by a feature vector with binary elements taking value 1 if the corresponding word is present in the document and 0 if the word is not present.

**Multinomial document model:** a document is represented by a feature vector with integer elements whose value is the frequency of that word in the document.

**Example:** Consider the vocabulary:

$$V = \{\text{blue, red, dog, cat, biscuit, apple}\}.$$

In this case  $|V| = d = 6$ . Now consider the (short) document "the blue dog ate a blue biscuit". If  $\mathbf{d}^B$  is the Bernoulli feature vector for this document, and  $\mathbf{d}^M$  is the multinomial feature vector, then we

\*Heavily based on notes inherited from Steve Renals and Iain Murray.

would have:

$$\begin{aligned}\mathbf{d}^B &= (1, 0, 1, 0, 1, 0)^T \\ \mathbf{d}^M &= (2, 0, 1, 0, 1, 0)^T\end{aligned}$$

To classify a document we use equation (1), which requires estimating the likelihoods of the document given the class,  $P(D|C)$  and the class prior probabilities  $P(C)$ . To estimate the likelihood,  $P(D|C)$ , we use the Naive Bayes assumption applied to whichever of the two document models we are using.

### 2 The Bernoulli document model

As mentioned above, in the Bernoulli model a document is represented by a binary vector, which represents a point in the space of words. If we have a vocabulary  $V$  containing a set of  $|V|$  words, then the  $t$ th dimension of a document vector corresponds to word  $w_t$  in the vocabulary. Let  $\mathbf{b}_i$  be the feature vector for the  $i$ th document  $D_i$ ; then the  $t$ th element of  $\mathbf{b}_i$ , written  $b_{it}$ , is either 0 or 1 representing the absence or presence of word  $w_t$  in the  $i$ th document.

Let  $P(w_t|C)$  be the probability of word  $w_t$  occurring in a document of class  $C$ ; the probability of  $w_t$  not occurring in a document of this class is given by  $(1 - P(w_t|C))$ . If we make the naive Bayes assumption, that the probability of each word occurring in the document is independent of the occurrences of the other words, then we can write the document likelihood  $P(D_i|C)$  in terms of the individual word likelihoods  $P(w_t|C)$ :

$$P(D_i|C) \sim P(\mathbf{b}_i|C) = \prod_{t=1}^{|V|} [b_{it}P(w_t|C) + (1 - b_{it})(1 - P(w_t|C))]. \quad (2)$$

This product goes over all words in the vocabulary. If word  $w_t$  is present, then  $b_{it}=1$  and the required probability is  $P(w_t|C)$ ; if word  $w_t$  is not present, then  $b_{it}=0$  and the required probability is  $1 - P(w_t|C)$ . We can imagine this as a model for generating document feature vectors of class  $C$ , in which the document feature vector is modelled as a collection of  $|V|$  weighted coin tosses, the  $t$ th having a probability of success equal to  $P(w_t|C)$ .

The *parameters* of the likelihoods are the probabilities of each word given the document class  $P(w_t|C)$ ; the model is also parameterised by the prior probabilities,  $P(C)$ . We can learn (estimate) these parameters from a training set of documents labelled with class  $C=k$ . Let  $n_k(w_t)$  be the number of documents of class  $C=k$  in which  $w_t$  is observed; and let  $N_k$  be the total number of documents of that class. Then we can estimate the parameters of the word likelihoods as,

$$\hat{P}(w_t | C=k) = \frac{n_k(w_t)}{N_k}, \quad (3)$$

the relative frequency of documents of class  $C=k$  that contain word  $w_t$ . If there are  $N$  documents in total in the training set, then the prior probability of class  $C=k$  may be estimated as the relative frequency of documents of class  $C=k$ :

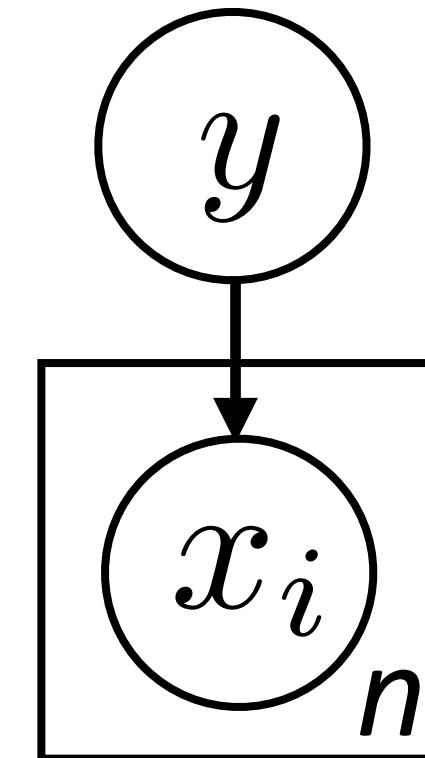
$$\hat{P}(C=k) = \frac{N_k}{N}. \quad (4)$$

Thus given a training set of documents (each labelled with a class), and a set of  $K$  classes, we can estimate a Bernoulli text classification model as follows:

# Naive Bayes: Summary

- ▶ Model

$$P(x, y) = P(y) \prod_{i=1}^n P(x_i|y)$$



- ▶ Inference

$$\operatorname{argmax}_y \log P(y|x) = \operatorname{argmax}_y \left[ \log P(y) + \sum_{i=1}^n \log P(x_i|y) \right]$$

- ▶ Alternatively:  $\log P(y=+|x) - \log P(y=-|x) > 0$

$$\Leftrightarrow \log \frac{P(y=+)}{P(y=-)} + \sum_{i=1}^n \log \frac{P(x_i|y=+)}{P(x_i|y=-)} > 0$$

Linear model!  
 $w^\top f(x) > 0$

- ▶ Learning: maximize  $P(x, y)$  by reading counts off the data

# Problems with Naive Bayes

---

*the film was beautiful, stunning cinematography and gorgeous sets, but boring —*

$$P(x_{\text{beautiful}}|+) = 0.1 \quad P(x_{\text{beautiful}}|-) = 0.01$$

$$P(x_{\text{stunning}}|+) = 0.1 \quad P(x_{\text{stunning}}|-) = 0.01$$

$$P(x_{\text{gorgeous}}|+) = 0.1 \quad P(x_{\text{gorgeous}}|-) = 0.01$$

$$P(x_{\text{boring}}|+) = 0.01 \quad P(x_{\text{boring}}|-) = 0.1$$

- ▶ Correlated features compound: *beautiful* and *gorgeous* are not independent!
- ▶ Naive Bayes is naive, but another problem is that it's *generative*: spends capacity modeling  $P(x,y)$ , when what we care about is  $P(y|x)$
- ▶ Discriminative models model  $P(y|x)$  directly (SVMs, most neural networks, ...)

