Machine Learning Recap (linear classification)

Wei Xu

(many slides from Greg Durrett)

Trivia Time

Q: max/min of log prob.?

Coursework Plan

- Four programming projects (33%)
  - Implementation-oriented, PyTorch
  - 1.5~2 weeks per assignment
  - fairly substantial implementation effort except P0
- Three written assignments (20%) + in-class midterm exam (15%)
  - Mostly math and theoretical problems related to ML / NLP
- Final project (25%) + in-class presentation of a recent research paper (2%)
- Participation (5%)

Course Requirements

- **Probability** (e.g. conditional probabilities, conditional independence, Bayes Rule)
- **Linear Algebra** (e.g., multiplying vectors and matrices, matrix inversion)
- **Multivariable Calculus** (e.g., calculating gradients of functions with several variables)
- **Programming / Python experience** (medium-to-large scale project, **debug PyTorch** codes when there are no error messages)
- Prior exposure to machine learning

There will be a lot of math and programming!

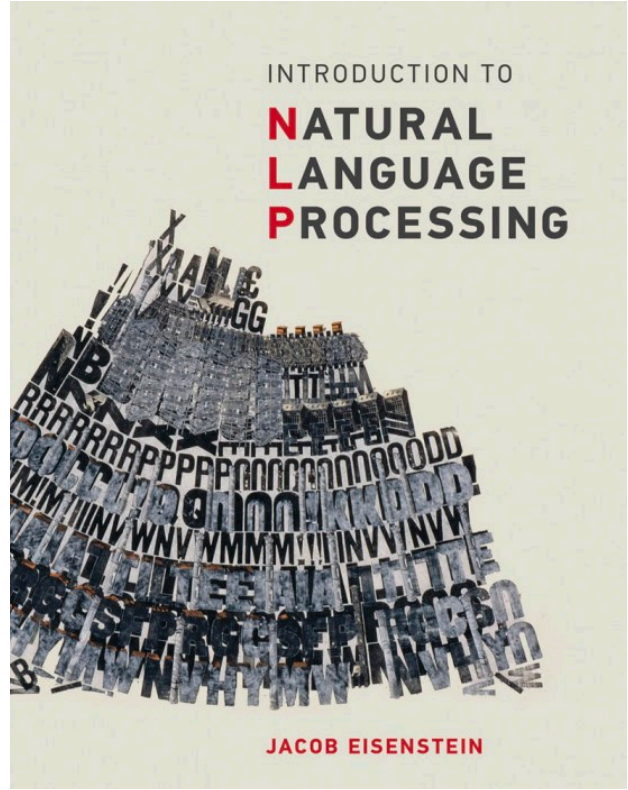
Background Test

- Problem Set 0 (math background) is released, **due Thursday Jan 9**.
- Project 0 (programming - logistic regression) is also released, due Friday Jan 17.
- Take **CS 4641/7641 Machine Learning** and (Math 2550 or Math 2551 or Math 2561 or Math 2401 or Math 24X1 or 2X51) **before (not in the same semester)** this class.
- If you want to understand the lectures better and complete homework with more ease, taking also CS 4644/7643 Deep Learning before this class.

This and next Lecture

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

Readings



Chapter 2 & 4

(+ J&M ch 5)

Chapter 2

Linear text classification

We begin with the problem of **text classification**: given a text document, assign it a discrete label $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of possible labels. Text classification has many applications, from spam filtering to the analysis of electronic health records. This chapter describes some of the most well known and effective algorithms for text classification, from a mathematical perspective that should help you understand what they do and why they work. Text classification is also a building block in more elaborate natural language processing tasks. For readers without a background in machine learning or statistics, the material in this chapter will take more time to digest than most of the subsequent chapters. But this investment will pay off as the mathematical principles behind these basic classification algorithms reappear in other contexts throughout the book.

2.1 The bag of words

To perform text classification, the first question is how to represent each document, or instance. A common approach is to use a column vector of word counts, e.g., $\boldsymbol{x} = [0, 1, 1, 0, 0, 2, 0, 1, 13, 0 \dots]^\top$, where x_j is the count of word j. The length of $\boldsymbol{x}$ is $V \triangleq |\mathcal{V}|$, where $\mathcal{V}$ is the set of possible words in the vocabulary. In linear classification, the classification decision is based on a weighted sum of individual feature counts, such as word counts.

The object $\boldsymbol{x}$ is a vector, but it is often called a **bag of words**, because it includes only information about the count of each word, and not the order in which the words appear. With the bag of words representation, we are ignoring grammar, sentence boundaries, paragraphs — everything but the words. Yet the bag of words model is surprisingly effective for text classification. If you see the word *whale* in a document, is it fiction or nonfiction? What if you see the word *molybdenum*? For many labeling problems, individual words can be strong predictors.

To predict a label from a bag-of-words, we can assign a score to each word in the vocabulary, measuring the compatibility with the label. For example, for the label FICTION, we might assign a positive score to the word *whale*, and a negative score to the word *molybdenum*. These scores are called **weights**, and they are arranged in a column vector $\boldsymbol{\theta}$.

Suppose that you want a multiclass classifier, where $K \triangleq |\mathcal{Y}| > 2$. For example, you might want to classify news stories about sports, celebrities, music, and business. The goal is to predict a label $\hat{y}$, given the bag of words $\boldsymbol{x}$, using the weights $\boldsymbol{\theta}$. For each label $y \in \mathcal{Y}$, we compute a score $\Psi(\boldsymbol{x}, y)$, which is a scalar measure of the compatibility between the bag-of-words $\boldsymbol{x}$ and the label y. In a linear bag-of-words classifier, this score is the vector inner product between the weights $\boldsymbol{\theta}$ and the output of a **feature function** $\boldsymbol{f}(\boldsymbol{x}, y)$,

$$\Psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_j \theta_j f_j(\boldsymbol{x}, y). \qquad [2.1]$$

As the notation suggests, $\boldsymbol{f}$ is a function of two arguments, the word counts $\boldsymbol{x}$ and the label y, and it returns a vector output. For example, given arguments $\boldsymbol{x}$ and y, element j of this feature vector might be,

$$f_j(\boldsymbol{x}, y) = \begin{cases} x_{whale}, & \text{if } y = \text{FICTION} \\ 0, & \text{otherwise} \end{cases} \qquad [2.2]$$

This function returns the count of the word *whale* if the label is FICTION, and it returns zero otherwise. The index j depends on the position of *whale* in the vocabulary, and of FICTION in the set of possible labels. The corresponding weight θ_j then scores the compatibility of the word *whale* with the label FICTION.[1] A positive score means that this word makes the label more likely.

The output of the feature function can be formalized as a vector:

$$\boldsymbol{f}(\boldsymbol{x}, y = 1) = [\boldsymbol{x}; \underbrace{0; 0; \dots; 0}_{(K-1)\times V}] \qquad [2.3]$$

$$\boldsymbol{f}(\boldsymbol{x}, y = 2) = [\underbrace{0; 0; \dots; 0}_{V}; \boldsymbol{x}; \underbrace{0; 0; \dots; 0}_{(K-2)\times V}] \qquad [2.4]$$

$$\boldsymbol{f}(\boldsymbol{x}, y = K) = [\underbrace{0; 0; \dots; 0}_{(K-1)\times V}; \boldsymbol{x}], \qquad [2.5]$$

where $[\underbrace{0; 0; \dots; 0}_{(K-1)\times V}]$ is a column vector of $(K-1) \times V$ zeros, and the semicolon indicates vertical concatenation. For each of the K possible labels, the feature function returns a

[1] In practice, both $\boldsymbol{f}$ and $\boldsymbol{\theta}$ may be implemented as a dictionary rather than vectors, so that it is not necessary to explicitly identify j. In such an implementation, the tuple (*whale*, FICTION) acts as a key in both dictionaries; the values in $\boldsymbol{f}$ are feature counts, and the values in $\boldsymbol{\theta}$ are weights.

Classification

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 (i.e., model parameters) 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	0	1	1	0	...

- Very large vector space (size of vocabulary), sparse features
- Requires *indexing* the features

What are features?

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

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

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

Tf-idf Weighting

- Tf*idf
  - Tf: term frequency

$$tf = \log_{10}(\text{count}(t,\ d) + 1)$$

word-doc co-occurrences

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	17
solider	2	80	62	89
fool	36	58	1	4
clown	20	15	2	3

  - Idf: inverse document frequency

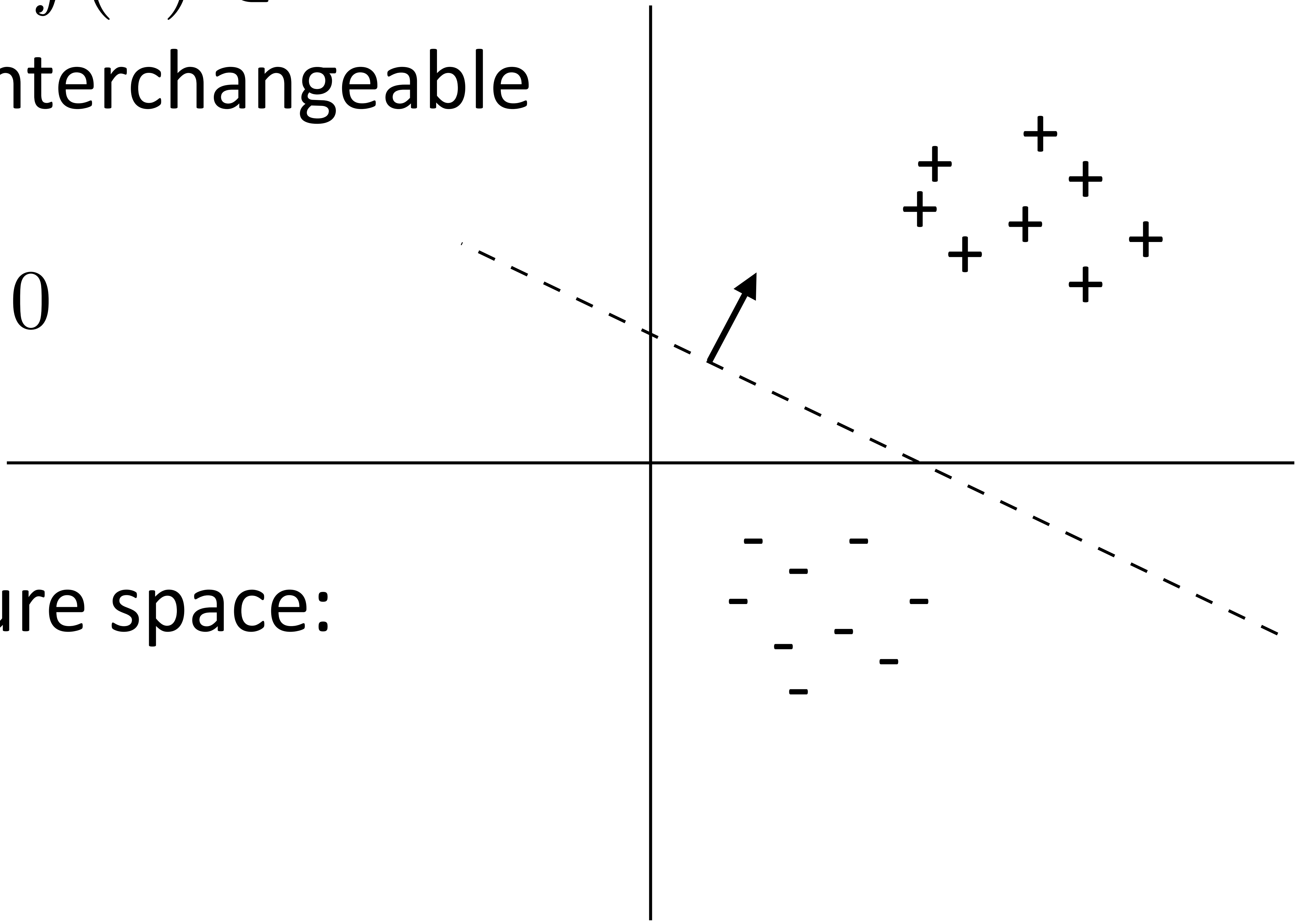
$$idf_i = \log_{10}\left(\frac{N}{df_i}\right)$$

Total number of docs in collection (N)

number of docs that have word i (df_i)

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: $w^\top f(x) + b > 0$
  $w^\top f(x) > 0$



