Introduction to NLP

Probabilistic models

Compressed out of NLP courses from **Dan Jurafsky** (Stanford), & David Bamman (Berkeley), Michael Collins (MIT & Columbia), and some online (Udemy) courses

Book: **Speech and Language Processing** by Jurafsky & Martin (3rd edition)

Why teach NLP in SMU?

1. Language/text has a **symbolic** structure
2. It is all about **machine learning** these days
3. NLP is a core part of **Artificial Intelligence**
    - However, there is no NLP at FEL

After this short NLP block, you should be able to:

- Recognize some classic NLP tasks when encountered
- Understand some modern NLP methods and models:
    1. probabilistic models
    2. vector/matrix models
    3. neural models
- Implement and/or use these in practice (Python)

What is NLP?

NLP = Natural Language Processing

- a.k.a. computational linguistics (from a linguist’s point of view)

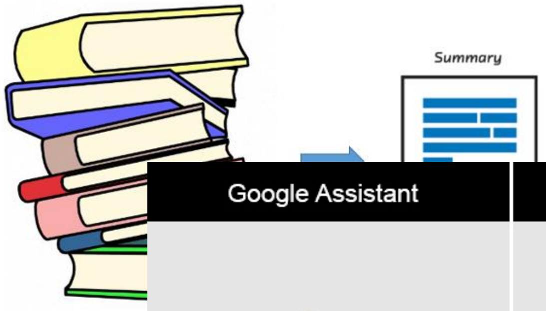
Intersection of:

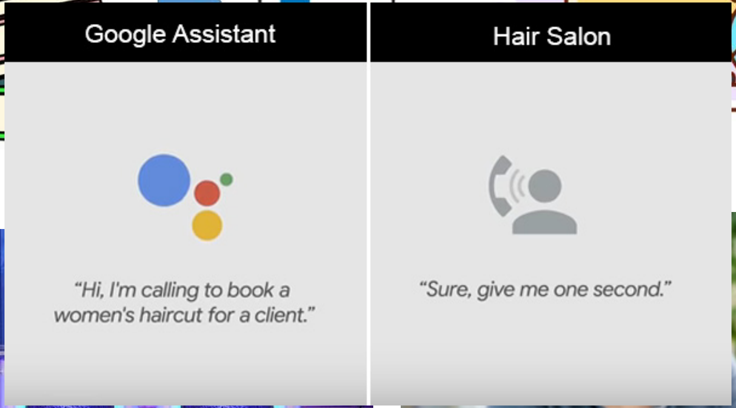
- Linguistics
- AI/ML
- CS

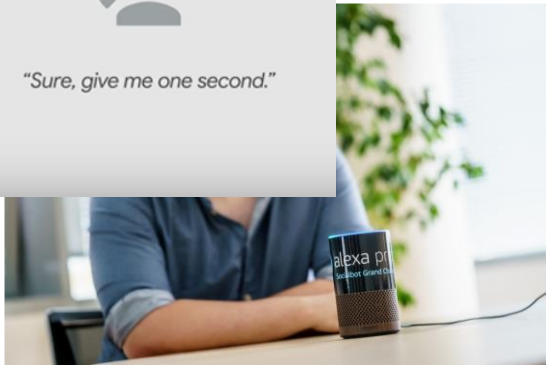
Goal: process language with computers to perform useful things…

Why learn NLP?: Practical viewpoint

- Part of speech tagging
- Named entity recognition
- Language modelling
- Topic modelling
- Information extraction
- Text Summarization
- Machine translation
- Question answering
- Conversational agents

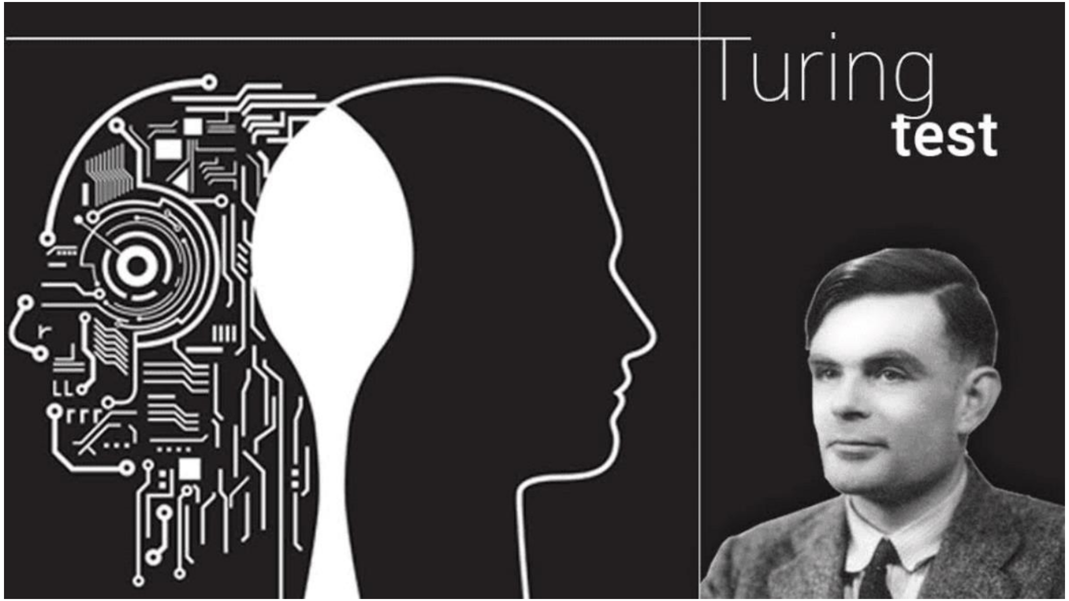


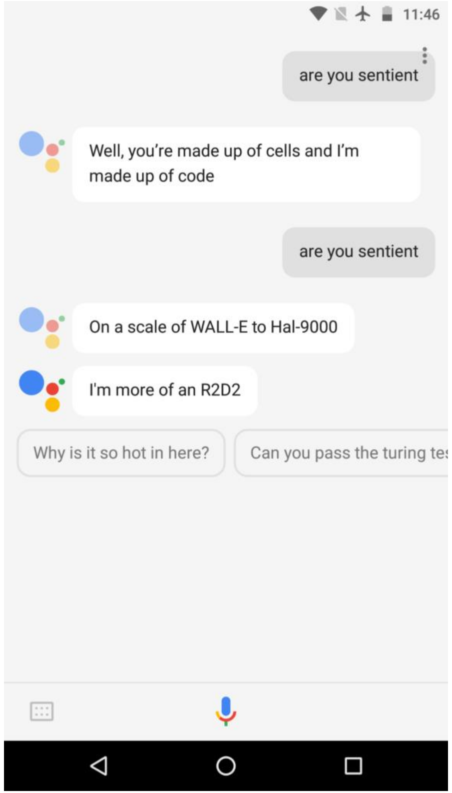




Why learn NLP?: Theoretical viewpoint

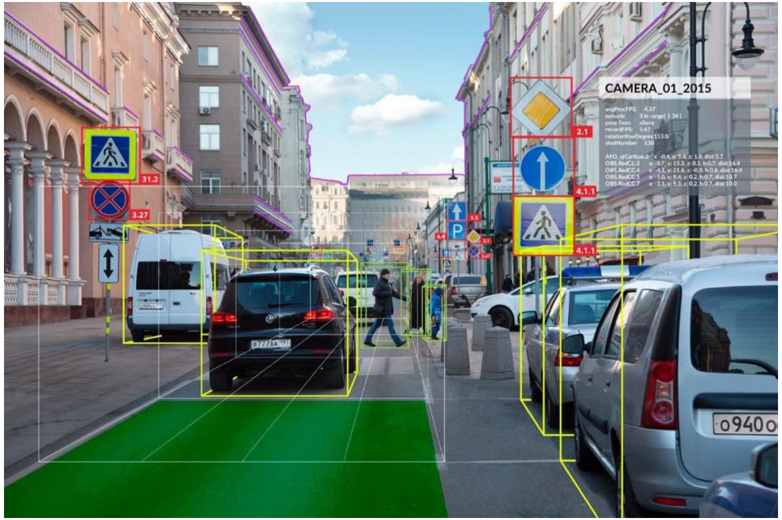
- Language is the natural testbed for intelligence!

