SFU NatLangLab

CMPT 413/825: Natural Language Processing

Text Classification

Fall 2020
2020-09-18

Adapted from slides from Anoop Sarkar, Danqi Chen and Karthik Narasimhan

Announcements

- Remaining lectures on language modeling (LM) on Canvas

  CMPT413 D100 / CMPT825 G100 > Files > Lectures > Videos

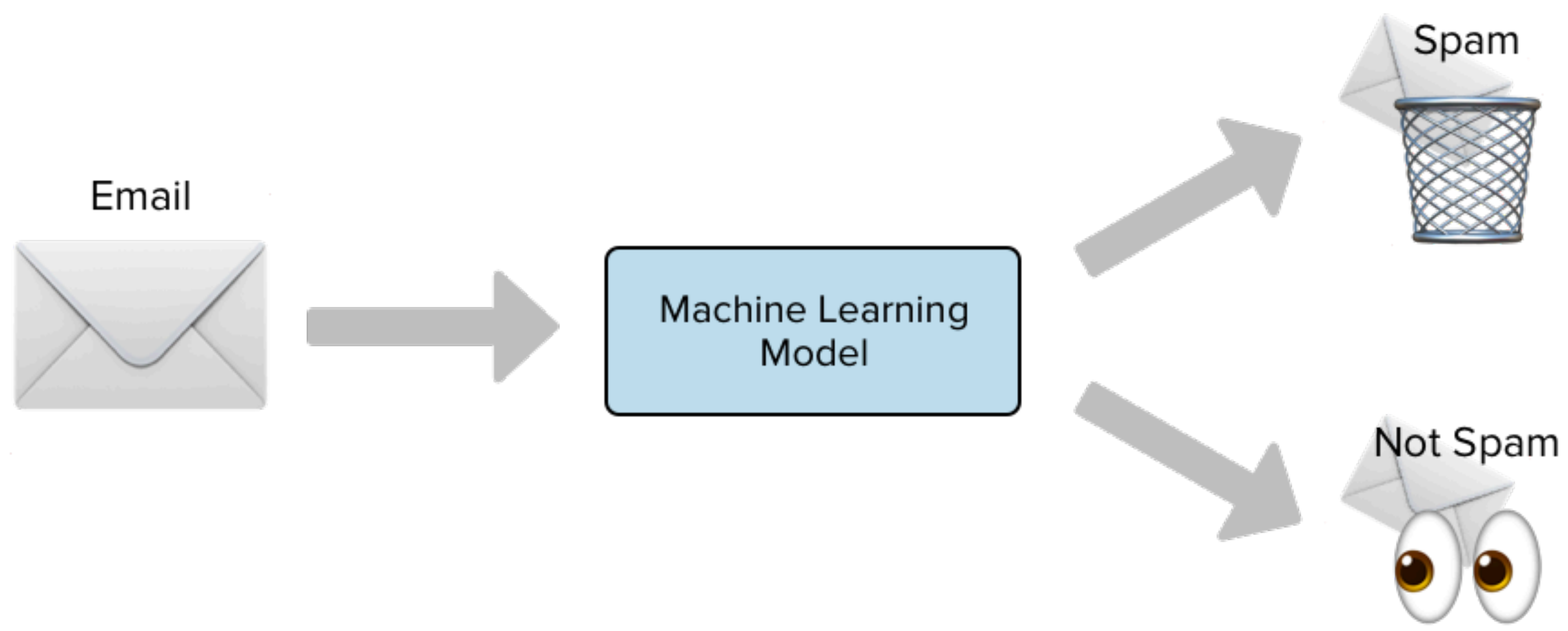
- Initial grades for HW0-Programming section released

  - You have until 11:59 Friday to resubmit / address any comments in the feedback

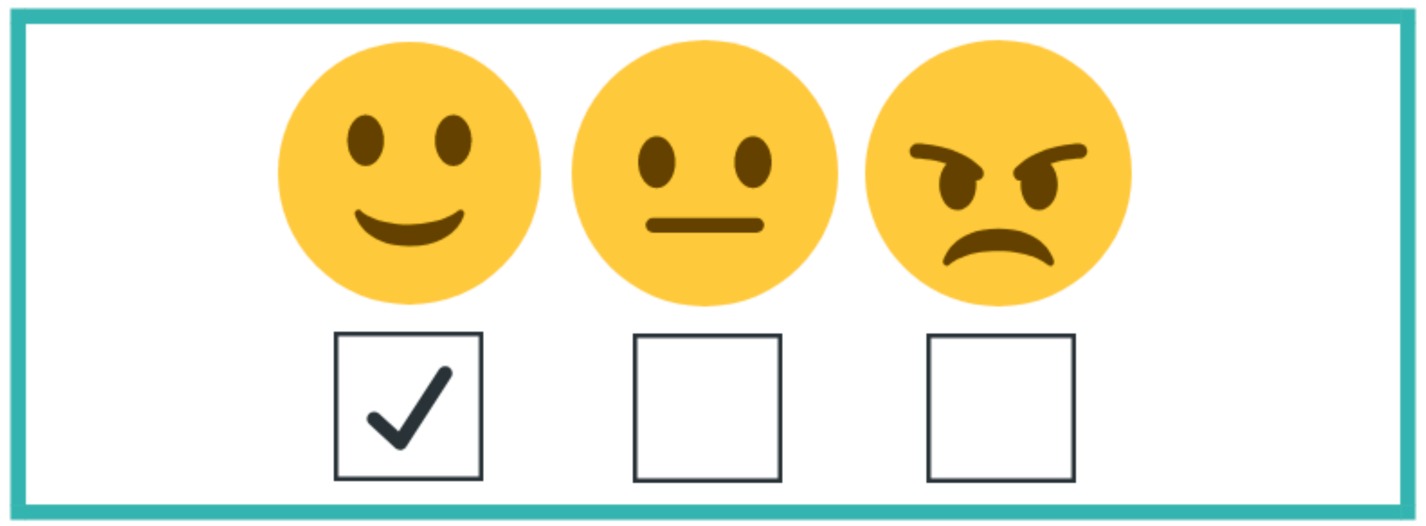
- We will aim to have final grades for HW0 out next week

- For those that do not have group / single student groups, we have created a Piazza group through which you can contact each other.

Why classify?



Spam detection



Sentiment analysis

- Authorship attribution
- Language detection
- News categorization

- neg unbelievably disappointing
- pos Full of zany characters and richly applied satire, and some great plot twists
- pos this is the greatest screwball comedy ever filmed
- neg It was pathetic. The worst part about it was the boxing scenes.