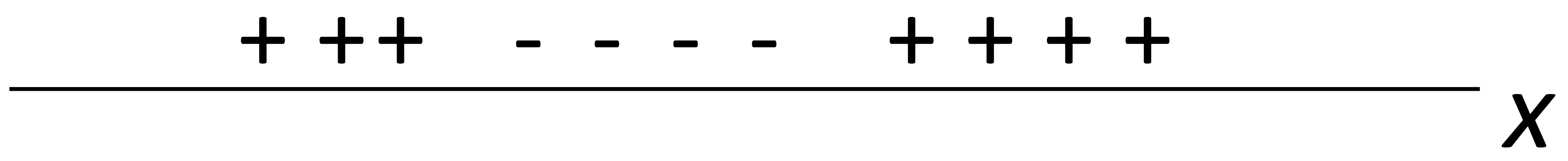
- Can delete bias if we augment feature space:

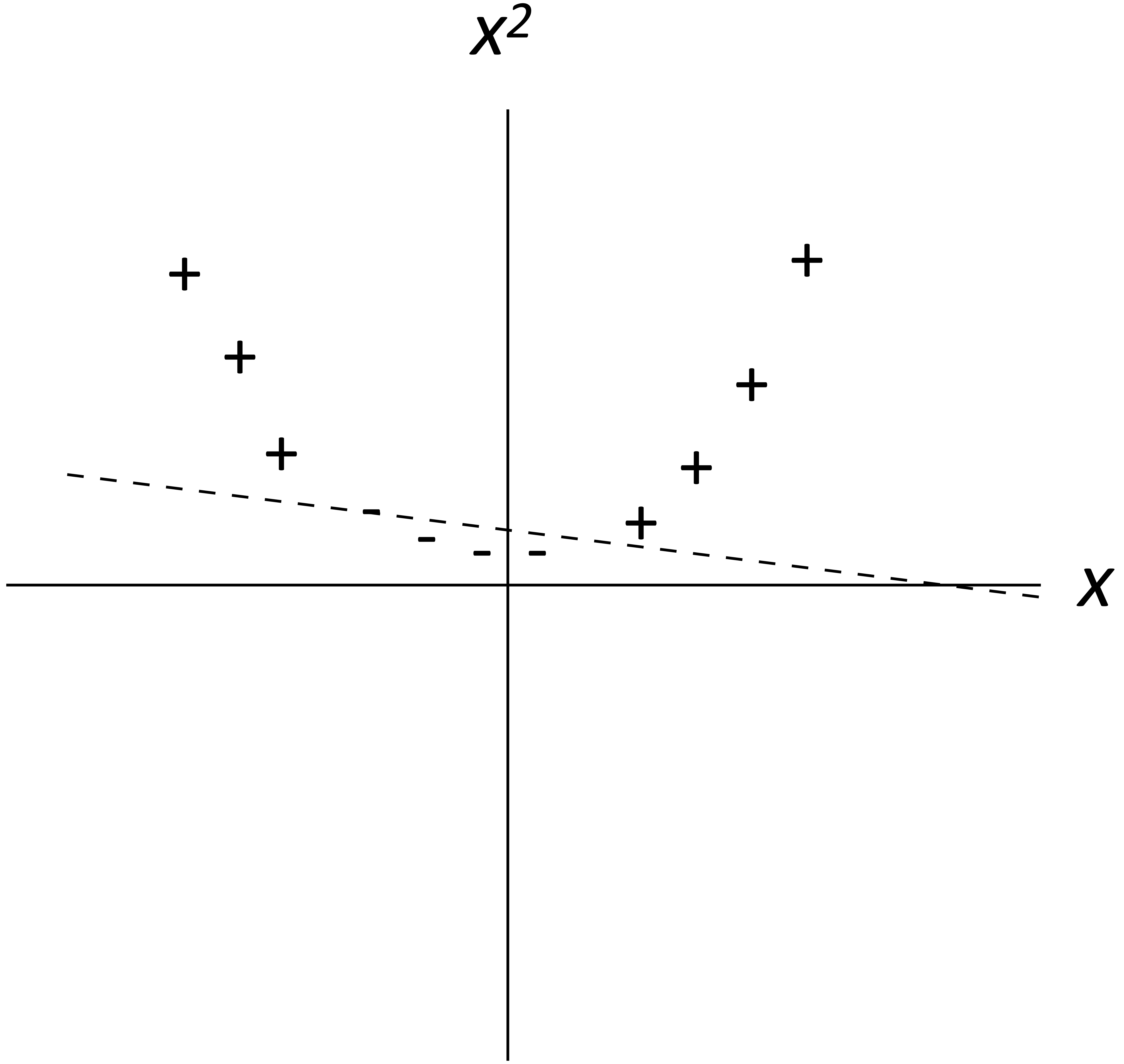
$f(x)$ = [0.5, 1.6, 0.3]

↓

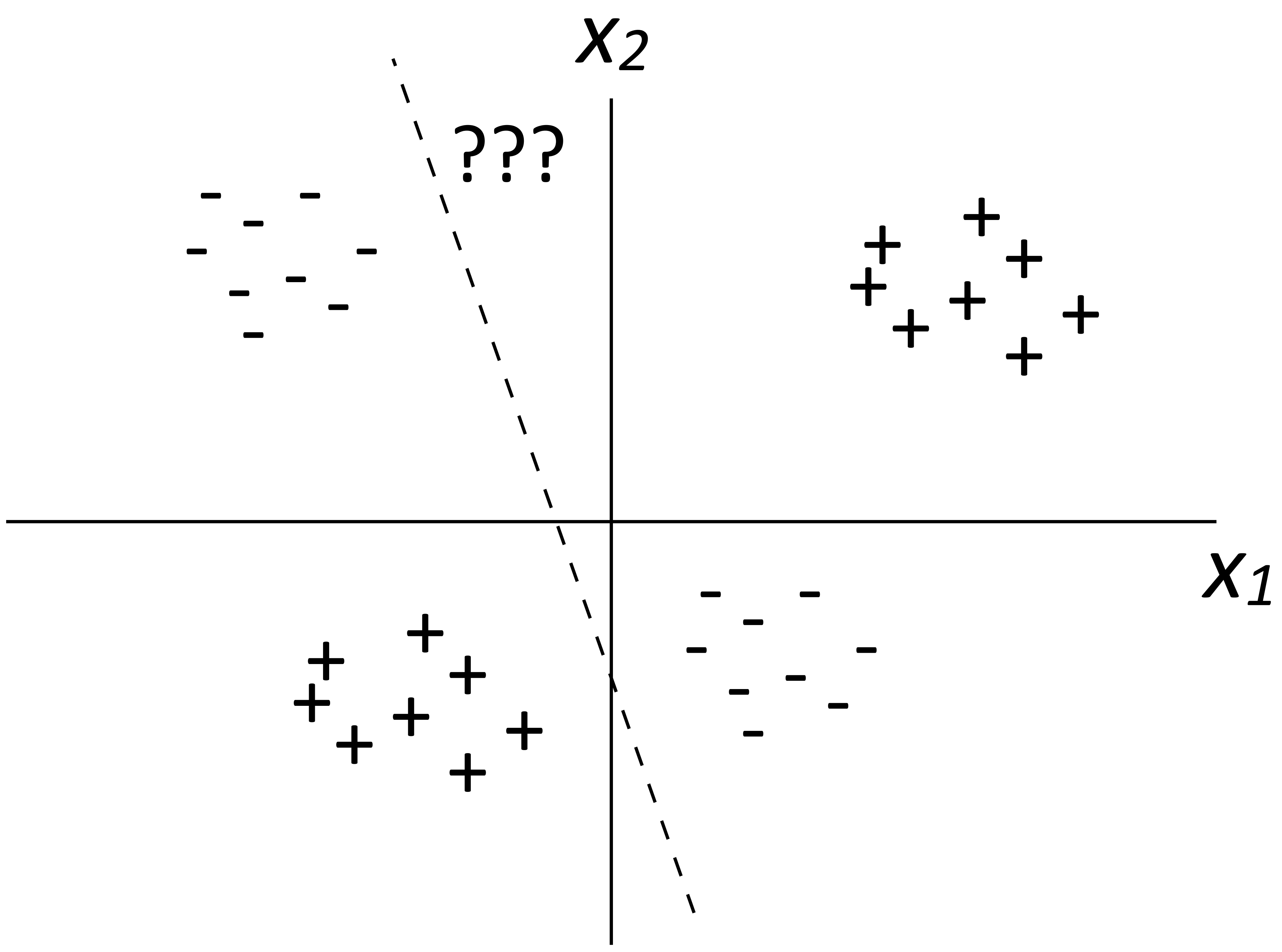
[0.5, 1.6, 0.3, **1**]

Linear functions are powerful!

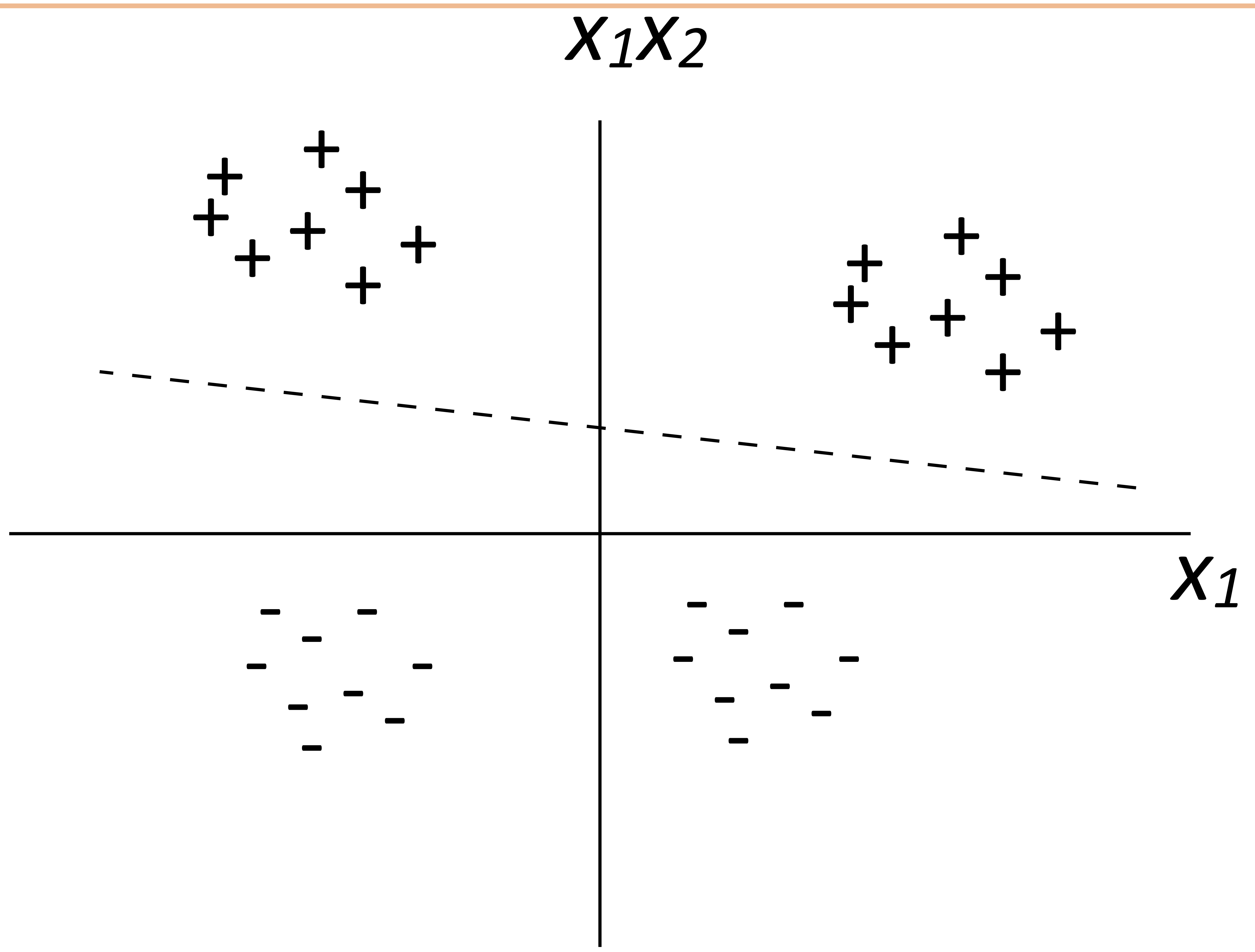




Linear functions are powerful!