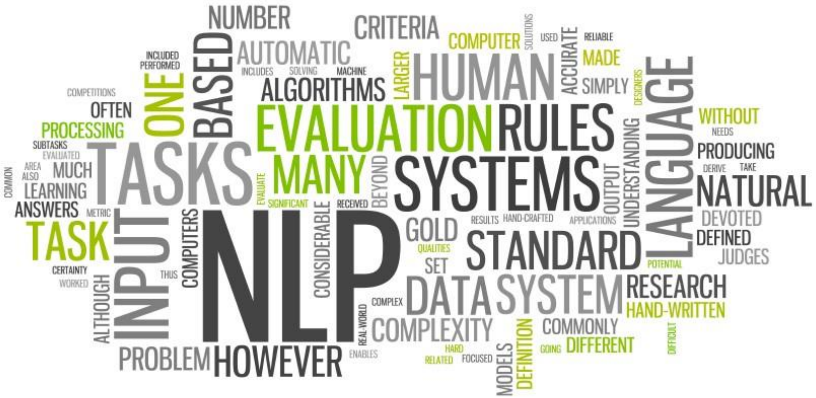




Why learn NLP?: Theoretical viewpoint

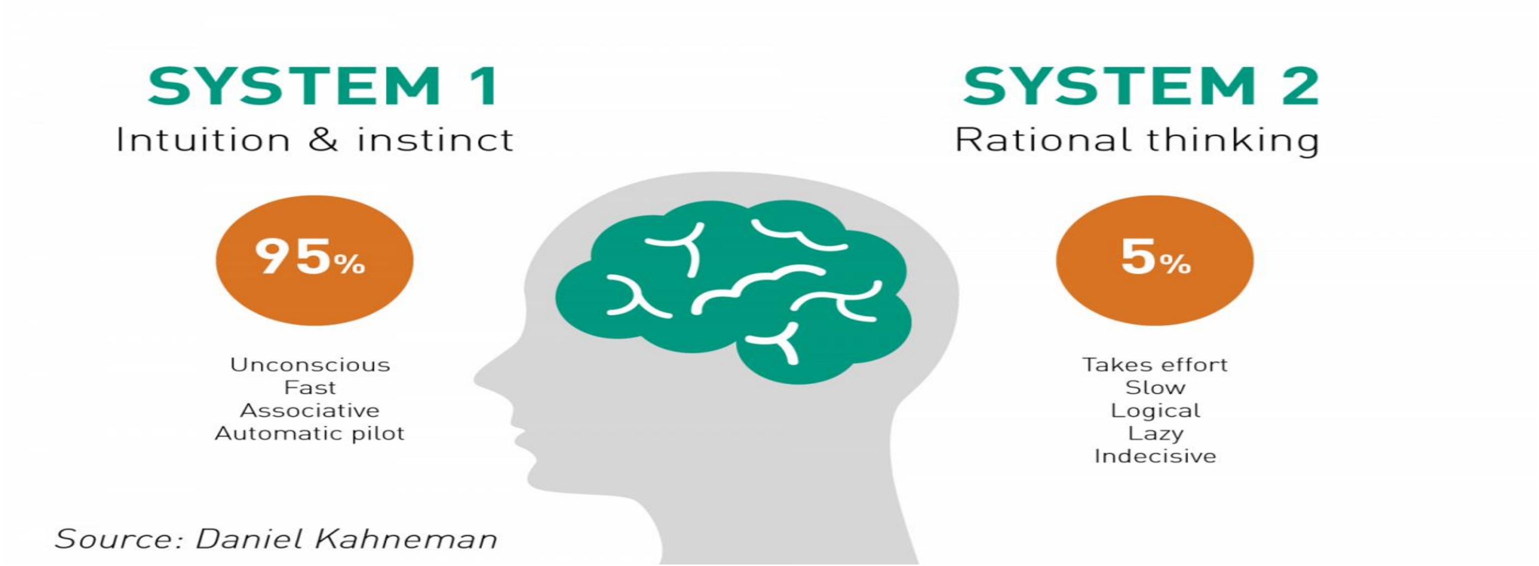
- Language is the natural testbed for intelligence! **Why**?
- There are 2 most abundant sources of data: Visual and Textual





Why learn NLP?: Theoretical viewpoint

- Language is the natural testbed for intelligence! **Why**?
- There are 2 most abundant sources of data: Visual and Textual
- However, while even insects can see, Language is characteristic to humans



Probabilistic Models

- Language modelling

Probabilistic Language Models

- Goal: assign probability to a sentence

- Machine Translation:
  - P(**high** winds tonite) > P(**large** winds tonite)
- Spell Correction
  - The office is about fifteen **minuets** from my house
    - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- + Summarization, question-answering, …

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

- Related task: probability of an upcoming word:

  $P(w_5|w_1,w_2,w_3,w_4)$

- A model that computes either of these:

  $P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$ is called a **language model** or **LM**.

- Alternative name: **grammar**

How to compute P(W)

- How to compute this joint probability:

  P(its, water, is, so, transparent, that)

- Let's start with the Bayes rule:

  $P(A,B) = P(A)\,P(B|A)$

- And now more generally ("**Chain Rule of Probability**")

$$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$$

Joint probability of words in sentence

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

- Could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- **Too many possible sentences**!
- We’ll never see enough data for estimating these

Markov Assumption


Andrei Markov

- A simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- Or maybe a bit less restrictive

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Simplest case: Unigram model

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model:

```
fifth, an, of, futures, the, an, incorporated, a, a, the,
inflation, most, dollars, quarter, in, is, mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the
```

Bigram model

= Condition on the previous word:

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
    - because language has **long-distance dependencies**:

      "*The **computer** which I had just put into the machine room on the fifth floor **crashed**.*"

- But we can often get away with N-gram models in practice

Probabilistic Language Modelling

- Estimating N-gram Probabilities

Estimating bigram probabilities

- Using Maximum Likelihood Estimate:

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Example: Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i’m looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i’m looking for a good place to eat breakfast
- when is caffe venezia open during the day
- …….

Raw bigram counts

- Out of 9222 sentences:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw bigram probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Bigram estimates of sentence probabilities

P(<s> I want english food </s>) =

P(I|<s>) × P(want|I) × P(english|want) × P(food|english) × P(</s>|food) = .000031

- What types of knowledge in a LM?
  - P(english|want) = .0011
  - P(chinese|want) = .0065
  - P(to|want) = .66
  - P(eat | to) = .28
  - P(food | to) = 0
  - P(want | spend) = 0
  - P (i | <s>) = .25

Practical Issues

- We do everything in log space!
  - to avoid numeric underflow
    - also adding is faster than multiplying
    - though log can be slower than multiplication

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

https://books.google.com/ngrams

Probabilistic Language Modelling

- Evaluation and Perplexity

Extrinsic evaluation of N-gram models

- Does our language model prefer good sentences to bad ones?
    - Assign higher probability to “real” or “frequently observed” sentences
        - Than “ungrammatical” or “rarely observed” sentences?

- Best evaluation for comparing models A and B
    - Put each model in a task
        - spelling corrector, speech recognizer, MT system
    - Run the task, get an accuracy for A and for B
        - How many misspelled words corrected properly
        - How many words translated correctly
    - Compare accuracy for A and B

Difficulty of extrinsic evaluation

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So:
  - Sometimes we use **intrinsic** evaluation: **perplexity**
  - Bad approximation
    - unless the test data looks just like the training data
    - So generally only useful in pilot experiments
  - But is helpful to think about.

Intuition of Perplexity

- The **Shannon Game**:
  How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

- Unigrams are terrible at this game. (Why?)

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

A better model of a text is one which assigns a higher probability to the word that actually occurs

- The best language model is one that best predicts an unseen test set
  - Gives the highest P(sentence)

Perplexity

Perplexity is the inverse probability of the sentence, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule: $$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

For bigrams: $$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

*perplexity is also closely related to **cross-entropy** $PP(W) = 2^{H(W)} = 2^{-\frac{1}{N} \log_2 P(w_1, w_2, \ldots, w_N)}$

The Shannon Game intuition for perplexity

- Perplexity is a “*weighted equivalent branching factor*”
- How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
  - Perplexity = 10
- How hard is recognizing (30,000) names at Microsoft.
  - Perplexity = 30,000
- Let's imagine a call-routing phone system gets 120K calls and has to recognize
  a. "Operator" (let's say this occurs 1 in 4 calls)
  b. "Sales" (1in 4)
  c. "Support" (1 in 4)
  d. 30,000 different names (each name occurring 1 time in the 120K calls)
  - We get the perplexity of this sequence of length 120Kby first multiplying 120K probabilities
    - (90K of which are 1/4 and 30K of which are 1/120K), and then taking the inverse 120,000th root:

$$
\begin{aligned}
PP(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Perplexity = (¼ * ¼ * ¼* ¼ * ¼ * …. * 1/120K * 1/120K * ….)^(-1/120K)

- This can be arithmetically simplified to just N = 4: the operator (1/4), the sales (1/4), the tech support (1/4), and the 30,000 names (1/120,000): **Perplexity = ((¼ * ¼ * ¼ * 1/120K)^(-1/4) = 52.6**

Lower perplexity = better model

- Training 38 million words, test 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

The Shannon Visualization Method

- Choose a random bigram (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
  - And so on until we choose </s>
  - Finally string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

Approximating Shakespeare: Random Sampling

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

Shakespeare as a corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types
  - out of V^2= 844 million possible bigrams.
- So 99.96% of the possible bigrams were never seen
  - have zero entries in the table
- Quadrigrams even worse:
  - What's coming out looks like Shakespeare because it ***is*** Shakespeare!

Probabilistic Language Modelling

- Overfitting and Smoothing

The perils of overfitting: Zeros

- Training set:

  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

  P("offer" | denied the) = 0

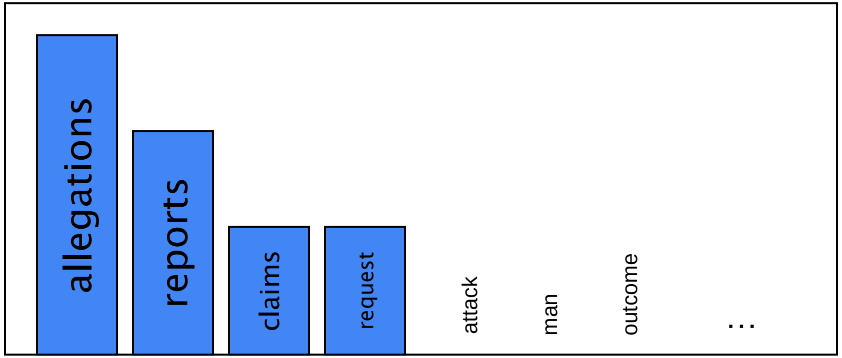
- Test set:

  … denied the offer
  … denied the loan

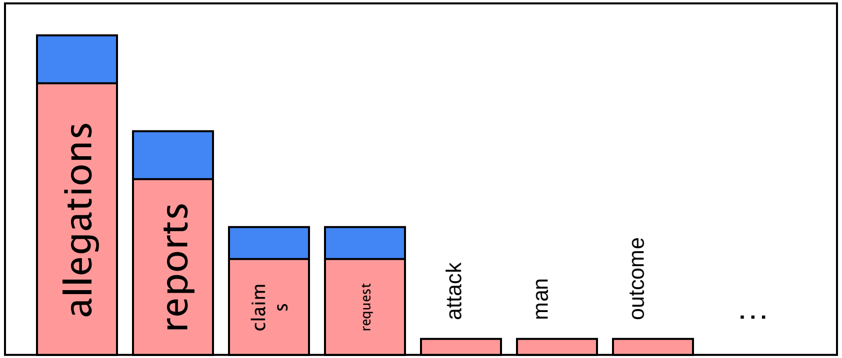
- Bigrams with zero probability!
  - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

The intuition of smoothing

- When we have sparse statistics:
  - P(w | denied the)
    - 3 allegations
    - 2 reports
    - 1 claims
    - 1 request
    - 7 total



- Steal probability mass to generalize better
  - P(w | denied the)
    - 2.5 allegations
    - 1.5 reports
    - 0.5 claims
    - 0.5 request
    - 2 other
    - 7 total



Add-one estimation

- Also called **Laplace smoothing**
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

No longer a MLE!