Other Examples

Intent detection

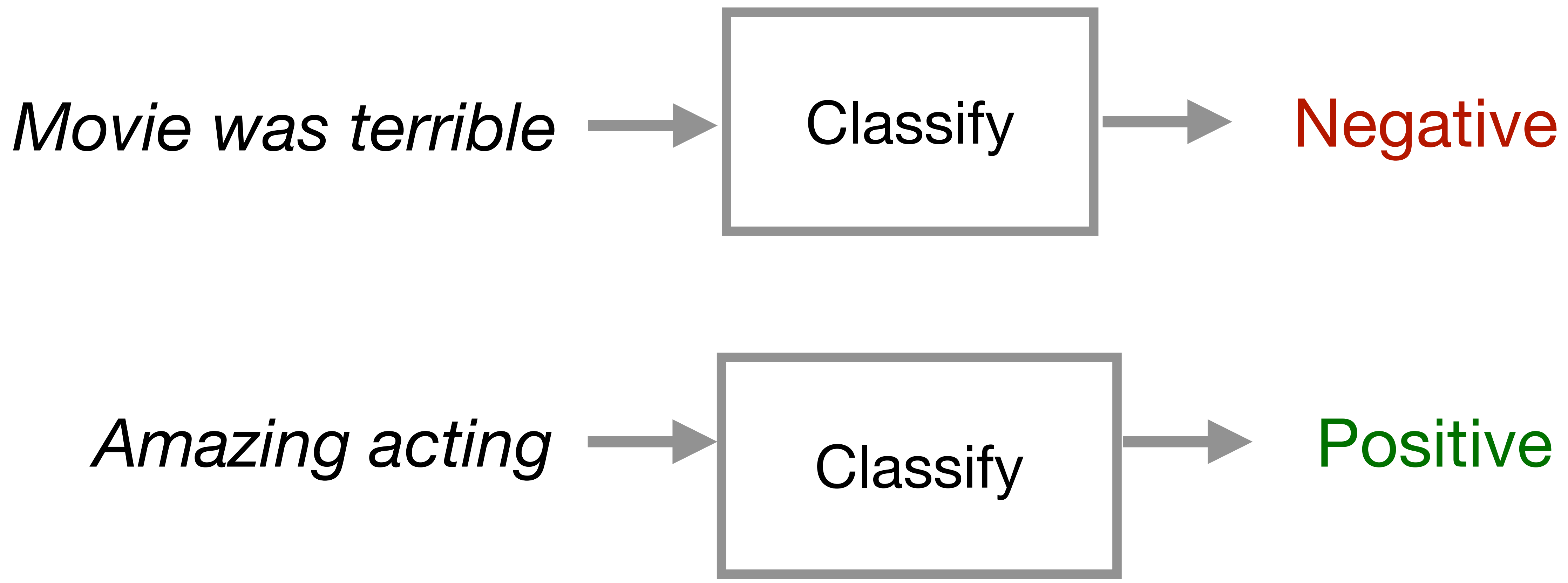
- ADDR_CHANGE I just moved and want to change my address.
- ADDR_CHANGE Please help me update my address.
- FILE_CLAIM I just got into a terrible accident and I want to file a claim.
- CLOSE_ACCOUNT I'm moving and I want to disconnect my service.

Prepositional phrase attachment

- noun attach: *I bought* **the shirt with pockets**
- verb attach: **I bought** *the shirt* **with my credit card**
- noun attach: *I washed* **the shirt with mud**
- verb attach: **I washed** *the shirt* **with soap**

Classification: The Task

- Inputs:
  - A document **d**
  - A set of classes $\mathbf{C = \{c_1, c_2, c_3, \ldots, c_m\}}$
- Output:
  - Predicted class c for document d



Rule-based classification

- Combinations of features on words in document, meta-data

  IF there exists word w in document d such that w in [good, great, extra-ordinary, …],
  THEN output Positive

  IF email address ends in [*ithelpdesk.com*, *makemoney.com*, *spinthewheel.com*, …]
  THEN output SPAM

- Simple, can be very accurate
- But: rules may be hard to define (and some even unknown to us!)
  - Expensive
  - Not easily generalizable

Supervised Learning: Let's use statistics!

- Data-driven approach

  Let the machine figure out the best patterns to use!

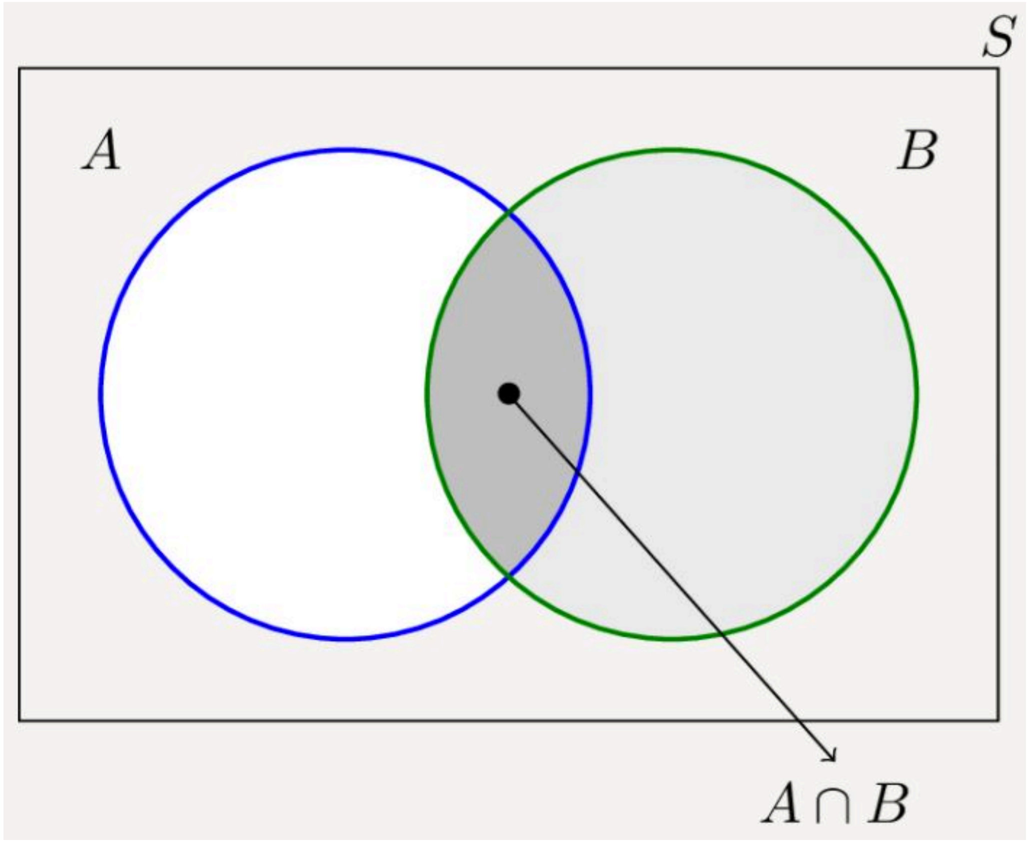
- Inputs:
  - Set of *m* classes $C = \{c_1, c_2, \ldots, c_m\}$
  - Set of *n* 'labeled' documents: $\{(d_1, c_1), (d_2, c_2), \ldots, (d_n, c_n)\}$
- Output: Trained classifier, $F : d \rightarrow c$
  - What form should F take?
  - How to learn F?