$f(x) = [x_1, x_2]$



$f(x) = [x_1, x_2, x_1^2, x_2^2, x_1x_2]$

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

http://ciml.info/dl/v0_99/ciml-v0_99-ch11.pdf

https://www.quora.com/Why-is-kernelized-SVM-much-slower-than-linear-SVM

Naive Bayes

Naive Bayes

- Data point $x = (x_1, ..., x_n)$, 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

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

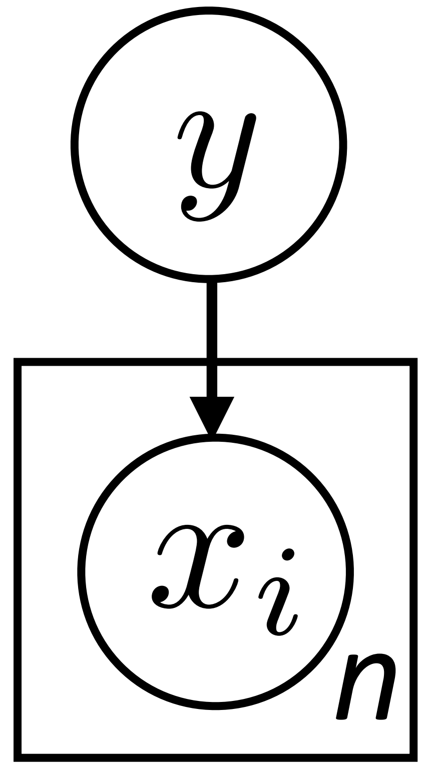
Bayes' Rule

constant: irrelevant for finding the max

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

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

"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?

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:

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

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

Maximum Likelihood Estimation

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

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

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 Estimation

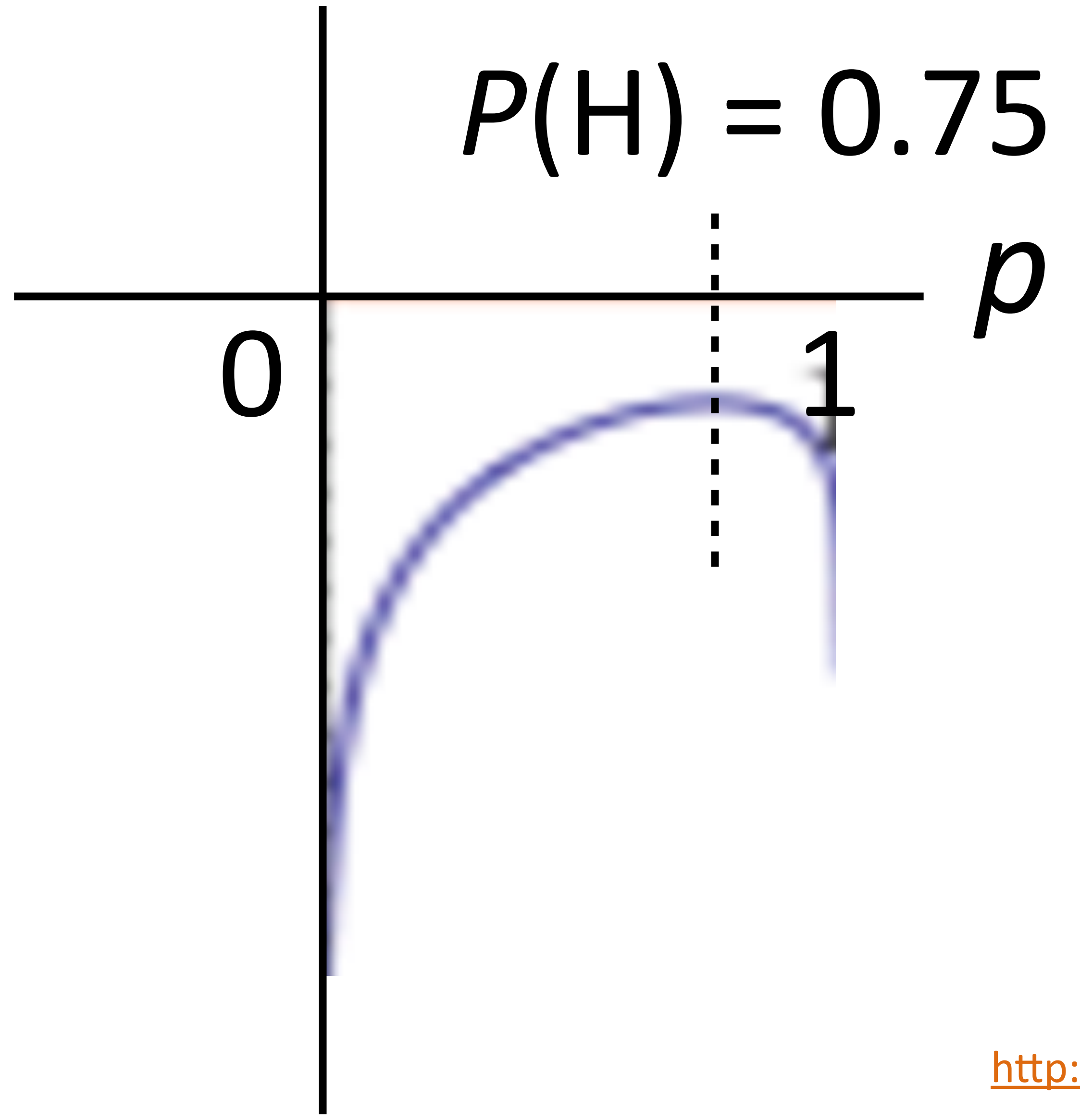
- Imagine a coin flip which is heads with probability *p*

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

- Easier: maximize *log* likelihood

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

log likelihood



http://fooplot.com/

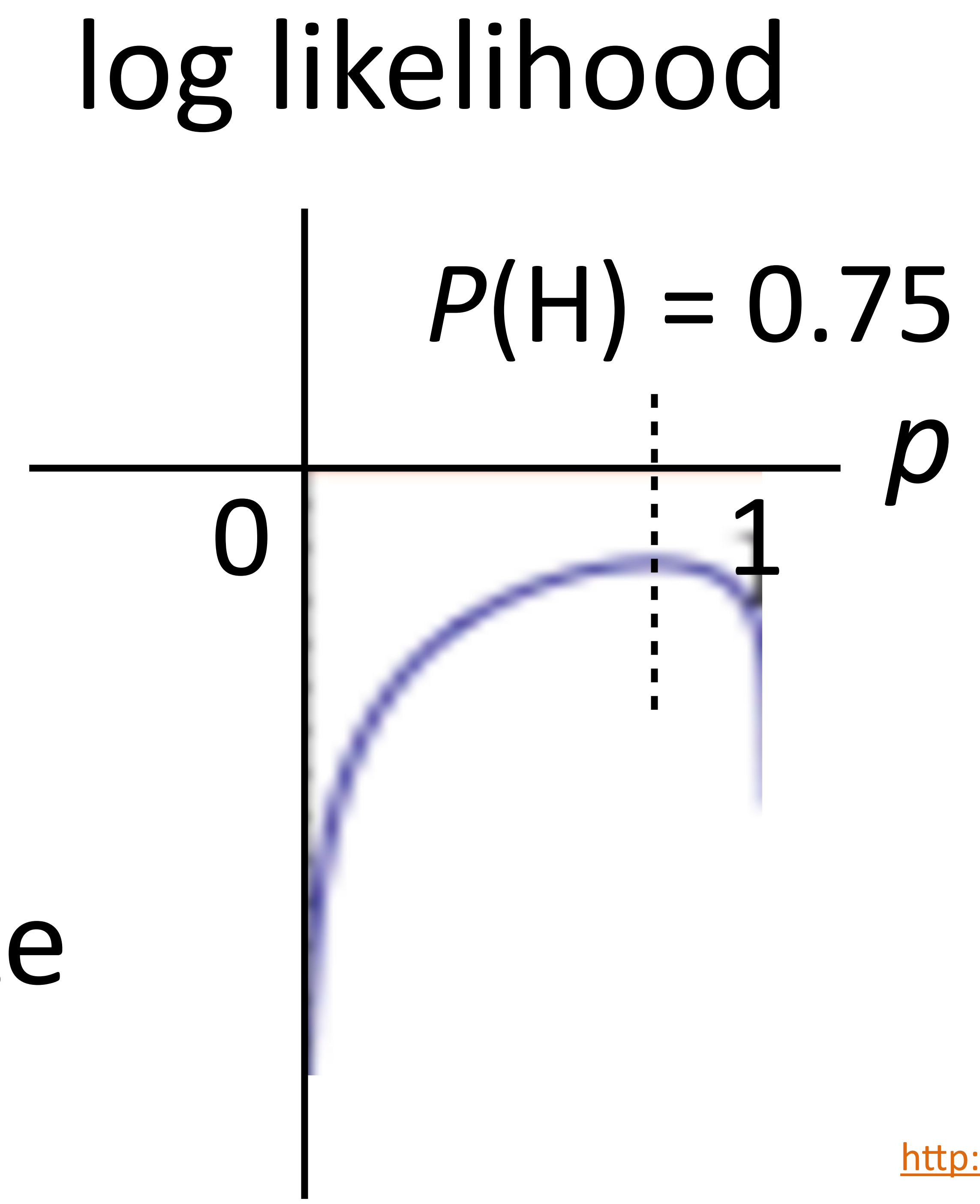
Maximum Likelihood Estimation

- 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

http://fooplot.com/

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($word_i$| +) and P($word_i$| -):
    - number of + (or -) documents $word_i$ is observed, divide by the total number of documents of + (or -) documents

This is for Bernoulli (binary features) document model!

http://socialmedia-class.org/slides_AU2017/Shimodaira_note07.pdf

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	—

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

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

prior

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

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

word likelihood

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

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

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

- Multinominal 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

http://socialmedia-class.org/slides_AU2017/Shimodaira_note07.pdf

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:

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

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:

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

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 would have:

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

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

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 $P(w_t|C)$:

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

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,

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

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 $C=k$:

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

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:

http://socialmedia-class.org/slides_AU2017/Shimodaira_note07.pdf

Zero Probability Problem

- What if we have seen no training document with the word "fantastic" and classified in the topic positive?

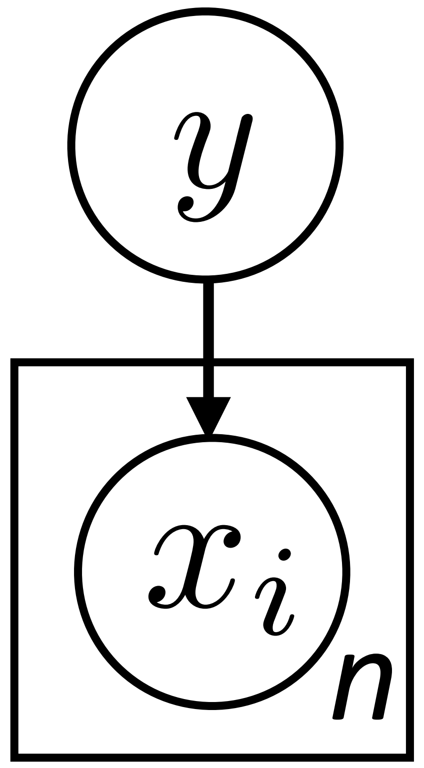
$$P(y|x) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

- Laplace (add-1) Smoothing
  - Word likelihood — P(word$_i$| +) and P(word$_i$| -):
    - frequency of word$_i$ is observed **plus 1**

Naive Bayes: Summary

- Model

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



- Inference

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

- 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 —

$$P(x_{\text{beautiful}}|+) = 0.1 \qquad P(x_{\text{beautiful}}|-) = 0.01$$
$$P(x_{\text{stunning}}|+) = 0.1 \qquad P(x_{\text{stunning}}|-) = 0.01$$
$$P(x_{\text{gorgeous}}|+) = 0.1 \qquad P(x_{\text{gorgeous}}|-) = 0.01$$
$$P(x_{\text{boring}}|+) = 0.01 \qquad P(x_{\text{boring}}|-) = 0.1$$