# Generative vs. Discriminative Models

---

- ▶ Generative models:  $P(x, y)$ 
  - ▶ Bayes nets / graphical models
  - ▶ Some of the model capacity goes to explaining the distribution of  $x$ ;  
prediction uses Bayes rule post-hoc
  - ▶ Can sample new instances  $(x, y)$
- ▶ Discriminative models:  $P(y|x)$ 
  - ▶ SVMs, logistic regression, CRFs, most neural networks
  - ▶ Model is trained to be good at prediction, but doesn't model  $x$
  - ▶ We'll come back to this distinction throughout this class

# Logistic Regression

# Logistic Regression

$$P(y = +|x) = \text{logistic}(w^\top x)$$

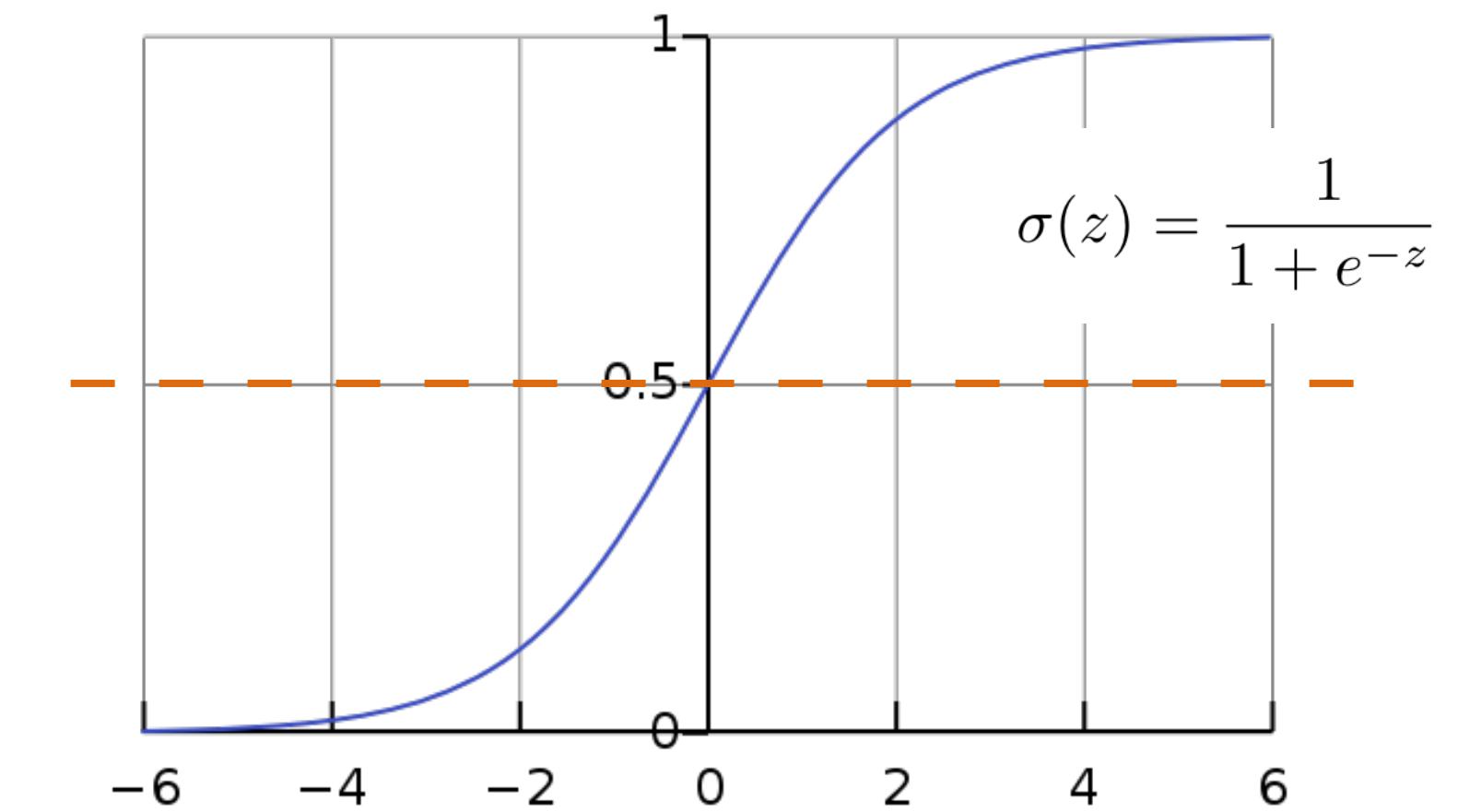
$$P(y = +|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{1 + \exp(\sum_{i=1}^n w_i x_i)}$$

► Decision rule:  $P(y = +|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$

► To learn weights: maximize discriminative log likelihood of data  $P(y|x)$

$$\mathcal{L}(x_j, y_j = +) = \log P(y_j = +|x_j)$$

$$\begin{aligned} &= \sum_{i=1}^n w_i x_{ji} - \log \left( 1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right) \right) \\ &\text{sum over features} \end{aligned}$$



# Logistic Regression

$$P(y = +|x) = \text{logistic}(w^\top x)$$

$$P(y = +|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{1 + \exp(\sum_{i=1}^n w_i x_i)}$$

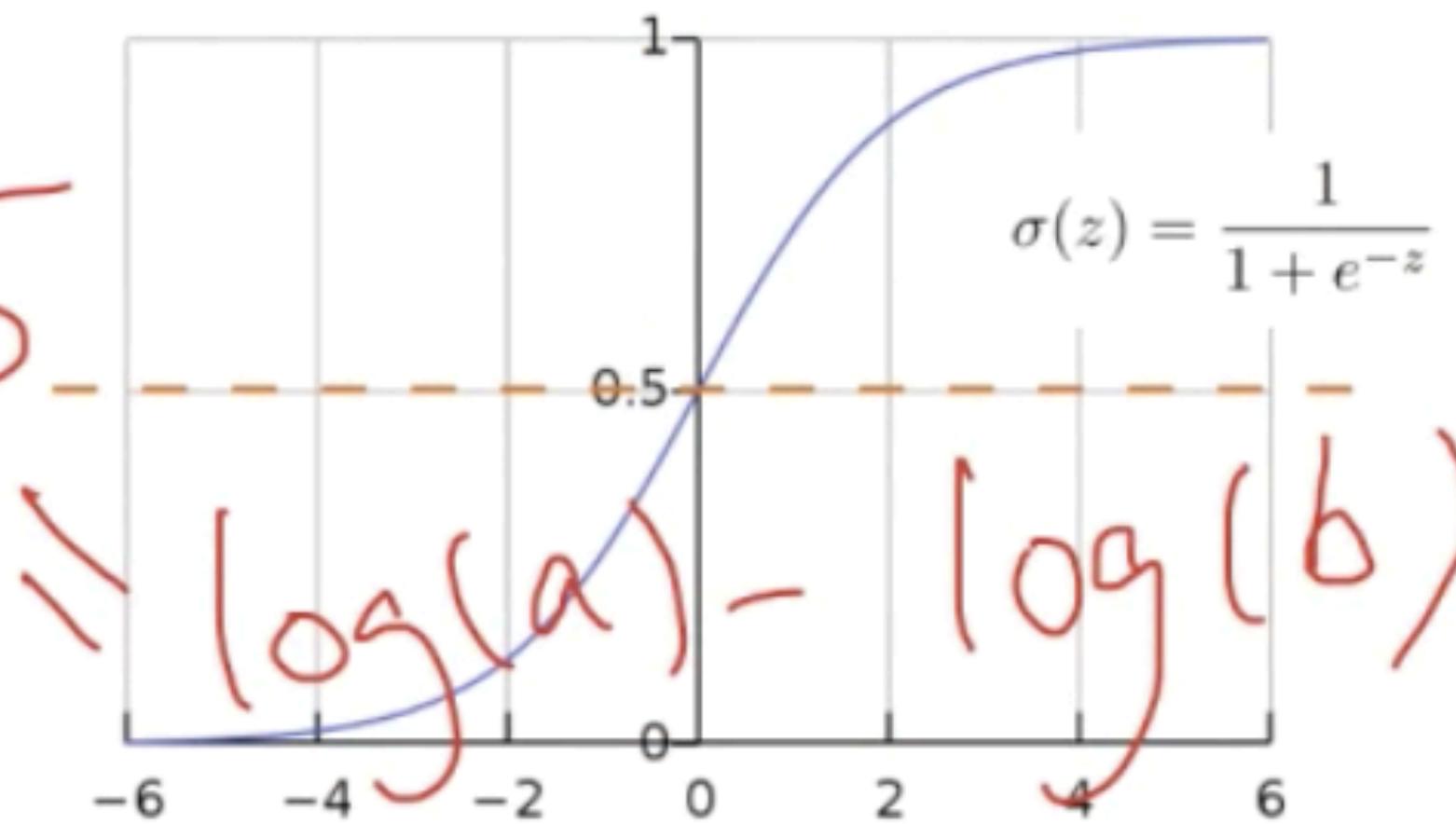
► Decision rule:  $P(y = +|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$