Recall: general guidelines for model building

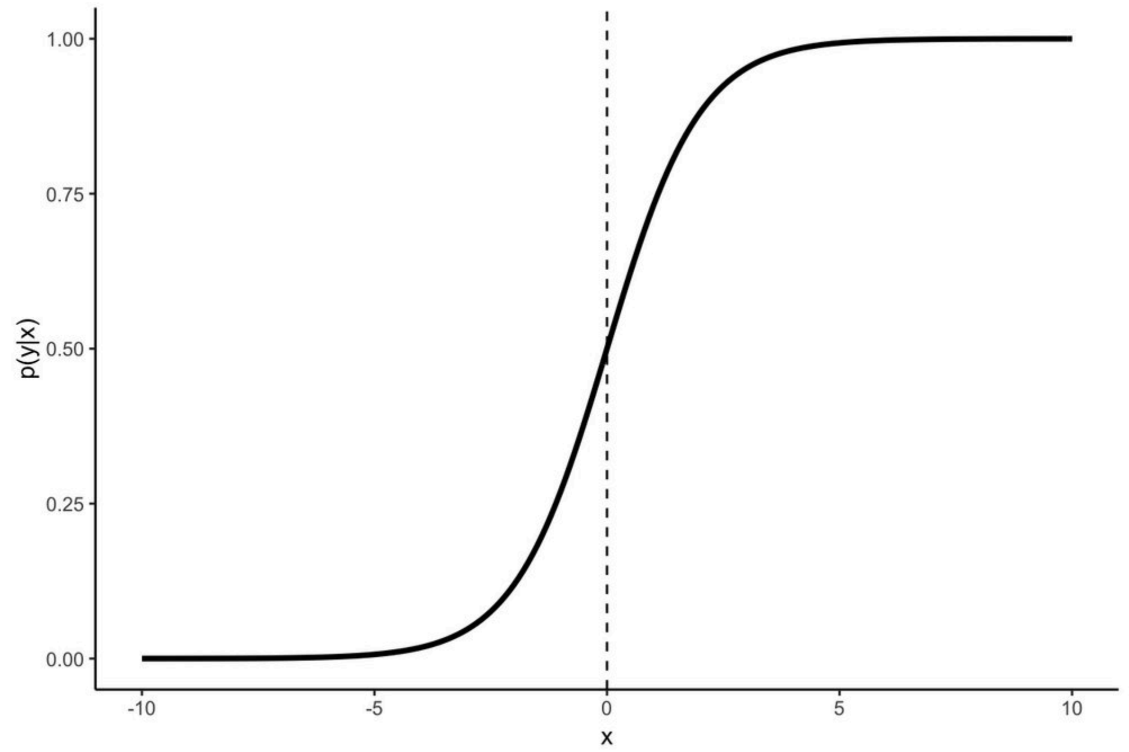
Two steps to building a probability model:

1. Define the model
   - What form should F take?
   - What independence assumptions do we make?
   - What are the model parameters (probability values)?
2. Estimate the model parameters (training/learning)
   - How to learn F?