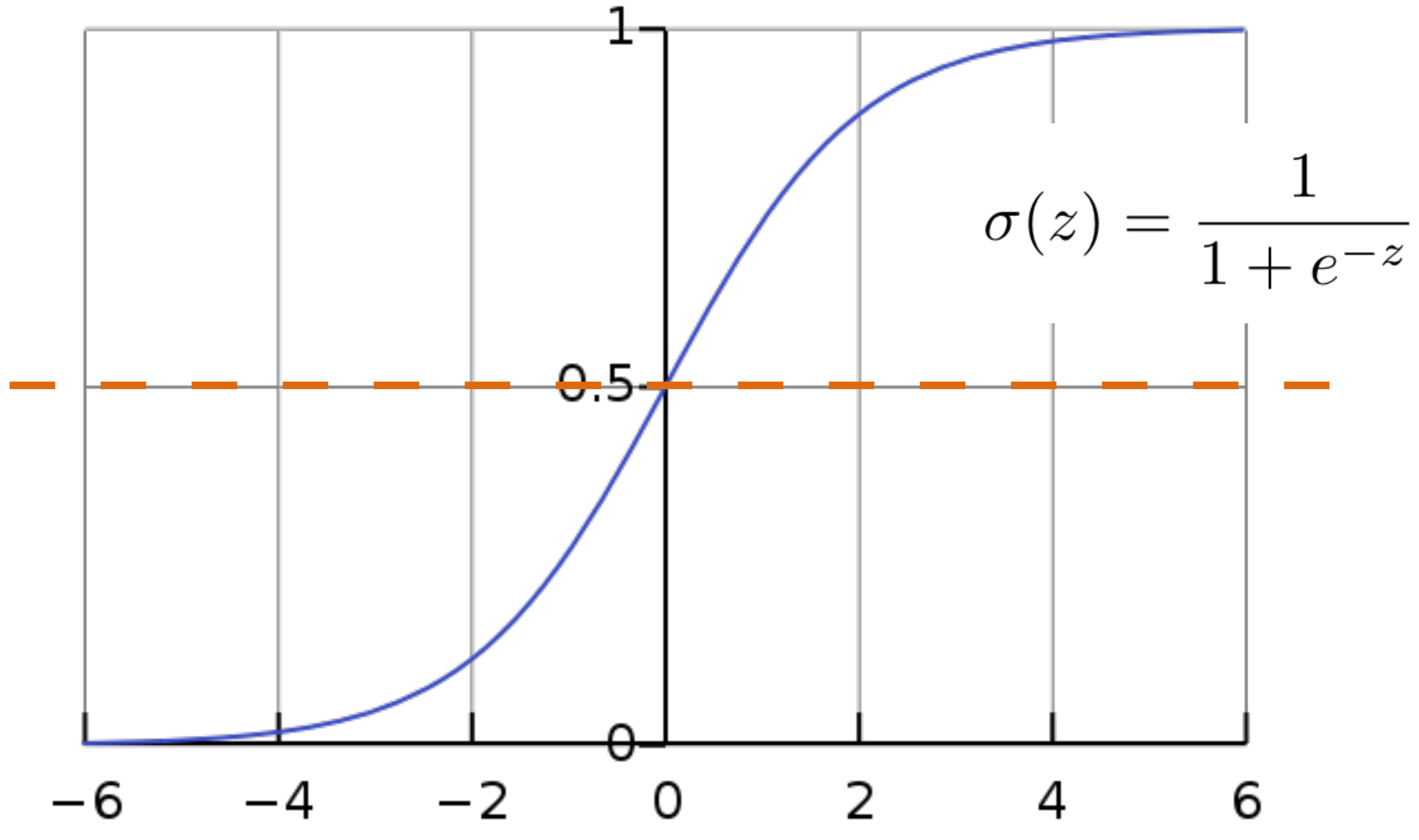
- Correlated features compound: *beautiful* and *gorgeous* are not independent!
- Naive Bayes is naive, but another problem is that it's *generative*: spends capacity modeling P(x,y), when what we care about is P(y|x)
- Discriminative models model P(y|x) directly (SVMs, most neural networks, …)

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)

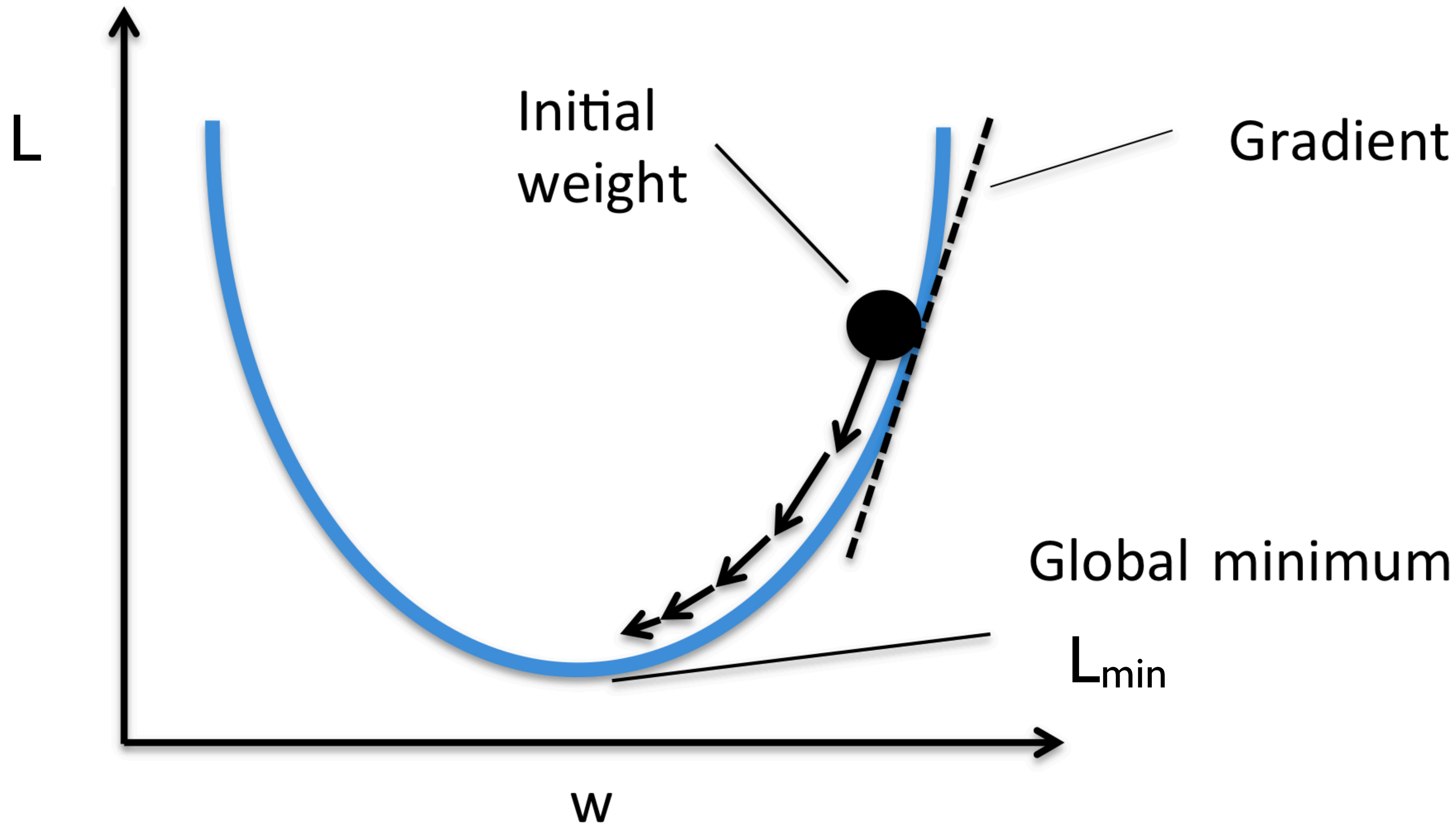
$$\mathcal{L}(x_j, y_j=+) = \log P(y_j=+|x_j)$$

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

sum over features

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)

Logistic Regression

chain rule:

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

maximize!

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

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

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

deriv. of log

$$\frac{\partial \log \mathrm{x}}{\partial \mathrm{x}} = \frac{1}{\mathrm{x}}$$

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

deriv. of exp

$$\frac{\partial e^{\mathrm{x}}}{\partial \mathrm{x}} = e^{\mathrm{x}}$$

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

Logistic Regression

- Recall that $y_j = 1$ for positive instances, $y_j = 0$ for negative instances.
- Gradient of w_i on positive example $= x_{ji}(1 - P(y_j = +|x_j))$

  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

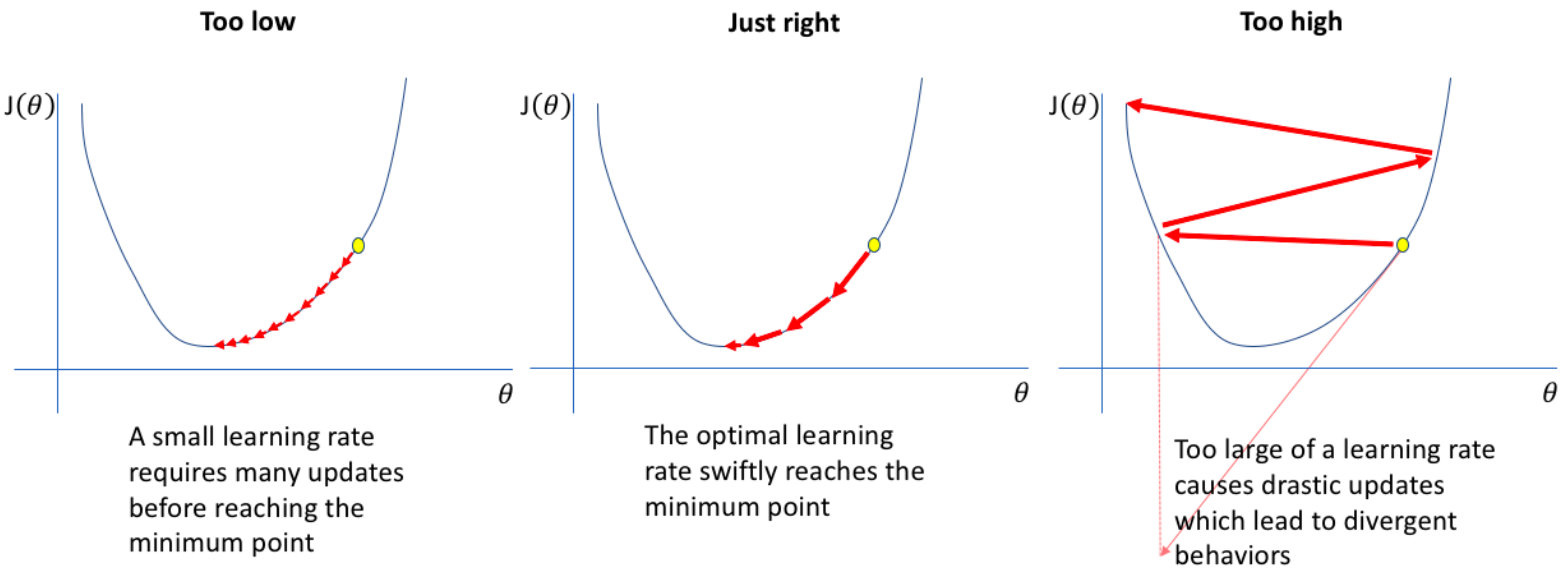
log likelihood of data P(y|x)   data points (*j*)

- 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: $\frac{\partial \mathcal{L}(w)}{\partial w} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n}\right)$

