Machine Learning Recap (linear classification)

Wei Xu

(many slides from Greg Durrett)

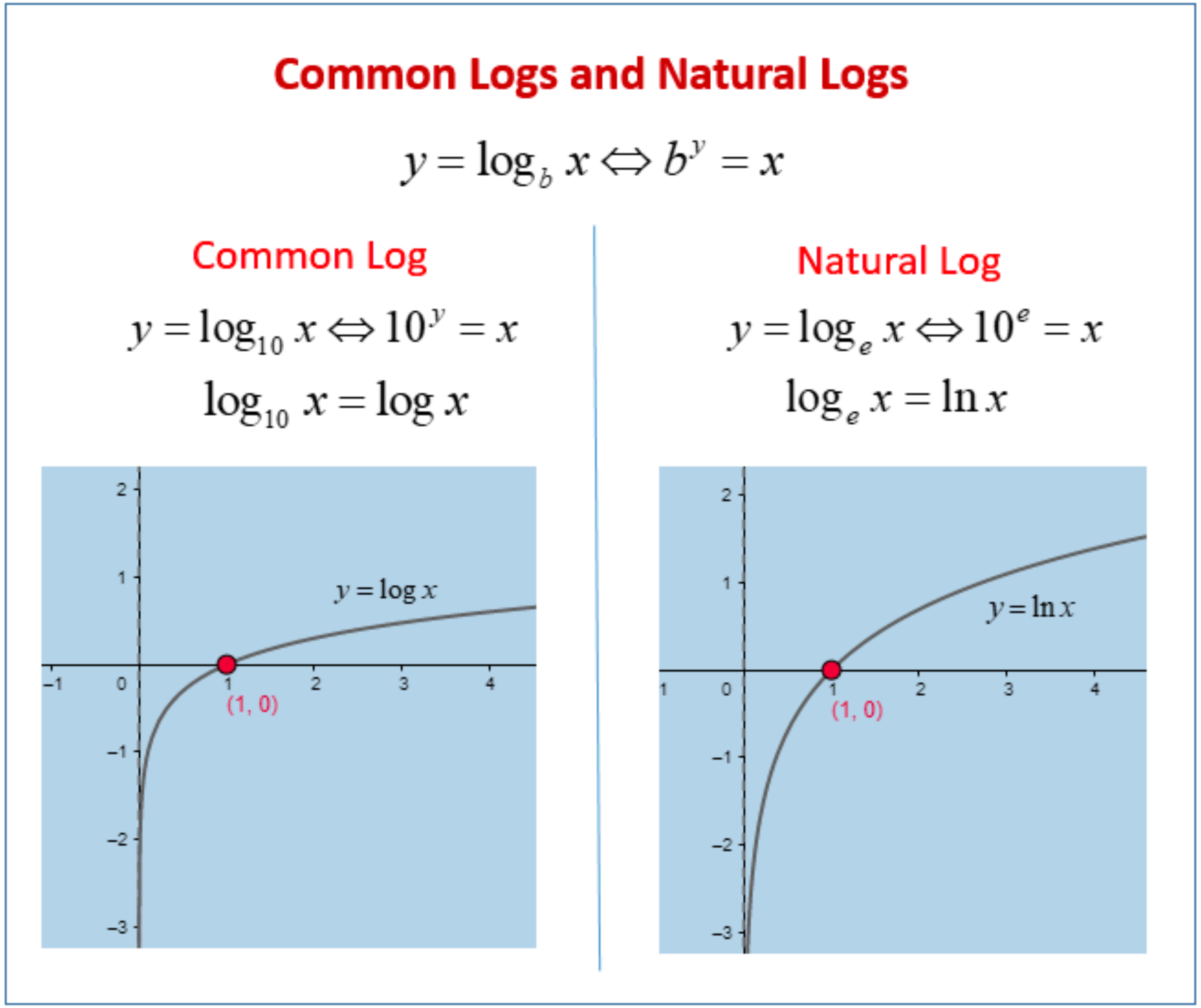
Course website: https://cocoxu.github.io/CS7650_spring2024/

Trivia Time

Q: max/min of log prob.?

Trivia Time

Q: max/min of log prob.?



Common Logs and Natural Logs

$$y = \log_b x \Leftrightarrow b^y = x$$

Common Log

$$y = \log_{10} x \Leftrightarrow 10^y = x$$

$$\log_{10} x = \log x$$

Natural Log

$$y = \log_e x \Leftrightarrow 10^e = x$$

$$\log_e x = \ln x$$

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

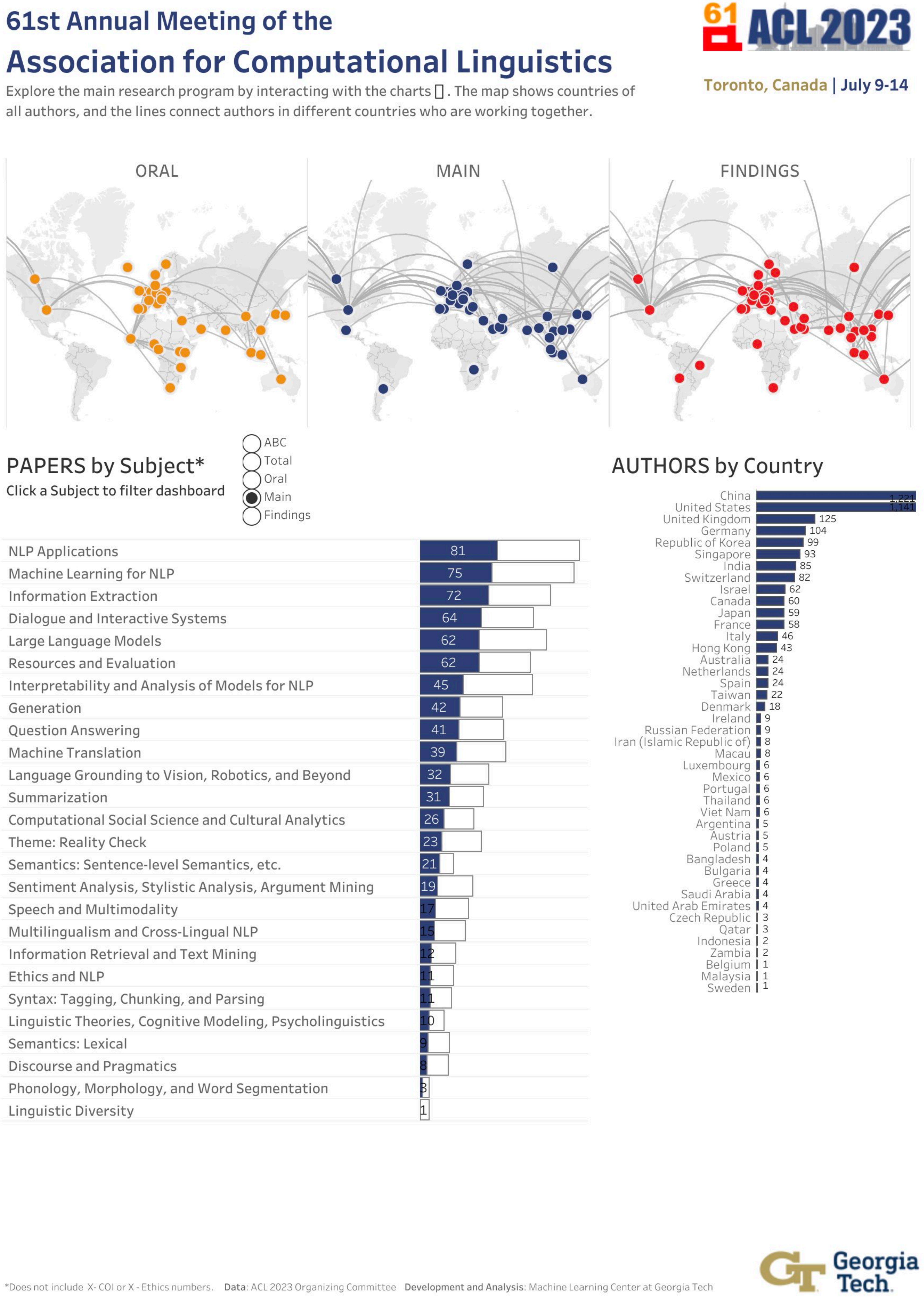
There will be a lot of math and programming!

