Binary Classification

Wei Xu

(many slides from Greg Durrett and Vivek Srikumar)

Administrivia

- Homework 1 will be released soon.
  - 2-3 written questions
  - One programming task:
    - Logistic Regression for Text Classification (Hate Speech)

Outline of the Course

ML and structured prediction for NLP

Neural Networks semantics

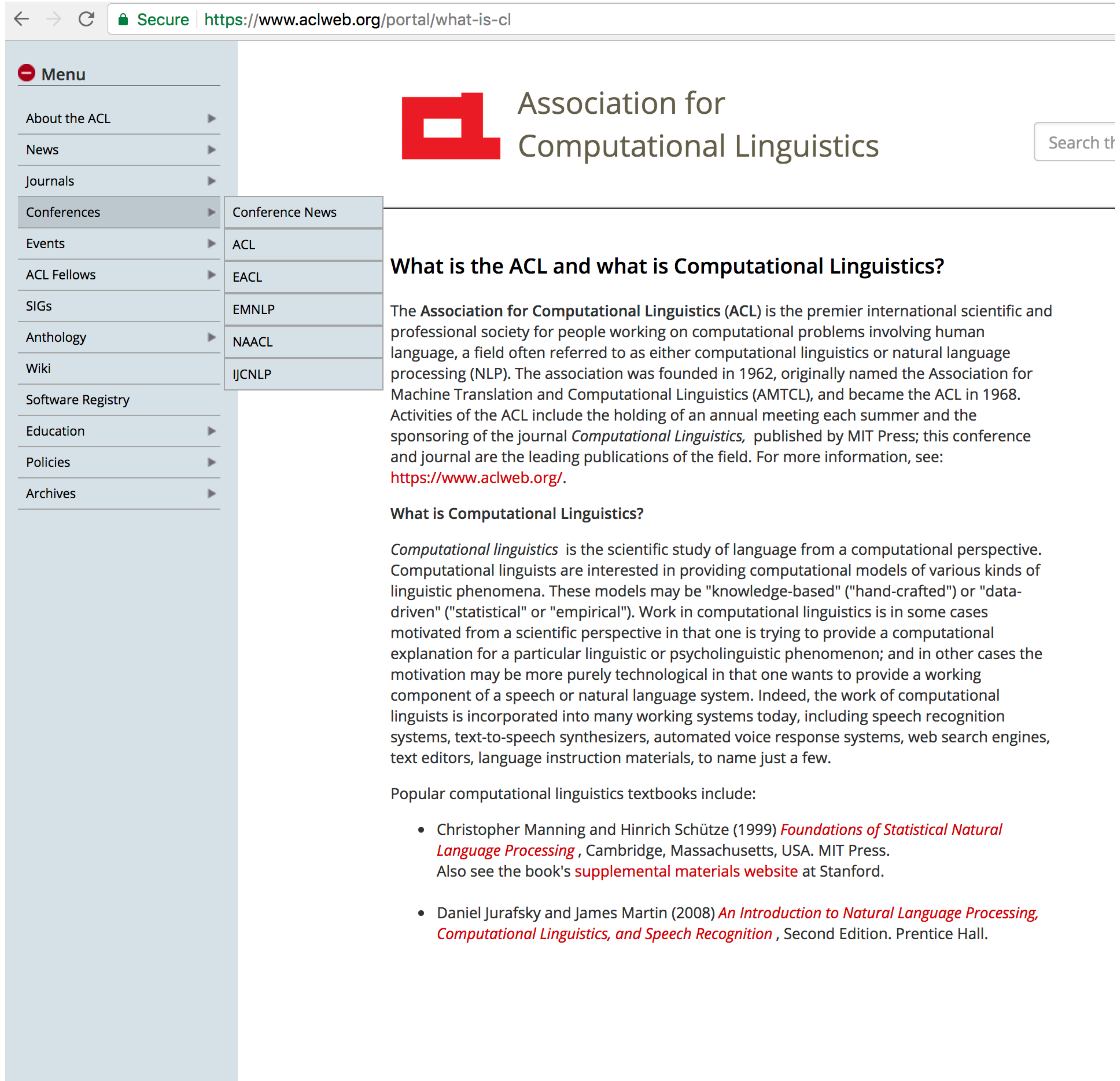
Applications: MT, IE, summarization, dialogue, etc.

Date	Topics (tentative and subject to change)	Readings
1/14/2021	first day of class	
1/18/2021	No class - MLK national holiday	
1/20/2021	**Course Overview - 1st lecture**	J+M 1
1/25/2021	Binary Classification (naive bayes and logisitic regression)	J+M 4, Eisenstein 2.0-2.5, 4.1,4.3-4.5,
1/27/2021	Multiclass Classification	J+M 5, Eisenstein 4.2
2/1/2021	Neural Networks (feedforward networks)	Eisenstein 3.1-3.3, J+M 7.1-7.4
2/3/2021	Neural Networks (back propogation)	Eisenstein 3.1-3.3, J+M 7.1-7.4
2/8/2021	PyTorch Tutorial, Sequence Models	J+M 8
2/10/2021	Viterbi Algorithm	Eisenstein 7.0-7.4
2/15/2021	Conditional Random Fields	Eisenstein 7.5, 8.3
2/17/2021	N-gram Language Models	
2/22/2021	Word Embeddings	Eisenstein 3.3.4, 14.5, 14.6, J+M 6
2/24/2021	Recurrent Neural Networks	J+M 9, Goldberg 10,11
3/1/2021	Convolutional Neural Networks	Goldberg 9, Eisenstein 3.4, 7.6
3/3/2021	Statistical Machine Translation	Eisenstein 18.1, 18.2
3/8/2021	**(Guest Lecture)**	
3/10/2021	Sequence-to-Sequence Model	J+M 10
3/15/2021	Attention and Copy Mechanism	Eisenstein 18.3, 18.4
3/17/2021	Question Answering / Reading Comprehension	SQuaD, BiDAF
3/17/2021	Withdrawal deadline	
3/22/2021	Parsing	
3/24/2021	No class - mid-semester break	
3/29/2021	Neural Machine Translation	Google NMT
3/31/2021	Transformer Model	Attention is all you need
4/5/2021	**Generation (Guest Lecture)**	
4/7/2021	Information Extraction	Eisenstein 13, 17
4/12/2021	**Dialog (Guest Lecture)**	
4/14/2021	Pre-trained Language Models / BERT	
4/19/2021	**Computational Social Science (Guest Lecture)**	
4/21/2021	Speech Recognition	
4/26/2021	Final class day	

NLP Research

	Area	Long submissions	Accepts	Accept rate (%)
1.	Applications	65	14	28.8
2.	Dialogue and Interactive Systems	126	38	30.2
3.	Discourse and Pragmatics	33	7	21.2
4.	Document Analysis	48	8	16.7
5.	Generation	96	32	33.3
6.	Information Extraction and Text Mining	155	37	23.9
7.	Linguistic Theories, Cognitive Modeling and Psycholinguistics	39	9	23.1
8.	Machine Learning	148	38	25.7
8.	Machine Translation	102	27	26.5
10.	Multidisciplinary and Area Chair COI	69	21	30.4
11.	Multilinguality	43	11	25.6
12.	Phonology Morphology and Word Segmentation	26	7	26.9
13.	Question Answering	99	32	32.3
14.	Resources and Evaluation	70	26	37.1
15.	Sentence-level semantics	69	14	20.3
15.	Sentiment Analysis and Argument Mining	91	24	26.4
17.	Social Media	51	14	27.5
18.	Summarization	48	11	22.9
19.	Tagging Chunking Syntax and Parsing	50	17	34.0
20.	Textual Inference and Other Areas of Semantics	44	16	36.4
21.	Vision Robotics Multimodal Grounding and Speech	56	20	35.7
22.	Word-level Semantics	78	20	25.6

ACL 2019 conference


Secure https://www.aclweb.org/portal/what-is-cl

Menu

About the ACL
News
Journals
Conferences
Events
ACL Fellows
SIGs
Anthology
Wiki
Software Registry
Education
Policies
Archives

Conference News
ACL
EACL
EMNLP
NAACL
IJCNLP

Association for Computational Linguistics

Search th

What is the ACL and what is Computational Linguistics?