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Compare with raw bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Probabilistic Language Modelling

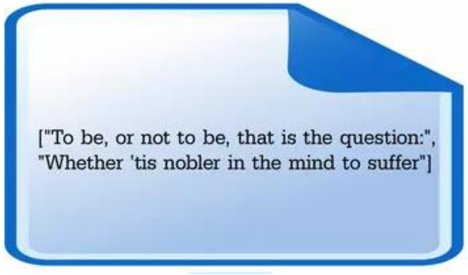
- Supervised Text Classification

Text classification?

- Spam detection
- Authorship identification
- Age/Gender recognition
- Language identification
- Sentiment classification
- Topic classification
- …

Text classification: task

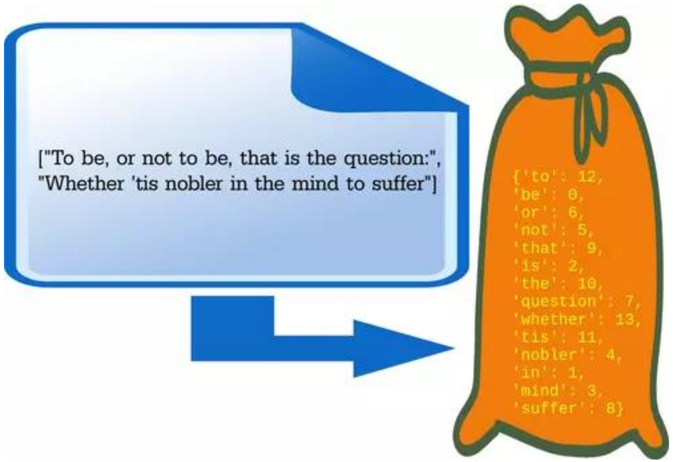
- *Input:*
  - a document ***d***
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
  - A training set of ***m*** hand-labeled documents $(d_1,c_1),\ldots,(d_m,c_m)$
- *Output:*
  - a learned classifier $f{:}d \rightarrow c$

$f($ 

$) = c$

Text classification: methods

- Naturally, any kind of classifier can be used
    - Rule-based systems
    - **Naïve Bayes**
    - Logistic regression
    - Support-vector machines
    - Neural networks
    - ...



The bag of words representation

$f($

seen	2
sweet	1
whimsical	1
recommend	1
happy	1
...	...

$) = c$

Naive Bayes

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

- How to predict the class **c** for a document **d**?
  - let's apply the Bayes rule again!

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\, P(c \mid d)$$

MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}} \frac{P(d \mid c)P(c)}{P(d)}$$

Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}}\, P(d \mid c)P(c)$$

Dropping the denominator

$$= \underset{c \in C}{\operatorname{argmax}}\, P(x_1, x_2, \ldots, x_n \mid c)P(c)$$

"Likelihood" "Prior"

Document d represented as **features $x_1 \ldots x_n$**

Naive Bayes: Tractability Problem

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}} P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

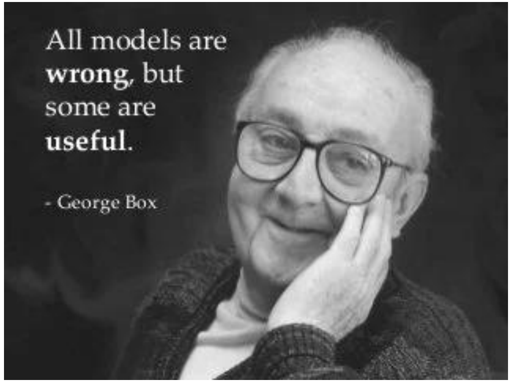
O(|X|^n•|C|) parameters!

How often does this class occur?

Could only be estimated if a very, very large number of training examples was available.

We can just count the relative frequencies in a corpus

Naive Bayes: Independence Assumptions



- **Bag of Words assumption**
  - Assume word position doesn't matter

$$P(x_1, x_2, \ldots, x_n \mid c)$$

- **Conditional Independence**
  - Assume the feature probabilities $P(x_i|c_j)$ are independent given the class c.

$$P(x_1, \ldots, x_n \mid c) = P(x_1 \mid c) \bullet P(x_2 \mid c) \bullet P(x_3 \mid c) \bullet \ldots \bullet P(x_n \mid c)$$

- Naive Bayes model inference*:*

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in positions} P(x_i \mid c_j)$$

Naive Bayes: log space

- Multiplying a lot of small number leads to underflow problems…
  - Solution - **move to log space**!

- Instead of: $$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j) \prod_{i \in positions} P(x_i \mid c_j)$$

- We calculate: $$c_{\text{NB}} = \underset{c_j \in C}{\operatorname{argmax}} \left[\log P(c_j) + \sum_{i \in \text{positions}} \log P(x_i | c_j) \right]$$

- Notes:
  1) Taking log doesn't change the ranking of classes!
     - The class with highest probability also has highest log probability!
  2) It's a linear model:
     - Just a max of a sum of weights: a **linear** function of the inputs

- So naive bayes is a **linear classifier**

Naive Bayes: Learning the parameters

- You have seen this before: maximum likelihood estimates!
  - simply use the frequencies in the data
- The prior for the class probabilities: $\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$
- The likelihood for the words: $\hat{P}(w_i \mid c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$
  - "merge" all words for each class

Problem with Maximum Likelihood

what about completely unknown words?

- What if we have seen no training documents with the word ***fantastic*** and classified in the topic **positive** (***thumbs-up)***?

$$\hat{P}(\text{"fantastic"} \mid \text{positive}) = \frac{count(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} count(w, \text{positive})} = 0$$

- Zero probabilities cannot be conditioned away, no matter the other evidence!

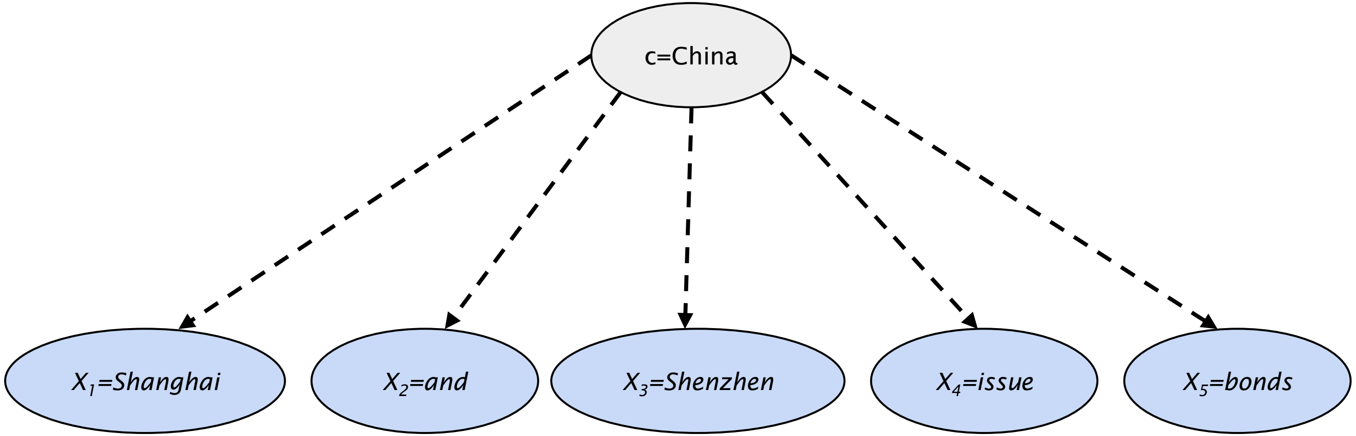
$$c_{MAP} = \operatorname{argmax}_c \hat{P}(c) \prod_i \hat{P}(x_i \mid c)$$

= 0 !

- Solution?
  - **Smoothing** to the rescue!

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} \left(count(w, c) + 1\right)} = \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

Generative Model for Multinomial Naïve Bayes



Naïve Bayes and Language Modeling

- Naïve bayes classifiers can use any sort of feature
  - URL, email address, dictionaries, network features
- But if, as in the previous slides, we use **only** words as features
- Then Naïve bayes has an important similarity to language modeling:
- **Each class = a unigram language model**
- Assigning each word a probability: **P(word|class)**
- Assigning each sentence a probability **P(sentence | class) = ∏ P(word|class)**

Each class = a unigram language model!

<u>Class *pos*</u>

0.1	I
0.1	love
0.01	this
0.05	fun
0.1	film
…	

I	love	this	fun	film
0.1	0.1	.05	0.01	0.1

P(sentence | c=pos) = 0.0000005

Naive Bayes as a Language Model

- Which class assigns the higher probability to a sentence?