Background Test

- Problem Set 0 (math background) is released, **due Thursday Jan 11**.
- Project 0 (programming - logistic regression) is also released, due Friday Jan 19.
- Take **CS 4641/7641 Machine Learning** and (Math 2550 or Math 2551 or Math 2561 or Math 2401 or Math 24X1 or 2X51) before 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.

NLP and ML Conferences

- ACL/NAACL/EMNLP
- ICRL/NeurIPS/ICML
- and …



61st Annual Meeting of the
Association for Computational Linguistics

Explore the main research program by interacting with the charts . The map shows countries of all authors, and the lines connect authors in different countries who are working together.

PAPERS by Subject*

Click a Subject to filter dashboard

○ ABC
○ Total
○ Oral
● Main
○ Findings

NLP Applications	81
Machine Learning for NLP	75
Information Extraction	72
Dialogue and Interactive Systems	64
Large Language Models	62
Resources and Evaluation	62
Interpretability and Analysis of Models for NLP	45
Generation	42
Question Answering	41
Machine Translation	39
Language Grounding to Vision, Robotics, and Beyond	32
Summarization	31
Computational Social Science and Cultural Analytics	26
Theme: Reality Check	23
Semantics: Sentence-level Semantics, etc.	21
Sentiment Analysis, Stylistic Analysis, Argument Mining	19
Speech and Multimodality	17
Multilingualism and Cross-Lingual NLP	15
Information Retrieval and Text Mining	12
Ethics and NLP	11
Syntax: Tagging, Chunking, and Parsing	11
Linguistic Theories, Cognitive Modeling, Psycholinguistics	10
Semantics: Lexical	9
Discourse and Pragmatics	8
Phonology, Morphology, and Word Segmentation	3
Linguistic Diversity	1

AUTHORS by Country

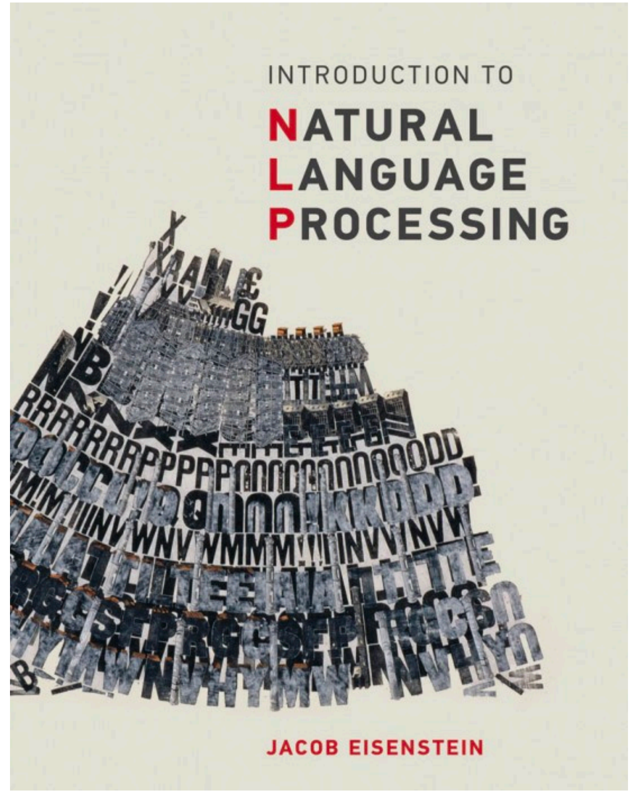
China	1,221
United States	1,141
United Kingdom	125
Germany	104
Republic of Korea	99
Singapore	93
India	85
Switzerland	82
Israel	62
Canada	60
Japan	59
France	58
Italy	46
Hong Kong	43
Australia	24
Netherlands	24
Spain	24
Taiwan	22
Denmark	18
Ireland	9
Russian Federation	9
Iran (Islamic Republic of)	8
Macau	8
Luxembourg	6
Mexico	6
Portugal	6
Thailand	6
Viet Nam	6
Argentina	5
Austria	5
Poland	5
Bangladesh	4
Bulgaria	4
Greece	4
Saudi Arabia	4
United Arab Emirates	4
Czech Republic	3
Qatar	3
Indonesia	2
Zambia	2
Belgium	1
Malaysia	1
Sweden	1

*Does not include X- COI or X - Ethics numbers. **Data**: ACL 2023 Organizing Committee **Development and Analysis**: Machine Learning Center at Georgia Tech