The **Association for Computational Linguistics (ACL)** is the premier international scientific and professional society for people working on computational problems involving human language, a field often referred to as either computational linguistics or natural language processing (NLP). The association was founded in 1962, originally named the Association for Machine Translation and Computational Linguistics (AMTCL), and became the ACL in 1968. Activities of the ACL include the holding of an annual meeting each summer and the sponsoring of the journal *Computational Linguistics*, published by MIT Press; this conference and journal are the leading publications of the field. For more information, see: https://www.aclweb.org/.

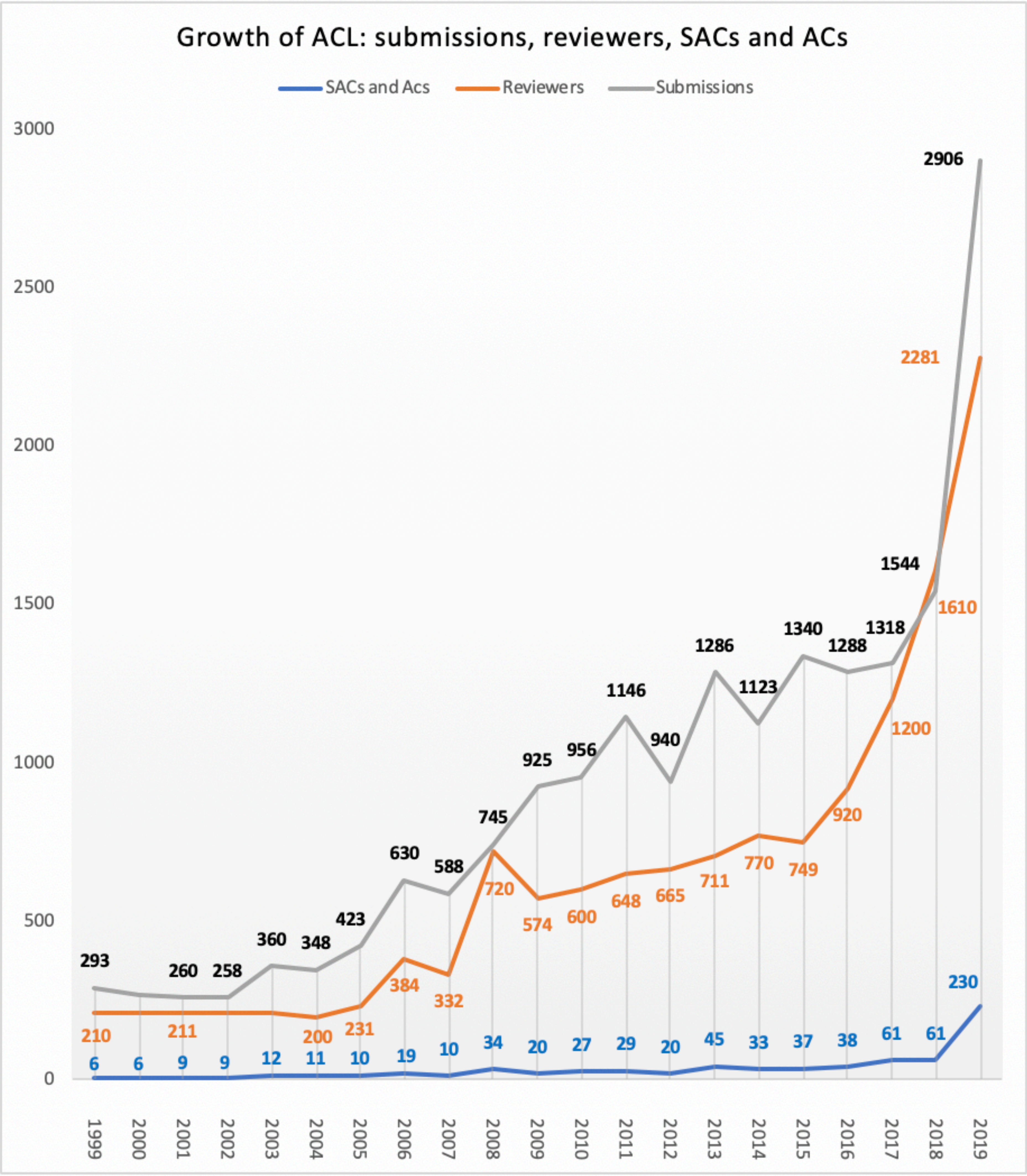
What is Computational Linguistics?

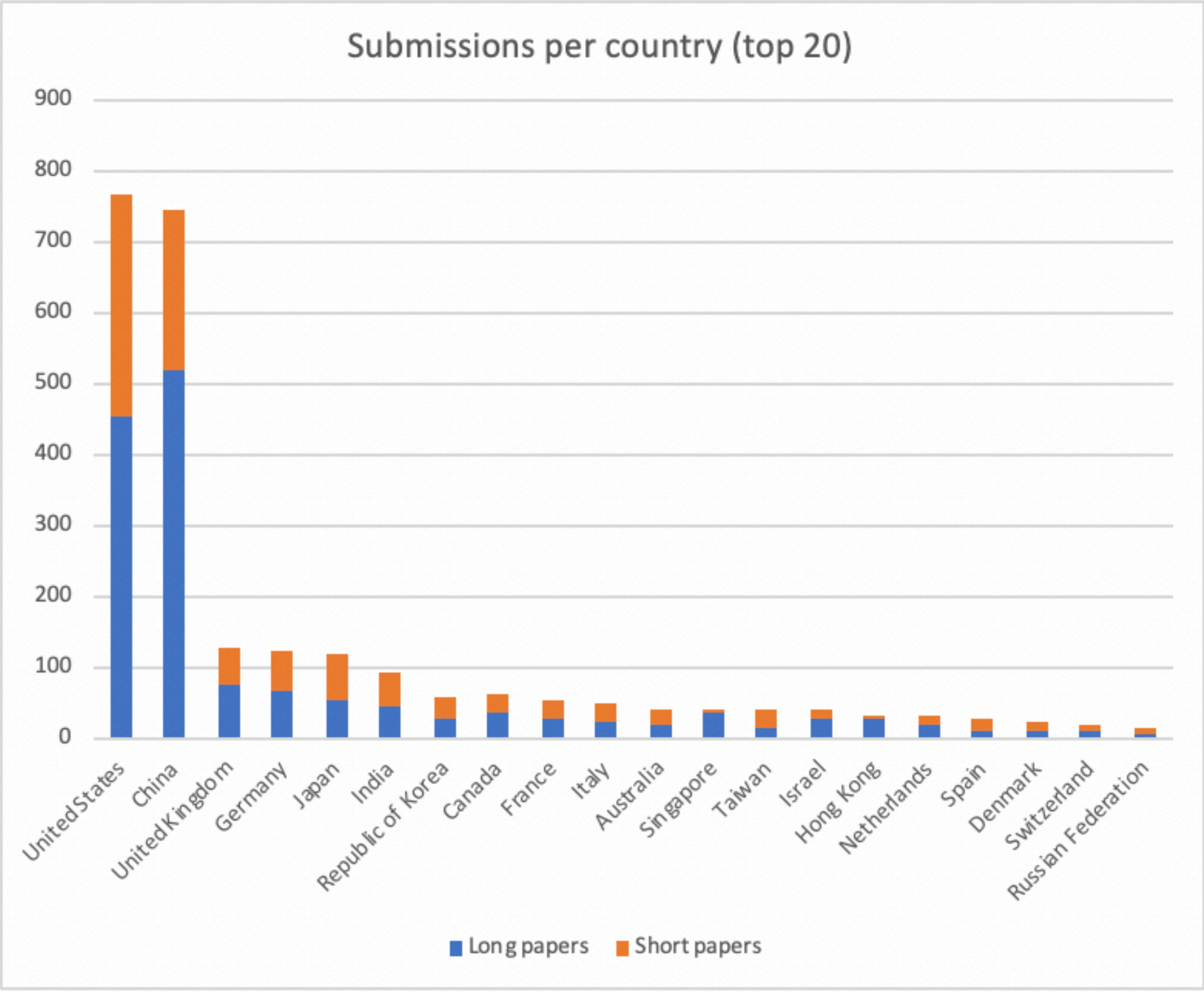
Computational linguistics is the scientific study of language from a computational perspective. Computational linguists are interested in providing computational models of various kinds of linguistic phenomena. These models may be "knowledge-based" ("hand-crafted") or "data-driven" ("statistical" or "empirical"). Work in computational linguistics is in some cases motivated from a scientific perspective in that one is trying to provide a computational explanation for a particular linguistic or psycholinguistic phenomenon; and in other cases the motivation may be more purely technological in that one wants to provide a working component of a speech or natural language system. Indeed, the work of computational linguists is incorporated into many working systems today, including speech recognition systems, text-to-speech synthesizers, automated voice response systems, web search engines, text editors, language instruction materials, to name just a few.