► To learn weights: maximize discriminative log likelihood of data  $P(y|x)$

$$\mathcal{L}(x_j, y_j = +) = \log P(y_j = +|x_j)$$

$$= \sum_{i=1}^n w_i x_{ji} - \log \left( 1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right) \right)$$

sum over features



~~log (RXP(c))~~

C

chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} = \frac{\partial f(g)}{\partial g} \frac{\partial g(x)}{\partial x}$$

# Logistic Regression

$$\mathcal{L}(x_j, y_j = +) = \log P(y_j = + | x_j) = \boxed{\sum_{i=1}^n w_i x_{ji}} - \log \left( 1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right) \right)$$

$$\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w_i} = \boxed{x_{ji}} - \frac{\partial}{\partial w_i} \log \left( \boxed{1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right)} \right)$$

$$= x_{ji} - \frac{1}{\boxed{1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right)}} \frac{\partial}{\partial w_i} \left( \boxed{1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right)} \right)$$

$$= x_{ji} - \frac{1}{1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right)} x_{ji} \exp \left( \sum_{i=1}^n w_i x_{ji} \right)$$

deriv. of log  
 $\frac{\partial \log x}{\partial x} = \frac{1}{x}$

$$= x_{ji} - x_{ji} \frac{\exp \left( \sum_{i=1}^n w_i x_{ji} \right)}{1 + \exp \left( \sum_{i=1}^n w_i x_{ji} \right)} = x_{ji} (1 - P(y_j = + | x_j))$$

deriv. of exp  
 $\frac{\partial e^x}{\partial x} = e^x$

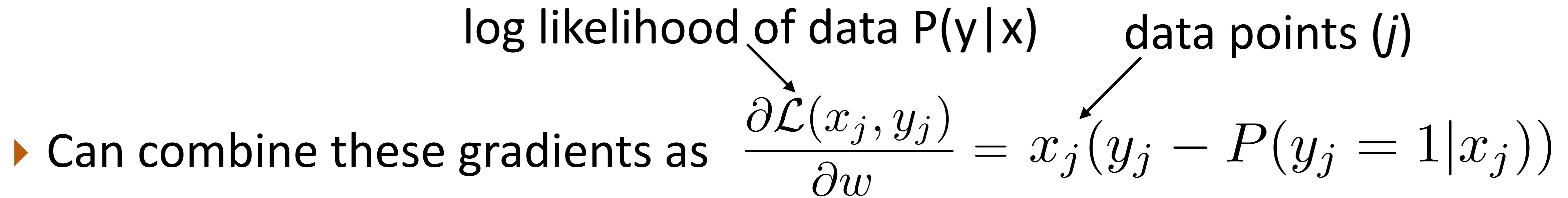
# Logistic Regression

---

- ▶ Recall that  $y_j = 1$  for positive instances,  $y_j = 0$  for negative instances.
- ▶ Gradient of  $w_i$  on positive example  $= x_{ji}(y_j - P(y_j = +|x_j))$ 
  - If  $P(+)$  is close to 1, make very little update
  - Otherwise make  $w_i$  look more like  $x_{ji}$ , which will increase  $P(+)$
- ▶ Gradient of  $w_i$  on negative example  $= x_{ji}(-P(y_j = +|x_j))$ 
  - If  $P(+)$  is close to 0, make very little update
  - Otherwise make  $w_i$  look less like  $x_{ji}$ , which will decrease  $P(+)$
- ▶ Can combine these gradients as  $\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$