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(sentence|pos) > P(sentence|neg)

Probabilistic Language Modelling

- Unsupervised Topic Modelling

Topic models

- **Unsupervised** models for discovering **hidden** “topics” or “themes” in documents
  - Clusters/groups of terms that tend to occur together.
- Input:
  - set of documents
  - number of “topics” to learn
- Output:
  - extracted topics (clusters)
  - topic distribution for each document
  - topic distribution for each word in a document

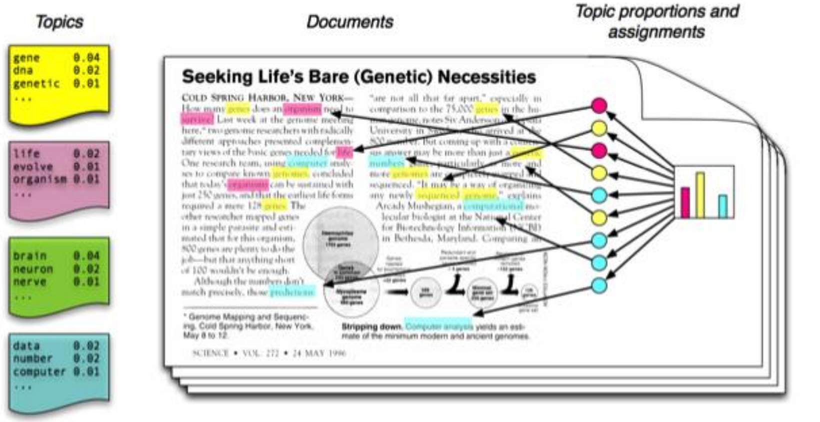
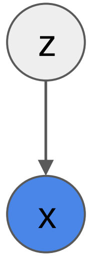


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77-84.

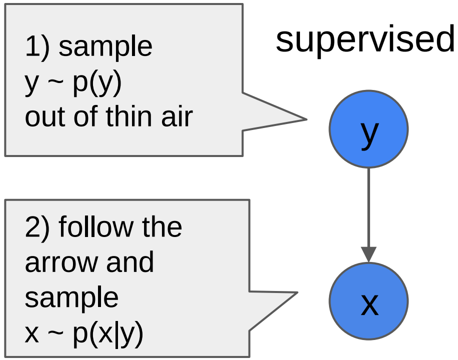
Probabilistic topic models

- Assume a probabilistic generative process that yields the documents
    - this can be hierarchical and quite complex
- Adopts the language of probabilistic graphical models (Bayes nets)
    - Simply a visual way of writing the joint probability
        - Nodes represent variables (blue = observed, grey = latent)
        - Arrows indicate conditional relationships
            - “The probability of x is dependent on z”

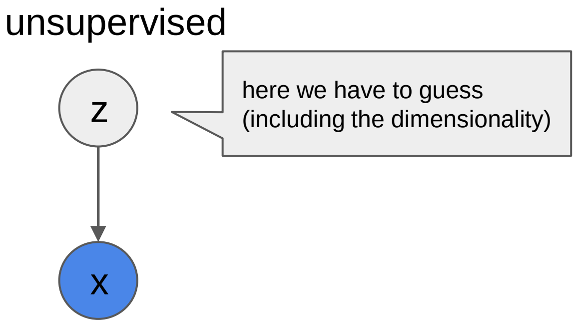


- A latent variable is one that’s unobserved, either because:
    - we are predicting it (but have observed that variable for other data points)
    - it is **unobservable** (e.g., a “topic” of a document)

Graphical models tell a “story” of doc generation



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



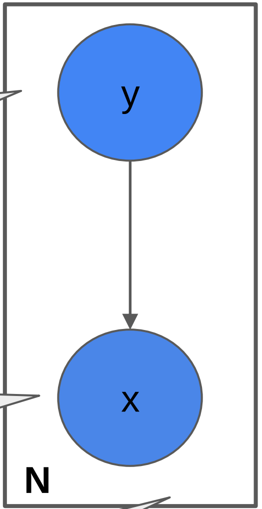
$$P(z|x) = \frac{P(x|z)P(z)}{P(x)}$$

one needs to
resort to EM

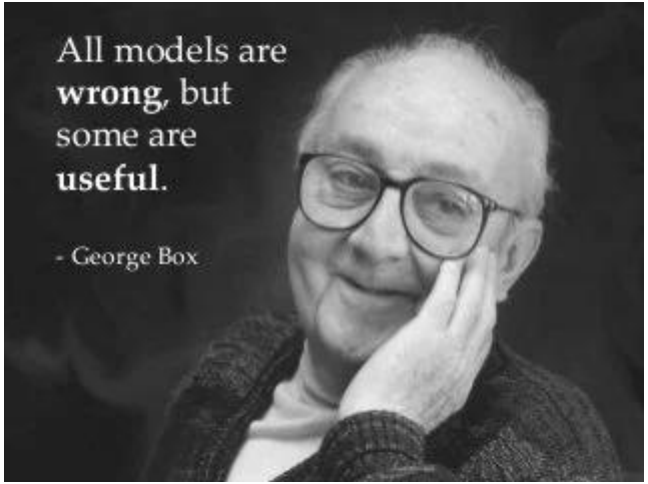
The “story”: Plate notation

e.g. p(y) = [70% spam, 30% ham]

e.g. p(x|y) = “Multinomial[Y=spam]”



for i = 1..N:
    y(i) ~ p(Y)
    x(i) ~ p(X | Y = y(i))



Obviously, that’s not a good way to generate an email…

Email: “*Hello*, *lottery*, *welcomes*, *deadline*, *100%*, *viagra*, *inherit*, …”

Latent Dirichlet Allocation (LDA)

- The absolute classic method of choice for probabilistic topic modelling

David Blei

SLEDOVAT

Professor of Statistics and Computer Science, Columbia University
E-mailová adresa ověřena na: columbia.edu - Domovská stránka
Machine Learning Statistics Probabilistic topic models Bayesian nonparametrics
Approximate posterior infer...

Citace ZOBRAZIT VŠECHNY

	Všechny	Od 2017
Citace	106223	59185
h-index	96	80
i10-index	205	184

Andrew Ng

SLEDOVAT

Stanford University
E-mailová adresa ověřena na: cs.stanford.edu - Domovská stránka
Machine Learning Deep Learning AI

Citace ZOBRAZIT VŠECHNY

	Všechny	Od 2017
Citace	195969	114815
h-index	134	103
i10-index	295	269

Michael I. Jordan

SLEDOVAT

Professor of Electrical Engineering and Computer Sciences and Professor of Statistics, UC Berkeley
E-mailová adresa ověřena na: cs.berkeley.edu - Domovská stránka
machine learning computer science statistics artificial intelligence optimization

Citace ZOBRAZIT VŠECHNY

	Všechny	Od 2017
Citace	226651	106096
h-index	186	129
i10-index	629	480

Journal of Machine Learning Research 3 (2003) 993-1022 Submitted 2/02; Published 1/03

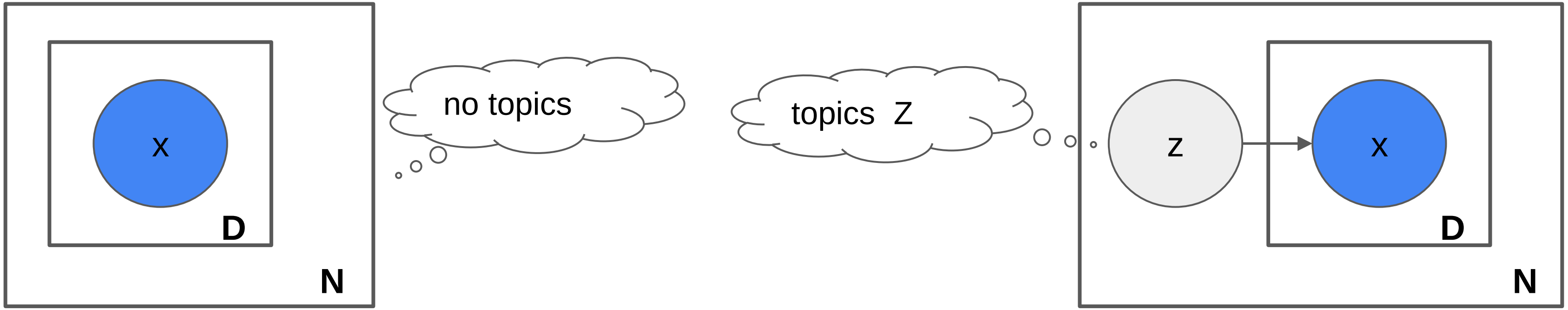
Latent Dirichlet Allocation

David M. Blei BLEI@CS.BERKELEY.EDU
Computer Science Division
University of California
Berkeley, CA 94720, USA

Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department
Stanford University
Stanford, CA 94305, USA

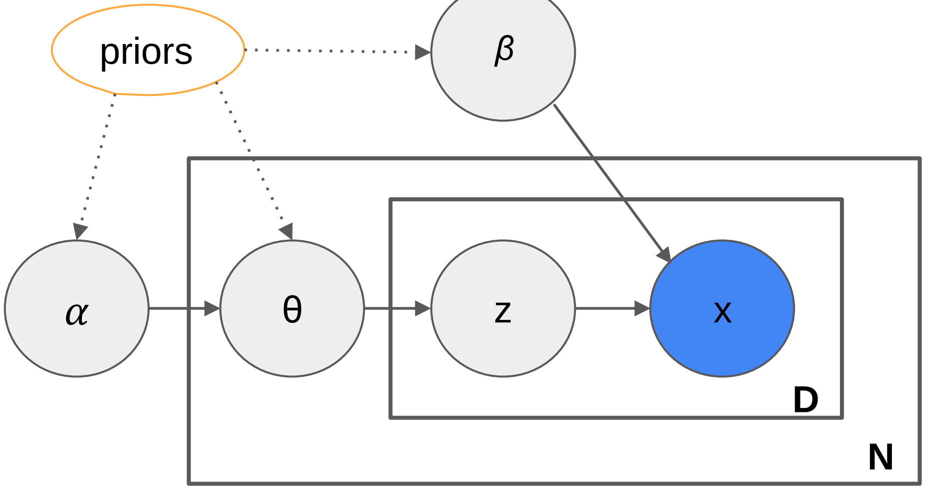
Michael I. Jordan JORDAN@CS.BERKELEY.EDU
Computer Science Division and Department of Statistics
University of California
Berkeley, CA 94720, USA