Popular computational linguistics textbooks include:

- Christopher Manning and Hinrich Schütze (1999) *Foundations of Statistical Natural Language Processing*, Cambridge, Massachusetts, USA. MIT Press. Also see the book's supplemental materials website at Stanford.
- Daniel Jurafsky and James Martin (2008) *An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Second Edition. Prentice Hall.

ACL'19 at a Glance





This 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

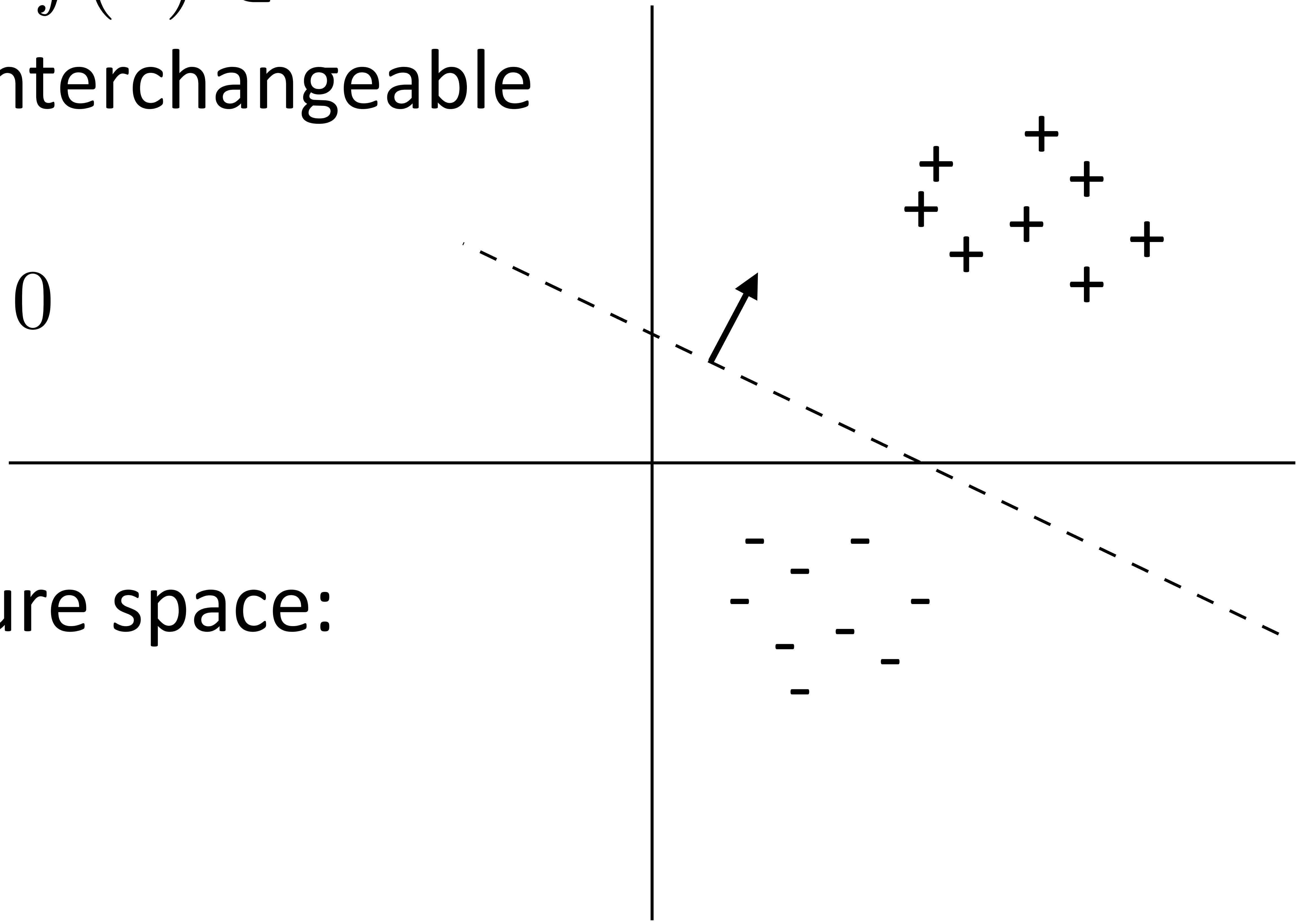
Classification

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

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



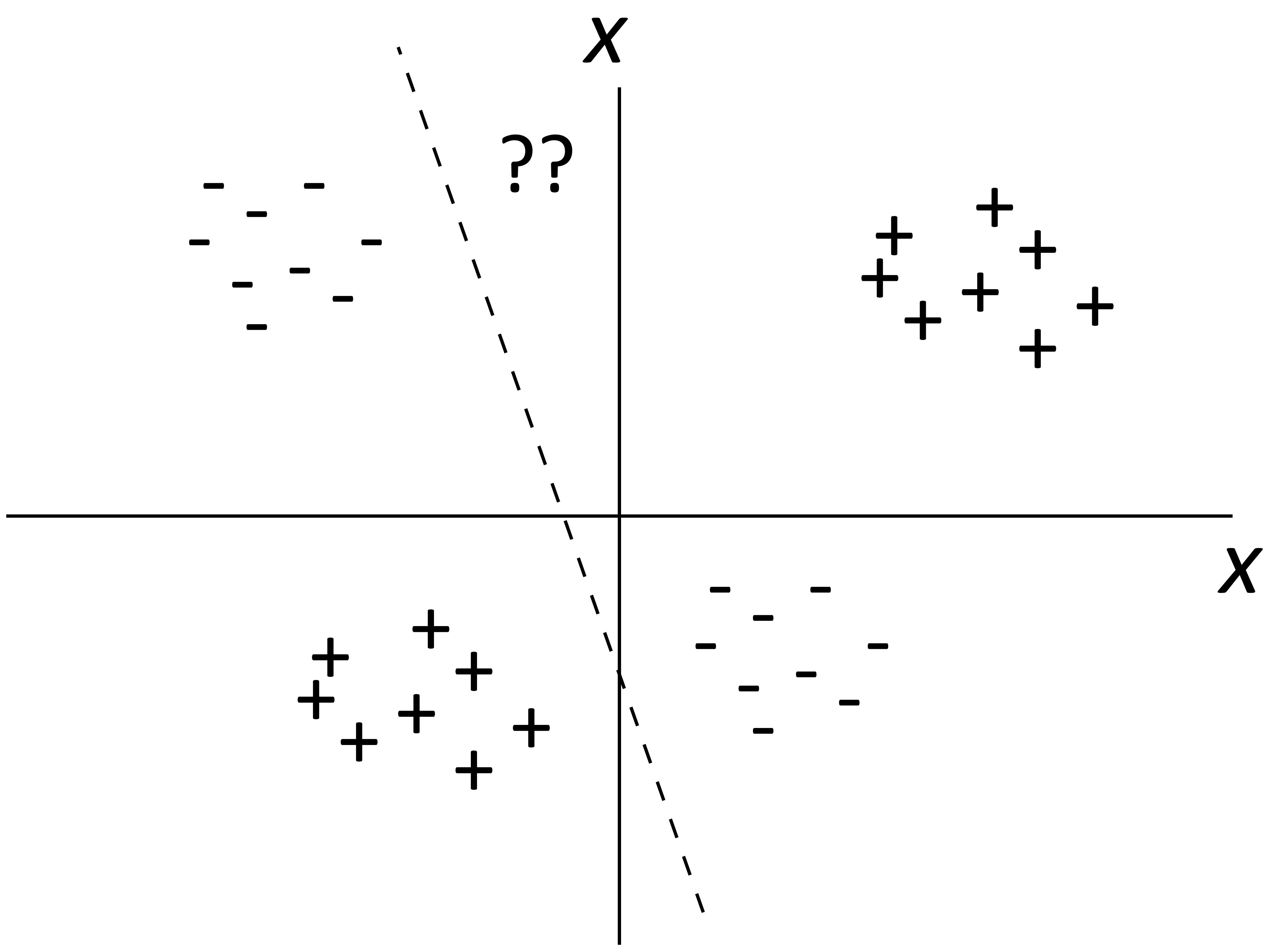
- Can delete bias if we augment feature space:

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

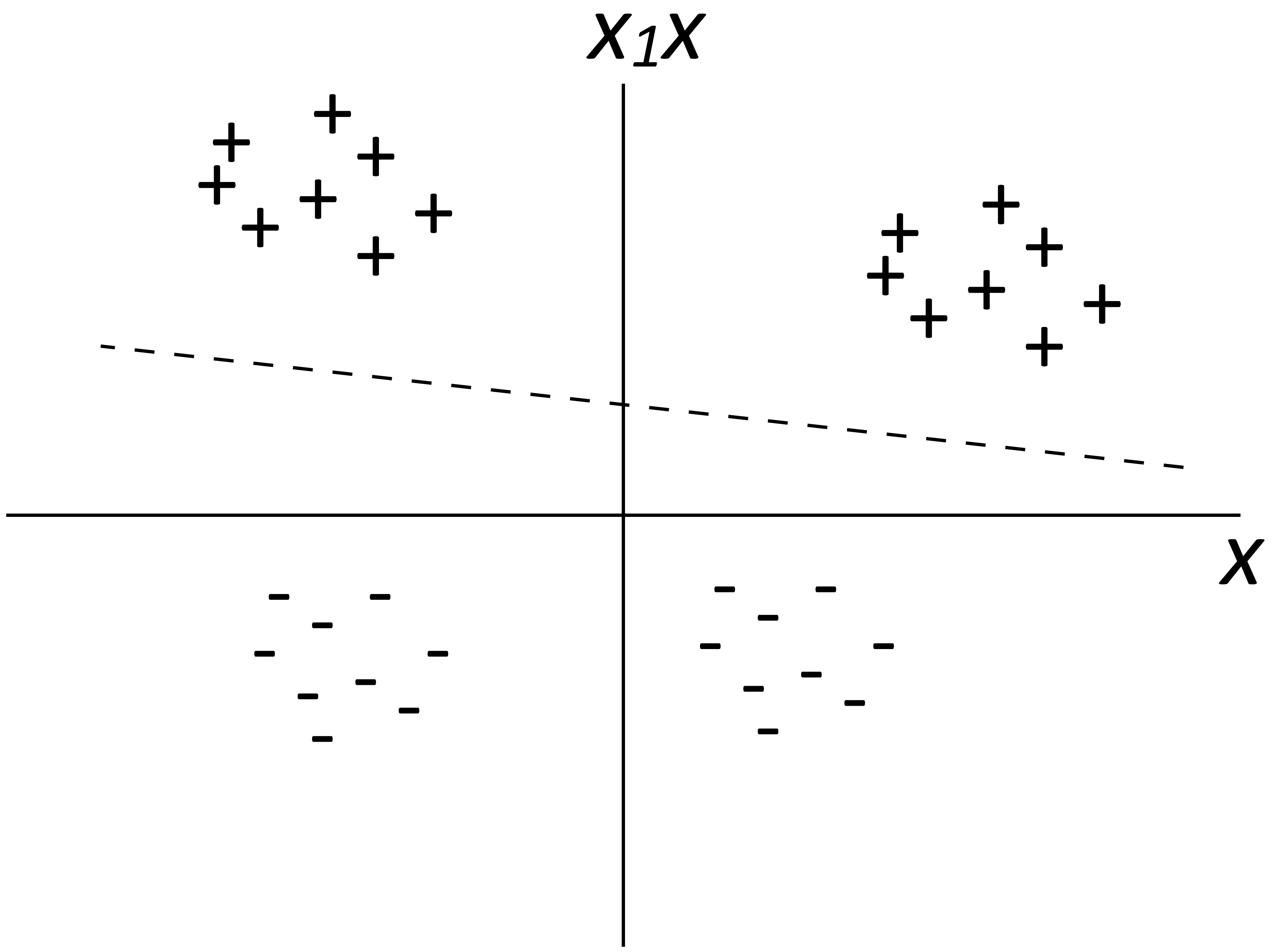
  ↓