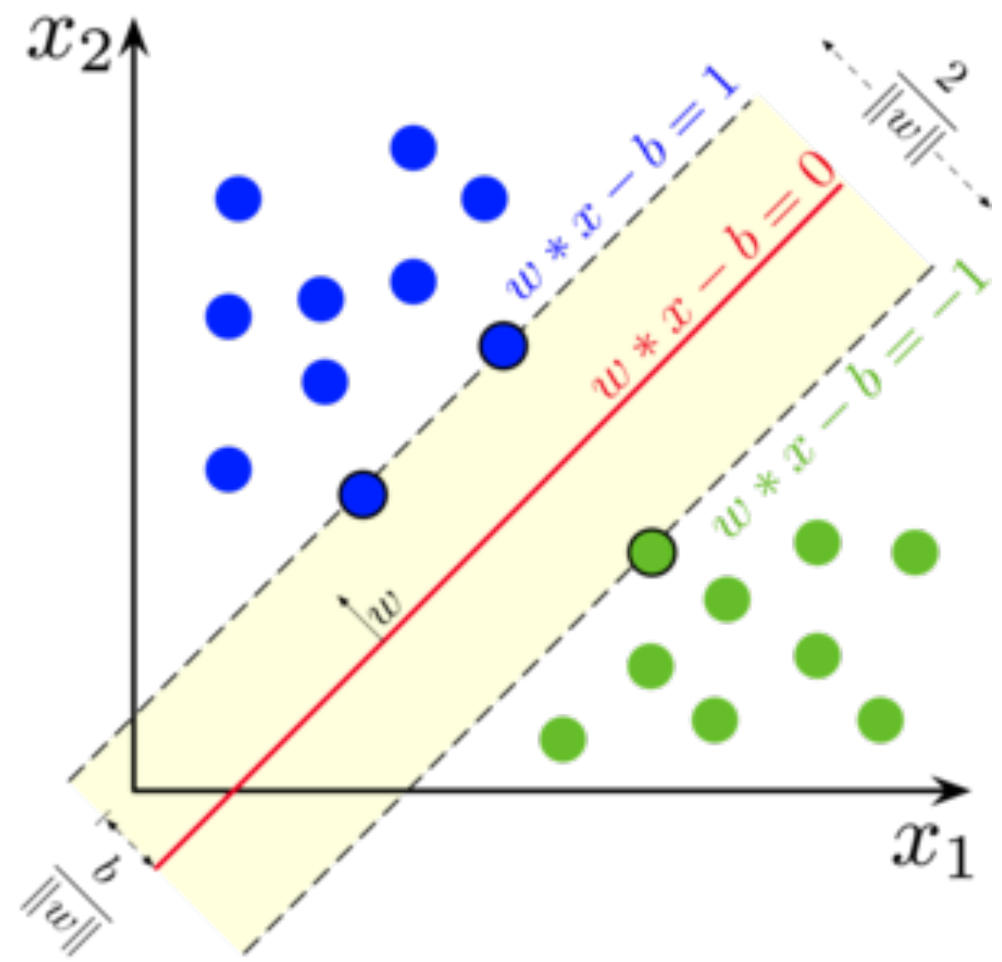
Types of supervised classifiers



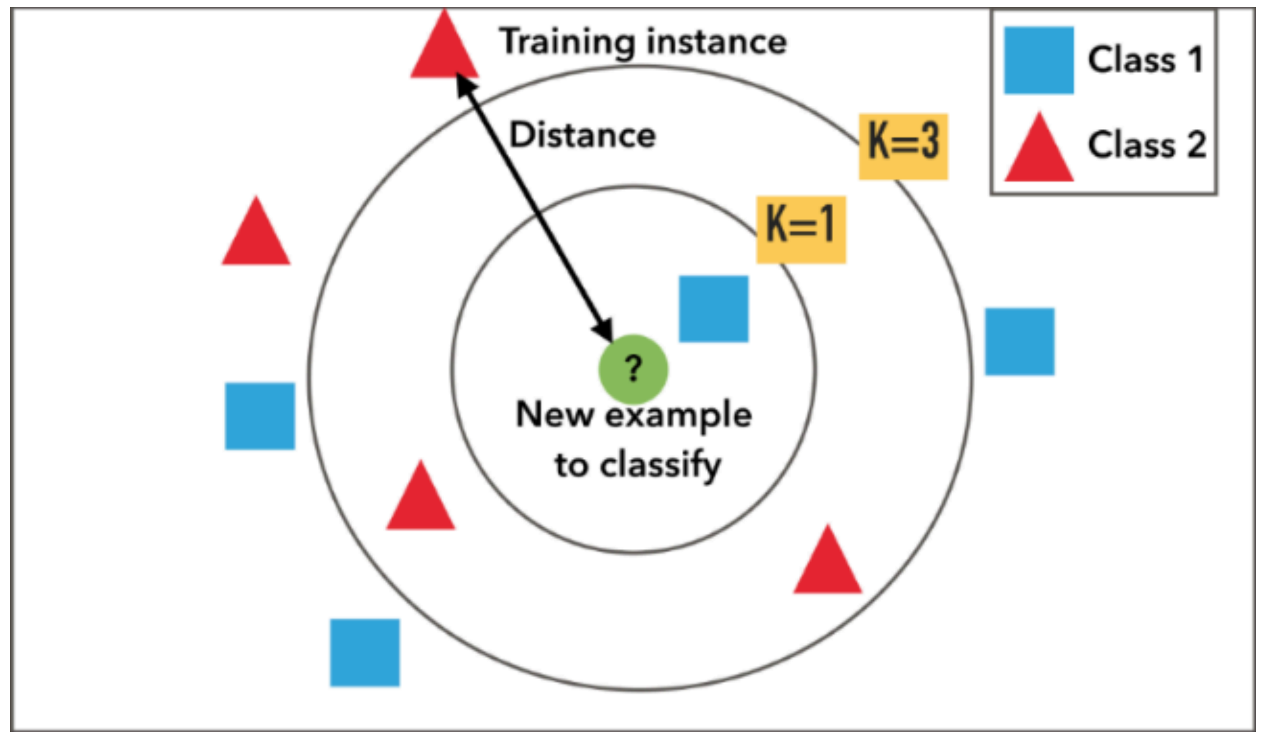
Naive Bayes



Logistic regression



Support vector machines



k-nearest neighbors

Naive Bayes Classifier
General setting

- Let the input x be represented as r features: f_j, $1 \leq j \leq r$

- Let y be the output classification

- We can have a simple classification model using Bayes rule

$$\underbrace{P(y \mid x)}_{\text{Posterior}} = \frac{\overbrace{P(y)}^{\text{Prior}} \cdot \overbrace{P(x \mid y)}^{\text{Likelihood}}}{P(x)}$$

- Make strong (naive) conditional independence assumptions

$$P(x \mid y) = \prod_{j=1}^{r} P(f_j \mid y) \xrightarrow{\text{Bayes rule}} P(y \mid x) \propto P(y) \cdot \prod_{j=1}^{r} P(f_j \mid y)$$

Naive Bayes classifier for text classification

- For text classification: input x is document $d = (w_1, \ldots, w_k)$
- Use as our features the words w_j, $1 \leq j \leq |V|$ where V is our vocabulary
- c is the output classification
- Predicting the best class:

$$c_{MAP} = \arg\max_{c \in C} P(c \mid d)$$

maximum a posteriori (MAP) estimate

$$= \arg\max_{c \in C} \frac{P(c)P(d \mid c)}{P(d)}$$

$$= \arg\max_{c \in C} P(c)P(d \mid c)$$

$P(d \mid c) \rightarrow$ Conditional probability of generating document d from class c

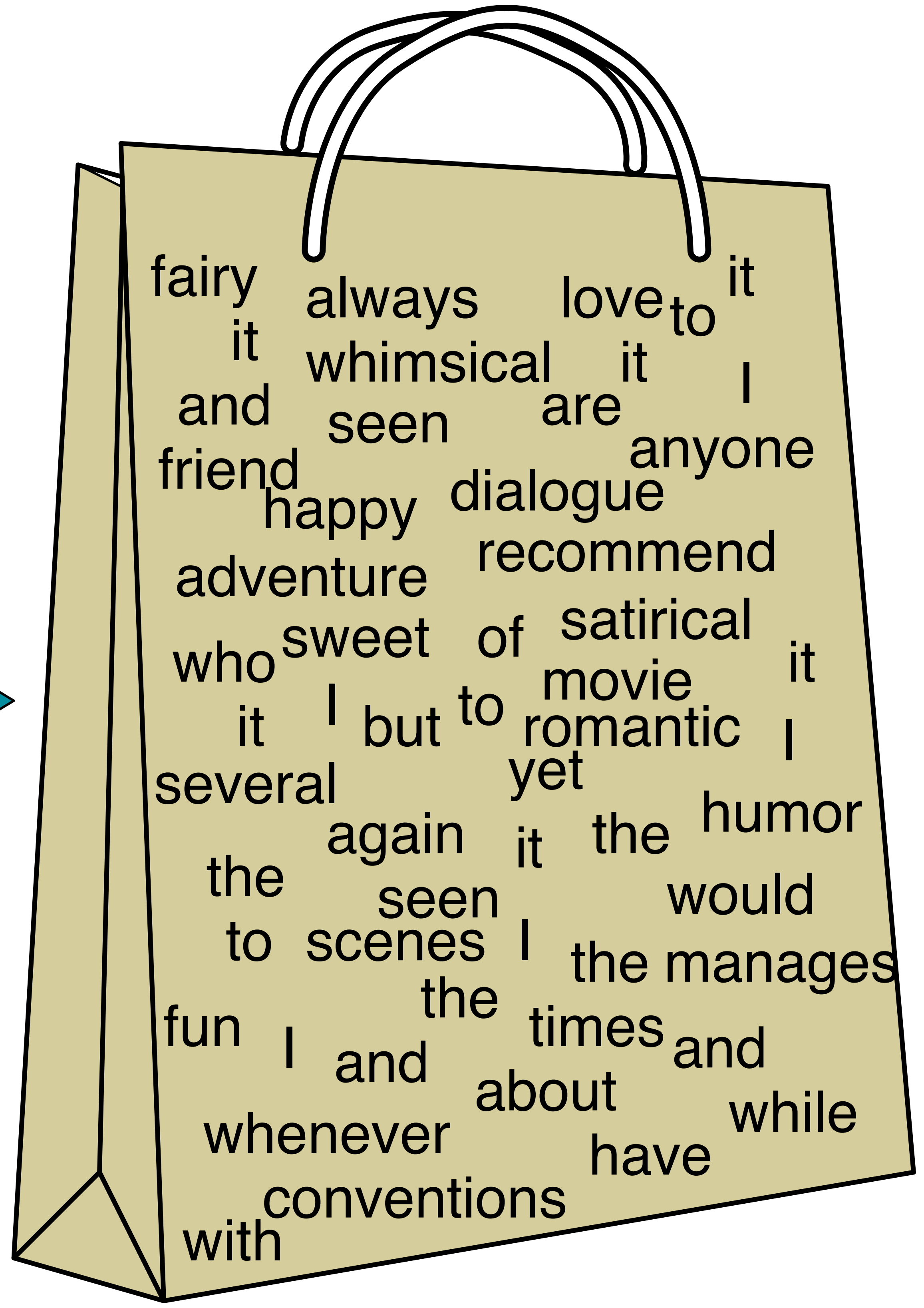
$P(c) \rightarrow$ Prior probability of class c

How to represent P(d | c)?