Learning Rate

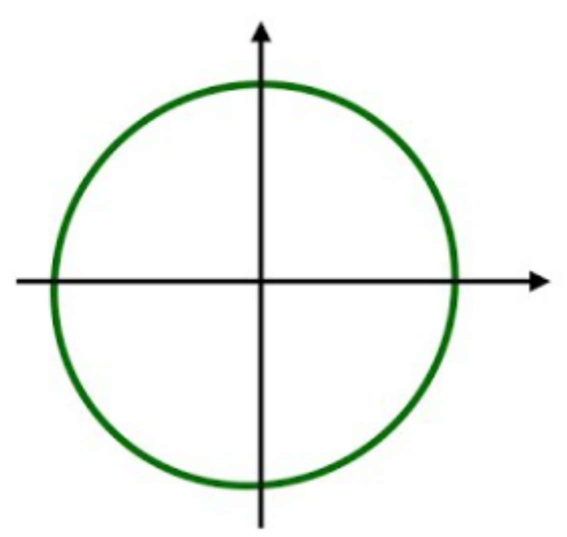


Credit: Jeremy Jordan

Regularization

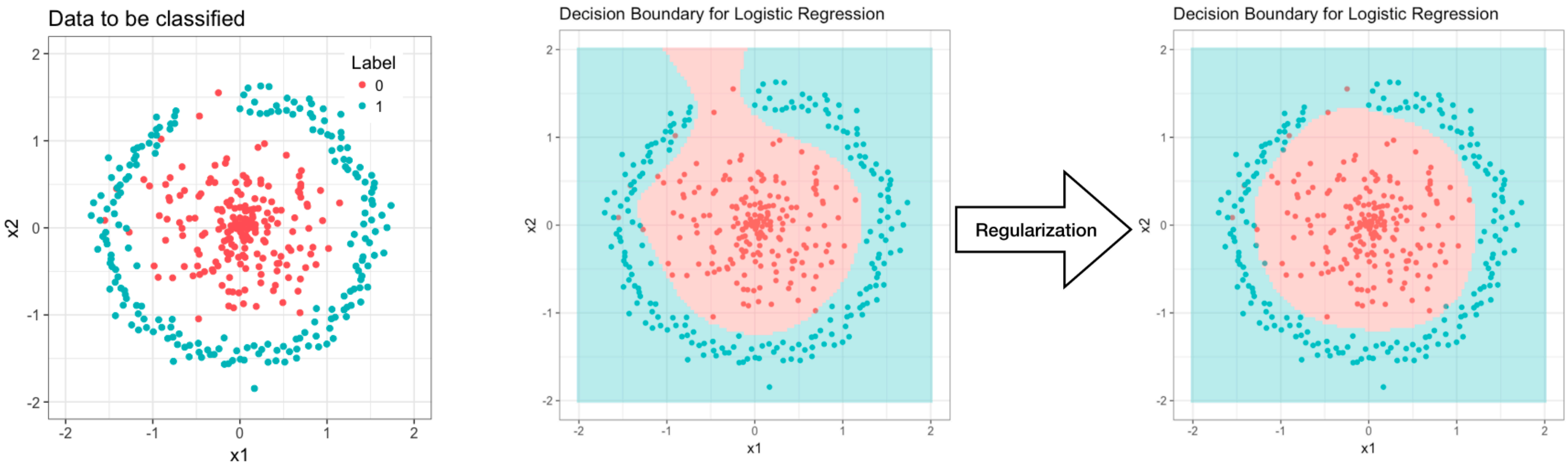
- Regularizing an objective can mean many things, including an L2-norm penalty to the weights:

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



- 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

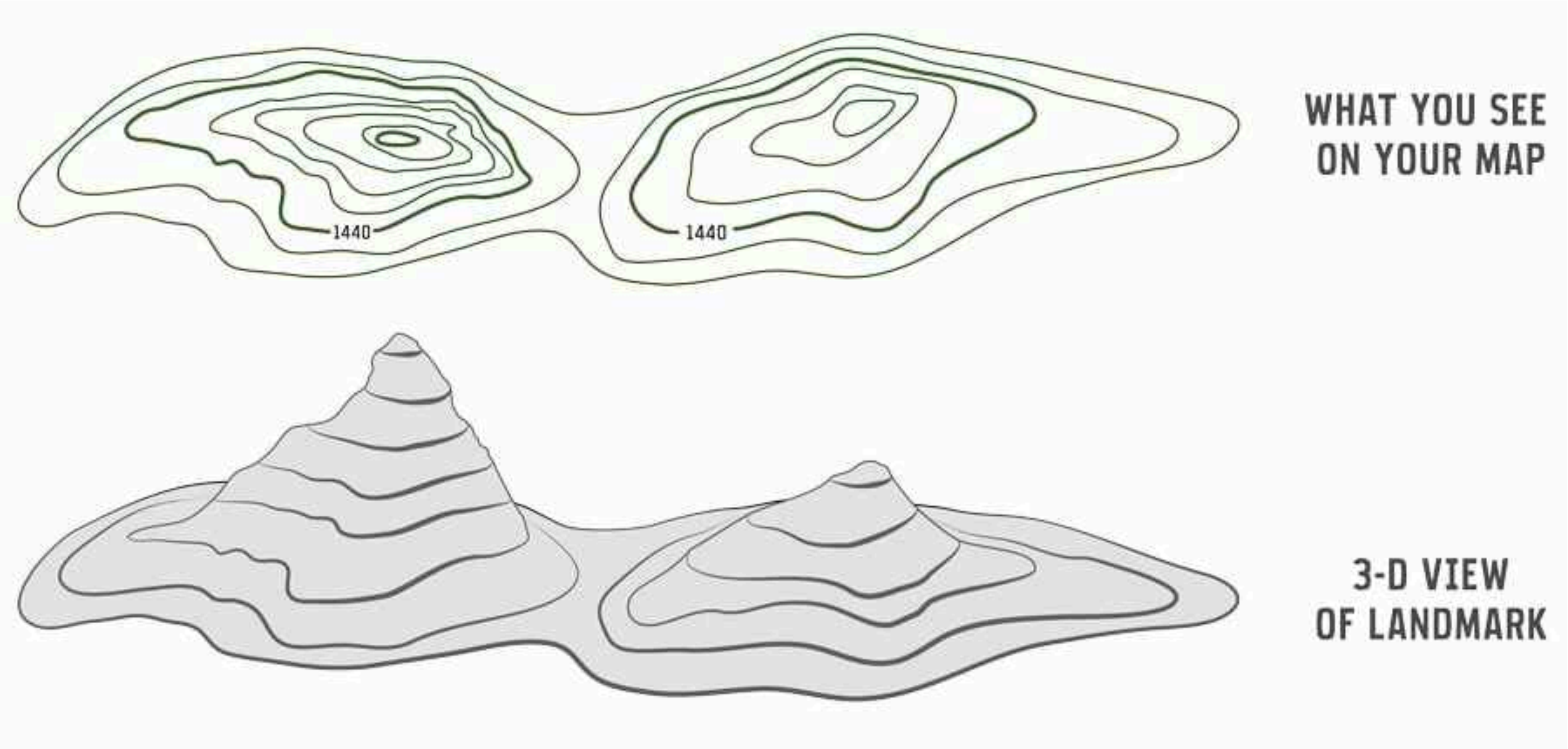
Regularization



$f(x) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2, \dots]$

https://towardsdatascience.com/understanding-regularization-in-machine-learning-5a0369ac73b9

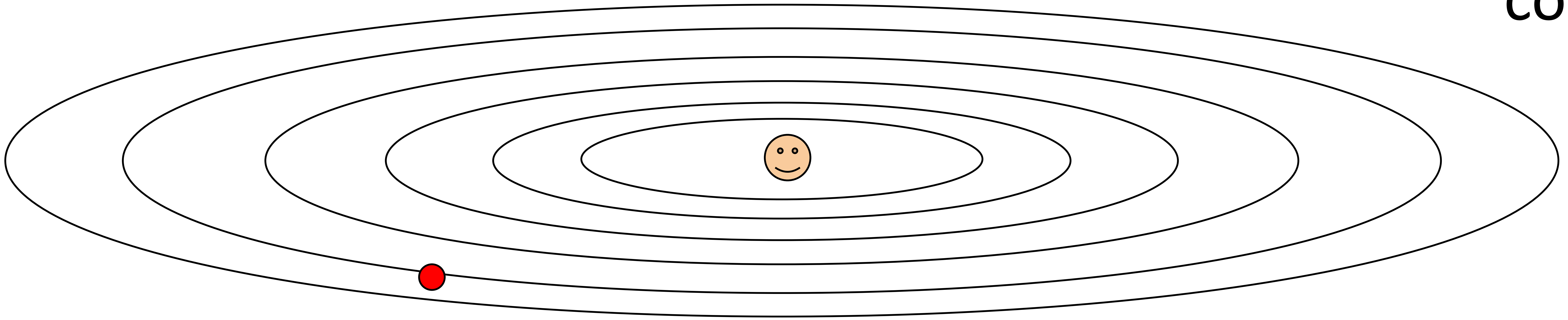
Optimization



- Gradient descent

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

Q: What if loss changes quickly in one direction and slowly in another direction?



Credit: Stanford CS231n

Feature Scaling

Optimization

- Gradient descent
  - Very simple to code up
  - "First-order" technique: only relies on having gradient

$$w \leftarrow w - \alpha g, \quad g = \frac{\partial}{\partial w}\mathcal{L}$$

- Newton's method
  - Second-order technique
  - Optimizes quadratic instantly

$$w \leftarrow w - \left(\frac{\partial^2}{\partial w^2}\mathcal{L}\right)^{-1} g$$

Inverse Hessian: *n* x *n* mat, expensive!

- Quasi-Newton methods: L-BFGS, etc. approximate inverse Hessian

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

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

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

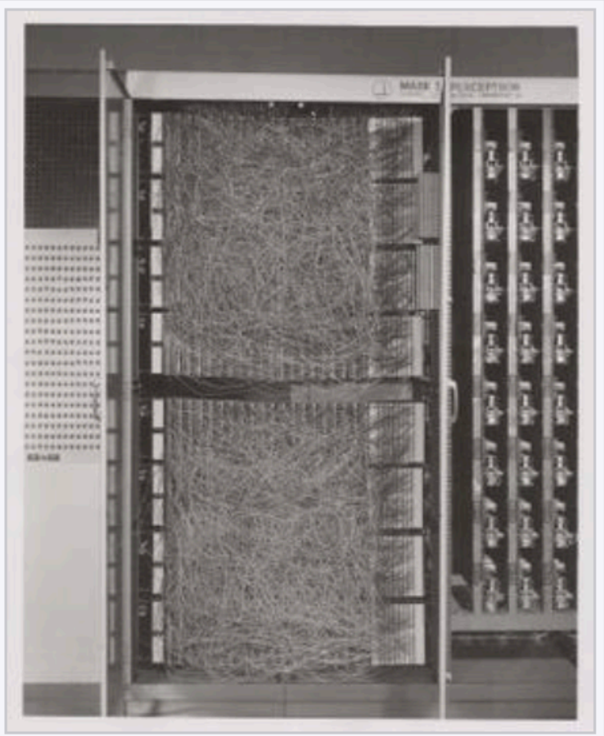
- Learning: gradient ascent on the (regularized) discriminative log-likelihood

Perceptron/SVM

Perceptron

History [edit]

V · T · E



Mark I Perceptron machine, the first implementation of the perceptron algorithm. It was connected to a camera with 20×20 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.[2]:213

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

The perceptron algorithm was invented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt,[3] funded by the United States Office of Naval Research.[4]

The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron". This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.[2]:193

In a 1958 press conference organized by the US Navy, Rosenblatt made statements about the perceptron that caused a heated controversy among the fledgling AI community; based on Rosenblatt's statements, *The New York Times* reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."[4]

Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had greater processing power than perceptrons with one layer (also called a single layer perceptron).

Single layer perceptrons are only capable of learning linearly separable patterns. For a classification task with some step activation function a single node will have a single line dividing the data points forming the patterns. More nodes can create more dividing lines, but those lines must somehow be combined to form more complex classifications. A second layer of perceptrons, or even linear nodes, are sufficient to solve a lot of otherwise non-separable problems.