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

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

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		0	1	1	0 …

- 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*

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

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

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

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

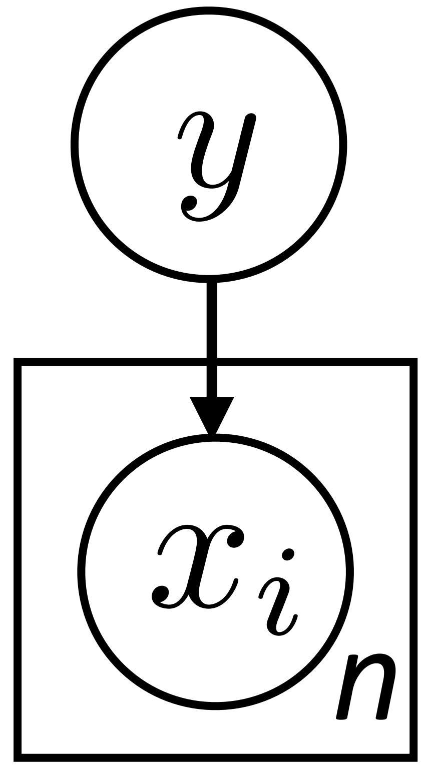
Bayes' Rule

constant: irrelevant for finding the max

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

"Naive" assumption: conditional independence

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



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

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 = $1.\ 1\ 1\ 1\ldots \times 2^{+1023}$