- Option 1: represent the entire sequence of words
  - $P(w_1, w_2, w_3, \ldots, w_k | c)$ *(too many sequences!)*
- Option 2: Bag of words
  - Assume position of each word is irrelevant
    (both absolute and relative)
  - $P(w_1, w_2, w_3, \ldots, w_k | c) = P(w_1 | c)P(w_2 | c)\ldots P(w_k | c)$
  - Probability of each word is *conditionally independent*
    given class c

Bag of words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



word	count
it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Predicting with Naive Bayes

- Once we assume that the position of each word is irrelevant and that the words are conditionally independent given class c, we have:

$$P(d \mid c) = P(w_1, w_2, w_3, \ldots, w_k \mid c) = P(w_1 \mid c)P(w_2 \mid c)\ldots P(w_k \mid c)$$

- The maximum a posteriori (MAP) estimate is now:

$\hat{P}$ is used to indicate the estimated probability

$$c_{\text{MAP}} = \arg\max_{c \in C} P(c)P(d \mid c) = \arg\max_{c \in C} \hat{P}(c) \prod_{i=1}^{k} \hat{P}(w_i \mid c)$$

Note that k is the number of tokens (words) in the document.

The index i is the position of the token.

Naive Bayes as a generative model

$$P(c) \xrightarrow{\text{Sample}} c_1 \longrightarrow P(d|c) \xrightarrow{\text{Sample}} d_1$$

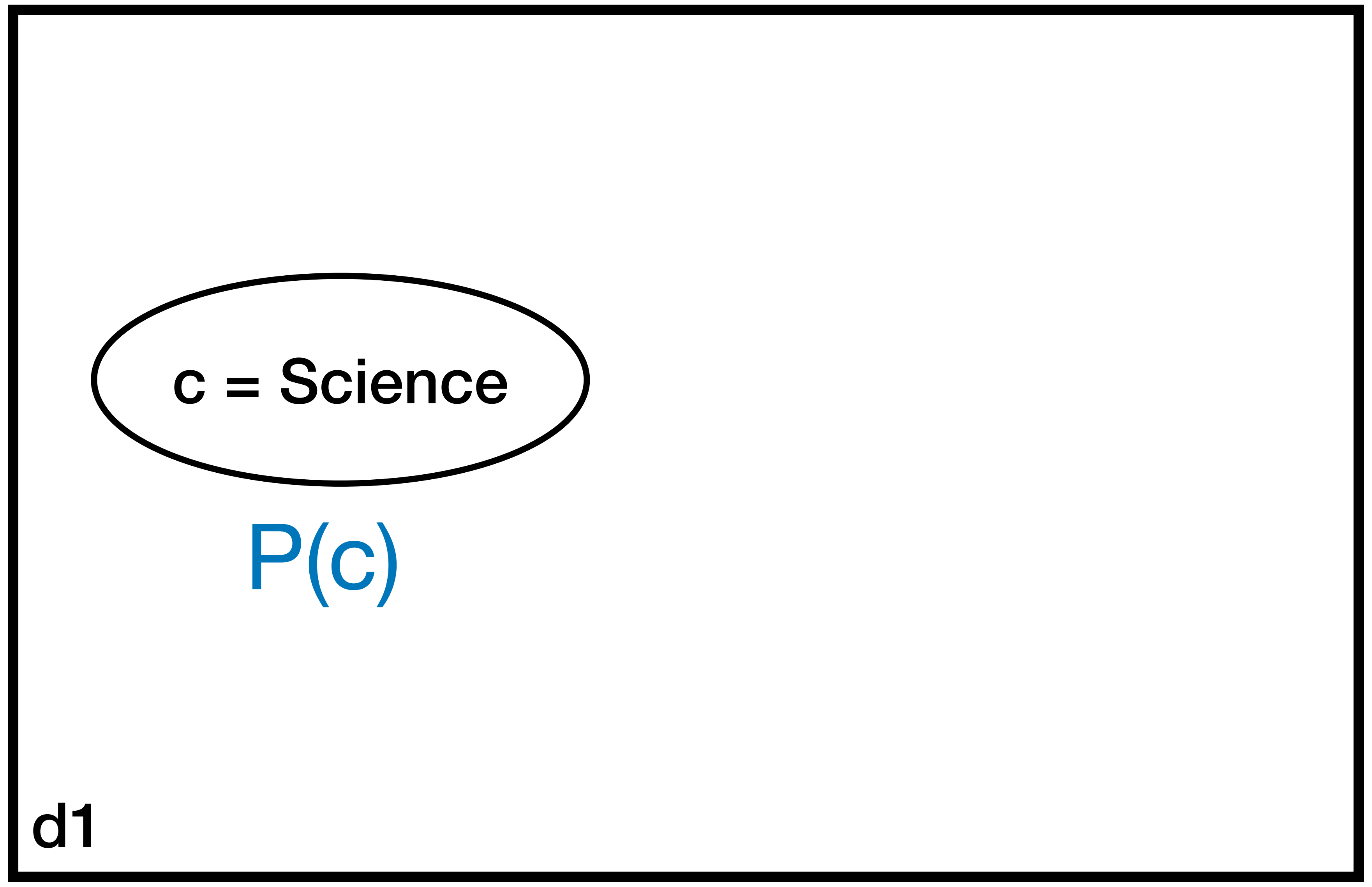
$$P(c) \xrightarrow{\text{Sample}} c_2 \longrightarrow P(d|c) \xrightarrow{\text{Sample}} d_2$$

$$\vdots$$

$$P(c) \xrightarrow{\text{Sample}} c_n \longrightarrow P(d|c) \xrightarrow{\text{Sample}} d_n$$

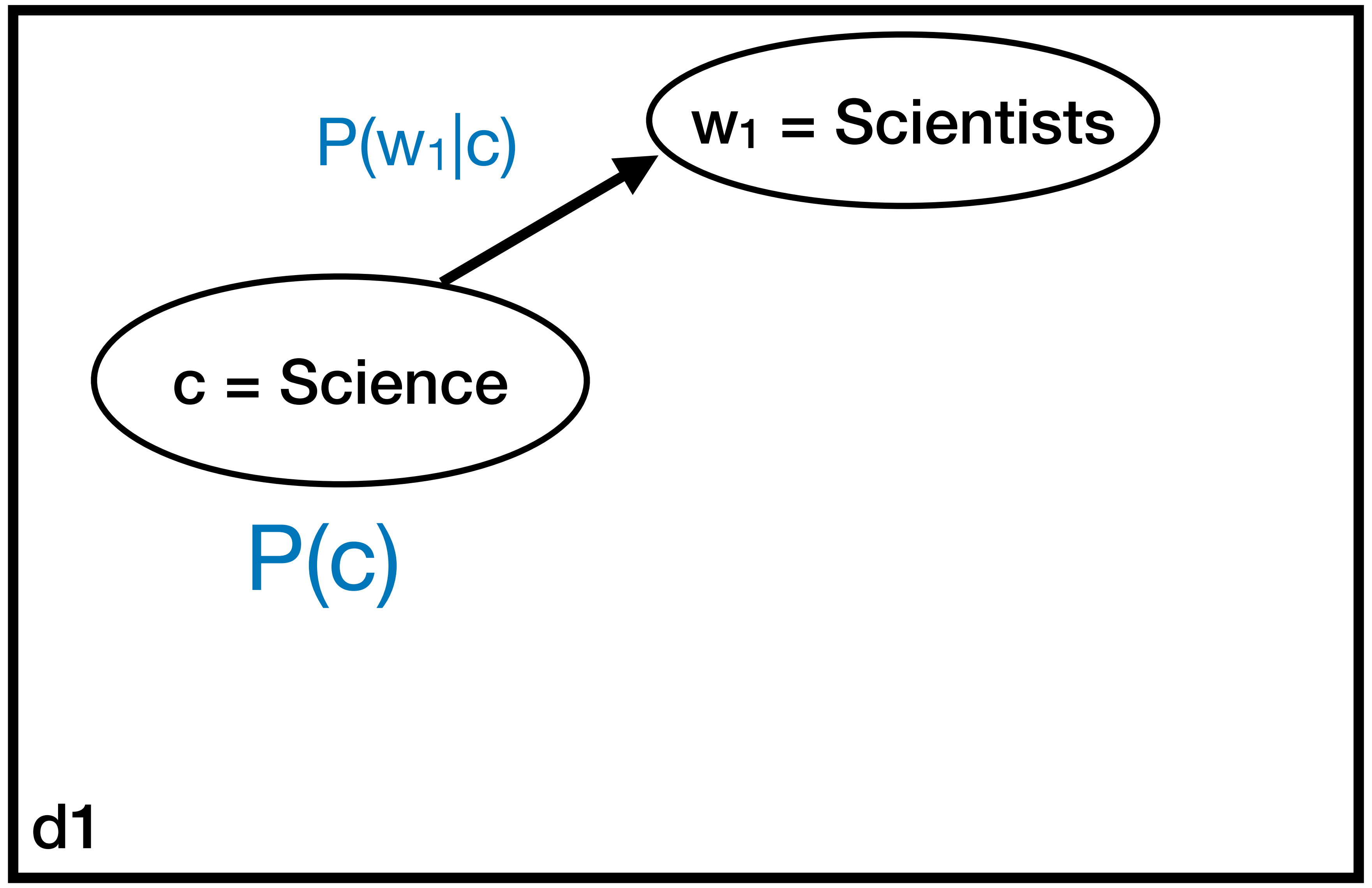
Generate the entire data set one document at a time