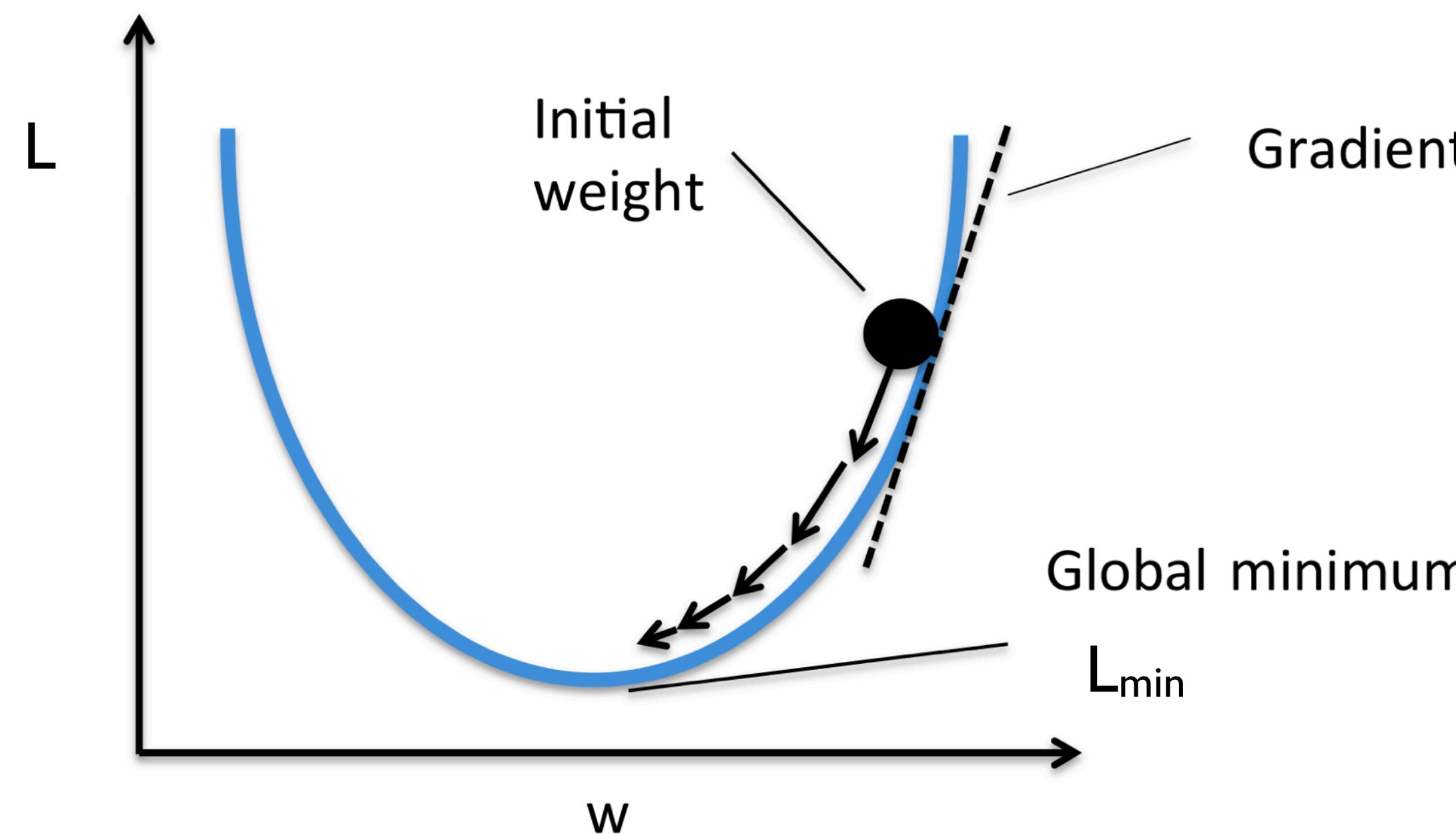
# Gradient Decent

---

- ▶ Can combine these gradients as  $\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$   

- ▶ Training set log-likelihood:  $\mathcal{L}(w) = \frac{1}{m} \sum_{j=1}^m \mathcal{L}(x_j, y_j)$
- ▶ Gradient vector:  $\frac{\partial \mathcal{L}(w)}{\partial w} = \left( \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right)$

# Gradient Decent

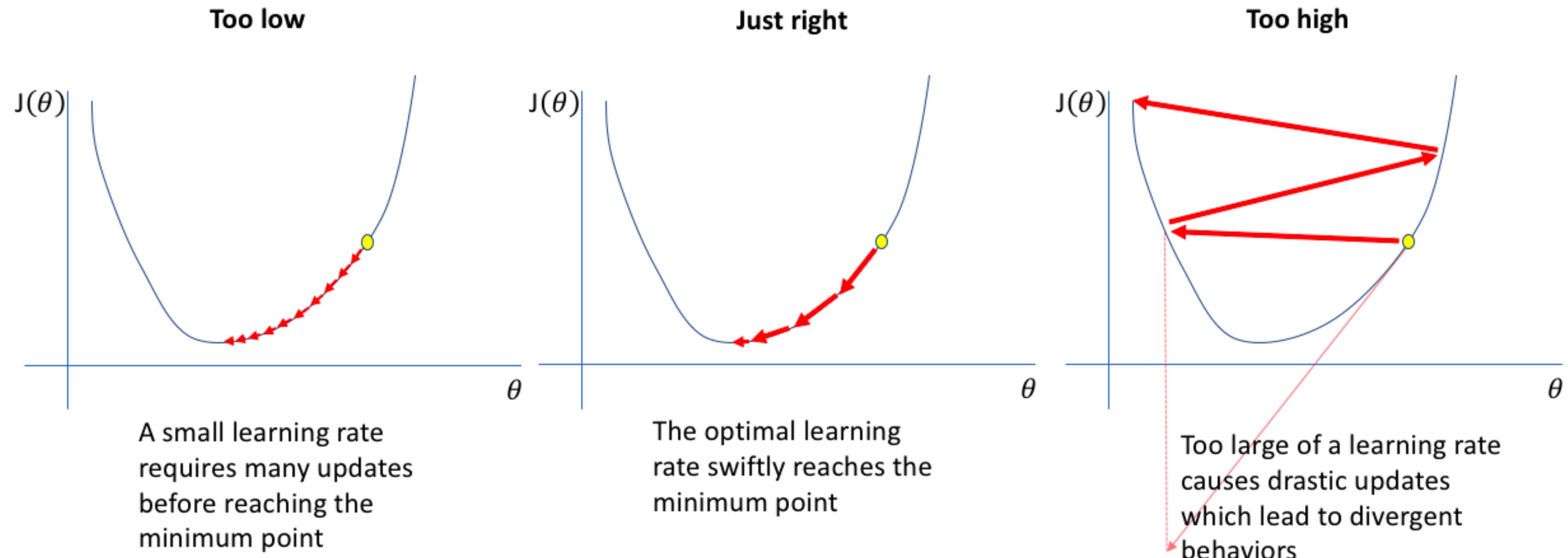
- ▶ Gradient decent (or ascent) is an iterative optimization algorithm for finding the minimum (or maximum) of a function.



Repeat until convergence {  
     $w := w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$   
}

learning rate (step size)

# Learning Rate



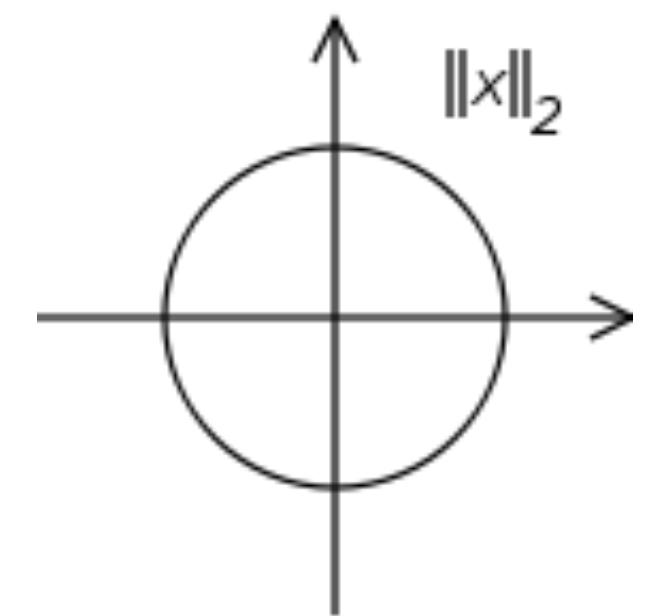
Credit: Jeremy Jordan

# Regularization

---

- ▶ Regularizing an objective can mean many things, including an L2-norm penalty to the weights:

$$\sum_{j=1}^m \mathcal{L}(x_j, y_j) - \lambda \|w\|_2^2$$



- ▶ Keeping weights small can prevent overfitting
- ▶ For most of the NLP models we build, explicit regularization isn't necessary
  - ▶ Early stopping
  - ▶ Large numbers of sparse features are hard to overfit in a really bad way
  - ▶ For neural networks: dropout and gradient clipping