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

- 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

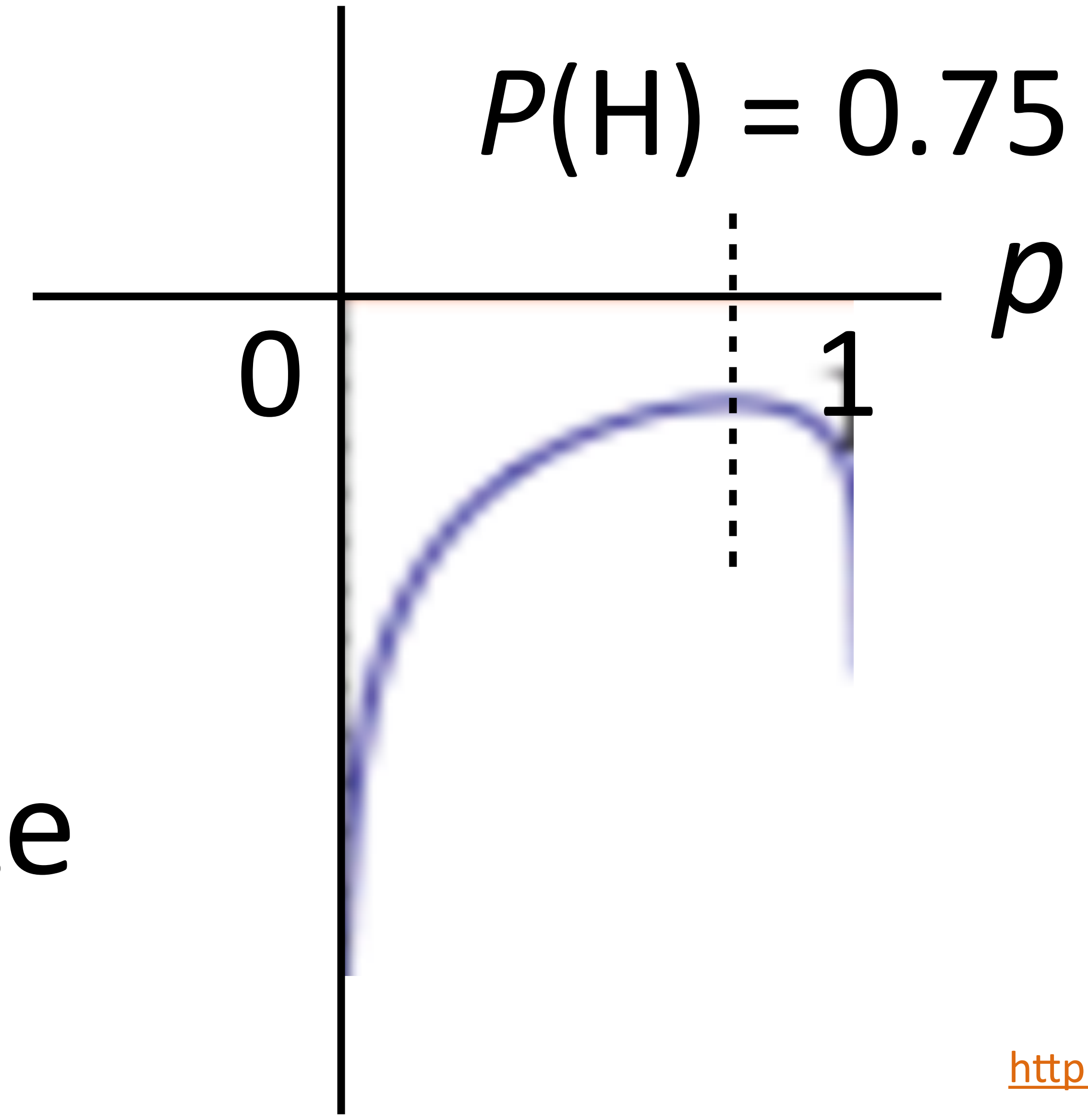
Maximum Likelihood Estimation

- 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

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

log likelihood



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

http://fooplot.com/

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	—

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

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

prior

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

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

word likelihood

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

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

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

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

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($word_i$| +) and P($word_i$| -):
    - frequency of $word_i$ 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

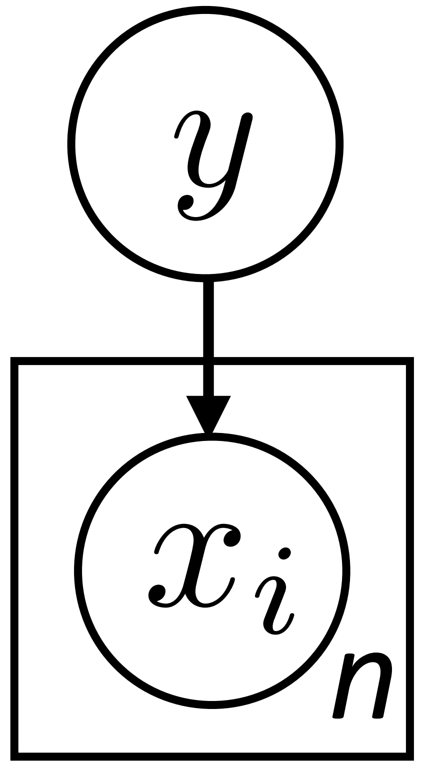
- Multinominal document model"
  - A document is represented by integer elements
  - Feature value is the frequency of that word in the document
  - See textbook and lecture note by Hiroshi Shimodaira linked below for more details

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

Naive Bayes: 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, …)