[PDF] **Latent dirichlet allocation** [PDF] jmlr.org
DM Blei, AY Ng, MI Jordan - Journal of machine Learning research, 2003 - jmlr.org
We describe **latent Dirichlet allocation** (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in ...
Uložit Citovat Počet citací tohoto článku: 41579 Související články Všechny verze (počet: 108)



unigram

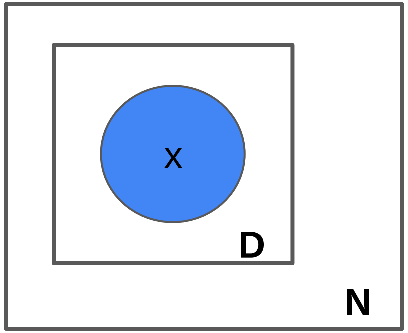
mixture of unigrams



Bayesian topic model

The “story” of corpus generation: unigrams

```
for i = 1..N:
    for j = 1..D:
        x(i,j) ~ p(X)
```

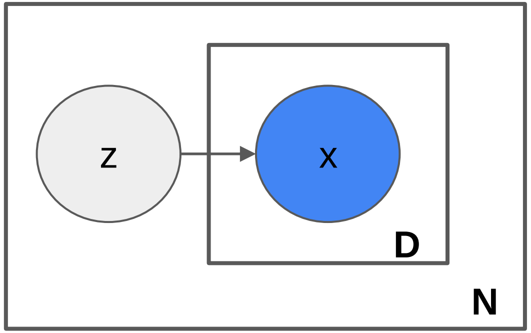


Probability of a document:

$$p(x) = \prod_{j=1}^{D} p(x_j)$$

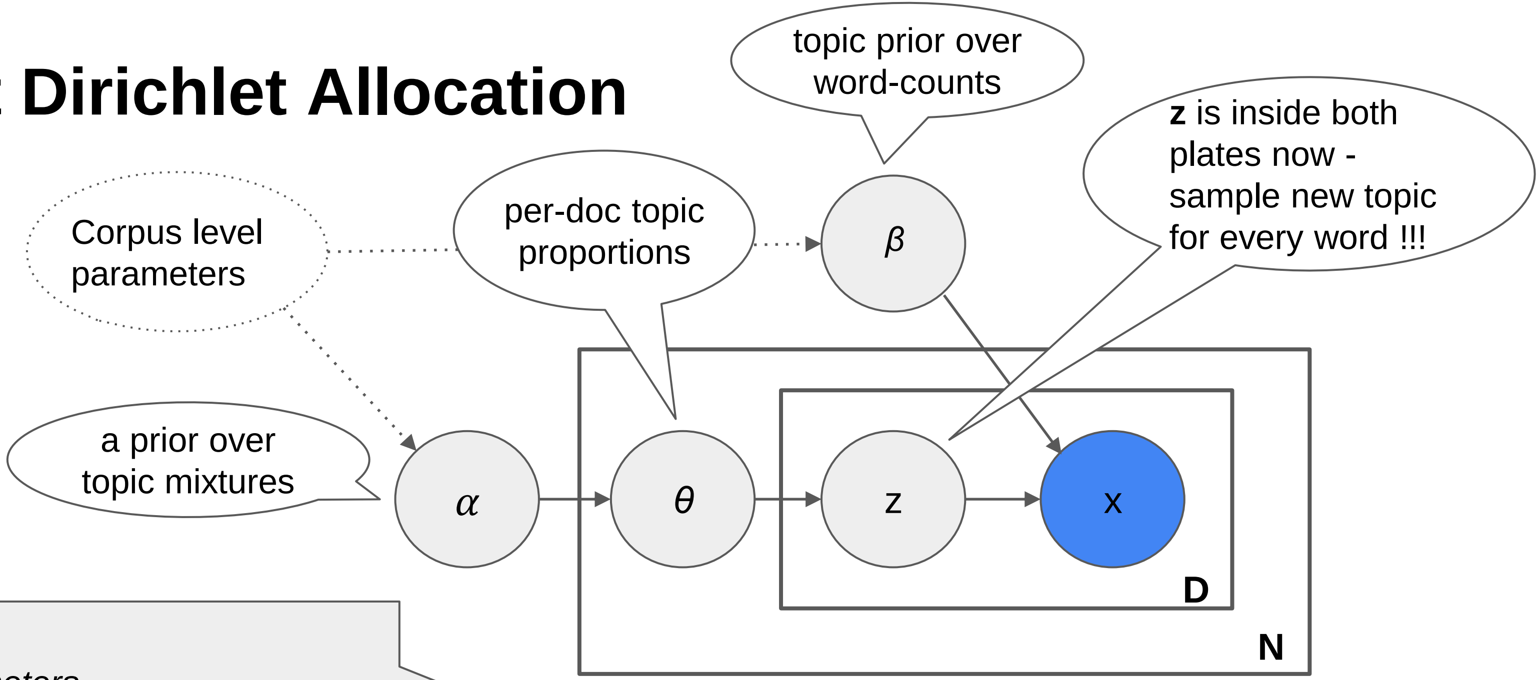
The “story” of corpus generation: mixture models

for i = 1..N:
    z(i) ~ p(Z)
    for j = 1..D:
        x(i,j) ~ p(X | Z = z(i))



$$p(z, x) = p(z) \prod_{j=1}^{D} p(x_j | z)$$

Latent Dirichlet Allocation



a topic model

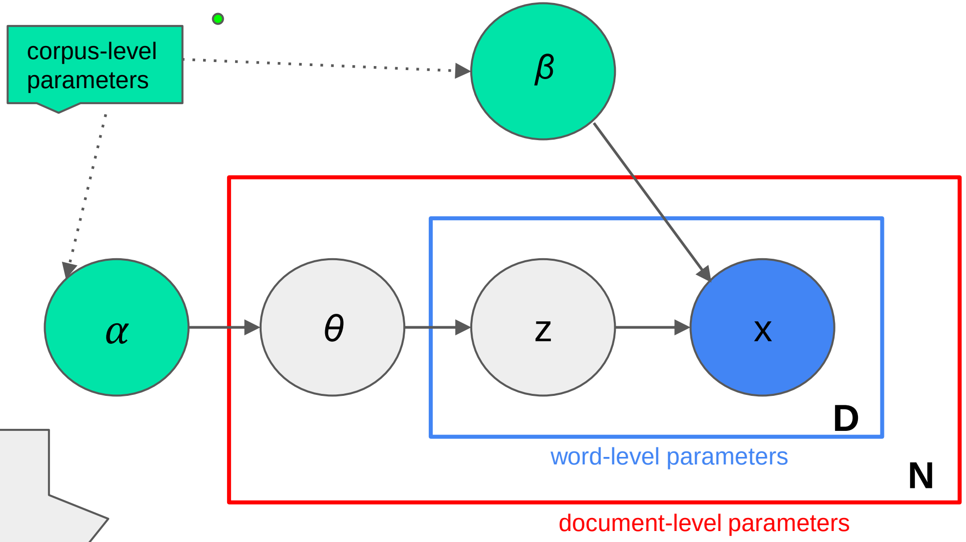
α, β = *parameters*
for i = 1..N:
  θ(i) ~ Dirichlet(α)
  for j = 1..D:
    z(i,j) ~ Multinom(θ*(i)*)
    x(i,j) ~ p(X | Z = z(i,j), β)

$$p(\theta, z, x|\alpha, \beta) = p(\theta|\alpha) \prod_{j=1}^{D} p(z_j|\theta)p(x_j|z_j, \beta)$$

Latent Dirichlet Allocation

+ number of topics = **K**

+ vocabulary size = **V**



multi-dim. distributions!