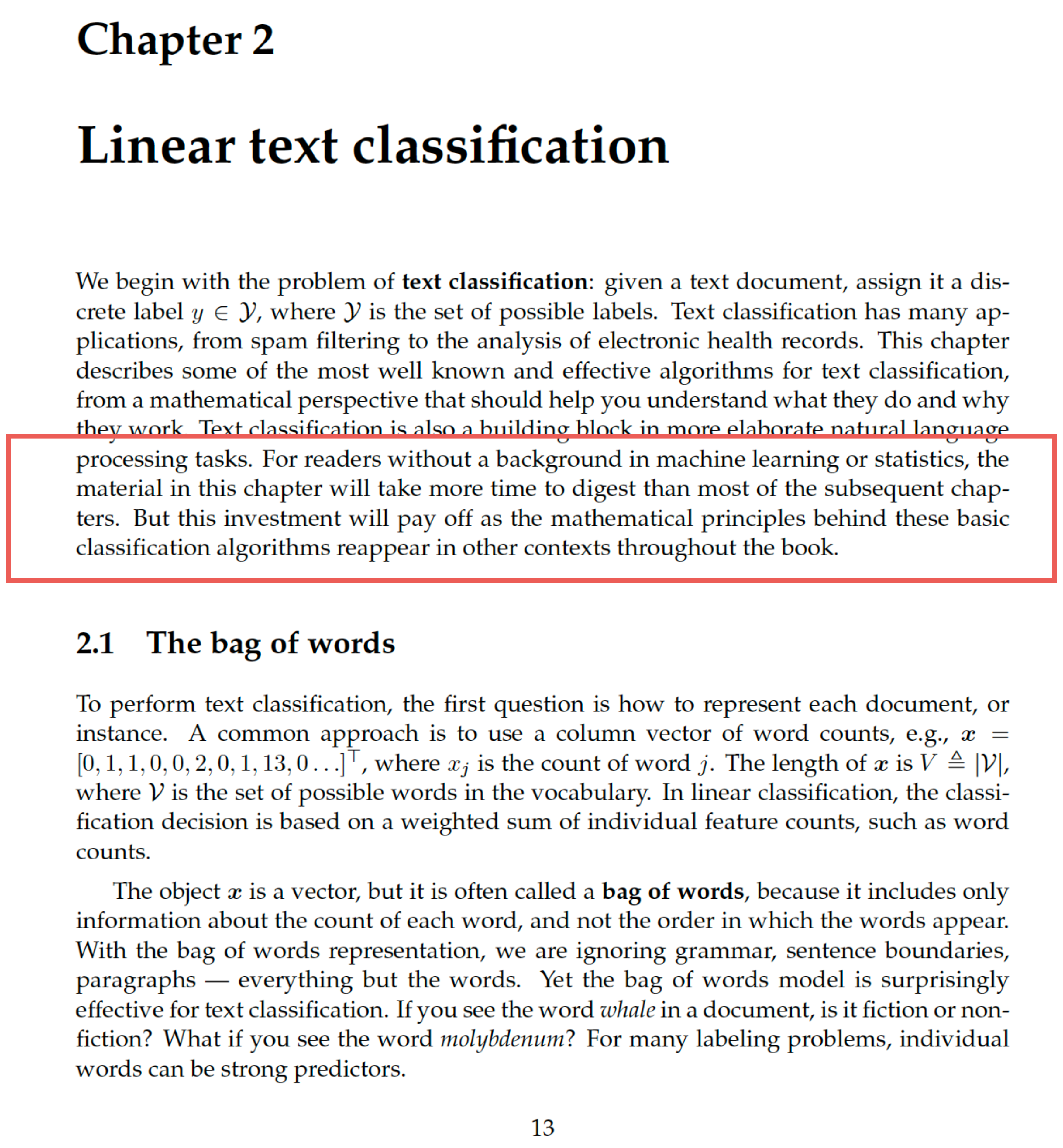
This and next Lecture

- 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

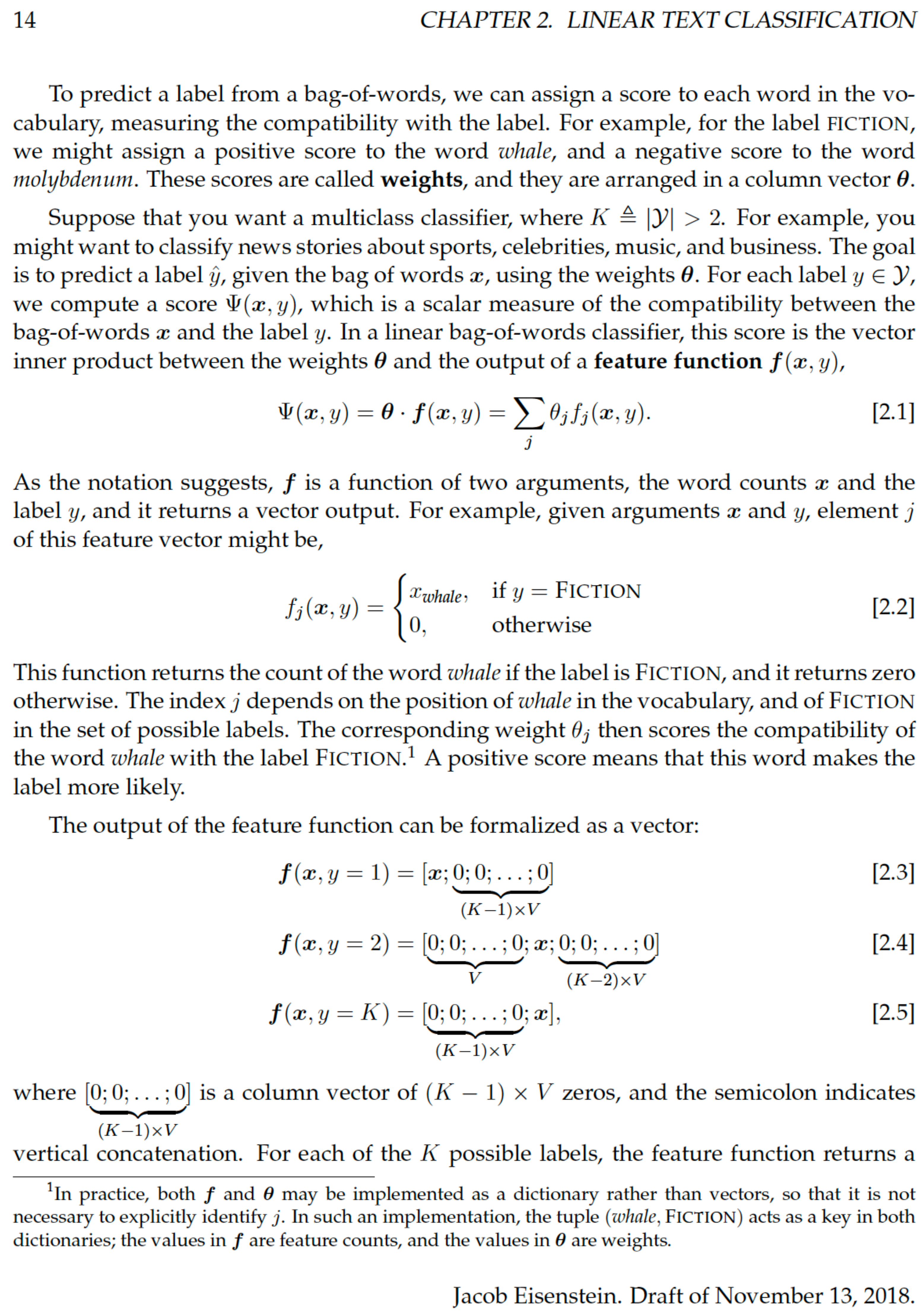
Linear text classification

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., $\boldsymbol{x} = [0, 1, 1, 0, 0, 2, 0, 1, 13, 0 \ldots]^\top$, 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:

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

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