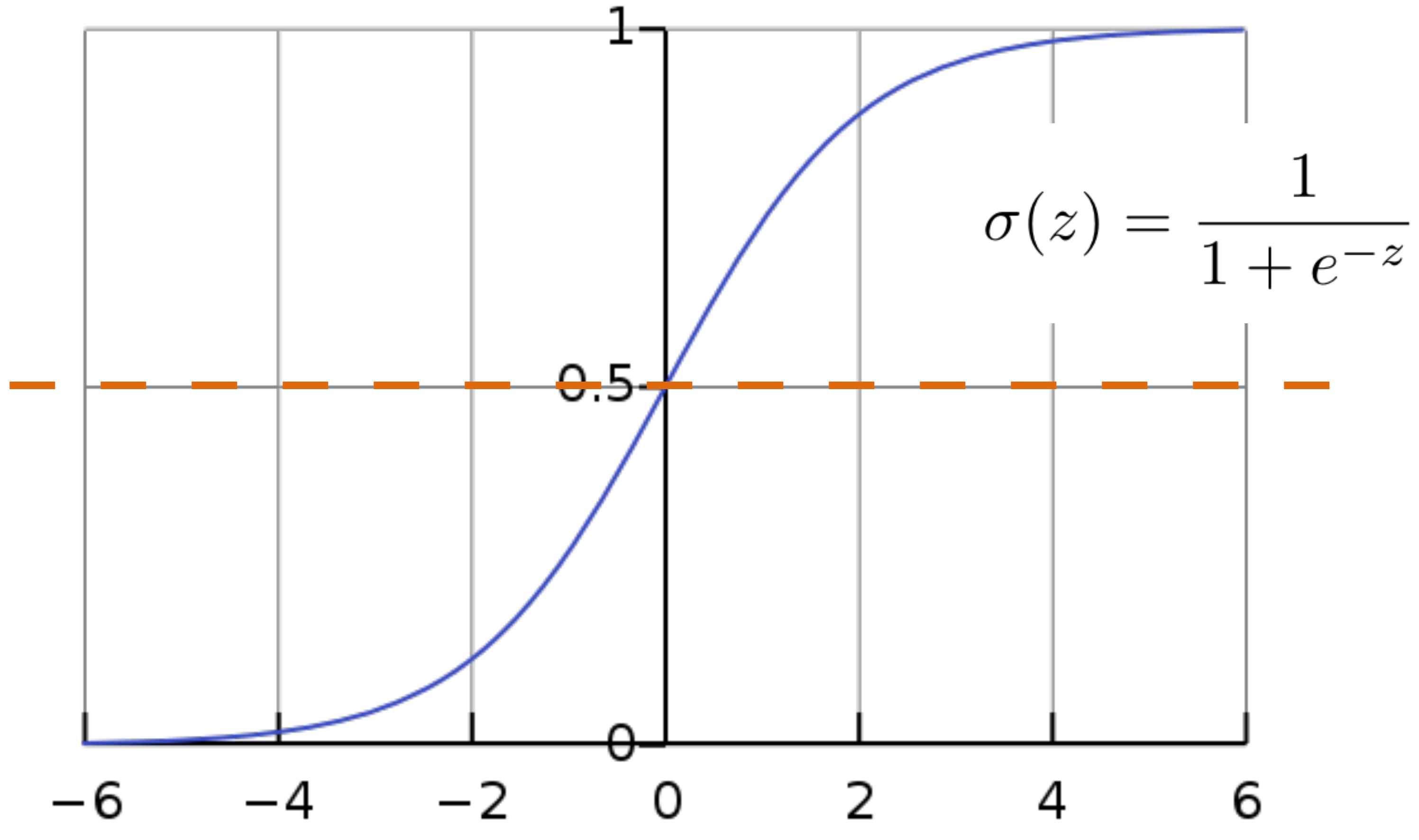
Generative vs. Discriminative Models

- 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

Break!

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

Logistic Regression

chain rule:

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

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

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

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

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 $= x_{ji}(y_j - P(y_j = +|x_j))$

  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

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

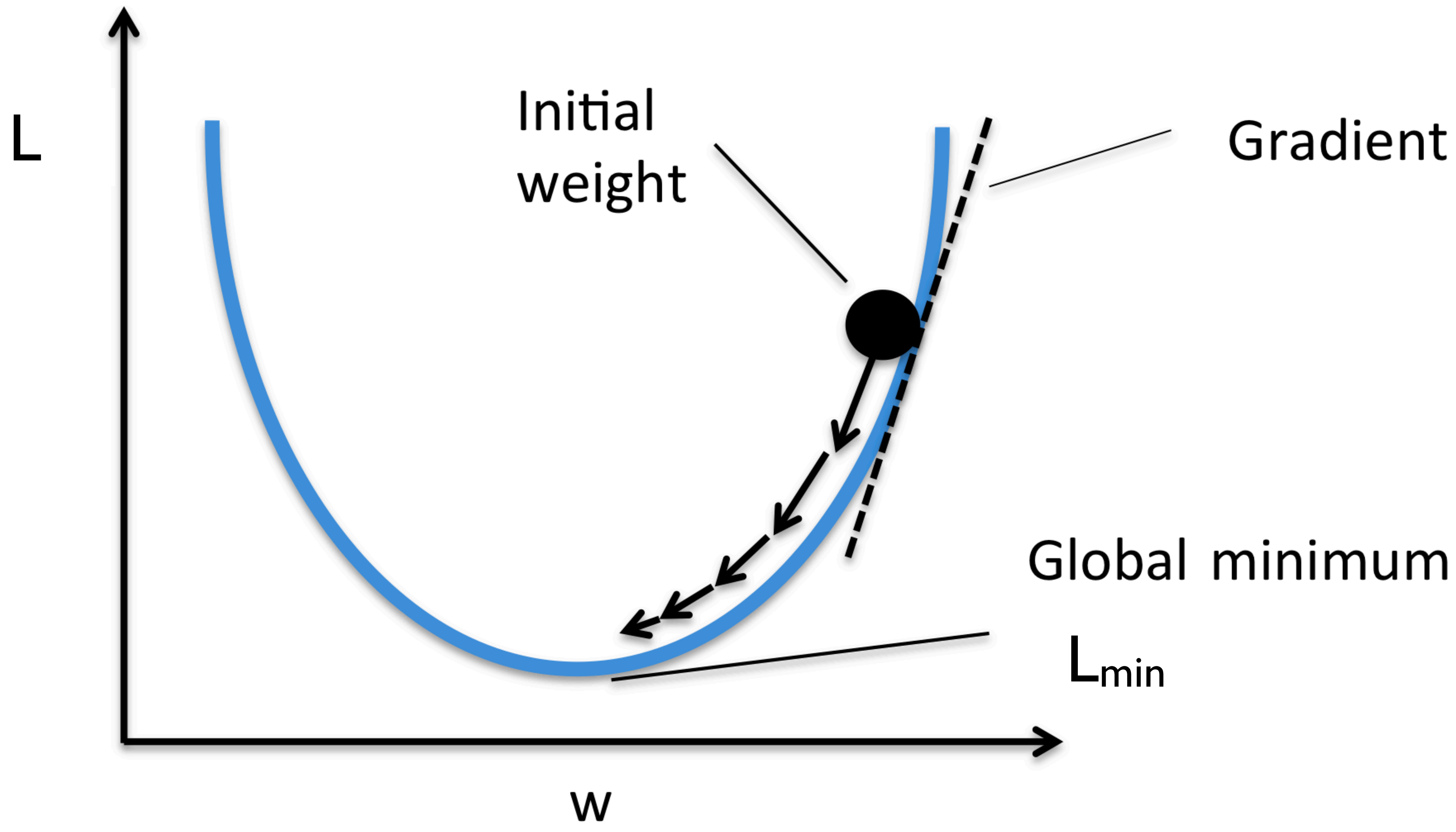
- Can combine these gradients as $\dfrac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$

- Training set log-likelihood: $\mathcal{L}(w)$

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

Gradient Decent

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



Repeat until convergence {

from j = 1 to m

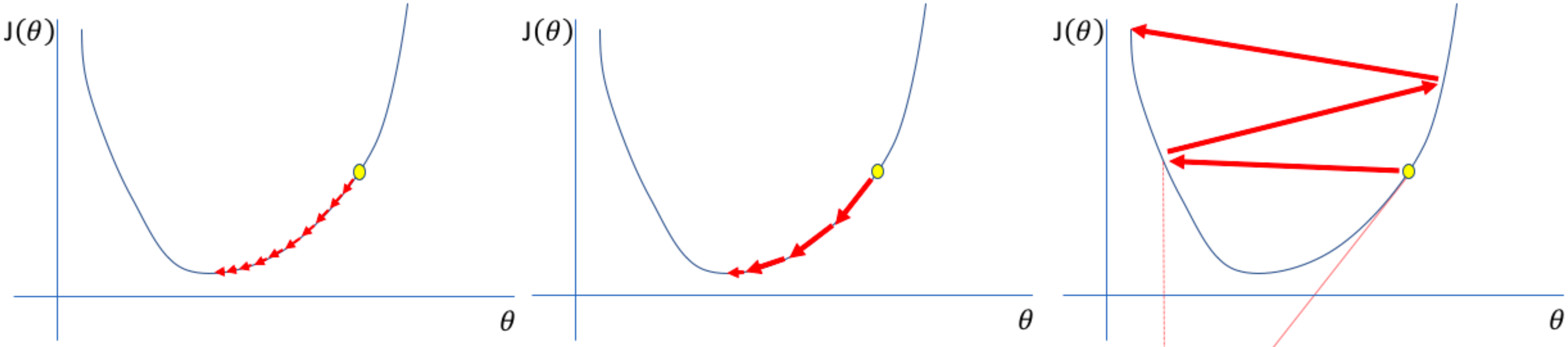
$$w := w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$$

}

learning rate (step size)