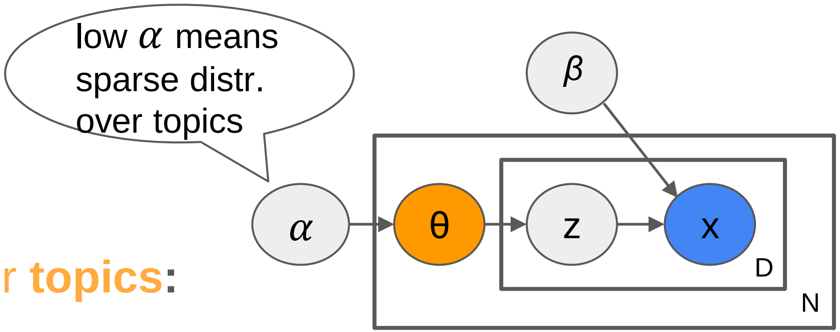
α, $\beta \in \mathbb{R}^{KxV}$
for i = 1..N:
    $\theta(i)$ ~ **Dirichlet**$^K(\alpha)$ # $\theta \in \mathbb{R}^{NxK}$
    for j = 1..D_i:
        z(i,j) ~ **Multinom**$^K(\theta(i))$
        x(i,j) ~ **Multinom**^{V}(z(i,j), β)

$$p(\theta, z, x|\alpha, \beta) = p(\theta|\alpha) \prod_{j=1}^{D} p(z_j|\theta) p(x_j|z_j, \beta)$$

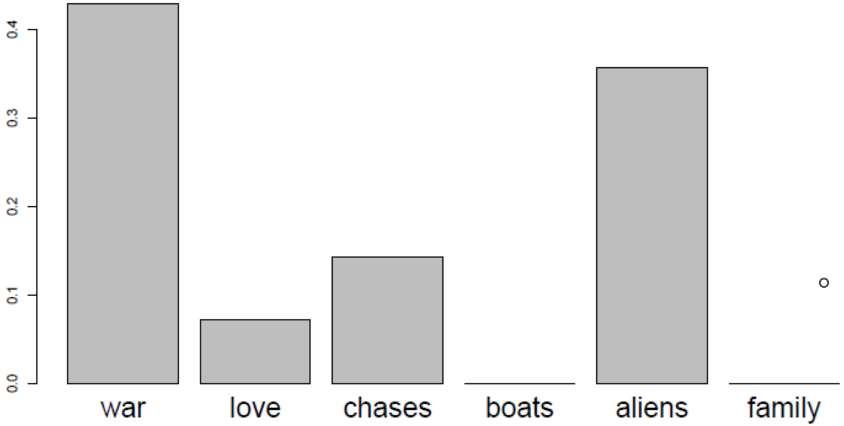
Bayesian machine learning

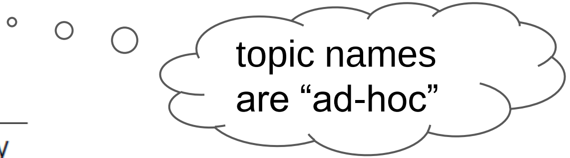
- Under normal circumstances, the θ would be a normal parameter
  - e.g. a weight in a neural network
- But in Bayesian ML, (almost) everything is a random variable
  - hence we get a distribution over the “weights” θ
  - and this distribution has a hyperparameter α
  - specifically $\theta \sim \mathit{Dirichlet}(\alpha)$
    - typically its symmetric variant where α is a scalar (all topics a-priori equally likely)
- Why Dirichlet ?
  - Intuitively, Dirichlet is a distribution over positive (probability) vectors that sum up to one
    - = parameters for discrete multinomial distributions (of topics)
  - Moreover, it is a **conjugate prior for the multinomial distribution**

Topic model step-by-step

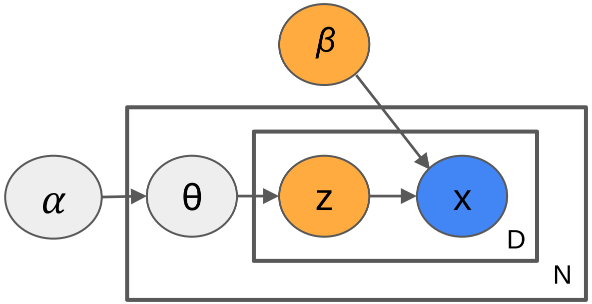


- Each document has distribution over **topics**:





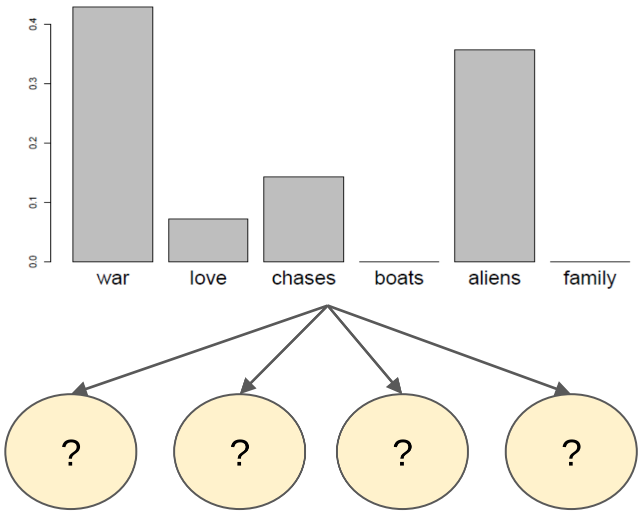
Topic model step-by-step



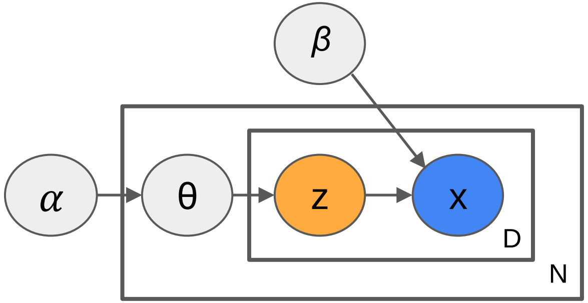
- A topic is a distribution over words:

- e.g., P("adore" | topic=love) = 0.18

Topic model step-by-step

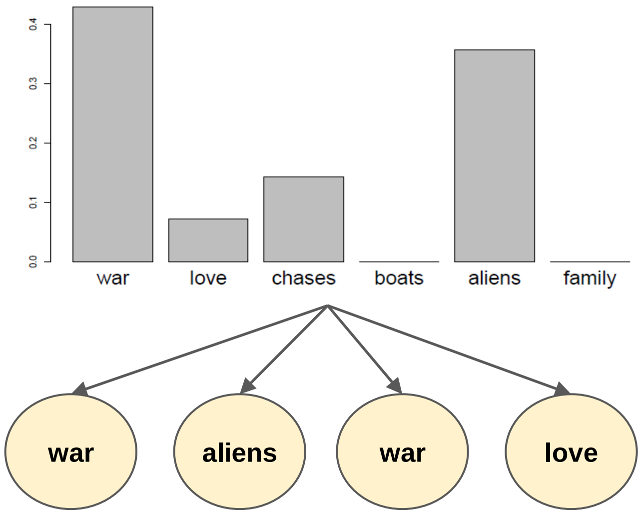


P(topic | topic distribution θ)

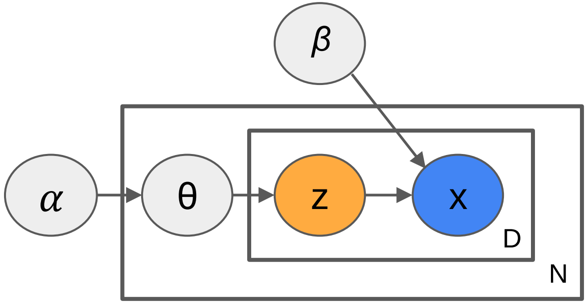


z ~ Multinomial(θ)

Topic model step-by-step

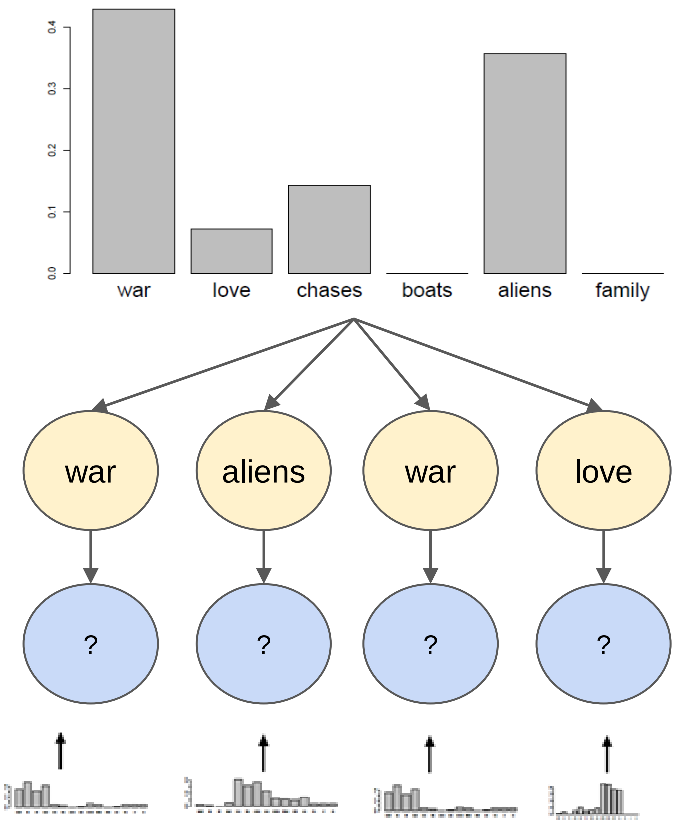


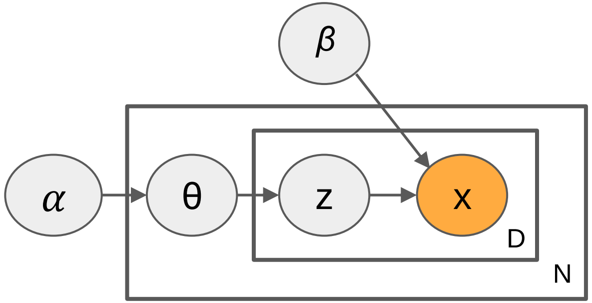
P(topic | topic distribution θ)