Naive Bayes as a generative model



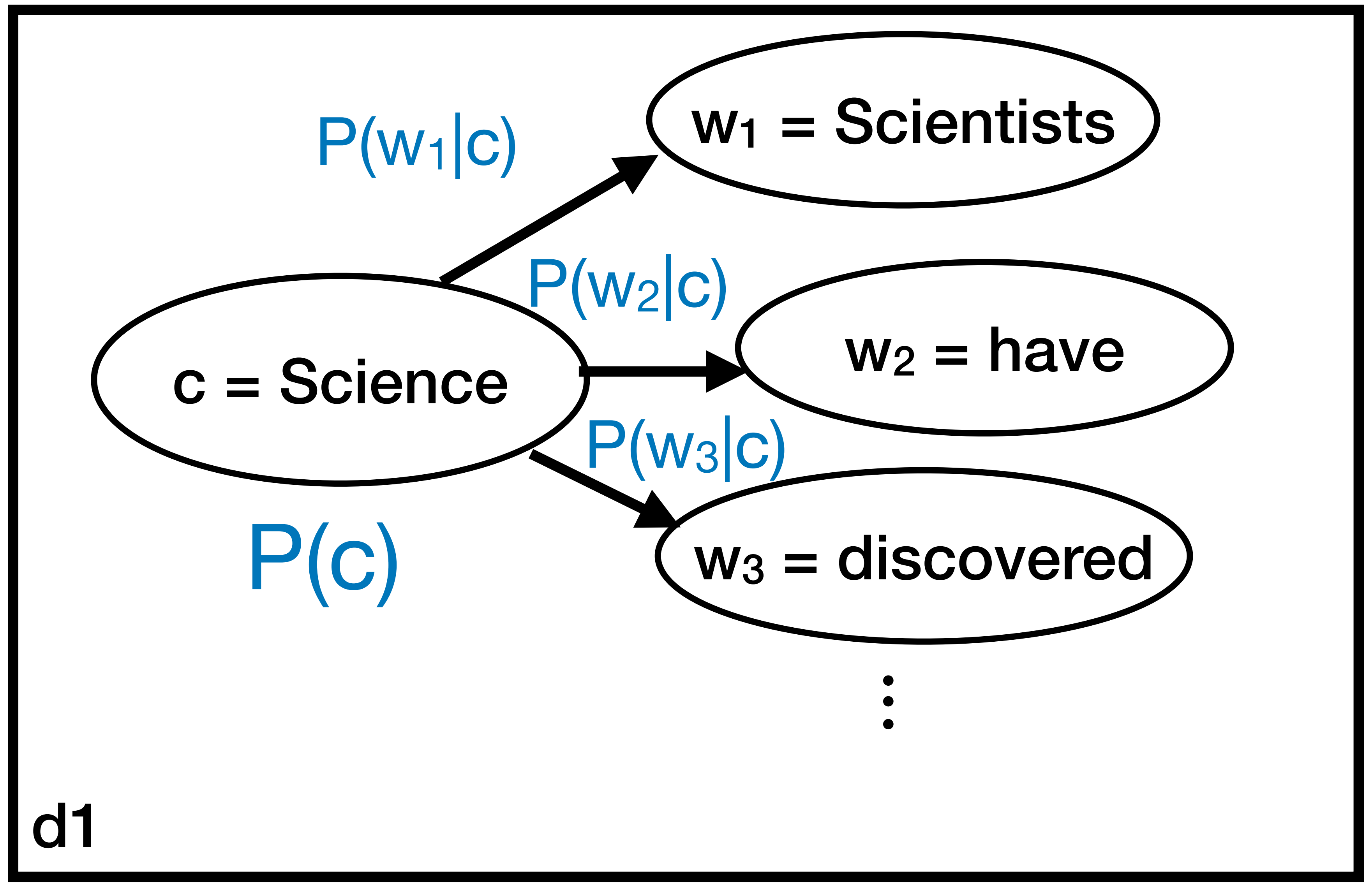
Sample a category

Naive Bayes as a generative model



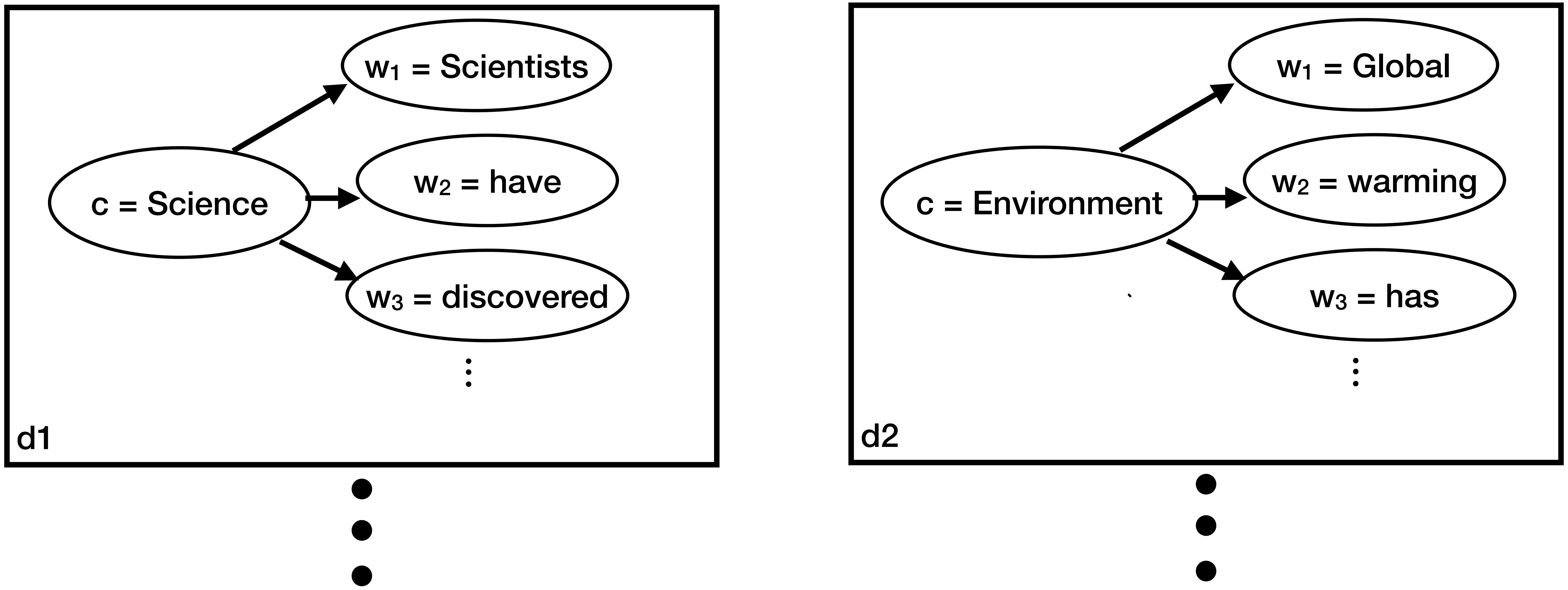
Sample words

Naive Bayes as a generative model



Generate the entire data set one document at a time

Naive Bayes as a generative model



Generate the entire data set one document at a time

Estimating probabilities

Maximum likelihood estimate

$$\hat{P}(c_j) = \frac{\text{Count}(c_j)}{n}$$

$$\hat{P}(w_i|c_j) = \frac{\text{Count}(w_i, c_j)}{\sum_{w \in V}[\text{Count}(w, c_j)]}$$

Data sparsity

- What about when count('amazing', *positive*) = *0?*
  - Implies P('amazing' | *positive*) = 0
- Given a review document, d = "…. most amazing movie ever …"

$$c_{MAP} = \arg\max_{c \in C} \hat{p}(c) \prod_{i=1}^{k} P(w_i \mid c)$$

$$= \arg\max_{c \in C} \hat{p}(c) \cdot 0 = \arg\max_{c \in C} 0$$

Can't determine the best c!

Solution: Smoothing!

Maximum likelihood estimate

$$\hat{P}(c_j) = \frac{\text{Count}(c_j)}{n}$$

$$\hat{P}(w_i|c_j) = \frac{\text{Count}(w_i, c_j)}{\sum_{w \in V}[\text{Count}(w, c_j)]}$$

Smoothing

$$\hat{P}(w_i|c) = \frac{\text{Count}(w_i, c) + \alpha}{\sum_{w \in V}[\text{Count}(w, c_j) + \alpha]}$$

Laplace smoothing

- Simple, easy to use
- Effective in practice

Overall process

- Input: Set of annotated documents $\{(d_i, c_i)\}_{i=1}^{n}$

A. Compute vocabulary **V** of all words

B. Calculate

$$\hat{P}(c_j) = \frac{\text{Count}(c_j)}{n}$$

C. Calculate

$$\hat{P}(w_i|c_j) = \frac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V}[\text{Count}(w, c_j) + \alpha]}$$

D. (Prediction) Given document $d = (w_1, w_2, \ldots, w_k)$

$$c_{\text{MAP}} = \arg\max_{c} \hat{P}(c) \prod_{i=1}^{k} \hat{P}(w_i|c)$$

Variants

Multinomial Naive Bayes
Normal counts (0,1,2,…)
for each document

Binary (Multinomial) NB / Bernoulli NB
Binarized counts (0/1)
For each document

Name based on the distribution of the features

$$P(f_i | y) \rightarrow P(w_i | c)$$

Naive Bayes Example

$$\hat{P}(c) = \frac{N_c}{N}$$

Smoothing with $\alpha = 1$

$$\hat{P}(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$$

	Doc	Words	Class
Training	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
Test	5	Chinese Chinese Chinese Tokyo Japan	?

Priors:

$P(c) = \frac{3}{4}$

$P(j) = \frac{1}{4}$

Conditional Probabilities:

P(Chinese|*c*) = (5+1) / (8+6) = 6/14 = 3/7

P(Tokyo|*c*) = (0+1) / (8+6) = 1/14

P(Japan|*c*) = (0+1) / (8+6) = 1/14

P(Chinese|*j*) = (1+1) / (3+6) = 2/9

P(Tokyo|*j*) = (1+1) / (3+6) = 2/9

P(Japan|*j*) = (1+1) / (3+6) = 2/9