In 1969 a famous book entitled *Perceptrons* by Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function. It is often believed (incorrectly) that they also conjectured that a similar result would hold for a multi-layer perceptron network. However, this is not true, as both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR function. (See the page on *Perceptrons (book)* for more information.) Nevertheless, the often-miscited Minsky/Papert text caused a significant decline in interest and funding of neural network research. It took ten more years until neural network research experienced a resurgence in the 1980s. This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

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

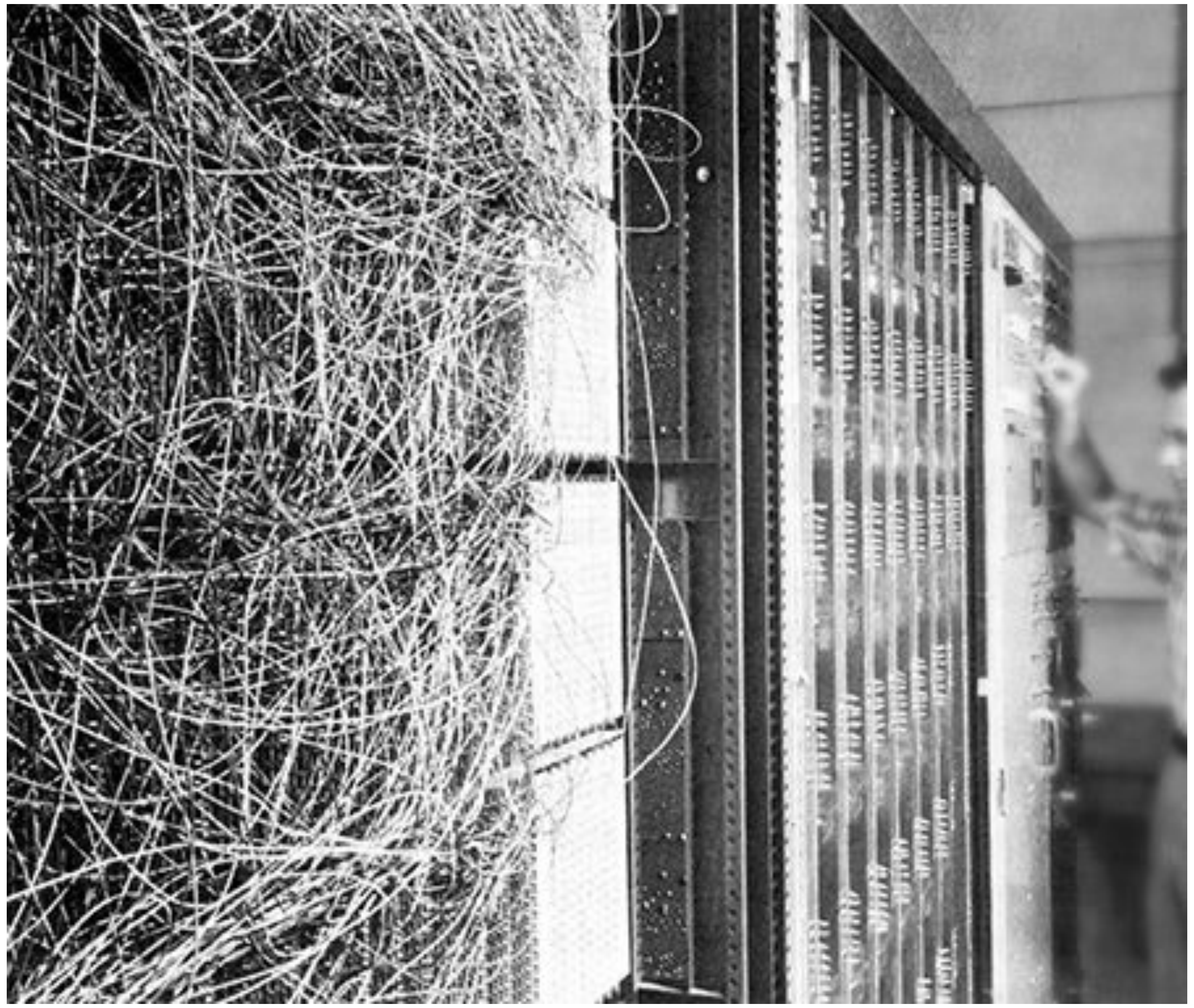
The perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.[7]

Frank Rosenblatt (1928-1971)

PhD 1956 from Cornell

A Bit of History

- The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
- Perceptron (Frank Rosenblatt, 1957)
- Artificial Neuron (McCulloch & Pitts, 1943)



McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} x_i < 0$$

Perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

The IBM Automatic Sequence Controlled Calculator, called Mark I by Harvard University's staff. It was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

https://www.youtube.com/watch?v=SaFQAoYV1Nw
https://www.youtube.com/watch?time_continue=71&v=cNxadbrN_aI&feature=emb_logo

Perceptron - artificial neuron

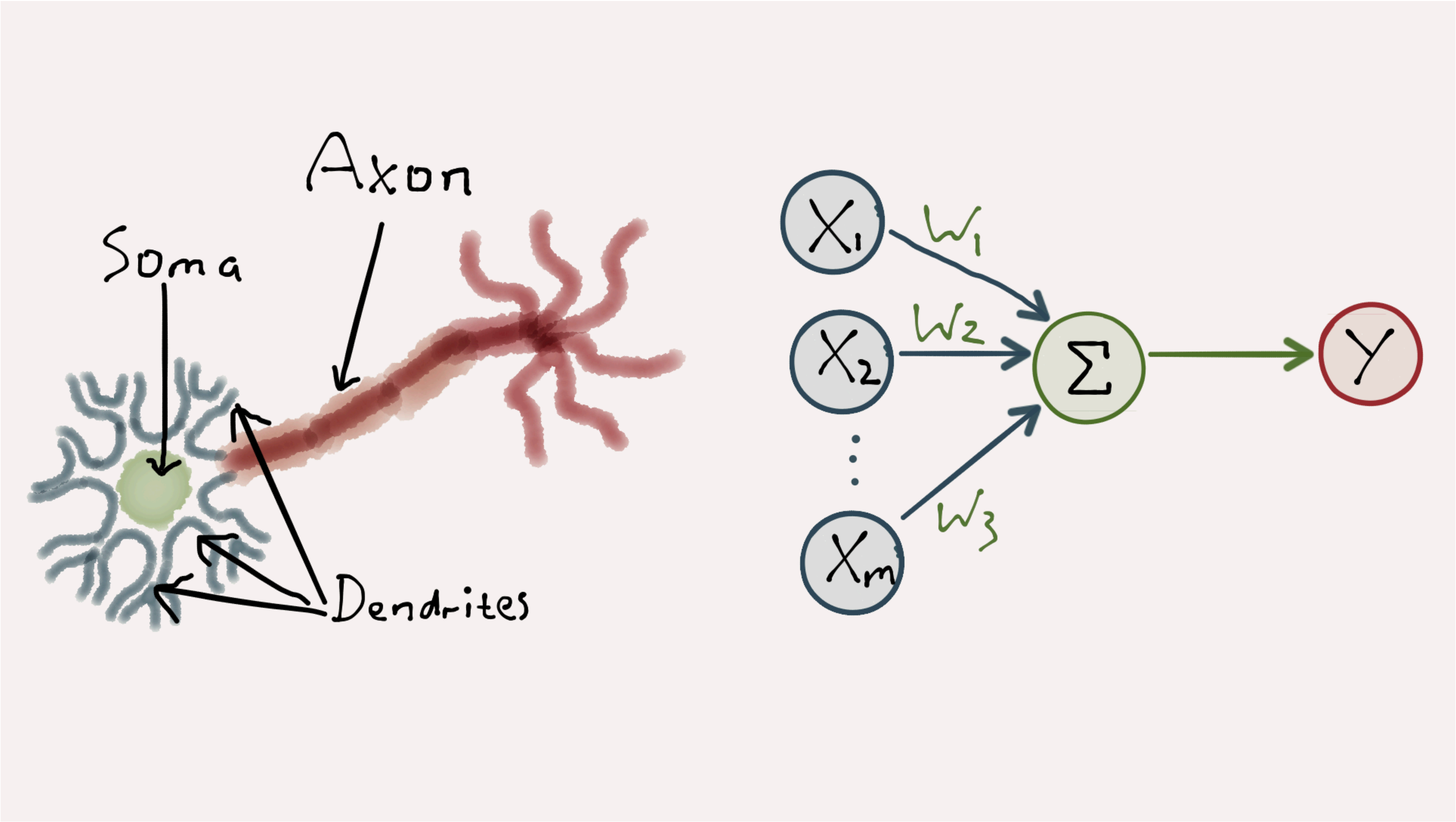


Figure from https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html

Perceptron

- Simple error-driven learning approach similar to logistic regression

- Decision rule: $w^\top x > 0$
  - If incorrect: if positive, $w \leftarrow w + x$

    if negative, $w \leftarrow w - x$

Logistic Regression

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

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

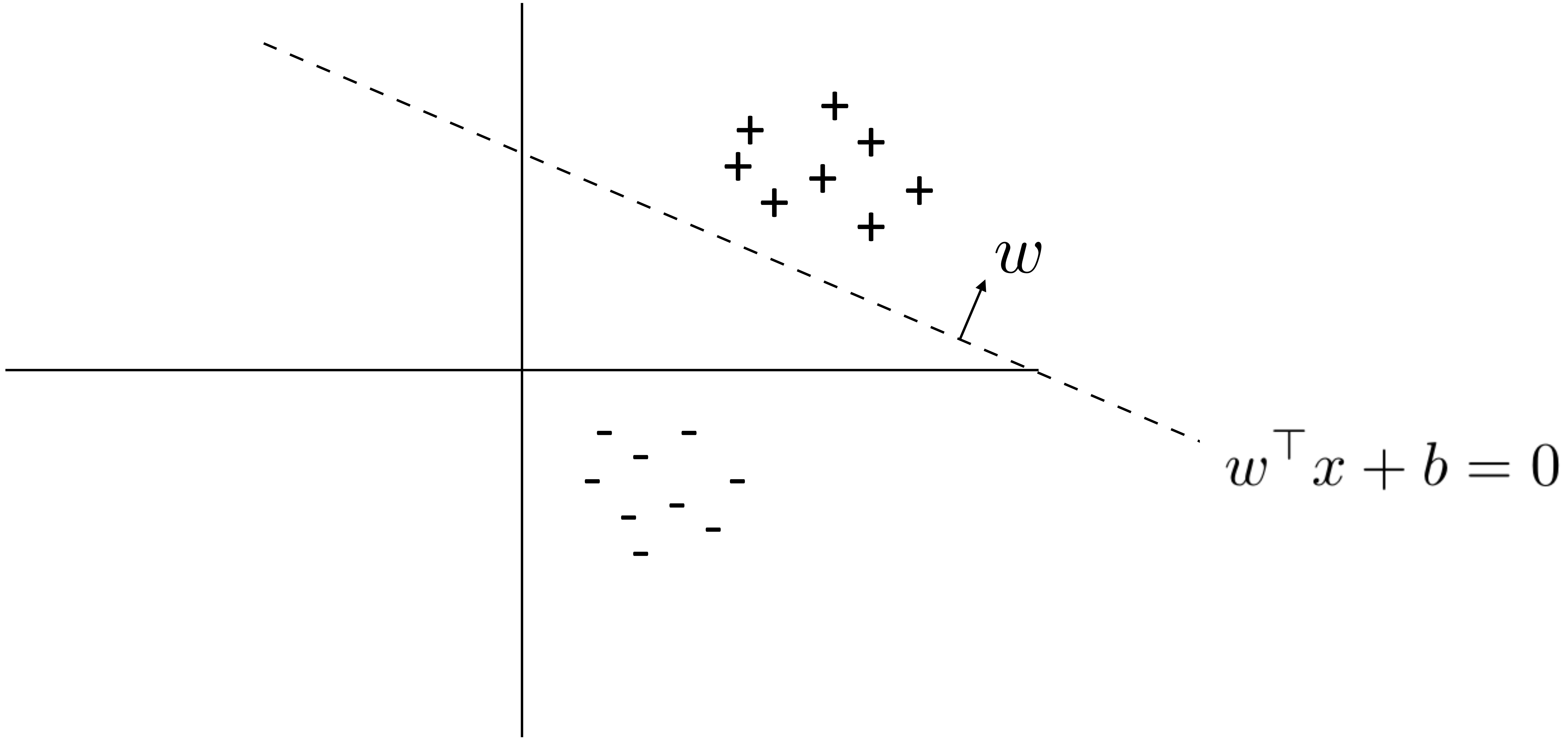
- Algorithm is very similar to logistic regression