# Logistic Regression: Summary

---

- ▶ Model

$$P(y = +|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{1 + \exp(\sum_{i=1}^n w_i x_i)}$$

- ▶ Inference

$\operatorname{argmax}_y P(y|x)$  fundamentally same as Naive Bayes

$$P(y = 1|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$$

- ▶ Learning: gradient ascent on the (regularized) discriminative log-likelihood

# Logistic Regression vs. Naive Bayes

---

- ▶ Both are (log) linear models  $w^\top f(x)$
- ▶ Logistic regression doesn't assume conditional independence of features
  - ▶ Weights are trained independently
  - ▶ Can handle highly correlated overlapping features
- ▶ Naive Bayes assume conditional independence of features
  - ▶ Weights are trained jointly

# Perceptron/SVM

# Perceptron

---

- ▶ Simple error-driven learning approach similar to logistic regression
- ▶ Decision rule:  $w^T x > 0$ 
  - ▶ If incorrect: if positive,  $w \leftarrow w + x$
  - if negative,  $w \leftarrow w - x$
- ▶ Algorithm is very similar to logistic regression
- ▶ Guaranteed to eventually separate the data if the data are separable

## Logistic Regression

$$w \leftarrow w + x(1 - P(y = 1|x))$$

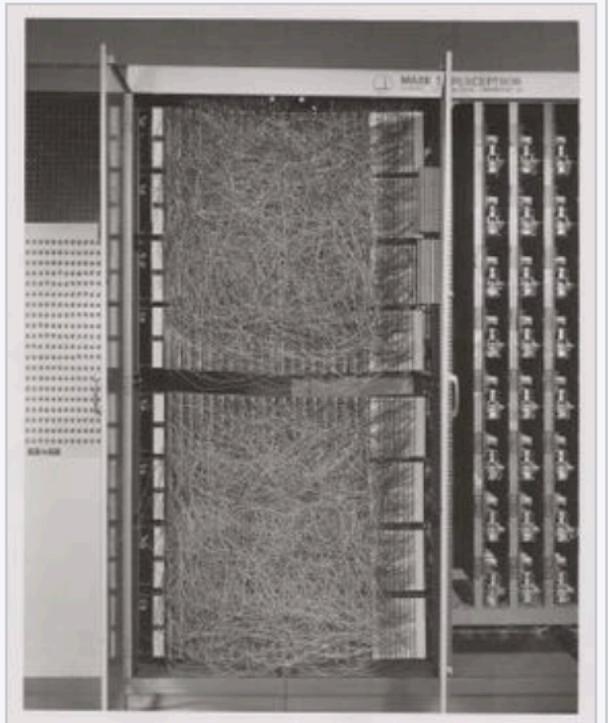
$$w \leftarrow w - xP(y = 1|x)$$

# Perceptron

## History [ edit ]

V · T · E

See also: [History of artificial intelligence](#) § [Perceptrons and the attack on connectionism](#), and [AI winter](#) § [The abandonment of connectionism in 1969](#)



Mark I Perceptron machine, the first implementation of the perceptron algorithm. It was connected to a camera with 20x20 cadmium sulfide photocells to make a 400-pixel image. The main visible feature is a patch panel that set different combinations of input features. To the right, arrays of potentiometers that implemented the adaptive weights.<sup>[2]:213</sup>

original text are shown and corrected.

The kernel perceptron algorithm was already introduced in 1964 by Aizerman et al.<sup>[5]</sup> Margin bounds guarantees were given for the Perceptron algorithm in the general non-separable case first by Freund and Schapire (1998),<sup>[1]</sup> and more recently by Mohri and Rostamizadeh (2013) who extend previous results and give new L1 bounds.<sup>[6]</sup>

The perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.<sup>[7]</sup>



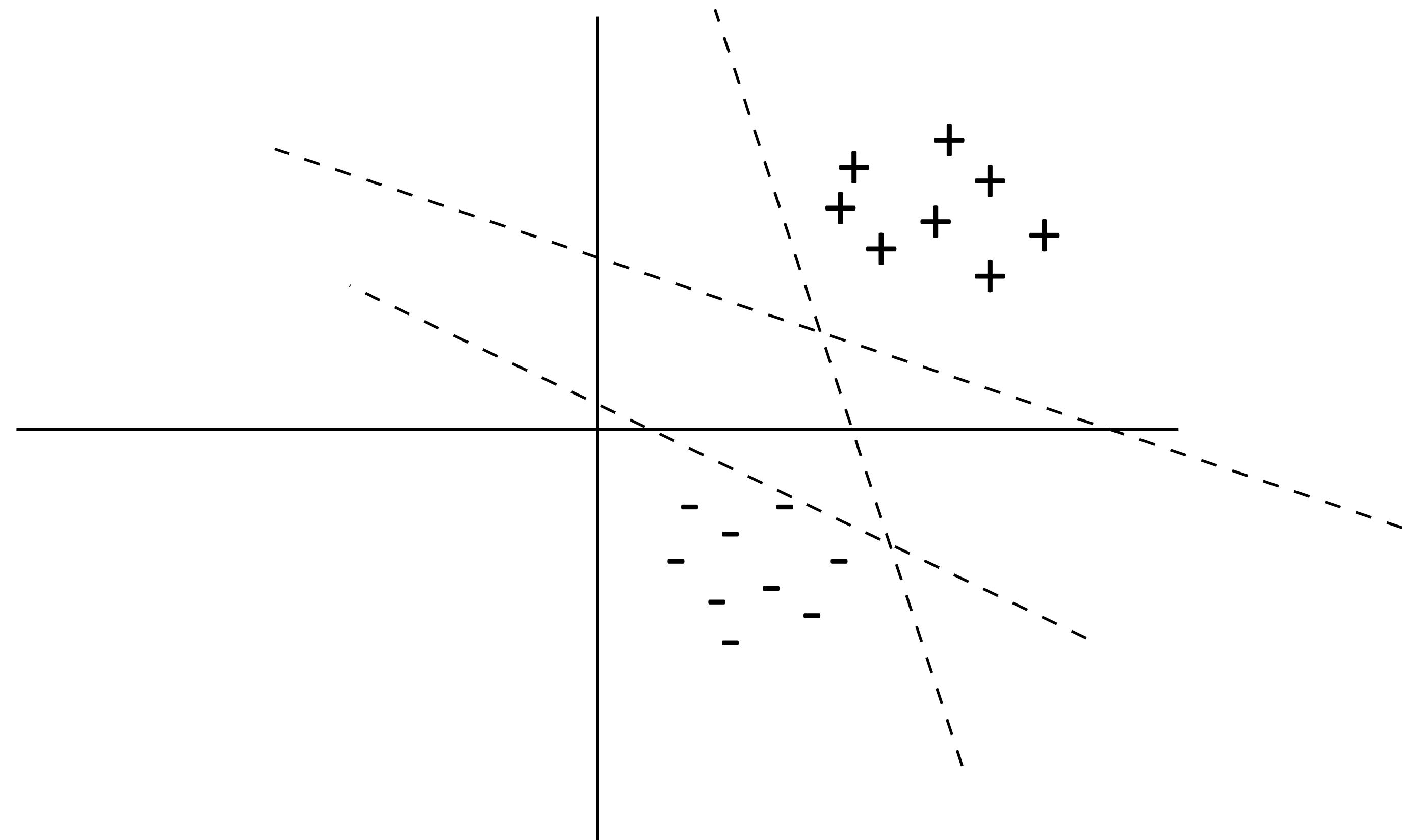
Frank Rosenblatt (1928-1971)