Learning Rate

Too low | **Just right** | **Too high**



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

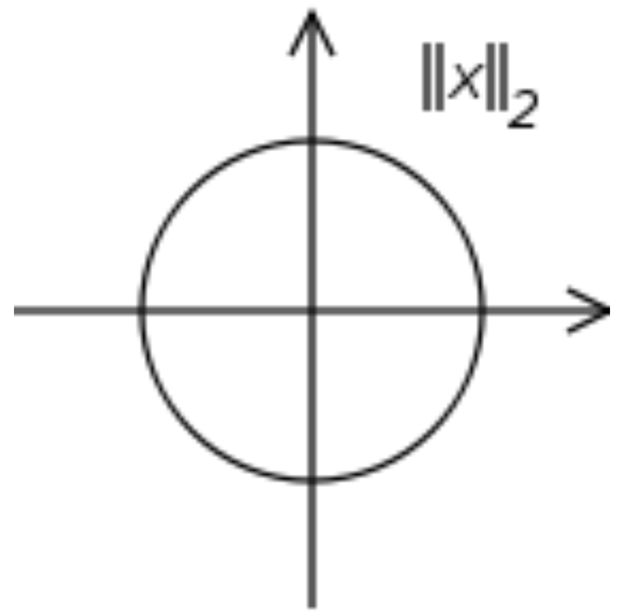
Too large of a learning rate causes drastic updates which lead to divergent behaviors

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

$\mathrm{argmax}_y P(y|x)$ 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: $w^\top x > 0$
  - 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

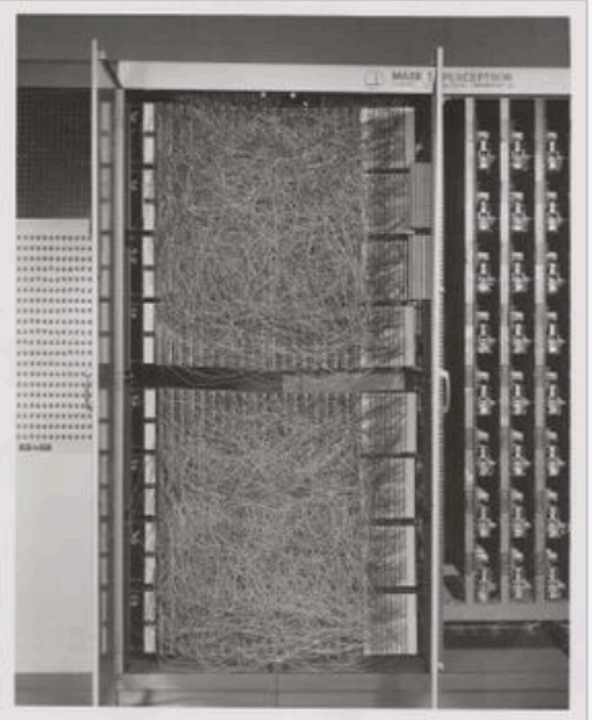
- Guaranteed to eventually separate the data if the data are separable

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

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]

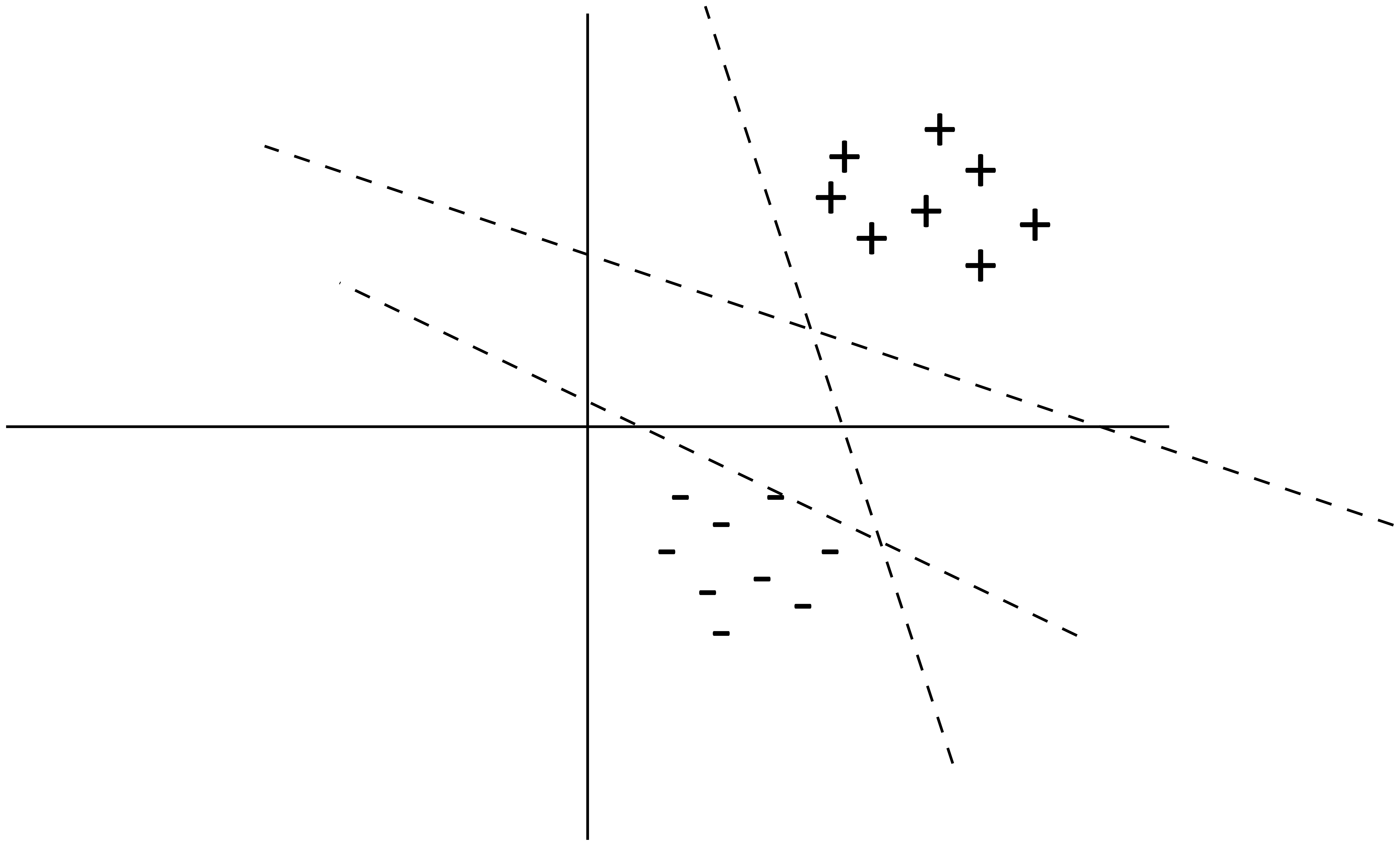
The perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.[7]

Frank Rosenblatt (1928-1971)