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

where $\underbrace{[0; 0; \ldots; 0]}_{(K-1)\times V}$ 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

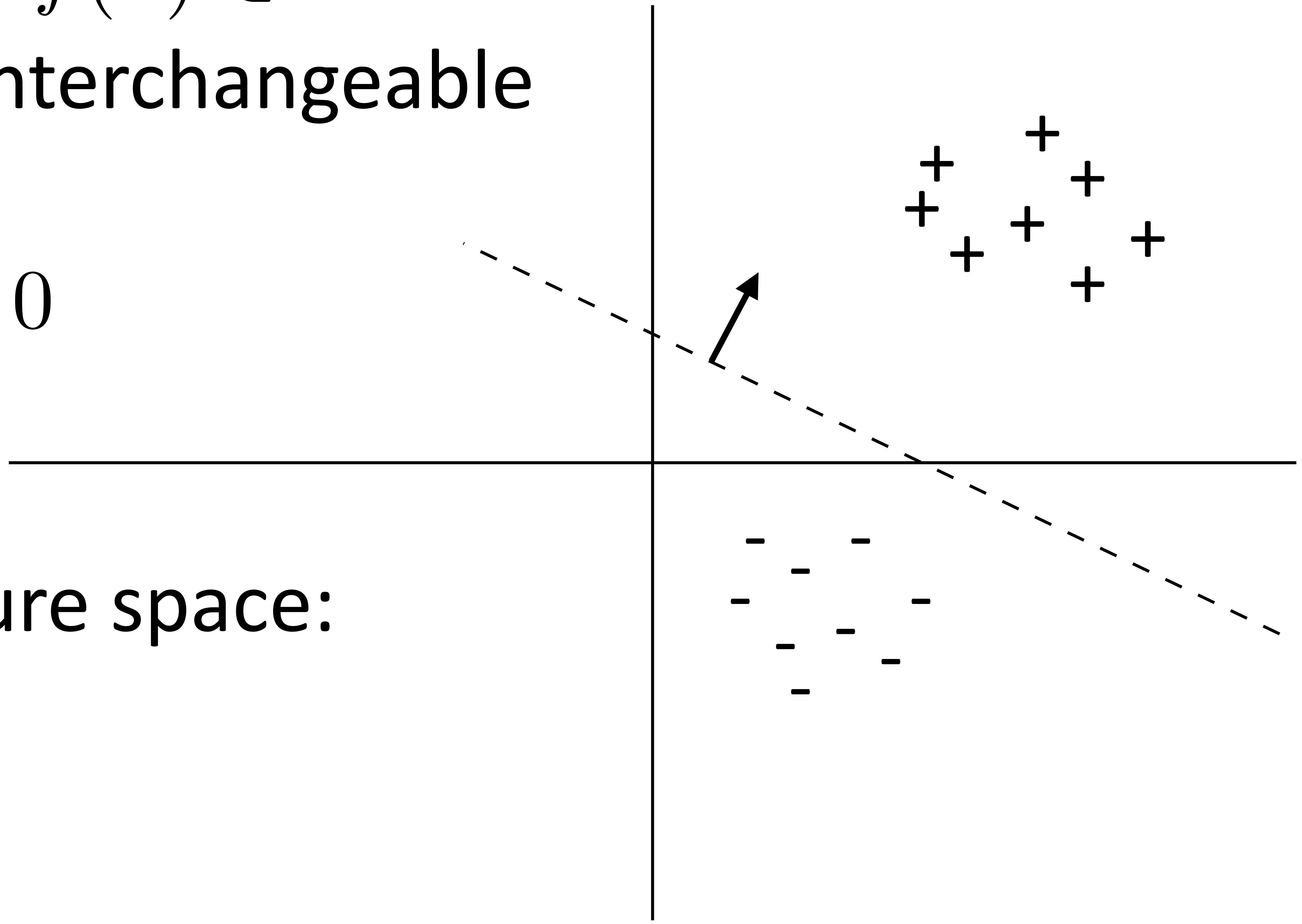
$$idf_i = \log_{10}(\frac{N}{df_i})$$

Total number of docs in collection

number of docs that have word 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: $w^\top f(x) + b > 0$
  $w^\top f(x) > 0$



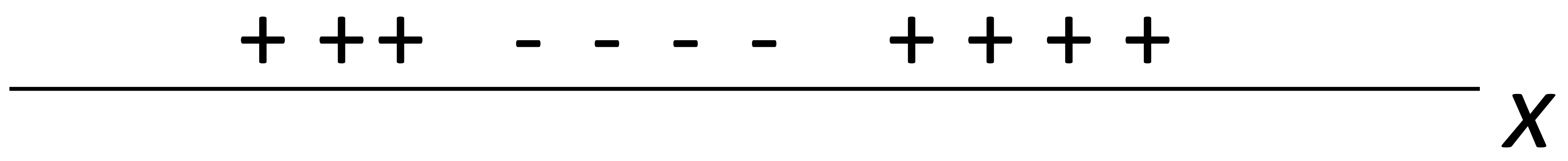
- Can delete bias if we augment feature space:

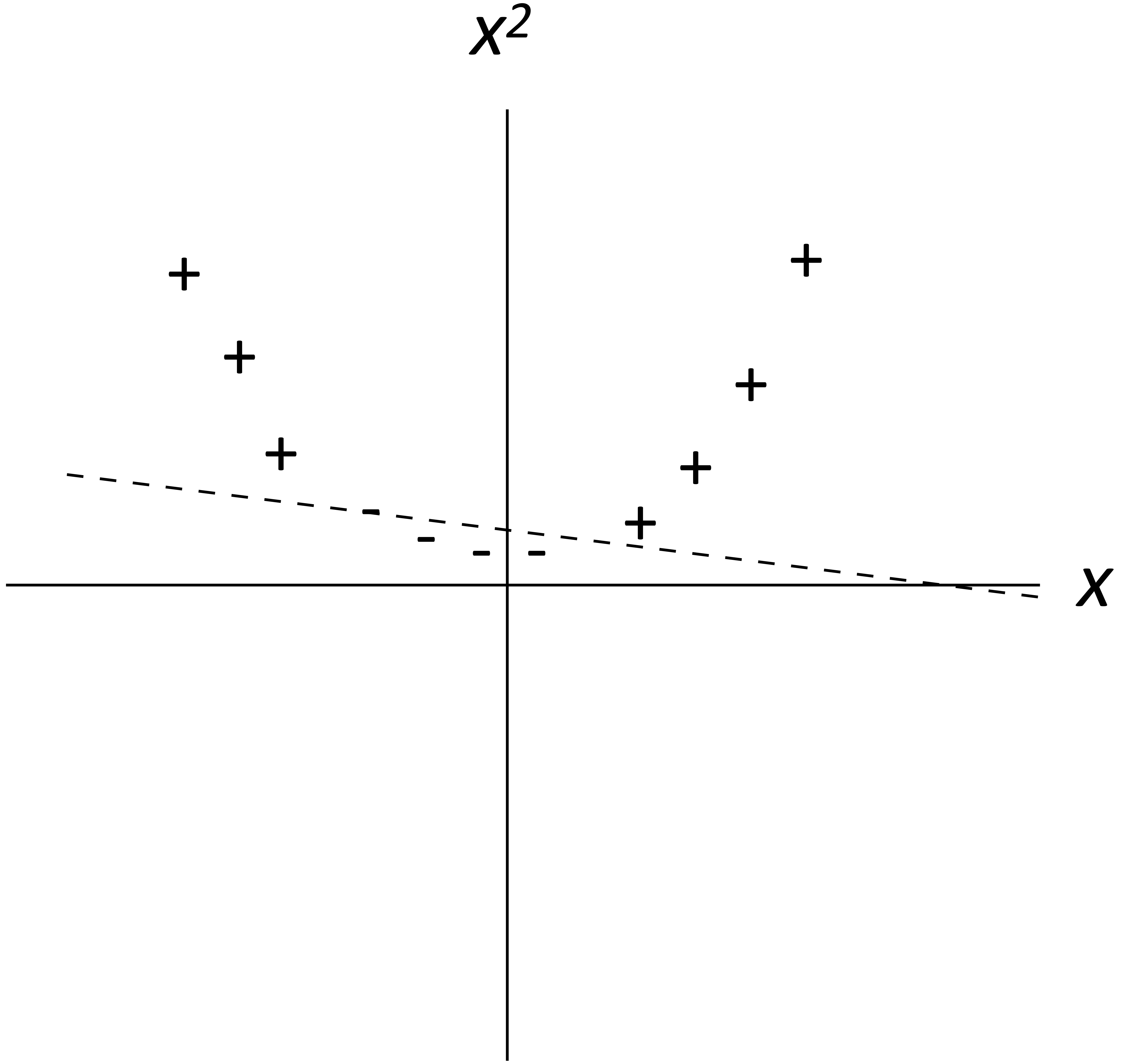
$f(x)$ = [0.5, 1.6, 0.3]

↓

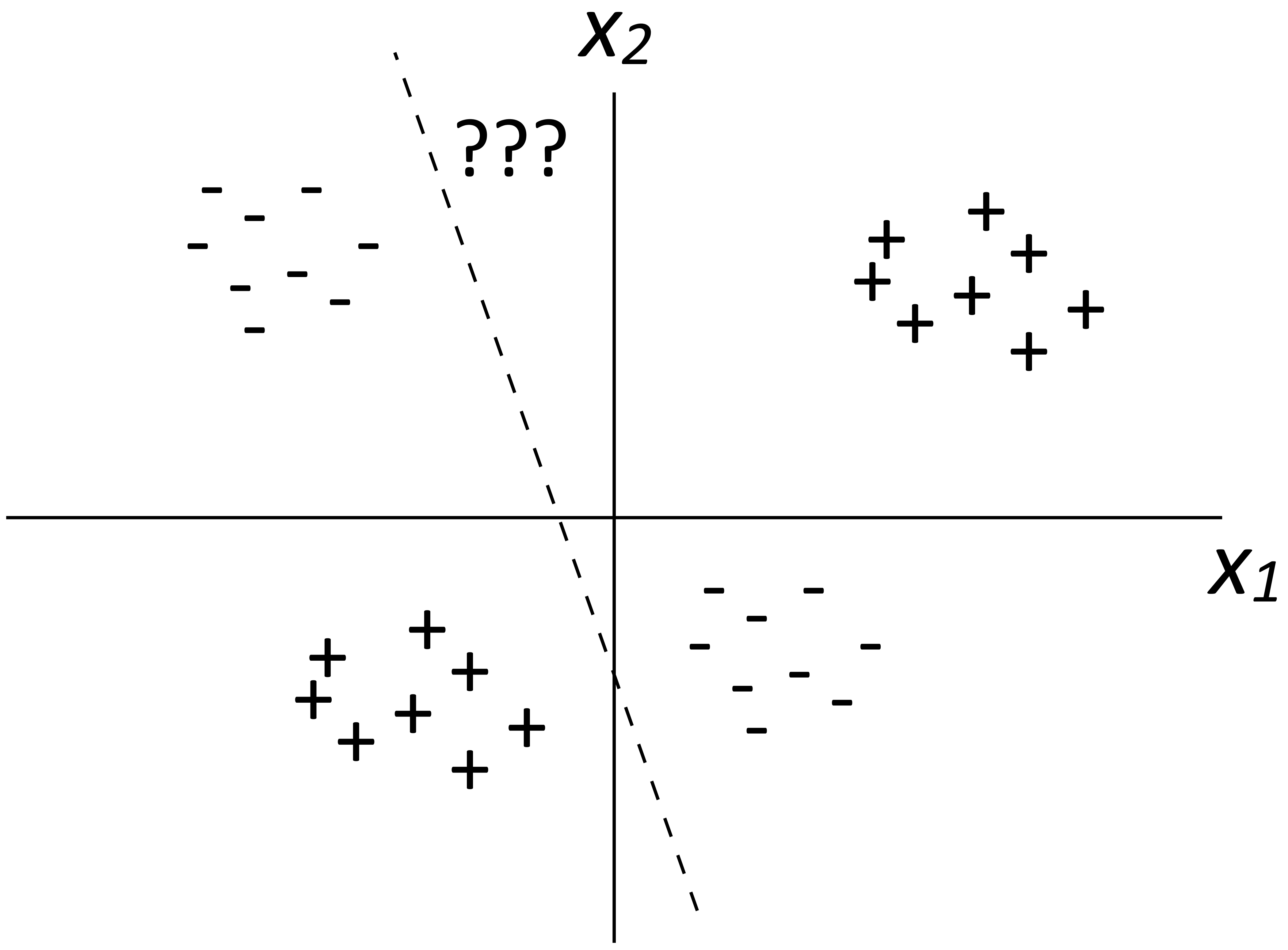
[0.5, 1.6, 0.3, **1**]

Linear functions are powerful!