$z \sim \text{Multinomial}(\theta)$

Topic model step-by-step

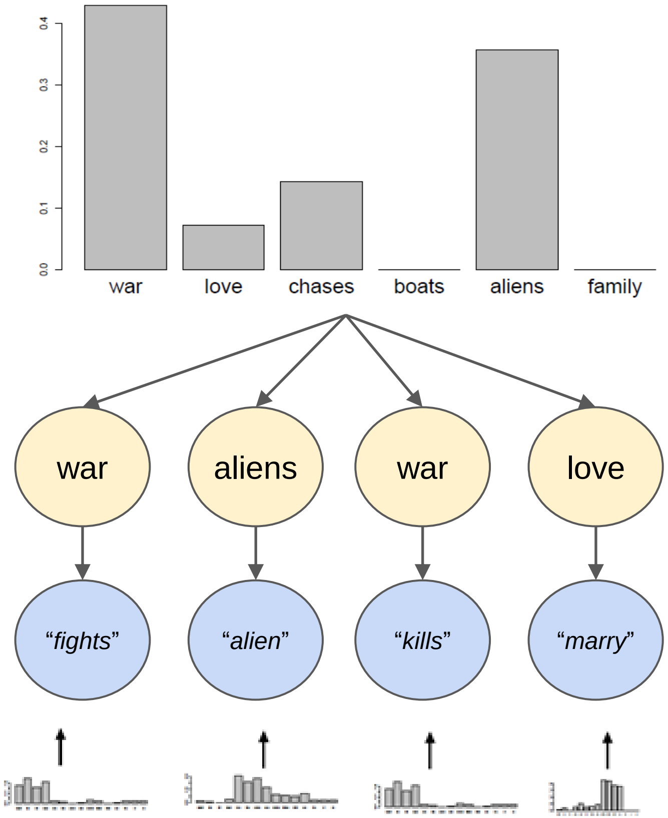


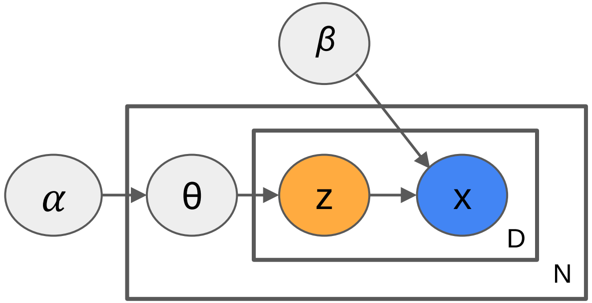


x ~ Multinomial(z,β)

P(word | topic z, β)

Topic model step-by-step

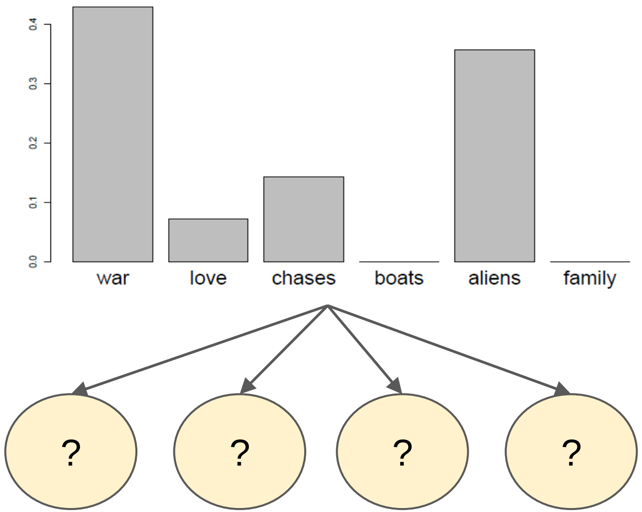


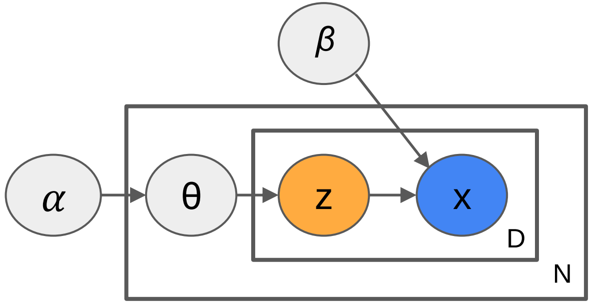


x ~ Multinomial(z,β)

P(word | topic z, β)

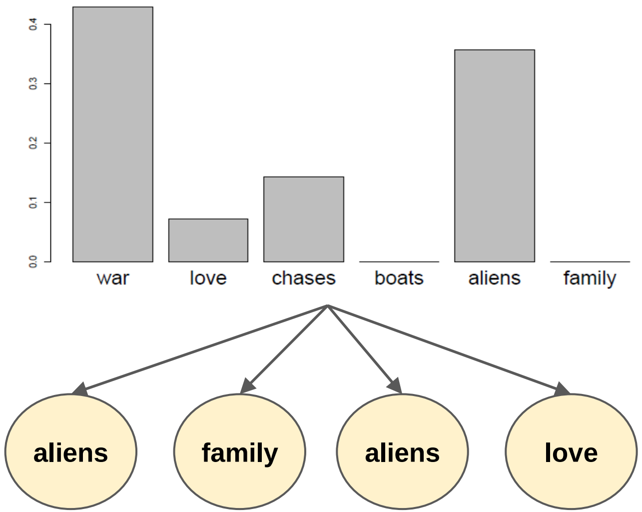
Topic model step-by-step

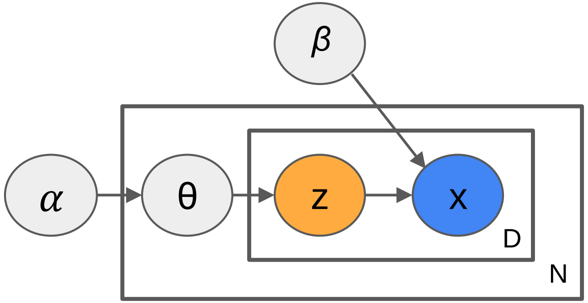




P(topic | topic distribution θ)

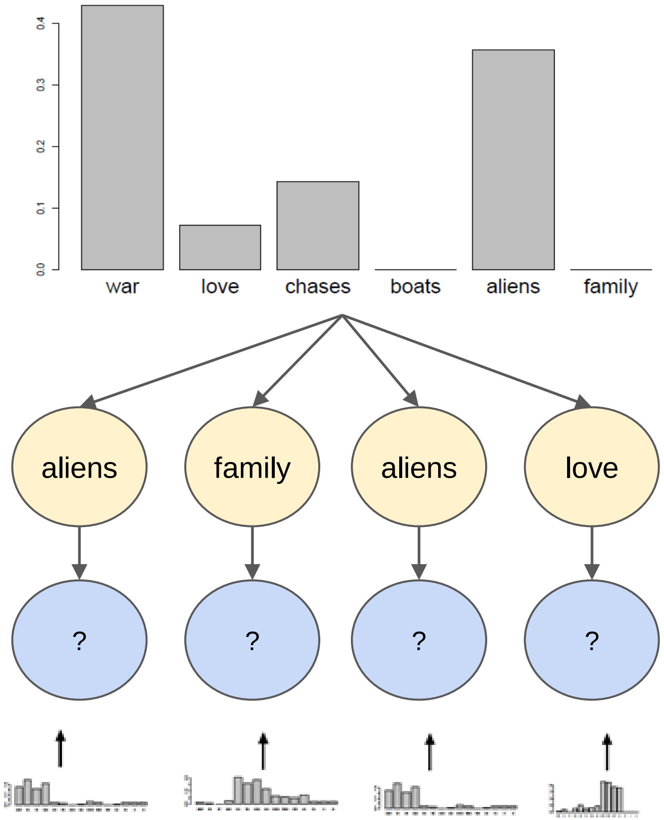
Topic model step-by-step

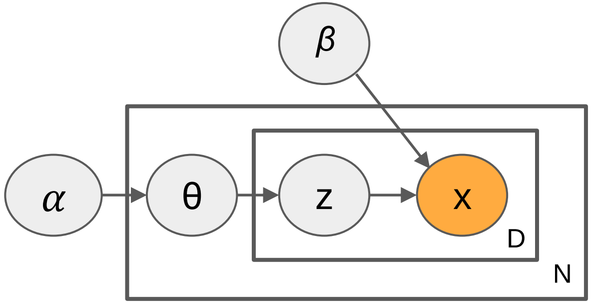




P(topic | topic distribution θ)

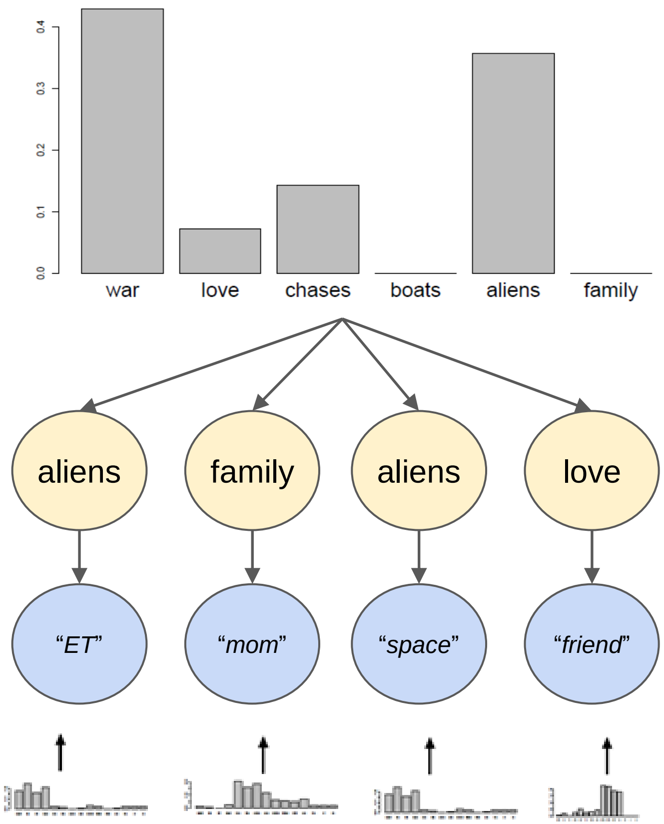
Topic model step-by-step

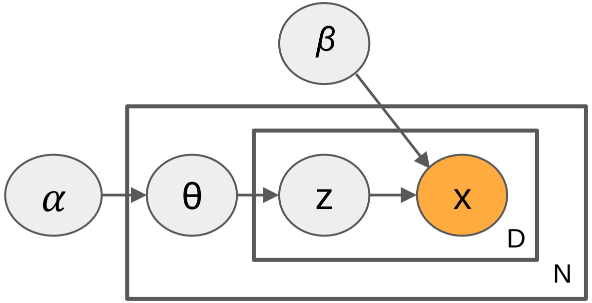




P(word | topic z, β)