PhD 1956 from Cornell

# Support Vector Machines

---

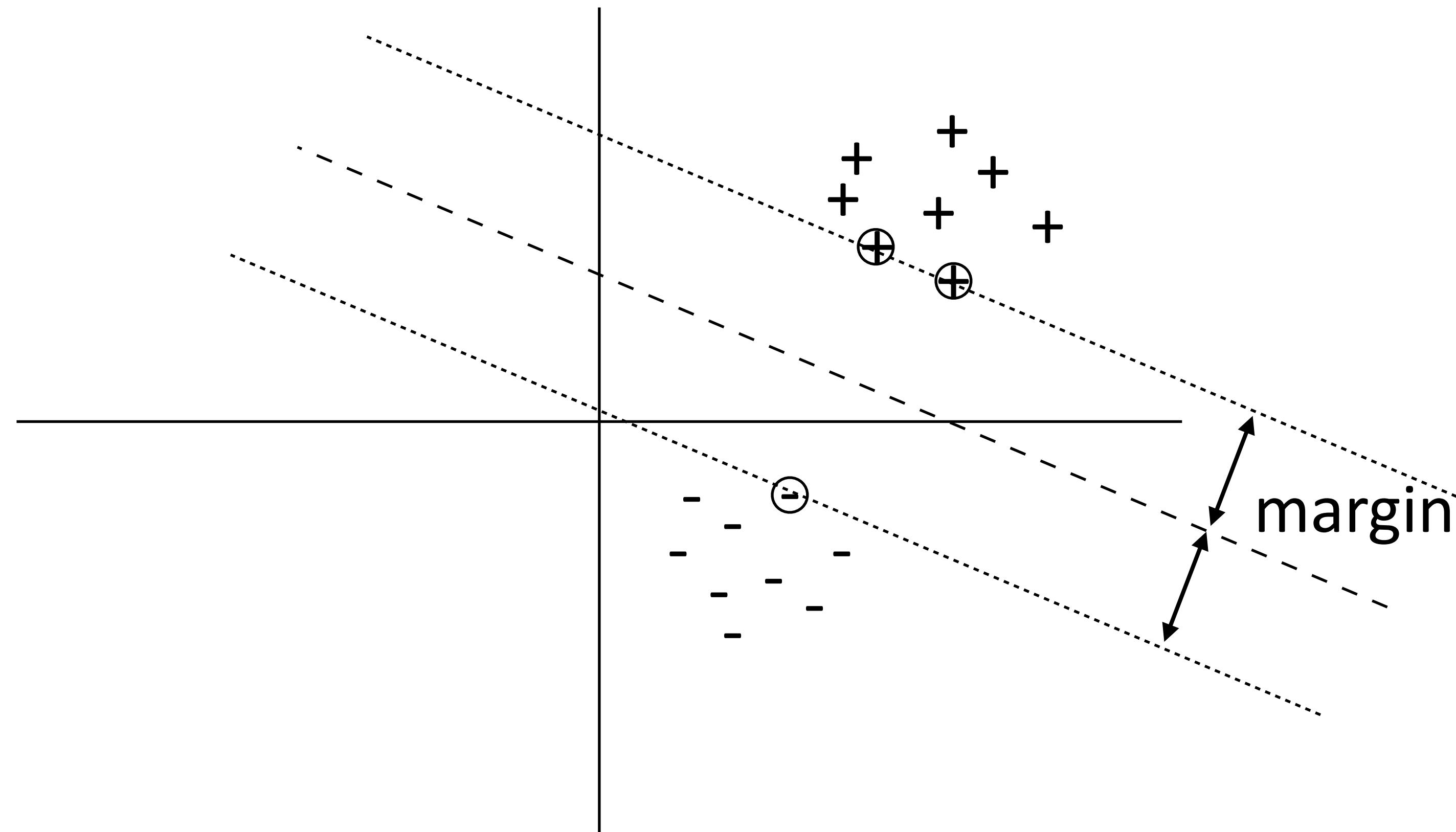
- ▶ Many separating hyperplanes — is there a best one?