- Perceptron guaranteed to eventually separate the data if the data are separable

http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

Perceptron

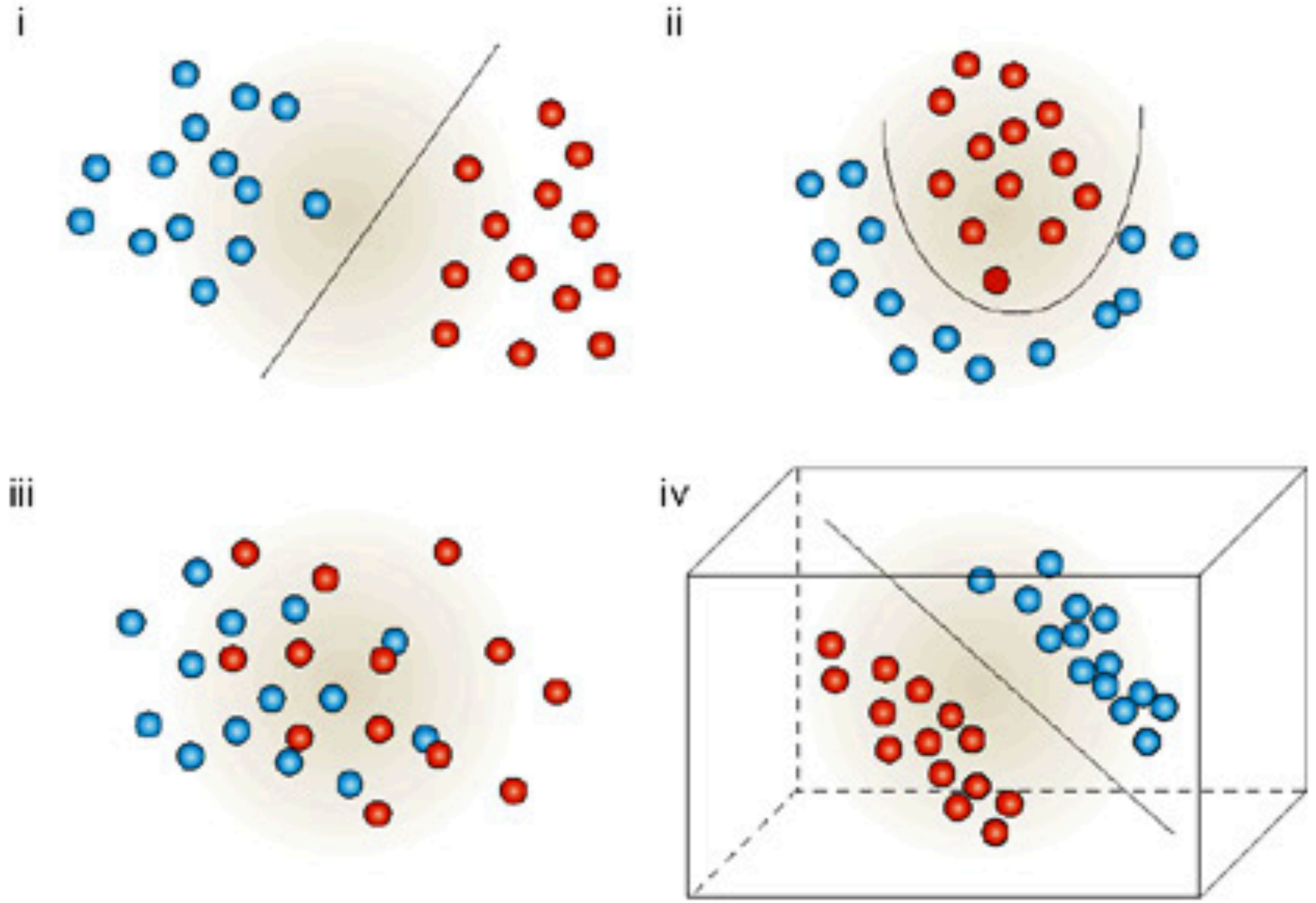
- Separating hyperplane



Two vectors have a zero dot product if and only if they are perpendicular

Linear Separability

- In general, two groups are linearly separable in n-dimensional space, if they can be separated by an (n-1)-dimensional hyperplane.



What does “converge” mean?

- It means that it can make an entire pass through the training data without making any more updates.

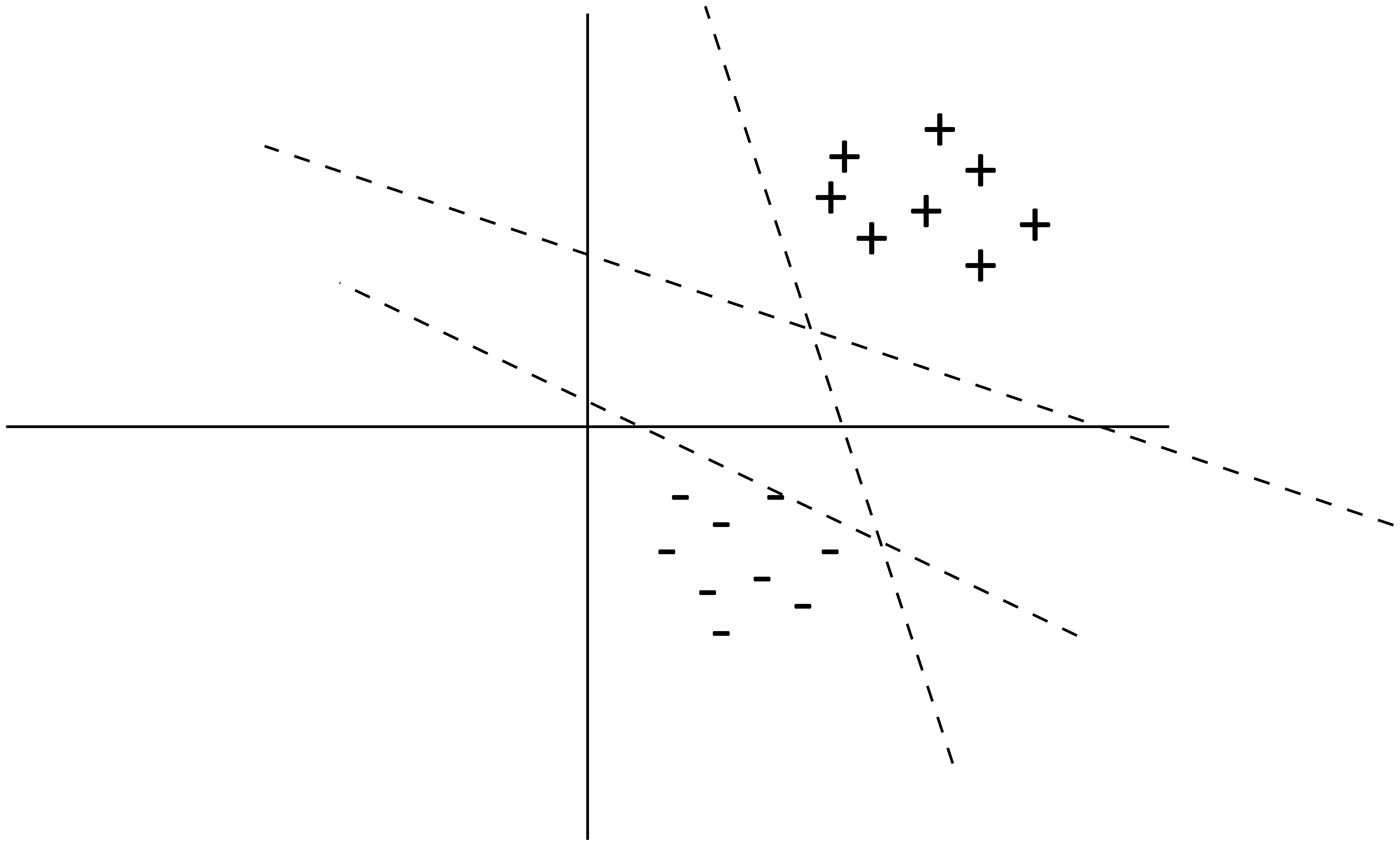
- In other words, Perceptron has correctly classified every training example.

- Geometrically, this means that it was found some hyperplane that correctly segregates the data into positive and negative examples

http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

Support Vector Machines

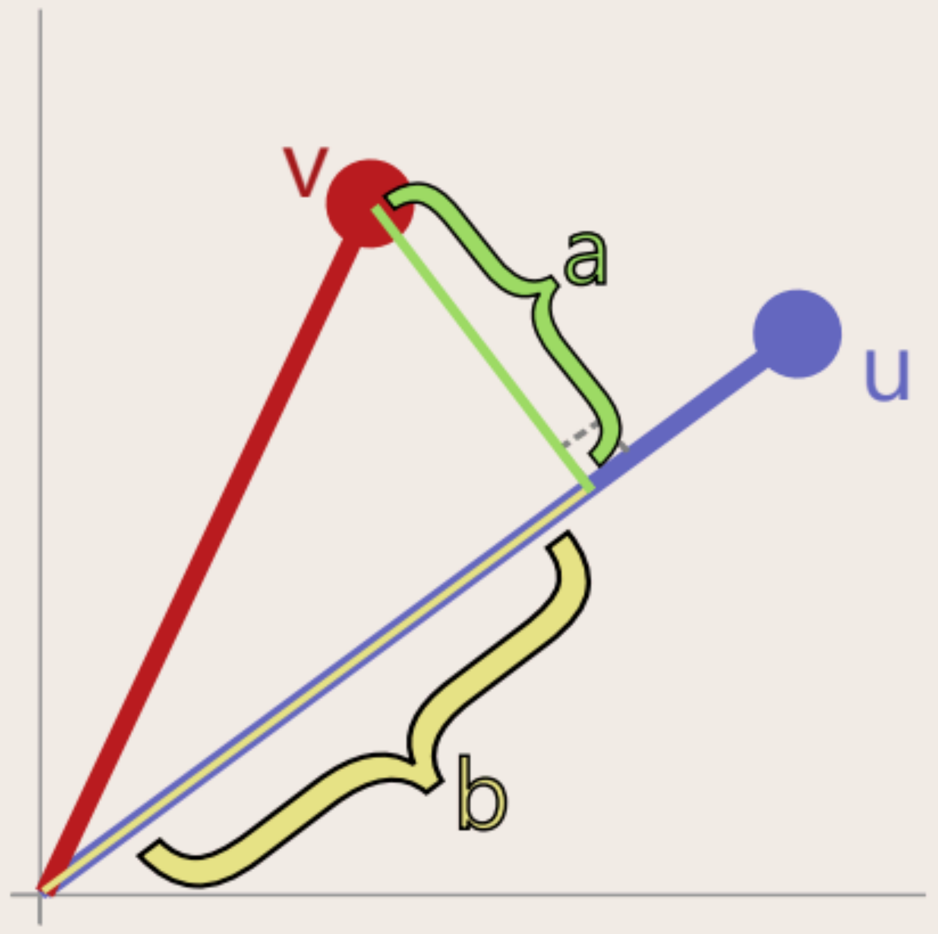
- Many separating hyperplanes — is there a best one?



Dot Product (math review)

MATH REVIEW | DOT PRODUCTS

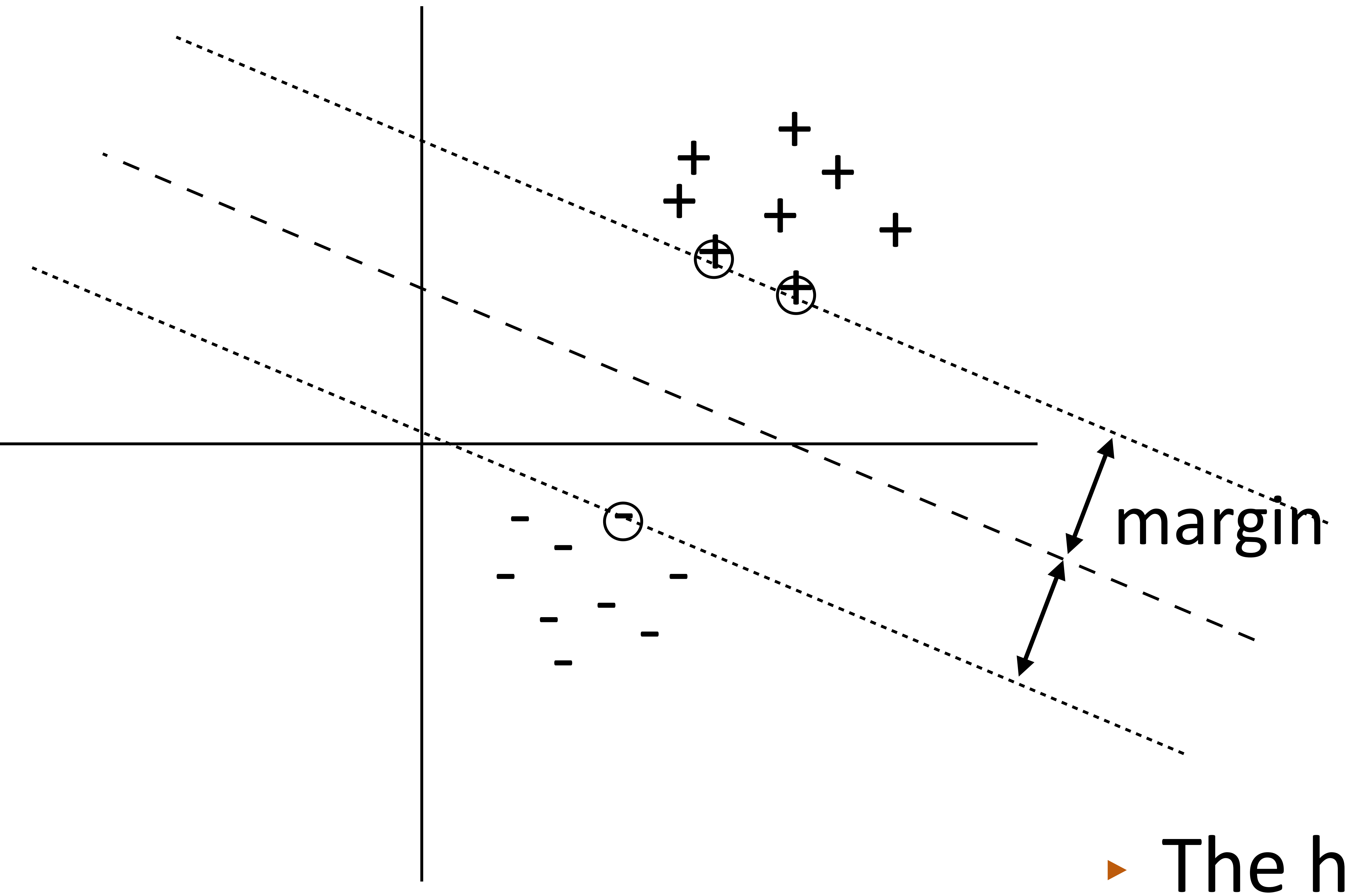
Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ their dot product $\boldsymbol{u} \cdot \boldsymbol{v}$ is $\sum_d u_d v_d$. The dot product grows large and positive when $\boldsymbol{u}$ and $\boldsymbol{v}$ point in same direction, grows large and negative when $\boldsymbol{u}$ and $\boldsymbol{v}$ point in opposite directions, and is zero when their are perpendicular. A useful geometric interpretation of dot products is **projection**. Suppose $||\boldsymbol{u}|| = 1$, so that $\boldsymbol{u}$ is a **unit vector**. We can think of any other vector $\boldsymbol{v}$ as consisting of two components: (a) a component in the direction of $\boldsymbol{u}$ and (b) a component that's perpendicular to $\boldsymbol{u}$. This is depicted geometrically to the right: Here, $\boldsymbol{u} = \langle 0.8, 0.6 \rangle$ and $\boldsymbol{v} = \langle 0.37, 0.73 \rangle$. We can think of $\boldsymbol{v}$ as the sum of two vectors, $\boldsymbol{a}$ and $\boldsymbol{b}$, where $\boldsymbol{a}$ is parallel to $\boldsymbol{u}$ and $\boldsymbol{b}$ is perpendicular. The length of $\boldsymbol{b}$ is exactly $\boldsymbol{u} \cdot \boldsymbol{v} = 0.734$, which is why you can think of dot products as projections: the dot product between $\boldsymbol{u}$ and $\boldsymbol{v}$ is the "projection of $\boldsymbol{v}$ onto $\boldsymbol{u}$."