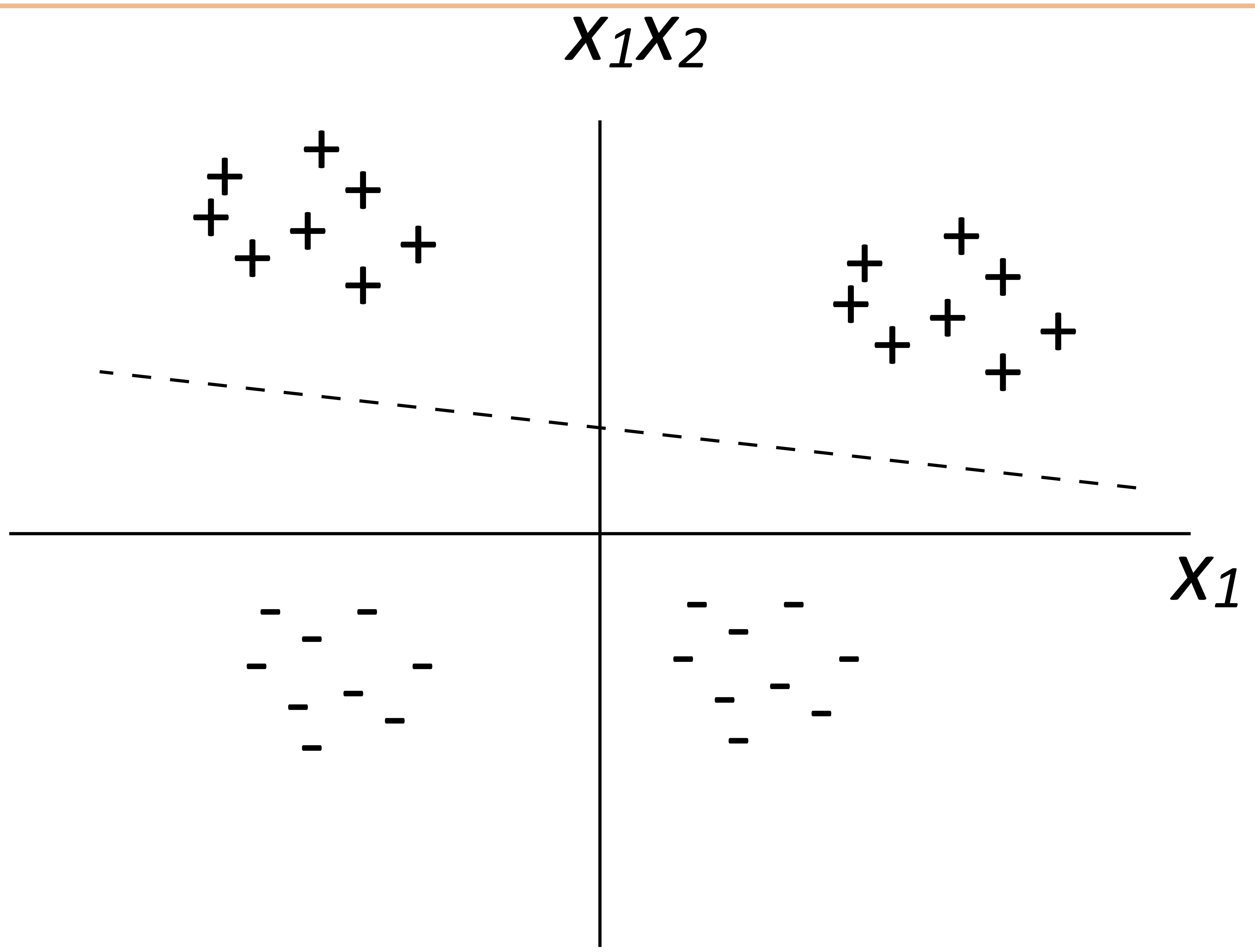




Linear functions are powerful!



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



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

- “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

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

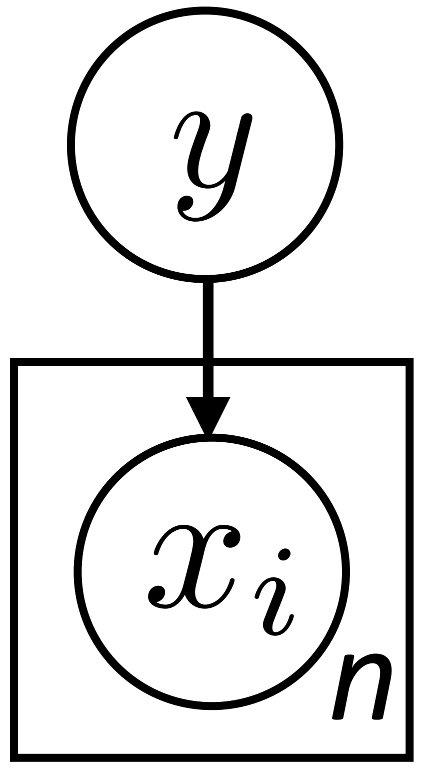
Bayes' Rule

constant: irrelevant for finding the max

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

"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?