# Support Vector Machines

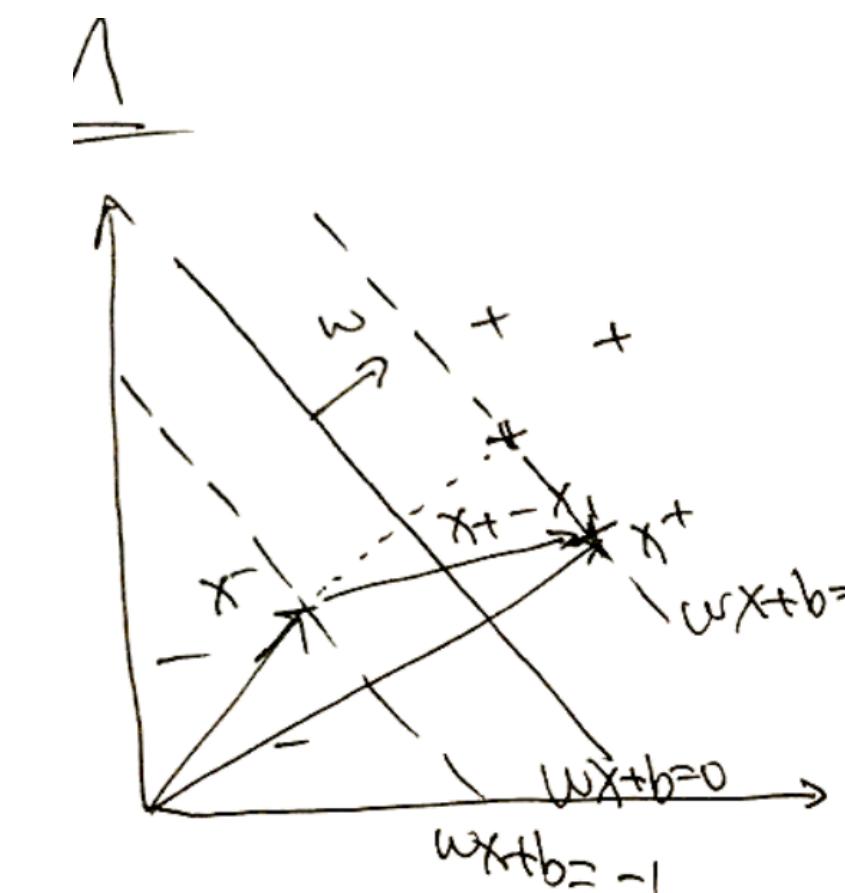
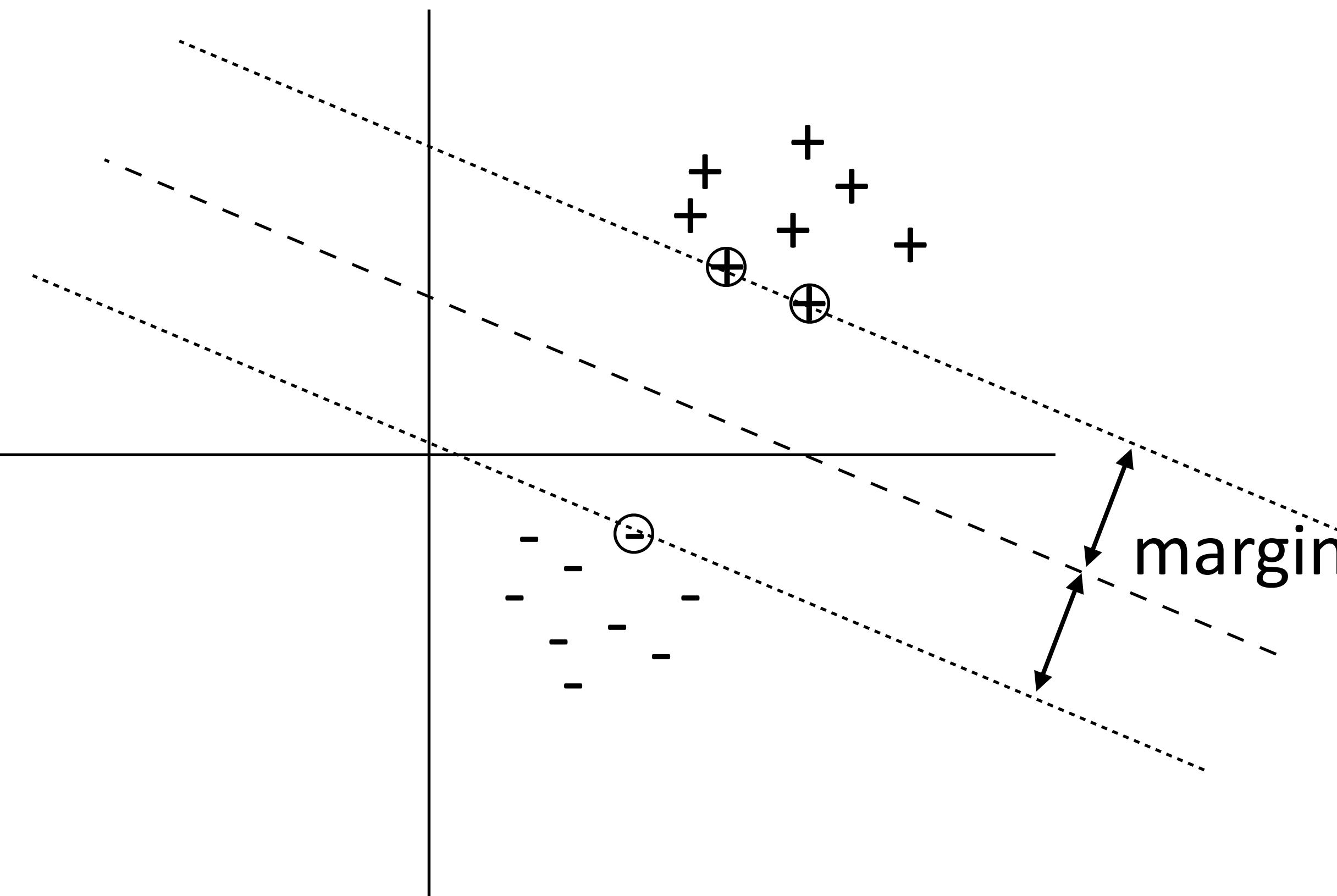
---

- ▶ Many separating hyperplanes — is there a best one?



# Support Vector Machines

- ▶ Many separating hyperplanes — is there a best one?



hyperplane will lie exactly halfway between the nearest positive point and nearest negative point.

$$\text{Margin WIDTH} = (x_+ - x_-) \cdot \frac{w}{\|w\|} = \frac{2}{\|w\|}$$

$$wx_+ + b = 1$$

$$wx_- + b = -1$$

$$\max \frac{2}{\|w\|} \sim \max \frac{1}{\|w\|} \sim \min \|w\| \sim \min \frac{1}{\|w\|}$$

# Support Vector Machines (extracurricular)

- ▶ Constraint formulation: find  $w$  via following quadratic program:

$$\text{Minimize } \|w\|_2^2$$

$$\text{s.t. } \forall j \quad w^\top x_j \geq 1 \text{ if } y_j = 1$$

$$w^\top x_j \leq -1 \text{ if } y_j = 0$$

minimizing norm with  
fixed margin  $\Leftrightarrow$   
maximizing margin

As a single constraint:

$$\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1$$

- ▶ Generally no solution (data is generally non-separable) — need slack!

# N-Slack SVMs (extracurricular)

$$\begin{aligned} \text{Minimize} \quad & \lambda \|w\|_2^2 + \sum_{j=1}^m \xi_j \\ \text{s.t.} \quad & \forall j \quad (2y_j - 1)(w^\top x_j) \geq 1 - \xi_j \quad \forall j \quad \xi_j \geq 0 \end{aligned}$$

- ▶ The  $\xi_j$  are a “fudge factor” to make all constraints satisfied
- ▶ Take the gradient of the objective:

$$\frac{\partial}{\partial w_i} \xi_j = 0 \text{ if } \xi_j = 0 \quad \frac{\partial}{\partial w_i} \xi_j = (2y_j - 1)x_{ji} \text{ if } \xi_j > 0$$
$$= x_{ji} \text{ if } y_j = 1, \quad -x_{ji} \text{ if } y_j = 0$$