Choosing a class:

$P(c|d5) \propto 3/4 * (3/7)^3 * 1/14 * 1/14 \approx 0.0003$

$P(j|d5) \propto 1/4 * (2/9)^3 * 2/9 * 2/9 \approx 0.0001$

Some details

- Vocabulary is important
- Tokenization matters: it can affect your vocabulary
  - Tokenization = how you break your sentence up into tokens / words
  - Make sure you are consistent with your tokenization!
    - *aren't* $\boxed{\texttt{aren't}}$
$\boxed{\texttt{arent}}$
$\boxed{\texttt{are}}$ $\boxed{\texttt{n't}}$
$\boxed{\texttt{aren}}$ $\boxed{\texttt{t}}$
    - Emails, URLs, phone numbers, dates, emoticons
  - Special multi-word tokens: NOT_happy

Some details

- Vocabulary is important
- Tokenization matters: it can affect your vocabulary
  - Tokenization = how you break your sentence up into tokens / words
  - Make sure you are consistent with your tokenization!
- Handling unknown words in test not in your training vocabulary?
  - Remove them from your test document! Just ignore them.
- Handling stop words (common words like *a*, *the* that may not be useful)
  - Remove them from the training data!

Better to use
- Modified counts (tf-idf) that down weighs frequent, unimportant words
- Better models!

Features

- In general, Naive Bayes can use any set of features, not just words
  - URLs, email addresses, Capitalization, …
  - Domain knowledge can be crucial to performance

Top features for Spam detection

Rank	Category	Feature	Rank	Category	Feature
1	Subject	Number of capitalized words	1	Subject	Min of the compression ratio for the bz2 compressor
2	Subject	Sum of all the character lengths of words	2	Subject	Min of the compression ratio for the zlib compressor
3	Subject	Number of words containing letters and numbers	3	Subject	Min of character diversity of each word
4	Subject	Max of ratio of digit characters to all characters of each word	4	Subject	Min of the compression ratio for the lzw compressor
5	Header	Hour of day when email was sent	5	Subject	Max of the character lengths of words
		(a)			(b)

Spam URLs Features

1	URL	The number of all URLs in an email	1	Header	Day of week when email was sent
2	URL	The number of unique URLs in an email	2	Payload	Number of characters
3	Payload	Number of words containing letters and numbers	3	Payload	Sum of all the character lengths of words
4	Payload	Min of the compression ratio for the bz2 compressor	4	Header	Minute of hour when email was sent
5	Payload	Number of words containing only letters	5	Header	Hour of day when email was sent

Naive Bayes and Language Models

- If features = bag of words, NB gives a per class unigram language model!