PhD 1956 from Cornell

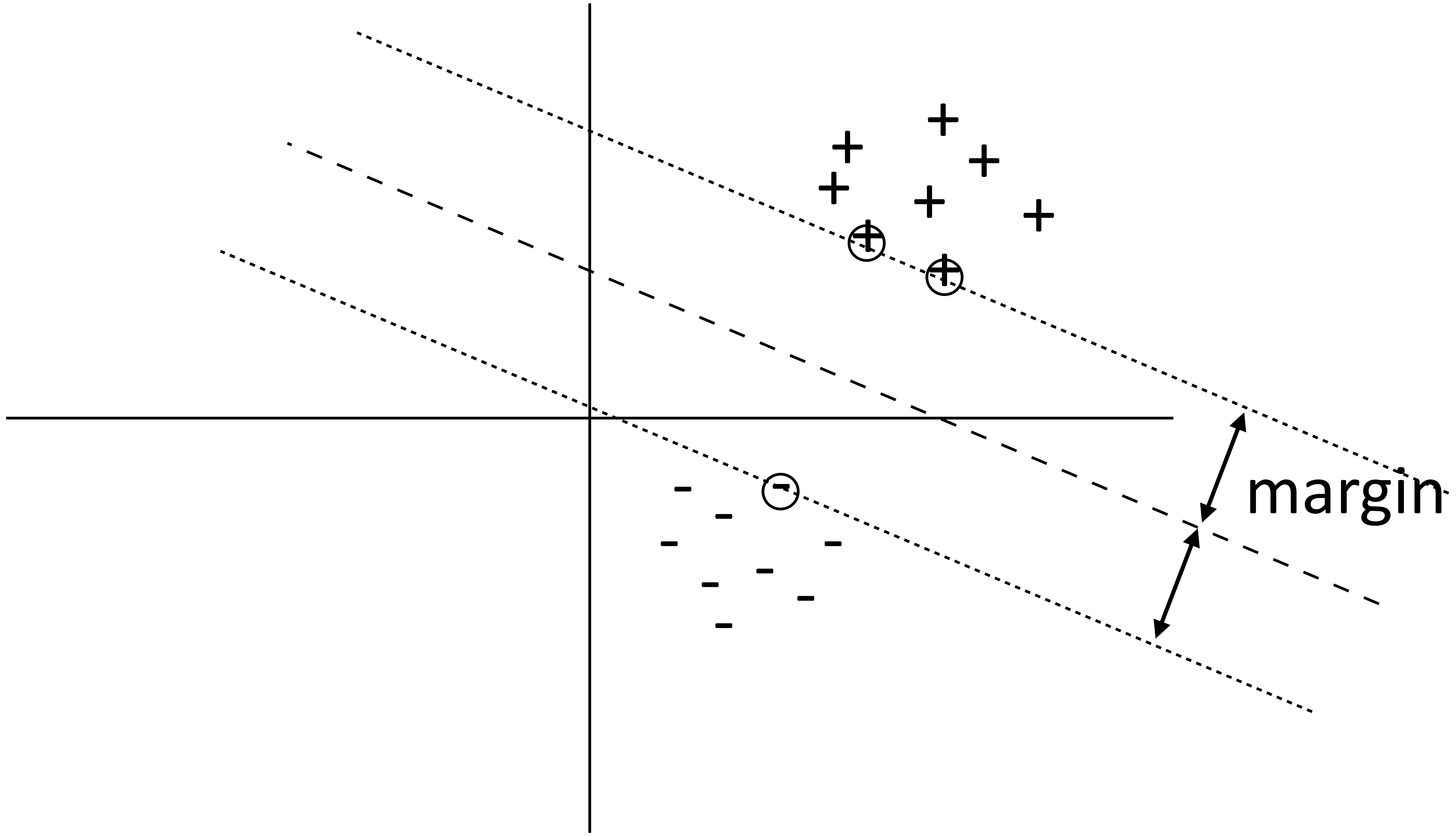
Support Vector Machines (extracurricular)

- Many separating hyperplanes — is there a best one?



Support Vector Machines (extracurricular)

- Many separating hyperplanes — is there a best one?



Support Vector Machines (extracurricular)

- 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!

N-Slack SVMs (extracurricular)

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:

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

$$= x_{ji} \text{ if } y_j = 1, \; -x_{ji} \text{ if } y_j = 0$$

- Looks like the perceptron! But updates more frequently

LR, Perceptron, SVM (extracurricular)

- Gradients on Positive Examples

Logistic regression

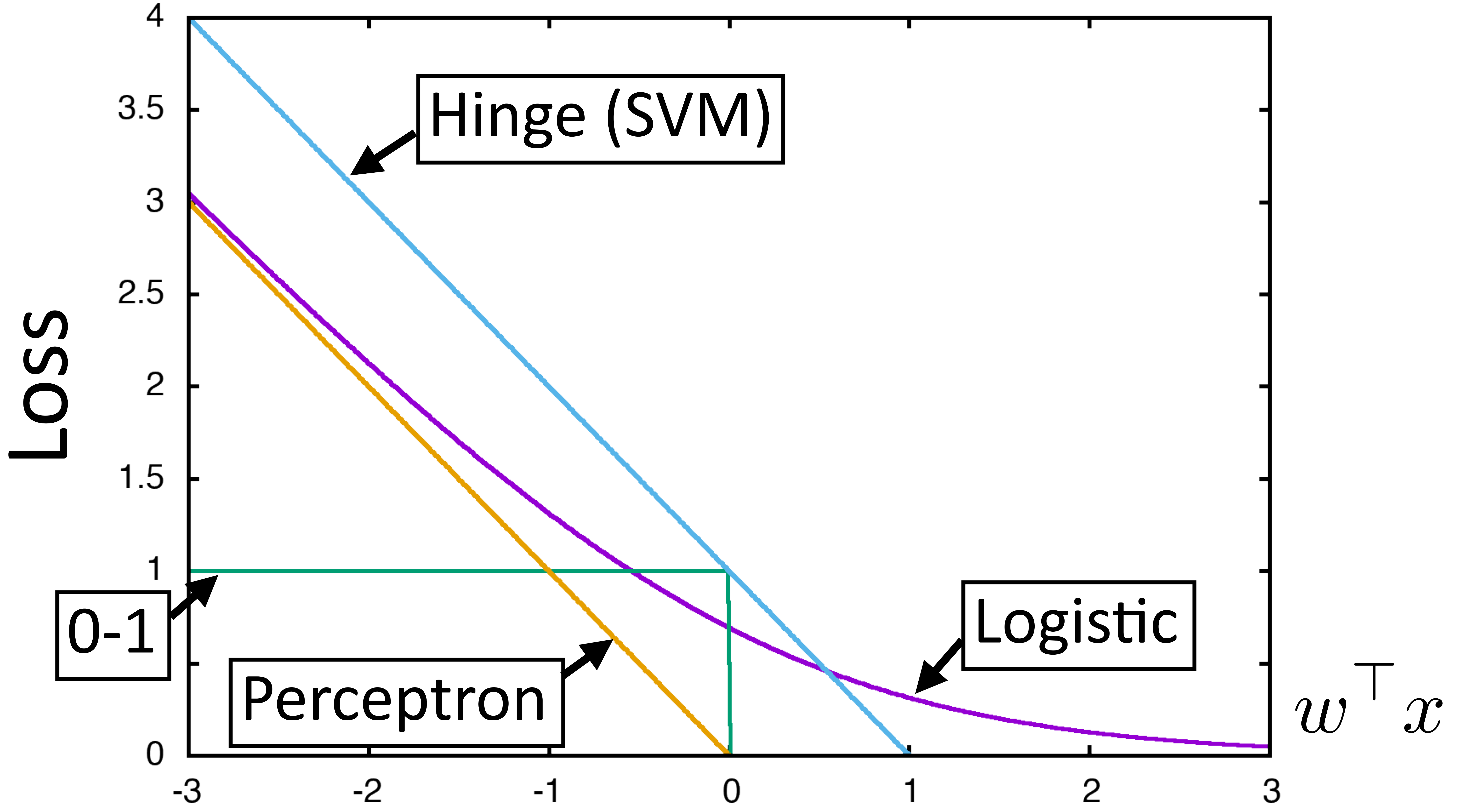
$$x(1 - \mathrm{logistic}(w^\top x))$$

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*

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

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

Wang and Manning (2012)

Recap

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

  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”

Optimization — next time...

- 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