- ▶ Looks like the perceptron! But updates more frequently

# LR, Perceptron, SVM (extracurricular)

## ► Gradients on Positive Examples

Logistic regression

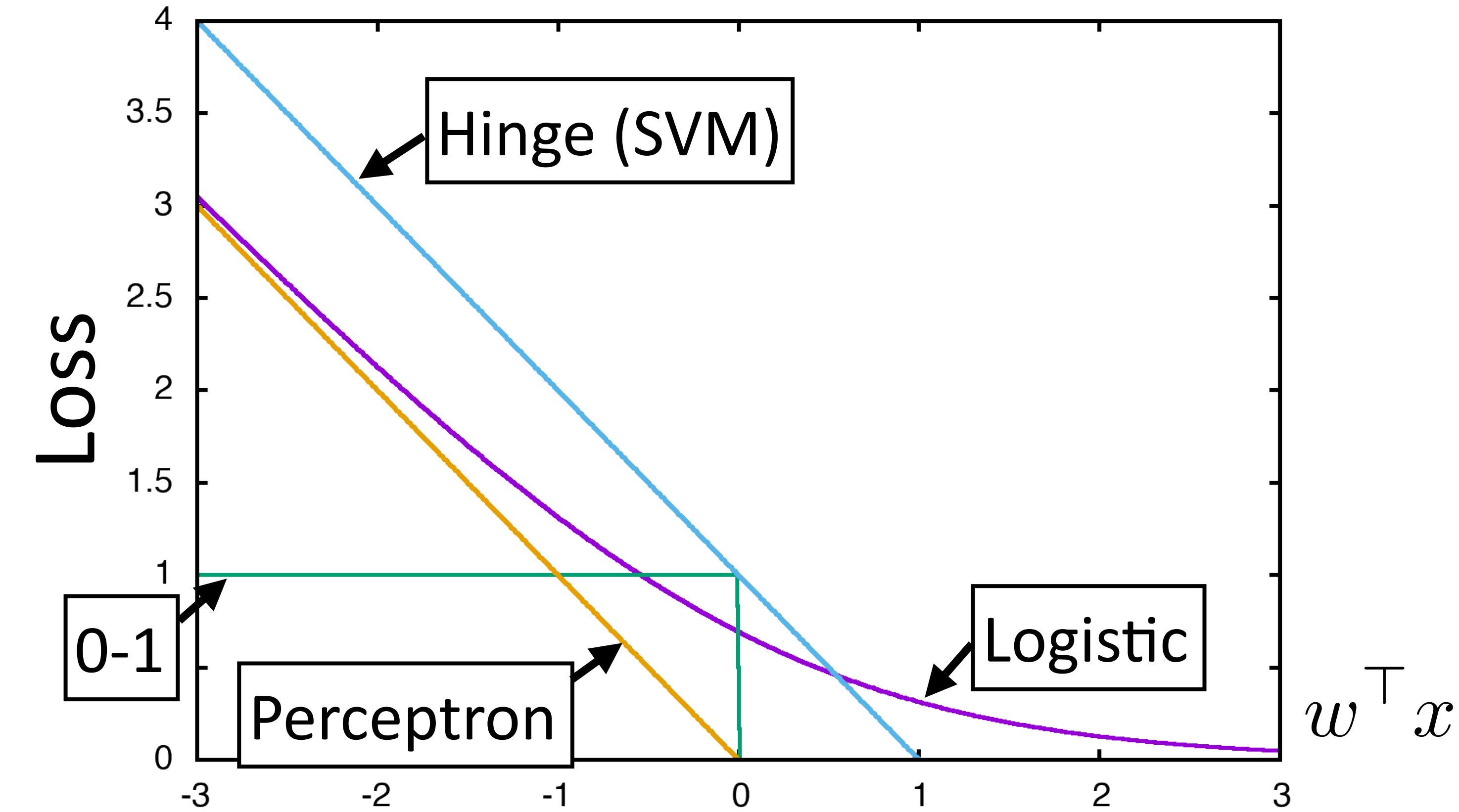
$$x(1 - \text{logistic}(w^\top x))$$

Perceptron

$$x \text{ if } w^\top x < 0, \text{ else } 0$$

SVM (ignoring regularizer)

$$x \text{ if } w^\top x < 1, \text{ else } 0$$



\*gradients are for maximizing things, which is why they are flipped

# Sentiment Analysis

---

*this movie was great! would watch again* + +

*the movie was gross and overwrought, but I liked it* + +

*this movie was not really very enjoyable* — —

- ▶ Bag-of-words doesn't seem sufficient (discourse structure, negation)
- ▶ There are some ways around this: extract bigram feature for “not X” for all X following the *not*

# Sentiment Analysis

---

	Features	# of features	frequency or presence?	NB	ME	SVM
(1)	unigrams	16165	freq.	<b>78.7</b>	N/A	72.8
(2)	unigrams	"	pres.	81.0	80.4	<b>82.9</b>
(3)	unigrams+bigrams	32330	pres.	80.6	80.8	<b>82.7</b>
(4)	bigrams	16165	pres.	77.3	<b>77.4</b>	77.1
(5)	unigrams+POS	16695	pres.	81.5	80.4	<b>81.9</b>
(6)	adjectives	2633	pres.	77.0	<b>77.7</b>	75.1
(7)	top 2633 unigrams	2633	pres.	80.3	81.0	<b>81.4</b>
(8)	unigrams+position	22430	pres.	81.0	80.1	<b>81.6</b>

- ▶ Simple feature sets can do pretty well!

# Sentiment Analysis

Method	RT-s	MPQA
MNB-uni	77.9	85.3
MNB-bi	<b>79.0</b>	<b>86.3</b>
SVM-uni	76.2	86.1
SVM-bi	77.7	<u>86.7</u>
NBSVM-uni	<b>78.1</b>	85.3
NBSVM-bi	<u>79.4</u>	<b>86.3</b>
RAE	76.8	85.7
RAE-pretrain	<b>77.7</b>	<b>86.4</b>
Voting-w/Rev.	63.1	81.7
Rule	62.9	81.8
BoF-noDic.	75.7	81.8
BoF-w/Rev.	76.4	84.1
Tree-CRF	77.3	86.1
BoWSVM	—	—

Kim (2014) CNNs

**81.5    89.5**

← Naive Bayes is doing well!

Ng and Jordan (2002) — NB  
can be better for small data

Before neural nets had taken off  
— results weren't that great

Wang and Manning (2012)

# Recap

---

- ▶ Logistic regression:  $P(y = 1|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{(1 + \exp(\sum_{i=1}^n w_i x_i))}$

Decision rule:  $P(y = 1|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$

Gradient (unregularized):  $x(y - P(y = 1|x))$

- ▶ Logistic regression, perceptron, and SVM are closely related
- ▶ All gradient updates: “make it look more like the right thing and less like the wrong thing”

# Optimization — next ...

---

- ▶ Range of techniques from simple gradient descent (works pretty well) to more complex methods (can work better), e.g., Newton's method, Quasi-Newton methods (LBFGS), Adagrad, Adadelta, etc.
- ▶ Most methods boil down to: take a gradient and a step size, apply the gradient update times step size, incorporate estimated curvature information to make the update more effective

# QA Time

---



DO YOU HAVE  
ANY QUESTIONS?