- For class c, assigning each word: $P(w \mid c)$

  assigning sentence: $P(s \mid c) = \prod_{w \in s} P(w \mid c)$

Example with positive and negative sentiments

Model pos	
0.1	I
0.1	love
0.01	this
0.05	fun
0.1	film

I	love	this	fun	film
0.1	0.1	0.01	0.05	0.1

$$P(s \mid pos) = 0.0000005$$

Naive Bayes as a language model

- Which class assigns the higher probability to s?

Model pos	
0.1	I
0.1	love
0.01	this
0.05	fun
0.1	film

Model neg	
0.2	I
0.001	love
0.01	this
0.005	fun
0.1	film

I	love	this	fun	film
0.1	0.1	0.01	0.05	0.1
0.2	0.001	0.01	0.005	0.1

$P(s|pos) > P(s|neg)$

Advantages of Naive Bayes

- Very Fast, low storage requirements
- Robust to Irrelevant Features

  Irrelevant Features cancel each other without affecting results
- Very good in domains with many equally important features

  Decision Trees suffer from *fragmentation* in such cases – especially if little data
- Optimal if the independence assumptions hold: If assumed independence is correct, then it is the Bayes Optimal Classifier for problem
- A good dependable baseline for text classification
  - **But we will see other classifiers that give better accuracy**

Failings of Naive Bayes (1)

- Independence assumptions are too strong

x1	x2	Class: x_1 XOR x_2
1	1	0
0	1	1
1	0	1
0	0	0

- XOR problem: Naive Bayes cannot learn a decision boundary

Independence assumption broken!

- Both variables are jointly required to predict class

Failings of Naive Bayes (2)

- Class imbalance:
  - One or more classes have more instances than others
  - Data skew causes NB to prefer one class over the other
    - 100 documents with class=MA and “Boston” occurring once each
    - 10 documents with class=BC and “Vancouver” occurring once each
    - New document d: “Boston Boston Vancouver Vancouver Vancouver”

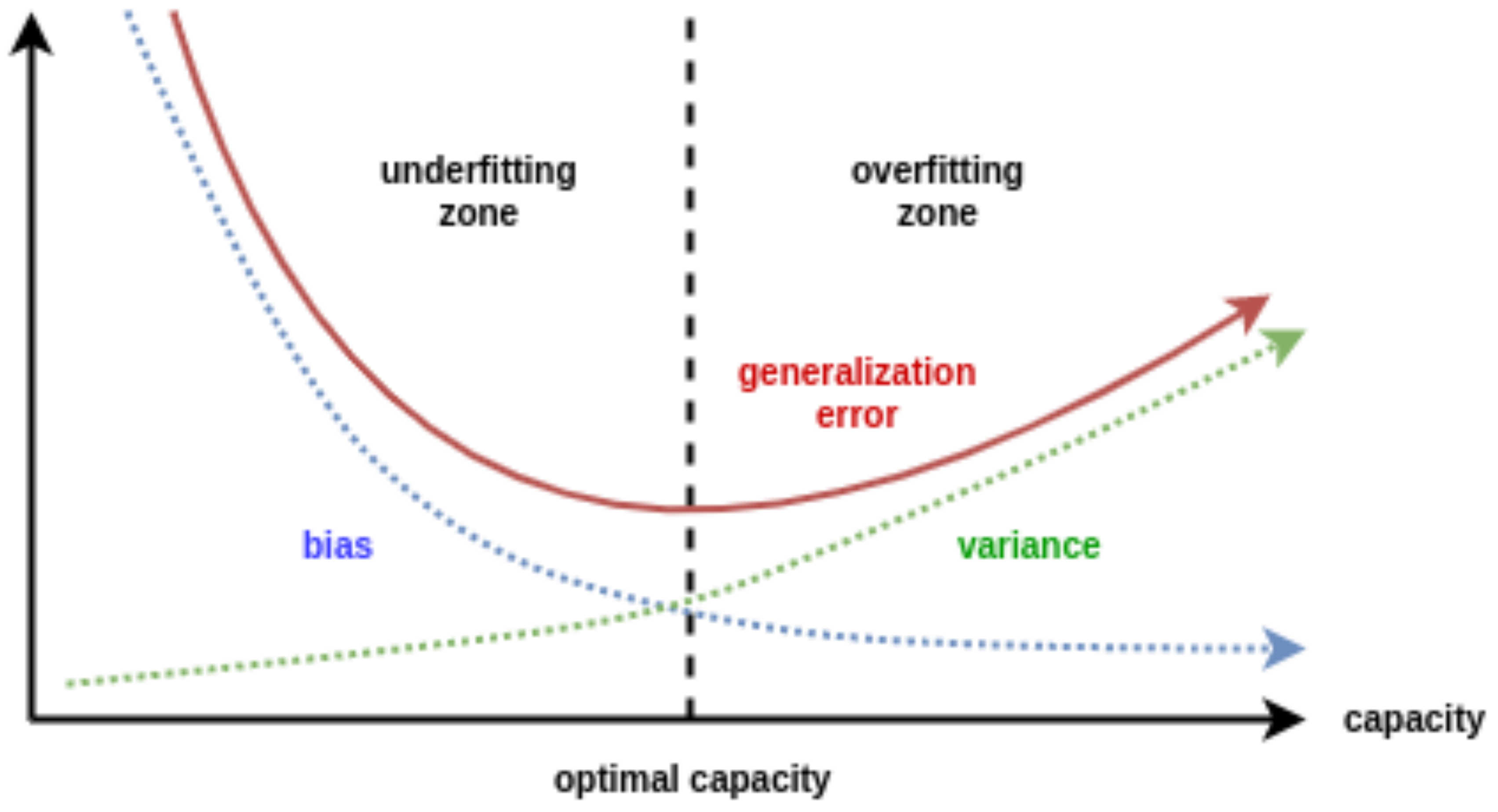
$$P(\text{class} = \text{MA} \mid d) > P(\text{class} = \text{BC} \mid d)$$

Does not handle rare classes well

- Okay if test distribution follows training and you don’t care about the rare classes
- Low macro-average metrics

Re-weight classes if needed

When to use Naive Bayes



- Small data sizes:
  - Naive Bayes is great! (high bias)
  - Rule-based classifiers might work well too
- Medium size datasets:
  - More advanced classifiers might perform better (e.g. SVM, logistic regression)
- Large datasets:
  - Naive Bayes becomes competitive again (although most classifiers work well)

Practical text classification

- Domain knowledge is crucial to selecting good features
- Handle class imbalance by re-weighting classes
- Use log scale operations instead of multiplying probabilities
- Since $\log(xy) = \log(x) + \log(y)$:

$$c_{\text{MAP}} = \arg\max_{c_j \in C} \log P(c_j) + \sum_{i=1}^{k} \log P(x_i \mid c_j)$$

better to sum logs of probabilities than to multiply probabilities

- Class with highest un-normalized log probability score is still the most probable
- Model is now just max of sum of weights