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

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

- Problem — floating point underflow

S	exponent	significand
1	11 bits	52 bits

Largest = 1. 1 1 1… x 2^{+1023}

Smallest = 1.000… X 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:

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

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

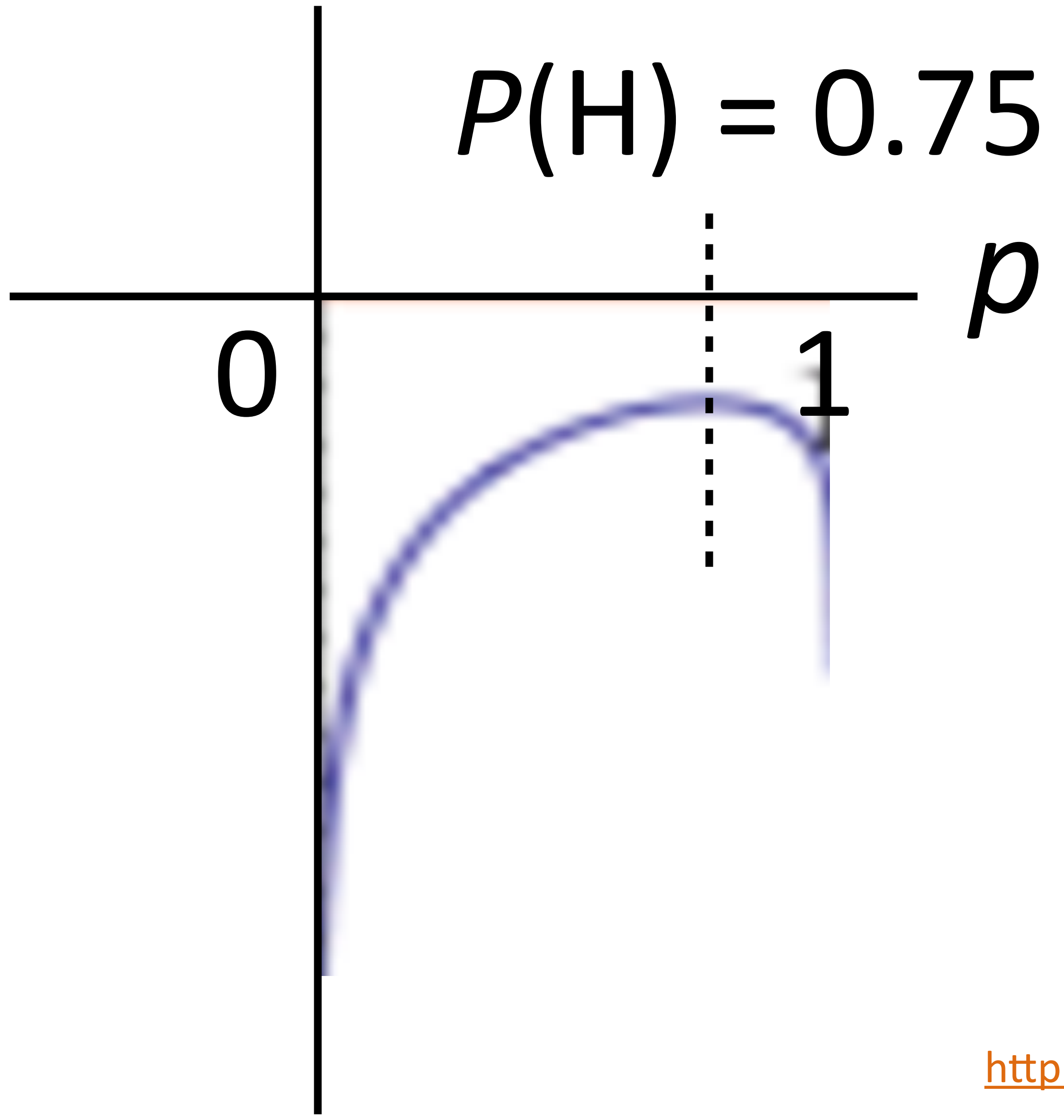
- Imagine a coin flip which is heads with probability *p*

- 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

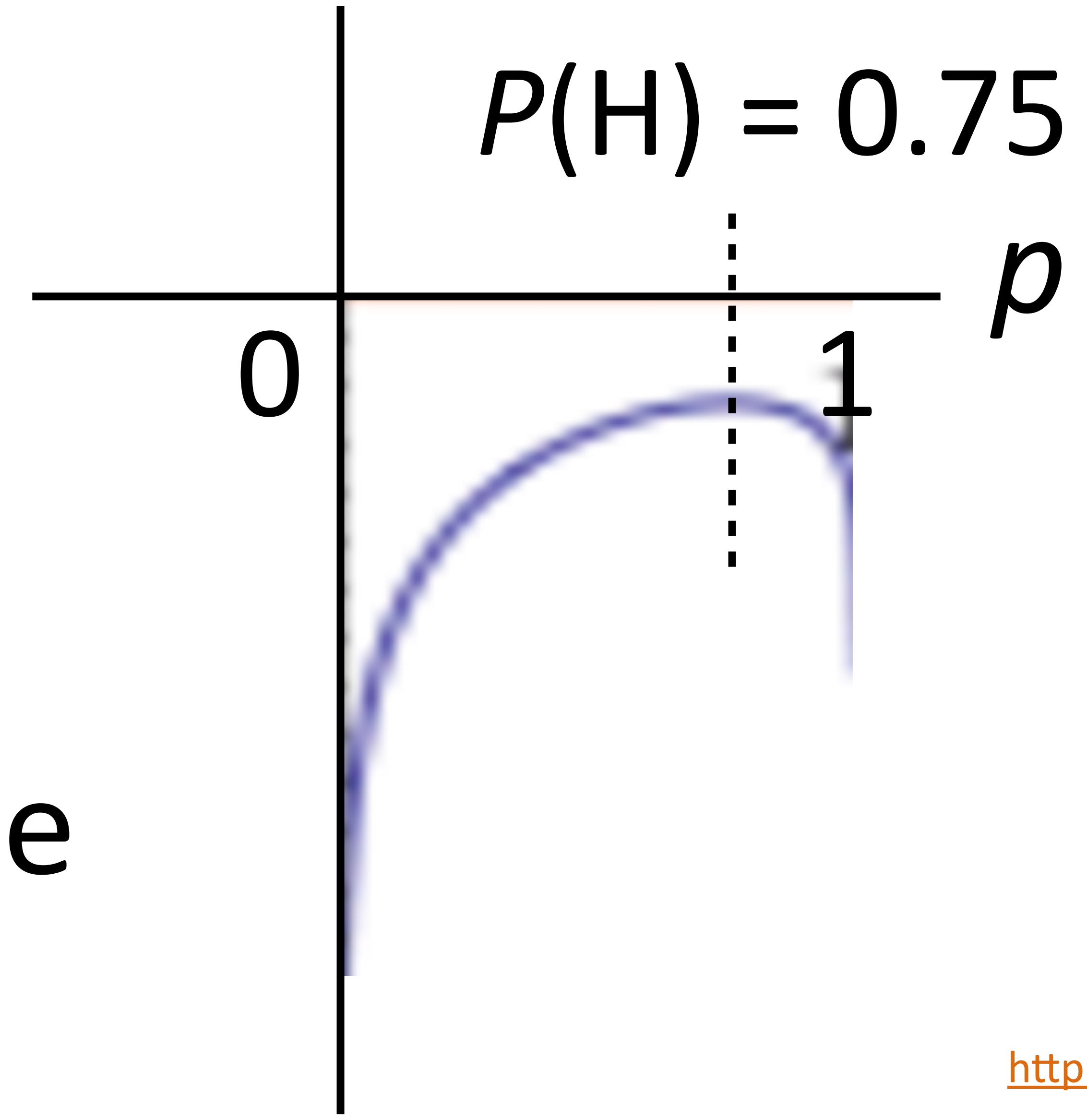
- Imagine a coin flip which is heads with probability *p*