Credit: Hal Daumé III

Support Vector Machines

- Many separating hyperplanes — is there a best one?



- The hyperplane lies exactly halfway between the nearest positive and negative example.

Support Vector Machines

- Constraint formulation: find *w* via following quadratic program:

Minimize $\|w\|_2^2$

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 <=> 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!

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

N-Slack SVMs

Minimize $$\lambda \|w\|_2^2 + \sum_{j=1}^{m} \xi_j$$

s.t. $\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1 - \xi_j \qquad \forall j \quad \xi_j \geq 0$

- The ξ_j are a "fudge factor" to make all constraints satisfied

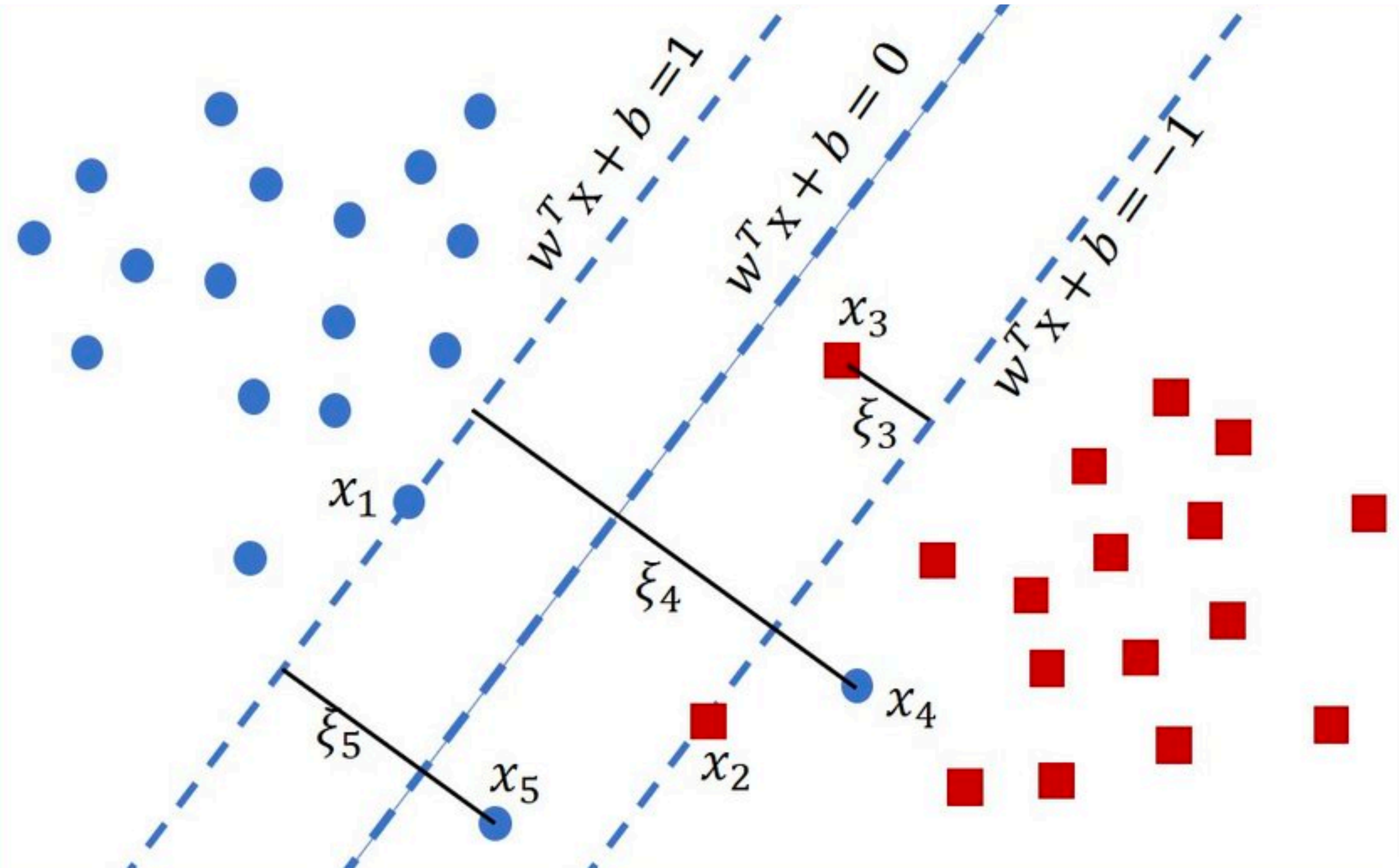


Image credit: Lang Van Tran

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

N-Slack SVMs

Minimize $\lambda\|w\|_2^2 + \sum_{j=1}^{m} \xi_j$

s.t. $\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1 - \xi_j \qquad \forall j \quad \xi_j \geq 0$

- The ξ_j are a "fudge factor" to make all constraints satisfied
- Take the gradient of the objective (flip for maximizing):

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

- Looks like the perceptron! But updates more frequently

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

LR, Perceptron, SVM

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

  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"

LR, Perceptron, SVM

- 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)

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

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

http://ciml.info/dl/v0_99/ciml-v0_99-ch07.pdf

LR, Perceptron, SVM

- Loss on Positive Examples

Logistic Loss

$$log(1 + e^{-z})$$

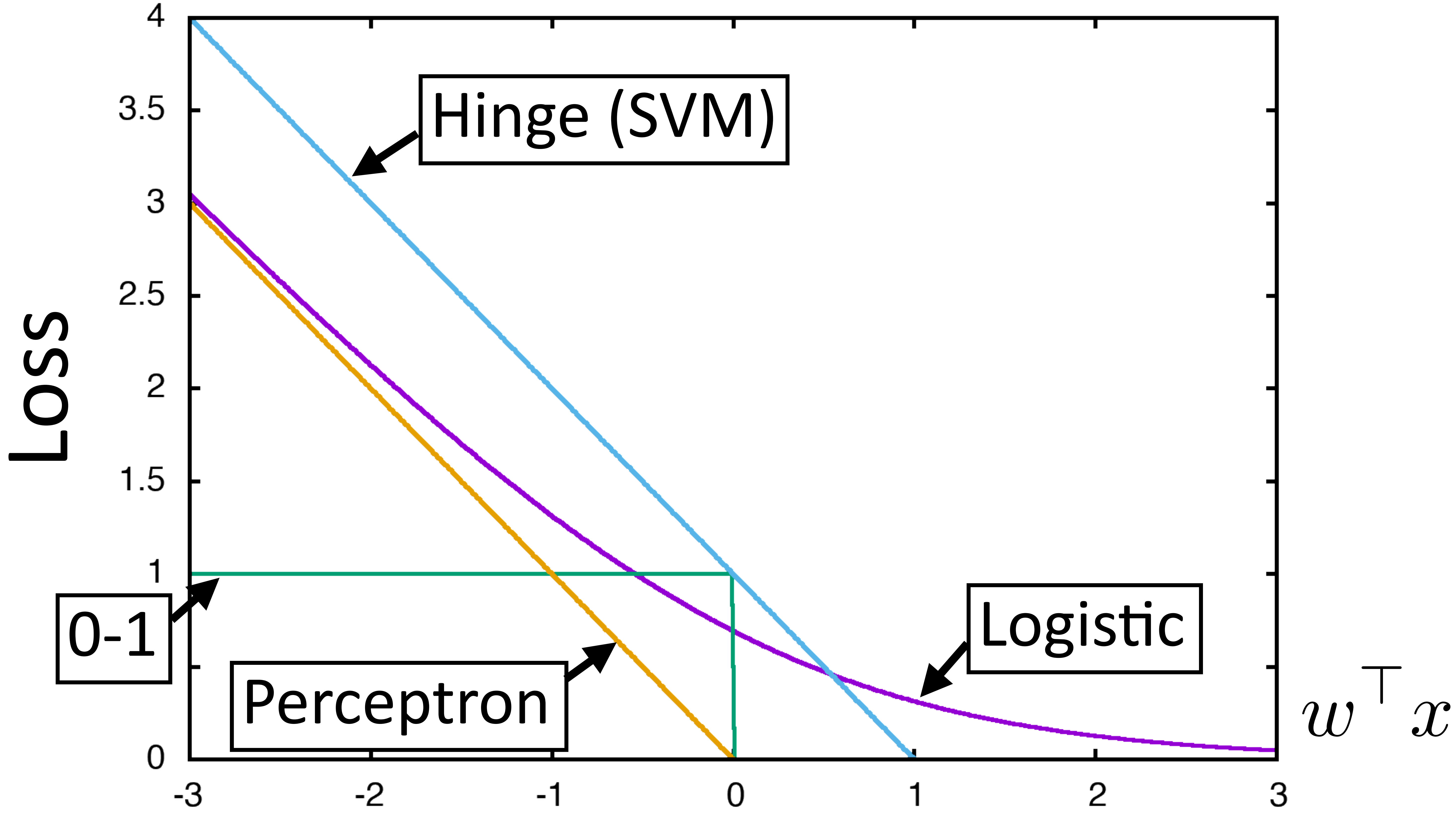
Perceptron Loss

$$\max(0, -z)$$

Hinge Loss

$$\max(0, 1 - z)$$

$$z = w^\top x$$



http://ciml.info/dl/v0_99/ciml-v0_99-ch07.pdf

Optimization — more later …

- 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

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*

Bo Pang, Lillian Lee, Shivakumar Vaithyanathan (2002)

Sentiment Analysis

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

- Simple feature sets can do pretty well!

Bo Pang, Lillian Lee, Shivakumar Vaithyanathan (2002)

Sentiment Analysis

Method	RT-s	MPQA
MNB-uni	77.9	85.3
MNB-bi	**79.0**	**86.3**
SVM-uni	76.2	86.1
SVM-bi	77.7	**86.7**
NBSVM-uni	**78.1**	85.3
NBSVM-bi	**79.4**	**86.3**
RAE	76.8	85.7
RAE-pretrain	77.7	**86.4**
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

Recursive Auto-encoder. Before neural nets had taken off — results weren't that great

Wang and Manning (2012)

Summary

- Logistic regression, SVM, and perceptron are closely related

- SVM and perceptron inference require taking maxes, logistic regression has a similar update but is “softer” due to its probabilistic nature

- All gradient updates: “make it look more like the right thing and less like the wrong thing”

QA Time