Topic model step-by-step





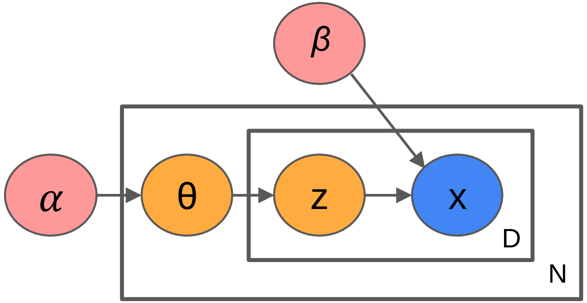
P(word | topic z, β)

Assumptions

- The only information we have are distributions of **words** across the **documents**

- No sequential information
  - topics for words are independent of each other given the set of topics for a document
- Each particular word has one topic
  - but in general we can obtain the same word from different topics!
- Every document has one topic distribution
- Topics don’t have arbitrary correlations (Dirichlet prior)

Learning the parameters



- What are the topic distributions for each document?

- What are the word distributions for each topic?

- Find the **parameters** that **maximize** the **likelihood** of the training **data**!
    - using variational EM or Gibbs sampling

Inferred Distributions: topics+words

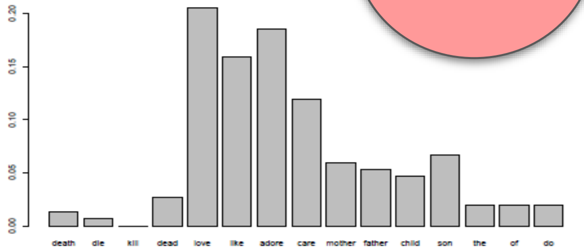
$\theta \in \mathbb{R}^{N\mathrm{x}K}$

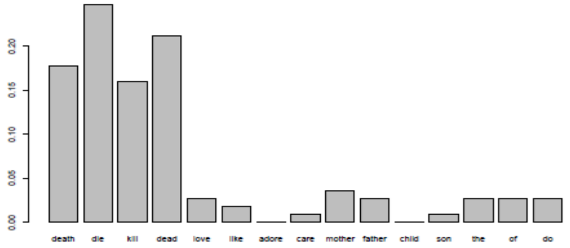
{album, band, music}	{government, party, election}	{game, team, player}
album	government	game
band	party	team
music	election	player
song	state	win
release	political	play

{god, call, give}	{company, market, business}	{math, number, function}
god	company	math
call	market	number
give	business	function
man	year	code
time	product	set

{city, large, area}	{math, energy, light}	{law, state, case}
city	math	law
large	energy	state
area	light	case
station	field	court
include	star	legal

$\beta \in \mathbb{R}^{K\mathrm{x}V}$



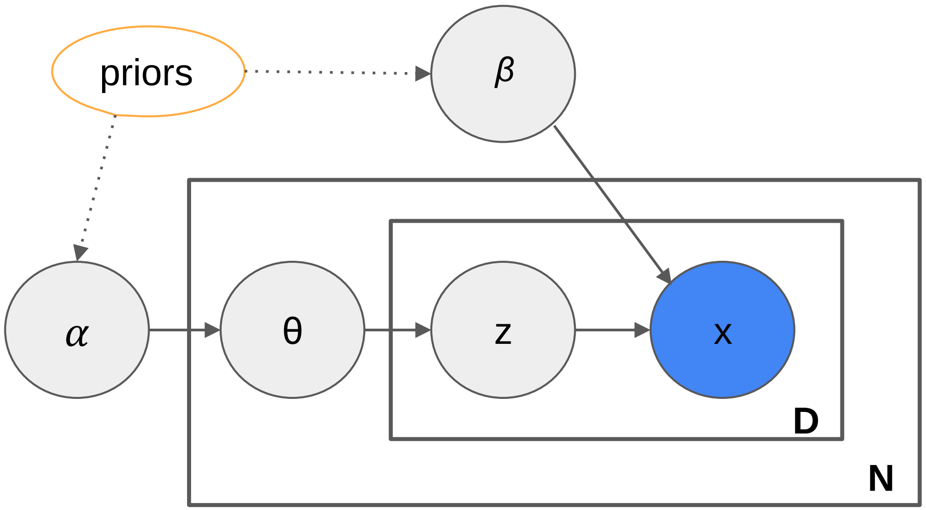


Smoothed LDA

- Empirically maximizing the likelihood of training data can be problematic
- Particularly in LDA, $\beta \in \mathbb{R}^{K \times V}$ is an unseen but **fixed** value
  - i.e., it is a **point estimate** with no distribution
  - this means zero probability for unseen words!

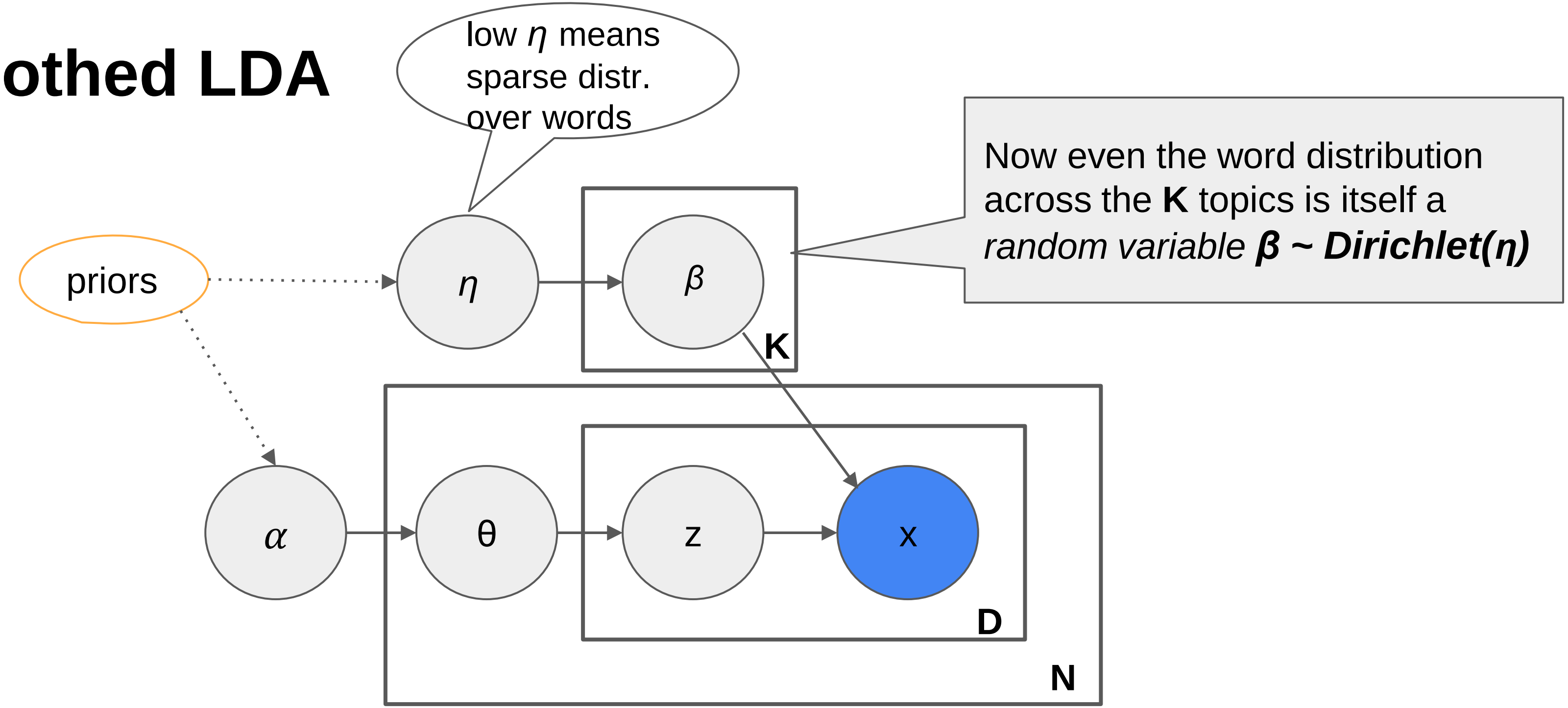
- What can we do? You already know what…
  - Apply **smoothing!**
    - e.g. +1 (Laplace) that we discussed
    - actually used in practice, but rather ad-hoc
- A more principled solution?
  - **go more Bayesian!**

Smoothed LDA



A Bayesian topic model

Smoothed LDA



A **smoothed** Bayesian topic model

- With this ***η*** we now have a **Dirichlet** prior for all the words (even unseen in the corpus!)

How to use in practice

a) Implement your own LDA and variational EM…

b) **from** sklearn.decomposition **import** LatentDirichletAllocation

- not to be confused with another LDA = Linear Discriminant Analysis
  - used to be in sklearn.lda.LDA
  - now: sklearn.discriminant_analysis.LinearDiscriminantAnalysis

- Other popular libraries:
  - Gensim
    - from Radim Rehurek
  - Spacy

- In the next lecture, we will see a complementary approach with matrix decompositions