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

- Easier: maximize *log* likelihood

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

log likelihood



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

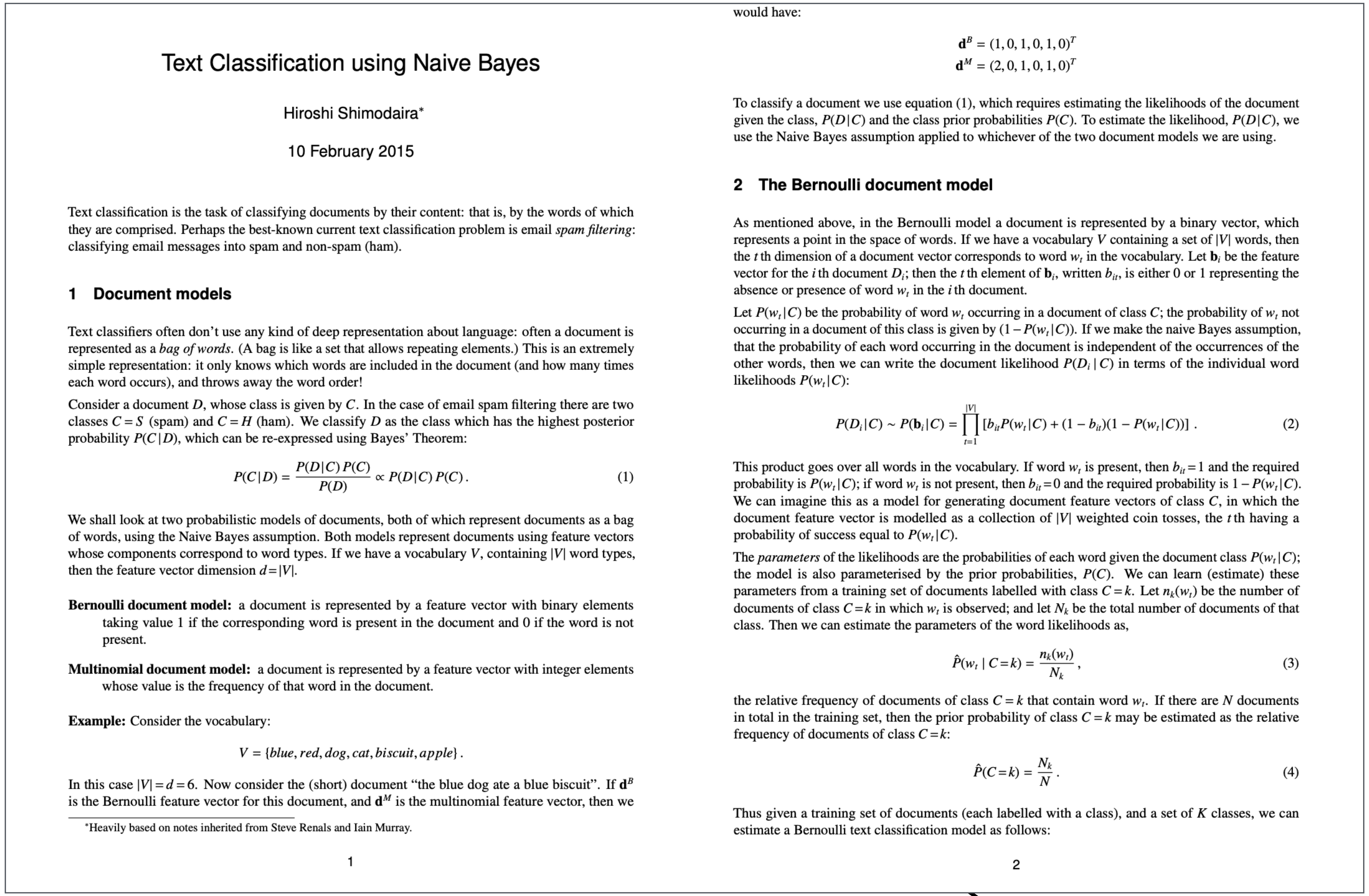
Naive Bayes

- 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:

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

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:

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

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?

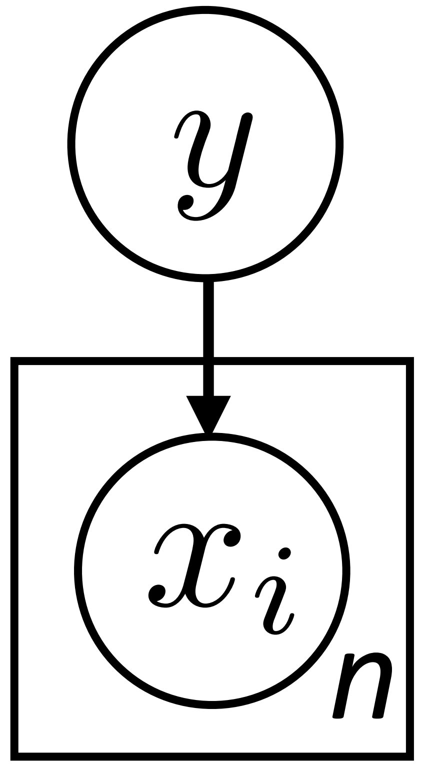
$$P(y|x) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

- 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 —

$$
\begin{array}{ll}
P(x_{\text{beautiful}}|+) = 0.1 & P(x_{\text{beautiful}}|-) = 0.01 \\
P(x_{\text{stunning}}|+) = 0.1 & P(x_{\text{stunning}}|-) = 0.01 \\
P(x_{\text{gorgeous}}|+) = 0.1 & P(x_{\text{gorgeous}}|-) = 0.01 \\
P(x_{\text{boring}}|+) = 0.01 & P(x_{\text{boring}}|-) = 0.1
\end{array}
$$

- 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, …)

QA Time

DO YOU HAVE ANY QUESTIONS?